Managing your stateless and stateful applications with Kubernetes provides efficiencies and simplifies automation. However, before using StatefulSets for your own stateful applications, you should consider if any of the following apply:

* You embrace microservices
* You frequently create new service footprints that include stateful applications
* Your current solution for storing state can't scale to meet predicted demand
* Your stateful applications can meet performance requirements without using specialized hardware and could effectively run on the same hardware used for stateless applications
* You value flexible reallocation of resources, consolidation, and automation over squeezing the most and having highly predictable performance

If any of the previous bullets apply to your situation, it may make sense to use Kubernetes for your stateful applications.

**Background**

**ConfigMaps**: A type of Kubernetes resource that is used to decouple configuration artifacts from image content to keep containerized applications portable. The configuration data is stored as key-value pairs.

**Headless Service**: A headless service is a Kubernetes service resource that won't load balance behind a single service IP. Instead, a headless service returns a list of DNS records that point directly to the pods that back the service. A headless service is defined by declaring the clusterIP property in a service spec and setting the value to None. StatefulSets currently require a headless service to identify pods in the cluster network.

**Stateful Sets**: Similar to Deployments in Kubernetes, StatefulSets manage the deployment and scaling of pods given a container spec.StatefulSets differ from Deployments in that the Pods in a stateful set are not interchangeable. Each pod in a StatefulSet has a persistent identifier that it maintains across any rescheduling. The pods in a StatefulSet are also ordered. This provides a guarantee that one pod can be created before following pods. In this Lab, this is useful for ensuring the control plane node is provisioned first.

**PersistentVolumes (PVs) and PersistentVolumeClaims (PVCs)**: PVs are Kubernetes resources that represent storage in the cluster. Unlike regular Volumes which exist only until while containing pod exists, PVs do not have a lifetime connected to a pod. Thus, they can be used by multiple pods over time, or even at the same time. Different types of storage can be used by PVs including NFS, iSCSI, and cloud-provided storage volumes, such as AWS EBS volumes. Pods claim PV resources through PVCs.

**MySQL replication**: This Lab uses a single primary, asynchronous replication scheme for MySQL. All database writes are handled by a single primary. The database replicas asynchronously synchronize with the primary. This means the primary will not wait for the data to be copied onto the replicas. This can improve the performance of the primary at the expense of having replicas that are not always exact copies of the primary. Many applications can tolerate slight differences in the data and are able to improve the performance of database read workloads by allowing clients to read from the replicas.

Requisitos

**- Instalar AWS CLI, AWS IAM Authenticator y Kubectl**

<https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>

<https://docs.aws.amazon.com/es_es/cli/latest/userguide/install-cliv2-linux.html>

<https://weaveworks-gitops.awsworkshop.io/60_workshop_6_ml/00_prerequisites.md/50_install_aws_iam_auth.html>

<https://docs.aws.amazon.com/es_es/eks/latest/userguide/install-aws-iam-authenticator.html>

<https://kubernetes.io/es/docs/tasks/tools/install-kubectl/>

<https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html>

**- Crear un cluster en Kubernetes**

<https://www.youtube.com/watch?v=aZd0UolVwD4>

# [AWS EKS | Create EKS Cluster on AWS using Console | Install Kubernetes on AWS]

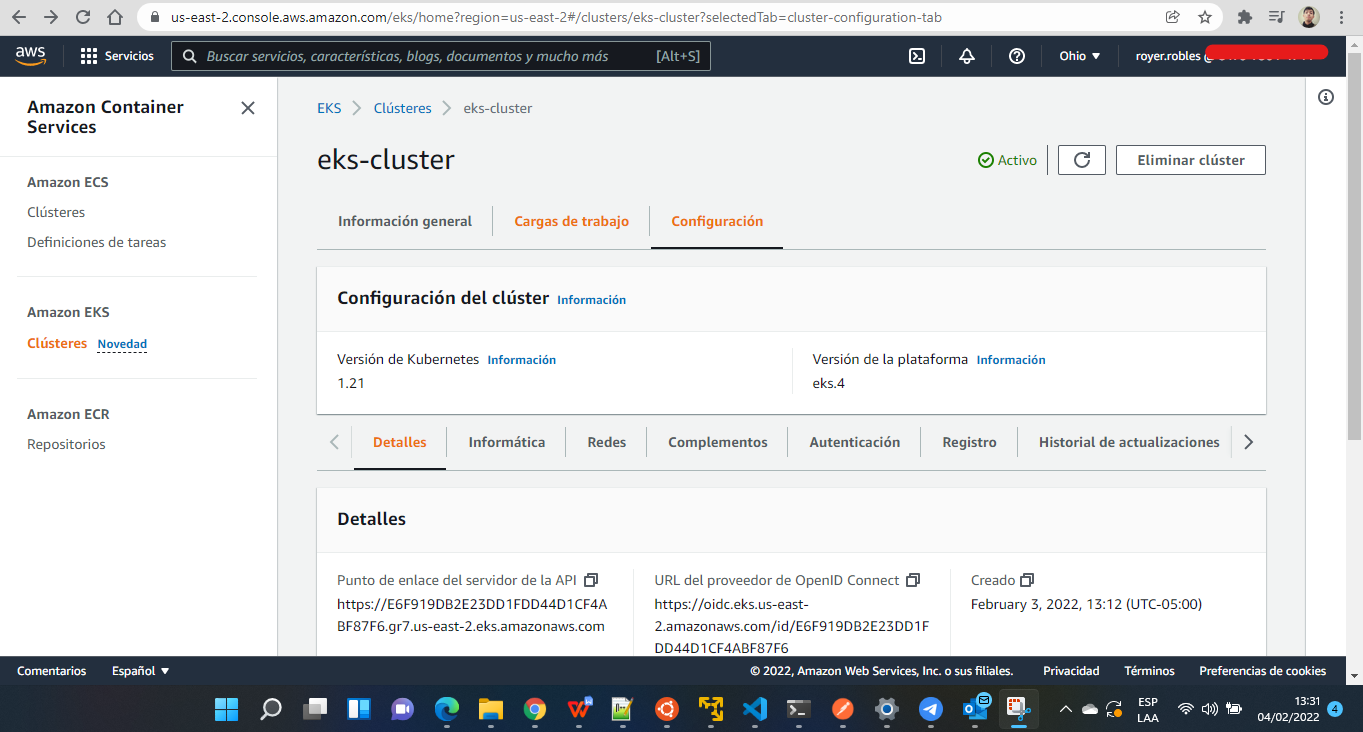
<https://learnitguide.net/2020/08/how-to-create-eks-cluster-on-aws-using-console.html>

[**How to Create EKS Cluster on AWS using Console**]

<https://logz.io/blog/amazon-eks-cluster/>

**[Deploying a Kubernetes Cluster with Amazon EKS]**

**Eks-cluster**



**- Estableciendo conexión al cluster**

<https://www.youtube.com/watch?v=uLnrHqzoArc>

# [Why can’t I connect to my Amazon EKS cluster?]

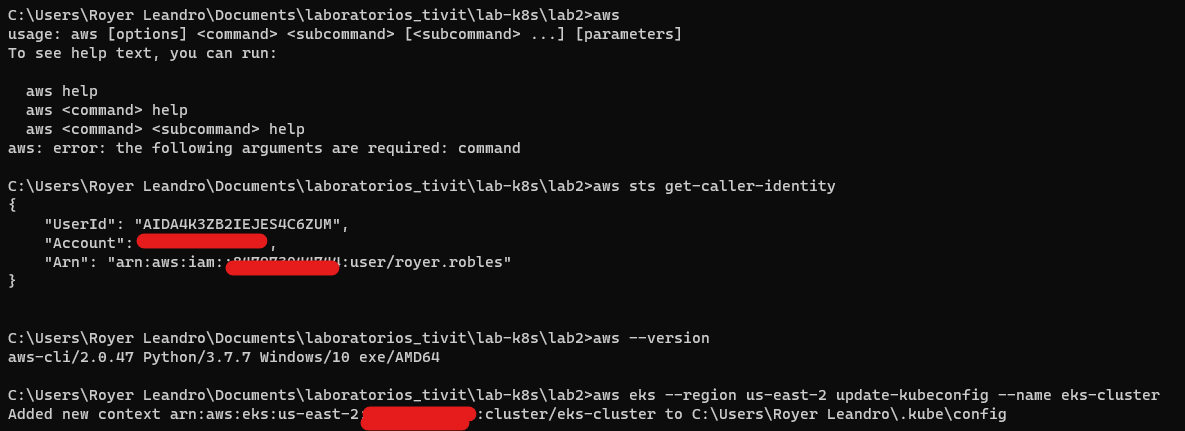
<https://aws.amazon.com/es/premiumsupport/knowledge-center/eks-cluster-connection/>

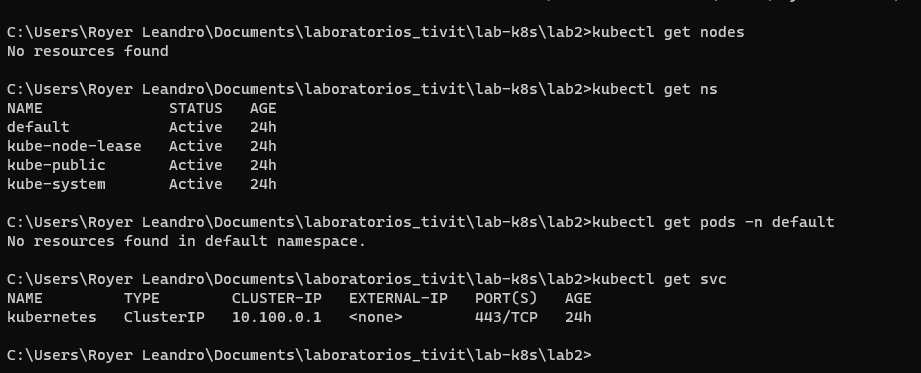
<https://hub.qovery.com/guides/tutorial/how-to-connect-to-your-eks-cluster-with-kubectl/>

**[How to connect to your EKS cluster with kubectl]**

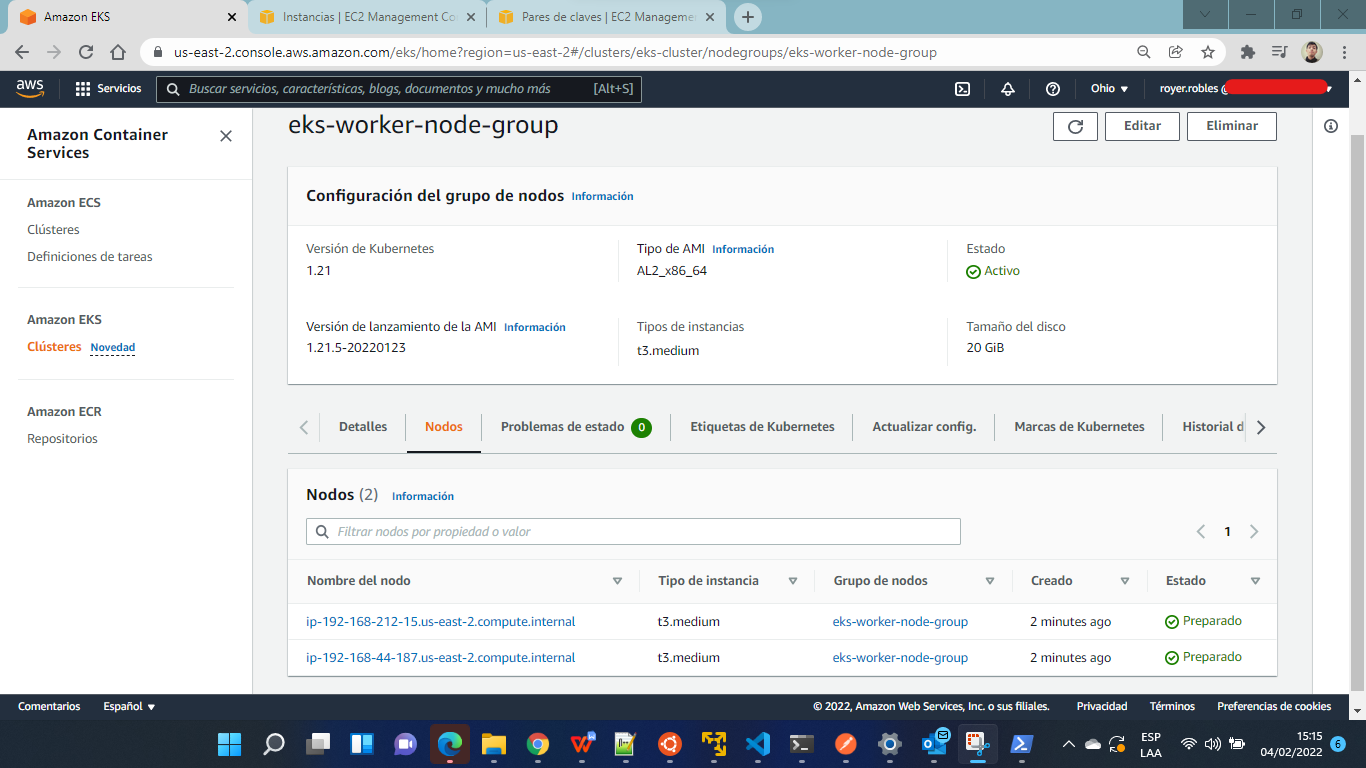
Comandos

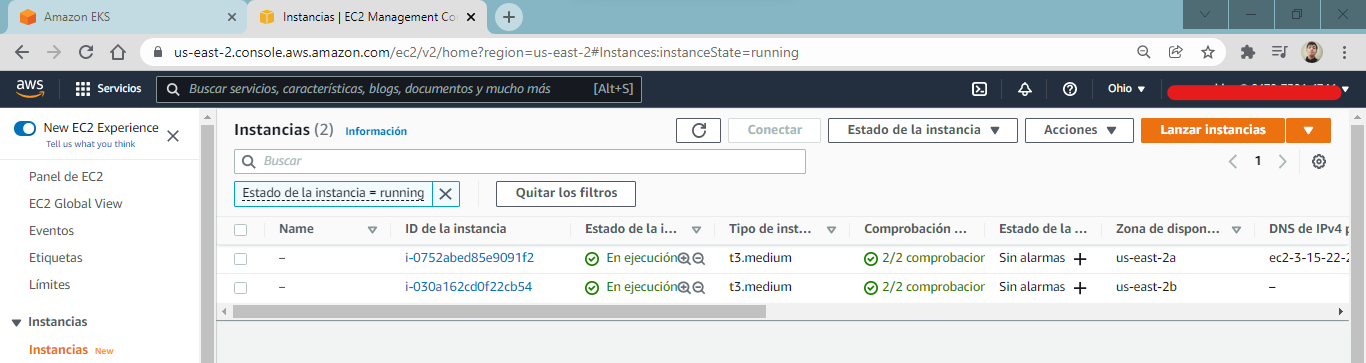
|  |
| --- |
| aws --version  aws sts get-caller-identity  aws eks --region us-east-2 update-kubeconfig --name eks-cluster  kubectl get nodes  kubectl get ns  kubectl get pods -n default  kubectl get svc |

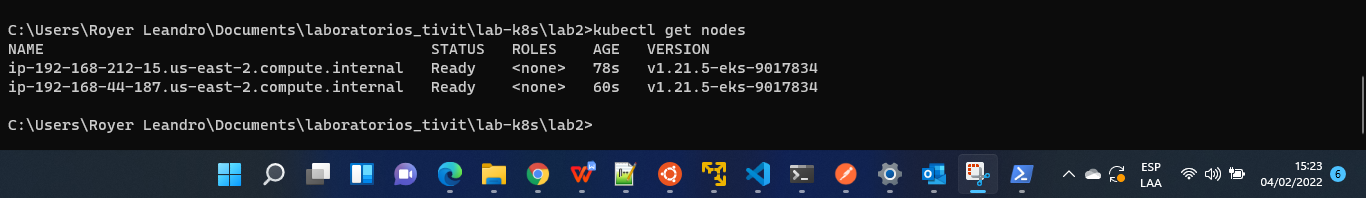




**Creando nodos workers**







Ejecutar las siguientes sentencias:

***Crear YML del ConfigMap***

cat <<EOF > mysql-configmap.yaml

apiVersion: v1

kind: ConfigMap

metadata:

name: mysql

labels:

app: mysql

data:

master.cnf: |

# Apply this config only on the primary.

[mysqld]

log-bin

slave.cnf: |

# Apply this config only on replicas.

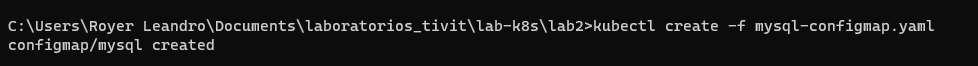
[mysqld]

super-read-only

EOF

***Crear el ConfigMap***

kubectl create -f mysql-configmap.yaml



***Crear el YML del servicio***

cat <<EOF > mysql-services.yaml

# Headless service for stable DNS entries of StatefulSet members.

apiVersion: v1

kind: Service

metadata:

name: mysql

labels:

app: mysql

spec:

ports:

- name: mysql

port: 3306

clusterIP: None

selector:

app: mysql

---

# Client service for connecting to any MySQL instance for reads.

# For writes, you must instead connect to the primary: mysql-0.mysql.

apiVersion: v1

kind: Service

metadata:

name: mysql-read

labels:

app: mysql

spec:

ports:

- name: mysql

port: 3306

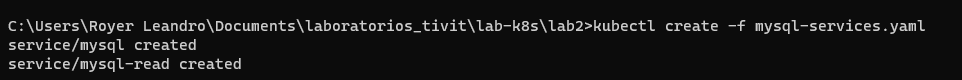
selector:

app: mysql

EOF

***Crear el Servicio***

kubectl create -f mysql-services.yaml



***Crear un YML default Storage***

cat <<EOF > mysql-storageclass.yaml

kind: StorageClass

apiVersion: storage.k8s.io/v1

metadata:

name: general

provisioner: kubernetes.io/aws-ebs

parameters:

type: gp2

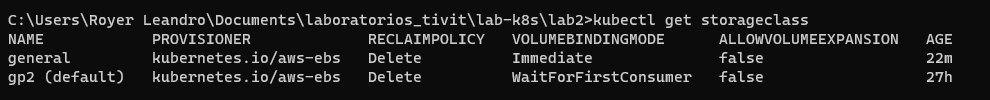
EOF

***Crear el storage class***

kubectl create -f mysql-storageclass.yaml



kubectl get storageclass



***Crear YML  MySQL StatefulSet:***

cat <<'EOF' > mysql-statefulset.yaml

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: mysql

spec:

selector:

matchLabels:

app: mysql

serviceName: mysql

replicas: 3

template:

metadata:

labels:

app: mysql

spec:

initContainers:

- name: init-mysql

image: mysql:5.7

command:

- bash

- "-c"

- |

set -ex

# Generate mysql server-id from pod ordinal index.

[[ `hostname` =~ -([0-9]+)$ ]] || exit 1

ordinal=${BASH\_REMATCH[1]}

echo [mysqld] > /mnt/conf.d/server-id.cnf

# Add an offset to avoid reserved server-id=0 value.

echo server-id=$((100 + $ordinal)) >> /mnt/conf.d/server-id.cnf

# Copy appropriate conf.d files from config-map to emptyDir.

if [[ $ordinal -eq 0 ]]; then

cp /mnt/config-map/master.cnf /mnt/conf.d/

else

cp /mnt/config-map/slave.cnf /mnt/conf.d/

fi

volumeMounts:

- name: conf

mountPath: /mnt/conf.d

- name: config-map

mountPath: /mnt/config-map

- name: clone-mysql

image: gcr.io/google-samples/xtrabackup:1.0

command:

- bash

- "-c"

- |

set -ex

# Skip the clone if data already exists.

[[ -d /var/lib/mysql/mysql ]] && exit 0

# Skip the clone on primary (ordinal index 0).

[[ `hostname` =~ -([0-9]+)$ ]] || exit 1

ordinal=${BASH\_REMATCH[1]}

[[ $ordinal -eq 0 ]] && exit 0

# Clone data from previous peer.

ncat --recv-only mysql-$(($ordinal-1)).mysql 3307 | xbstream -x -C /var/lib/mysql

# Prepare the backup.

xtrabackup --prepare --target-dir=/var/lib/mysql

volumeMounts:

- name: data

mountPath: /var/lib/mysql

subPath: mysql

- name: conf

mountPath: /etc/mysql/conf.d

containers:

- name: mysql

image: mysql:5.7

env:

- name: MYSQL\_ALLOW\_EMPTY\_PASSWORD

value: "1"

ports:

- name: mysql

containerPort: 3306

volumeMounts:

- name: data

mountPath: /var/lib/mysql

subPath: mysql

- name: conf

mountPath: /etc/mysql/conf.d

resources:

requests:

cpu: 100m

memory: 200Mi

livenessProbe:

exec:

command: ["mysqladmin", "ping"]

initialDelaySeconds: 30

timeoutSeconds: 5

readinessProbe:

exec:

# Check we can execute queries over TCP (skip-networking is off).

command: ["mysql", "-h", "127.0.0.1", "-e", "SELECT 1"]

initialDelaySeconds: 5

timeoutSeconds: 1

- name: xtrabackup

image: gcr.io/google-samples/xtrabackup:1.0

ports:

- name: xtrabackup

containerPort: 3307

command:

- bash

- "-c"

- |

set -ex

cd /var/lib/mysql

# Determine binlog position of cloned data, if any.

if [[ -f xtrabackup\_slave\_info ]]; then

# XtraBackup already generated a partial "CHANGE MASTER TO" query

# because we're cloning from an existing replica.

mv xtrabackup\_slave\_info change\_master\_to.sql.in

# Ignore xtrabackup\_binlog\_info in this case (it's useless).

rm -f xtrabackup\_binlog\_info

elif [[ -f xtrabackup\_binlog\_info ]]; then

# We're cloning directly from primary. Parse binlog position.

[[ `cat xtrabackup\_binlog\_info` =~ ^(.\*?)[[:space:]]+(.\*?)$ ]] || exit 1

rm xtrabackup\_binlog\_info

echo "CHANGE MASTER TO MASTER\_LOG\_FILE='${BASH\_REMATCH[1]}',\

MASTER\_LOG\_POS=${BASH\_REMATCH[2]}" > change\_master\_to.sql.in

fi

# Check if we need to complete a clone by starting replication.

if [[ -f change\_master\_to.sql.in ]]; then

echo "Waiting for mysqld to be ready (accepting connections)"

until mysql -h 127.0.0.1 -e "SELECT 1"; do sleep 1; done

echo "Initializing replication from clone position"

# In case of container restart, attempt this at-most-once.

mv change\_master\_to.sql.in change\_master\_to.sql.orig

mysql -h 127.0.0.1 <<EOF

$(<change\_master\_to.sql.orig),

MASTER\_HOST='mysql-0.mysql',

MASTER\_USER='root',

MASTER\_PASSWORD='',

MASTER\_CONNECT\_RETRY=10;

START SLAVE;

EOF

fi

# Start a server to send backups when requested by peers.

exec ncat --listen --keep-open --send-only --max-conns=1 3307 -c \

"xtrabackup --backup --slave-info --stream=xbstream --host=127.0.0.1 --user=root"

volumeMounts:

- name: data

mountPath: /var/lib/mysql

subPath: mysql

- name: conf

mountPath: /etc/mysql/conf.d

resources:

requests:

cpu: 100m

memory: 50Mi

volumes:

- name: conf

emptyDir: {}

- name: config-map

configMap:

name: mysql

volumeClaimTemplates:

- metadata:

name: data

spec:

accessModes: ["ReadWriteOnce"]

resources:

requests:

storage: 2Gi

storageClassName: general

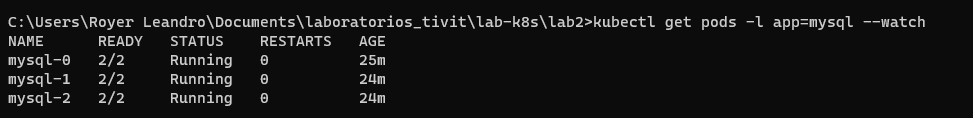
EOF

***Crear YML  MySQL StatefulSet:***

kubectl create -f mysql-statefulset.yaml

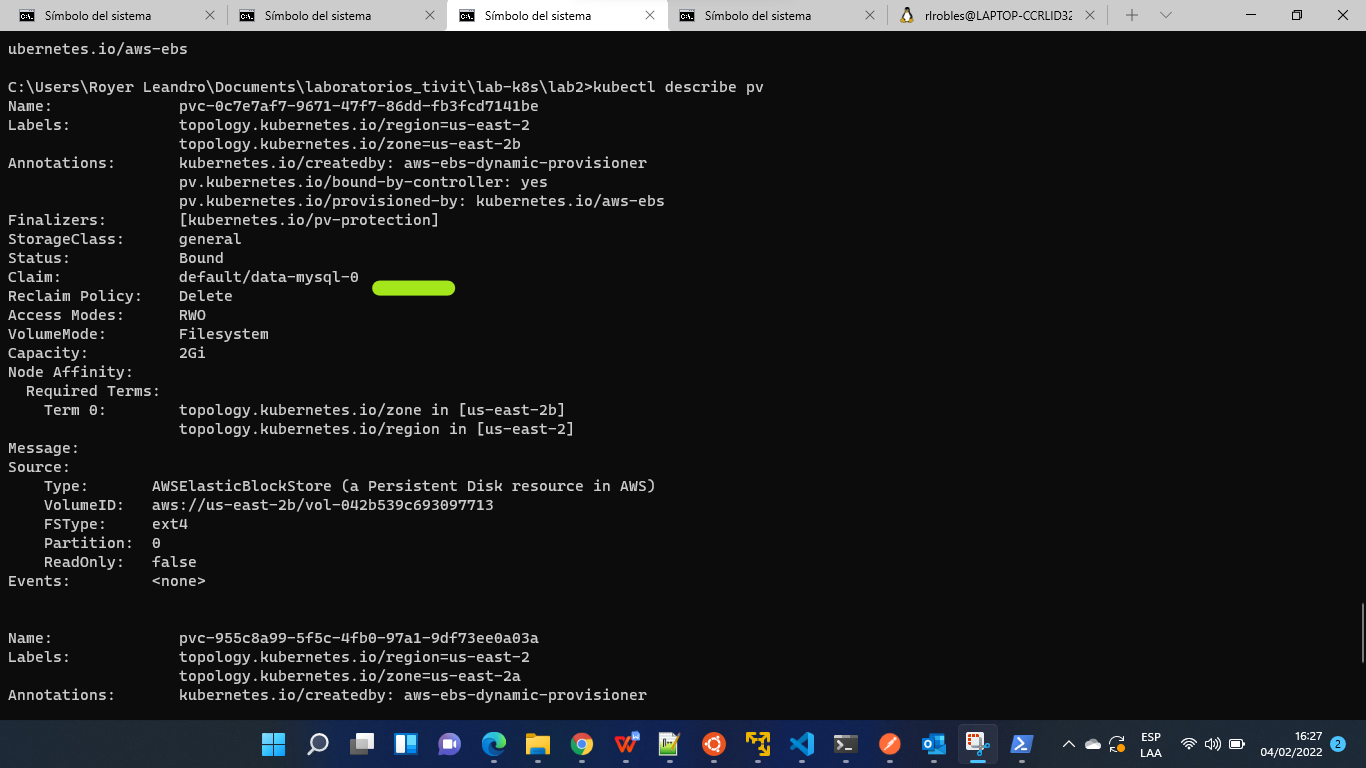


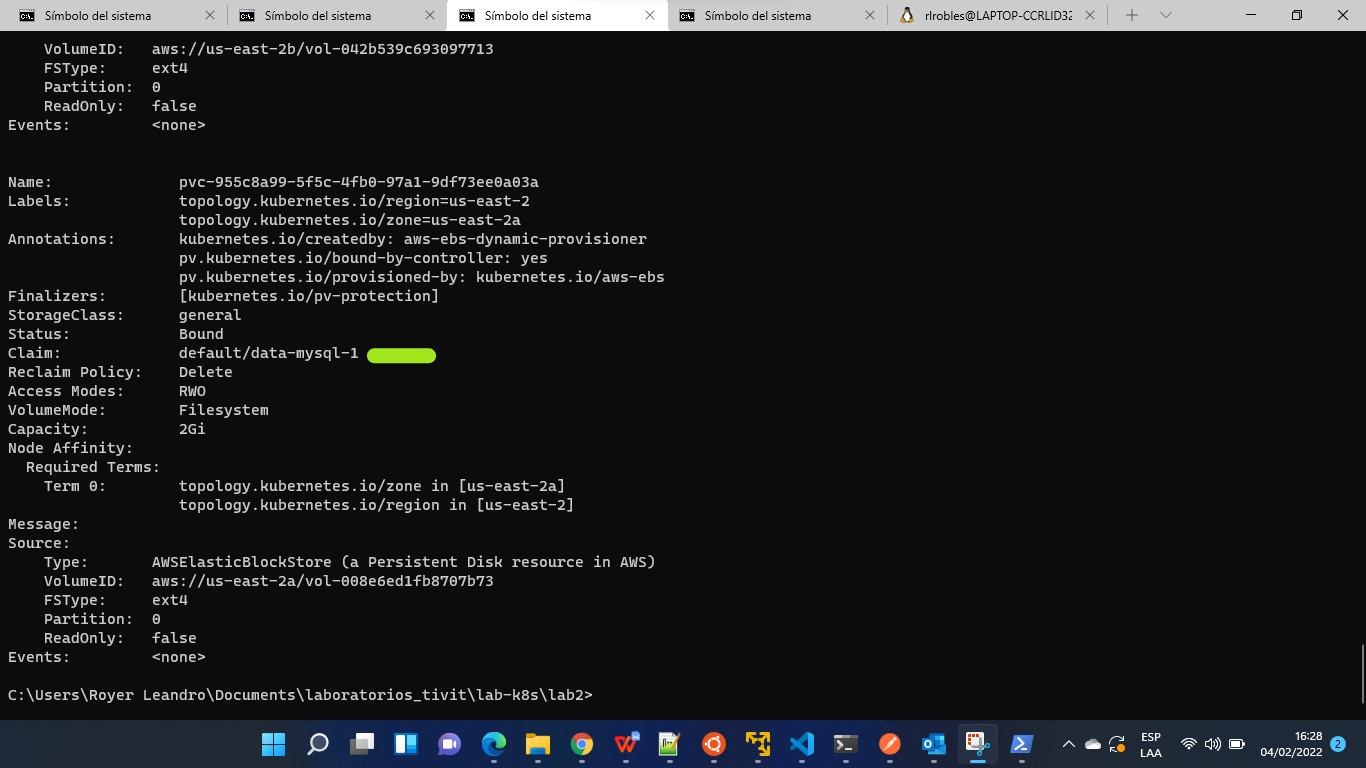
kubectl get pods -l app=mysql --watch



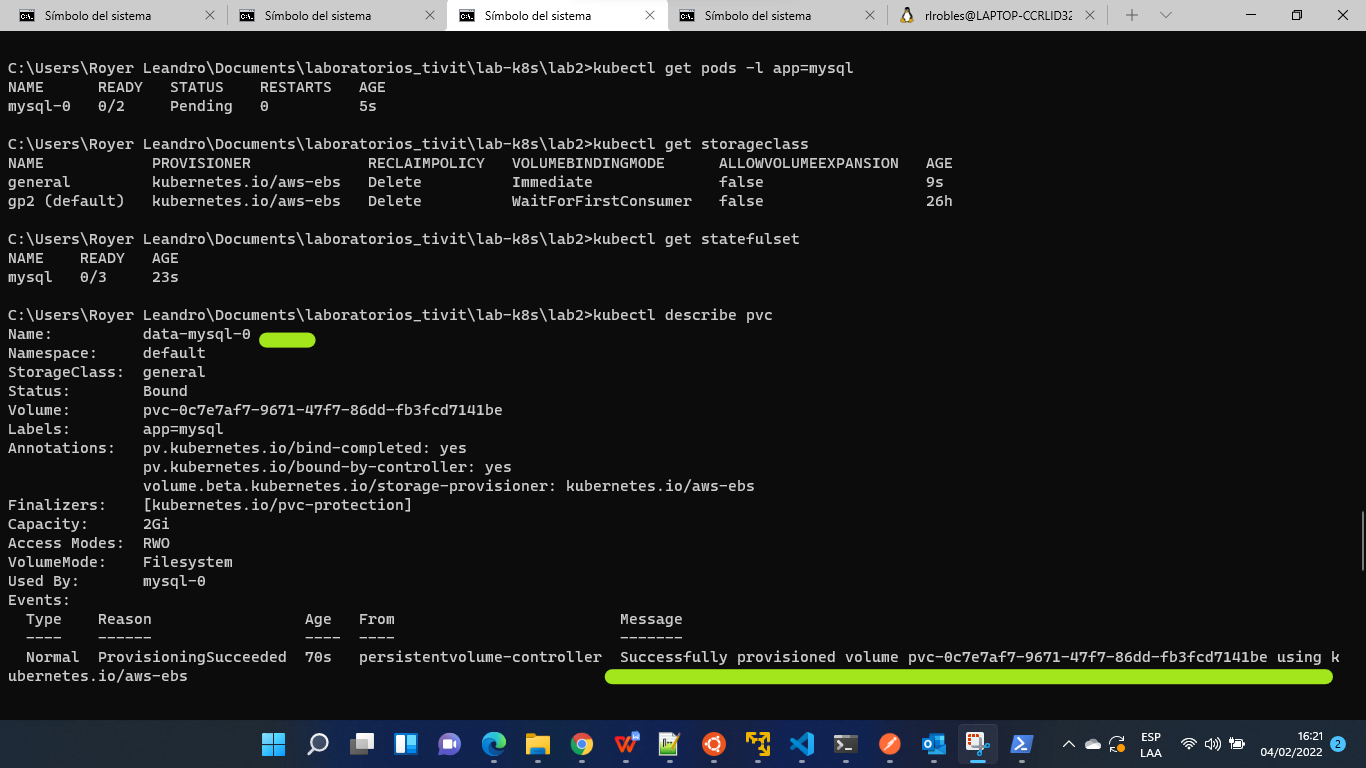
***Solicitar describe de los volumen group de los persistance group***

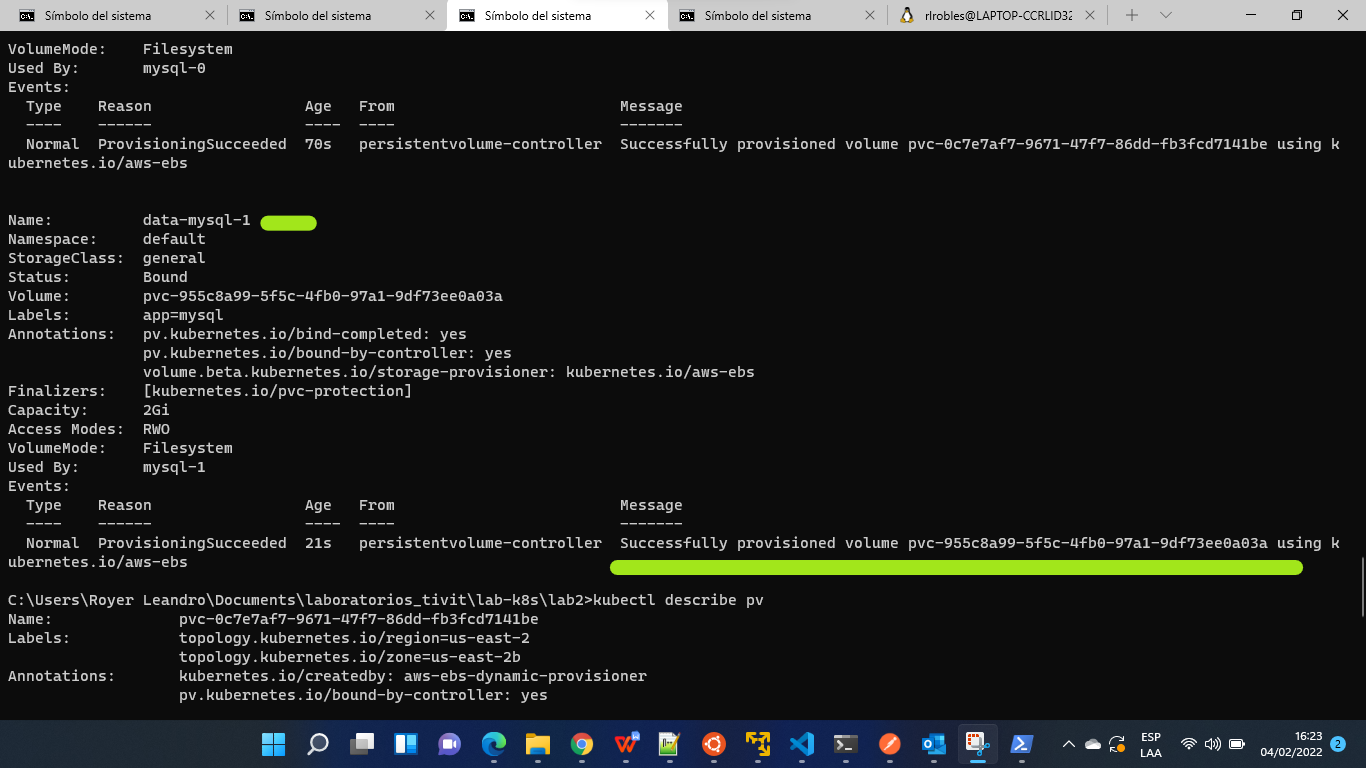
kubectl describe pv



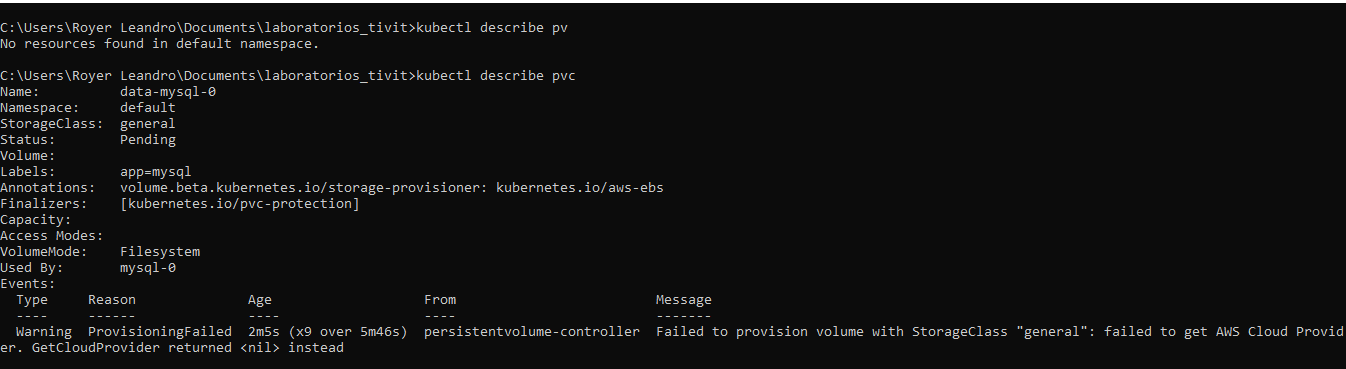


kubectl describe pvc



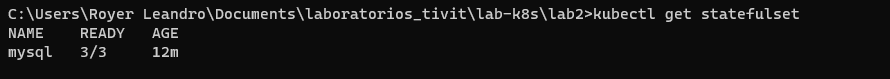


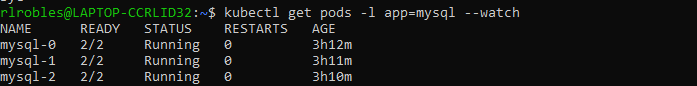
**Posibles Errores**



***Solicitar get***

kubectl get statefulset





***Crear sesion temporal en contenedor***

kubectl run mysql-client --image=mysql:5.7 -i -t --rm --restart=Never --\

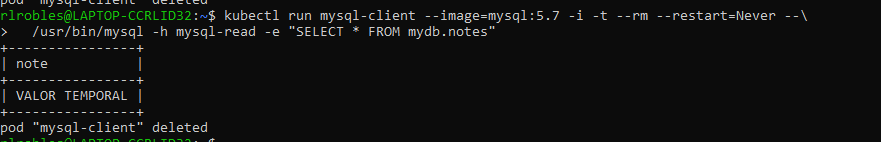
/usr/bin/mysql -h mysql-0.mysql -e "CREATE DATABASE mydb; CREATE TABLE mydb.notes (note VARCHAR(250)); INSERT INTO mydb.notes VALUES (VALOR TEMPORAL);"



***Ejecutar el siguiente query***

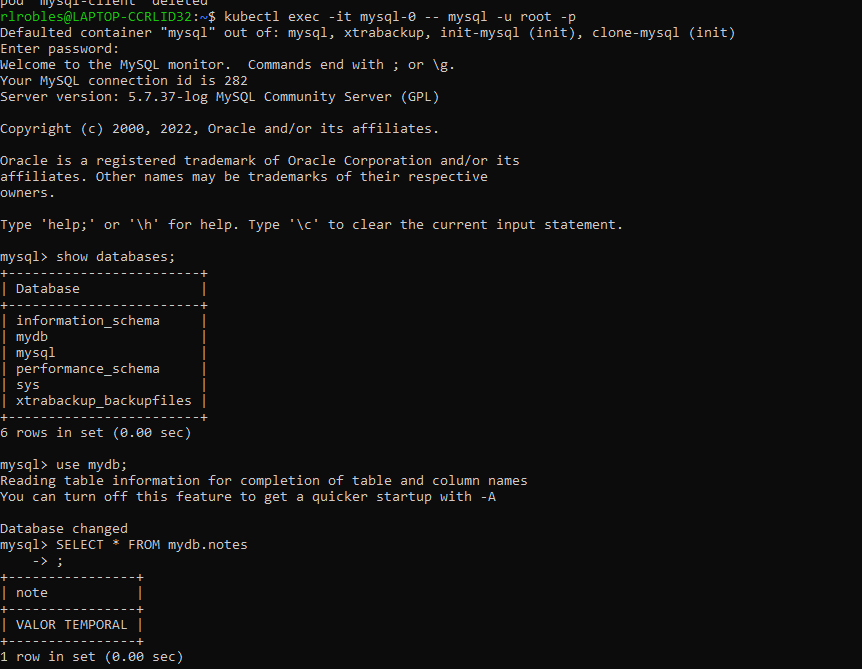
kubectl run mysql-client --image=mysql:5.7 -i -t --rm --restart=Never --\

/usr/bin/mysql -h mysql-read -e "SELECT \* FROM mydb.notes"



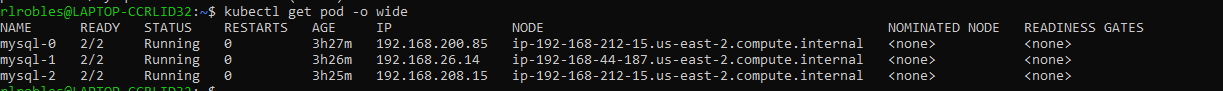
***CTRL +C***

(Opcional)



Ejecutar el siguiente commando para listar los pods:

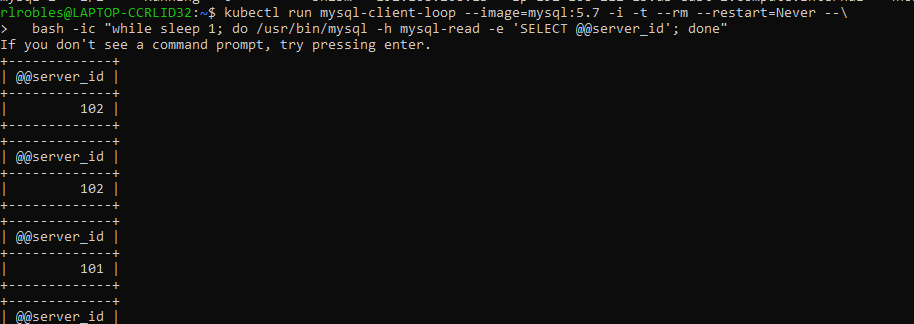
kubectl get pod -o wide



***Ejecute un comando SQL que genere la ID del servidor MySQL para confirmar que las solicitudes se distribuyen a diferentes pods***

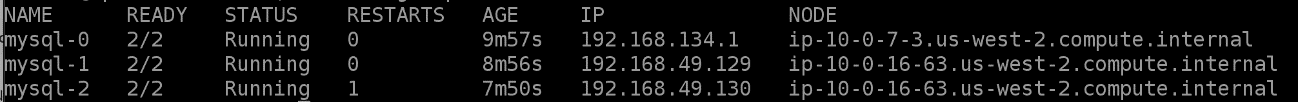
kubectl run mysql-client-loop --image=mysql:5.7 -i -t --rm --restart=Never --\

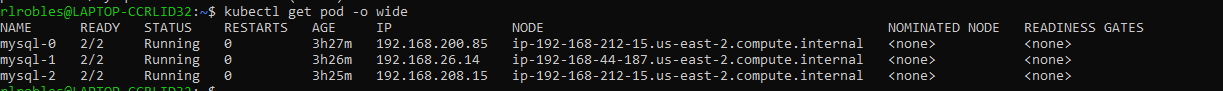
bash -ic "while sleep 1; do /usr/bin/mysql -h mysql-read -e 'SELECT @@server\_id'; done"



***Ejecutar el siguiente commando para listar los pods:***

kubectl get pod -o wide

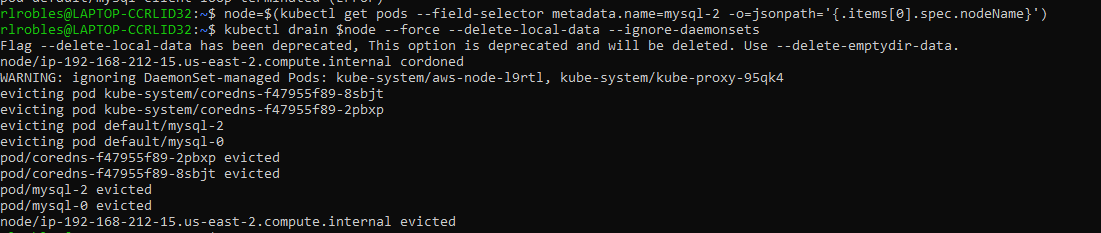




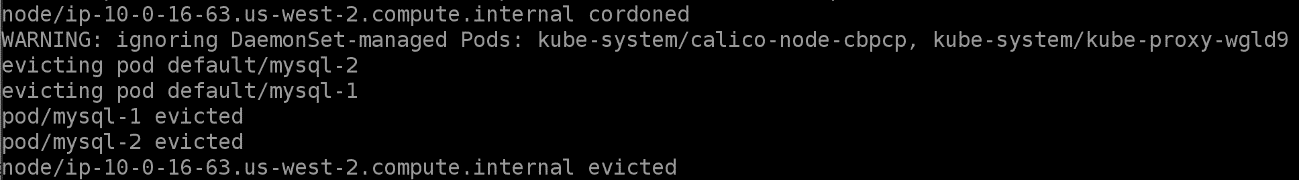
***Ingrese el siguiente comando para simular que el nodo que ejecuta el pod mysql-2 fuera de servicio por mantenimiento***

node=$(kubectl get pods --field-selector metadata.name=mysql-2 -o=jsonpath='{.items[0].spec.nodeName}')

kubectl drain $node --force --delete-local-data --ignore-daemonsets

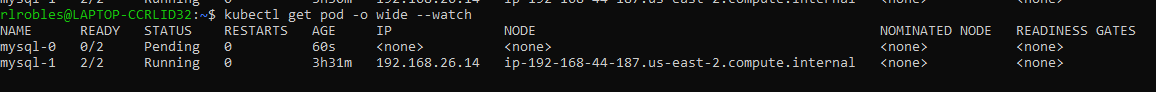


***donde reemplaza la variable de entorno del nodo se establece en el nombre del nodo que ejecuta mysql-2 usando un selector de campo y una salida jsonpath para seleccionar el nombre de nodo del Pod. Se parecerá a ip-10-0 - # - #. Us-west-2.compute-internal. El comando de drenaje evita que se programen nuevos pods en el nodo y luego desaloja los pods existentes programados para él.***



***Observe cómo se reprograma el pod mysql-2 en un nodo diferente:***

kubectl get pod -o wide –watch



***Descordone el nodo que drenó para que los pods se puedan programar en él nuevamente:***

kubectl uncordon $node

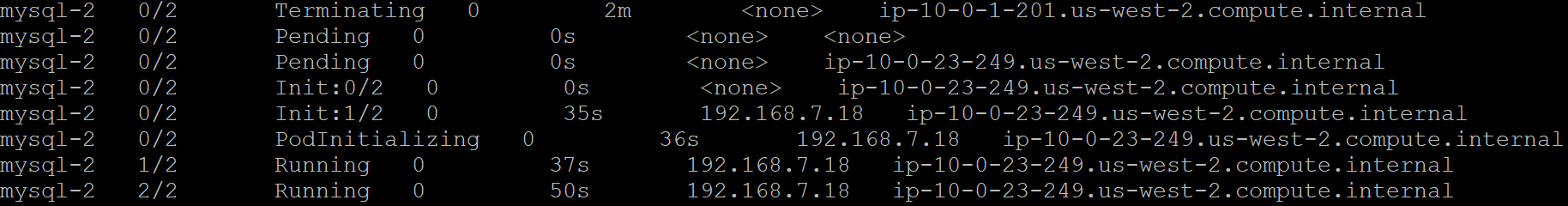


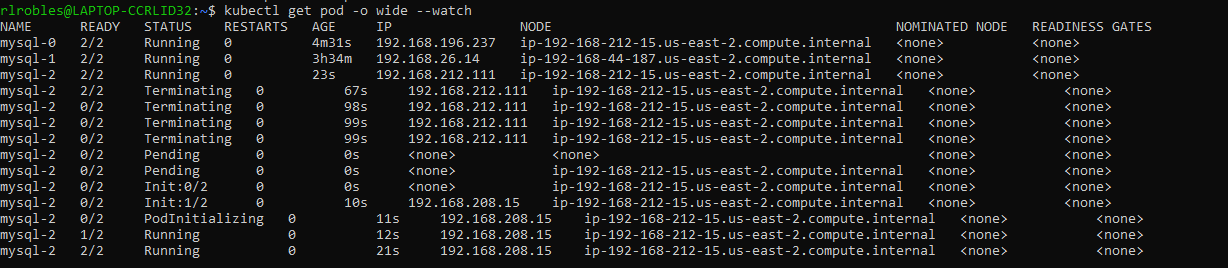
***Elimine el pod mysql-2 para simular una falla de nodo y observe cómo se reprograma automáticamente:***

kubectl delete pod mysql-2



kubectl get pod mysql-2 -o wide –watch





***CTRL +C***

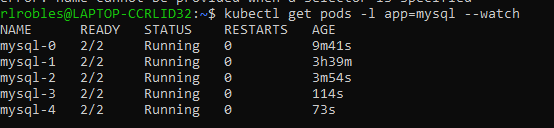
***Escale el número de réplicas hasta 5:***

kubectl scale --replicas=5 statefulset mysql



***Observe cómo se programan nuevos pods en el clúster:***

kubectl get pods -l app=mysql –watch

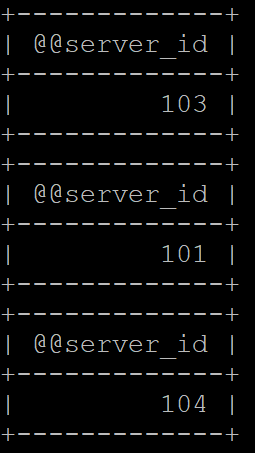


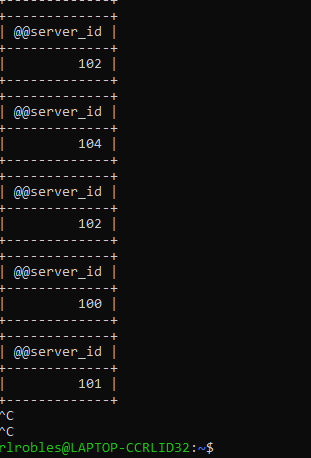
***CTRL +C***

Verifique que vea las nuevas ID de servidor MySQL:

kubectl run mysql-client-loop --image=mysql:5.7 -i -t --rm --restart=Never --\

bash -ic "while sleep 1; do /usr/bin/mysql -h mysql-read -e 'SELECT @@server\_id'; done"

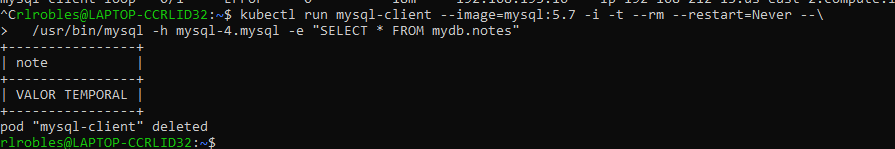




Confirme que los datos estén replicados en el nuevo pod mysql-4:

kubectl run mysql-client --image=mysql:5.7 -i -t --rm --restart=Never --\

/usr/bin/mysql -h mysql-4.mysql -e "SELECT \* FROM mydb.notes"

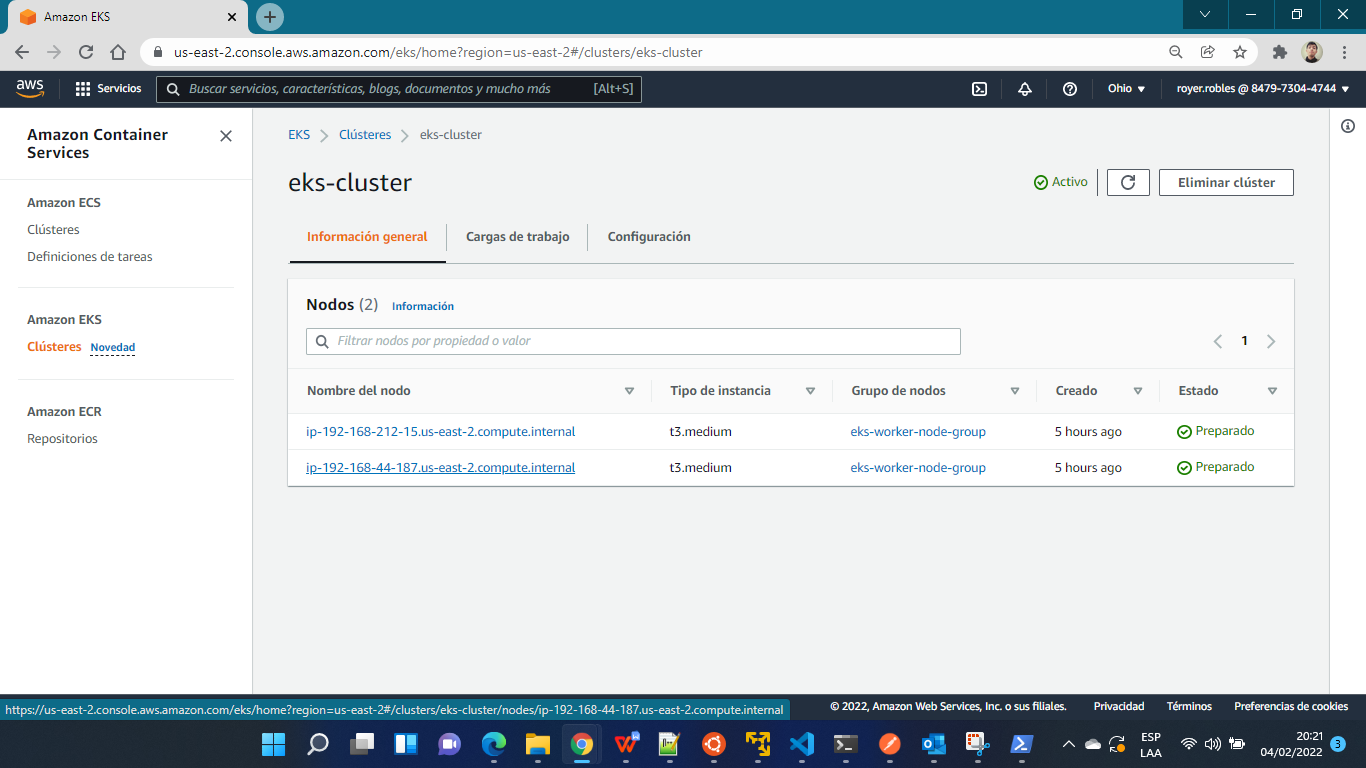


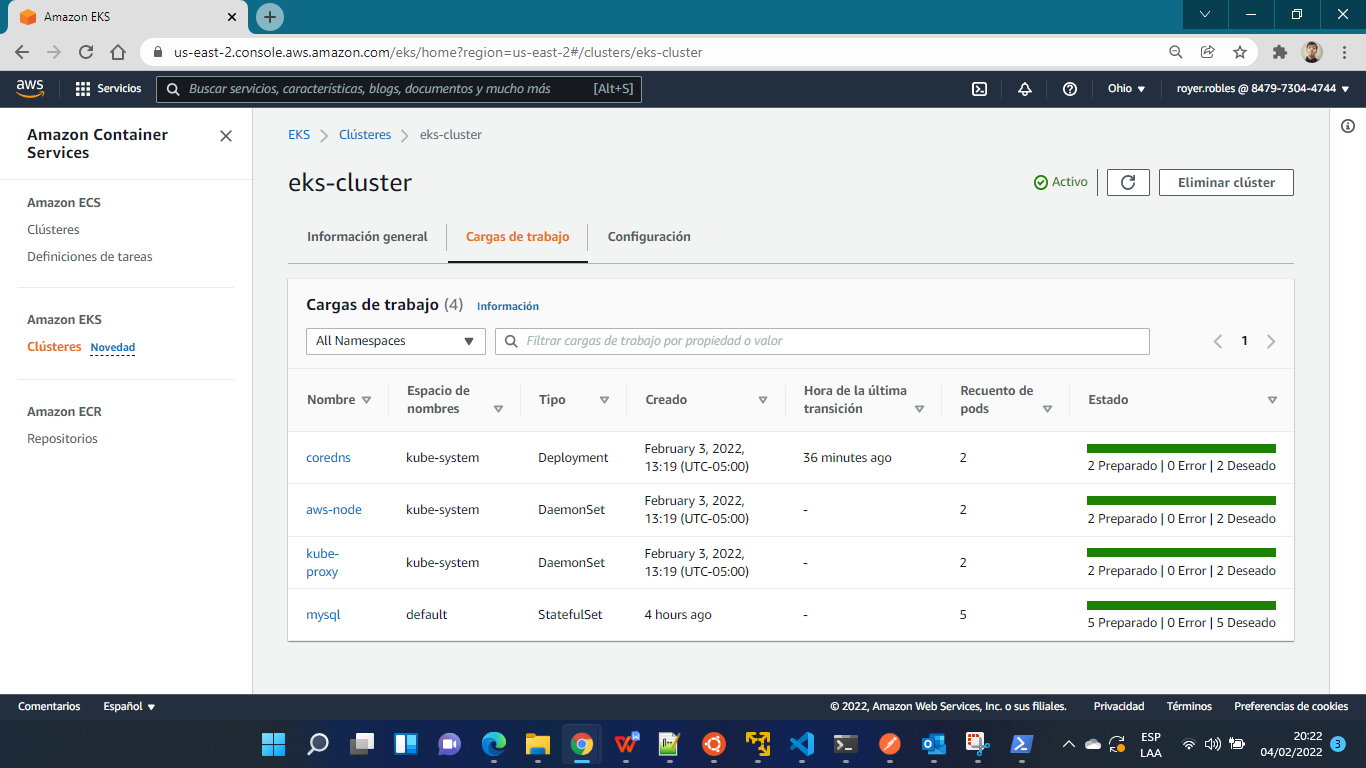
Muestra la IP virtual interna del punto final de lectura de mysql:

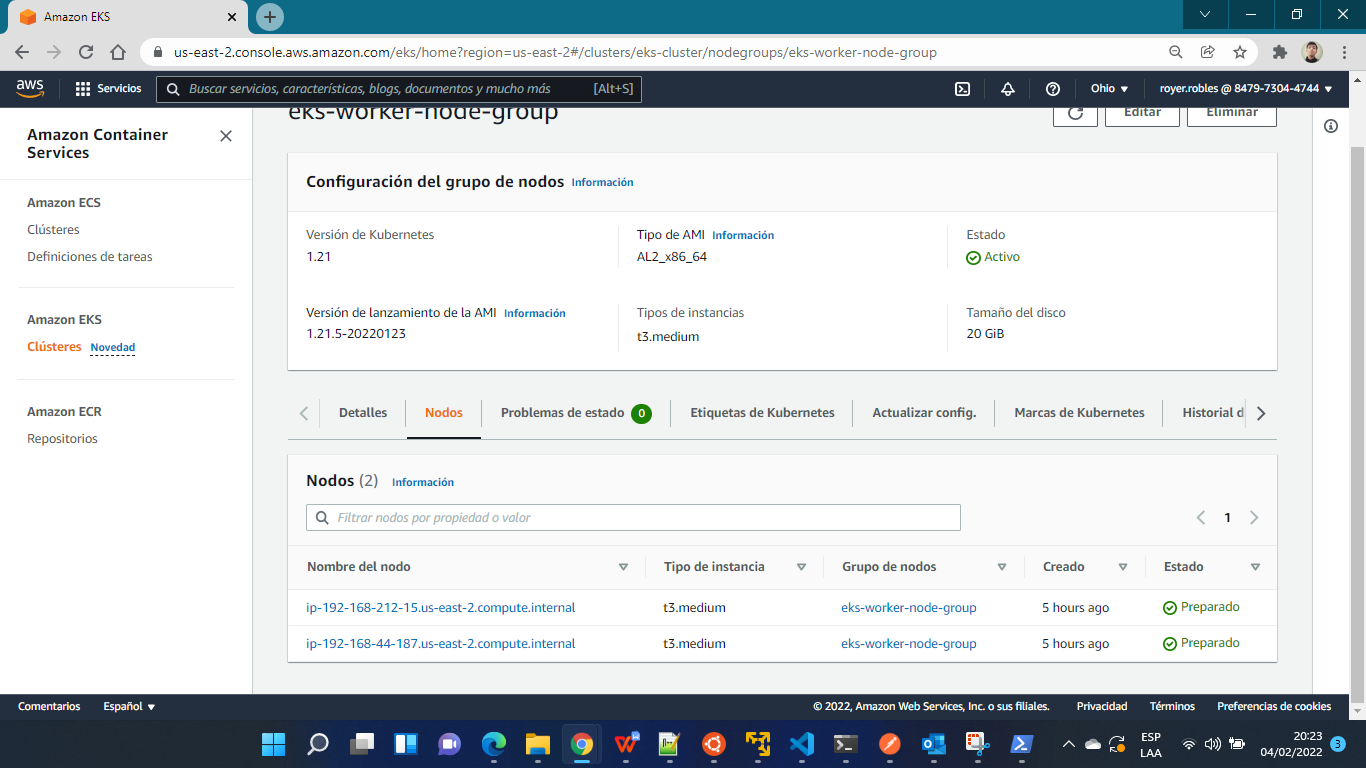
kubectl get services mysql-read

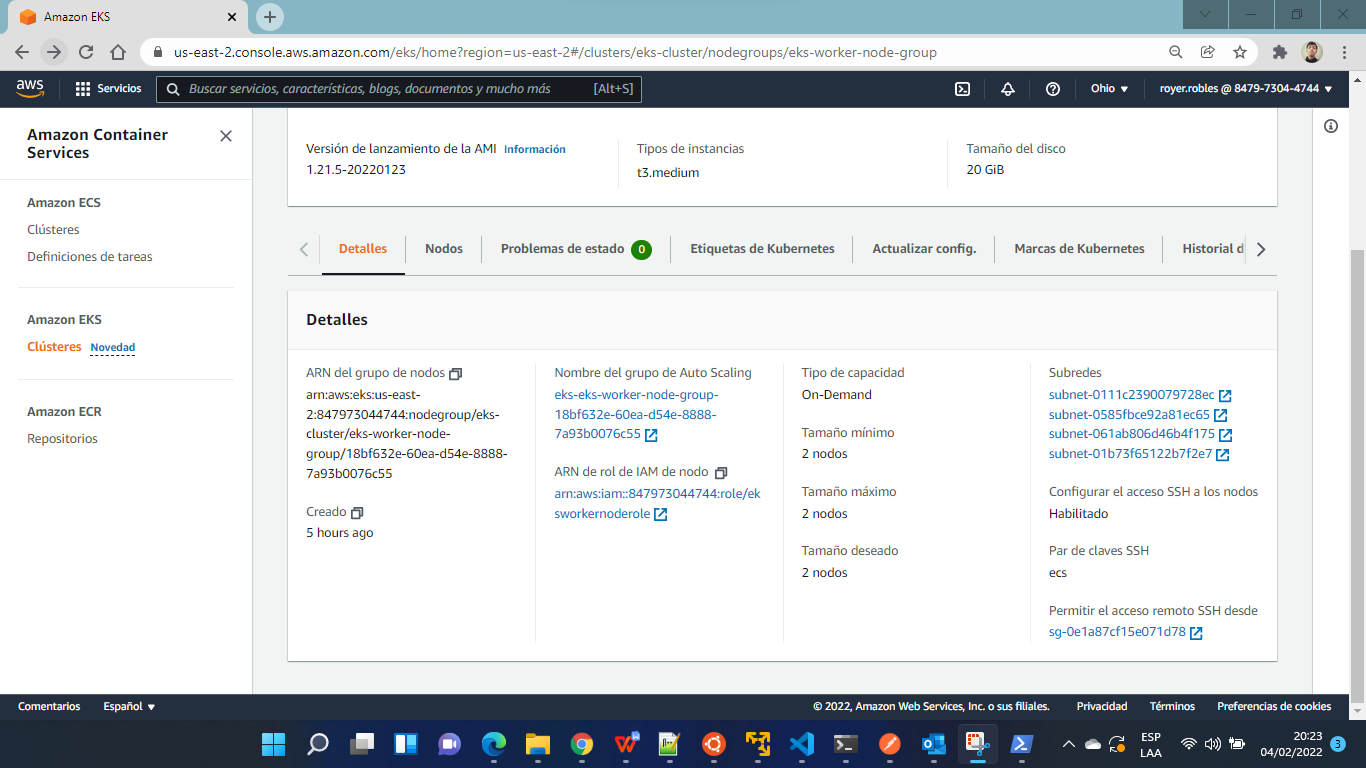
alt











Diagrama

Descripción generada automáticamente