Instalar Minikube:

Es necesario instalar Minikube en un sistema tipo Unix. Esto significa que esta guía cubre como instalar y ejecutar Minikube en una de las siguientes opciones:

- En Linux
- En macOS
- En Windows desde Windows Subsystem for Linux (WSL)

Instalar Minikube en un sistema tipo Unix:

IMPORTANTE: Cuando se especifica abrir una terminal es necesario abrir una terminal desde un sistema tipo Unix. Esto implica que si su sistema operativo es Windows, es necesario abrir una terminal de WSL.

1. Abrir una terminal y ejecutar el siguiente comando:

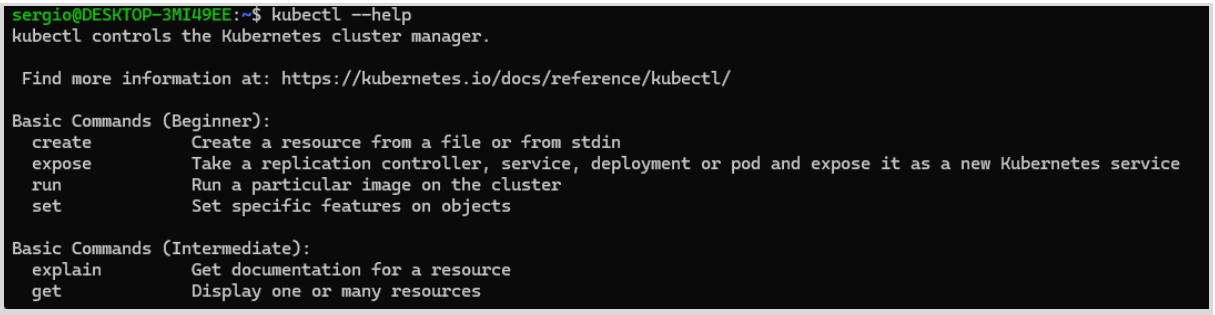
curl -LO https://github.com/kubernetes/minikube/releases/latest/download/minikube-linux-amd64

2. Ejecutar el siguiente comando

sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64

3. Comprobar que se ha instalado ejecutando:

kubectl --help

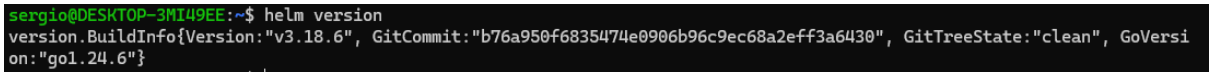


4. Instalar helm ejecutando el siguiente comando:

sudo snap install helm --classic

5. Comprobar que helm se ha instalado correctamente ejecutando el siguiente comando

helm version

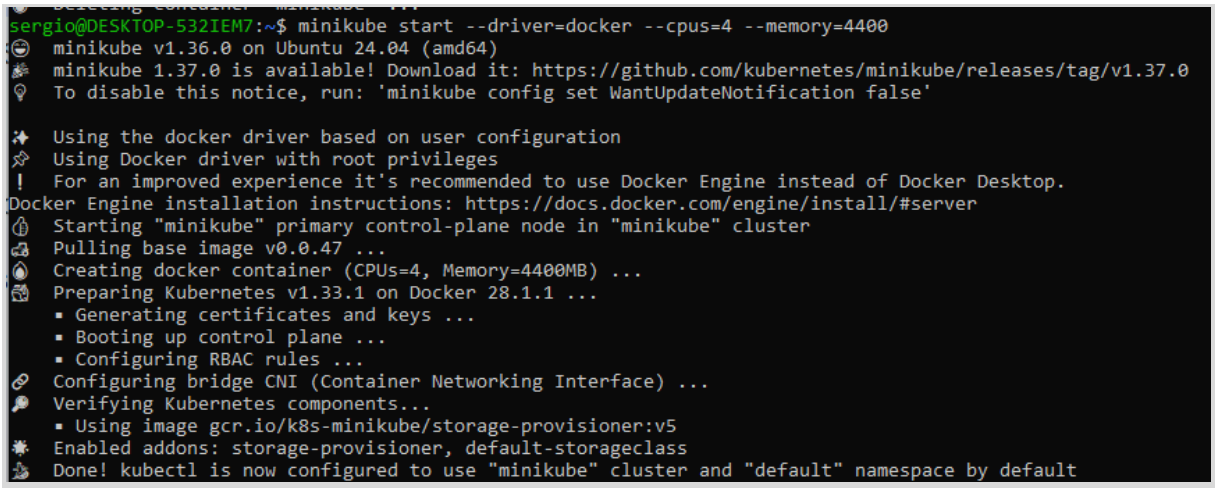


Crear Cluster:

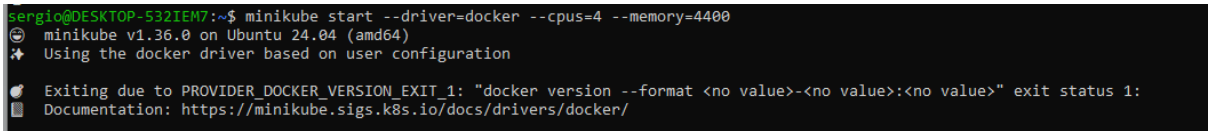
IMPORTANTE es necesario ejecutar el comando especificado en el paso 1 cada vez que se reinicia el ordenador (al apagar el ordenador el cluster muere). Recordad que es necesario tener abierto Docker Desktop

1. Crear el cluster ejecutando el siguiente comando. Es necesario tener **abierto Docker Desktop**. En este caso se indica que el driver a utilizar por Minikube es docker, el número de cpus asociadas a kubernetes va a ser 2, y la RAM asociada asociada a Kubernetes va a ser 4400 mb.

`minikube start --driver=docker --cpus=4 --memory=4400`

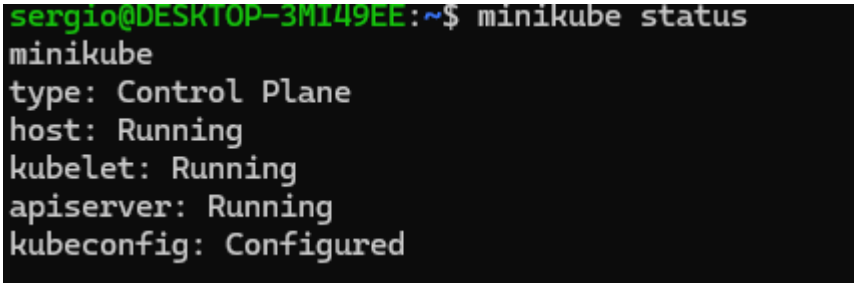


PD: Recordad que es necesario tener Docker Desktop abierto, si no os dará el siguiente error



2. Comprobar que el cluster se ha iniciado correctamente escribiendo

`minikube status`



3. Habilitar el complemento de Ingress

`minikube addons enable ingress`

Para verificar que se ha instalado correctamente hay que ejecutar el siguiente comando y comprobar que te aparecen los pods de nginx.

kubectl get pods -n ingress-nginx

```
sergio@DESKTOP-532IEM7:~$ kubectl get pods -n ingress-nginx
NAME                                        READY   STATUS      RESTARTS   AGE
ingress-nginx-admission-create-wx7j8        0/1     Completed   0          43s
ingress-nginx-admission-patch-km866         0/1     Completed   0          43s
ingress-nginx-controller-67c5cb88f-vk7pb    1/1     Running     0          43s
```

4. Para monitorizar el cluster desde el navegador se puede correr el siguiente servicio. Se ejecuta en segundo plano para no bloquear la terminal.

minikube dashboard &

```
sergio@DESKTOP-532IEM7:~$ 🤔  Verifying dashboard health ...
🚀  Launching proxy ...
🤔  Verifying proxy health ...
🎉  Opening http://127.0.0.1:42863/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in
your default browser...
👉  http://127.0.0.1:42863/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/
```

Para acceder al dashboard desde un navegador hay que acceder al enlace que aparece en la terminal. Este enlace varía cada vez que se ejecuta el comando.

Desplegar los servicios comunes:

El despliegue de los siguientes servicios (Keycloak, Minio, PostgreSQL y Vault) se realiza mediante charts de helm. Para desplegar dichos servicios hay que seguir los siguientes pasos:

1. Clonar el repositorio del deployer (https://github.com/INESData/inesdata-deployment)

git clone https://github.com/INESData/inesdata-deployment

cd inesdata-deployment/common

2. Modificar el fichero situado en la ruta **inesdata-deployment/common/values.yaml**. En este fichero hay que definir distintas contraseñas y añadir atributos que faltan para el correcto despliegue de los servicios. En este mismo repositorio se encuentra un fichero values.yaml relleno correctamente para tomarlo como referencia. Concretamente hay que:
    a. Los campos a cambiar tienen el valor “xxxxCHANGEMExxxx”
    b. Hay que añadir dos atributos en keycloak -> auth:
        i. adminUser: El atributo username no sirve para nada, se puede borrar y sustituir por este.
        ii. adminPassword: El atributo password no sirve para nada, se puede borrar y sustituir por este

```yaml
keycloak:
  auth:
    adminUser: xxxxCHANGEMExxxx
    adminPassword: xxxxCHANGEMExxxx
  postgresql:
```

c. El atributo postgresql -> auth -> password tiene que tener el mismo valor que el atributo keycloak -> externalDatabase -> password

```yaml
keycloak:
  externalDatabase:
    host: "{{ .Release.Name }}-postgresql"
    user: keycloak
    password: xxxxCHANGEMExxxx
    database: keycloak
    port: 5432
```

```yaml
postgresql:
  auth:
    postgresPassword: xxxxCHANGEMExxxx
    # We initially setup the database for keycloak
    username: keycloak
    password: xxxxCHANGEMExxxx
    database: keycloak
```

d. Hay que añadir las siguientes variables de entorno en keycloak => keycloakConfigCli:
   i. KEYCLOAK_USER: En value hay que poner el mismo valor definido en keycloak => auth => adminUser
   ii. KEYCLOAK_PASSWORD: En value hay que poner el mismo valor definido en keycloak => auth => adminPassword

```yaml
keycloak:
  keycloakConfigCli:
    enabled: true
    extraEnv:
      - name: KEYCLOAK_USER
        value: "xxxxCHANGEMExxxx"
      - name: KEYCLOAK_PASSWORD
        value: "xxxxCHANGEMExxxx"
    configuration:
```

e. Hay que especificar la imagen de keycloak, debido a que anteriormente se utilizaba la imagen de bitnami y este repositorio se ha movido a bitnamiLegacy. Para ello se ha añadido lo siguiente:

```yaml
keycloak:
  image:
      repository: bitnamilegacy/keycloak
      tag: 24.0.4-debian-12-r1
  auth:
```

f. Hay que especificar la imagen de postgresql, debido a que anteriormente se utilizaba la imagen de bitnami y este repositorio se ha movido a bitnamiLegacy. Para ello se ha añadido lo siguiente:

```yaml
postgresql:
  image:
      repository: bitnamilegacy/postgresql
      tag: 16.3.0-debian-12-r9
  auth:
```

g. Hay que especificar la imagen de keycloakConfigCli, debido a que anteriormente se utilizaba la imagen de bitnami y este repositorio se ha movido a bitnamiLegacy. Para ello se ha añadido lo siguiente:

```yaml
keycloak:
  keycloakConfigCli:
    image:
      repository: bitnamilegacy/keycloak-config-cli
      tag: 5.12.0-debian-12-r4
    enabled: true
```

3. Añadir los repos y dependencias de helm

`helm repo add minio https://charts.min.io/`

`helm repo add hashicorp https://helm.releases.hashicorp.com`

`helm dependency build`

4. Añadir en el fichero de hosts las siguientes urls para ser accesibles desde el navegador. En Windows (aunque se esté utilizando WSL) el fichero hosts se encuentra en C:\Windows\System32\drivers\etc\hosts. En Linux el fichero de host se encuentra en /etc/hosts:

127.0.0.1 keycloak.dev.ed.inesdata.upm
127.0.0.1 keycloak-admin.dev.ed.inesdata.upm
127.0.0.1 minio.dev.ed.inesdata.upm
127.0.0.1 console.minio-s3.dev.ed.inesdata.upm

```
# End of section
127.0.0.1   keycloak.dev.ed.inesdata.upm
127.0.0.1   keycloak-admin.dev.ed.inesdata.upm
127.0.0.1   minio.dev.ed.inesdata.upm
127.0.0.1   console.minio-s3.dev.ed.inesdata.upm
```

5. Desplegar

`helm install -f values.yaml -n common-srvs --create-namespace common-srvs .`

En caso de hacer falta borrar el despliegue hay que ejecutar el siguiente comando

`helm uninstall common-srvs -n common-srvs`

Si se quiere actualizar la configuración, ejecutar el siguiente comando

`helm upgrade -f values.yaml -n common-srvs --create-namespace common-srvs .`

6. Para comprobar que los servicios son accesibles hay que ejecutar el siguiente comando. Después de ejecutarlos los servicios son accesibles desde el navegador escribiendo las urls que se han dado de alta en el fichero de hosts

`minikube tunnel`

7. Hacer el unseal de Vault. Para ello hacemos los siguientes pasos:

IMPORTANTE: El unseal de Vault se recomienda realizarlo **desde** el directorio **inesdata-deployment/common**. Esto es debido a que al ejecutar los comandos, que se especifican a continuación, se genera un fichero ligado al despliegue de los servicios comunes.

Obtener el id del pod de Vault ejecutando

`kubectl get pods -n common-srvs`

```
sergio@DESKTOP-3MI49EE:~/inesdata-deployment/common$ kubectl get pods -n common-srvs
NAME                                READY   STATUS    RESTARTS   AGE
common-srvs-keycloak-0              1/1     Running   0          11m
common-srvs-minio-76b64c87c5-t65t5  1/1     Running   0          11m
common-srvs-postgresql-0            1/1     Running   0          11m
common-srvs-vault-0                 0/1     Running   0          11m
```

Ejecutar el siguiente comando para iniciar el proceso de unseal dentro del pod. Este comando **genera** un fichero llamado **init-keys-vault.json** en el mismo **directorio donde se ejecuta el comando**. En dicho fichero hay dos atributos importantes:

- unseal_keys_hex: Contraseña utilizada para realizar el unseal de Vault.
- root_token: Token maestro para acceder a Vault. Se utiliza para acceder a Vault.

`kubectl exec -it common-srvs-vault-0 -n common-srvs -- vault operator init -key-shares=1 -key-threshold=1 -format=json > init-keys-vault.json`


```
{
  "unseal_keys_b64": [
    "AZx65eprFLf9jxUmWiQpxwzk7Ax+i0MtZha7hU7rLAg="
  ],
  "unseal_keys_hex": [
    "019c7ae5ea6b14b7fd8f15265a2429c70ce4ec0c7e8b432d6616bb854eeb2c08"
  ],
  "unseal_shares": 1,
  "unseal_threshold": 1,
  "recovery_keys_b64": [],
  "recovery_keys_hex": [],
  "recovery_keys_shares": 0,
  "recovery_keys_threshold": 0,
  "root_token": "hvs.P7bsOtEogHrZWhEQdPlvOrCA"
}
```


Ejecutar el siguiente comando

`kubectl exec -it common-srvs-vault-0 -n common-srvs -- vault operator unseal <unseal_keys_hex>`

```
sergio@DESKTOP-532IEM7:~/inesdata-deployment/common$ kubectl exec -it common-srvs-vault-0 -n common-srvs -- vault ope
rator unseal 019c7ae5ea6b14b7fd8f15265a2429c70ce4ec0c7e8b432d6616bb854eeb2c08
Key             Value
---             -----
Seal Type       shamir
Initialized     true
Sealed          false
Total Shares    1
Threshold       1
Version         1.17.2
Build Date      2024-07-05T15:19:12Z
Storage Type    file
Cluster Name    vault-cluster-9ff1bd59
Cluster ID      87cefc17-a0a0-59e0-8f2b-8cfcc2adeb5c
HA Enabled      false
```

8. Comprobar que el pod de Vault ya está corriendo correctamente (desde dashboard revisando los logs o desde kubectl con el siguiente comando)

kubectl get pods -n common-srvs

```
sergio@DESKTOP-3MI49EE:~/inesdata-deployment/common$ kubectl get pods -n common-srvs
NAME                                READY   STATUS    RESTARTS   AGE
common-srvs-keycloak-0              1/1     Running   0          15m
common-srvs-minio-76b64c87c5-t65t5  1/1     Running   0          15m
common-srvs-postgresql-0            1/1     Running   0          15m
common-srvs-vault-0                 1/1     Running   0          15m
```

9. Acceder a Vault y realizar los siguientes pasos.

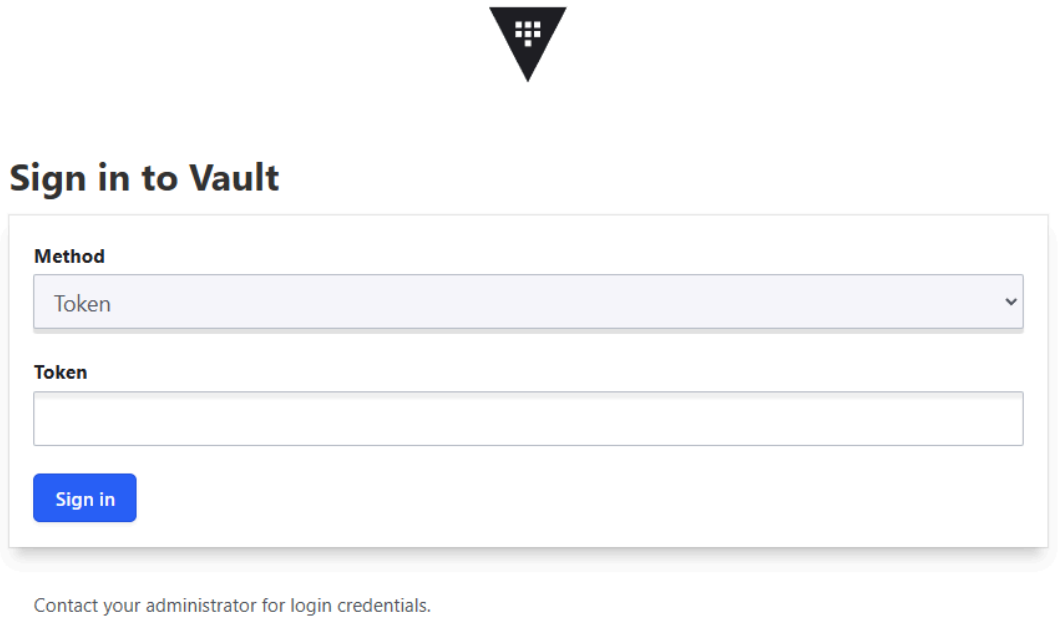
Levantar el tunel de IPs para poder acceder a Vault desde el navegador. En este caso no se levanta el proceso en segundo plano porque una vez terminado este paso (el paso 9) se mata este proceso para no poder acceder a Vault mediante el tunel

kubectl port-forward common-srvs-vault-0 -n common-srvs 8200:8200

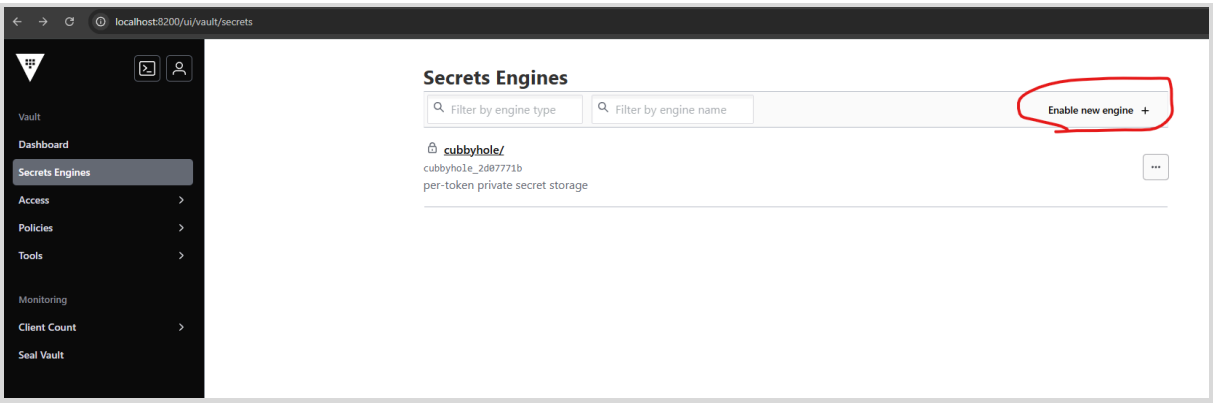
```
sergio@DESKTOP-532IEM7:~$ kubectl port-forward common-srvs-vault-0 -n common-srvs 8200:8200
Forwarding from 127.0.0.1:8200 -> 8200
Forwarding from [::1]:8200 -> 8200
```

En el navegador escribir la siguiente dirección http://localhost:8200/ui/vault/secrets

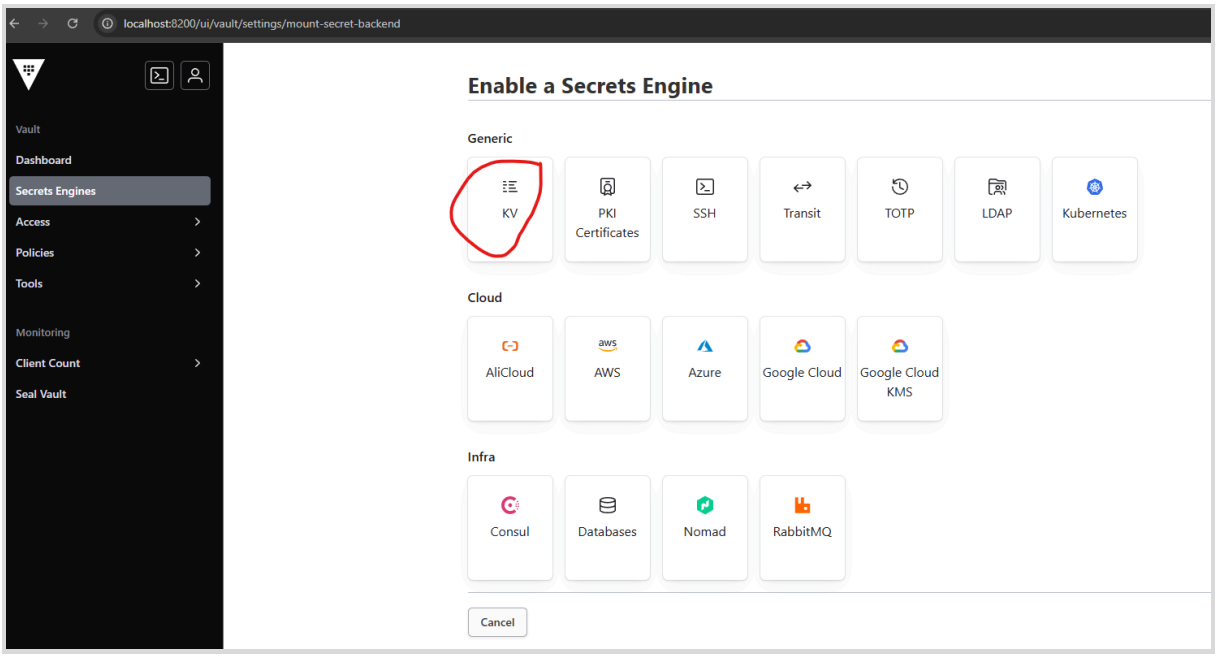
Cuando nos pida el token de acceso hay que escribir el token que se ha obtenido anteriormente en el paso 7 (**root_token** del fichero **init-keys-vault.json**).



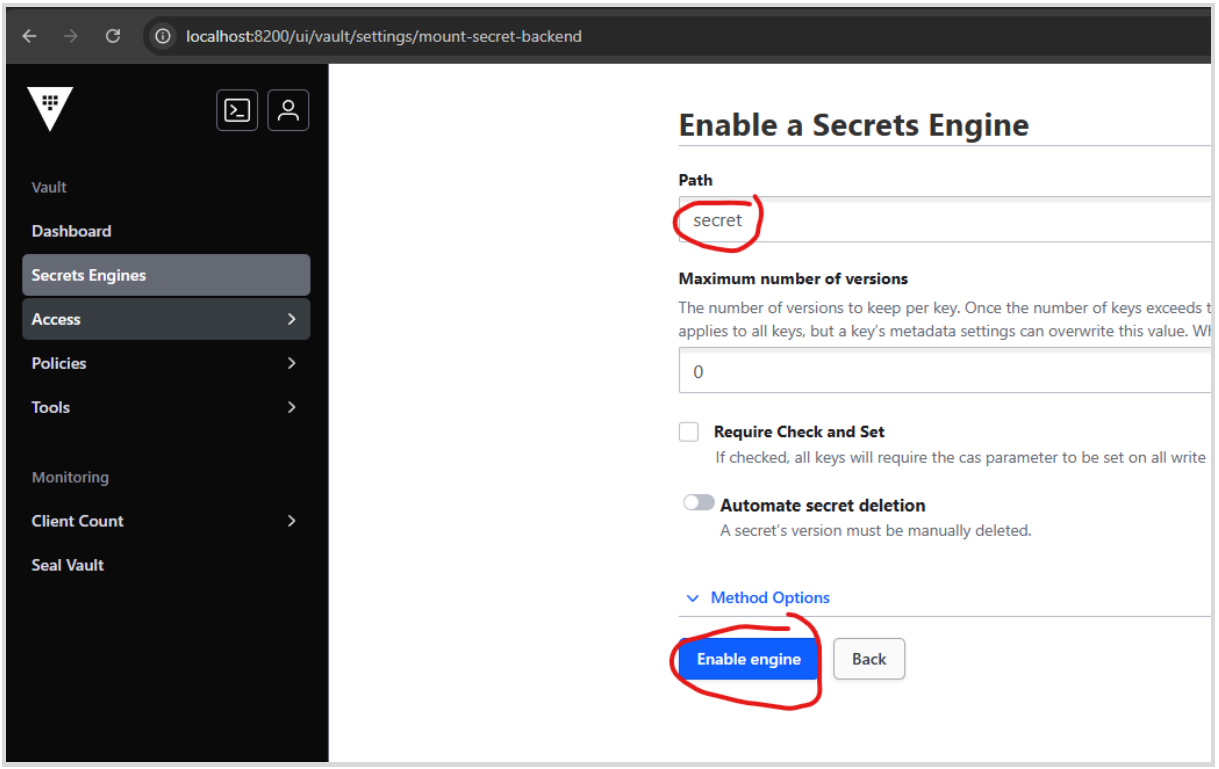
A continuación hay que hacer click en “Enable new engine”



Hay que hacer click en KV



En Path hay que escribir “secret” y luego pulsar en “Enable engine”



10. Matar el proceso que levanta el tunel escribiendo ctrl + c desde la terminal

```
sergio@DESKTOP-532IEM7:~$ kubectl port-forward common-srvs-vault-0 -n common-srvs 8200:8200
Forwarding from 127.0.0.1:8200 -> 8200
Forwarding from [::1]:8200 -> 8200
Handling connection for 8200
Handling connection for 8200
Handling connection for 8200
Handling connection for 8200
Handling connection for 8200
Handling connection for 8200
^Csergio@DESKTOP-532IEM7:~$
```

11. Reiniciar el pod (borrarlo porque se levanta solo)

```
kubectl delete -n common-srvs pod common-srvs-vault-0
```

12. Volver a ejecutar el comando para realizar el unseal de Vault. Cada vez que Vault se reinicia se vuelve a bloquear.

```
kubectl exec -it common-srvs-vault-0 -n common-srvs -- vault operator unseal <unseal_keys_hex>
```

Crear espacio de datos y desplegar conectores:

Configurar deployer:

1. Instalar python 3.10 siguiendo lo que pone en el siguiente enlace https://gist.github.com/rutcreate/c0041e842f858ceb455b748809763ddb
2. Comprobar la versión de python escribiendo lo siguiente

```
python3.10 –version
```

```
sergio@DESKTOP-532IEM7:~/inesdata-deployment$ python3.10 --version
Python 3.10.18
```

3. Situarnos en el directorio raíz del proyecto

```
cd inesdata-deployment
```

4. Crear environment. Este comando va a crear un nuevo directorio en la ruta inesdata-deployment/venv

```
python3.10 -m venv venv
```

5. Iniciar el environment

IMPORTANTE: Para que dicho comando funcione es necesario estar situado en el directorio inesdata-deployment

```
source venv/bin/activate
```

```
sergio@DESKTOP-532IEM7:~/inesdata-deployment$ source venv/bin/activate
(venv) sergio@DESKTOP-532IEM7:~/inesdata-deployment$
```

6. Comprobar el fichero requirements.txt. El contenido de dicho fichero debería de ser el siguiente. Esta comprobación se debe a que el contenido de este fichero en el repositorio está mal (GMV no incluyó la versión correcta de las librerías).

```
anyio==4.4.0
argon2-cffi==23.1.0
argon2-cffi-bindings==21.2.0
async-property==0.2.2
certifi==2024.6.2
cffi==1.16.0
charset-normalizer==3.3.2
click==8.1.7
```

cryptography==42.0.8
deprecation==2.1.0
exceptiongroup==1.2.1
h11==0.14.0
httpcore==1.0.5
httpx==0.27.0
hvac==2.3.0
idna==3.7
Jinja2==3.1.2
jwcrypto==1.5.6
MarkupSafe==3.0.2
minio==7.2.7
packaging==24.1
psycopg2-binary==2.9.9
pycparser==2.22
pycryptodome==3.20.0
PySocks==1.7.1
python-keycloak==4.1.0
requests==2.32.3
requests-toolbelt==1.0.0
sniffio==1.3.1
typing_extensions==4.12.2
urllib3==2.2.2

7. Instalar los requisitos

IMPORTANTE: Para que dicho comando funcione es necesario estar situado en el directorio inesdata-deployment

```
pip install -r requirements.txt
```

8. Modificar el fichero situado en inesdata-deployment/dataspace/step-1/templates/**registration-service-deployment.yaml** añadiendo lo siguiente. En este mismo repositorio se encuentra un fichero registration-service-deployment.yaml relleno correctamente para tomarlo como referencia

```
{{- with .Values.hostAliases }}
hostAliases:
{{- toYaml . | nindent 8 }}
{{- end }}
```

9. Modificar el fichero situado en inesdata-deployment/connector/templates/**connector-deployment.yaml** añadiendo lo siguiente. En este mismo repositorio se encuentra un fichero connector-deployment.yaml relleno correctamente para tomarlo como referencia

{{- with .Values.hostAliases }}
hostAliases:
{{- toYaml . | nindent 8 }}
{{- end }}

10. Modificar el fichero inesdata-deployment/**deployer.config** con la configuración que se encuentra en el fichero **deployer-dev.config** que se encuentra en este repositorio. De dicho fichero es necesario modificar los siguientes campos:
    - PG_PASSWORD: El valor del fichero inesdata-deployment/common/values.yaml especificado en postgresql => auth => postgresPassword
    - KC_USER: El valor del fichero inesdata-deployment/common/values.yaml especificado en keycloak => auth => adminUser

- KC_PASSWORD: El valor del fichero inesdata-deployment/common/values.yaml especificado en keycloak => auth => adminPassword
- VT_TOKEN: Token de Vault. Está almacenado en el fichero init-keys-vault.json en el atributo root_token. Este fichero se ha generado en la sección “Desplegar los servicios comunes”.

```
ENVIRONMENT=DEV
PG_HOST=localhost
PG_USER=postgres
PG_PASSWORD=
KC_URL=http://keycloak-admin.dev.ed.inesdata.upm
KC_USER=
KC_PASSWORD=
KC_INTERNAL_URL=http://keycloak.dev.ed.inesdata.upm
VT_URL=http://localhost:8200
VT_TOKEN=
DATABASE_HOSTNAME=common-srvs-postgresql.common-srvs.svc
KEYCLOAK_HOSTNAME=keycloak.dev.ed.inesdata.upm
MINIO_HOSTNAME=minio.dev.ed.inesdata.upm
VAULT_URL=http://common-srvs-vault.common-srvs.svc:8200
```

11. Para crear un espacio de datos y conector hay que seguir los pasos que se especifican en las secciones de más abajo. Pero antes es necesario abrir los siguientes túneles. Los túneles se ejecutan en segundo plano para no bloquear la terminal

```
kubectl port-forward common-srvs-postgresql-0 -n common-srvs 5432:5432 &
```

```
kubectl port-forward common-srvs-vault-0 -n common-srvs 8200:8200 &
```

Crear un nuevo espacio de datos:

Antes de crear el espacio de datos hay que decidir un nombre único para dicho espacio de datos. Por convención el nombre del espacio de datos no puede contener ni espacios, ni -, ni _. Por ejemplo, un nombre válido puede ser test3. Destacar que no puede existir ya un espacio de datos con el nombre previamente seleccionado. En esta sección el nombre del espacio de datos se va a tomar como <**NAME_DS**>.

Antes de crear el espacio de datos hay que también decidir un Namespace para el espacio de datos. Por convención dicho Namespace debe ser igual al nombre del espacio de datos. Por ejemplo, para el espacio de datos test3, su Namespace sería también test3. En esta sección el Namespacio del espacio de datos se va a tomar como <**NAMESPACE_DS**>, cuyo valor va a ser igual a <**NAME_DS**>.

EJEMPLO: Debajo de cada comando va a aparecer un mensaje de ejemplo. Para dichos ejemplos se va a tomar que:

- <**NAME_DS**> = demo

- <**NAMESPACE_DS**> = <**NAME_DS**> = demo

1. Abrir los siguientes túneles. Los túneles se ejecutan en segundo plano para no bloquear la terminal

`kubectl port-forward common-srvs-postgresql-0 -n common-srvs 5432:5432 &`

`kubectl port-forward common-srvs-vault-0 -n common-srvs 8200:8200 &`

2. Situarse en el directorio inesdata-deployment y tener activado el entorno virtual de python (el entorno virtual está activo si dentro de la terminal de WSL aparece (venv) a la izquierda).

`cd inesdata-deployment`

`source venv/bin/activate`

3. Crear el espacio de datos ejecutando el siguiente comando.

`python deployer.py dataspace create <`**NAME_DS**`>`

EJEMPLO: `python deployer.py dataspace create demo`

Si por lo que sea el comando falla, antes de volver a ejecutar el comando es necesario ejecutar el comando delete con los mismos argumentos.

`python deployer.py dataspace delete <`**NAME_DS**`>`

EJEMPLO: `python deployer.py dataspace delete demo`

4. Renombrar el fichero dataspace/step-1/values.yaml.{**NAME_DS}** a dataspace/step-1/values-{**NAME_DS}**.yaml
5. Renombrar el fichero dataspace/step-2/values.yaml.{**NAME_DS}** a dataspace/step-2/values-{**NAME_DS}**.yaml
6. Comprobar la IP de Minikube ejecutando el siguiente comando. La IP obtenida se va a llamar a partir de ahora {**IP_MINIKUBE**}

`minikube ip`

```
(venv) sergio@DESKTOP-532IEM7:~/inesdata-deployment$ minikube ip
192.168.49.2
(venv) sergio@DESKTOP-532IEM7:~/inesdata-deployment$
```

7. Modificar el fichero dataspace/step-1/values-{**NAME_DS}**.yaml añadiendo al final las urls de los servicios comunes. Concretamente hay que añadir lo siguiente

IMPORTANTE: Si la **tabulación** no queda **igual que en la imagen de abajo** el despliegue se va a realizar de forma incorrecta. No va a saltar ningún mensaje de error pero el programa no va a funcionar correctamente.

hostAliases:
- ip: "{**IP_MINIKUBE**}"
  hostnames:
  - "keycloak.dev.ed.inesdata.upm"
  - "keycloak-admin.dev.ed.inesdata.upm"
  - "minio.dev.ed.inesdata.upm"
  - "console.minio-s3.dev.ed.inesdata.upm"

```
ingress:
  registration:
    hostname:
      registration-service-demo.dev.ds.inesdata.upm
hostAliases:
- ip: "192.168.49.2"
  hostnames:
  - "keycloak.dev.ed.inesdata.upm"
  - "keycloak-admin.dev.ed.inesdata.upm"
  - "minio.dev.ed.inesdata.upm"
  - "console.minio-s3.dev.ed.inesdata.upm"
```

8. Desplegar el registration service utilizando helm. Para ello ejecutamos los siguientes comandos:

cd dataspace/step-1

helm install -f values-{**NAME_DS**}.yaml -n <**NAMESPACE_DS**> --create-namespace <**NAME_DS**>-dataspace-s1 .

EJEMPLO: helm install -f values-demo.yaml -n demo --create-namespace demo-dataspace-s1 .

Si se quiere desinstalar, ejecutar el siguiente comando

helm uninstall <**NAME_DS**>-dataspace-s1 -n <**NAMESPACE_DS**>

EJEMPLO: helm uninstall demo-dataspace-s1 -n demo

Si se quiere actualizar la configuración, ejecutar el siguiente comando

cd dataspace/step-1

helm upgrade-f values-{**NAME_DS**}.yaml -n <**NAMESPACE_DS**> --create-namespace <**NAME_DS**>-dataspace-s1 .

EJEMPLO: helm upgrade -f values-demo.yaml -n demo --create-namespace demo-dataspace-s1 .

9. Comprobar que el pod se ha levantado correctamente. Para ello hay que ejecutar el siguiente comando para poder identificar el ID del pod que se acaba de desplegar

kubectl get pods -n <**NAMESPACE_DS**>

EJEMPLO: kubectl get pods -n demo

Una vez que el pod aparezca que está desplegado, identificado el ID del pod, y que haya pasado un tiempo prudencial (1 minuto más o menos) hay que comprobar los logs del conector para comprobar realmente que el registration service se ha desplegado de manera correcta. El identificador de dicho pod se va a llamar <**POD_RS**>

kubectl logs <**POD_RS**> -n <**NAMESPACE_DS**>

EJEMPLO: kubectl logs demo-registration-service-7668f9747c-xrkfq -n demo

10. Añadir en el fichero de hosts las url del registration service para ser accesibles desde el navegador. En Windows (aunque se esté utilizando WSL) el fichero hosts se encuentra en C:\Windows\System32\drivers\etc\hosts. En Linux el fichero de host se encuentra en /etc/hosts.

```
# End of section
127.0.0.1   keycloak.dev.ed.inesdata.upm
127.0.0.1   keycloak-admin.dev.ed.inesdata.upm
127.0.0.1   minio.dev.ed.inesdata.upm
127.0.0.1   console.minio-s3.dev.ed.inesdata.upm
127.0.0.1   registration-service-demo.dev.ds.inesdata.upm
```

EJEMPLO: La url del registration service sería registration-service-demo.dev.ds.inesdata.upm, por tanto en el fichero de hosts habría que añadir 127.0.0.1 registration-service-demo.dev.ds.inesdata.upm

11. Antes de desplegar el portal público hay que crear un primer conector. El despliegue del portal público se explica en la sección “**Crear el portal público**”. Esto es necesario porque para desplegar el portal público hay que crear un conector especial llamado “conector promotor”. Lo único que tiene de especial este conector es que el portal público va a hacer queries al catálogo de dicho conector para poder mostrar información desde la web. Para crear un conector mirar la sección “**Crear un nuevo conector**”.

Crear un nuevo conector:

Antes de crear un conector hay que decidir a qué espacio de datos va a pertenecer dicho conector (el espacio de datos debe de estar creado previamente). El nombre del espacio de datos al que va a pertenecer el conector se va a tomar como {**NAME_DS**}. Posteriormente, hay que decidir un nombre único para dicho conector. Destacar que el nombre del conector debe ser único para todos los espacios de datos. Si ya existe un conector con dicho nombre

(independientemente del espacio de datos al que pertenezca el conector), va a salir un error al ejecutar el programa. En esta sección el nombre del conector se va a tomar como {**NAME_CONN**}. Para que el nombre del conector sea único, por convención el nombre del conector va a ser conn-{**NAME**}-{**NAME_DS**} donde:

- {**NAME_DS**} es el nombre del espacio de datos al que va a pertenecer dicho conector
- {**NAME**} es un nombre único dentro del espacio de datos al que va a pertenecer el conector. Por convención este nombre único no puede contener ni espacios, ni -, ni _. Por ejemplo, un nombre válido puede ser oeg . Por ejemplo si dentro del espacio de datos test3 ya existe un conector llamado conn-oeg-test3, no puedo elegir oeg cómo {**NAME**}

EJEMPLO: Debajo de cada comando va a aparecer un mensaje de ejemplo. Para dichos ejemplos se va a tomar que:

- <**NAME_DS**> = demo
- <**NAMESPACE_DS**> = <**NAME_DS**> = demo
- {**NAME_CONN**} = conn-oeg-demo

1. Abrir los siguientes túneles. Los túneles se ejecutan en segundo plano para no bloquear la terminal

```
kubectl port-forward common-srvs-postgresql-0 -n common-srvs 5432:5432 &

kubectl port-forward common-srvs-vault-0 -n common-srvs 8200:8200 &
```

2. Situarse en el directorio inesdata-deployment y tener activado el entorno virtual de python (el entorno virtual está activo si dentro de la terminal de WSL aparece (venv) a la izquierda).

```
cd inesdata-deployment

source venv/bin/activate
```

3. Hacer ejecutable el script inesdata-deployment\scripts\generate-cert.sh hay que hacerlo ejecutable ejecutando los siguientes comandos

```
cd scripts

chmod a+x generate-cert.sh

cd ..
```

4. Ejecutar el siguiente comando para crear el conector.

python deployer.py connector create <**NAME_CONN**> <**NAME_DS**>

EJEMPLO: python deployer.py connector create conn-oeg-demo demo

Si por lo que sea el comando falla, antes de volver a ejecutar el comando es necesario ejecutar el comando delete con los mismos argumentos.

python deployer.py connector delete <**NAME_CONN**> <**NAME_DS**>

EJEMPLO: python deployer.py connector delete conn-oeg-demo demo

5. Renombrar el fichero connector/values.yaml.{**NAME_CONN**} a connector/values-{**NAME_CONN**}.yaml . Además, en dicho fichero es necesario modificar los siguientes valores:
    a. ingress hostname: Dicho valor se genera mal y hay que modificarlo para que apunte a la url del conector. En este valor aparece dos veces seguidas el nombre del espacio de datos y es necesario modificarlo para que solo aparezca una vez el nombre del espacio de datos. Por ejemplo, si aparece conn-oeg-test3-test3.ds.inesdata-project.eu hay que modificarlo para que aparezca conn-oeg-test3.ds.inesdata-project.eu

Ejemplo del antes

```yaml
ingress:
  hostname: conn-oeg-demo-demo.dev.ds.inesdata.upm
  protocol: http
```

Ejemplo del despues

```yaml
ingress:
  hostname: conn-oeg-demo.dev.ds.inesdata.upm
  protocol: http
```

    b. keyclock hostname: Si dicho valor se genera vacío, es necesario copiar y pegar aquí el valor que aparece en keyclock external.

Ejemplo del antes

```yaml
keycloak:
  # comsrv prefix comes from the Helm release of the common services
  hostname:
  external: keycloak.dev.ed.inesdata.upm
  protocol: http
```

Ejemplo del después

```yaml
keycloak:
  # comsrv prefix comes from the Helm release of the common services
  hostname: keycloak.dev.ed.inesdata.upm
  external: keycloak.dev.ed.inesdata.upm
  protocol: http
```

6. Creación de bucket, política y usuario de MinIO. A partir de ahora {**ENVIRONMENT**} hace referencia a la variable con el mismo nombre que se encuentra en el fichero de configuración. Lo normal es que sea PRO. Para ello ejecutar los siguientes comandos:

Comprobar el ID del Pod donde se encuentra desplegado Minio. Para ello hay que ejecutar el siguiente comando. El ID de Minio es el que empieza por comsrvs-minio, y se va a llamar a partir de ahora {**POD_MINIO**}

kubectl get pods -n common-srvs

cat deployments/{**ENVIRONMENT**}/{**NAME_DS**}/policy-{**NAME_DS**}-{**NAME_CONN**}.json | kubectl exec -i -n common-srvs {**POD_MINIO**} -- sh -c 'cat > /tmp/policy-{**NAME_DS**}-{**NAME_CONN**}.json'

EJEMPLO: cat deployments/DEV/demo/policy-demo-conn-oeg-demo.json | kubectl exec -i -n common-srvs comsrvs-minio-84fb6f7f7-g7rqs -- sh -c 'cat > /tmp/policy-demo-conn-oeg-demo.json'

kubectl exec -it {**POD_MINIO**} -n common-srvs -- /bin/bash

EJEMPLO: kubectl exec -it comsrvs-minio-84fb6f7f7-g7rqs -n common-srvs -- /bin/bash

Necesitamos la contraseña del administrador de Minio. Esta contraseña se encuentra en inesdata-depoyment/common/values.yaml en minio => rootPassword. Esta contraseña se va a llamar <**MINIO_ADMIN_PASSWORD**>

```
minio:
  mode: standalone
  rootUser: admin
  rootPassword: edgarData1234
```

mc alias set minio http://127.0.0.1:9000 admin <**MINIO_ADMIN_PASSWORD**>

EJEMPLO: mc alias set minio http://127.0.0.1:9000 admin edgarData1234

mc mb minio/<**NAME_DS**>-<**NAME_CONN**>

EJEMPLO: mc mb minio/demo-conn-oeg-demo

Ahora se crean las credenciales de Minio para el conector. Dichas credenciales son <**USER_MINIO**>, <**USER_MINIO_PASSWORD**>, <**ACCESS_KEY**> y <**SECRET_KEY**> y se encuentran en el fichero deployments/{**ENVIRONMENT**}/{**NAME_DS**}/credentials-connector-{**NAME_CONN**}.json

```
"minio": {
    "access_key": "YORPIE5TFdsy9gPA",
    "secret_key": "AhAtA9i8nTqZaa4VCwbt0mFkM0O9BYz49dxwV0Uv",
    "user": "conn-oeg-demo",
    "passwd": "pqUphIBWDLCz68G7"
}
```

mc admin user add minio <**USER_MINIO**> <**USER_MINIO_PASSWORD**>

EJEMPLO: mc admin user add minio conn-oeg-demo pqUphIBWDLCz68G7

mc admin user svcacct add minio <**USER_MINIO**> --access-key <**ACCESS_KEY**> --secret-key <**SECRET_KEY**>

EJEMPLO: mc admin user svcacct add minio conn-oeg-demo --access-key YORPIE5TFdsy9gPA --secret-key AhAtA9i8nTqZaa4VCwbt0mFkM0O9BYz49dxwV0Uv

mc admin policy create minio <**USER_MINIO**>-policy /tmp/policy-{**NAME_DS**}-{**NAME_CONN**}.json

EJEMPLO: mc admin policy create minio conn-oeg-demo-policy /tmp/policy-demo-conn-oeg-demo.json

mc admin policy attach minio <**USER_MINIO**>-policy -user=<**USER_MINIO**>

EJEMPLO: mc admin policy attach minio conn-oeg-demo-policy -user=conn-oeg-demo

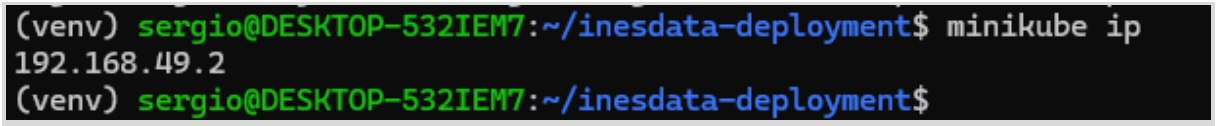
rm /tmp/policy-{**NAME_DS**}-{**NAME_CONN**}.json

EJEMPLO:rm /tmp/policy-demo-conn-oeg-demo.json

exit

7. Comprobar la IP de Minikube ejecutando el siguiente comando. La IP obtenida se va a llamar a partir de ahora {**IP_MINIKUBE**}

minikube ip



8. Modificar el fichero connector/values-{**NAME_CONN**}.yaml añadiendo al final las urls de los servicios comunes. Concretamente hay que añadir lo siguiente

IMPORTANTE: Si la **tabulación** no queda **igual que en la imagen de abajo** el despliegue se va a realizar de forma incorrecta. No va a saltar ningún mensaje de error pero el programa no va a funcionar correctamente.

```
hostAliases:
- ip: "{IP_MINIKUBE}"
  hostnames:
  - "keycloak.dev.ed.inesdata.upm"
  - "keycloak-admin.dev.ed.inesdata.upm"
  - "minio.dev.ed.inesdata.upm"
  - "console.minio-s3.dev.ed.inesdata.upm"
  - "registration-service-{NAME_DS}.dev.ds.inesdata.upm"
```

```
  vault:
    url: http://common-srvs-vault.common-srvs.svc:8200
    token: hvs.CAESIKeJqKlOhGgFRmoOCVa0vB9z-nlAERJEVrHHw1m
    path: demo/conn-oeg-demo/
hostAliases:
- ip: "192.168.49.2"
  hostnames:
  - "keycloak.dev.ed.inesdata.upm"
  - "keycloak-admin.dev.ed.inesdata.upm"
  - "minio.dev.ed.inesdata.upm"
  - "console.minio-s3.dev.ed.inesdata.upm"
  - "registration-service-demo.dev.ds.inesdata.upm"
```

9. Si ya hay conectores desplegados es necesario modificar el fichero connector/values-{**NAME_CONN**}.yaml añadiendo al final las urls conectores que ya se encuentran desplegados.

```
  vault:
    url: http://common-srvs-vault.common-srvs.svc:8200
    token: hvs.CAESIKeJqKlOhGgFRmoOCVa0vB9z-nlAERJEVrHH
    path: demo/conn-oeg-demo/
hostAliases:
- ip: "192.168.49.2"
  hostnames:
  - "keycloak.dev.ed.inesdata.upm"
  - "keycloak-admin.dev.ed.inesdata.upm"
  - "minio.dev.ed.inesdata.upm"
  - "console.minio-s3.dev.ed.inesdata.upm"
  - "registration-service-demo.dev.ds.inesdata.upm"
  - "conn-prueba-demo.dev.ds.inesdata.upm"
```

10. Añadir en el fichero de hosts las url del conector para ser accesibles desde el navegador. En Windows (aunque se esté utilizando WSL) el fichero hosts se encuentra en C:\Windows\System32\drivers\etc\hosts. En Linux el fichero de host se encuentra en /etc/hosts.

```
# End of section
127.0.0.1   keycloak.dev.ed.inesdata.upm
127.0.0.1   keycloak-admin.dev.ed.inesdata.upm
127.0.0.1   minio.dev.ed.inesdata.upm
127.0.0.1   console.minio-s3.dev.ed.inesdata.upm
127.0.0.1   registration-service-demo.dev.ds.inesdata.upm
127.0.0.1   conn-oeg-demo.dev.ds.inesdata.upm
```

EJEMPLO: La url del conector sería conn-oeg-demo.dev.ds.inesdata.upm, por tanto en el fichero de hosts habría que añadir 127.0.0.1 conn-oeg-demo.dev.ds.inesdata.upm

11. Es necesario añadir la url del conector en los ficheros values.yaml del resto de conectores desplegados en el mismo espacio de datos. Dichos ficheros se encuentran en el directorio inesdata-deployment/connector. Es necesario añadir en el hostname la url del nuevo conector a desplegar como se puede ver en la imagen de abajo, y posteriormente ejecutar el siguiente comando:

helm upgrade -f values-{**NAME_CONN**}.yaml -n <**NAMESPACE_DS**> <**NAME_CONN**>-<**NAME_DS**> .

EJEMPLO: helm upgrade -f values-conn-prueba-demo.yaml -n demo conn-prueba-demo-demo .

```
  vault:
    url: http://common-srvs-vault.common-srvs.svc:8200
    token: hvs.CAESIKuYCt47ZaNwFltt3qBspCfxiVHe3YIUw3a1zui
    path: demo/conn-prueba-demo/
hostAliases:
- ip: "192.168.49.2"
  hostnames:
  - "keycloak.dev.ed.inesdata.upm"
  - "keycloak-admin.dev.ed.inesdata.upm"
  - "minio.dev.ed.inesdata.upm"
  - "console.minio-s3.dev.ed.inesdata.upm"
  - "registration-service-demo.dev.ds.inesdata.upm"
  - "conn-oeg-demo.dev.ds.inesdata.upm"
```

12. Es necesario modificar el fichero dataspace/step-1/values-{**NAME_DS}**.yaml, donde **NAME_DS** tiene que ser el espacio de datos donde se está desplegando el conector. Es necesario añadir en el hostname la url del nuevo conector a desplegar como se puede ver en la imagen de abajo, y posteriormente ejecutar el siguiente comando:

cd dataspace/step-1

helm upgrade-f values-{**NAME_DS}**.yaml -n <**NAMESPACE_DS**> --create-namespace <**NAME_DS**>-dataspace-s1 .

EJEMPLO: helm upgrade-f values-demo.yaml -n demo --create-namespace demo-dataspace-s1 .

```
hostAliases:
- ip: "192.168.49.2"
  hostnames:
  - "keycloak.dev.ed.inesdata.upm"
  - "keycloak-admin.dev.ed.inesdata.upm"
  - "minio.dev.ed.inesdata.upm"
  - "console.minio-s3.dev.ed.inesdata.upm"
  - "conn-oeg-demo.dev.ds.inesdata.upm"
  - "conn-prueba-demo.dev.ds.inesdata.upm"
```

13. Desplegar el conector utilizando helm. Para ello ejecutamos los siguientes comandos:

cd connector

helm install -f values-{**NAME_CONN**}.yaml -n <**NAMESPACE_DS**> <**NAME_CONN**>-<**NAME_DS**> .

EJEMPLO: helm install -f values-conn-oeg-demo.yaml -n demo conn-oeg-demo-demo .

Si se quiere desinstalar, ejecutar el siguiente comando

helm uninstall <**NAME_CONN**>-<**NAME_DS**> -n <**NAMESPACE_DS**>

EJEMPLO: helm uninstall conn-oeg-demo-demo -n demo

Si se quiere actualizar la configuración, ejecutar el siguiente comando

cd connector

helm upgrade -f values-{**NAME_CONN**}.yaml -n <**NAMESPACE_DS**> <**NAME_CONN**>-<**NAME_DS**> .

EJEMPLO: helm upgrade -f values-conn-oeg-demo.yaml -n demo conn-oeg-demo-demo .

14. Comprobar que el pod se ha levantado correctamente. Para ello hay que ejecutar el siguiente comando para poder identificar el ID del pod que se acaba de desplegar

kubectl get pods -n <**NAMESPACE_DS**>

EJEMPLO: kubectl get pods -n demo

Una vez que el pod aparezca que está desplegado, identificado el ID del pod, y que haya pasado un tiempo prudencial (1 minuto más o menos) hay que comprobar los logs del conector para comprobar realmente que el conector se ha desplegado de manera correcta. El identificador de dicho pod se va a llamar <**POD_CONN**>

kubectl logs <**POD_CONN**> -n <**NAMESPACE_DS**>

EJEMPLO: kubectl logs conn-oeg-demo-5f7cbdcf86-h28cg -n demo

15. Finalmente para comprobar que el conector de verdad está levantado hay que acceder a la url del conector poniendo al final "**/inesdata-connector-interface/**". Por ejemplo:

http://conn-oeg-demo.dev.ds.inesdata.upm/inesdata-connector-interface/

IMPORTANTE: Recordad que para poder acceder a los servicios expuestos en Minikube es necesario ejecutar antes el siguiente comando

minikube tunnel

Crear el portal público:

TBD