



# API & UI

## Contexto

En esta parte del desarrollo nos dedicamos a desarrollar una API CRUD con FAST API usando python. El objetivo principal de esta API es un sistema de control de stock para negocios, con capacidad de crear y contener usuarios. También, el agregado de los productos en stock, con sus características, entre ellas, nombre, código, precio, cantidad, descripción e imagen ilustrativa, la cual es subida a través de la API.

Además en esta instancia se tomó la decisión de la base de datos a usar, considerando el entorno de producción, al ser Kubernetes terminamos optando por MySQL, por facilidad de desplegar y para interactuar con la base de datos se utilizó el ORM SQLAlchemy.

Por último, desarrolló una interfaz de usuario (UI) basada en NODE JS, haciendo uso del framework Next JS, el cual permite el uso de componentes de React para generar aplicaciones multi página. A través de esta interfaz gráfica, el usuario será capaz de:

- Observar todos los productos registrados en la base de datos y filtrar por código o nombre de producto a través de una barra de búsqueda.
- En caso de estar registrado como operador del sistema, se podrá acceder a un panel de administración.
- A través del panel de administración, será posible agregar, editar o borrar productos. Al momento de editar o agregar un producto se podrá subir una imagen ilustrativa del mismo.
- No se posee una interfaz de registro, estos se deberán hacer via API con ayuda de algun desarrollador.
- Cabe destacar que cada usuario tendra productos asociados a el, no podra ver ni operar sobre productos de otros usuarios en el panel de administracion.

Para la etapa de desarrollo local, se utilizó Docker como herramienta de contenerización, que permite luego subir imágenes al Docker Hub. A través de la herramienta Docker Compose, la cual permitió simular un ambiente parecido al de producción.

# Como generar el ambiente de desarrollo local

Como primer paso, se debe tener instalado poetry y nodejs.

Primero, se deben instalar las dependencias de python con poetry.

```
poetry install
```

Luego se debe ingresar a la carpeta de la ui e instalar las dependencias de nodejs

```
cd ./ui  
npm install
```

Para ejecutar la aplicacion hace falta poseer docker instalado en el equipo, ademas se debe crear una copia de `.env.sample` y renombrarla como `.env` para que docker sea capaz de cargar las configuraciones.

Una vez realizado esto, para ejecutar el proyeco, se debe usar:

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

Esto levantara la base de datos (MySQL), la API y la (UI).

El tag `--build` fuerza a que al inicio del despliegue se re cree la imagen de la API y UI. No seria necesario en un simple reinicio.

Para parar la infraestructura se debe usar:

```
````bash  
docker compose down
```

En caso de querer correr en modo desarrollo, se debe iniciar el deploy y luego utilizar:

```
docker compose down api  
# 0  
docker compose down ui
```

El comando a seleccionar dependera de en que entorno se deba trabajar, dado que el primero da de baja el contenedor de API y el segundo el de la UI.

Para correr la API en modo desarrollo se puede hacer uso de los sugientes comandos:

```
# Usando poetry  
poetry run uvicorn app.main:app --reload
```

```
# o si la terminal ya se encuentra en el entorno virtual
uvicorn app.main:app --reload
```

Y en el caso de la UI dentro de la carpeta ui:

```
npm run dev
```

## Estructura

El proyecto, este se organiza internamente por carpetas:

La carpeta `app` contiene todo lo referente a la API, y se compone de las subcarpetas:

- `api` : Contiene toda la logica correspondiente a las rutas de la API (Definicion de endpoints).
- `core` : Contiene utilidades generales al proyecto de la API, como configuraciones globales.
- `crud` : Contiene las funciones CRUD (Create, Read, Update, Delete) las cuales interactuan con la base de datos.
- `db` : Contiene utilidades referidas a la conexion con la base de datos.
- `models` : Contiene las clases constructoras de objetos utilizadas en el proyecto. Se diferencian dos tipos de clases, la primera clases de SQLAlchemy y clases de Pydantic. Las primeras son las encargadas de generar las tablas en la base de datos y gestionar la recuperacion e insercion de datos y las segundas son utilizadas como respuestas de los endpoints.

La carpeta `ui` contiene todo lo referente a la UI, esta se compone de las subcarpetas:

- `components` : Contiene componentes de react utilizados en las diferentes paginas o en diferentes componentes.
- `pages` : Contiene la estructura de paginas y ruteo del servidor web. Es decir, el `index.ts` representa al `index.html` de `/`, mientras que el `/admin/index.ts` representa el `/admin/index.html`.
- Dentro de `pages` a su vez notamos una carpeta `api`, la cual funciona de intermediario entre la API de Fast Api y el servidor web de nextjs. Mediante esta, podemos evitar exponer directamente la API a los clientes.
- `public` : Contiene los recursos publicos de la pagina web.
- `utils` : Contiene definicion de interfaces (Parecidas a las clases pero solo denotan una estructura de datos) y configuraciones como la ruta de la API.

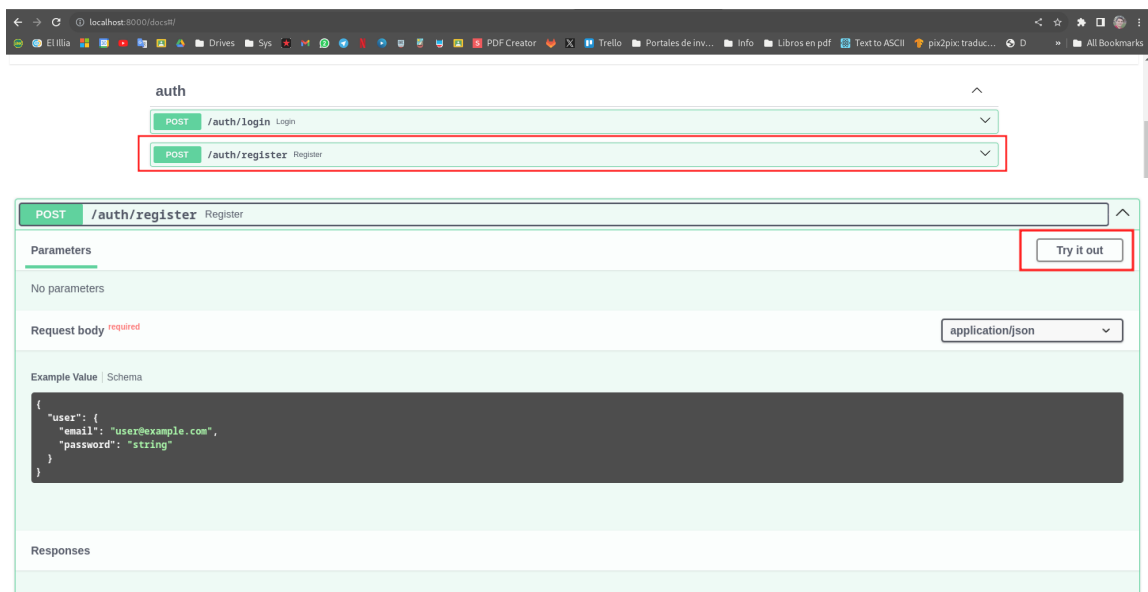
## Aclaracion sobre NextJS

Next.js es un framework basado en React que facilita la renderización del código en el servidor, entregando al cliente un conjunto de archivos HTML, CSS y JavaScript con la lógica de acciones correspondiente. Cada página React, definida como exportación por defecto, debe ser pre-renderizada y enviada al cliente. Además, el uso de `getServerSideProps` permite ejecutar código antes de la renderización, posibilitando la limitación de acceso a páginas basándose en la presencia de cookies. Por otro lado, las rutas de API ubicadas en `ui/pages/api` son procesadas en el servidor.

## Ejemplos y pruebas

Previo a la integracion con la UI, una vez todos los endpoints estaban desarrollados, se procedio a probar cada uno de estos endpoints. Para esto se utilizo el Swagger que viene incorporado en Fast API que se encuentra en `http://localhost:8000/docs` :

Para todos los endpoints, a la hora probarlos, deberemos seguir estos dos pasos, desplegar dicho endpoint y seleccionar el boton `Try it out` .



En caso de querer crear un usuario:

POST /auth/register Register

Parameters

Cancel

Reset

No parameters

Request body required

application/json

```
{
  "user": {
    "email": "usuario@email.com",
    "password": "12345678"
  }
}
```

Execute

POST /auth/register Register

Parameters

Cancel

Reset

No parameters

Request body required

application/json

```
{
  "user": {
    "email": "usuario@email.com",
    "password": "12345678"
  }
}
```

Execute

Clear

Responses

Curl

```
curl -X 'POST' \
  http://localhost:8080/auth/register' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "user": {
      "email": "usuario@email.com",
      "password": "12345678"
    }
  }'
```

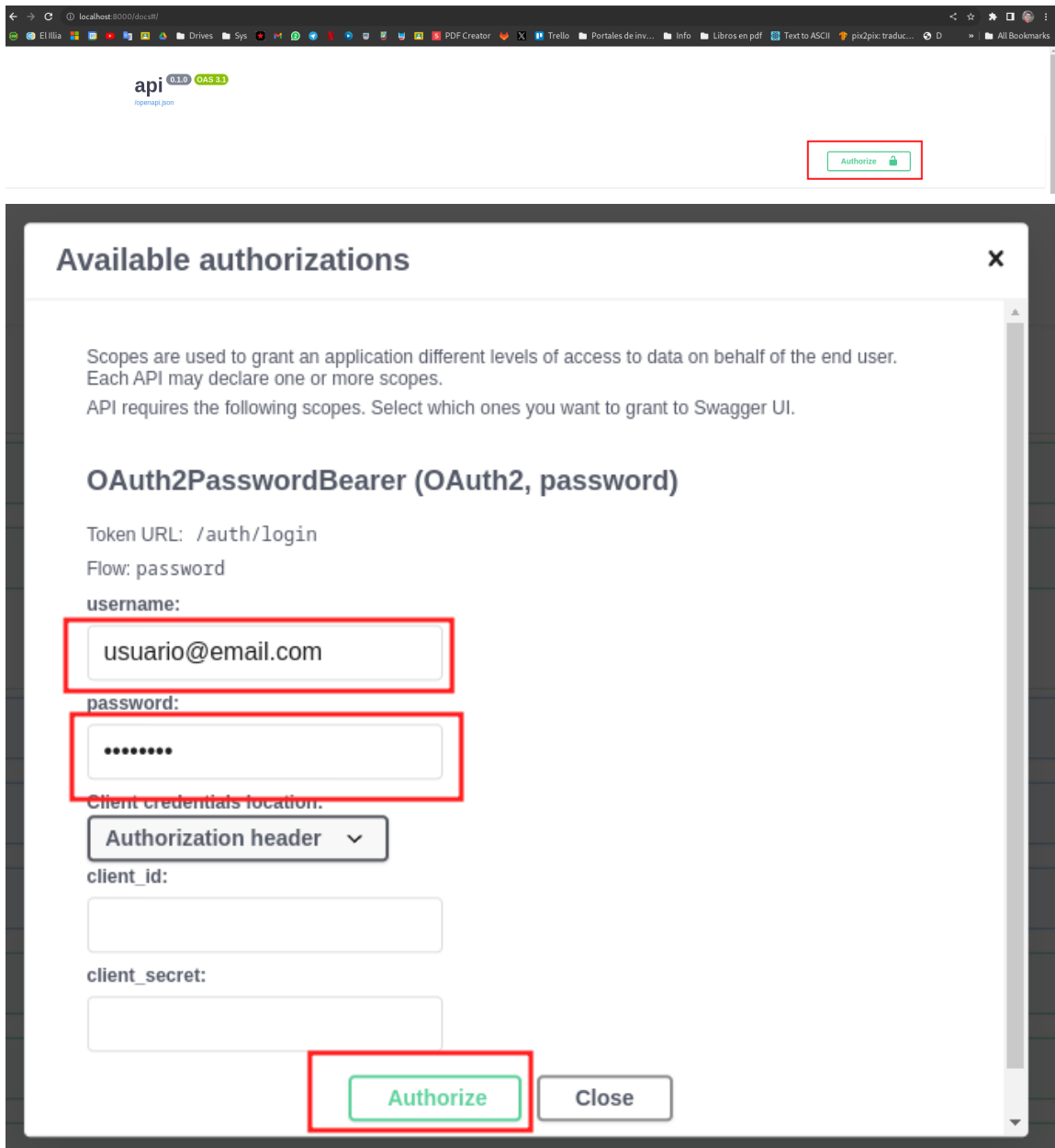
Request URL

```
http://localhost:8080/auth/register
```

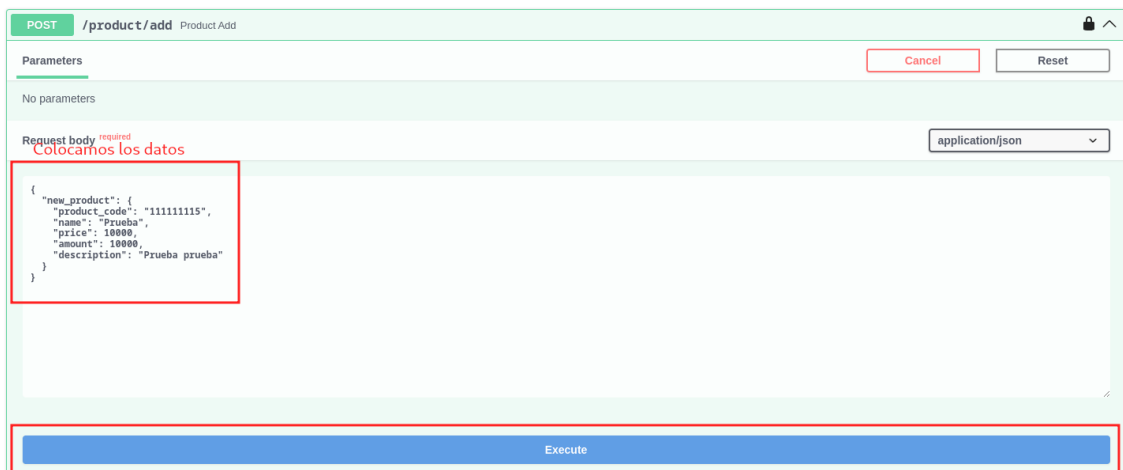
Server response

| Code | Details   |
|------|---|
| 201  | <div>Response body<pre>{   "user": {     "email": "usuario@email.com",     "id": 1,     "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFnbnCI6InVzdwFyYW9AZW1haWV29tIiwiaWQiOiJ1bnV4cCI6MTcwMzZzNDM0NH0.3m0Vjp6wP8oRpkfsvDXDlVbk2dJKN3RShnu0LaixQ"   } }</pre></div> <div><div>Download</div></div> |

Para iniciar sesion con dicho usuario y mantener las credenciales guardadas para el resto de endpoints

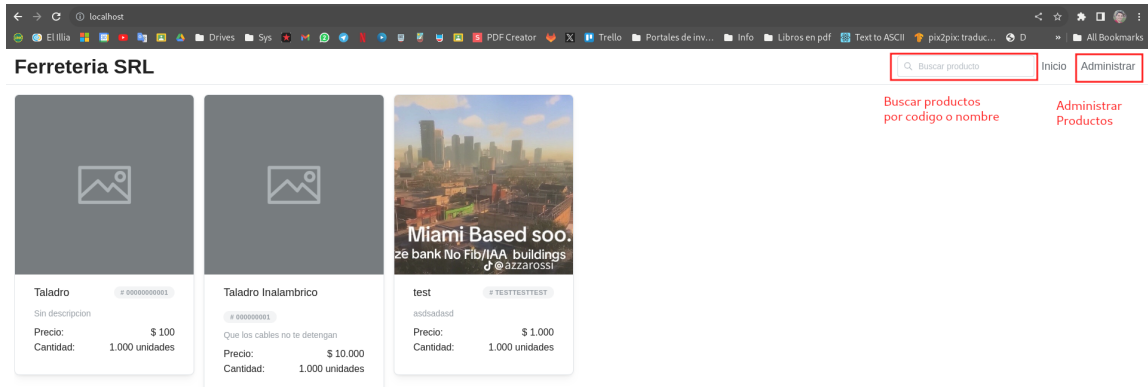


Para crear un producto:

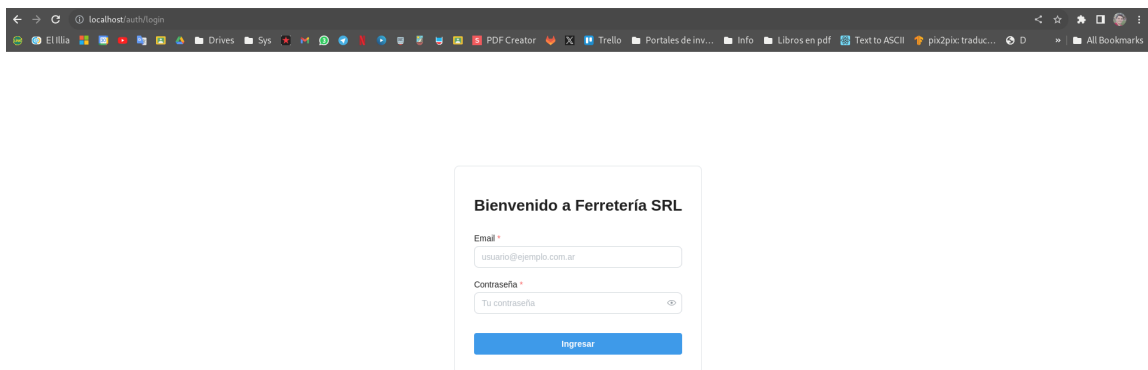


Para obtener todos los productos debemos usar solo el boton `Try it out` sobre el endpoint `/products/get/all`.

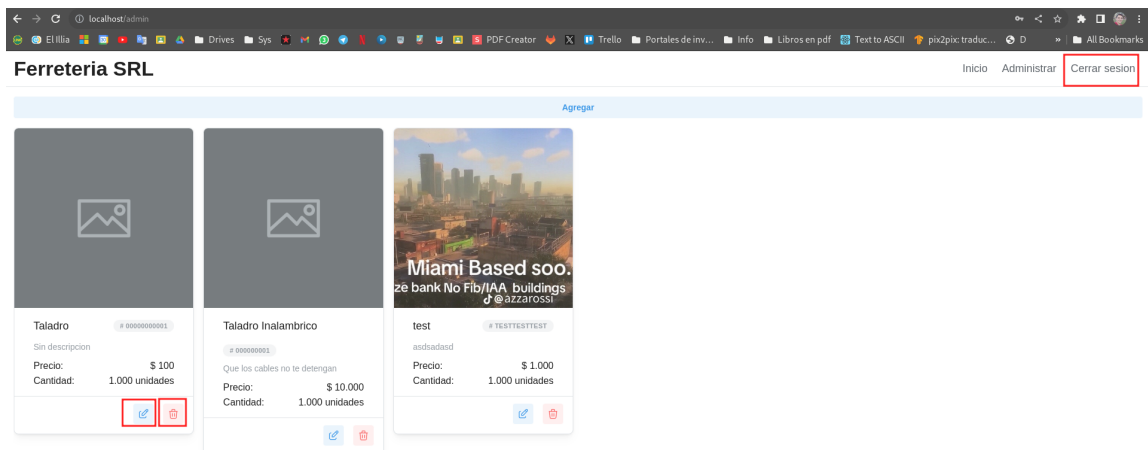
Una vez comprobada la funcionalidad de dichos endpoints, se procedio a desarrollar y probar la UI:



Si seleccionamos administrar, al no estar logeados, nos redireccionara a la pagina de login:



Una vez logeados podemos ir a la pagina admin donde veremos nuestros productos, editarlos y borrarlos, ademas de agregarlos.





Para agregar un producto:


localhost/admin

Elilia Drives Sys PDF Creator Trello Portales de inv... Info Libros en pdf Text to ASCII pix/pix: traduc... All Bookmarks

## Ferreteria SRL

Inicio Administrar Cerrar sesion

Agregar




**Taladro**

Sin descripcion

Precio: \$ 100

Cantidad: 1.000 unidades




**Taladro Inalambrico**

Que los cables no te detengan

Precio: \$ 10.000

Cantidad: 1.000 unidades



**test**

asdsadasd

Precio: \$ 1.000

Cantidad: 1.000 unidades

### Nuevo producto

**Nombre \***

Prueba

**Codigo \***

11111115555

**Precio \***

10000


**Cantidad \***

10000

**Descripcion**

**Remplezar imagen**

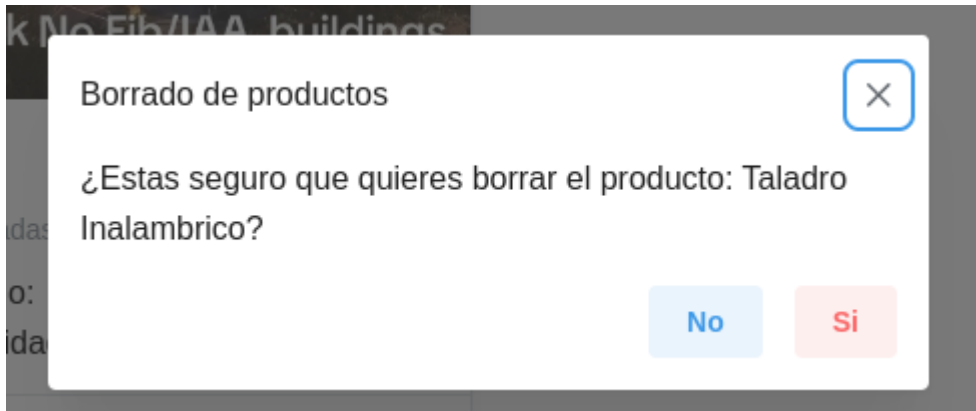
**Quitar imagen**



**Crear**

Para editar, la interfaz es la misma, solo que se accede del boton de editar.

Por ultimo para borrar, precionamos el icono de la papelera y nos saldra una confirmacion:



## Cómo levantar un Clúster y desplegar nuestra aplicación

### Vagrant

Vagrant es una herramienta que podés usar para crear y gestionar entornos de desarrollo virtualizados de manera fácil y reproducible. Su uso típico es facilitar la creación de máquinas virtuales con configuraciones específicas para el desarrollo de proyectos.

Para comenzar un proyecto de Vagrant en el directorio /vagrant, el usuario puede seguir estos pasos:

#### Instalación de Vagrant

Antes que nada, necesitás instalar Vagrant en tu máquina. Esto se puede hacer descargando el instalador desde el sitio oficial y siguiendo las [instrucciones](#)

#### Instalar el provider de VirtualBox

Descargar de la [página oficial](#).

**¡Importante!** Para poder configurar cierta red privada deberemos crear o modificar el archivo `/etc/vbox/networks.conf` añadiendo la red de la siguiente manera:

```
sudo su
echo "*" 0.0.0.0/0" > /etc/vbox/networks.conf
```

## Crear las claves públicas y privadas para las conexiones SSH

Creamos nuestra propia clave pública y privada con `ssh-keygen`, procuramos no poner passphrase para que no se la solicite a las VMs a la hora de iniciarlas.

```
# ~/.ssh/
> ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/aagustin/.ssh/id_rsa): vagrant_key
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in vagrant_key
Your public key has been saved in vagrant_key.pub
The key fingerprint is:
SHA256:K4v2o7EKOLfjCMLk7zVD4v234234c06peueU aagustin@hp-agustin
The key's randomart image is:
+---[RSA 4096]-----+
|
|
|
|
| . . . S . |
|= . 3 o .. o.. |
|*o.+.*.... +.++. |
|o=o.B4== . *++ |
|..*+*=++ .+o. E |
+---[SHA256]-----+
```

## Instalación de Ansible

Ejecutamos los siguientes comandos en nuestra máquina host:

```
sudo apt update
sudo apt install software-properties-common
sudo apt-add-repository ppa:ansible/ansible
sudo apt update
sudo apt install ansible
```

# Personalizar nuestros archivos de configuración y aprovisionar

## Configuraciones de perfil

Ingresamos al archivo `ansible/group_vars/all.yml` y elegimos la cantidad de nodos que deseemos, también la versión de Kubernetes. Además seleccionamos los recursos.

En nuestro caso se dejó de la siguiente manera:

```
---
settings:
  env: 'test_api'
  users:
    test_api:
      prod_test: false

  environment: ""

  user_dir_path: /home/aagustin
  node_home_dir: /home/vagrant

  shared_folders:
    - host_path: ./shared_folder
      vm_path: /home/vagrant

  cluster_name: Kubernetes Cluster

  ssh:
    user: "vagrant"
    password: "vagrant"
    private_key_path: /home/aagustin/.ssh/vagrant_key # Crearse una para que funcione
    public_key_path: /home/aagustin/.ssh/vagrant_key.pub # Crearse una para que
funcione

  nodes:
    control:
      cpu: 2
      memory: 4096
    workers:
      count: 2
      cpu: 2
      memory: 4096

  network:
    control_ip: 192.168.100.171
    dns_servers:
      - 8.8.8.8
      - 1.1.1.1
    pod_cidr: 172.16.1.0/16
    service_cidr: 172.17.1.0/18
```

```
software:  
  box: bento/ubuntu-22.04  
  calico: 3.25.0  
  kubernetes: 1.26.1-00  
  os: xUbuntu_22.04  
  
  flannel: 0.23.0  
  contained: v1.5
```

## Levantar máquinas virtuales de Vagrant

Levantarlas:

```
# en <project-dir>/k8s/  
vagrant up
```

Si deseamos eliminarlas:

```
# en <project-dir>/k8s/  
vagrant destroy
```

## Aprovisionamos con Ansible

Desde la máquina host, posicionados en `<project-dir>/k8s/` , ejecutaremos los siguientes comandos:

```
ansible -i ansible/inventory_local.yml -m ping all
```

Y nos aseguraremos de tener respuesta de todos los nodos creados.

Y luego aprovisionamos con:

```
ansible-playbook -vvv ansible/site.yml -i ansible/inventory_local.yml
```

## Acceso a los nodos mediante SSH

Para acceder al nodo master:

```
ssh -i ~/.ssh/vagrant_key vagrant@192.168.100.171
```

Si queremos entrar como `root` simplemente cambiamos el usuario a `root`.

# Minikube

Se detectaron problemas de DNS en el Cluster de Vagrant, estos son fácilmente verificables generando un pod de test [dns-utils](#) y aplicando el siguiente comando:

```
kubectrl exec -i -t dnsutils -- nslookup <nombre-servicio>.svc.default.cluster.local
```

Donde deberíamos obtener la IP del servicio o del pod al que estamos queriendo llegar con ese dominio. En el caso del Cluster de Vagrant no se obtenían las direcciones IP correspondientes, esto nos asegura que el `core-dns` no está funcionando como corresponde lo que indica una mala instalación de `kubelet`.

Resulta, que por experiencias previas sabíamos que esto podía ser así debido a que la configuración propuesta de Vagrant no permite los reinicios debido a que se corrompen las carpetas compartidas y al no poder reiniciar no se aplican como corresponde las configuraciones pertinentes al aprovisionamiento de Kubernetes.

Dicho esto, y entendiendo además que se complican las pruebas en local, se tomó la decisión de conservar la versión utilizada de Kubernetes (1.26.1) y hacer uso de Minikube, una herramienta opensource que mediante la creación de una máquina virtual nos permite disponer de un entorno sencillo de Kubernetes con la mayor parte de sus funcionalidades.

Para levantar el cluster necesario, ya teniendo Minikube instalado, aplicamos el siguiente comando:

```
minikube start --kubernetes-version='1.26.1' --memory='4096' --cpus='2' --disk-size='10GB'  
--vm=true --nodes=2 -p "test-1.26-2nodes"  
minikube addons enable metrics-server
```

Podemos borrar nuestro perfil creado haciendo:

```
minikube delete -p test-1.26-2nodes
```

Podemos levantar el Dashboard para una mejor visualización de lo que ocurre con:

```
minikube dashboard -p test-1.26-2nodes
```

## Levantar base de datos MySQL

### ¿Por qué MySQL y no MongoDB u alguna otra?

Se hizo la búsqueda de información pertinente para poder hacer una selección adecuada para los requerimientos y en la misma se encontró que entre las bases

de datos más utilizadas dentro de los entornos de Kubernetes son MySQL y PostgreSQL y que además, el uso de una base de datos en un entorno de Kubernetes no es una decisión que debamos tomar a la ligera, debido a que resulta muy complicado el mantenimiento y la misma configuración.

Entendiendo lo anterior, se optó por MySQL como base de datos para el proyecto y la misma se desplegará sobre Kubernetes en una configuración denominada "Statefulset", que permite desplegar nombres de pods y volúmenes persistentes.

### **¿Qué arquitectura elegimos y por qué?**

Se probaron 3 diferentes formas para levantar la base de datos en las cuales nos hemos topado con distitos inconvenientes por lo cuales hemos llegado a una decisión final.

Dentro de las pruebas se lograron diferenciar conceptos importantes:

#### **HEADLESS SERVICES VS. CLUSTERIP SERVICES VS. NODEPORT SERVICES**

Los Headless Services no asignan una dirección IP virtual estable, en lugar de eso, devuelven directamente las IP de los pods seleccionados, se usa cuando necesitas acceder a cada instancia individualmente, por ejemplo, en bases de datos distribuidas. Estos realizan "loadbalancing" a la hora de tener que distribuir el tráfico (no le pegan siempre al mismo pod).

Los ClusterIP Services asignan una IP virtual estable que representa el servicio y dirige el tráfico a los pods seleccionados. Son ideales para servicios internos en el clúster que solo necesitan ser accedidos internamente. Estos realizan balanceo de carga entre los pods seleccionados.

Los NodePort Services exponen el servicio en un puerto en cada nodo del clúster, permitiendo el acceso externo. Son útiles cuando necesitas acceder al servicio desde fuera del clúster, pero no es ideal para producción debido a posibles problemas de seguridad. Este tipo de servicio proporciona balanceo de carga entre los nodos, pero no entre los pods directamente.

#### **STATEFULSET**

Es un controlador que mantiene un conjunto de pods con identidades únicas y persistentes. Se utiliza para aplicaciones que requieren identidades persistentes, como bases de datos, donde cada pod tiene un estado y un nombre único. Garantiza que los pods se creen y se escalen de manera ordenada, y proporciona almacenamiento persistente para cada pod.

A diferencia de Deployments, que son más adecuados para aplicaciones sin estado, un StatefulSet es ideal cuando se necesita mantener un estado específico y único para cada instancia, como en el caso de bases de datos.

Supongamos que tenemos un StatefulSet para una base de datos MySQL y decidimos llamar al StatefulSet "mysql-db". La nomenclatura para los nombres de los pods y persistent volumes generados por el StatefulSet sería algo como:

- Pods generados por el StatefulSet:

```
mysql-db-0  
mysql-db-1  
mysql-db-2
```

- Persistent Volumes generados por el StatefulSet:

```
mysql-db-pv-0  
mysql-db-pv-1  
mysql-db-pv-2
```

Cada pod y cada volumen persistente tendría un nombre único basado en el nombre del StatefulSet y un índice que refleja su posición en la secuencia.

Ahora, si tuviéramos un Deployment común para una aplicación web llamado "web-app", los nombres de los pods generados serían más genéricos y probablemente se basarían en un identificador aleatorio o algún nombre de base, como:

- Pods generados por el Deployment:

```
web-app-74d8b8b5f9-abcde  
web-app-74d8b8b5f9-12345  
web-app-74d8b8b5f9-xyzab
```

- Persistent Volumes generados por el Deployment: No habría persistent volumes específicos asociados directamente con los pods generados por el Deployment, a menos que configures almacenamiento persistente externamente.

En resumen, la diferencia radica en la predictibilidad y consistencia de los nombres. Los StatefulSets generan nombres que reflejan la posición y el propósito específico del pod en la secuencia, mientras que los Deployments comunes pueden generar nombres más aleatorios o genéricos.

## REPLICAR LA BASE DE DATOS

Supongamos que gracias a los servicios configurados y nuestra API logramos escribir sobre la base de datos del Statefulset de MySQL, que para este ejemplo consta de 3 pods y por consiguiente 3 PV's.



Si hacemos intentos reiterados para escribir sobre la base de datos resultaría que, gracias al balanceo de cargas realizado por el servicio ClusterIP o Headless, escribiríamos algunos datos en una de las réplicas, otros en la siguiente y así sucesivamente, resultando que, cuando quisieramos leer, no obtendríamos todos los datos de una vez ya que cada réplica es una base de datos independientes y requerimos de una lógica extra para hacer la replicación de la información y así esta esté disponible en todas las replicas.

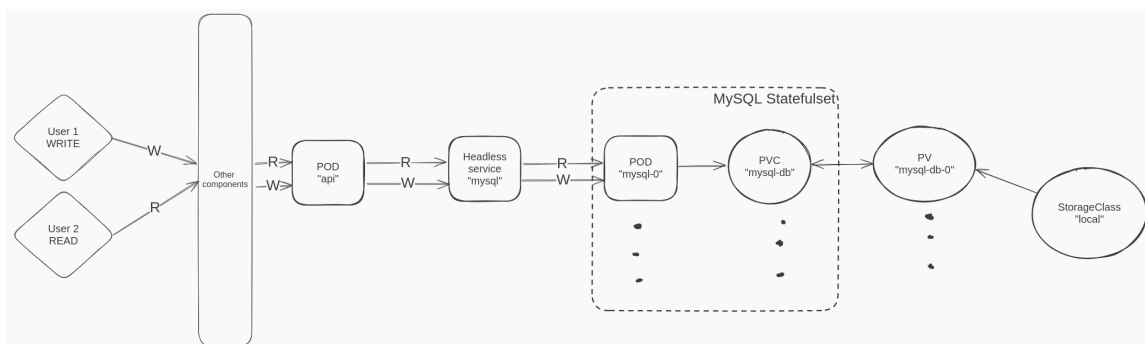
En el contexto de las DB, podemos configurar un conjunto de réplicas utilizando un controlador StatefulSet y configurar la replicación de cada tipo de DB dentro de los pods.

### ELECCIÓN TOMADA

Entendiendo lo anterior, creemos que un Headless service y una configuración Statefulset es la mayor elección para desplegar nuestra base de datos.

A la hora de la configuración sobre Minikube se encontró que la configuración de replicación de la DB que nos ofrece de ejemplo la documentación de Kubernetes no funciona como corresponde.

Debido a lo nombrado anteriormente es que se optó por una base de datos en una configuración Statefulset pero que contenga una única réplica así nos olvidamos del problema de la no-replicación.



### Manifiestos de Kubernetes de la DB

Aquí se brinda el manifiesto utilizado:

```
---
apiVersion: v1
kind: Service
metadata:
  name: mysql-headless
spec:
  clusterIP: None
  selector:
    app: db
  ports:
```

```

- name: tcp
  protocol: TCP
  port: 3306
---
# $ echo -n "root" | base64
# cm9vdA==
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  ROOT_PASSWORD: cm9vdA==
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: mysql-initdb-config
data:
  init.sql: |
    CREATE DATABASE IF NOT EXISTS api;
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql
spec:
  replicas: 1
  serviceName: mysql-headless
  selector:
    matchLabels:
      app: db
  template:
    metadata:
      labels:
        app: db
  spec:
    terminationGracePeriodSeconds: 10
    containers:
      - name: mysql
        image: mysql:5.7
        ports:
          - containerPort: 3306
        env:
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                key: ROOT_PASSWORD
                name: mysecret
        volumeMounts:
          - name: mysql-initdb
            mountPath: /docker-entrypoint-initdb.d
          - name: data
            mountPath: /var/lib/mysql
    volumes:
      - name: mysql-initdb
        configMap:

```

```

    name: mysql-initdb-config
volumeClaimTemplates:
- metadata:
    name: data
  spec:
    accessModes: [ "ReadWriteOnce" ]
    resources:
      requests:
        storage: 1Gi

```

Desplegar:

```
kubectl apply -f <nombre-manifiesto-db>.yaml
```

Eliminar:

```
kubectl delete -f <nombre-manifiesto-db>.yaml
```

La DB quedará accesible entonces vía el siguiente dominio:

```
mysql+pymysql://root:root@mysql-0.mysql-headless.default.svc.cluster.local:3306/api
```

## Backend - Desplegar la API

La API debe ser accesible por fuera con el uso de un servicio tipo NodePort y a su vez debe poder acceder a la DB mediante el uso de DNS, pegandole al pod correspondiente (o al servicio, da igual porque tenemos un único pod) de la DB.

Para ello se creó el siguiente manifiesto:

```

apiVersion: v1
kind: Service
metadata:
  name: api
spec:
  type: NodePort
  ports:
    - port: 8000
      targetPort: 8000
      nodePort: 31000
  selector:
    app: api
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api
labels:
  app: api
spec:
  replicas: 1

```

```

selector:
  matchLabels:
    app: api
template:
  metadata:
    labels:
      app: api
spec:
  containers:
    - name: api-backend-fastapi
      image: aagustinconti/fast-api-products:latest #DockerHub repo
      ports:
        - containerPort: 8000
      imagePullPolicy: Always
      env:
        - name: MYSQL_HOST
          value: mysql-0.mysql-headless.default.svc.cluster.local
        - name: MYSQL_USER
          value: root
        - name: MYSQL_PASSWORD
          value: root
        - name: MYSQL_DATABASE
          value: api
        - name: DATABASE_URI
          value: 'mysql+pymysql://root:root@mysql-0.mysql-headless.default.svc.cluster.local:3306/todo'

```

Desplegar:

```
kubectl apply -f <nombre-manifiesto-api>.yaml
```

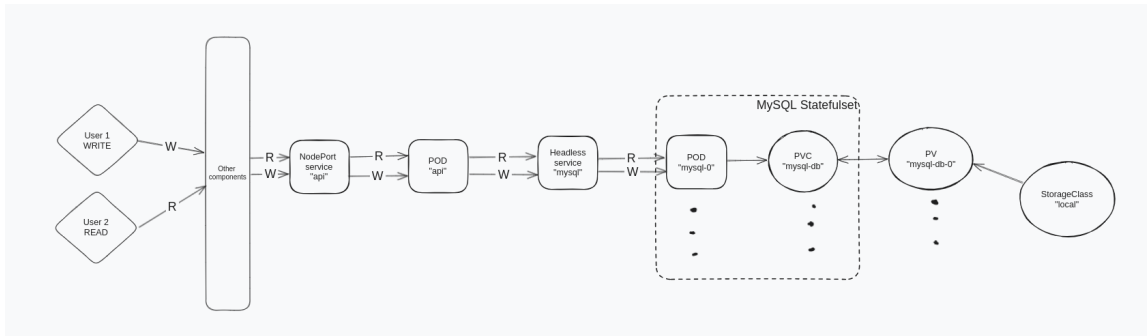
Eliminar:

```
kubectl delete -f <nombre-manifiesto-api>.yaml
```

Como dato a tener en cuenta, la API estará disponible en el puerto 31000 de la IP del nodo worker. Internamente el container tendrá expuesto el puerto 8000, el cual estará apuntado por el servicio tipo NodePort para poder exponerlo al 31000, justamente.

Otro dato importante es que se tuvo que dockerizar la API y subir la imagen al DockerHub para que Kubernetes la pueda descargar, como se ve, esta está disponible en `aagustinconti/fast-api-products:latest`.

La gráfica quedaría de la siguiente manera:



## Frontend - Desplegar la UI

La UI debe poder comunicarse con la API y a su vez debe ser accesible desde el puerto 80 de la IP del nodo así los usuarios pueden acceder a la misma. Para lo anterior se creó otro servicio tipo NodePort y a su vez se le pasaron los datos importantes para que se pueda conectar a la API, como el host: `mysql-0.mysql-headless.default.svc.cluster.local`, la clave, el usuario y la DB.

El manifiesto utilizado fue el siguiente:

```
apiVersion: v1
kind: Service
metadata:
  name: ui
spec:
  type: NodePort
  ports:
    - port: 80
      targetPort: 80
      nodePort: 30001
  selector:
    app: ui
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ui
  labels:
    app: ui
spec:
  replicas: 1
  selector:
    matchLabels:
      app: ui
  template:
    metadata:
      labels:
        app: ui
    spec:
      containers:
        - name: api-frontend-node
          image: aagustinconti/ui-node-products:latest # DockerHub repo
```

```

ports:
  - containerPort: 80
imagePullPolicy: Always
env:
  # <service-name>.<namespace>.svc.cluster.local:<service-port>
  - name: API_URL
    value: 'http://api:8000'
  - name: PORT
    value: '80'

```

Desplegar:

```
kubectl apply -f <nombre-manifiesto-ui>.yaml
```

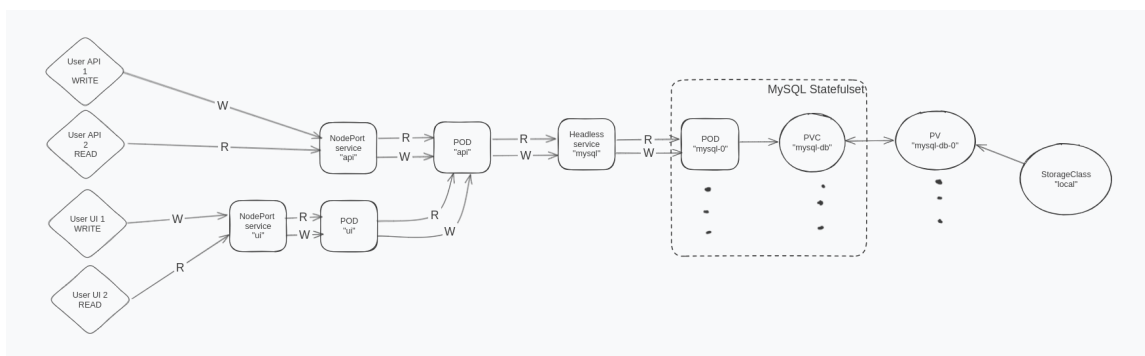
Eliminar:

```
kubectl delete -f <nombre-manifiesto-ui>.yaml
```

Nuevamente se utilizó Docker para crear la imagen y subirla al DockerHub para que sea accesible por Kubernetes a través de `aagustinconti/ui-node-products:latest`.

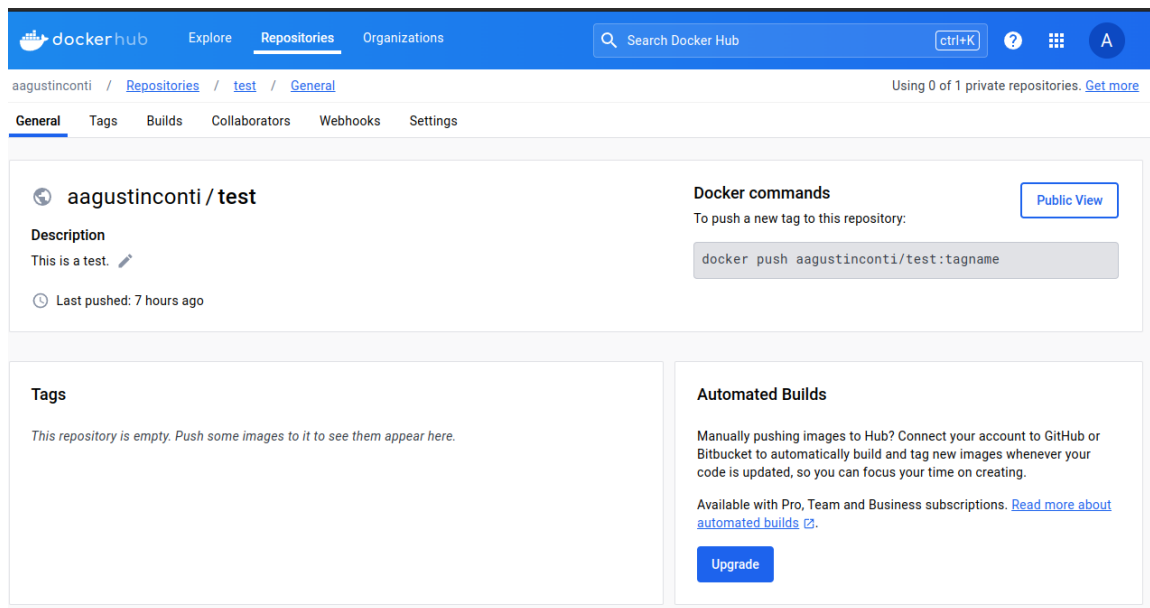
Por otro lado debemos observar algo clave, entre los valores de las variables de entorno que le suministramos, encontramos que encuentra perfectamente a la api mediante el servicio "api" en el puerto 8000 (utilizando la resolución de DNS).

El diagrama queda como sigue:



¿Como paso la Dockerfile a DockerHub?

1. Crearse una cuenta en [DockerHub](#)
2. Crear un repositorio público



### 1. Buildear la imagen en local

```
docker build -f ./app/Dockerfile .
```

### 1. Checkeamos la creación de la imagen

```
docker images
```

### 1. Aplicar el siguiente comando

```
docker tag <nombre-imagen> <dockerhub-user>/<nombre-imagen>  
docker push <dockerhub-user>/<nombre-repositorio-dockerhub>
```

## Monitoreo de los recursos utilizados

### Grafana y Prometheus

#### Instalación

#### 1. Descargar repositorio del operador:

```
git clone https://github.com/prometheus-operator/kube-prometheus.git  
cd kube-prometheus
```

#### 1. Instalar el operador:

```
kubectl apply --server-side -f manifests/setup
```

#### 1. Instalar todos los elementos:

```
kubectl apply -f manifests
```

1. Chekeamos que se hayan levantado todos los servicios, replicaset y pods:

```
kubectl get all -n monitoring
```

1. Editamos los servicios para poder exponerlos, de ClusterIP a NodePort a todos:

```
kubectl edit svc grafana -n monitoring
```

```
kubectl edit svc prometheus-k8s -n monitoring
```

```
kubectl edit svc alertmanager-main -n monitoring
```

1. Listamos los nodos, buscamos la IP del worker:

```
kubectl get nodes -o wide
```

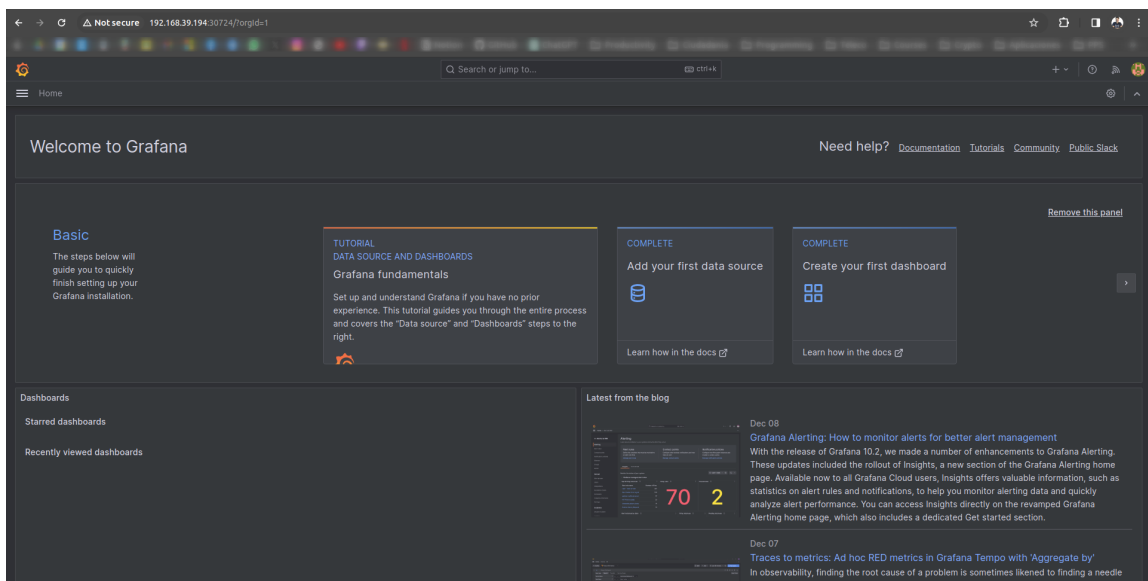
1. Listamos todos los servicios y buscamos el puerto al que se ha expuesto:

```
kubectl get svc -A
```

Checkeamos los puertos grafana , prometheus-k8s y alertmanager-main que tengan "NodePort".

1. Ingresamos a <ip-nodo-worker>:<puerto-svc>

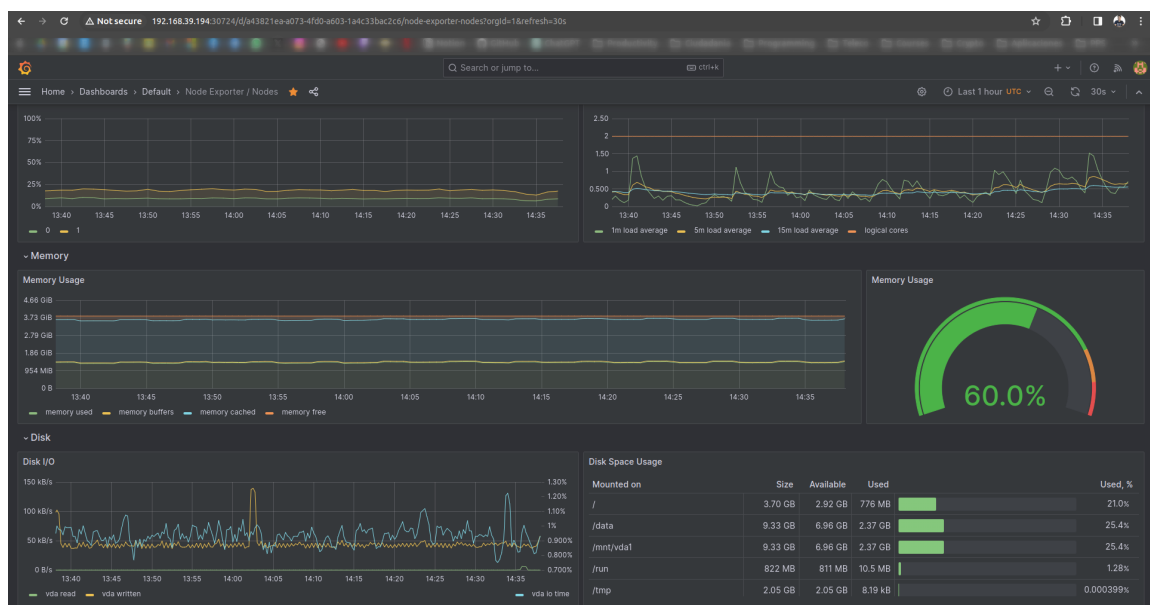
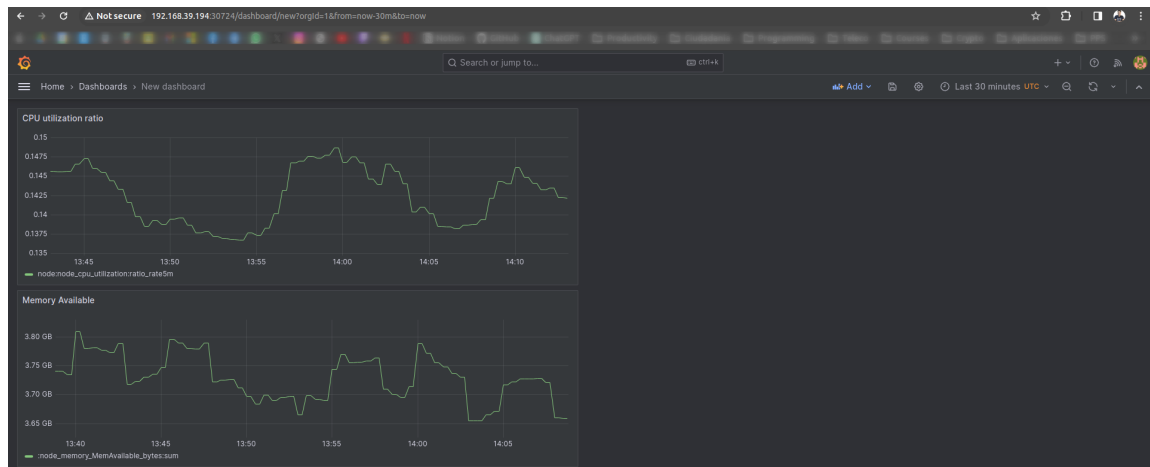
## Ejemplo grafana





user: admin , psw: admin

Añadimos de datasource a Prometheus y configuramos un dashboard con algunos datos, por ejemplo, memoria disponible:



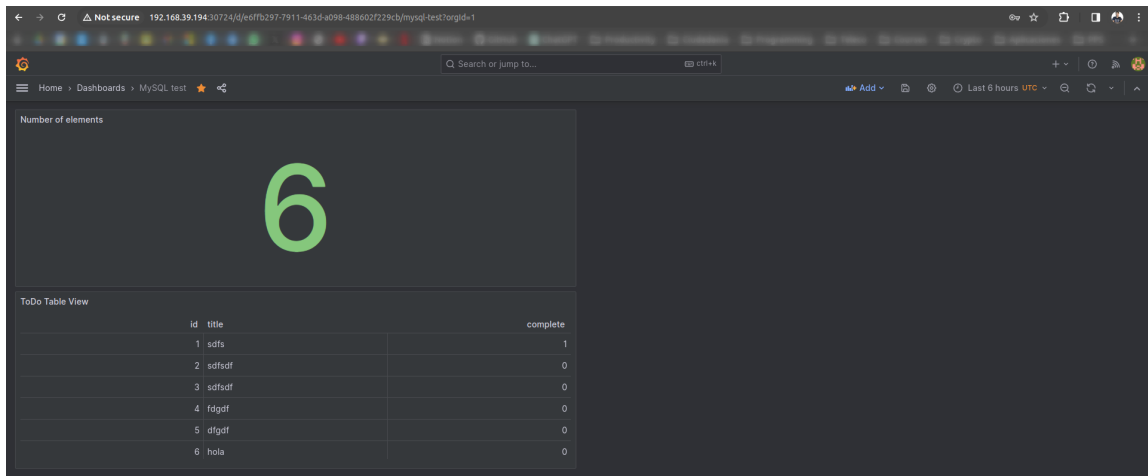
## Levantar la base de datos MySQL en Grafana

Vamos a Data Sources , luego Add Data Sources , luego elegimos MySQL y colocamos:

- Host URL: mysql-0.mysql-headless.default.svc.cluster.local:3306
- Database name: <nombre-db>
- Username: root
- Password: root

Hacemos click en Save and Test .

Luego podremos crear un dashboard donde traigamos como tabla, por ejemplo, los datos de la tabla:



## k6

### ¿Qué es k6?

k6 es una herramienta de código abierto destinada a realizar pruebas de carga y rendimiento en aplicaciones y servicios web. Utilizando JavaScript/ES6 para escribir scripts de prueba, k6 destaca por su sencilla sintaxis y soporte para protocolos como HTTP/1.1, HTTP/2 y WebSockets. Diseñado para ejecutarse en la nube y ser fácilmente integrado en flujos de trabajo de CI/CD, k6 proporciona informes detallados y métricas para evaluar el rendimiento de las aplicaciones bajo diversas condiciones de carga. Su flexibilidad y extensibilidad lo convierten en una herramienta valiosa para evaluar y mejorar la escalabilidad de sistemas web.

Entonces, k6 simplifica la realización de pruebas de rendimiento al ofrecer una solución basada en JavaScript con características como integración en la nube, generación de informes detallados y soporte para varios protocolos, proporcionando a los equipos de desarrollo la capacidad de evaluar y optimizar el rendimiento de sus aplicaciones de manera efectiva.

Gracias a ella pudimos plantear pruebas de stress locales hacia diferentes host y endpoints de nuestra api para poder verificar el cumplimiento de los requerimientos acordados con el cliente.

## K6 junto con Prometheus y Grafana

La visualización de datos a través de Grafana en conjunto con Prometheus ofrece una visión integral y en tiempo real del rendimiento del cluster que hospeda la aplicación. Al integrar estadísticas proporcionadas por k6 con los datos de consumo de recursos obtenidos mediante Prometheus, los equipos de desarrollo y operaciones pueden identificar patrones y correlaciones significativas. Esto permite una comprensión profunda de cómo las pruebas de carga impactan los recursos del cluster y cómo estos afectan directamente al rendimiento de la aplicación en producción. Grafana, con sus capacidades de visualización altamente personalizables, proporciona paneles intuitivos y gráficos dinámicos que facilitan la identificación de tendencias, cuellos de botella y áreas de mejora, permitiendo una toma de decisiones informada para optimizar la escalabilidad y la eficiencia del sistema.

La combinación de k6, Prometheus y Grafana proporciona una sinergia poderosa al permitir que los equipos monitoreen y analicen simultáneamente el rendimiento de las pruebas de carga y el comportamiento del cluster en producción. Esta integración facilita la identificación proactiva de posibles problemas y la optimización continua del rendimiento, respaldando la toma de decisiones basada en datos con una representación visual clara y detallada de la salud y eficiencia del sistema en tiempo real.

## Instalación de k6

En Debian/Ubuntu, seguimos los pasos de la [página de K6](#).

Simplemente correremos en consola lo siguiente:

```
sudo gpg -k
sudo gpg --no-default-keyring --keyring /usr/share/keyrings/k6-archive-keyring.gpg --
keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
C5AD17C747E3415A3642D57D77C6C491D6AC1D69
echo
"deb [signed-by=/usr/share/keyrings/k6-archive-keyring.gpg] https://dl.k6.io/deb stable
main" | sudo tee /etc/apt/sources.list.d/k6.list
sudo apt-get update
sudo apt-get install k6
```

## Correr el test de carga

Para correr cualquier test lo único que debemos hacer es poner en consola lo siguiente:

```
# on k6 dir
k6 run test-api/<nombre-del-test>.js
```

## Pruebas realizadas

Las pruebas se centraron en cumplir con los requisitos de resistencia del cliente, priorizando la capacidad de la interfaz de usuario para soportar más de 5000 usuarios simultáneos en su página principal. Además, se llevaron a cabo pruebas adicionales, como la creación masiva de usuarios y productos para evaluar la robustez de la API y la base de datos. Otra prueba se centró en medir el consumo de red al recuperar datos en formato base64 de imágenes completas desde la base de datos. Por último, se evaluó el rendimiento del procesamiento de la API, incluyendo pruebas de carga en el proceso de inicio de sesión que involucra tokens y contraseñas encriptadas con BCrypt. En todos los escenarios, se monitorearon detenidamente la CPU, la RAM, la lectura/escritura del disco y el consumo de red para obtener una visión completa del rendimiento del sistema.

### Test 1: UI

Archivo de configuración de K6 utilizado:

```
import http from 'k6/http';
import { sleep } from 'k6';

export const options = {
  stages: [
    { duration: '15s', target: 1000 },
    { duration: '30s', target: 2500 },
    { duration: '1m', target: 5000 },
  ],
};

export default function () {
  http.get('http://192.168.39.203:30001/');
  sleep(1);
}
```

Resultado de consola de k6:

```
data_received.....: 141 MB 1.1 MB/s
data_sent.....: 1.3 MB 10 kB/s
http_req_blocked.....: avg=74.96ms min=951ns med=5.29µs max=7.3s
p(90)=529.72µs p(95)=5.64ms
http_req_connecting.....: avg=42.82ms min=0s med=0s max=7.3s
p(90)=357.86µs p(95)=447.3µs
http_req_duration.....: avg=25.35s min=8.63ms med=23.68s max=1m0s
p(90)=46.43s p(95)=49.46s
{ expected_response:true }...: avg=24.8s min=8.63ms med=23.67s max=59.98s
```

```

p(90)=44.26s p(95)=47.46s
  http_req_failed.....: 1.57% ✓ 227 ✗ 14208
  http_req_receiving.....: avg=84.93µs min=0s med=80.47µs max=4.41ms
p(90)=113.12µs p(95)=127.36µs
  http_req_sending.....: avg=430.38ms min=5.32µs med=21.77µs max=9.55s
p(90)=83.96µs p(95)=4.17s
  http_req_tls_handshaking.....: avg=0s min=0s med=0s max=0s
p(90)=0s p(95)=0s
  http_req_waiting.....: avg=24.92s min=8.44ms med=23.68s max=1m0s
p(90)=41.57s p(95)=49.46s
  http_reqs.....: 14435 110.617989/s
  iteration_duration.....: avg=26.4s min=1s med=24.69s max=1m1s p(90)=47.
43s p(95)=50.46s
  iterations.....: 14435 110.617989/s
  vus.....: 744 min=42 max=4984
  vus_max.....: 5000 min=5000 max=5000

running (2m10.5s), 0000/5000 VUs, 14435 complete and 0 interrupted iterations
default ✓ [=====] 0000/5000 VUs
1m45s

```

Resultado de Grafana:



Test 2: User login

Archivo de configuración de K6 utilizado:

```

import http from 'k6/http';
import { check, sleep } from 'k6';

export const options = {
  stages: [
    { duration: '15s', target: 1000 },
    { duration: '30s', target: 2500 },
    { duration: '1m', target: 5000 },
  ],
};

export default function () {
  // Datos del usuario para el login
  const username = 'aagustin@gmail.com';
  const password = '12345678';

  // Realizar la solicitud POST para el login del usuario
  const res = http.post('http://192.168.39.203:31000/auth/login', {
    grant_type: '',
    username,
    password,
    scope: '',
    client_id: '',
    client_secret: '',
  }, {
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded',
      'Accept': 'application/json',
    },
  });

  // Verificar si la respuesta es exitosa (código 2xx)
  check(res, {
    'status was 2xx': (r) => r.status >= 200 && r.status < 300,
  });

  // Introducir una pausa de 1 segundo entre las solicitudes
  sleep(1);
}

```

Resultado de consola de k6:

```

✗ status was 2xx
  ↳ 3% — ✓ 407 / ✗ 12074

checks.....: 3.26% ✓ 407 ✗ 12074
data_received.....: 126 kB 993 B/s
data_sent.....: 3.5 MB 28 kB/s
http_req_blocked.....: avg=513.73ms min=2.45µs med=1.11ms max=3.13s
p(90)=3.02s p(95)=3.03s
http_req_connecting.....: avg=513.25ms min=0s med=949.58µs max=3.05s
p(90)=3.02s p(95)=3.03s
http_req_duration.....: avg=29.51s min=452.04ms med=31.15s max=50.44s
p(90)=37.05s p(95)=41.74s
{ expected_response:true }...: avg=9.24s min=452.04ms med=7.96s max=32.5s

```

```

p(90)=19.42s p(95)=27.41s
  http_req_failed.....: 96.73% ✓ 12074 ✗ 407
  http_req_receiving.....: avg=4.57µs min=0s med=0s max=8.86ms
p(90)=0s p(95)=0s
  http_req_sending.....: avg=718.89µs min=11.89µs med=66.83µs max=101.88ms
p(90)=510.25µs p(95)=1.15ms
  http_req_tls_handshaking.....: avg=0s min=0s med=0s max=0s
p(90)=0s p(95)=0s
  http_req_waiting.....: avg=29.51s min=451.86ms med=31.15s max=50.44s
p(90)=37.05s p(95)=41.74s
  http_reqs.....: 12481 98.263787/s
  iteration_duration.....: avg=31.22s min=1.46s med=33.61s max=51.91s
p(90)=38.37s p(95)=43.05s
  iterations.....: 12481 98.263787/s
  vus.....: 982 min=41 max=4995
  vus_max.....: 5000 min=5000 max=5000

```

running (2m07.0s), 0000/5000 VUs, 12481 complete and 0 interrupted iterations  
 default ✓ [=====] 0000/5000 VUs  
 1m45s

Resultado de Grafana:



Test 3: Register new users

Archivo de configuración de K6 utilizado:

```

import http from 'k6/http';
import { check, sleep } from 'k6';

// Función para generar una cadena aleatoria de longitud dada
function generateRandomString(length) {
  const charset =
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
  let result = '';
  for (let i = 0; i < length; i++) {
    const randomIndex = Math.floor(Math.random() * charset.length);
    result += charset.charAt(randomIndex);
  }
  return result;
}

export const options = {
  stages: [
    { duration: '15s', target: 1000 },
    { duration: '30s', target: 2500 },
    { duration: '1m', target: 5000 },
  ],
};

export default function () {
  // Generar datos aleatorios para cada usuario
  const email = `user${Math.floor(Math.random() * 100000)}@example.com`;
  const password = generateRandomString(8);

  // Crear el objeto de usuario
  const user = {
    email,
    password,
  };

  // Realizar la solicitud POST para registrar un nuevo usuario
  const res = http.post('http://192.168.39.203:31000/auth/register', JSON.stringify({
    user }, {
    headers: {
      'Content-Type': 'application/json',
      'Accept': 'application/json',
    },
  }));

  // Verificar si la respuesta es exitosa (código 2xx)
  check(res, {
    'status was 2xx': (r) => r.status >= 200 && r.status < 300,
  });

  // Introducir una pausa de 1 segundo entre las solicitudes
  sleep(1);
}

```

Resultado de consola de k6:



```

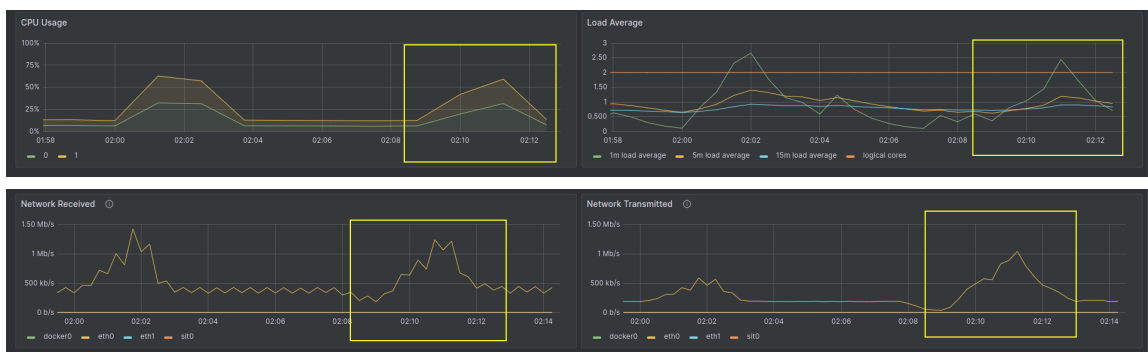
X status was 2xx
  ↳ 2% — ✓ 345 / X 11276

checks.....: 2.96% ✓ 345    X 11276
data_received.....: 122 kB 952 B/s
data_sent.....: 2.8 MB 22 kB/s
http_req_blocked.....: avg=241.39ms min=2.51µs med=610.72µs max=7.14s
p(90)=1.03s p(95)=1.05s
http_req_connecting.....: avg=241.29ms min=0s med=532.82µs max=7.14s
p(90)=1.03s p(95)=1.05s
http_req_duration.....: avg=31.44s min=372.26ms med=31.69s max=58.3s
p(90)=41.86s p(95)=46.66s
{ expected_response:true }...: avg=8.98s min=372.26ms med=6.89s max=37.73s
p(90)=20.51s p(95)=25.54s
http_req_failed.....: 97.03% ✓ 11276    X 345
http_req_receiving.....: avg=2.46µs min=0s med=0s max=532.61µs
p(90)=0s p(95)=0s
http_req_sending.....: avg=229.78µs min=10.67µs med=59.28µs max=11.64ms
p(90)=408.53µs p(95)=1.26ms
http_req_tls_handshaking.....: avg=0s min=0s med=0s max=0s
p(90)=0s p(95)=0s
http_req_waiting.....: avg=31.44s min=372.09ms med=31.69s max=58.3s
p(90)=41.86s p(95)=46.66s
http_reqs.....: 11621 91.116311/s
iteration_duration.....: avg=32.78s min=1.37s med=32.84s max=59.53s
p(90)=43.03s p(95)=47.82s
iterations.....: 11621 91.116311/s
vus.....: 766 min=41 max=4994
vus_max.....: 5000 min=5000 max=5000

running (2m07.5s), 0000/5000 VUs, 11621 complete and 0 interrupted iterations
default ✓ [=====] 0000/5000 VUs
1m45s

```

Resultado de Grafana:



**Memory Usage**

4.66 GiB  
3.73 GiB  
2.79 GiB  
1.86 GiB  
954 MiB  
0 B

01:59 02:00 02:01 02:02 02:03 02:04 02:05 02:06 02:07 02:08 02:09 02:10 02:11 02:12 02:13

memory used memory buffers memory cached memory free

**Disk I/O**

15 MiB/s  
10 MiB/s  
5 MiB/s  
0 B/s

02:00 02:02 02:04 02:06 02:08 02:10 02:12 02:14

vda read vda written vda io time

**Disk Space Usage**

| Mounted on          | Size    | Available | Used    | Used % |
|---------------------|---------|-----------|---------|--------|
| /var/lib/containers | 9.33 GB | 2.54 GB   | 6.79 GB | 72.8%  |
| /var/lib/cni        | 9.33 GB | 2.54 GB   | 6.79 GB | 72.8%  |
| /var/lib/bulkkit    | 9.33 GB | 2.54 GB   | 6.79 GB | 72.8%  |
| /var/lib/containers | 9.33 GB | 2.54 GB   | 6.79 GB | 72.8%  |

```

    "amount": ${amount}
  }
}, {
  headers: {
    'Content-Type': 'application/json',
    'Authorization': `Bearer ${userToken}`,
  },
});

// Verificar si la respuesta es exitosa (código 2xx)
check(res, {
  'status was 2xx': (r) => r.status >= 200 && r.status < 300,
});

// Introducir una pausa de 1 segundo entre las solicitudes
sleep(1);
}

```

Resultado de consola de k6:

```

✗ status was 2xx
↳ 10% — ✓ 1316 / ✗ 10795

checks.....: 10.86% ✓ 1316 ✗ 10795
data_received.....: 323 kB 2.4 kB/s
data_sent.....: 5.6 MB 42 kB/s
http_req_blocked.....: avg=203.4ms min=2.42µs med=755.2µs max=1.03s
p(90)=1.02s p(95)=1.02s
http_req_connecting.....: avg=203.31ms min=0s med=638.59µs max=1.03s
p(90)=1.02s p(95)=1.02s
http_req_duration.....: avg=30.57s min=8.11ms med=30.84s max=55.6s
p(90)=47.06s p(95)=50.89s
{ expected_response:true }...: avg=1.54s min=8.11ms med=287.59ms max=27.99s
p(90)=1.17s p(95)=3.11s
http_req_failed.....: 89.13% ✓ 10795 ✗ 1316
http_req_receiving.....: avg=11.27µs min=0s med=0s max=5.32ms p(90)=49.
44µs p(95)=74.14µs
http_req_sending.....: avg=105.04µs min=10.8µs med=57.93µs max=4.71ms
p(90)=206.62µs p(95)=298.7µs
http_req_tls_handshaking.....: avg=0s min=0s med=0s max=0s
p(90)=0s p(95)=0s
http_req_waiting.....: avg=30.57s min=8.02ms med=30.84s max=55.6s
p(90)=47.06s p(95)=50.89s
http_reqs.....: 12111 89.698041/s
iteration_duration.....: avg=31.93s min=1s med=32.03s max=57.1s p(90)=48.
34s p(95)=52.15s
iterations.....: 12111 89.698041/s
vus.....: 94 min=34 max=4999
vus_max.....: 5000 min=5000 max=5000

running (2m15.0s), 0000/5000 VUs, 12111 complete and 94 interrupted iterations
default ✓ [=====] 0060/5000 VUs
1m45s

```

## Resultado de Grafana:



## Test 5: Get images

Archivo de configuración de K6 utilizado:

```
import http from 'k6/http';
import { check, sleep } from 'k6';

export const options = {
  stages: [
    { duration: '15s', target: 1000 },
    { duration: '30s', target: 2500 },
    { duration: '1m', target: 5000 },
  ],
};

export default function () {
  // Proporcionar una lista de IDs de las imágenes
  const imageIds = [
    '57d2adf1-8062-4908-a73f-02f4c9320d35',
    '23598a4e-2d32-4b3a-b55e-00773c8cbe46',
  ];

  // Realizar solicitudes GET para obtener imágenes
  imageIds.forEach((imageId) => {
    const res = http.get('http://192.168.39.203:31000/image/get/?id=${imageId}', {
      headers: {
```

```

    'Accept': 'application/json',
  },
});

// Verificar si la respuesta es exitosa (código 2xx)
check(res, {
  'status was 2xx': (r) => r.status >= 200 && r.status < 300,
});

// Introducir una pausa de 1 segundo entre las solicitudes
sleep(1);
});
}

```

Resultado de consola de k6:

```

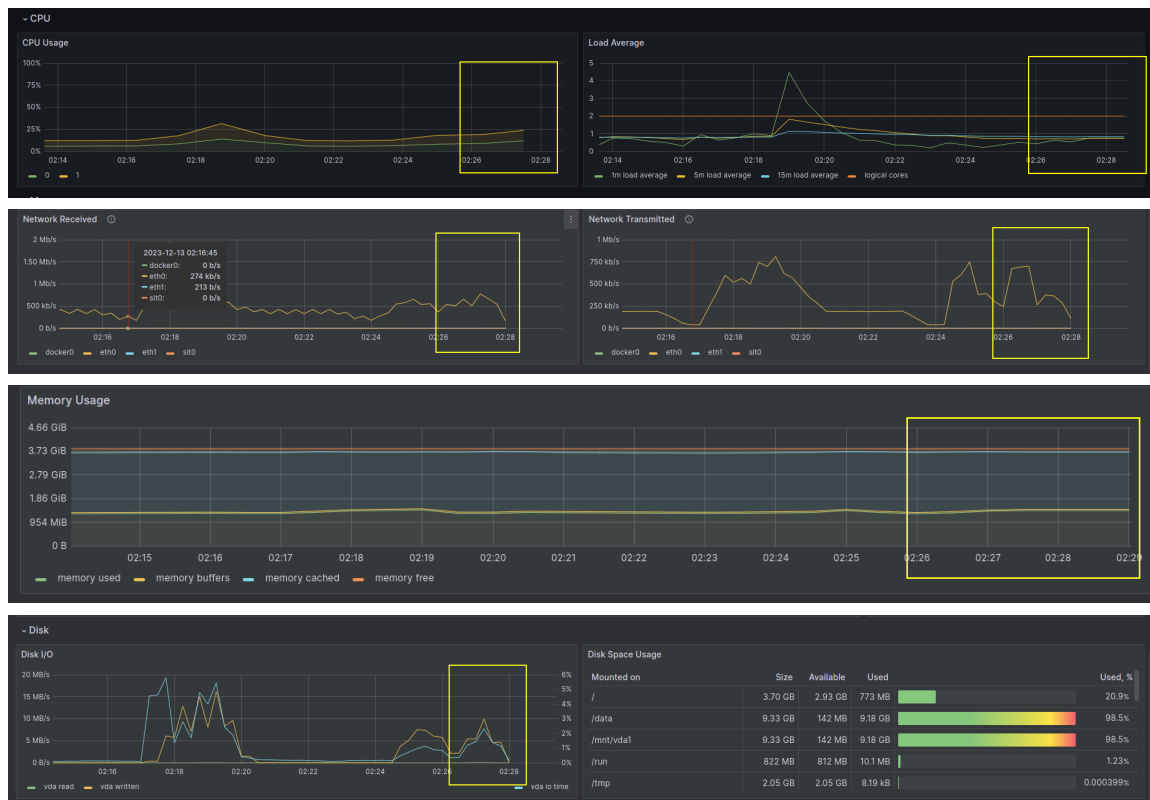
✗ status was 2xx
↳ 0% — ✓ 46 / ✗ 12242

checks.....: 0.37% ✓ 46      ✗ 12242
data_received.....: 2.2 MB 16 kB/s
data_sent.....: 2.2 MB 17 kB/s
http_req_blocked.....: avg=781.12ms min=4.71µs med=1.15ms max=7.24s
p(90)=3.03s p(95)=3.06s
http_req_connecting.....: avg=780.93ms min=0s med=991.98µs max=7.24s
p(90)=3.03s p(95)=3.06s
http_req_duration.....: avg=30.27s min=73.76ms med=30.33s max=1m0s
p(90)=48.87s p(95)=54.06s
{ expected_response:true }...: avg=419.68ms min=73.76ms med=414.18ms max=800.
17ms p(90)=684.03ms p(95)=763.76ms
http_req_failed.....: 99.62% ✓ 12242 ✗ 46
http_req_receiving.....: avg=254.2ms min=0s med=0s max=30.77s
p(90)=0s p(95)=0s
http_req_sending.....: avg=74.72ms min=11.36µs med=97.25µs max=26.8s
p(90)=704.63µs p(95)=1.56ms
http_req_tls_handshaking.....: avg=0s min=0s med=0s max=0s
p(90)=0s p(95)=0s
http_req_waiting.....: avg=29.95s min=6.22ms med=30.33s max=1m0s
p(90)=48.55s p(95)=53.75s
http_reqs.....: 12288 90.987543/s
iteration_duration.....: avg=1m4s min=26.73s med=1m7s max=1m35s
p(90)=1m28s p(95)=1m30s
iterations.....: 5423 40.155065/s
vus.....: 1442 min=42 max=4999
vus_max.....: 5000 min=5000 max=5000

running (2m15.1s), 0000/5000 VUs, 5423 complete and 1442 interrupted iterations
default ✓ [=====] 1432/5000 VUs
1m45s

```

Resultado de Grafana:



## Conclusiones de las pruebas realizadas

En términos generales, hemos logrado cumplir con los requisitos establecidos por la empresa, confirmando que la plataforma es capaz de soportar hasta 5000 usuarios en su página de inicio, aunque esta evaluación consideró un número limitado de productos al inicio de la prueba. No obstante, las pruebas adicionales revelaron limitaciones notables, principalmente relacionadas con la capacidad del disco tanto en términos de la cantidad de productos y usuarios que podemos agregar como en su rendimiento de lectura/escritura, siendo este un factor determinante en las operaciones que involucran la base de datos.

En todas las instancias en las que la base de datos estuvo involucrada, como la creación de nuevos productos y usuarios, el rendimiento del disco se destacó como el factor crítico. Observamos que las pruebas centradas en usuarios afectaron más al CPU, especialmente cuando la API tuvo que encriptar o desencriptar contraseñas. A lo largo de todas las pruebas, la memoria mostró una estabilidad notable, mientras que la red experimentó una mayor actividad durante las transmisiones y recepciones de información en las solicitudes a la API.

Como conclusión, consideramos que el sistema es fiable para el uso actual de la empresa. Sin embargo, en caso de requerir una mayor creación de usuarios o un mayor número de solicitudes, se sugiere la implementación de un Ingress Controller y un balanceador de carga, junto con el escalado de los Deployments

de la UI, la API y la DB. En particular, para la base de datos, se recomienda configurarla como un ReplicaSet para optimizar su rendimiento.