

Tarea 2: Kubernetes

Diego David Puerta Diaz
Renato Bacigalupo Ortiz-Arrieta

Ejercicio 1.1 y 1.2

Primero es necesario instalar minikube, kubectl y VirtualBox (o cualquier hypervisor) para poder hacer el deploy de un cluster kubernetes.

Para hacer esto ejecutamos los siguientes comandos para descargar e instalar minikube y kubectl (asumiendo que ya se tiene VirtualBox instalado):

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube
```

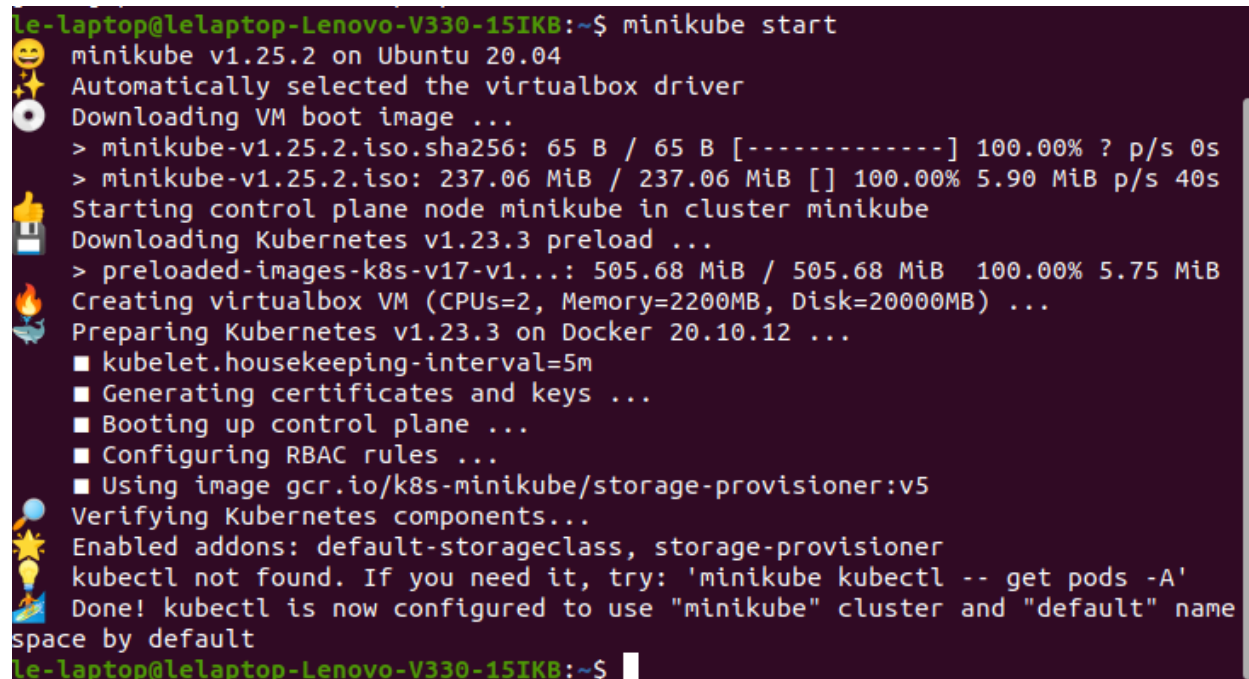
```
curl -LO "https://dl.k8s.io/release/${curl -L -s https://dl.k8s.io/release/stable.txt}/bin/linux/amd64/kubectl"
```

```
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

Una vez que ya se tiene instalados estos dos componentes se puede hacer ejecutar el cluster. Para hacer deploy de un cluster de un solo nodo de ejemplo con dos CPU's, 2200 MB de RAM y 20 Gb, ejecutamos el siguiente comando:

```
minikube start
```

Esto nos da el siguiente output en la consola:



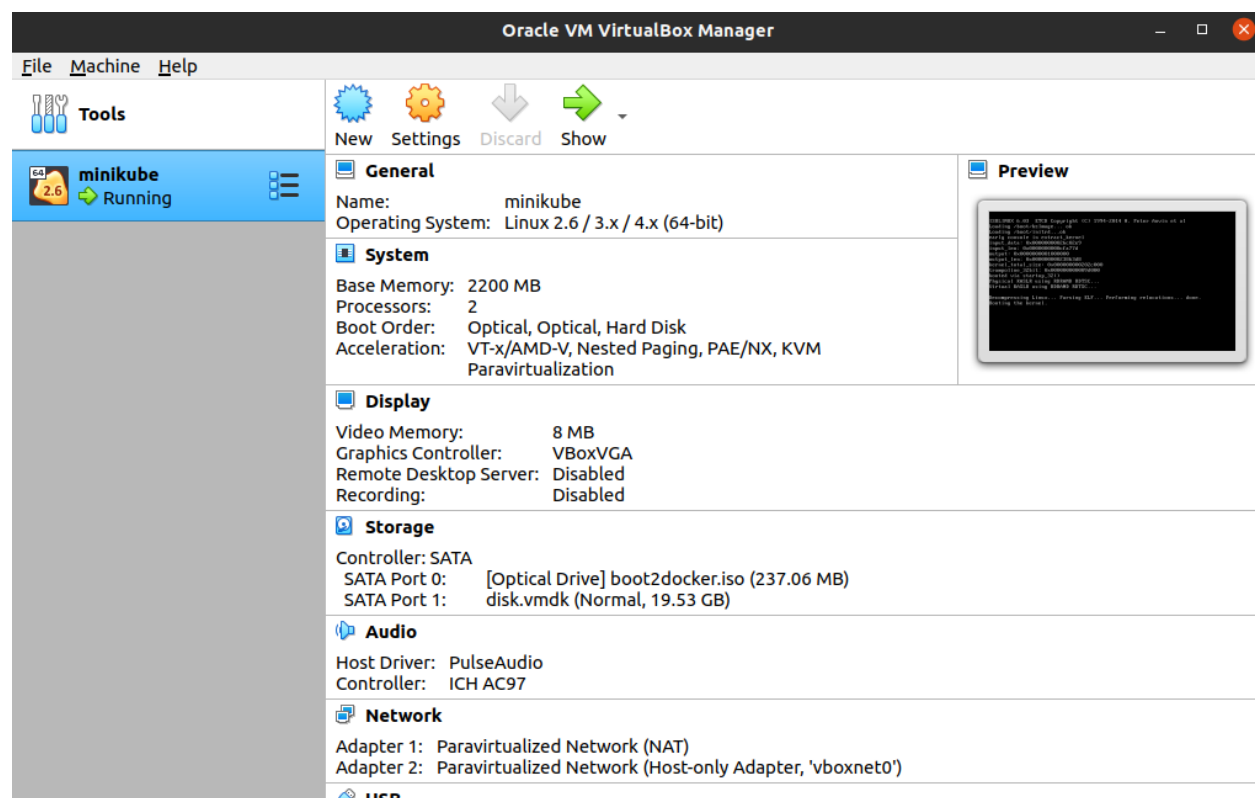
```
le-laptop@lelaptop-Lenovo-V330-15IKB:~$ minikube start
🐹 minikube v1.25.2 on Ubuntu 20.04
🌟 Automatically selected the virtualbox driver
💿 Downloading VM boot image ...
> minikube-v1.25.2.iso.sha256: 65 B / 65 B [-----] 100.00% ? p/s 0s
> minikube-v1.25.2.iso: 237.06 MiB / 237.06 MiB [] 100.00% 5.90 MiB p/s 40s
👍 Starting control plane node minikube in cluster minikube
📦 Downloading Kubernetes v1.23.3 preload ...
> preloaded-images-k8s-v17-v1...: 505.68 MiB / 505.68 MiB 100.00% 5.75 MiB
🔥 Creating virtualbox VM (CPUs=2, Memory=2200MB, Disk=20000MB) ...
🚧 Preparing Kubernetes v1.23.3 on Docker 20.10.12 ...
   ■ kubelet.housekeeping-interval=5m
   ■ Generating certificates and keys ...
   ■ Booting up control plane ...
   ■ Configuring RBAC rules ...
   ■ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🔍 Verifying Kubernetes components...
💡 Enabled addons: default-storageclass, storage-provisioner
🔧 kubectl not found. If you need it, try: 'minikube kubectl -- get pods -A'
💡 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
le-laptop@lelaptop-Lenovo-V330-15IKB:~$
```

Esto quiere decir que nuestro cluster ya está ejecutándose en un VM de VirtualBox. Podemos verificar esto ejecutando el siguiente comando:

```
minikube status
```

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

La siguiente imagen demuestra el VM creado por minikube en el dashboard de VirtualBox:



Ahora intentaremos ejecutar una imagen dentro de un pod (container) para probar. Ejecutamos el siguiente comando:

```
kubectl run myshell --image busybox -- sh
```

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~$ kubectl run myshell --image busybox -- sh
pod/myshell created
le-laptop@lelaptop-Lenovo-V330-15IKB:~$ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
myshell   0/1     Completed 4 (47s ago) 99s
```

El output nos muestra que se creó un nuevo pod de forma correcta y este ya terminó de ejecutar.

Por último, ejecutaremos siguientes comandos:

```
kubectl cluster-info
kubectl get nodes -owide
```

Estos dos comandos muestran información general del cluster y la cantidad de nodos con una pequeña cantidad de información de los mismos respectivamente. Sus outputs fueron los siguientes:

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~$ kubectl cluster-info
Kubernetes control plane is running at https://192.168.59.102:8443
CoreDNS is running at https://192.168.59.102:8443/api/v1/namespaces/kube-system/service
s/kube-dns:dns/proxy
```

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~$ kubectl get nodes -owide
NAME          STATUS    ROLES          AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION   CONTAINER-RUNTIME
minikube      Ready    control-plane,master   30m   v1.23.3   192.168.59.100 <none>        Buildroot 2021.02.4   4.19.202   docker://20.10.12
```

Ahora haremos el mismo proceso, pero esta vez ejecutaremos un cluster con dos nodos. Para hacer el deploy de este cluster con dos nodos ejecutamos el siguiente comando:

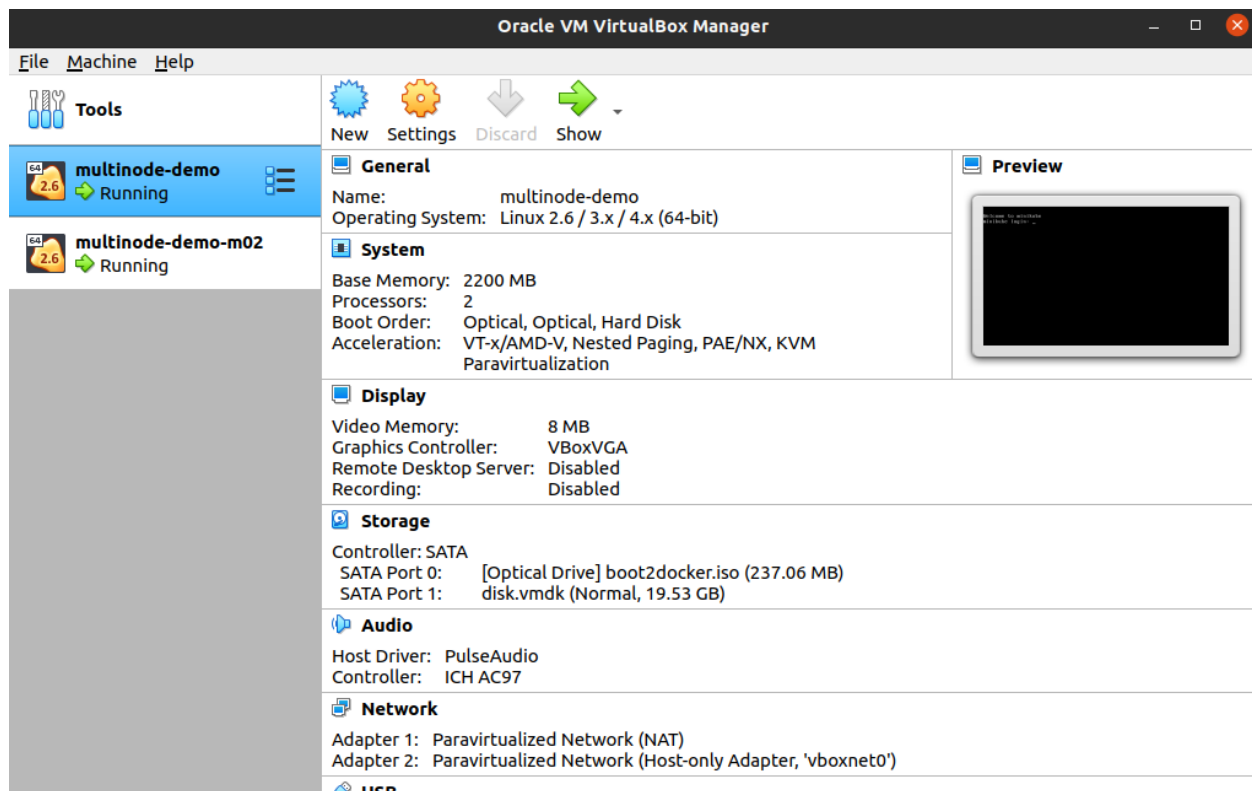
```
minikube start --nodes 2 -p multinode-demo
```

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~$ minikube start --nodes 2 -p multinode-demo
🌻 [multinode-demo] minikube v1.25.2 on Ubuntu 20.04
🔧 Automatically selected the virtualbox driver
👍 Starting control plane node multinode-demo in cluster multinode-demo
🔥 Creating virtualbox VM (CPUs=2, Memory=2200MB, Disk=20000MB) ...
🐳 Preparing Kubernetes v1.23.3 on Docker 20.10.12 ...
   ▪ kubelet.housekeeping-interval=5m
   ▪ kubelet.cni-conf-dir=/etc/cni/net.mk
   ▪ Generating certificates and keys ...
   ▪ Booting up control plane ...
   ▪ Configuring RBAC rules ...
🔗 Configuring CNI (Container Networking Interface) ...
🔍 Verifying Kubernetes components...
   ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟 Enabled addons: storage-provisioner, default-storageclass

👍 Starting worker node multinode-demo-m02 in cluster multinode-demo
🔥 Creating virtualbox VM (CPUs=2, Memory=2200MB, Disk=20000MB) ...
🌐 Found network options:
   ▪ NO_PROXY=192.168.59.102
🐳 Preparing Kubernetes v1.23.3 on Docker 20.10.12 ...
   ▪ env NO_PROXY=192.168.59.102
🔍 Verifying Kubernetes components...
```

Este comando crea un cluster con dos nodos con las mismas especificaciones del cluster anterior.

Esta vez se crearon dos nodos, lo que significa que se crean dos VM's en VirtualBox, esto lo podemos verificar en el dashboard:



Ejecutamos los mismos comandos anteriores y estos fueron los resultados:

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~$ minikube -p multinode-demo status
multinode-demo
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

multinode-demo-m02
type: Worker
host: Running
kubelet: Running
```

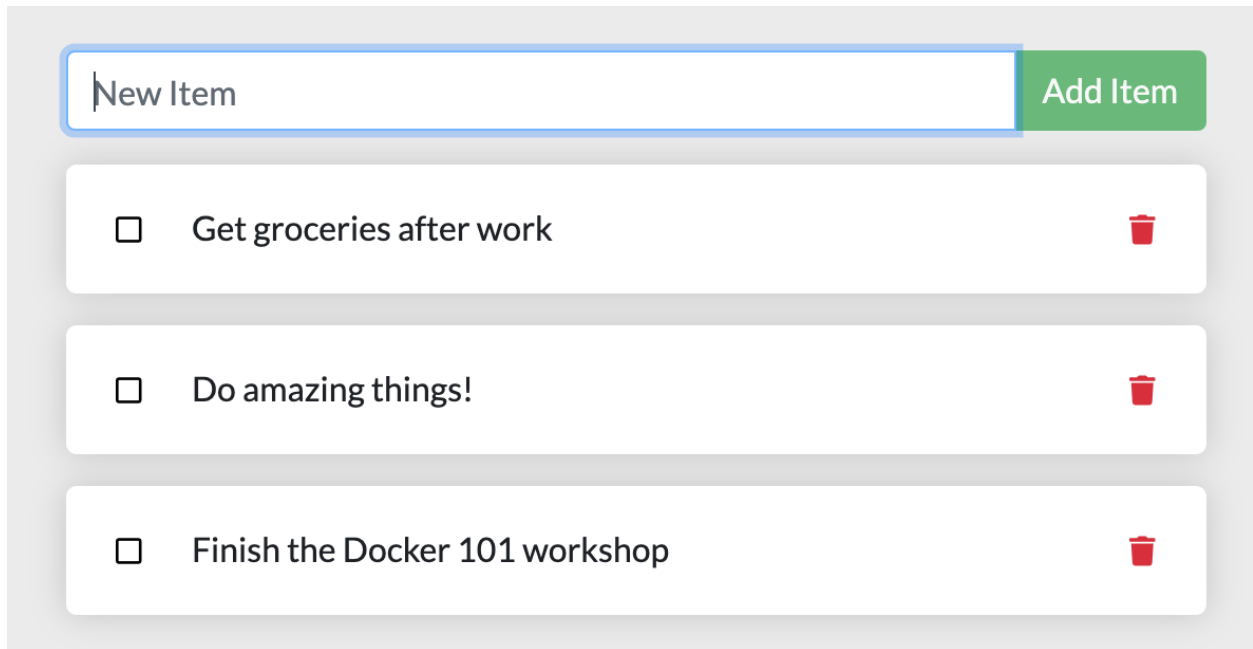
```
le-laptop@lelaptop-Lenovo-V330-15IKB:~$ kubectl cluster-info
Kubernetes control plane is running at https://192.168.59.105:8443
CoreDNS is running at https://192.168.59.105:8443/api/v1/namespaces/kube-system/
services/kube-dns:dns/proxy
```

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION
CONTAINER-RUNTIME								
multinode-demo	Ready	control-plane,master	6m16s	v1.23.3	192.168.59.105	<none>	Buildroot 2021.02.4	4.19.202
multinode-demo-m02	Ready	<none>	2m40s	v1.23.3	192.168.59.106	<none>	Buildroot 2021.02.4	4.19.202

Ejercicio 2

La aplicación que escogimos para esta tarea es una aplicación web basada en containers. Esta es una aplicación donde se pueden añadir nuevos ítems con un botón y eliminarlos si se quiere.

La siguiente imagen muestra esta app:



The screenshot shows a web application interface. At the top, there is a text input field containing the placeholder text "New Item" and a green button labeled "Add Item". Below this, there is a list of three items, each in a white box with a light gray border. Each item consists of a checkbox, a text label, and a red trash icon on the right. The items are: "Get groceries after work", "Do amazing things!", and "Finish the Docker 101 workshop".

Link de Referencia:

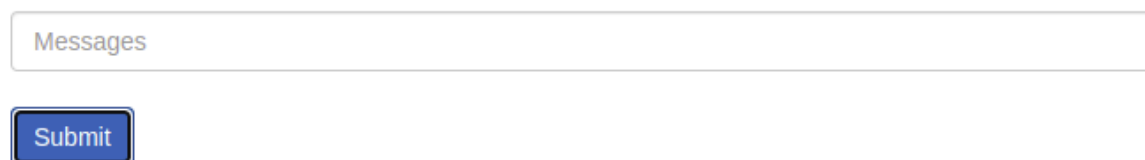
https://docs.docker.com/get-started/02_our_app/

Además de esta primera app, y debido a que esta es considerablemente simple, haremos uso de otra app parecida, pero hecha de forma más compleja con un backend y una base de datos. En esta app solo es posible añadir nuevos ítems y no quitar, sin embargo, estos son guardados en una base de datos con el uso del api que proporciona el backend. El propósito de esta aplicación es de ser usada como Libro de Invitados, es decir, invitados pueden guardar su información.

Link de Referencia:

<https://kubernetes.io/docs/tutorials/stateless-application/guestbook/>

Guestbook



The screenshot shows the "Guestbook" application interface. It features a large text input field with the placeholder text "Messages". Below the input field is a blue button labeled "Submit".

Link de GitHub:

<https://github.com/Renato01011/Tarea-2-Kubernetes-Ejercicio-2>

Ejercicio 3

Para ejecutar la primera aplicación que mencionamos anteriormente primero necesitamos convertir estos archivos en una imagen de docker y subirlas a docker.

Para esto después de crear el Dockerfile corremos el siguiente comando para hacer build a la imagen.

`docker build -t singlenodeapp .`

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2/App$ sudo docker build -t singlenodeapp .
Sending build context to Docker daemon 6.775MB
Step 1/7 : FROM node:12-alpine
12-alpine: Pulling from library/node
df9b9388f04a: Pull complete
3bf6d7380205: Pull complete
7939e601ee5e: Pull complete
31f0fb9de071: Pull complete
Digest: sha256:d4b15b3d48f42059a15bd659be60afe21762aae9d6cbea6f124440895c27db68
Status: Downloaded newer image for node:12-alpine
--> bb6d28039b8c
Step 2/7 : RUN apk add --no-cache python2 g++ make
--> Running in f0242f37521b
fetch https://dl-cdn.alpinelinux.org/alpine/v3.15/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.15/community/x86_64/APKINDEX.tar.gz
(1/22) Installing binutils (2.37-r3)
(2/22) Installing libgomp (10.3.1_git20211027-r0)
(3/22) Installing libatomic (10.3.1_git20211027-r0)
(4/22) Installing libgphobos (10.3.1_git20211027-r0)
(5/22) Installing gmp (6.2.1-r1)
(6/22) Installing isl22 (0.22-r0)
(7/22) Installing mpfr4 (4.1.0-r0)
```

Verificamos que la imagen este bien con el siguiente comando:

`docker images`

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2/App$ sudo docker images
REPOSITORY          TAG             IMAGE ID        CREATED         SIZE
singlenodeapp       latest          51bec5ad7623    About a minute ago  406MB
node                12-alpine      bb6d28039b8c    2 weeks ago     91MB
```

Ahora subimos esta imagen a docker con el siguiente comando:

`docker push [docker_username]/singlenodeapp`

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2/App$ sudo docker push renato01011/singlenodeapp
Using default tag: latest
The push refers to repository [docker.io/renato01011/singlenodeapp]
ee37fb20d20d: Pushed
ffc7693a2506: Pushed
b44fe78327f7: Pushed
f6bf2986d85f: Pushed
7f30cde3f699: Pushed
fe810f5902cc: Pushed
dfd8c046c602: Pushed
4fc242d58285: Pushed
latest: digest: sha256:fcdb8fc5b8e27d30bce4a66e048359cdd9f0db9154d8e51e0b71c55ce24229043 size: 2000
```

Una vez tengamos la imagen en docker podemos crear un archivo manifestó .yaml, para que un pod descargue esta imagen de docker como su imagen:

```
apiVersion: v1
kind: Pod
metadata:
  name: itemWebPage
spec:
  containers:
  - name: itemWebPage01
    image: renato01011/singlenodeapp
    ports:
    - containerPort: 6379
```

Ahora podemos iniciar el cluster:

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2$ minikube start
🐹 minikube v1.25.2 on Ubuntu 20.04
🌟 Automatically selected the virtualbox driver
👍 Starting control plane node minikube in cluster minikube
🔥 Creating virtualbox VM (CPUs=2, Memory=2200MB, Disk=20000MB) ...
🚧 Preparing Kubernetes v1.23.3 on Docker 20.10.12 ...
  ▪ kubelet.housekeeping-interval=5m
  ▪ Generating certificates and keys ...
  ▪ Booting up control plane ...
  ▪ Configuring RBAC rules ...
  ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟 Enabled addons: storage-provisioner, default-storageclass
🔍 Verifying Kubernetes components...
🏁 Done! kubectl is now configured to use "minikube" cluster and "default" name space by default
```

Una vez iniciado podemos crear el pod con el siguiente comando:

kubectl create -f SingleNodeAppManifest.yml

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2$ kubectl create -f SingleNodeAppManifest.yml
pod/item-web-page created
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
item-web-page	0/1	ContainerCreating	0	21s

Esto nos creó el nuevo container/pod con nuestra imagen de docker ejecutándose. Podemos ver más información del mismo ejecutando el comando:

kubectl describe pods/item-web-page

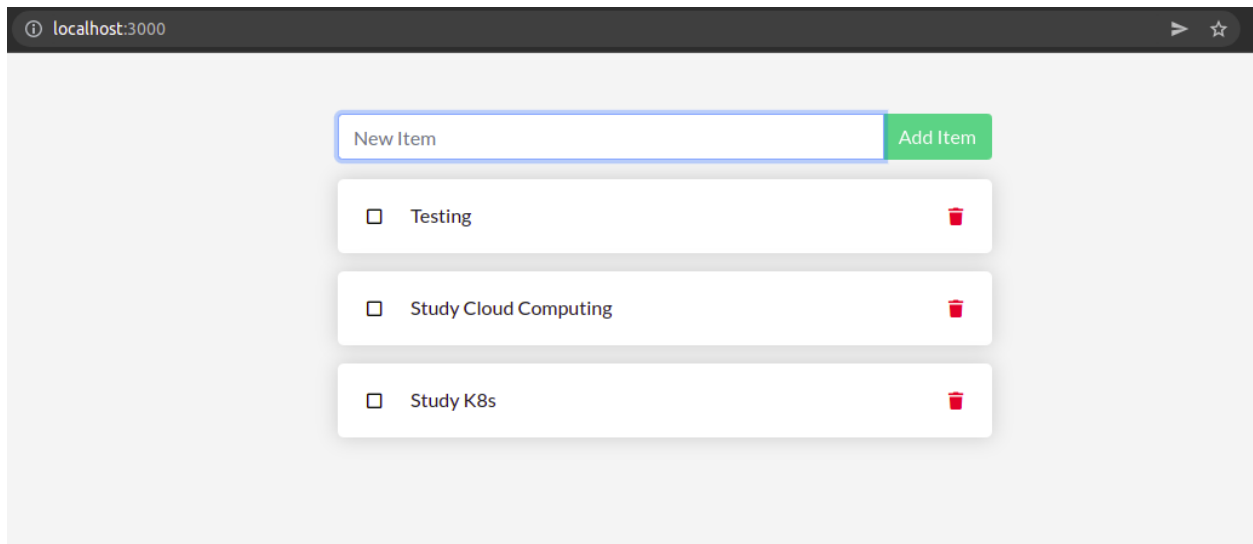
```
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2$ kubectl describe pods/item-web-page
Name:          item-web-page
Namespace:     default
Priority:       0
Node:          minikube/192.168.59.107
Start Time:    Fri, 29 Apr 2022 00:39:40 -0500
Labels:        <none>
Annotations:   <none>
Status:        Running
IP:            172.17.0.3
IPs:           IP: 172.17.0.3
Containers:
  item-web-page01:
    Container ID:  docker://6acb3d47c1d38e3046c1fb7ad086dea9c57df11adee5ca594c2da05311fe4d16
    Image:         renato01011/singlenodeapp
    Image ID:      docker-pullable://renato01011/singlenodeapp@sha256:fc88fc5b8e27d30bce4a66e048359cdd9f0db9154d8e51e0b71c55ce24229043
    Port:         6379/TCP
    Host Port:     0/TCP
    State:         Running
      Started:     Fri, 29 Apr 2022 00:40:20 -0500
    Ready:         True
    Restart Count: 0
    Environment:   <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-xh7rc (ro)
Conditions:
  Type           Status
  Initialized    True
  Ready          True
  ContainersReady True
  PodScheduled   True
Volumes:
```

Cuando termine de iniciar el pod y se esté ejecutando nuestra aplicación podemos probarla primero haciendo un port-forward del puerto de pod a el puerto de localhost de nuestro host. Ejecutamos el siguiente comando para lograr esto:

kubectl port-forward item-web-page 3000:3000

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2$ kubectl port-forward item-web-page 3000:3000
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
Handling connection for 3000
```


Esto hará que nuestra podamos comunicarnos el puerto 3000 del pod con el puerto 3000 del host. Y si entramos a localhost:3000 en cualquier explorador web veremos nuestra app ejecutándose:



Para hacer deploy a la segunda aplicación en multinodo, no necesitamos hacer el proceso inicial, ya que todos los containers que necesitamos ya tienen una imagen que podemos traer de docker.

Esto quiere decir que podemos iniciar el cluster primero como lo hicimos anteriormente:

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2/MultiNode$ minikube start --nodes 2 -p multinode-demo
[emoticon] [multinode-demo] minikube v1.25.2 on Ubuntu 20.04
[stars] Automatically selected the virtualbox driver
[thumbs up] Starting control plane node multinode-demo in cluster multinode-demo
[fire] Creating virtualbox VM (CPUs=2, Memory=2200MB, Disk=20000MB) ...
[cloud] Preparing Kubernetes v1.23.3 on Docker 20.10.12 ...
  ■ kubelet.housekeeping-interval=5m
  ■ kubelet.cni-conf-dir=/etc/cni/net.mk
  ■ Generating certificates and keys ...
  ■ Booting up control plane ...
  ■ Configuring RBAC rules ...
🔗 Configuring CNI (Container Networking Interface) ...
  ■ Using image gcr.io/k8s-minikube/storage-provisioner:v5
💡 Verifying Kubernetes components...
🌟 Enabled addons: storage-provisioner, default-storageclass

👍 Starting worker node multinode-demo-m02 in cluster multinode-demo
🔥 Creating virtualbox VM (CPUs=2, Memory=2200MB, Disk=20000MB) ... \
```

Una vez iniciado necesitamos crear todos los pods necesarios para esta aplicación. Además, esta aplicación hace uso de servicios los cuales también tenemos que iniciar. Ejecutamos un comando cambiando el nombre los archivos dependiendo al manifestó que queremos invocar.

kubectl apply -f [nombre-de-manifesto].yaml

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2/MultiNode$ kubectl apply -f redis-leader-deployment.yaml
deployment.apps/redis-leader created
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2/MultiNode$ kubectl apply -f redis-leader-service.yaml
service/redis-leader created
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2/MultiNode$ kubectl apply -f redis-follower-deployment.yaml
deployment.apps/redis-follower created
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2/MultiNode$ kubectl apply -f redis-follower-service.yaml
service/redis-follower created
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2/MultiNode$ kubectl apply -f frontend-deployment.yaml
deployment.apps/frontend created
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2/MultiNode$ kubectl apply -f frontend-service.yaml
service/frontend created
```

Al finalizar tendremos todos los servicios y los pods que necesitamos para ejecutar esta aplicación. Para verificar si los nodos y servicios están bien creados y ejecutándose, utilizaremos los dos siguientes comandos:

Kubectl get pods

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2/MultiNode$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
frontend-57df59b89c-7kjm	0/1	ContainerCreating	0	3m3s
frontend-57df59b89c-kfrdw	0/1	ContainerCreating	0	3m3s
frontend-57df59b89c-mz657	0/1	ContainerCreating	0	3m3s
redis-follower-84fcc94dfc-ckkwt	1/1	Running	0	3m29s
redis-follower-84fcc94dfc-llr2d	1/1	Running	0	3m29s
redis-leader-766465cd9c-9qf8d	1/1	Running	0	4m13s

Kubectl get services

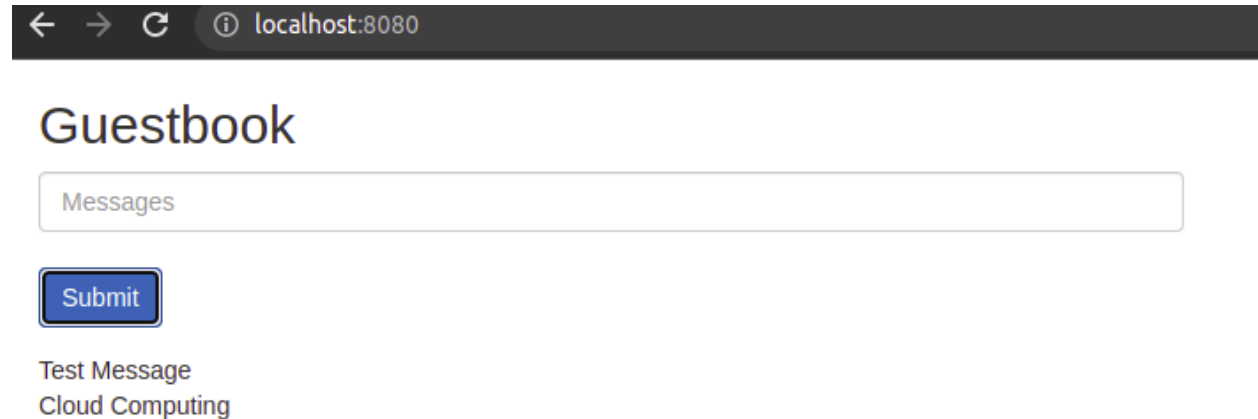
```
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2/MultiNode$ kubectl get services
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
frontend	ClusterIP	10.96.147.240	<none>	80/TCP	3m25s
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	11m
redis-follower	ClusterIP	10.101.196.67	<none>	6379/TCP	3m56s
redis-leader	ClusterIP	10.106.147.130	<none>	6379/TCP	4m26s

Una vez los pods terminen de iniciar podemos probar la aplicación del mismo modo que probamos la aplicación anterior, es decir haciendo un port-forward de pod a host.

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2/MultiNode$ kubectl port-forward svc/frontend 8080:80
Forwarding from 127.0.0.1:8080 -> 80
Forwarding from [::1]:8080 -> 80
```

Ahora podemos entrar a localhost:8080, y verificar nuestra app ejecutándose.



Además, podemos verificar si nuestro hypervisor está haciendo uso de los dos nodos, asignando los pods a los dos y no solo a un nodo. Podemos verificar a que nodo este asignado que pod con el siguiente comando:

kubectl get pods -o wide

```
le-laptop@lelaptop-Lenovo-V330-15IKB:~/Desktop/Universidad/Ciclo 9/Cloud Computing/Tarea 2/MultiNode$ kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
frontend-57df59b89c-7kjmr	1/1	Running	0	12m	10.244.1.4	multinode-demo-m02	<none>	<none>
frontend-57df59b89c-kfrdw	1/1	Running	0	12m	10.244.0.4	multinode-demo	<none>	<none>
frontend-57df59b89c-mz657	1/1	Running	0	12m	10.244.1.5	multinode-demo-m02	<none>	<none>
redis-follower-84fcc94dfc-ckkwt	1/1	Running	0	13m	10.244.0.3	multinode-demo	<none>	<none>
redis-follower-84fcc94dfc-llr2d	1/1	Running	0	13m	10.244.1.3	multinode-demo-m02	<none>	<none>
redis-leader-766465cd9c-9qf8d	1/1	Running	0	14m	10.244.1.2	multinode-demo-m02	<none>	<none>

Como podemos ver en la sección de "Node", existen 4 pods en el segundo nodo y dos pods en el primero. Esto quiere decir que la aplicación esta efectivamente corriendo en más de un nodo.

Diagramas de Flujo

Diagrama de flujo a alto nivel describiendo la primera aplicación:

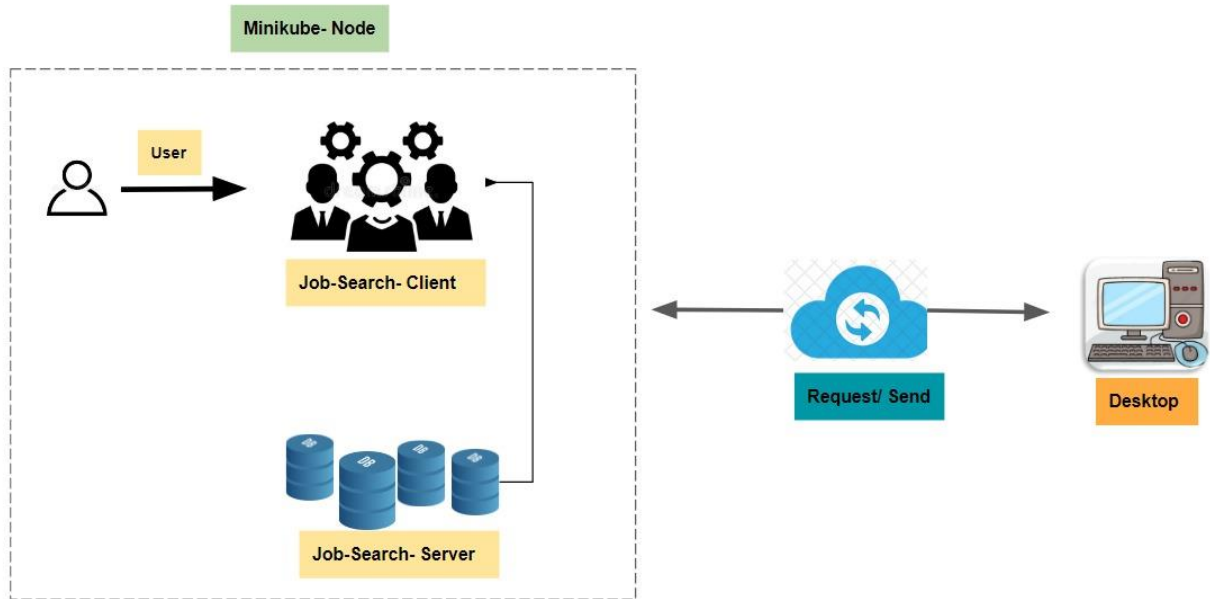


Diagrama de flujo a bajo nivel describiendo la segunda aplicación:

