Esta guía describe cómo configurar la infraestructura para servir la solución propuesta en el trabajo de fin de grado.

Primero necesitaremos unn grupo de maquinas (virtuales y/o reales) que esten en una misma subred y esta tenga acceso a internet.

1 cambiar la config de internet de la **máquina virtual** y ponerla en una **red Nat** (para que en virtualbox se puedan ver las máquinas entre ellas y puedan acceder a internet.

2. añadir usuario a suddoers. (usar groups para ver si es sudo)

```
Unset
adduser nombre_usuario sudo
```

```
adduser vagrant sudo
```
hacer logout de la sesion.

3. instalar ssh (server en la maquina virtual) necesario para la instalacion sencilla
    1. Poner las replicas en sources para poder actualizar.

```
Unset
sudo nano /etc/apt/sources.list
```

Escribir esto y quitar lo anterior de sources.list

```
Unset
deb http://deb.debian.org/debian/ bookworm main
non-free-firmware
deb-src http://deb.debian.org/debian/ bookworm main
non-free-firmware

deb http://security.debian.org/debian-security
bookworm-security main non-free-firmware
deb-src http://security.debian.org/debian-security
bookworm-security main non-free-firmware

# bookworm-updates, to get updates before a point release
is made;
# see
https://www.debian.org/doc/manuals/debian-reference/ch02.
en.html#_updates_and_backports
deb http://deb.debian.org/debian/ bookworm-updates main
non-free-firmware
deb-src http://deb.debian.org/debian/ bookworm-updates
main non-free-firmware
```

```
sudo apt update
```

2 instalar ssh

```
sudo apt-get install openssh-server
```

3 cambiar la configuración en para permitir loguin por pubkey y root(necesito escalabilidad sudo sin que nunca pregunte contraseña)

```
sudo nano /etc/ssh/sshd_config
```

PermitRootLogin yes
PubkeyAuthentication yes

4 reiniciar el servicio

```
sudo systemctl restart sshd
```

4 hacer esto en la máquina de **controlexterno**

```
ssh-keygen
```

5 copiar la clave pública a todas las maquinas

```
ssh-copy-id root@10.0.2.6;
```

```
ssh-copy-id root@10.0.2.4;ssh-copy-id root@10.0.2.5
```

7 en la máquina **controlexterno** descargamos el instalador

```
mkdir kubernetes

cd kubernetes
```

Unset

```
wget https://github.com/k0sproject/k0sctl/releases/download/v0.17.5/k0sctl-linux-x64 && mv k0sctl-linux-x64 k0sctl && chmod 777 k0sctl
```

8 con ./k0sctl init > k0sctl.yaml podemos ponerlo pero aqui esta ya el hecho para esta configuración **olvidarse de lo de abajo y hacerlo a mano (no funciona copiar y pegar por el indentado)** se puede copiar pero hay que editarlo luego.

Python

```
apiVersion: k0sctl.k0sproject.io/v1beta1
kind: Cluster
metadata:
  name: k0s-cluster
spec:
  hosts:
  - ssh:
      address: 10.0.0.4
      user: root
      port: 22
      keyPath: /home/vagrant/.ssh/id_rsa
      role: controller+worker
  - ssh:
      address: 10.0.0.5
      user: root
      port: 22
      keyPath: /home/vagrant/.ssh/id_rsa
      role: worker
  - ssh:
      address: 10.0.0.6
      user: root
      port: 22
      keyPath: /home/vagrant/.ssh/id_rsa
      role: worker
  - ssh:
      address: 10.0.0.7
      user: root
      port: 22
      keyPath: /home/vagrant/.ssh/id_rsa
      role: worker
  k0s:
      version: null
      versionChannel: stable
      dynamicConfig: false
      config: {}
```

9 aplicar la configuración

```
sudo ./k0sctl apply --config k0sctl.yaml

./k0sctl kubeconfig > kubeconfig

export KUBECONFIG=/home/vagrant/kubernetes/kubeconfig
```

10 instalar **kubectl** y **curl**

```
sudo apt install curl

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

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

sudo kubectl version --client
```

10 instalar **snap** para helm

```
sudo apt-get install snapd
```

11 poner el path en la consola

```
sudo nano /home/vagrant/.bashrc
```

añadir

```
export PATH=$PATH:/snap/bin
export KUBECONFIG=/home/vagrant/kubernetes/kubeconfig
```

12 instalar **helm**

```
sudo snap install helm --classic
```

13 instalar el repo de superset

```
Unset
helm repo add superset https://apache.github.io/superset
```

Unset

```
helm search repo superset
```

14 configurar los nuevos discos para el almacenamiento de datos

Unset

```
#!/bin/bash

# Este script configura un disco para usar LUKS y LVM en
Debian o sistemas similares.
# Debes proporcionar el nombre del disco (por ejemplo,
sdb) y un identificador único.

if [[ $# -ne 2 ]]; then
     echo "Uso: $0 <nombre_del_disco>
<identificador_unico>"
     echo "Ejemplo: $0 sdb 1"
     exit 1
fi

sudo apt update
sudo apt install cryptsetup mdadm lvm2

DISCO="/dev/$1"
IDENTIFICADOR=$2
NOMBRE_CIFRADO="crypt${IDENTIFICADOR}"
GRUPO_VOLUMEN="vg${IDENTIFICADOR}"
VOLUMEN_LOGICO="lv${IDENTIFICADOR}"
PUNTO_MONTAJE="/mnt/encrypted_${IDENTIFICADOR}"

# Paso 1: Cifrado del disco con LUKS
echo "Cifrando el disco $DISCO..."
sudo cryptsetup luksFormat $DISCO

echo "Abriendo el disco cifrado..."
sudo cryptsetup open $DISCO $NOMBRE_CIFRADO
```

```
# Paso 2: Configuración de LVM
echo "Creando volumen físico (PV)..."
sudo pvcreate /dev/mapper/$NOMBRE_CIFRADO

echo "Creando grupo de volúmenes (VG)..."
sudo vgcreate $GRUPO_VOLUMEN /dev/mapper/$NOMBRE_CIFRADO

echo "Creando volumen lógico (LV)..."
sudo lvcreate -l 100%FREE -n $VOLUMEN_LOGICO
$GRUPO_VOLUMEN

# Paso 3: Crear y montar el sistema de archivos
echo "Formateando el volumen lógico con ext4..."
sudo mkfs.ext4 /dev/$GRUPO_VOLUMEN/$VOLUMEN_LOGICO

echo "Creando punto de montaje..."
sudo mkdir -p $PUNTO_MONTAJE

echo "Montando el volumen lógico..."
sudo mount /dev/$GRUPO_VOLUMEN/$VOLUMEN_LOGICO
$PUNTO_MONTAJE

# Paso 4: Configuración automática del montaje al inicio
echo "Configurando desbloqueo automático y montaje al
inicio..."
echo "$NOMBRE_CIFRADO $DISCO none luks,discard" | sudo
tee -a /etc/crypttab
echo "/dev/$GRUPO_VOLUMEN/$VOLUMEN_LOGICO $PUNTO_MONTAJE
ext4 defaults 0 2" | sudo tee -a /etc/fstab

echo "Configuración completa. El disco $DISCO está
cifrado y configurado con LVM en $PUNTO_MONTAJE."
```

chmod 777 setup_disk.sh

uso: ./setup_disk.sh sdb 1

15 crear el **pv**(persistent volumen) en el cluster

apiVersion: v1
kind: PersistentVolume
metadata:

```
  name: local-pv-3
spec:
  capacity:
        storage: 19Gi  # Ajusta la capacidad según sea necesario
  volumeMode: Filesystem
  accessModes:
        - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  local:
        path: /mnt/encrypted_1
  nodeAffinity:
        required:
        nodeSelectorTerms:
        - matchExpressions:
        - key: kubernetes.io/hostname
        operator: In
        values:
                - worker3


apiVersion: v1
kind: PersistentVolume
metadata:
  name: local-pv-2
spec:
  capacity:
        storage: 19Gi  # Ajusta la capacidad según sea necesario
  volumeMode: Filesystem
  accessModes:
        - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  local:
        path: /mnt/encrypted_1
  nodeAffinity:
        required:
        nodeSelectorTerms:
        - matchExpressions:
        - key: kubernetes.io/hostname
        operator: In
        values:
                - worker2
```

kubectl apply -f local-pv-2.yaml

kubectl apply -f local-pv-3.yaml

comando para ver que se crearon bien
`kubectl get pv`
16 configurar el **storageClass** correspondiente

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: local-storage
provisioner: kubernetes.io/no-provisioner
volumeBindingMode: WaitForFirstConsumer
reclaimPolicy: Retain

`kubectl apply -f local-storage-class.yaml`

kubectl get storageclass

17 Configurar los valores de **my-values.yalm** para desplegar la aplicación en el cluster.
cambiar

storageClass: local-storage

y en SECRET poner la clave esa generada.

```
extraSecretEnv:
  # MAPBOX_API_KEY: ...
  # # Google API Keys: https://console.cloud.google.com/apis/credentials
  # GOOGLE_KEY: ...
  # GOOGLE_SECRET: ...
  #   # Generate your own secret key for encryption. Use openssl rand -base64 42 to generate a good key
        SUPERSET_SECRET_KEY: 'PONER LA CLAVE AQUI'
```

helm upgrade --install --values my-values.yaml superset superset/superset

kubectl port-forward service/superset 8088:8088 --namespace default

Unset

```
helm install superset superset/superset -f my-values.yaml
```

Unset

```
k0s config export > k0s-config-current.yaml
```

18 Instalando lo necesario para poder acceder a la aplicacion (MetalLB, ingress, etc)

kubectl apply -f
https://raw.githubusercontent.com/metallb/metallb/v0.14.5/config/manifests/metallb-native.yaml

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: first-pool
  namespace: metallb-system
spec:
  addresses:
  - 10.0.2.100-10.0.2.110
```

kubectl apply -f pool-1.yaml

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: example
  namespace: metallb-system
spec:
  ipAddressPools:
  - first-pool
```

kubectl apply -f l2-advertisement.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: superset-web
  labels:
        app.kubernetes.io/name: superset  # Asegúrate de que esto coincide con
cómo etiquetas tus recursos relacionados con Superset
spec:
  selector:
        app: superset  # Esto debe coincidir exactamente con las etiquetas en los
pods de Superset
  ports:
  - name: http
        port: 8089     # Puerto en el que el servicio estará disponible externamente
        protocol: TCP
```

```
    targetPort: 8088  # Asume que Superset está escuchando en el puerto 8088 dentro de los pods
  type: LoadBalancer
```

kubectl apply -f superset-web-service.yaml

```
helm pull oci://ghcr.io/nginxinc/charts/nginx-ingress --untar --version 1.2.0

cd nginx-ingress

kubectl apply -f crds
```

helm install nginx-ingress .

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: superset
  annotations:
    kubernetes.io/ingress.class: "nginx"
spec:
  rules:
  - host: superset.home-k8s.lab
    http:
    paths:
    - path: /
    pathType: Prefix
    backend:
    service:
    name: superset
    port:
    number: 8088
```

lo de arriba es antiguo lo nuevo con la configuración para que no pete cuando hace consultas:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: superset
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.org/client-max-body-size: "100m"
    nginx.org/proxy-connect-timeout: "5m"
    nginx.org/proxy-read-timeout: "5m"
    nginx.org/proxy-send-timeout: "5m"
```

```
spec:
  rules:
  - host: superset.home-k8s.lab
        http:
        paths:
        - path: /
        pathType: Prefix
        backend:
        service:
        name: superset
        port:
        number: 8088
```

kubectl apply -f ingress.yaml

apt install python3-psycopg2 python3-requests python3-pandas

para que pueda usar https

Paso 1: Crear un Certificado TLS Autofirmado

openssl genpkey -algorithm RSA -out tls.key -pkeyopt rsa_keygen_bits:2048

openssl req -new -x509 -key tls.key -out tls.crt -days 365

Paso 2: Crear un Secreto en Kubernetes (en namespace default)

kubectl create secret tls default-cert --cert=tls.crt --key=tls.key

Paso 3: Configurar el `values.yaml` del Chart de Helm

Edita tu archivo `values.yaml` para configurar el secreto TLS por defecto.

1. **Editar `values.yaml`:**
   Busca las secciones `controller.defaultTLS.secret` y `controller.wildcardTLS.secret` en tu archivo `values.yaml` y configúralas para usar el secreto que creaste.
   yaml

```
controller:
  ...
  defaultTLS:
        secret: "<tu-namespace>/default-cert"
  ...
  wildcardTLS:
        secret: "<tu-namespace>/default-cert"
```

esto se puede consegir de donde se descargo el repo con helm y ahi esta values.yaml

Paso 4: Desplegar el Chart de Helm

Despliega el chart de Helm con la configuración actualizada.

helm upgrade --install nginx-ingress nginx-ingress -f values-nginx-ingress.yaml

despues cambiar la configuración de ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: superset
  annotations:
        kubernetes.io/ingress.class: "nginx"
        nginx.org/client-max-body-size: "100m"
        nginx.org/proxy-connect-timeout: "5m"
        nginx.org/proxy-read-timeout: "5m"
        nginx.org/proxy-send-timeout: "5m"

spec:
  tls:
  - hosts:
        - superset.home-k8s.lab
        secretName: default-cert
  rules:
  - host: superset.home-k8s.lab
        http:
        paths:
        - path: /
        pathType: Prefix
        backend:
        service:
        name: superset
```

port:
number: 8088

Creación del cargador ETL(no necesario ya que el contenedor usado esta subido y es público)

1 instalar docker

Paso 1: Actualizar el sistema

Antes de instalar cualquier paquete, es una buena práctica actualizar la lista de paquetes y asegurarte de que tu sistema está actualizado.

bash
```
sudo apt update
sudo apt upgrade -y
```

Paso 2: Instalar paquetes necesarios

Algunos paquetes necesarios deben ser instalados para que Docker se instale correctamente.

bash
```
sudo apt install apt-transport-https ca-certificates curl
software-properties-common gnupg2 -y
```

Paso 3: Agregar la clave GPG oficial de Docker

Para asegurar que el software que estás instalando es auténtico, añade la clave GPG del repositorio oficial de Docker.

bash
```
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo
apt-key add -
```

Paso 4: Añadir el repositorio de Docker a APT sources

Agrega el repositorio de Docker a los repositorios de APT de tu sistema para que puedas instalar y actualizar Docker desde el repositorio oficial.

bash

```
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/debian $(lsb_release -cs)
stable"
```

Paso 5: Actualizar la base de datos de paquetes APT

Después de añadir el nuevo repositorio, actualiza la base de datos de paquetes APT.

bash
```
sudo apt update
```

Paso 6: Instalar Docker CE (Community Edition)

Ahora, instala Docker Community Edition (la versión gratuita de Docker) y otros paquetes esenciales para Docker.

bash
```
sudo apt install docker-ce docker-ce-cli containerd.io -y
```

Paso 7: Verificar la instalación

Una vez que la instalación haya terminado, verifica que Docker esté instalado correctamente ejecutando el siguiente comando.

bash
```
sudo systemctl status docker
```

Este comando mostrará el estado del servicio de Docker. Deberías ver que está activo y en ejecución.

Paso 8: Agregar tu usuario al grupo Docker

Para ejecutar comandos de Docker sin `sudo`, añade tu usuario al grupo Docker.

bash
```
sudo usermod -aG docker ${USER}
```

Después de ejecutar este comando, cierra la sesión y vuelve a iniciarla para que los cambios tengan efecto, o puedes ejecutar `su - ${USER}` para recargar los grupos de usuario.

Paso 9: Probar Docker

Finalmente, prueba que Docker esté funcionando correctamente.

bash

```
Unset
docker run hello-world
```

2 hacer un contenedor

Paso 1: Crear el Dockerfile

Aquí te muestro cómo crear un `Dockerfile` que instale Python y todas las dependencias necesarias:

dockerfile

```
# Usar una imagen base oficial de Python
FROM python:3.9-slim

# Establecer el directorio de trabajo
WORKDIR /app

# Copiar el script de Python en el contenedor
COPY . /app

# Instalar paquetes necesarios usando pip
RUN pip install --no-cache-dir requests pandas numpy
psycopg2-binary

# Comando para ejecutar el script
CMD ["python", "./your_script.py"]
```

Detalles del Dockerfile

- **`FROM python:3.9-slim`**: Esto establece la imagen base como Python 3.9 slim, que es una versión ligera adecuada para muchos proyectos Python.
- **`WORKDIR /app`**: Esto establece `/app` como el directorio de trabajo donde se ejecutará el comando. Cualquier instrucción `RUN`, `CMD`, `ENTRYPOINT`, `COPY` y `ADD` que siga en el Dockerfile se ejecutará en este directorio.
- **`COPY . /app`**: Copia todos los archivos del directorio actual (donde está el Dockerfile) en el contenedor, en el directorio `/app`.

- `RUN pip install ...`: Instala las dependencias necesarias en el contenedor.

Paso 2: Construir la Imagen de Docker

Una vez que tienes el `Dockerfile`, construye la imagen de Docker ejecutando:

bash
```
docker build -t my-python-app .
```

Este comando construye la imagen usando el Dockerfile en el directorio actual y la etiqueta como `my-python-app`. ejecutarlo con "`docker build --no-cache -t my-python-app .`"si da **problemas.**

Paso 3: Ejecutar el Contenedor

Después de construir la imagen, ejecuta el contenedor:

bash

Unset
```
docker run --name my-running-app my-python-app
```

3 Subir el contenedor a la nube

`docker login`

acordarse de que tiene que ser el usuario todo en minusculas

1. **Subir la Imagen al Registro de Contenedores**
   Antes de poder desplegar tu contenedor en Kubernetes, necesitas subir la imagen del contenedor a un registro accesible por tu cluster de Kubernetes. Aquí usaremos Docker Hub como ejemplo.
   1. **Etiquetar tu imagen**:
      bash

```
docker tag my-python-app cmv0099/my-python-app:v1
```

   2. **Subir tu imagen**:
      bash

```
docker push cmv0099/my-python-app:v1
```

Asegúrate de reemplazar `yourusername` con tu nombre de usuario en el registro de Docker.

tambien si usas el mismo tag hay que asegurarse que kubernetes descarga el nuevo contenedor para ello se puede hacer imagePullPolicy: Always

apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: mi-cronjob
spec:
  schedule: "*/5 * * * *"
  jobTemplate:
        spec:
        template:
        spec:
        containers:
        - name: mi-contenedor
        image: mi-registro/mi-imagen:mi-tag
        imagePullPolicy: Always
        restartPolicy: OnFailure

. Poner el contenedor en kubernetes

- **Paso 1: Crear un Secreto en Kubernetes**
  Supongamos que necesitas configurar las variables de entorno `USUARIO` y `PASSWORD`. Deberías crear un secreto en Kubernetes para almacenar esta información de forma segura.
    - **Crear un archivo de secreto (secret.yaml)**:
      yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  USUARIO: Cambiar_esto  # Base64 encoded value
  PASSWORD: Cambiar_esto   # Base64 encoded value
```

- Asegúrate de codificar tus valores en Base64.
- **Aplicar el secreto en Kubernetes y el resto de secretos de los comandos siguientes**:
  bash

```bash
kubectl apply -f secret.yaml
kubectl apply -f python-etl-auth-secret.yaml
kubectl apply -f python-etl-data-trafficcontrol-secret.yaml
kubectl apply -f python-etl-db-secret.yaml
kubectl apply -f ingress.yaml
```

para habilitar el trabajo cron de ETL ejecutar:

```
kubectl apply -f my-python-etl-capacity-cronjob.yaml
```

```
kubectl apply -f my-python-etl-trafficcontrol-cronjob.yaml
```

Para instalar prophet debes añadir esto en el despliegue de superset:

Unset

```
RUN pip install --upgrade pip
RUN pip install lunarcalendar tqdm "pystan<3.0" && pip
install "prophet>=1.0.1, <1.1"
RUN pip install holidays==0.13
```

IMPORTANTE necesita más de 8GB de memoria ram para poder instalarse