

**UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA  
FACULTAD DE PRODUCCIÓN Y SERVICIOS  
ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN**



*CLOUD*

---

Laboratorio : Application - K8s

---

**SEMESTRE:**

*2025 - A*

**DOCENTE:**

*MAMANI ALIAGA, ALVARO HENRY*

**PRESENTADO POR :**

*NIFLA LLALLACACHI, MANUEL ANGEL*

**CUI:**

*20202234*

**Arequipa – Perú**

**2025**

# Adaptación de CinemAQP con Docker a K8s

## 1. Introducción

En este proyecto se busca adaptar un sistema web compuesto por **Base de Datos (PostgreSQL)**, **Backend (FastAPI)** y **Frontend (Flask)** a un entorno de orquestación con **Kubernetes**. El objetivo es garantizar que el proyecto sea **escalable, portable y adaptable a clústeres de Kubernetes** en instancias de AWS EC2, incorporando buenas prácticas de persistencia, exposición de servicios y microservicios futuros.

La estrategia seguida hasta ahora se centró en levantar primero el proyecto con **Docker Compose**, validando la interacción entre los contenedores y su funcionalidad completa antes de migrar a Kubernetes.

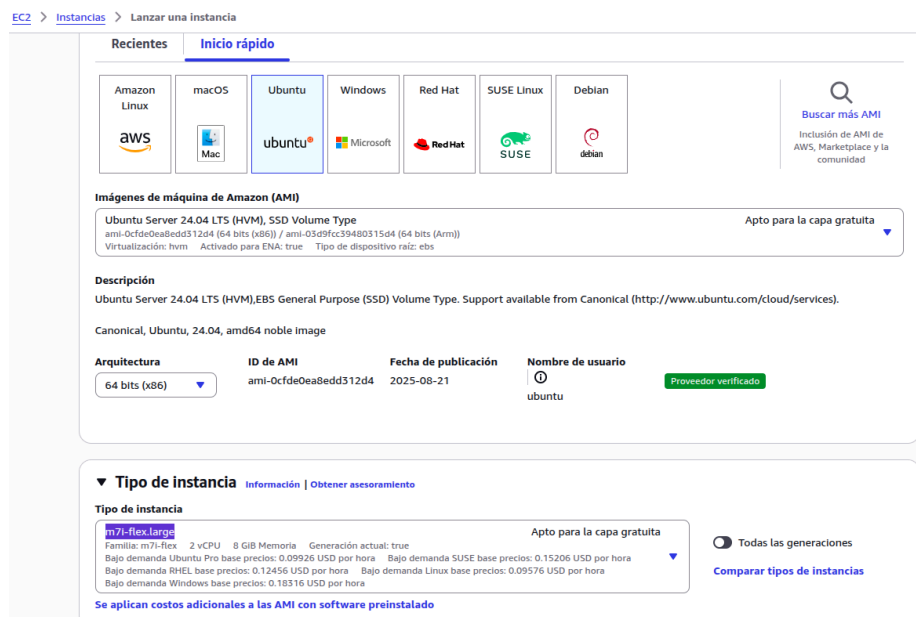
Github: [Link Repositorio](#)

## 4. Desarrollo y Pasos Iniciales Realizados

### 4.1. Creación de las instancias EC2:

Primeramente se procede a la creación de 3 instancias (master y workers) para poder crear el cluster. Para ello se crearon en base a las siguientes características:

- Se seleccionó **Ubuntu 20.04 LTS** como sistema operativo para garantizar compatibilidad con Docker y k3s.
- Se eligió el Tipo de instancia como **m7i-flex.large**.



EC2 > Instancias > Lanzar una instancia

Recientes Inicio rápido

Amazon Linux macOS Ubuntu Windows Red Hat SUSE Linux Debian

aws Mac ubuntu Microsoft Red Hat SUSE Debian

Buscar más AMI  
Inclusión de AMI de AWS, Marketplace y la comunidad

Imágenes de máquina de Amazon (AMI)

Ubuntu Server 24.04 LTS (HVM), SSD Volume Type  
ami-0cfde0ea8edd312d4 (64 bits (x86)) / ami-05d9fcc39480315d4 (64 bits (Arm))  
Virtualización: hvm Activado para ENA: true Tipo de dispositivo raíz: ebs

Apto para la capa gratuita

Descripción  
Ubuntu Server 24.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).  
Canonical, Ubuntu, 24.04, amd64 noble image

Arquitectura ID de AMI Fecha de publicación Nombre de usuario

64 bits (x86) ami-0cfde0ea8edd312d4 2025-08-21 ubuntu

Proveedor verificado

Tipo de instancia Información Obtener asesoramiento

Tipo de instancia

m7i-flex.large Apto para la capa gratuita

Familia: m7i-flex 2 vCPU 8 GiB Memoria Generación actual: true  
Bajo demanda Ubuntu Pro base precios: 0.09926 USD por hora Bajo demanda SUSE base precios: 0.15206 USD por hora  
Bajo demanda RHEL base precios: 0.12456 USD por hora Bajo demanda Linux base precios: 0.09576 USD por hora  
Bajo demanda Windows base precios: 0.18316 USD por hora

Todas las generaciones

Comparar tipos de instancias

Se aplican costos adicionales a las AMI con software preinstalado

- Se creó el **Security Group** para permitir:
  - SSH (puerto 22)**: acceso seguro a la instancia.
  - HTTP/Frontend (puerto 8080)**: acceso desde el navegador a la interfaz de usuario.
  - Otros puertos necesarios para el backend (por ejemplo 8000) y servicios adicionales si se requiriera como Postgre entre otros.

Tipo	Protocolo	Intervalo de puertos	Origen
TCP personalizado	TCP	8080	0.0.0.0/0
TCP personalizado	TCP	6443	0.0.0.0/0
TCP personalizado	TCP	8000	0.0.0.0/0
PostgreSQL	TCP	5432	0.0.0.0/0
HTTP	TCP	80	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0
SSH	TCP	22	0.0.0.0/0

- Se optó por crear llaves de acceso para el manejo de terminal por la computadora local a través de SSH:

**▼ Par de claves (inicio de sesión)** [Información](#)

Puede utilizar un par de claves para conectarse de forma segura a la instancia. Asegúrese de que tiene acceso al par de claves de la instancia.

**Nombre del par de claves - obligatorio**

k8ss ▼ 🔄 Crear

- Se lanzó en total 3 instancias (1 master y 2 workers)

**Instancias (3)** [Información](#)

🔍 Buscar instancia por atributo o etiqueta (case-sensitive) Todos los ... ▼

<input type="checkbox"/>	Name 🔗	ID de la instancia	Estado de la i...	Tipo de inst...	Comprobación de	Estad...
<input type="checkbox"/>	K8s_master	i-0ac3f4359ffd25237	🟢 En ejecución 🔍	m7i-flex.large	🟢 3/3 comprobaci...	Ver i...
<input type="checkbox"/>	K8s_node02	i-0386fe44cfa0550c1	🟢 En ejecución 🔍	m7i-flex.large	🟢 3/3 comprobaci...	Ver i...
<input type="checkbox"/>	K8s_node03	i-003b83d549079a78c	🟢 En ejecución 🔍	c7i-flex.large	🟢 3/3 comprobaci...	Ver i...

## 4.2. Conexión a la instancia EC2 vía SSH:

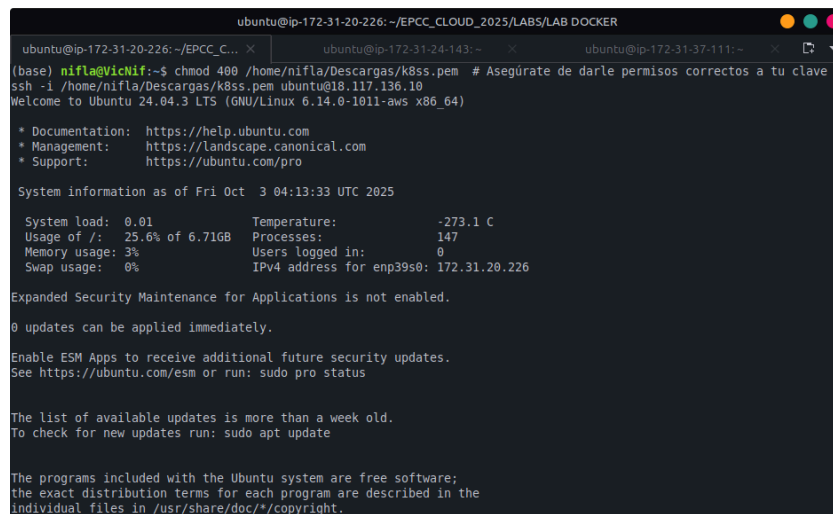
- Se utilizó el archivo de clave .pem generado al crear la instancia, garantizando acceso seguro con el siguiente comando. Y así conectarlos con:

```
chmod 400 /ruta/a/tu/clave.pem # Asegúrate de darle permisos correctos a tu clave
```

```
ssh -i /ruta/a/tu/clave.pem ubuntu@<ip-publica>
```

Reemplaza <ip-publica> con la dirección IP pública de cada instancia EC2.

- Se validó la conectividad y permisos del usuario en la instancia.



```
ubuntu@ip-172-31-20-226: ~/EPCC_CLOUD_2025/LABS/LAB DOCKER
(base) nifla@vicNif:~$ chmod 400 /home/nifla/Descargas/k8ss.pem # Asegúrate de darle permisos correctos a tu clave
ssh -i /home/nifla/Descargas/k8ss.pem ubuntu@18.117.136.10
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.14.0-1011-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Fri Oct  3 04:13:33 UTC 2025

System load:  0.01      Temperature:   -273.1 C
Usage of /:   25.6% of 6.71GB   Processes:    147
Memory usage: 3%          Users logged in: 0
Swap usage:  0%          IPv4 address for enp3s0: 172.31.20.226

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
```

### 4.3. Instalación de Docker y Docker Compose

- La instalación de Docker se realizó en las 3 instancias EC2, se ejecutó con el script oficial de Docker para instalar la versión más reciente compatible con Ubuntu.

```
# Actualizar y instalar dependencias
```

```
sudo apt update
```

```
sudo apt upgrade -y
```

```
# Instalar Docker
```

```
sudo apt install -y apt-transport-https ca-certificates curl sudo apt
install -y apt-transport-https ca-certificates curl
software-properties-common
```

```
curl -fsSL https://get.docker.com -o get-docker.sh
```

```
sudo sh get-docker.sh
```

```
# Verificar instalación
```

```
docker --version
```

# Agregar tu usuario al grupo Docker (opcional)

`sudo usermod -aG docker $USER`

```
ubuntu@ip-172-31-29-109:~$ sudo apt install -y apt-transport-https ca-certificates curl sudo apt install -y apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
E: Unable to locate package install
# Executing docker install script, commit: ae03142550f7efbfc843c8f55e7d2bc9351a74
+ sh -c apt-get -qq update >/dev/null
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get -y -qq install ca-certificates curl >/dev/null
+ sh -c install -m 0755 -d /etc/apt/keyrings
+ sh -c curl -fsSL "https://download.docker.com/linux/ubuntu/gpg" -o /etc/apt/keyrings/docker.asc
+ sh -c chmod a+r /etc/apt/keyrings/docker.asc
+ sh -c echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu noble stable" > /etc/apt/sources.list.d/docker.list
+ sh -c apt-get -qq update >/dev/null
+ sh -c DEBIAN_FRONTEND=noninteractive apt-get -y -qq install docker-ce docker-ce-cli containerd.io docker-compose-plugin docker-ce-rootless-extras docker-buildx-plugin doc
ker-model-plugin >/dev/null
Scanning processes...
Scanning candidates...
Scanning linux images...
+ sh -c docker version
Client: Docker Engine - Community
Version: 28.5.0
```

- La instalación de Docker Compose se realizó en las 3 Instancias donde: Se descargó la versión estable de Docker Compose y se configuraron permisos de ejecución.

`sudo curl -L`

`"https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose`

`sudo chmod +x /usr/local/bin/docker-compose`

#Verificar la instalación de Docker Compose:

`docker-compose --version`

```
ubuntu@ip-172-31-29-109:~$ docker --version
Docker version 28.5.0, build 887030f
ubuntu@ip-172-31-29-109:~$ sudo usermod -aG docker $USER
ubuntu@ip-172-31-29-109:~$ sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %    %         Dload  Upload   Total   Spent    Left     Speed
  0  0  0    0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
100 12.1M 100 12.1M  0  0 51.3M  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
ubuntu@ip-172-31-29-109:~$ docker-compose --version
docker-compose version 1.29.2, build 5becea4c
```

- Seguido se empieza a clonar el Repositorio de GitHub donde esta el proyecto de orquestación de contenedores con Docker.

- Instalar Git (si no lo tienes):

`sudo apt install git`

- Clonar el repositorio de GitHub: Se usa el siguiente comando para clonar tu repositorio donde tienes el archivo docker-compose.yml:

```
git clone https://github.com/tu-usuario/el-repositorio.git
cd el-repositorio
```

#### 4.4. Ejecutar el Proyecto con Docker Compose

- En la misma terminal, estando dentro de la carpeta del repositorio donde está el archivo docker-compose.yml, se ejecuta:

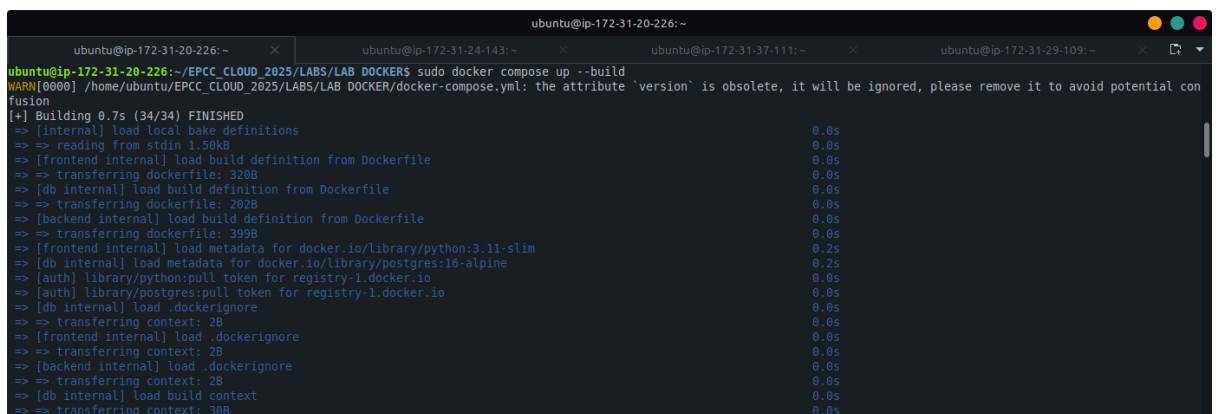
```
docker-compose up -d --build
```

Esto levantará los contenedores de frontend, backend y base de datos.

- Se Verifica que los contenedores estén corriendo:

```
docker ps
```

Esto te mostrará los contenedores que están corriendo, junto con los puertos expuestos.



```
ubuntu@ip-172-31-20-226:~$ sudo docker compose up --build
WARN[0000] /home/ubuntu/EPCC_CLOUD_2025/LABS/LAB DOCKER/docker-compose.yml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential con
fusion
[+] Building 0.7s (34/34) FINISHED
=> [internal] load local bake definitions                                0.0s
=> => reading from stdin 1.50kB                                          0.0s
=> [frontend internal] load build definition from Dockerfile            0.0s
=> => transferring dockerfile: 320B                                       0.0s
=> [db internal] load build definition from Dockerfile                  0.0s
=> => transferring dockerfile: 202B                                       0.0s
=> [backend internal] load build definition from Dockerfile             0.0s
=> => transferring dockerfile: 390B                                       0.0s
=> [frontend internal] load metadata for docker.io/library/python:3.11-slim 0.2s
=> [db internal] load metadata for docker.io/library/postgres:16-alpine 0.2s
=> [auth] library/python:pull token for registry-1.docker.io           0.0s
=> [auth] library/postgres:pull token for registry-1.docker.io          0.0s
=> [db internal] load .dockerignore                                      0.0s
=> => transferring context: 2B                                             0.0s
=> [frontend internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                             0.0s
=> [backend internal] load .dockerignore                                0.0s
=> => transferring context: 2B                                             0.0s
=> [db internal] load build context                                      0.0s
=> => transferring context: 39B                                           0.0s
```

#### 4.5. Acceder al Proyecto desde el Navegador desde un EC2

- Para acceder al frontend se utiliza de forma local: <http://localhost:8080>, pero en un EC2 se ejecuta en el navegador:

```
http://<IP-publica-EC2>:8080
```

### 5. Creación del Cluster

- Instalar k3s (Master Node):** En la instancia EC2 que será el **Master Node**, se ejecuta:

```
curl -sfL https://get.k3s.io | sh -
```

Esto instalará y configurará automáticamente k3s. Este script instala un clúster de Kubernetes con un solo nodo master y Traefik como Ingress Controller.

Se verifica que el Master Node está funcionando con :

```
sudo k3s kubect1 get nodes
```

- **Obtener el Token de Conexión para los Nodos Worker:** Para conectar los **Worker Nodes** al Master Node, se necesita un **token de acceso**. Para ello se ejecuta este comando en el **Master Node** para obtener el token:

```
sudo cat /var/lib/rancher/k3s/server/node-token
```

Se guarda este token, para usarlos en los Worker Nodes.

- **Instalar k3s en los Nodos Worker:** En cada Worker Node (nuevas instancias EC2), se conecta por vía SSH y ejecuta lo siguiente:

*# Sustituir <MASTER\_NODE\_IP> con la IP de tu nodo master*

*# Y <NODE\_TOKEN> con el token obtenido antes*

```
curl -sfL https://get.k3s.io | K3S_URL=https://<MASTER_NODE_IP>:6443  
K3S_TOKEN=<NODE_TOKEN> sh -
```

Esto hará que cada **Worker Node** se registre con el **Master Node**.

```
ubuntu@ip-172-31-20-226:~$ sudo k3s kubect1 get nodes
NAME                                STATUS    ROLES    AGE     VERSION
master-node                         Ready    master   10m     v1.21.4+k3s1
worker-node-1                       Ready    <none>    5m      v1.21.4+k3s1
worker-node-2                       Ready    <none>    5m      v1.21.4+k3s1

ubuntu@ip-172-31-29-109:~$ docker-compose --version
docker-compose version 1.29.2, build 5becea4c
ubuntu@ip-172-31-29-109:~$ curl -sfL https://get.k3s.io | K3S_URL=https://3.129.52.83:6443 K3S_TOKEN=K1057e7b084a13da91rver:e7784c15eea7a30f7a071ae155b685c9 sh -
[INFO] Finding release for channel stable
[INFO] Using v1.33.5+k3s1 as release
[INFO] Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.33.5+k3s1/sha256sum-amd64.txt
[INFO] Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.33.5+k3s1/k3s
[INFO] Verifying binary download
[INFO] Installing k3s to /usr/local/bin/k3s
[INFO] Skipping installation of SELinux RPM
[INFO] Creating /usr/local/bin/kubect1 symlink to k3s
[INFO] Creating /usr/local/bin/crictl symlink to k3s
[INFO] Skipping /usr/local/bin/ctr symlink to k3s, command exists in PATH at /usr/bin/ctr
[INFO] Creating killall script /usr/local/bin/k3s-killall.sh
[INFO] Creating uninstall script /usr/local/bin/k3s-agent-uninstall.sh
[INFO] env: Creating environment file /etc/systemd/system/k3s-agent.service.env
[INFO] systemd: Creating service file /etc/systemd/system/k3s-agent.service
[INFO] systemd: Enabling k3s-agent unit
Created symlink /etc/systemd/system/multi-user.target.wants/k3s-agent.service → /etc/systemd/system/k3s-agent.service.
[INFO] systemd: Starting k3s-agent
ubuntu@ip-172-31-29-109:~$
```

- En el **Master Node**, corre el siguiente comando para asegurarte de que todos los nodos (Master y Workers) estén correctamente registrados:

```
sudo k3s kubect1 get nodes
```

Deberías ver algo como esto:

NAME	STATUS	ROLES	AGE	VERSION
master-node	Ready	master	10m	v1.21.4+k3s1
worker-node-1	Ready	<none>	5m	v1.21.4+k3s1
worker-node-2	Ready	<none>	5m	v1.21.4+k3s1



```

ubuntu@ip-172-31-20-226:~
Last login: Sun Oct 5 19:57:38 2025 from 38.56.112.134
ubuntu@ip-172-31-20-226:~$ sudo k3s kubectl get nodes -o wide
NAME                STATUS    ROLES                  AGE      VERSION    INTERNAL-IP    EXTERNAL-IP    OS-IMAGE             KERNEL-VERSION    CONTAINER-RUNTIME
ip-172-31-20-226    Ready    control-plane,master   2d19h    v1.33.5+k3s1 172.31.20.226   <none>          Ubuntu 24.04.3 LTS   6.14.0-1014-aws   containerd://2.1.4-k3s1
ip-172-31-24-143    Ready    <none>                 2d19h    v1.33.5+k3s1 172.31.24.143   <none>          Ubuntu 24.04.3 LTS   6.14.0-1014-aws   containerd://2.1.4-k3s1
ip-172-31-37-111    Ready    <none>                 2d19h    v1.33.5+k3s1 172.31.37.111   <none>          Ubuntu 24.04.3 LTS   6.14.0-1014-aws   containerd://2.1.4-k3s1
ubuntu@ip-172-31-20-226:~$ sudo k3s kubectl get nodes
NAME                STATUS    ROLES                  AGE      VERSION
ip-172-31-20-226    Ready    control-plane,master   2d19h    v1.33.5+k3s1
ip-172-31-24-143    Ready    <none>                 2d19h    v1.33.5+k3s1
ip-172-31-37-111    Ready    <none>                 2d19h    v1.33.5+k3s1
ubuntu@ip-172-31-20-226:~$ sudo k3s kubectl get nodes
NAME                STATUS    ROLES                  AGE      VERSION
ip-172-31-20-226    Ready    control-plane,master   2d19h    v1.33.5+k3s1
ip-172-31-24-143    Ready    <none>                 2d19h    v1.33.5+k3s1
ip-172-31-37-111    Ready    <none>                 2d19h    v1.33.5+k3s1

```

## 6. Adaptación del Proyecto a Kubernetes

### 6.1. Construir y publicar imágenes en Docker Hub:

**Imagen (Docker)** = Empaqueta la app: código + runtime + dependencias + sistema base.

Así, donde sea que arranque (máquina local, un worker en AWS, etc.) se comporta igual.

#### ¿Para qué se crea?

Para que Kubernetes pueda descargar exactamente ese paquete y correrlo en cualquier nodo del clúster.

#### ¿Por qué subirla a un *hub/registry* (Docker Hub/ECR)?

Porque los nodos del clúster no tienen tu imagen local. Cuando se aplican los YAML, cada nodo la “pull-ea” desde el registry.

- Sin registry público/privado, K8s no sabría de dónde sacar la imagen.
- En compose local, `--build` compila la imagen en la máquina y la usaba ahí mismo; en K8s todos los nodos deben poder pull desde un sitio central (hub).

A. Para crear la imagen es necesario tener una cuenta y el usuario de Docker Hub:

<usuario> y nombres para las imágenes:

- <usuario>/cinema-backend:1.0
- <usuario>/cinema-frontend:1.0

B. Se ejecuta en el directorio del proyecto local donde están las carpetas de backend/, frontend/, db/:

*# Inicia sesión en Docker Hub*

*docker login*

*# Backend*

*cd backend*

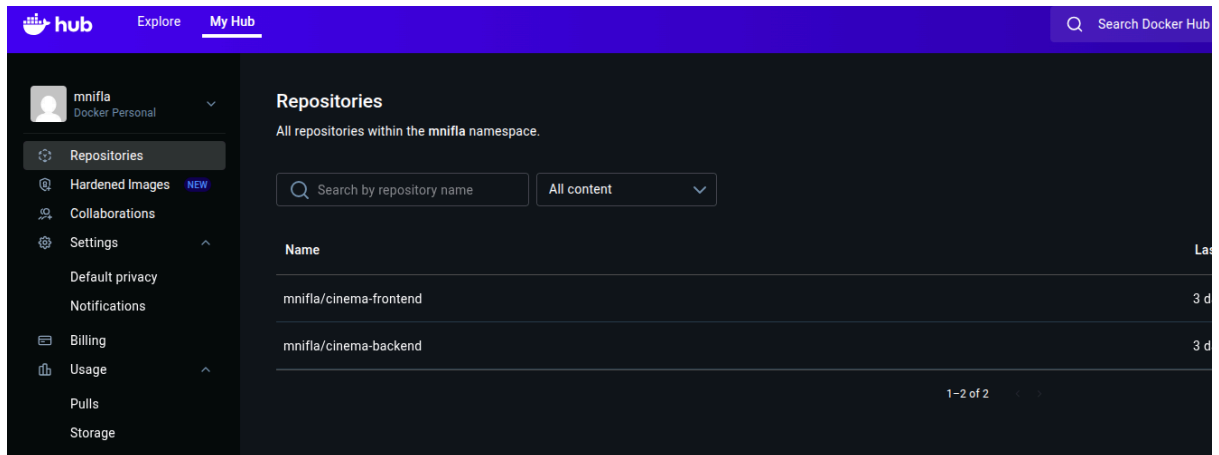
*docker build -t <usuario>/cinema-backend:1.0 .*

*docker push <usuario>/cinema-backend:1.0*

*cd ..*



```
# Frontend
cd frontend
docker build -t mnifla/cinema-frontend:1.0 .
docker push <usuario>/cinema-frontend:1.0
cd ..
```



C. Para la base de datos se usó la imagen oficial postgres:16-alpine y montaremos init.sql vía ConfigMap (K8s lo ejecuta en /docker-entrypoint-initdb.d).

## 6.2. Idealizar la arquitectura destino en K8s (k3s):

**Namespace:** cinema

**Postgres:** StatefulSet + PVC (almacenamiento local-path de k3s)

- Credenciales en Secret
- init.sql desde ConfigMap

**Backend:** Deployment (+ Service ClusterIP), variables para DB (usaremos DATABASE\_URL y variables separadas)

**Frontend (Flask UI):** Deployment (+ Service ClusterIP) con BACKEND\_BASE\_URL=http://backend:8000

**Ingress (Traefik k3s):**

- /api → backend:8000
- / → frontend:8080

## 6.3. Manifiestos K8s.

## ¿Qué son los archivos YAML y para qué sirven?

- Los YAML de Kubernetes son los “**manifiestos**”: documentos que describen el **estado deseado**.
- Kubernetes los lee y se encarga de **alcanzarlo y mantenerlo**.
- Para desplegar estos = ejecutar , se usa sobre esos archivos el comando:

*kubectl apply -f ...*

## ¿Cómo funcionan?

El API server guarda esos objetos; los controladores de K8s:

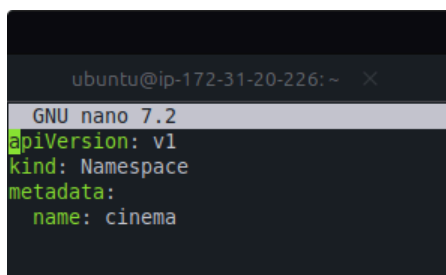
- crean **Pods** (según Deployment/StatefulSet),
- piden **imágenes** al registry,
- montan **volúmenes** (PVC),
- exponen **Services**,
- configuran **Ingress** en Traefik,
- y siguen **vigilando** que lo que corre **coincida** con lo que se pide.
- **Manifiesto** es el propio YAML (la “declaración” del estado deseado).

En el compose uno es quien “maneja” los contenedores. En K8s se declara qué quiere uno y K8s se encarga de cómo lograrlo y mantenerlo.

### D. Creación de los manifiestos:

#### a. Namespace [**00-namespace.yaml**]

Crea el “cajón” lógico **cinema** para agrupar todo (pods, servicios, secretos...). Orden y aislamiento.



```
ubuntu@ip-172-31-20-226: ~  
GNU nano 7.2  
apiVersion: v1  
kind: Namespace  
metadata:  
  name: cinema
```

#### b. Secret (credenciales Postgres) [**01-postgres-secret.yaml**]

Guarda **credenciales** (usuario/clave) de Postgres en un objeto **Secret** (no en texto plano en los Deployments).

```
ubuntu@ip-172-31-20-226:~  
GNU nano 7.2  
apiVersion: v1  
kind: Secret  
metadata:  
  name: postgres-secret  
  namespace: cinema  
type: Opaque  
stringData:  
  POSTGRES_USER: cinema  
  POSTGRES_PASSWORD: cinema  
  POSTGRES_DB: cinema
```

c. ConfigMap con `init.sql` [02-postgres-init-configmap.yaml]

Contiene el **init.sql** (semilla de tablas/datos) u otras configuraciones. Es un **ConfigMap** que la BD lee al iniciar.

```
ubuntu@ip-172-31-20-226:~  
GNU nano 7.2  
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: postgres-init  
  namespace: cinema  
data:  
  init.sql: |  
    -- Crea tablas y seeds (fragmento del ZIP; ajusta si deseas)  
    CREATE TABLE IF NOT EXISTS movies (  
      id SERIAL PRIMARY KEY,  
      title TEXT NOT NULL,  
      description TEXT,  
      duration_minutes INT NOT NULL CHECK (duration_minutes > 0)  
    );  
  
    CREATE TABLE IF NOT EXISTS screenings (  
      id SERIAL PRIMARY KEY,  
      movie_id INT NOT NULL REFERENCES movies(id) ON DELETE CASCADE,  
      show_time TIMESTAMP NOT NULL,  
      price NUMERIC(10,2) NOT NULL CHECK (price > 0)  
    );  
  
    CREATE TABLE IF NOT EXISTS tickets (  
      id SERIAL PRIMARY KEY,  
      screening_id INT NOT NULL REFERENCES screenings(id) ON DELETE CASCADE,  
      customer_name TEXT NOT NULL,  
      quantity INT NOT NULL CHECK (quantity > 0),  
      purchased_at TIMESTAMP NOT NULL DEFAULT NOW()  
    );  
  
    INSERT INTO movies (title, description, duration_minutes) VALUES  
    ('Inception', 'Ciencia ficción y sueños.', 148),  
    ('Interstellar', 'Viajes interestelares.', 169),  
    ('El Origen de Arequipa', 'Ficción local demo.', 102)  
    ON CONFLICT DO NOTHING;  
  
    INSERT INTO screenings (movie_id, show_time, price) VALUES  
    (1, NOW() + INTERVAL '1 hour', 18.50),  
    (1, NOW() + INTERVAL '4 hours', 18.50),  
    (2, NOW() + INTERVAL '2 hours', 20.00),
```

## d. Postgres StatefulSet + Service [03-postgres.yaml]

Define **Postgres** como **StatefulSet** (porque guarda estado) + su **Service** interno + su **PVC** (disco donde persiste datos).

```
ubuntu@ip-172-31-20-226:~$ nano 04-backend.yaml
GNU nano 7.2
apiVersion: v1
kind: Service
metadata:
  name: postgres
  namespace: cinema
spec:
  type: ClusterIP
  selector:
    app: postgres
  ports:
    - name: pg
      port: 5432
      targetPort: 5432
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres
  namespace: cinema
spec:
  serviceName: postgres
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: postgres:16-alpine
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 5432
            name: pg
          envFrom:
            - secretRef:
                name: postgres-secret
          volumeMounts:
            - name: pgdata
              mountPath: /var/lib/postgresql/data
            - name: init-sql
              mountPath: /docker-entrypoint-initdb.d
```

## e. Backend (Deployment + Service) [04-backend.yaml]

Define el **Deployment** del backend (cuántas réplicas, qué imagen usar, probes, variables, init “wait-db”) + su **Service** interno (DNS **backend** dentro del cluster).

```
GNU nano 7.2
apiVersion: v1
kind: Service
metadata:
  name: backend
  namespace: cinema
spec:
  type: ClusterIP
  selector:
    app: backend
  ports:
    - name: http
      port: 8000
      targetPort: 8000
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
  namespace: cinema
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      initContainers:
        - name: wait-dns
          image: busybox:1.36
          command: ["sh", "-c", "until getent hosts backend; do sleep 1; done"]
        - name: wait-db
          image: postgres:16-alpine
          env:
            - name: PGPASSWORD
              valueFrom:
                secretKeyRef:
                  name: postgres-secret
                  key: POSTGRES_PASSWORD
          command:
            - sh
            - -c
            - |
```

## f. Frontend (Deployment + Service) [05-frontend.yaml]

Define el **Deployment** del frontend + su **Service** interno (DNS frontend).

```
GNU nano 7.2
apiVersion: v1
kind: Service
metadata:
  name: frontend
  namespace: cinema
spec:
  type: ClusterIP
  selector:
    app: frontend
  ports:
    - name: http
      port: 8080
      targetPort: 8080
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  namespace: cinema
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend
          image: mnifla/cinema-frontend:1.0
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 8080
          env:
            - name: BACKEND_BASE_URL
              value: http://backend:8080
          readinessProbe:
            httpGet:
              path: /
              port: 8080
            initialDelaySeconds: 10
            periodSeconds: 5
          livenessProbe:
```

## g. Ingress (Traefik en k3s) [06-ingress.yaml]

Crea las **rutas HTTP** externas (Traefik) para que desde Internet el tráfico llegue al **Service** del frontend/backend según el path/host.

```
niFla@VicNiF: ~
GNU nano 7.2
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: cinema-ingress
  namespace: cinema
spec:
  ingressClassName: traefik # <- en vez de la anotaci
  rules:
    - http:
        paths:
          - path: /api
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 8080
          - path: /
            pathType: Prefix
            backend:
              service:
                name: frontend
                port:
                  number: 8080
```

**6.4. Despliegue en k3s.**

- E. Para poner en marcha la ejecución de la app con k8s, por medio de k3s se despliegan los manifiestos con los siguientes comando:

# En el master k3s

```
sudo k3s kubectl apply -f /k8s/00-namespace.yaml
```

```
sudo k3s kubectl apply -f /k8s/01-postgres-secret.yaml
sudo k3s kubectl apply -f /k8s/02-postgres-init-configmap.yaml
sudo k3s kubectl apply -f /k8s/03-postgres.yaml
# Se espera a que postgres esté Ready (ver abajo) y luego:
sudo k3s kubectl apply -f /k8s/04-backend.yaml
sudo k3s kubectl apply -f /k8s/05-frontend.yaml
sudo k3s kubectl apply -f /k8s/06-ingress.yaml
```

El **orden** es importante: primero base (ns/secret/config), luego **BD**, después **backend**, luego **frontend**, y al final **ingress** para exponer.

- Se crea el **namespace** cinema. Todo lo demás vivirá ahí.
- Se carga las **credenciales** de la BD en un **Secret** (para usarlas sin ponerlas en claro en otros YAML).
- Se carga el init.sql/config en un **ConfigMap** para inicializar Postgres (tablas/datos base).
- Se levanta **Postgres** (StatefulSet + PVC + Service).  
**Esperamos** a que esté **Ready** porque el backend depende de él.
- Se levanta el **backend** (Deployment + Service). Tiene un **init** que espera a la BD.  
Cuando la BD está lista, el backend arranca y expone /movies, etc., para el frontend.
- Se levanta el **frontend** (Deployment + Service). Habla con el backend vía Service DNS interno.
- Se crea la **puerta de entrada** (Traefik): define por qué **URL/host/path** el tráfico externo llega a tus Services.

F. Para poder monitorear el cluster y los Pod's se usan los siguientes comandos:

```
sudo k3s kubectl get nodes
sudo k3s kubectl -n cinema get pods -o wide
sudo k3s kubectl -n cinema get svc
sudo k3s kubectl -n cinema get ingress
```

- Mira si los **nodos** están **Ready**. Si un nodo está NotReady, ahí no se podrán crear pods.

```
ubuntu@ip-172-31-20-226:~/EPCC_CLOUD_2025/LABS/LAB_K8S/k8s$ sudo k3s kubectl get nodes
NAME                STATUS    ROLES    AGE   VERSION
ip-172-31-20-226    Ready     control-plane,master   2d14h   v1.33.5+k3s1
ip-172-31-24-143    Ready     <none>    2d14h   v1.33.5+k3s1
ip-172-31-37-111    Ready     <none>    2d14h   v1.33.5+k3s1
```

- Ves **cada Pod**, su **estado** (Running/Init/CrashLoopBackOff), su **IP interna** y **en qué nodo** está. Es el panel principal para saber si tu app ya arrancó bien.

```
ubuntu@ip-172-31-20-226:~$ sudo k3s kubectl -n cinema get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE                                NOMINATED NODE   READINESS GATES
backend-59f89b9f59-5rwg4            1/1     Running   1 (4h58m ago)  2d2h  10.42.0.26      ip-172-31-20-226                   <none>            <none>
backend-77445768bc-nsfkv            0/1     Init:0/2   1           2d2h  10.42.0.22      ip-172-31-20-226                   <none>            <none>
backend-84795d76bd-7z6dx            1/1     Running   1 (4h58m ago)  2d7h  10.42.0.25      ip-172-31-20-226                   <none>            <none>
frontend-85df4b5466-674k5          1/1     Running   15 (4h58m ago)  2d2h  10.42.0.27      ip-172-31-20-226                   <none>            <none>
frontend-85df4b5466-gwsjw          1/1     Running   13 (4h58m ago)  2d2h  10.42.0.23      ip-172-31-20-226                   <none>            <none>
postgres-0                          1/1     Running   1 (4h57m ago)  2d1h  10.42.2.46      ip-172-31-24-143                   <none>            <none>
```

- Lista los **Services** (IPs/puertos **dentro** del clúster). Verás backend, frontend, postgres con su **ClusterIP**. El frontend llama al backend usando estos nombres.

```
ubuntu@ip-172-31-20-226:~/EPCC_CLOUD_2025/LABS/LAB_K8S/k8s$ sudo k3s kubectl -n cinema get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
backend      ClusterIP   10.43.32.140  <none>        8000/TCP   2d9h
frontend     ClusterIP   10.43.255.156 <none>        8080/TCP   2d9h
postgres     ClusterIP   10.43.17.114  <none>        5432/TCP   2d9h
```

- Muestra las **rutas externas** (host/path) configuradas en Traefik y a qué **Service** apuntan. Aquí confirmas cómo acceder desde el navegador externo.

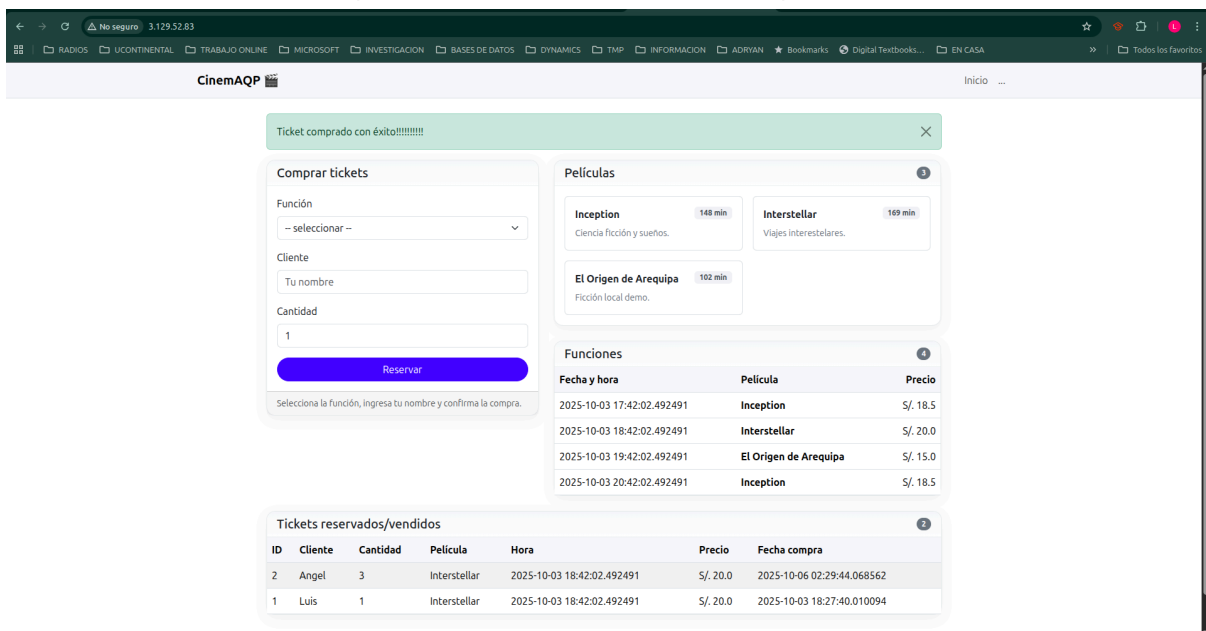
```
ubuntu@ip-172-31-20-226:~/EPCC_CLOUD_2025/LABS/LAB_K8S/k8s$ sudo k3s kubectl -n cinema get ingress
NAME          CLASS  HOSTS          ADDRESS                                                                 PORTS   AGE
cinema-ingress traefik *            172.31.20.226,172.31.24.143,172.31.29.109,172.31.37.111 80      2d9h
```

## G. Visualización de la app:

Para poder ver el correcto funcionamiento de la app desde la instancia de AWS se utiliza la siguiente instrucción:

```
# Frontend (servirá HTML):
curl -i http://<IP_PUBLICA_DEL_MASTER>/

# Backend (asumiendo expone /api o /health):
curl -i http://<IP_PUBLICA_DEL_MASTER>/api
```



## 7. CASOS CON K8S.

### 7.1. Agregar un nuevo nodo.



### ANTES (3 nodos):

```
nifla@VicNif: ~
nifla@VicNif: ~
nifla@VicNif: ~
ubuntu@ip-172-31-20-226:~/EPCC_CLOUD_2025/LABS/LAB DOCKERS$ sudo k3s kubectl get nodes -o wide
NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE KERNEL-VER
ip-172-31-20-226 Ready control-plane,master 2d14h v1.33.5+k3s1 172.31.20.226 <none> Ubuntu 24.04.3 LTS 6.14.0-101
ip-172-31-24-143 Ready <none> 2d14h v1.33.5+k3s1 172.31.24.143 <none> Ubuntu 24.04.3 LTS 6.14.0-101
ip-172-31-37-111 Ready <none> 2d14h v1.33.5+k3s1 172.31.37.111 <none> Ubuntu 24.04.3 LTS 6.14.0-101
ubuntu@ip-172-31-20-226:~/EPCC_CLOUD_2025/LABS/LAB DOCKERS$ sudo k3s kubectl -n cinema get pods -o wide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
backend-59f89b9f59-5rwg4 1/1 Running 1 (13m ago) 45h 10.42.0.26 ip-172-31-20-226 <none> <none>
backend-77445768bc-nsfkv 0/1 Init:0/2 1 45h 10.42.0.22 ip-172-31-20-226 <none> <none>
backend-84795d76bd-7z6dx 1/1 Running 1 (13m ago) 2d2h 10.42.0.25 ip-172-31-20-226 <none> <none>
frontend-85df4b5466-674k5 1/1 Running 15 (13m ago) 45h 10.42.0.27 ip-172-31-20-226 <none> <none>
frontend-85df4b5466-gwsjw 1/1 Running 13 (13m ago) 45h 10.42.0.23 ip-172-31-20-226 <none> <none>
postgres-0 1/1 Running 1 (13m ago) 44h 10.42.2.46 ip-172-31-24-143 <none> <none>
```

- **Agregar un nuevo nodo solo aumenta la capacidad del clúster.**
- **NO** se crean “más réplicas” automáticamente.
- El **scheduler** usará ese nodo **para pods nuevos** (o para reemplazar pods que estaban Pending), **pero no migra** pods que ya están corriendo solo porque llegó un nodo nuevo.
- **DaemonSets** sí crean **1 pod por nodo** automáticamente (ej.: agentes de logging), pero **Deployments/StatefulSets no**.

### ¿Qué cambia al sumar un nodo?

1. **Capacidad extra** (CPU/RAM/red) disponible para programar **nuevos pods**.
2. **Balance futuro**: cualquier **nuevo** pod que se cree (p.ej., al escalar réplicas o hacer un rollout) puede caer en el nodo nuevo.
3. **No rebalancea lo que ya corre**: Kubernetes por defecto **no mueve** pods en ejecución al nodo nuevo. (Si quisieras eso, existe el **Descheduler** como componente opcional para “reordenar” pods.)

### ¿Entonces el “master hace nuevas réplicas”?

**No.** El control plane (master) solo **mantiene el estado deseado**:

- Si tu Deployment pide **2 réplicas**, seguirás con 2.
- Si una se cae y hay capacidad, el scheduler **creará la de reemplazo** en el nodo disponible.
- Para tener **más réplicas**, las pides uno (o el **HPA** si se configura).

### DESPUES (4 nodos):

```
nifla@VicNif: ~
nifla@VicNif: ~
nifla@VicNif: ~
ubuntu@ip-172-31-20-226:~$ sudo k3s kubectl get nodes -o wide
NAME STATUS ROLES AGE VERSION INTERNAL-IP EXTERNAL-IP OS-IMAGE KERNEL-VER
ip-172-31-20-226 Ready control-plane,master 2d19h v1.33.5+k3s1 172.31.20.226 <none> Ubuntu 24.04.3 LTS 6.14
ip-172-31-24-143 Ready <none> 2d19h v1.33.5+k3s1 172.31.24.143 <none> Ubuntu 24.04.3 LTS 6.14
ip-172-31-29-109 Ready <none> 61s v1.33.5+k3s1 172.31.29.109 <none> Ubuntu 24.04.3 LTS 6.14
ip-172-31-37-111 Ready <none> 2d19h v1.33.5+k3s1 172.31.37.111 <none> Ubuntu 24.04.3 LTS 6.14
ubuntu@ip-172-31-20-226:~$ sudo k3s kubectl -n cinema get pods -o wide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS
backend-59f89b9f59-5rwg4 1/1 Running 1 (4h58m ago) 2d2h 10.42.0.26 ip-172-31-20-226 <none> <none>
backend-77445768bc-nsfkv 0/1 Init:0/2 1 2d2h 10.42.0.22 ip-172-31-20-226 <none> <none>
backend-84795d76bd-7z6dx 1/1 Running 1 (4h58m ago) 2d7h 10.42.0.25 ip-172-31-20-226 <none> <none>
frontend-85df4b5466-674k5 1/1 Running 15 (4h58m ago) 2d2h 10.42.0.27 ip-172-31-20-226 <none> <none>
frontend-85df4b5466-gwsjw 1/1 Running 13 (4h58m ago) 2d2h 10.42.0.23 ip-172-31-20-226 <none> <none>
postgres-0 1/1 Running 1 (4h57m ago) 2d1h 10.42.2.46 ip-172-31-24-143 <none> <none>
```

## 7.2. Stop del Nodo Master.

k3s de 1 master :

- Lo que **sí** sigue funcionando:
  - Los **Pods que ya están corriendo** en los workers **siguen** atendiendo el tráfico.
  - Los **Services** y el **routing** de red que ya existe siguen funcionando.
- Lo que **NO** se puede hacer mientras el máster está caído:
  - **Crear/actualizar** recursos (no hay API).
  - **Reprogramar** Pods caídos (si alguno muere, **no** se recrea hasta que el master vuelva).
  - **Escalado automático**, **rollouts**, cambios de config, etc.
- **Acceso externo:**
  - Si **Traefik/LoadBalancer** estaba **solo en el master** y el master muere, **pierdes la entrada externa** (aunque los Pods sigan vivos por dentro).
  - Si tienes Traefik accesible a través de otros nodos o detrás de un LB externo (NLB/ELB), se puede seguir entrando.

Es decir, con **1 master**, la app **puede seguir respondiendo** un buen rato si nada cambia ni se cae; **pero** se pierde la capacidad de “autoreparación” y de aplicar cambios hasta que el master regrese.

## 7.3. Stop de Nodo Worker.

**Pods stateless** (frontend/backend): el **Deployment/ReplicaSet** detecta pérdida y el **scheduler** los **reprograma** en los nodos vivos; el **Service** deja de enviar tráfico a los pods caídos.

**Postgres (1 réplica + PVC local)**: la app queda “en pie” mientras ese nodo viva. Si ese nodo muere definitivamente, perderás acceso a ese volumen local. Para producción en EC2:

```
ubuntu@ip-172-31-20-226:~$ sudo k3s kubectl -n cinema get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE
backend-59f89b9f59-5rwg4            1/1     Running   1 (4h58m ago)   2d2h   10.42.0.26      ip-172-31-20-226
backend-77445768bc-nsfkv            0/1     Init:0/2   1              2d2h   10.42.0.22      ip-172-31-20-226
backend-84795d76bd-7z6dx            1/1     Running   1 (4h58m ago)   2d7h   10.42.0.25      ip-172-31-20-226
frontend-85df4b5466-674k5          1/1     Running   15 (4h58m ago)  2d2h   10.42.0.27      ip-172-31-20-226
frontend-85df4b5466-gwsjw          1/1     Running   13 (4h58m ago)  2d2h   10.42.0.23      ip-172-31-20-226
postgres-0                          1/1     Running   1 (4h57m ago)   2d1h   10.42.2.46      ip-172-31-24-143
```

Al quitar 1 de los 3 nodos, los servicios de ese nodo caído pasa a ejecutarse en el master, y el nodo worker 1 se quedó, en este caso, con el servicio de la BD.

#### 7.4. Stop de todo el Cluster.

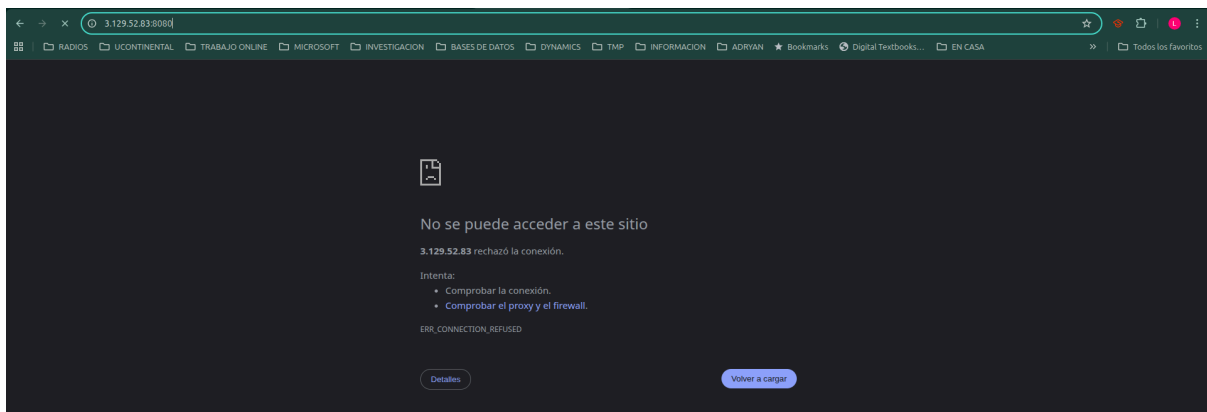
**Instancias (4)** Información

Buscar Instancia por atributo o etiqueta (case-sensitive) To

<input type="checkbox"/>	Name	ID de la instancia	Estado de la i...	Tipo de inst...
<input type="checkbox"/>	K8s_master	i-0ac3f4359ffd25237	Detenida	m7i-flex.large
<input type="checkbox"/>	K8s_node02	i-0386fe44cfa0550c1	Detenida	m7i-flex.large
<input type="checkbox"/>	K8s_node04	i-00194fb62397b619b	Detenida	m7i-flex.large
<input type="checkbox"/>	K8s_node03	i-003b83d549079a78c	Detenida	c7i-flex.large

Al detener las instancias EC2 **no destruye** el clúster k3s; lo **apaga**. Al volver a **iniciar** las instancias, el clúster **debería volver solo** si:

- El master (k3s **server**) arranca su servicio de systemd.
- Los workers (k3s **agent**) pueden volver a **conectarse** al master (misma IP/puerto 6443).
- Los discos (EBS) siguen presentes (no terminó o eliminó las instancias ni borraste volúmenes).
- No se depende de una **IP pública que cambió** para el ingreso (Traefik).



## 8. MICROSERVICIOS CON K8S.

### 8.1. Arquitectura de Microservicios (FastAPI)

Para adaptar a microservicios el proyecto del backend se separa en servicios pequeños, independientes y desplegables por separado, comunicándose vía HTTP.

Esto para desacoplar el backend monolítico en:

- `catalog-service` → `/movies`
- `showtime-service` → `/screenings`
- `booking-service` → `/tickets` (GET/POST)

Cómo se implementó

- Cada servicio con `main.py`, `requirements.txt`, `Dockerfile`.
- Variables de entorno para DB: `DB_HOST`/`PORT`/`USER`/`PASSWORD`/`NAME`.
- Deploys y Services en el chart Helm: `templates/catalog.yaml`, `showtime.yaml`, `booking.yaml`.

## 8.2. Deployments & Services (Kubernetes)

### Concepto

- **Deployment:** mantiene el número de réplicas de Pods y hace rolling updates.
- **Service:** IP/DNS estable para acceder a los Pods (descubrimiento interno).

### Uso:

Para correr réplicas de cada microservicio/frontend y exponerlos internamente con DNS tipo `catalog.cinema-ms.svc`.

Cómo se implementó

- En cada `templates/*service*.yaml`:
  - Deployment con `replicas`, `resources`, `readinessProbe`, `env`.
  - Service en puerto 8000 (microservicios) / 8080 (frontend).

## 8.3. Tecnologías Usadas:

### 8.3.1. Helm (Chart)

### Concepto

"Package manager" de K8s: empaqueta manifiestos como plantillas con variables (values.yaml), permite instalar/actualizar/rollback.

### Para qué se usó

Desplegar **todo el stack** de microservicios de forma repetible y parametrizable en un **namespace nuevo**.

### Cómo se implementó

- con Chart.yaml, values.yaml y templates/\*.yaml.

#### Flujo:

```
# 1) Edita values.yaml (tu Docker Hub, tag, host Ingress, namespace)
# 2) Instala/actualiza
helm upgrade --install cinema-ms . -n cinema-ms -f values.yaml
```

- Verificación:

```
sudo k3s kubectl -n cinema-ms get pods,svc,ingress -o wide
```

## 8.3.1. Envoy (API Gateway)

### Concepto

Proxy L7 que enruta y balancea tráfico HTTP/REST, centraliza rutas y oculta la topología interna.

### Para qué se usó

Exponer un solo endpoint /api para el frontend y derivar internamente:  
/api/movies → catalog, /api/screenings → showtime, /api/tickets  
→ booking.

### Cómo se implementó

- templates/api-gateway-configmap.yaml: envoy.yaml con las rutas y clusters DNS.
- templates/api-gateway.yaml: Deployment + Service (api-gateway:8080).
- En templates/frontend.yaml,  
BACKEND\_BASE\_URL="<http://api-gateway:8080/api>".

---

## Resultados

- Listo/implementado: Kubernetes (k3s), Docker+Hub, microservicios FastAPI, Helm, Envoy (API Gateway), Ingress (Traefik), Postgres StatefulSet con Secret/ConfigMap/PVC, HPA, anti-affinity, namespace separado.
- Sugerencia siguiente paso: Prometheus (del listado) para métricas; opcional Grafana para dashboards; RDS/Patroni para HA de DB.