



ANEXO I

Manual de Instalación de
los servicios de OpenPaDi
en un escenario
simplificado

Contenido

1.	Máquinas Virtuales utilizadas:.....	2
2.	Instalación y configuración de PostgreSQL.....	4
2.1.	Instalación de PostgreSQL	4
2.2.	Configuración de PostgreSQL para que no escuche solamente en local	4
2.3.	Creación de usuarios en PostgreSQL	5
3.	Instalación y configuración de MinIO.....	7
3.1.	Instalación de MinIO	7
3.2.	Creación de directorio para los datos de MinIO	8
3.3.	Configuración de las credenciales de acceso raíz de MinIO	8
3.4.	Creación del Servicio Systemd para MinIO	9
3.5.	Acceso a la Consola Web y creación del <i>bucket</i> OpenPaDi.....	11
4.	Instalación y configuración del clúster K3S	13
4.1.	Instalación de Docker en ambas máquinas	13
4.2.	Instalación del Nodo Máster de K3s en <i>op-web-1</i>	14
4.3.	Instalación de Node.js y npm en <i>op-web-1</i>	16
4.4.	Preparación, construcción y Dockerización del Frontend Svelte en <i>op-web-1</i>	16
4.5.	Dockerización de la aplicación <i>frontend</i>	20
4.6.	Construcción y Dockerización del <i>Backend API</i> en <i>op-api-1</i>	21
4.7.	Dockerización del <i>Backend API</i>	22
4.8.	Despliegue de las Aplicaciones OpenPaDi.....	23
4.9.	Instalación y Configuración de Cert-Manager.....	27
4.10.	Creación del Certificado TLS	29
4.11.	Configuración del <i>Ingress</i>	30
4.12.	Despliegue y Configuración de Keycloak	32
4.13.	Verificación de la integración de Keycloak con el <i>frontend</i>	42
5.	Validación final de la Integración completa del <i>frontend</i> con Keycloak y la API	45

Este manual tiene como objetivo ayudar en la instalación de OpenPaDi en las diferentes estancias de una infraestructura final.

Para simplificarlo se utilizarán 4 MVs en Virtual Box unidas en la misma red interna, representando cada una de ellas las máquinas principales con los siguientes roles:

- Ingress Controller – Frontend
- Backend
- BD relacional
- Almacenamiento de Objetos

1. Máquinas Virtuales utilizadas:



1. OP-web-1 (192.168.1.10): Ejecutará el nodo *master* de K3s y los contenedores del *frontend*, Keycloak y Traefik.
2. OP-API-1 (192.168.1.12): Ejecutará el nodo *worker* de K3s y el contenedor del *Backend API*.
3. OP-db-primary (192.168.1.13): Alojará PostgreSQL.
4. OP-storage-1 (192.168.1.14): Alojará el servicio de MinIO.

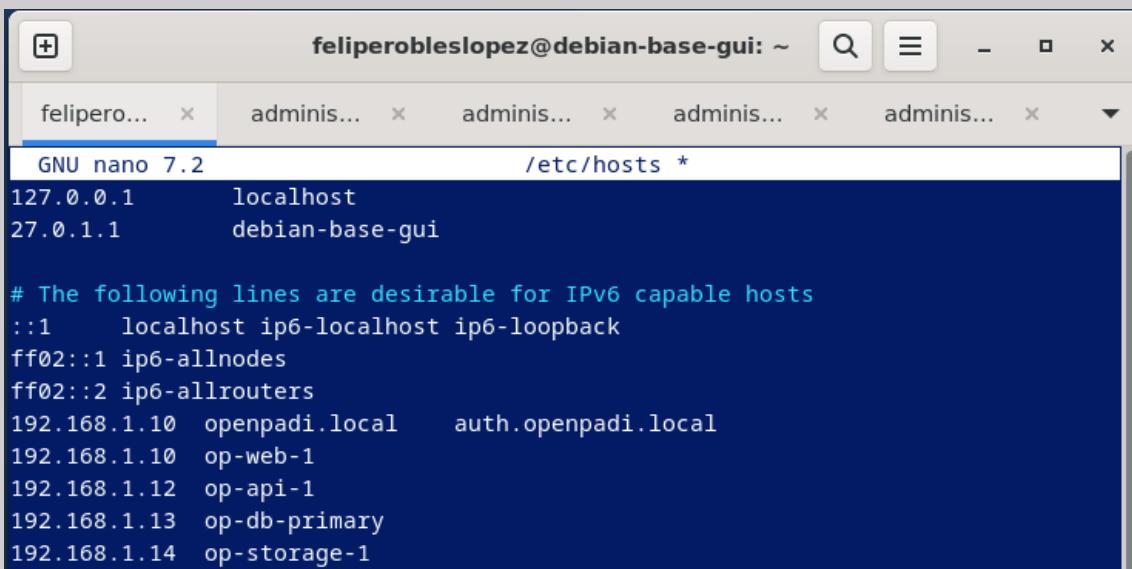
Todas ellas han sido configuradas de la siguiente manera:

- Recursos:
 - 2 CPU
 - 2 RAM
 - 20 GB Disco Duro
- Red:

- Adaptador 1: NAT
- Adaptador 2: Red interna
- SO: Ubuntu 22.04

Igualmente, es recomendable tener un cliente gráfico desde el que administrar las máquinas anteriores, conectándose a ellas por SSH. Esto es opcional, pero hará todo mucho más cómodo.

En este caso se utilizó una máquina con Debian 12 instalado. Como el propósito de este manual es simplemente instalar los diferentes servicios, no se va a configurar un DNS, por lo que nos valdrá con utilizar el fichero /etc/hosts de este cliente; es importante que resuelva los dominios *openapdi.local* y *auth.openapdi.local* con la IP del servidor donde irá alojado Traefik (OP-web1 – 192.168.1.10) y, opcionalmente, los *hostnames* de cada máquina:



```

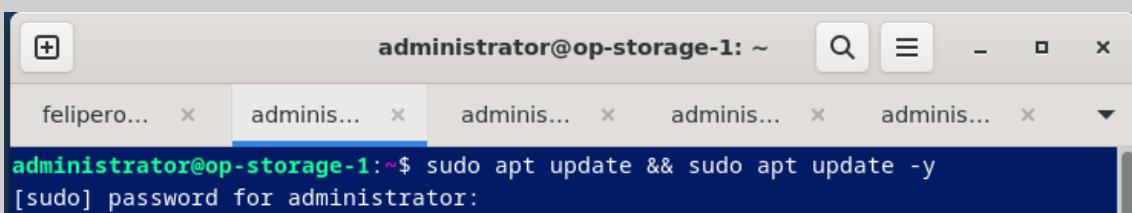
feliperobleslopez@debian-base-gui: ~
feliper... x adminis... x adminis... x adminis... x adminis... x
+-----+
GNU nano 7.2                               /etc/hosts *
127.0.0.1      localhost
27.0.1.1      debian-base-gui

# The following lines are desirable for IPv6 capable hosts
::1      localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
192.168.1.10  openapdi.local    auth.openapdi.local
192.168.1.10  op-web-1
192.168.1.12  op-api-1
192.168.1.13  op-db-primary
192.168.1.14  op-storage-1

```

Se recomienda igualmente actualizar cada una de las máquinas:

```
sudo apt update && sudo apt upgrade -y
```



```

administrator@op-storage-1: ~
feliper... x adminis... x adminis... x adminis... x adminis... x
administrator@op-storage-1:~$ sudo apt update && sudo apt update -y
[sudo] password for administrator:

```

2. Instalación y configuración de PostgreSQL



En esta sección se indican los pasos para instalar y configurar el servicio de base de datos relacional en *op-db-primary* (192.168.1.13).

2.1. Instalación de PostgreSQL

Lo primero será instalar PostgreSQL en esta máquina:

```
sudo apt install -y postgresql postgresql-contrib
```

Una vez instalado, comprobaremos que el servicio está corriendo sin problemas:

```
sudo systemctl status postgresql
```

A screenshot of a terminal window titled "administrator@op-db-primary: ~". The window shows several tabs open, with the current tab containing the command "sudo systemctl status postgresql". The output of the command is displayed:

```
administrator@op-db-primary:~$ sudo systemctl status postgresql
[sudo] password for administrator:
● postgresql.service - PostgreSQL RDBMS
  Loaded: loaded (/usr/lib/systemd/system/postgresql.service; enabled; presen>
  Active: active (exited) since Thu 2025-06-05 10:00:25 UTC; 22min ago
    Main PID: 22222 (code=exited, status=0/SUCCESS)
      CPU: 1ms

jun 05 10:00:25 op-db-primary systemd[1]: Starting postgresql.service - Postgre>
jun 05 10:00:25 op-db-primary systemd[1]: Finished postgresql.service - Postgre>
lines 1-8/8 (END)
```

2.2. Configuración de PostgreSQL para que no escuche solamente en local

Por defecto, PostgreSQL solo escucha conexiones locales. Vamos a modificar su fichero de configuración para que escuche en todas las interfaces:

```
sudo nano /etc/postgresql/16/main/postgresql.conf1
```

¹ Para este escenario se está utilizando la versión 16 de PostgreSQL; el número del directorio deberá ajustarse a la versión instalada.

```
#-----  
# CONNECTIONS AND AUTHENTICATION  
#-----  
  
# - Connection Settings -  
  
listen_addresses = '*'
```

Y añadiremos las siguientes líneas para permitir conexiones desde la red interna (192.168.1.0/24) y desde la red de pods de K3s (por defecto 10.42.0.0/16) al final del archivo *pg_hba.conf*:

```
sudo nano /etc/postgresql/16/main/pg_hba.conf
```

#	TYPE	DATABASE	USER	ADDRESS	METHOD
host	opadi_db	opadi_user	192.168.1.0/24	md5	
host	keycloak_db	keycloak_user	10.42.0.0/16	scram-sha-256	
host	keycloak_db	keycloak_user	192.168.1.0/24	scram-sha-256	

Una vez guardados ambos ficheros con la nueva configuración, reiniciaremos PostgreSQL:

```
sudo systemctl restart postgresql
```

```
administrator@op-db-primary:~$ sudo systemctl restart postgresql  
administrator@op-db-primary:~$ sudo systemctl status postgresql  
● postgresql.service - PostgreSQL RDBMS  
   Loaded: loaded (/usr/lib/systemd/system/postgresql.service; enabled; preset: enabled)  
   Active: active (exited) since Thu 2025-06-05 10:56:49 UTC; 4s ago  
     Process: 23379 ExecStart=/bin/true (code=exited, status=0/SUCCESS)  
   Main PID: 23379 (code=exited, status=0/SUCCESS)  
     CPU: 1ms  
  
jul 05 10:56:49 op-db-primary systemd[1]: Starting postgresql.service - PostgreSQL RDBMS...  
jul 05 10:56:49 op-db-primary systemd[1]: Finished postgresql.service - PostgreSQL RDBMS.  
administrator@op-db-primary:~$
```

2.3. Creación de usuarios en PostgreSQL

Accederemos a la consola de PostgreSQL:

```
sudo -u postgres psql
```

Y dentro ejecutaremos los comandos SQL para crear los usuarios y bases de datos necesarios para OpenPaDi:

- Para la API:

```
CREATE DATABASE opadi_db;
```

```
CREATE USER opadi_user WITH PASSWORD 'abc123..';
```

```
GRANT ALL PRIVILEGES ON DATABASE opadi_db TO opadi_user;
```

```
administrator@op-db-primary:~$ sudo -u postgres psql
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
Type "help" for help.

postgres=# CREATE DATABASE opadi_db;
CREATE DATABASE
postgres=# CREATE USER opadi_user WITH PASSWORD 'abc123..';
postgres'# GRANT ALL PRIVILEGES ON DATABASE opadi_db TO opadi_user;
postgres'#
```

- Para Keycloak:

```
CREATE DATABASE keycloak_db;
```

```
CREATE USER keycloak_user WITH PASSWORD 'abc123..';
```

```
GRANT ALL PRIVILEGES ON DATABASE keycloak_db TO keycloak_user;
```

```
postgres'# CREATE DATABASE keycloak_db;
postgres'# CREATE USER keycloak_user WITH PASSWORD 'abc123..';
postgres'# GRANT ALL PRIVILEGES ON DATABASE keycloak_db TO keycloak_user;
postgres'#
```

Podemos comprobar que todo se creó correctamente:

```

administrator@op-db-primary:~$ sudo -u postgres psql
psql (16.9 (Ubuntu 16.9-0ubuntu0.24.04.1))
Type "help" for help.

postgres=# \l
                                         List of databases
   Name    |     Owner      | Encoding | Locale Provider | Collate           | Ctype            | ICU Locale | ICU Rules | Access privileges
----+-----+-----+-----+-----+-----+-----+-----+-----+
keycloak_db | keycloak_user | UTF8    | libc       | es_ES.UTF-8 | es_ES.UTF-8 |             |             |
opadi_db    | opadi_user    | UTF8    | libc       | es_ES.UTF-8 | es_ES.UTF-8 |             |             |
postgres   | postgres      | UTF8    | libc       | es_ES.UTF-8 | es_ES.UTF-8 |             |             |
template0   | postgres      | UTF8    | libc       | es_ES.UTF-8 | es_ES.UTF-8 |             |             |
template1   | postgres      | UTF8    | libc       | es_ES.UTF-8 | es_ES.UTF-8 |             |             |
(5 rows)

postgres=# \du
                     List of roles
 Role name | Attributes
----+-----+
keycloak_user |
opadi_user  |
postgres    | Superuser, Create role, Create DB, Replication, Bypass RLS
postgres=#

```

3. Instalación y configuración de MinIO



La instalación de MinIO se hará en la máquina *op-storage-1* (192.168.1.14), que servirá como sistema de almacenamiento de objetos para guardar imágenes, PDFs o cualquier tipo de archivo en el que se guarden los documentos históricos digitalizados.

3.1. Instalación de MinIO

Descargaremos el binario de MinIO² y lo haremos ejecutable:

```
wget https://dl.min.io/server/minio/release/linux-amd64/minio
```

```
sudo mv minio /usr/local/bin/
```

```
sudo chmod +x /usr/local/bin/minio
```

² Es recomendable asegurarse de que el enlace de descarga es el correcto en el sitio web oficial de MinIO.

```
administrator@op-storage-1:~$ wget https://dl.min.io/server/minio/release/linux-amd64/minio
--2025-06-05 11:59:43-- https://dl.min.io/server/minio/release/linux-amd64/minio
Resolving dl.min.io (dl.min.io)... 178.128.69.202, 138.68.11.125
Connecting to dl.min.io (dl.min.io)|178.128.69.202|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 110756024 (106M) [application/octet-stream]
Saving to: 'minio'

minio          58%[=====]   61,80M  8,49KB/s    eta 1m 47s
```

```
administrator@op-storage-1:~$ sudo mv minio /usr/local/bin/
[sudo] password for administrator:
administrator@op-storage-1:~$ sudo chmod +x /usr/local/bin/minio
administrator@op-storage-1:~$
```

Y verificaremos que la instalación se completó correctamente:

```
minio --versión
```

```
administrator@op-storage-1:~$ minio --version
minio version RELEASE.2025-05-24T17-08-30Z (commit-id=ecde75f9112f8410cb6cacb4b76193f1475b587e)
Runtime: go1.24.3 linux/amd64
License: GNU AGPLv3 - https://www.gnu.org/licenses/agpl-3.0.html
Copyright: 2015-2025 MinIO, Inc.
administrator@op-storage-1:~$
```

3.2. Creación de directorio para los datos de MinIO

El siguiente paso será crear un directorio con los permisos y propietario del usuario que vayamos a usar para este servicio. Se podría crear un usuario dedicado, pero en este ejemplo utilizamos el usuario *administrator*.

```
sudo mkdir -p /srv/minio/data
```

```
sudo chown -R administrator:administrator /srv/minio
```

```
administrator@op-storage-1:~$ ls -ld /srv/minio
drwxr-xr-x 3 administrator administrator 4096 jun  5 12:08 /srv/minio
```

3.3. Configuración de las credenciales de acceso raíz de MinIO

Las credenciales raíz *MINIO_ROOT_USER* y *MINIO_ROOT_PASSWORD* se configuran mediante variables de entorno. Para ello, primero, crearemos un archivo de entorno:

```
sudo mkdir -p /etc/default
```

```
sudo nano /etc/default/minio
```

E introduciremos en él las siguientes líneas:

```
GNU nano 7.2                               /etc/default/minio *
MINIO_ROOT_USER="openpadiadmin"
MINIO_ROOT_PASSWORD="abc123.."
MINIO_VOLUMES="/srv/minio/data"
MINIO_OPTS="--console-address :9001"
```

³

3.4. Creación del Servicio Systemd para MinIO

Si queremos que MinIO se inicie automáticamente cada vez que se reinicie nuestro servidor *op-storage-1*, vamos a necesitar crear un servicio systemd.

Primero crearemos el archivo del servicio:

```
sudo nano /etc/systemd/system/minio.service
```

Y le añadiremos el siguiente contenido⁴:

³ Para mayor comodidad, se puede copiar y pegar el contenido desde el repositorio:
<https://github.com/robleslf/OpenPaDi/blob/main/docs/infrastructure-configs/minio/minio.conf>

⁴ Para mayor comodidad, puede copiarse desde
<https://github.com/robleslf/OpenPaDi/blob/main/docs/infrastructure-configs/minio/minio.service>

```
GNU nano 7.2          /etc/systemd/system/minio.service *
[Unit]
Description=MinIO Object Storage Server
Documentation=https://docs.min.io
Wants=network-online.target
After=network-online.target
AssertFileIsExecutable=/usr/local/bin/minio

[Service]
WorkingDirectory=/home/administrator

User=administrator
Group=administrator

Environment="MINIO_ROOT_USER=openpadiadmin"
Environment="MINIO_ROOT_PASSWORD=abc123.."
Environment="MINIO_OPTS[--console-address :9001]"

ExecStart=/usr/local/bin/minio server $MINIO_OPTS /srv/minio/data

Restart=always
RestartSec=10
LimitNOFILE=65536
LimitNPROC=4096
TimeoutStopSec=30
KillSignal=SIGTERM
SendSIGKILL=no

[Install]
WantedBy=multi-user.target
```

Recargaremos *systemd*, habilitaremos el servicio MinIO y lo iniciaremos:

```
sudo systemctl daemon-reload
sudo systemctl enable minio.service
sudo systemctl start minio.service
```

Finalmente, comprobaremos que el servicio se inició sin problemas y se encuentra activo y habilitado:

```
sudo systemctl status minio.service
```

```

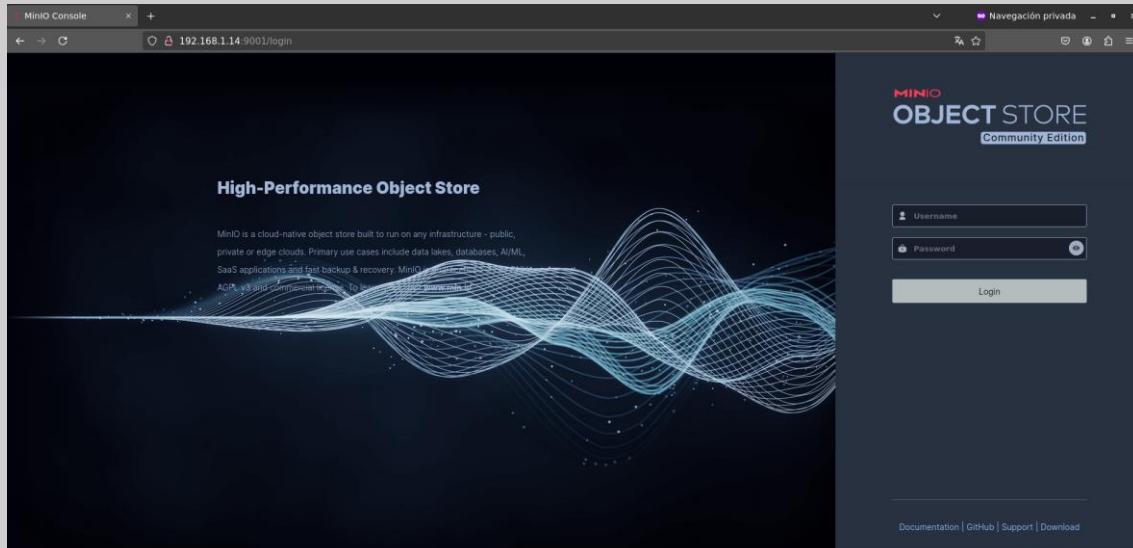
administrator@op-storage-1: ~
felipero... x adminis... x adminis... x adminis... x adminis... x adminis... x
● minio.service - MinIO Object Storage Server
   Loaded: loaded (/etc/systemd/system/minio.service; enabled; preset: enabled)
   Active: active (running) since Thu 2025-06-05 12:25:27 UTC; 6s ago
     Docs: https://docs.min.io
 Main PID: 18575 (minio)
    Tasks: 7 (limit: 4609)
   Memory: 200.0M (peak: 200.5M)
      CPU: 591ms
     CGroup: /system.slice/minio.service
             └─18575 /usr/local/bin/minio server --console-address :9001 /srv/m>

jun 05 12:25:27 op-storage-1 systemd[1]: Started minio.service - MinIO Object S>
jun 05 12:25:28 op-storage-1 minio[18575]: INFO: Formatting 1st pool, 1 set(s),>
jun 05 12:25:28 op-storage-1 minio[18575]: INFO: WARNING: Host local has more t>
jun 05 12:25:28 op-storage-1 minio[18575]: MinIO Object Storage Server
jun 05 12:25:28 op-storage-1 minio[18575]: Copyright: 2015-2025 MinIO, Inc.
jun 05 12:25:28 op-storage-1 minio[18575]: License: GNU AGPLv3 - https://www.gn>
jun 05 12:25:28 op-storage-1 minio[18575]: Version: RELEASE.2025-05-24T17-08-30>
jun 05 12:25:28 op-storage-1 minio[18575]: API: http://192.168.1.100:9000 http>
jun 05 12:25:28 op-storage-1 minio[18575]: WebUI: http://192.168.1.100:9001 htt>
jun 05 12:25:28 op-storage-1 minio[18575]: Docs: https://docs.min.io
lines 1-21/21 (END)

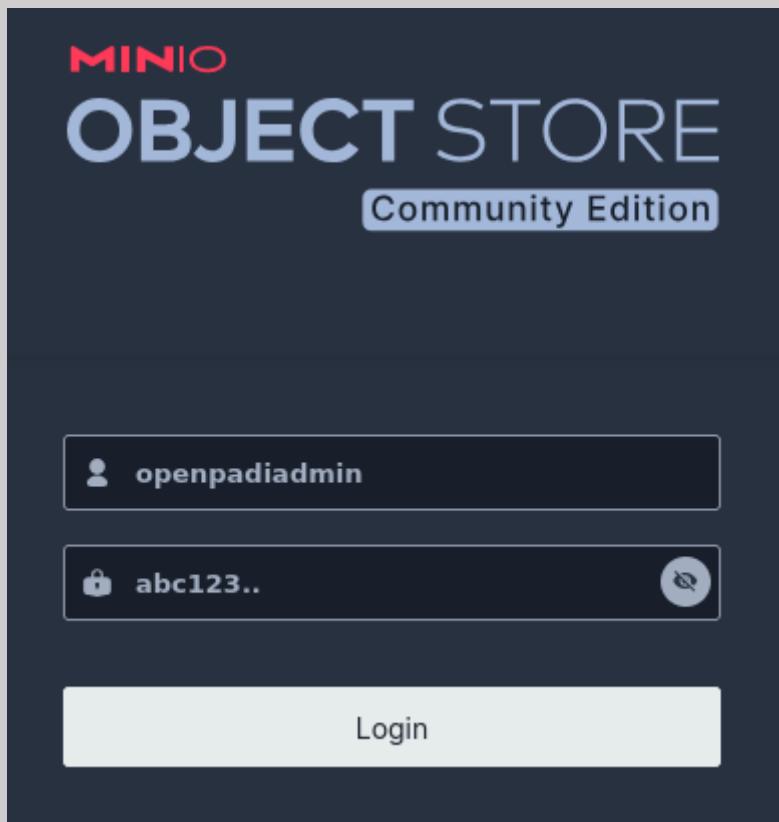
```

3.5. Acceso a la Consola Web y creación del bucket OpenPaDi

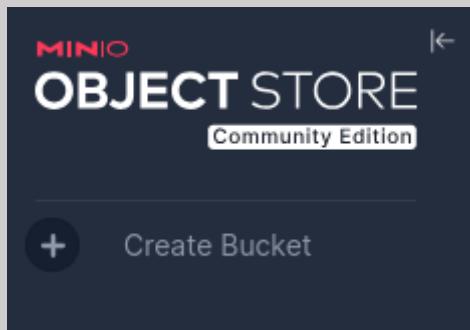
Para acceder a la consola de MinIO, desde el cliente gráfico nos conectaremos a <http://192.168.1.14:9001> desde el navegador:



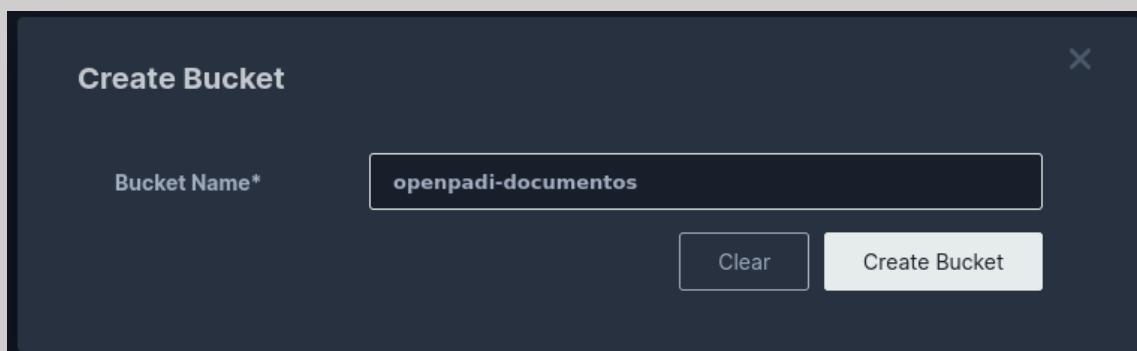
Nos autenticaremos con el usuario y contraseña que indicamos antes:



Una vez dentro, crearemos un nuevo *bucket*:



Le llamaremos *openpadi-documentos*:





4. Instalación y configuración del clúster K3S



La instalación de K3s se hará en las máquinas *op-web-1* (192.168.1.10), que actuará como nodo máster, y *op-api-1* (192.168.1.12), que será un nodo *worker*.

4.1. Instalación de Docker en ambas máquinas

Aunque KS3 viene con Containerd instalado y sería suficiente, tener Docker CLI será útil para la construcción de imágenes (aunque las imágenes necesarias para este manual están ya creadas⁵).

```
sudo apt install -y docker.io  
sudo systemctl enable --now docker  
sudo usermod -aG docker $USER
```

⁵ <https://hub.docker.com/repositories/robleslf>

```

administrator@op-web-1:~$ systemctl status docker.service
● docker.service - Docker Application Container Engine
  Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: en>
  Active: active (running) since Thu 2025-06-05 12:49:51 UTC; 17s ago
TriggeredBy: • docker.socket
    Docs: https://docs.docker.com
  Main PID: 21848 (dockerd)
    Tasks: 9
   Memory: 21.3M (peak: 21.6M)
     CPU: 334ms
    CGroup: /system.slice/docker.service
              └─21848 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/con>

jun 05 12:49:51 op-web-1 systemd[1]: Starting docker.service - Docker Application Engine...
jun 05 12:49:51 op-web-1 dockerd[21848]: time="2025-06-05T12:49:51.452307548Z" level=info msg="Starting containerd"
jun 05 12:49:51 op-web-1 dockerd[21848]: time="2025-06-05T12:49:51.453192481Z" level=info msg="Listening on fd://"
jun 05 12:49:51 op-web-1 dockerd[21848]: time="2025-06-05T12:49:51.453385270Z" level=info msg="Containerd has started"
jun 05 12:49:51 op-web-1 dockerd[21848]: time="2025-06-05T12:49:51.538264294Z" level=info msg="Containerd has started"
jun 05 12:49:51 op-web-1 dockerd[21848]: time="2025-06-05T12:49:51.884077196Z" level=info msg="Containerd has started"
jun 05 12:49:51 op-web-1 dockerd[21848]: time="2025-06-05T12:49:51.900647591Z" level=info msg="Containerd has started"
jun 05 12:49:51 op-web-1 dockerd[21848]: time="2025-06-05T12:49:51.900743586Z" level=info msg="Containerd has started"
jun 05 12:49:51 op-web-1 dockerd[21848]: time="2025-06-05T12:49:51.939142944Z" level=info msg="Containerd has started"

```

```

administrator@op-api-1:~$ systemctl status docker.service
● docker.service - Docker Application Container Engine
  Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: en>
  Active: active (running) since Thu 2025-06-05 12:50:13 UTC; 30s ago
TriggeredBy: • docker.socket
    Docs: https://docs.docker.com
  Main PID: 21902 (dockerd)
    Tasks: 9
   Memory: 21.6M (peak: 21.9M)
     CPU: 429ms
    CGroup: /system.slice/docker.service
              └─21902 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/con>

jun 05 12:50:12 op-api-1 systemd[1]: Starting docker.service - Docker Application Engine...
jun 05 12:50:12 op-api-1 dockerd[21902]: time="2025-06-05T12:50:12.624542978Z" level=info msg="Starting containerd"
jun 05 12:50:12 op-api-1 dockerd[21902]: time="2025-06-05T12:50:12.625428275Z" level=info msg="Listening on fd://"
jun 05 12:50:12 op-api-1 dockerd[21902]: time="2025-06-05T12:50:12.625650526Z" level=info msg="Containerd has started"
jun 05 12:50:12 op-api-1 dockerd[21902]: time="2025-06-05T12:50:12.724858434Z" level=info msg="Containerd has started"
jun 05 12:50:13 op-api-1 dockerd[21902]: time="2025-06-05T12:50:13.075703584Z" level=info msg="Containerd has started"
jun 05 12:50:13 op-api-1 dockerd[21902]: time="2025-06-05T12:50:13.092771165Z" level=info msg="Containerd has started"
jun 05 12:50:13 op-api-1 dockerd[21902]: time="2025-06-05T12:50:13.092912937Z" level=info msg="Containerd has started"
jun 05 12:50:13 op-api-1 dockerd[21902]: time="2025-06-05T12:50:13.136325473Z" level=info msg="Containerd has started"
jun 05 12:50:13 op-api-1 systemd[1]: Started docker.service - Docker Application Engine...

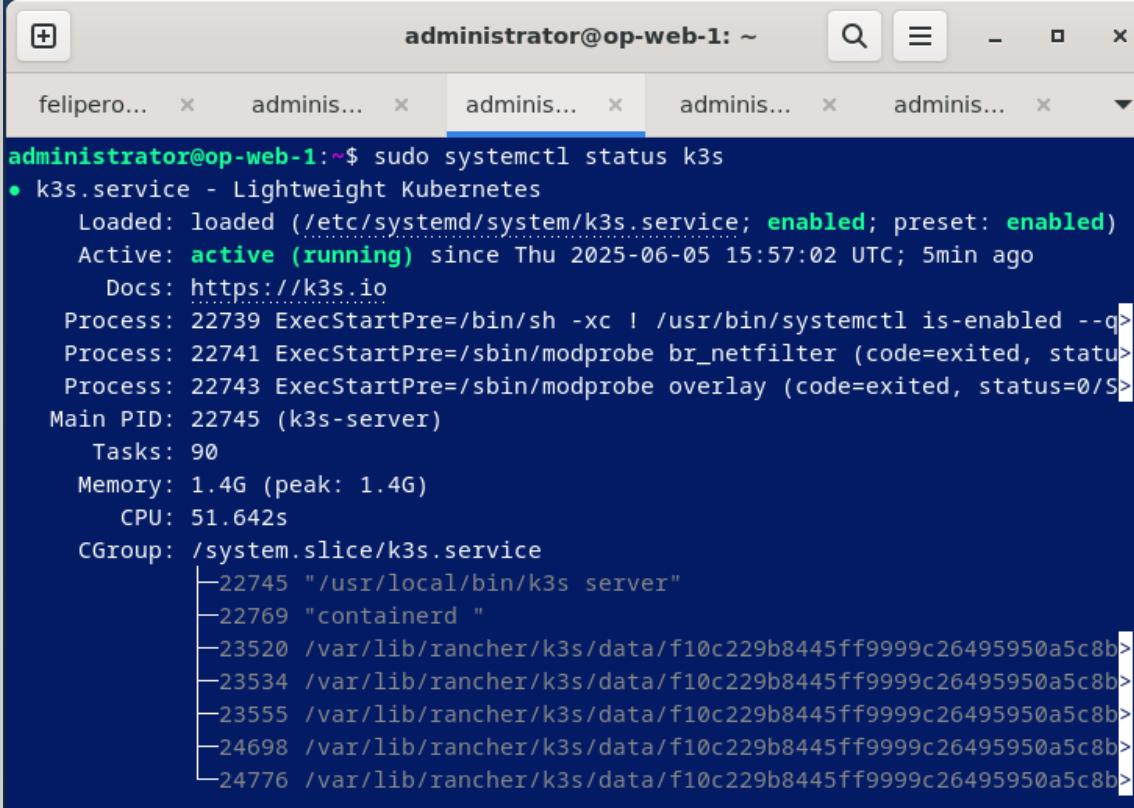
```

4.2. Instalación del Nodo Máster de K3s en *op-web-1*

Procederemos a la instalación de K3s en esta máquina, indicándole cuál es la IP de nuestro nodo dentro de la red interna. Esto es muy importante, ya que por defecto puede coger la IP que tenemos en el adaptador 1, que en este

escenario está en modo NAT y sería la recibida por DHCP a través de nuestro router doméstico.

```
curl -sfL https://get.k3s.io | INSTALL_K3S_EXEC="--node-ip 192.168.1.10 --flannel-iface enp0s8" sh -
```



A screenshot of a terminal window titled "administrator@op-web-1: ~". The window has multiple tabs open, with the fourth tab active. The command "sudo systemctl status k3s" is being run in the terminal. The output shows the k3s.service is active (running) since the previous day. It lists several processes and their main PID, tasks, memory usage, CPU time, and cgroup information. The cgroup tree starts with /system.slice/k3s.service and includes container IDs like 22745, 22769, 23520, 23534, 23555, 24698, and 24776.

```
administrator@op-web-1:~$ sudo systemctl status k3s
● k3s.service - Lightweight Kubernetes
  Loaded: loaded (/etc/systemd/system/k3s.service; enabled; preset: enabled)
  Active: active (running) since Thu 2025-06-05 15:57:02 UTC; 5min ago
    Docs: https://k3s.io
 Process: 22739 ExecStartPre=/bin/sh -xc ! /usr/bin/systemctl is-enabled --quiet k3s &
 Process: 22741 ExecStartPre=/sbin/modprobe br_netfilter (code=exited, status=0/SUCCESS)
 Process: 22743 ExecStartPre=/sbin/modprobe overlay (code=exited, status=0/SUCCESS)
 Main PID: 22745 (k3s-server)
   Tasks: 90
  Memory: 1.4G (peak: 1.4G)
    CPU: 51.642s
   CGroup: /system.slice/k3s.service
           └─22745 "/usr/local/bin/k3s server"
               ├─22769 "containerd"
               ├─23520 /var/lib/rancher/k3s/data/f10c229b8445ff9999c26495950a5c8b>
               ├─23534 /var/lib/rancher/k3s/data/f10c229b8445ff9999c26495950a5c8b>
               ├─23555 /var/lib/rancher/k3s/data/f10c229b8445ff9999c26495950a5c8b>
               ├─24698 /var/lib/rancher/k3s/data/f10c229b8445ff9999c26495950a5c8b>
               └─24776 /var/lib/rancher/k3s/data/f10c229b8445ff9999c26495950a5c8b>
```

A continuación, configuraremos *kubectl* para nuestro usuario:

```
mkdir -p $HOME/.kube
sudo cp /etc/rancher/k3s/k3s.yaml $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
chmod 600 $HOME/.kube/config
sudo kubectl get nodes
```

```
administrator@op-web-1:~$ sudo kubectl get nodes
NAME      STATUS      ROLES      AGE      VERSION
op-web-1   Ready      control-plane,master   8m49s   v1.32.5+k3s1
```

4.3. Instalación de Node.js y npm en *op-web-1*



Para el *frontend* necesitaremos instalar ambas en esta máquina:

```
sudo apt install -y nodejs npm
```

```
node -v
```

```
npm -v
```

```
administrator@op-web-1:~$ node -v  
v18.19.1
```

```
administrator@op-web-1:~$ npm -v  
9.2.0
```

4.4. Preparación, construcción y Dockerización del Frontend Svelte en *op-web-1*



Primero crearemos el directorio donde se alojará nuestro *frontend*:

```
cd ~
```

```
mkdir opadi-frontend
```

```
cd opadi-frontend
```

Una vez estemos en nuestro directorio, crearemos el esqueleto de lo que después será nuestra aplicación;

```
npx sv create .
```

Habrá que ir contestando las siguientes preguntas al asistente:

```
administrator@op-web-1:~/opadi-frontend$ npx sv create .  
Need to install the following packages:  
  sv@0.8.7  
Ok to proceed? (y) y
```

- ◆ Which template would you like?
 - SvelteKit minimal (barebones scaffolding for your new app)
 - SvelteKit demo
 - Svelte library

- ◆ Add type checking with TypeScript?
 - Yes, using TypeScript syntax
 - Yes, using JavaScript with JSDoc comments
 - No

- ◆ What would you like to add to your project? (use arrow keys / space bar)
 - prettier
 - eslint
 - vitest
 - playwright
 - tailwindcss
 - sveltekit-adapter (deployment - <https://svelte.dev/docs/kit/adapters>)
 - drizzle
 - lucia
 - mdsvex
 - paraglide
 - storybook

- ◆ sveltekit-adapter: Which SvelteKit adapter would you like to use?
 - auto
 - node
 - static (@sveltejs/adapter-static)
 - vercel
 - cloudflare
 - netlify

- ◆ Which package manager do you want to install dependencies with?
 - None
 - npm
 - yarn
 - pnpm
 - bun
 - deno

```
◆ Successfully installed dependencies
◆ Successfully formatted modified files
◆ Project next steps
  1: git init && git add -A && git commit -m "Initial commit" (optional)
  2: npm run dev -- --open

To close the dev server, hit Ctrl-C

Stuck? Visit us at https://svelte.dev/chat
```

You're all set!

A continuación, instalaremos las dependencias necesarias generadas por el paso anterior, así como las adicionales para OpenPaDi:

- Dependencias generadas:

```
npm install
```

- Dependencias adicionales para OpenPaDi:

```
npm install axios
```

```
npm install @sveltejs/adapter-static
```

```
npm install keycloak-js
```

La primera nos servirá para realizar peticiones HTTP a la API, la segunda para generar un *build* estático de la aplicación, y la tercera para la futura integración de Keycloak.

Ahora veremos que dentro de nuestro directorio se han creado los siguientes archivos y directorios:

```
administrator@op-web-1:~/opadi-frontend$ ls
eslint.config.js  node_modules  package-lock.json  src      svelte.config.js
jsconfig.json      package.json   README.md        static    vite.config.js
```

El siguiente paso será editar el archivo `svelte.config.js` con el siguiente contenido⁶:

```
GNU nano 7.2                                     svelte.config.js *
import adapter from '@sveltejs/adapter-static';

const config = {
  kit: {
    adapter: adapter({
      fallback: 'index.html'
    })
  }
};

export default config;
```

Modificaremos también lo que será la página principal de OpenPaDi a partir del fichero `src/routes/+page.svelte`; su contenido es bastante extenso, puede copiarse del siguiente enlace del repositorio:
<https://github.com/robleslf/OpenPaDi/blob/main/frontend-svelte/src/routes/%2Bpage.svelte>

Asimismo, aunque el servidor de Keycloak se configurará más adelante, vamos a crear el fichero `src/lib/keycloak.js` y le pegaremos el siguiente contenido:
<https://github.com/robleslf/OpenPaDi/blob/main/frontend-svelte/src/lib/keycloak.js>

Ahora, con todos los archivos listos, podemos construir nuestra aplicación Svelte desde nuestro directorio `~/opadi-frontend/`:

```
npm run build
```

```
> Using @sveltejs/adapter-static
  Wrote site to "build"
  ✓ done
```

Este proceso de construcción se realizará a través del Dockerfile que levanta el contenedor del *frontend*, pero no está de más haber comprobado que la aplicación se construye sin fallos.

⁶ Para mayor comodidad, puede copiar y pegarse el contenido desde el fichero del repositorio:
<https://github.com/robleslf/OpenPaDi/blob/main/frontend-svelte/svelte.config.js>

4.5. Dockerización de la aplicación *frontend*

Ahora crearemos dicho Dockerfile en este mismo directorio, y le añadiremos el siguiente contenido⁷:

```
GNU nano 7.2                               Dockerfile *
FROM node:20 AS builder

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build

FROM nginx:stable-alpine

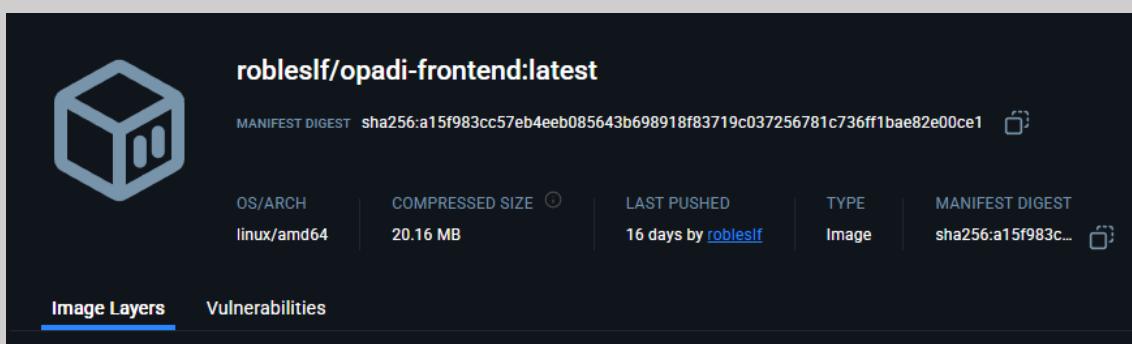
COPY --from=builder /app/build /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

Actualmente, la imagen construida a partir de este Dockerfile se encuentra alojada en Docker Hub⁸ y no es necesario crearla, no obstante, el proceso fue el siguiente:

```
docker build -t robleslf/opadi-frontend:latest .
docker login
docker push robleslf/opadi-frontend:latest
```



⁷ <https://github.com/robleslf/OpenPaDi/blob/main/frontend-svelte/Dockerfile>

⁸ <https://hub.docker.com/repository/docker/robleslf/opadi-frontend/tags/latest/sha256-a15f983cc57eb4eeb085643b698918f83719c037256781c736ff1bae82e00ce1>

4.6. Construcción y Dockerización del *Backend API* en *op-api-1*



Nuestra máquina *op-api-1* será la que actuará como nodo worker del clúster K3s; será aquí donde construyamos y *dockericemos* la API del *backend*.

Primero crearemos y nos moveremos al directorio *opadi-api*:

```
cd ~  
mkdir opadi-api  
cd opadi-api
```

Una vez dentro, prepararemos el entorno virtual de Python:

```
sudo apt install -y python3-venv  
python3 -m venv venv  
source venv/bin/activate
```

Deberíamos ver en el prompt que el estamos en el entorno:

```
administrator@op-api-1:~/opadi-api$ source venv/bin/activate  
(venv) administrator@op-api-1:~/opadi-api$
```

A continuación, crearemos el fichero *main.py*:

```
touch main.py
```

El contenido de *main.py* es extenso e incluye la lógica de los *endpoints*, conexión a la base de datos, integración con MinIO y la validación de tokens de Keycloak. Se puede obtener directamente del repositorio del proyecto:
<https://github.com/robleslf/OpenPaDi/blob/main/backend-api/main.py>

Crearemos, asimismo, el archivo *requirements.txt*, en el que listaremos todas las dependencias de Python necesarias para nuestra aplicación. El contenido de este lo podemos obtener directamente de aquí:
<https://github.com/robleslf/OpenPaDi/blob/main/backend-api/requirements.txt>

4.7. Dockerización del *Backend API*

Lo primero será crear nuestro *Dockerfile*, donde introduciremos el siguiente contenido⁹:

```
GNU nano 7.2                               Dockerfile *
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .

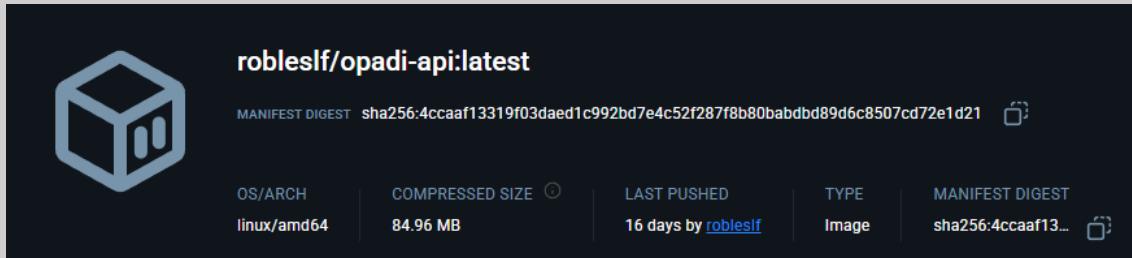
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

EXPOSE 8000

<Original>
```

Al igual que se hizo con el *frontend*, se construyó una imagen del *backend* de OpenPaDi a partir de este *Dockerfile* y se subió a Docker Hub¹⁰:



El proceso para ello fue el siguiente:

```
docker build -t robleslf/opadi-api:latest .
docker login
docker push robleslf/opadi-api:latest
```

⁹ <https://github.com/robleslf/OpenPaDi/blob/main/backend-api/Dockerfile>

¹⁰ <https://hub.docker.com/repository/docker/robleslf/opadi-api/tags/latest/sha256-4ccaaef13319f03daed1c992bd7e4c52f287f8b80babdbd89d6c8507cd72e1d21>

4.8. Despliegue de las Aplicaciones OpenPaDi

Prepararemos, en `op-web-1` el directorio donde almacenaremos los archivos YAML de Kubernetes:

```
cd ~  
mkdir -p k8s-manifests  
cd k8s-manifests
```

Una vez dentro del nuevo directorio, crearemos el manifiesto DE *Deployment deployment-api.yaml* para el *Backend API*. El contenido será el siguiente¹¹:

¹¹ Su contenido se puede copiar desde el repositorio:
<https://github.com/robleslf/OpenPaDi/blob/main/kubernetes-manifests/deployment-api.yaml>

```
kind: Deployment
metadata:
  name: opadi-api
  labels:
    app: opadi-api
spec:
  replicas: 1
  selector:
    matchLabels:
      app: opadi-api
  template:
    metadata:
      labels:
        app: opadi-api
    spec:
      hostAliases:
        - ip: "192.168.1.10"
          hostnames:
            - "auth.openpadi.local"
      containers:
        - name: opadi-api-container
          image: robleslf/opadi-api:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 8000
```

Crearemos también el manifiesto de Servicio para el *Backend API*, *service.api.yaml*. Será este servicio el que permita que otros componentes del clúster se comuniquen con la API usando el nombre DNS *op-api*:

```
apiVersion: v1
kind: Service
metadata:
  name: op-api
  labels:
    app: opadi-api
spec:
  selector:
    app: opadi-api
  ports:
    - protocol: TCP
      port: 8000
      targetPort: 8000
  type: ClusterIP
```

12

Aplicaremos ambos manifiestos con *kubectl apply*:

```
kubectl apply -f deployment-api.yaml
```

```
kubectl apply -f service-api.yaml
```

```
administrator@op-web-1:~/k8s-manifests$ sudo kubectl apply -f deployment-api.yaml
deployment.apps/opadi-api created
administrator@op-web-1:~/k8s-manifests$ sudo kubectl apply -f service-api.yaml
service/op-api created
administrator@op-web-1:~/k8s-manifests$
```

Repetiremos estos dos últimos pasos para el *frontend*, creándole su propio *Deployment* y su propio *Service*:

```
cd ~/opadi-frontend/
```

- *deployment.yaml*:

¹² <https://github.com/robleslf/OpenPaDi/blob/main/kubernetes-manifests/service-api.yaml>

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: opadi-frontend
  labels:
    app: opadi-frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: opadi-frontend
  template:
    metadata:
      labels:
        app: opadi-frontend
    spec:
      containers:
        - name: opadi-frontend
          image: robleslf/opadi-frontend:latest
          ports:
            - containerPort: 80
```

13

- *service-frontend.yaml:*

```
apiVersion: v1
kind: Service
metadata:
  name: opadi-frontend-service
spec:
  selector:
    app: opadi-frontend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

14

Aplicamos los manifiestos del *frontend*:

```
kubectl apply -f deployment.yaml
```

¹³ <https://github.com/robleslf/OpenPaDi/blob/main/frontend-svelte/deployment.yaml>

¹⁴ <https://github.com/robleslf/OpenPaDi/blob/main/frontend-svelte/service.yaml>

```
kubectl apply -f service.yaml
```

```
administrator@op-web-1:~/opadi-frontend$ sudo kubectl apply -f deployment.yaml
deployment.apps/opadi-frontend created
administrator@op-web-1:~/opadi-frontend$ sudo kubectl apply -f service.yaml
service/opadi-frontend-service created
administrator@op-web-1:~/opadi-frontend$
```

Una vez creados los servicios y despliegues, lo verificaremos:

```
kubectl get pods -o wide
```

```
kubectl get svc
```

```
administrator@op-web-1:~/opadi-frontend$ sudo kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP          NODE   NOMINATED-NODE   READINESS   GATES
opadi-api-5688868cc9-r9v7j   1/1    Running   0          7m44s  10.42.0.14  op-web-1  <none>        <none>
opadi-frontend-598df58597-5k6lc 1/1    Running   0          2m45s  10.42.0.15  op-web-1  <none>        <none>
administrator@op-web-1:~/opadi-frontend$ sudo kubectl get svc
NAME            TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes      ClusterIP   10.43.0.1   <none>        443/TCP   2d23h
op-api          ClusterIP   10.43.220.0  <none>        8000/TCP  7m42s
opadi-frontend-service  ClusterIP   10.43.86.23  <none>        80/TCP    2m44s
administrator@op-web-1:~/opadi-frontend$
```

4.9. Instalación y Configuración de Cert-Manager



Esta herramienta es nativa de Kunernetes. Primero la instalaremos en nuestro clúster K3s y después configuraremos un emisor de certificados (*Issuer*). Todo ello se hará en *op-web-1*, que es donde tenemos configurado *kubectl*.

Debemos situarnos en el directorio *k8s-manifests*:

```
cd ~/k8s-manifests/
```

Una vez allí, ejecutaremos el siguiente comando para aplicar los CDRs de Cert-Manager:

```
kubectl apply -f https://github.com/cert-manager/cert-
manager/releases/latest/download/cert-manager.crds.yaml
```

Deberíamos ver una salida similar a esta:

```
administrator@op-web-1:~/k8s-manifests$ sudo kubectl apply -f https://github.com/cert-manager/cert-
manager/releases/latest/download/cert-manager.crds.yaml
customresourcedefinition.apirextensions.k8s.io/certificaterequests.cert-manager.io created
customresourcedefinition.apirextensions.k8s.io/certificates.cert-manager.io created
customresourcedefinition.apirextensions.k8s.io/challenges.acme.cert-manager.io created
customresourcedefinition.apirextensions.k8s.io/clusterissuers.cert-manager.io created
customresourcedefinition.apirextensions.k8s.io/issuers.cert-manager.io created
customresourcedefinition.apirextensions.k8s.io/orders.acme.cert-manager.io created
administrator@op-web-1:~/k8s-manifests$
```

Opcionalmente, podemos verificar que los CDRs se han creado con el siguiente comando:

```
sudo kubectl get crd | grep cert-manager.io
```

```
administrator@op-web-1:~/k8s-manifests$ sudo kubectl get crd | grep cert-manager.io
certificaterequests.cert-manager.io          2025-06-08T15:32:23Z
certificates.cert-manager.io                2025-06-08T15:32:23Z
challenges.acme.cert-manager.io            2025-06-08T15:32:23Z
clusterissuers.cert-manager.io            2025-06-08T15:32:23Z
issuers.cert-manager.io                   2025-06-08T15:32:24Z
orders.acme.cert-manager.io              2025-06-08T15:32:24Z
administrator@op-web-1:~/k8s-manifests$
```

Lo siguiente será instalar el controlador de Cert-Manager:

```
kubectl apply -f https://github.com/cert-manager/cert-
manager/releases/latest/download/cert-manager.yaml
```

A continuación, verificaremos la instalación de Cert-Manager:

```
kubectl get pods -n cert-manager -w
```

```
administrator@op-web-1:~/k8s-manifests$ sudo kubectl get pods -n cert-manager -w
NAME                  READY   STATUS    RESTARTS   AGE
cert-manager-6468fc8f56-rss5x   1/1     Running   0          65s
cert-manager-cainjector-7fd85dcc7-fzqwk  1/1     Running   0          65s
cert-manager-webhook-57df45f686-9gq7w   1/1     Running   0          65s
```

¹⁵ Los pods pueden tardar unos minutos en iniciarse completamente. Habrá que esperar a que todos estén en *Running* y 1/1.

Una vez esté todo listo, pasaremos al siguiente paso: la creación de un emisor de Certificados Autofirmados¹⁶.

Primero crearemos el archivo `selfsigned-issuer.yaml`¹⁷:

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: selfsigned-issuer
spec:
  selfSigned: {}
```

Aplicaremos el manifiesto del `Issuer`:

```
kubectl apply -f selfsigned-issuer.yaml
```

Podemos verificarlo también:

```
kubectl get issuer
```

```
administrator@op-web-1:~/k8s-manifests$ sudo kubectl get issuer
NAME          READY   AGE
selfsigned-issuer  True   36s
administrator@op-web-1:~/k8s-manifests$
```

4.10. Creación del Certificado TLS

Ahora que tenemos Cert-Manager instalado y el emisor de Certificados Autofirmados listos, solicitaremos un certificado TLS para los dominios `openpadi.local` (que usaremos para nuestro `frontend`) y `auth.openpadi.local` (el que usaremos para Keycloak).

En el directorio `k8s-manifests` crearemos el archivo `certificate.yaml`¹⁸:

¹⁶ En una fase de producción, lo ideal será hacerlo con Let's Encrypt.

¹⁷ <https://github.com/robleslf/OpenPaDi/blob/main/kubernetes-manifests/selfsigned-issuer.yaml>

¹⁸ <https://github.com/robleslf/OpenPaDi/blob/main/kubernetes-manifests/certificate.yaml>

```

apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: openpadi-tls
spec:
  secretName: openpadi-tls
  dnsNames:
    - openpadi.local
  issuerRef:
    name: selfsigned-issuer
    kind: Issuer

```

Una vez creado, lo aplicaremos:

```
kubectl apply -f certificate.yaml
```

Podemos describir el certificado recién creado con el siguiente comando:

```
kubectl describe certificate openpadi-tls
```

Es importante sobre todo la sección *Status*, que debe estar en *True*:

```

Status:
  Conditions:
    Last Transition Time: 2025-06-08T16:19:59Z
    Message: Certificate is up to date and has not expired
    Observed Generation: 1
    Reason: Ready
    Status: True
    Type: Ready
  Not After: 2025-09-06T16:19:59Z
  Not Before: 2025-06-08T16:19:59Z
  Renewal Time: 2025-08-07T16:19:59Z

```

4.11. Configuración del *Ingress*

Con el certificado listo, podemos crear el manifiesto del *Ingress*; lo haremos a través del archivo *ingress.yaml*. Este es uno de los archivos más importantes de nuestro escenario, ya que se encarga de que Traefik (el *Ingress* por defecto en K3s), enrute el tráfico externo (HTTP y HTTPS) hacia los servicios de nuestro clúster. Vamos a encargarnos de que lo haga utilizando el certificado que acabamos de crear.

Crearemos este archivo en el mismo directorio donde estamos, *k8s-manifests*. Pegaremos el contenido desde el siguiente enlace del repositorio:
<https://github.com/robleslf/OpenPaDi/blob/main/kubernetes-manifests/ingress.yaml>

A continuación, aplicaremos el manifiesto:

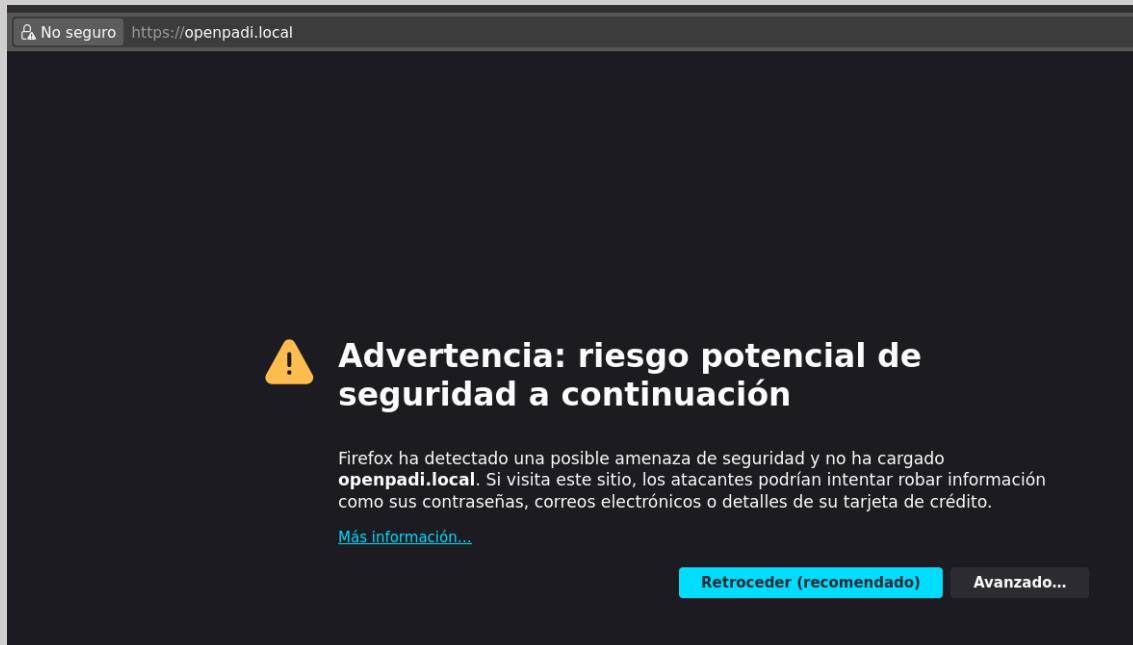
```
kubectl apply -f ingress.yaml
```

Verificaremos el estado del *Ingress*:

```
sudo kubectl get ingress opadi-frontend-ingress
```

```
sudo kubectl describe ingress opadi-frontend-ingress
```

Ahora verificaremos que el *Ingress* está funcionando. Desde el cliente gráfico, navegaremos a <https://openpadi.local>:



Certificado

(desconocido)

Validez

No antes	Sun, 08 Jun 2025 16:19:59 GMT
No después	Sat, 06 Sep 2025 16:19:59 GMT

Nombres alternativos del sujeto !

Nombre de la DNS	openpadi.local
------------------	----------------

Información de clave pública

Algoritmo	RSA
Tamaño de la clave	2048
Exponente	65537
Módulo	DF:8C:A2:AB:D2:92:51:A4:0F:95:05:76:59:EB:31:B4:0F:8B:58:FE:B0...

Misceláneo

Número de serie	0E:C9:B7:DE:46:83:60:AB:48:A3:C5:7D:AA:7A:91:FE
Algoritmo de firmas	SHA-256 with RSA Encryption
Versión	3
Descargar	

Podemos comprobarlo también mediante *curl*:

```
curl -kv https://openpadi.local/
curl -kv https://openpadi.local/api/documentos
```

4.12. Despliegue y Configuración de Keycloak



Vamos a preparar los manifiestos para Keycloak, los crearemos en el mismo directorio en el que hemos creado los anteriores, *k8s-manifests*:

- *keycloak-deployment.yaml*:
<https://github.com/robleslf/OpenPaDi/blob/main/kubernetes-manifests/keycloak-deployment.yaml>

- *keycloak-service.yaml*:

<https://github.com/robleslf/OpenPaDi/blob/main/kubernetes-manifests/keycloak-service.yaml>

Una vez creados, aplicaremos ambos con *kubectl*:

```
kubectl apply -f keycloak-deployment.yaml
```

```
kubectl apply -f keycloak-service.yaml
```

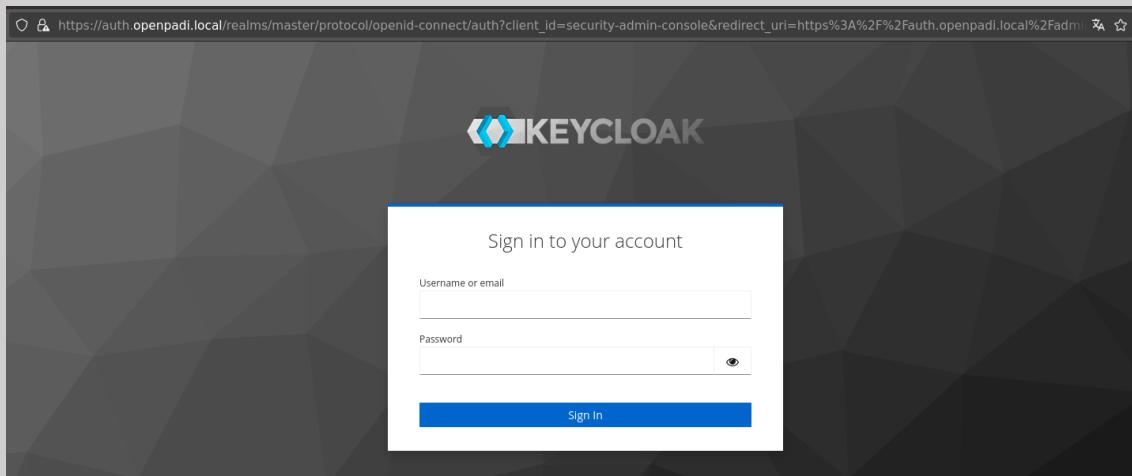
Y verificaremos el despliegue de Keycloak¹⁹:

```
kubectl get pods -l app=keycloak -w
```

```
sudo kubectl get svc keycloak-service
```

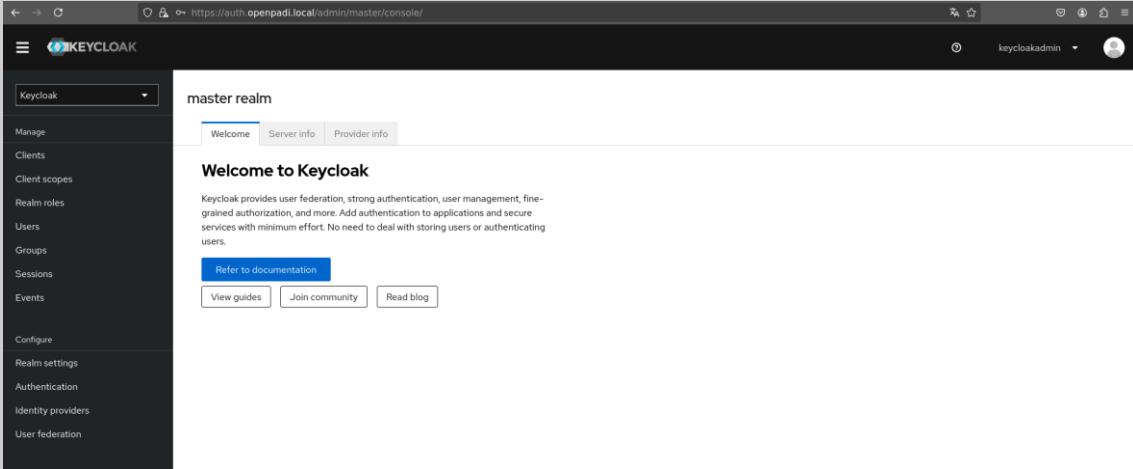
```
administrator@op-web-1:~/k8s-manifests$ sudo kubectl get pods -l app=keycloak -w
NAME           READY   STATUS    RESTARTS   AGE
keycloak-64b588d964-pzr6k   1/1     Running   0          10m
^Cadministrator@op-web-1:~/k8s-manifests$ sudo kubectl get svc keycloak-service
NAME        TYPE      CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
keycloak-service   ClusterIP   10.43.171.78   <none>       8080/TCP   68s
administrator@op-web-1:~/k8s-manifests$
```

El siguiente paso será acceder a la consola de Keycloak. Tal y como lo hemos configurado, podemos acceder desde el cliente gráfico en <https://auth.openpadi.local>:

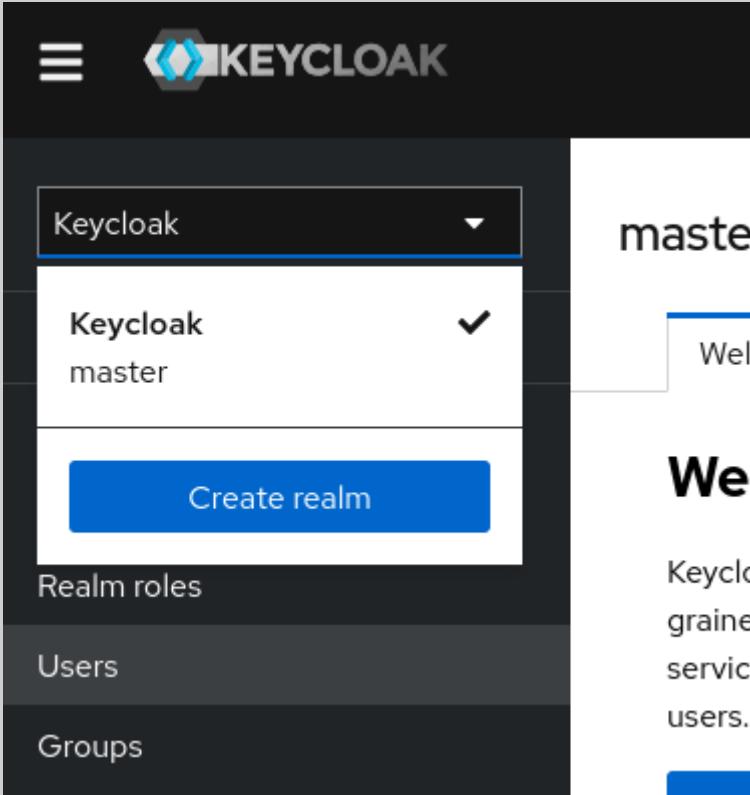


¹⁹ Esto puede tardar unos minutos, debemos esperar hasta que el *pod* tenga el *STATUS RUNNING* y el *READY* en 1/1.

Nos loguearemos con el usuario y contraseña que definimos en <https://github.com/robleslf/OpenPaDi/blob/main/kubernetes-manifests/keycloak-deployment.yaml>

A screenshot of a web browser displaying the Keycloak administration interface at the URL <https://auth.openpadi.local/admin/master/console/>. The page title is "master realm". The left sidebar contains a navigation menu with items such as "Manage", "Clients", "Client scopes", "Realm roles", "Users", "Groups", "Sessions", "Events", "Configure", "Realm settings", "Authentication", "Identity providers", and "User federation". The main content area displays the "Welcome to Keycloak" page, which includes a brief introduction about Keycloak's features like user federation, strong authentication, and fine-grained authorization, along with links to "Refer to documentation", "View guides", "Join community", and "Read blog".

Una vez dentro, crearemos un *realm* con nombre openpadi, que nos servirá como espacio aislado para gestionar usuarios, credenciales, roles y clientes en OpenPaDi²⁰:

A screenshot of the Keycloak admin interface showing the creation of a new realm. On the left, a sidebar menu is open under the "Keycloak" dropdown, showing "Keycloak" (with a checked checkbox), "master", "Create realm", "Realm roles", "Users", and "Groups". The main content area shows the "Welcome to Keycloak" page for the "master" realm. A large blue button labeled "Create realm" is prominently displayed.

²⁰ Es importante que el nombre del real y de los clientes coincidan con los que hemos indicado en <https://github.com/robleslf/OpenPaDi/blob/main/frontend-svelte/src/lib/keycloak.js>

Le daremos el nombre y lo dejaremos en *Enabled*:

Create realm

A realm manages a set of users, credentials, roles, and groups. A user belongs to and logs into a realm. Realms are isolated from one another and can only manage their own users.

Resource file Drag a file here or browse to upload

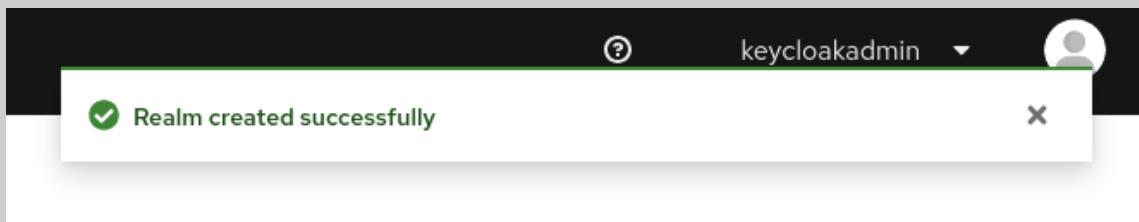
Upload a JSON file

1

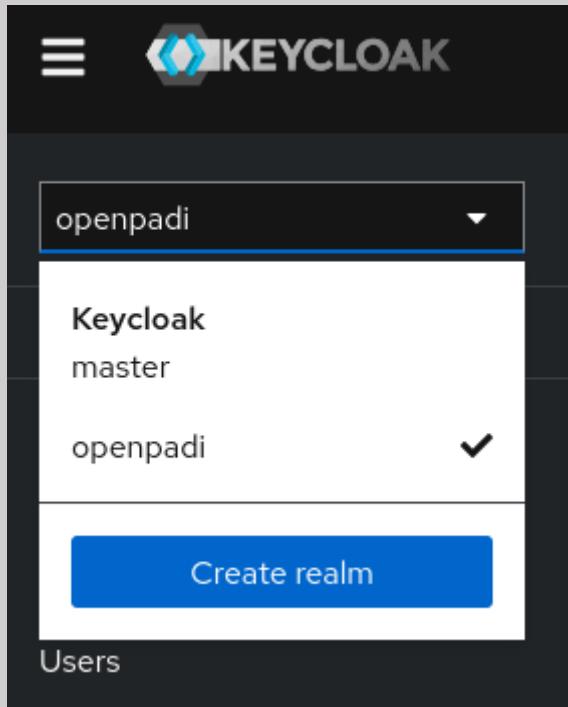
Realm name *

Enabled On

Nos mostrará la confirmación de que ha sido creado:



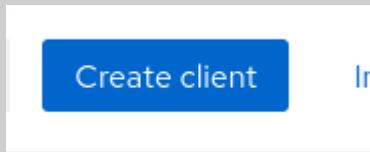
Y ahora podremos seleccionarlo en la pestaña de *realms*:



Una vez creado nuestro *realm*, crearemos nuestros clientes (nos aseguraremos de estar dentro de este *realm*): uno para nuestro *frontend* y otro para nuestro *backend*.

A screenshot of the 'Clients' list page in Keycloak. The left sidebar shows 'Clients' selected. The main area has tabs for 'Clients list', 'Initial access token', and 'Client registration'. It includes a search bar, a 'Create client' button, and a 'Refresh' button. A table lists several clients: 'account' (Client ID), '\${client_account}' (Name), OpenID Connect (Type), and '-' (Description); 'account-console' (Client ID), '\${client_account-console}' (Name), OpenID Connect (Type), and '-' (Description); 'admin-cli' (Client ID), '\${client_admin-cli}' (Name), OpenID Connect (Type), and '-' (Description); 'broker' (Client ID), '\${client_broker}' (Name), OpenID Connect (Type), and '-' (Description); 'realm-management' (Client ID), '\${client_realm-management}' (Name), OpenID Connect (Type), and '-' (Description); and 'security-admin-console' (Client ID), '\${client_security-admin-console}' (Name), OpenID Connect (Type), and '-' (Description).

En la pestaña de clientes, le daremos a *Create client*:



Crearemos primero el del *frontend*:

Clients > Create client

Create client

Clients are applications and services that can request authentication of a user.

1 General settings 2 Capability config 3 Login settings

Client type	OpenID Connect
Client ID *	openpadi-frontend
Name	OpenPaDi-frontend
Description	Cliente de OpenPaDI para frontend

Always display in UI Off

Next Back Cancel

En la pestaña de *Capability config* dejaremos desactivado *Client authentication*. De este modo lo convertiremos en un cliente público. Dejaremos *Standard Flow* activado.

Clients > Create client

Create client

Clients are applications and services that can request authentication of a user.

1 General settings 2 Capability config 3 Login settings

Client authentication <input checked="" type="checkbox"/> Off		
Authorization <input checked="" type="checkbox"/> Off		
Authentication flow	<input checked="" type="checkbox"/> Standard flow	<input checked="" type="checkbox"/> Direct access grants
	<input type="checkbox"/> Implicit flow	<input type="checkbox"/> Service accounts roles
	<input type="checkbox"/> OAuth 2.0 Device Authorization Grant	
	<input type="checkbox"/> OIDC CIBA Grant	

Next Back Cancel

En la pestaña de *Login settings*, en *Valid redirect URIs* es importante añadir el "*" para permitir redirecciones a cualquier subruta después del *login*. Cubriremos los campos del siguiente modo:

Create client
Clients are applications and services that can request authentication of a user.

1 General settings
2 Capability config
3 Login settings

Root URL https://openpadi.local

Home URL https://openpadi.local

Valid redirect URIs https://openpadi.local/*
+ Add valid redirect URIs

Valid post logout redirect URIs https://openpadi.local/*
+ Add valid post logout redirect URIs

Web origins https://openpadi.local
+ Add web origins

Save Back Cancel

Ahora crearemos el cliente del *backend*:

Create client
Clients are applications and services that can request authentication of a user.

1 General settings
2 Capability config
3 Login settings

Client type OpenID Connect

Client ID opadi-api

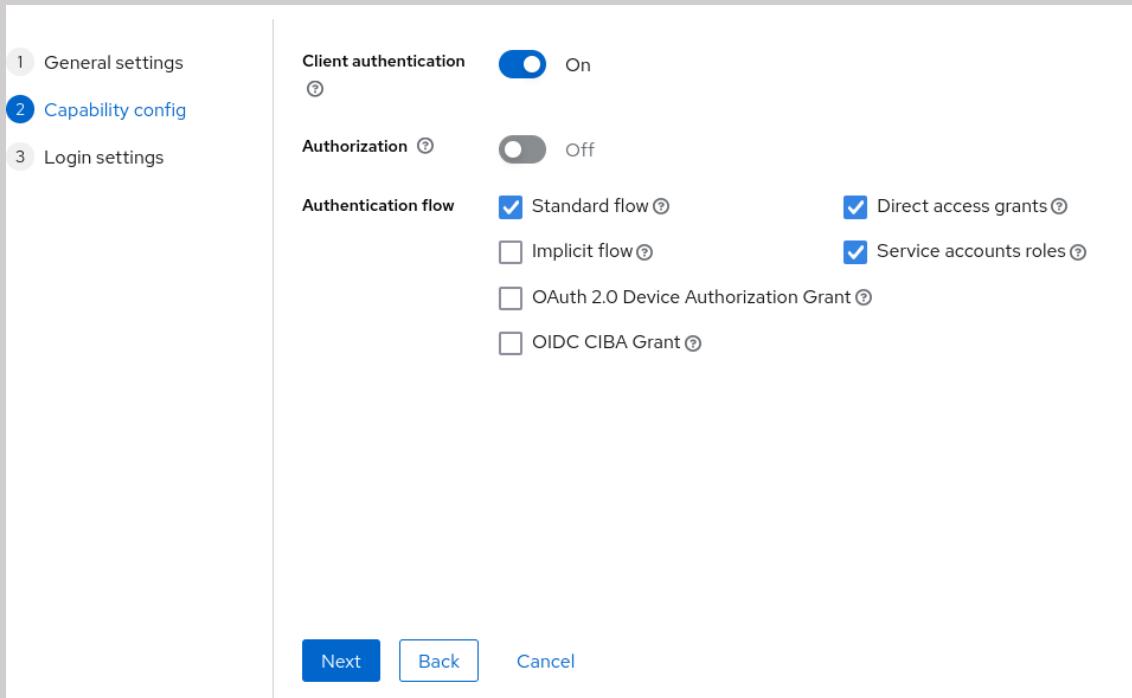
Name OpenPaDi API

Description Cliente de OpenPaDi para el backend

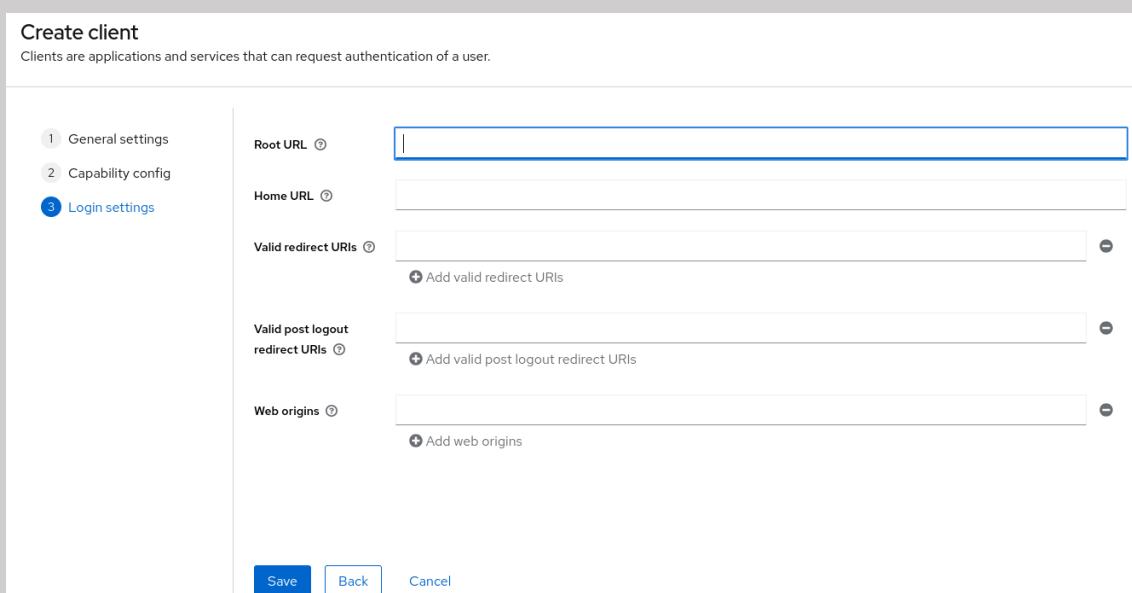
Always display in UI Off

Next Back Cancel

En la pestaña *Client config* activaremos *Client authentication*, ya que queremos que este sea un cliente confidencial. También activaremos *Service account roles* que, aunque en este proyecto no tendrá ninguna utilidad, en la aplicación final de OpenPaDi permitiría asignar roles:



En la pestaña de *Login settings* dejaremos todos los campos vacíos:



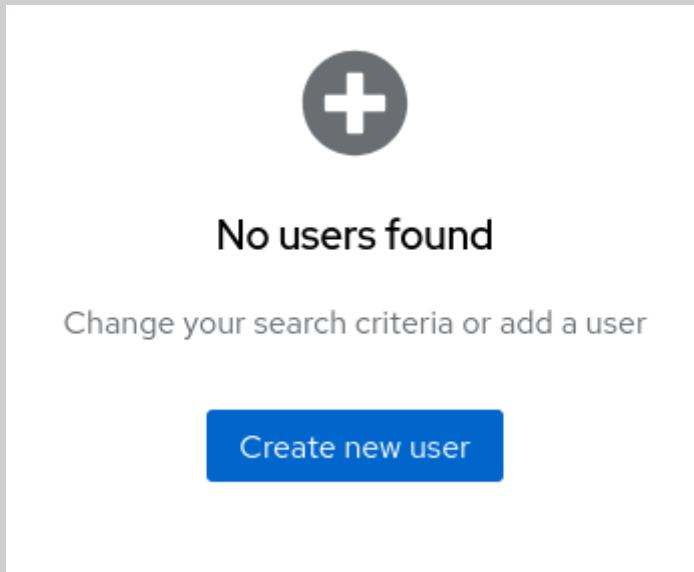
Una vez creado el cliente, iremos a la pestaña *Credentials* de este; deberíamos tener un *Secret*:

The screenshot shows the 'Credentials' tab of the 'opadi-api' client in the Keycloak admin console. The 'Client Authenticator' dropdown is set to 'Client Id and Secret'. Below it is a 'Client Secret' field containing a long string of characters, with options to 'Regenerate' or copy it. Another section for a 'Registration access token' is also present with similar regenerate and copy options. A 'Save' button is located at the bottom left of the form.

El siguiente paso será crear un usuario para probar que el flujo de autenticación funciona correctamente. Iremos a la pestaña *Users*:

The screenshot shows the 'Users' tab in the Keycloak admin console for the 'openpadi' realm. The sidebar has 'Manage' and 'Clients' sections, with 'Users' currently selected. The main area displays a message 'No users found' and includes a search bar with the placeholder 'Change your search criteria or add a user' and a blue 'Create new user' button.

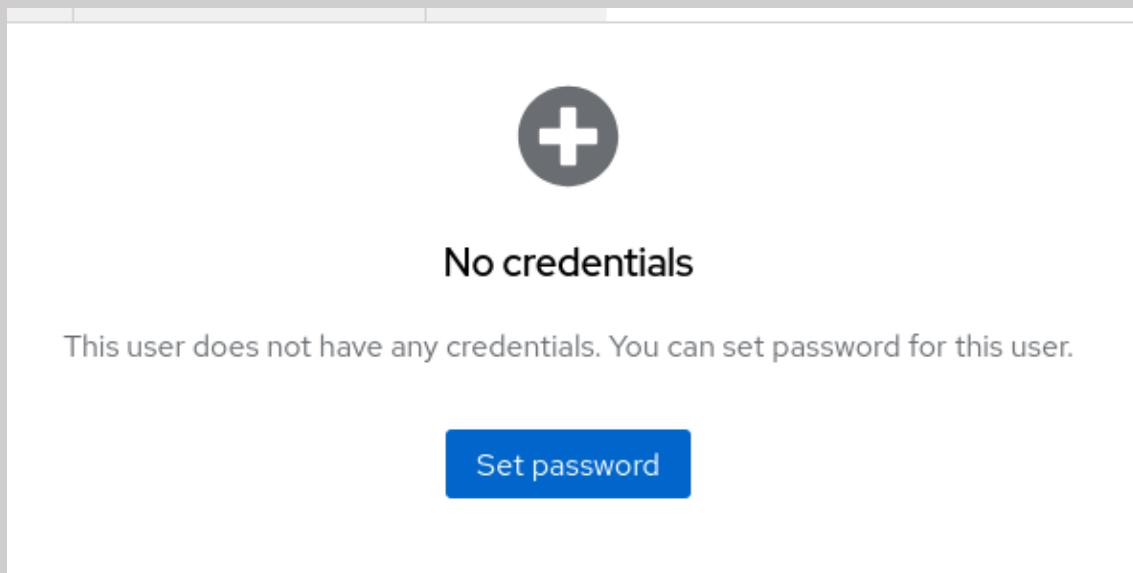
Le daremos a crear un usuario nuevo:



Le vamos a llamar *testuser*:

A screenshot of a user creation form. The title "General" is at the top left. On the right, there's a vertical sidebar with a "Jump to section" heading and a "General" link. The main area contains fields for "Username" (with a red asterisk) which has "testuser" typed into it, "Email" (empty), "First name" (empty), "Last name" (empty), and "Groups" (with a help icon) which has a "Join Groups" button. At the bottom are "Create" and "Cancel" buttons.

Iremos a la pestaña *Credentials* y le crearemos una contraseña:



Desmarcaremos la opción de contraseña temporal:

A screenshot of a modal dialog titled "Set password for testuser". The dialog contains two input fields: "Password *" and "Password confirmation *", both showing the value "abc123..". Below these fields is a toggle switch labeled "Temporary" with a question mark icon, which is currently set to "Off". At the bottom of the dialog are two buttons: a blue "Save" button on the left and a "Cancel" button on the right.

4.13. Verificación de la integración de Keycloak con el *frontend*

El siguiente paso será verificar la integración del *frontend* con Keycloak.
Volveremos a navegar a <https://openpadi.local>:

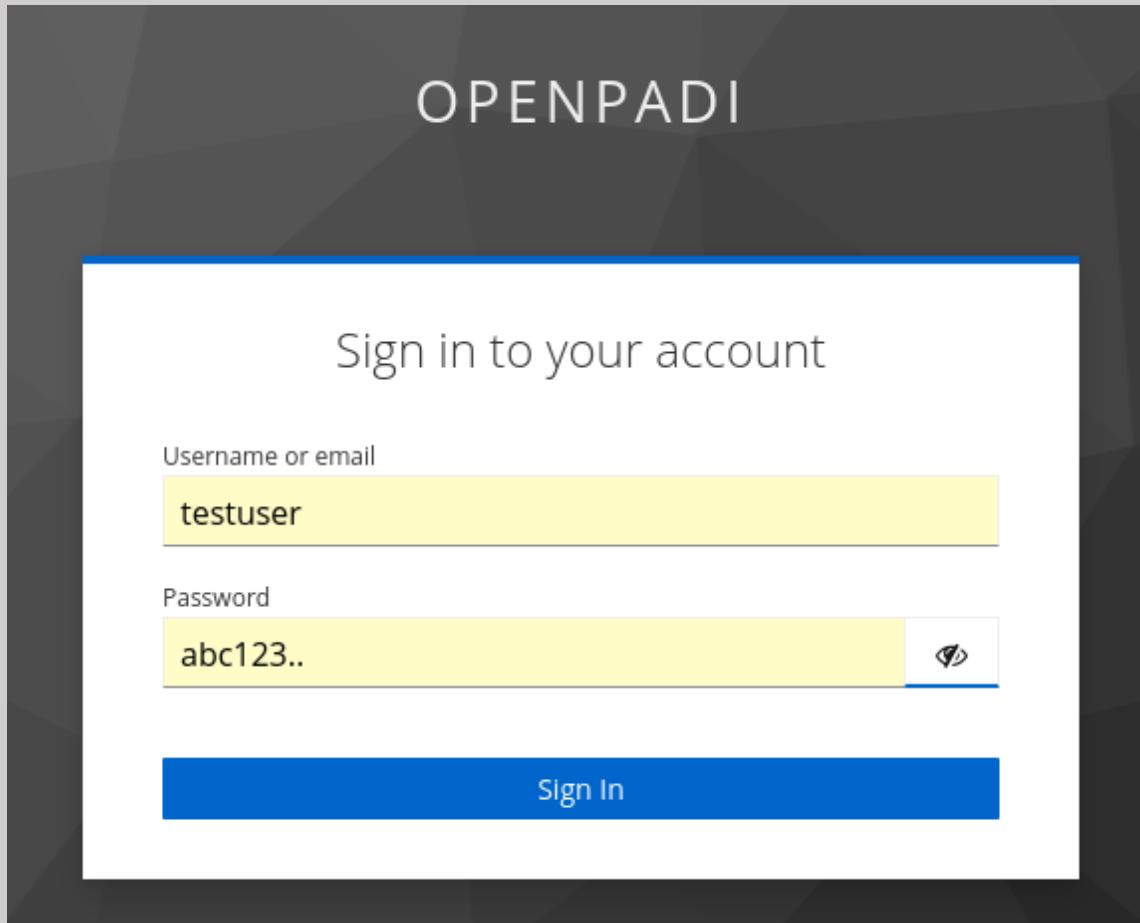


OpenPaDi - Repositorio de Transcripciones

Por favor, inicia sesión para continuar.

[Iniciar Sesión con Keycloak](#)

Y nos logueraremos con el usuario y contraseña que acabamos de crear:



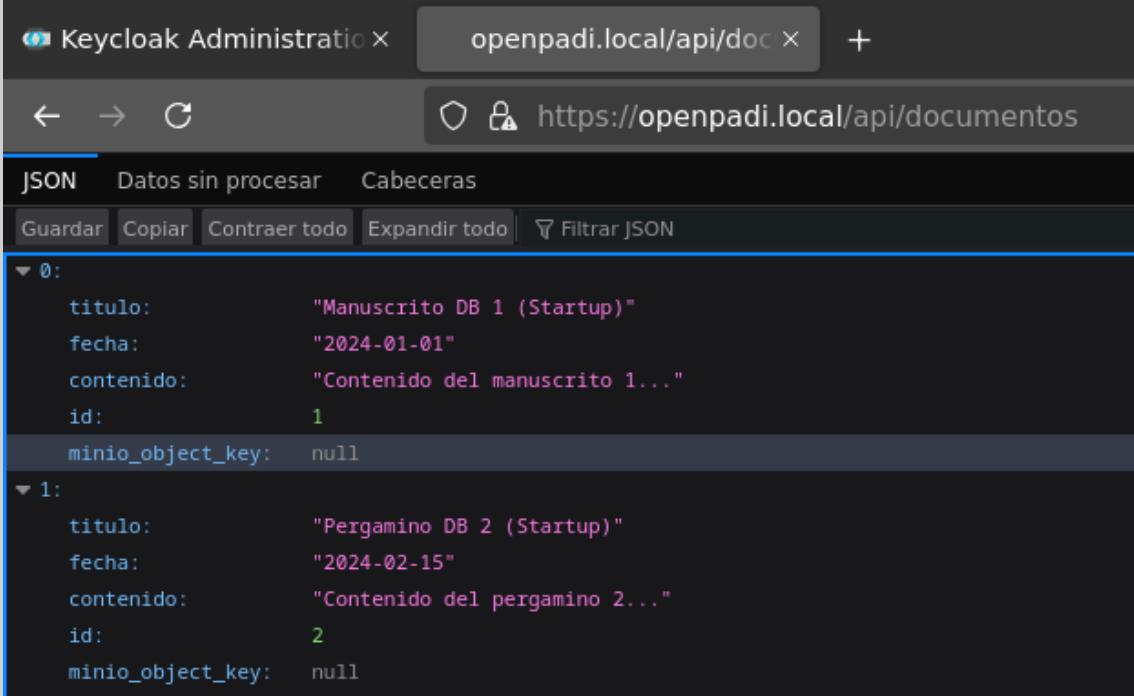
21

Debe dejarnos entrar:

A screenshot of the OpenPaDi dashboard. The top navigation bar shows the URL "openpadi.local/#state" and the address "https://openpadi.local". The main area features a red circular logo with a white bird and the letters "OP". Below the logo, the text "OpenPaDi - Repositorio de Transcripciones" is displayed in bold. A welcome message "Bienvenido, testuser!" is shown, along with a "Cerrar Sesión" button. A "Crear Nuevo Documento" form is present, with fields for "Título:" and "Fecha (YYYY-MM-DD)".

²¹ Si cuando creamos el usuario no le dimos un mail y nombre y apellidos, es muy posible que pida unos al entrar por primera vez.

Comprobaremos igualmente que nos deja ver la lista de documentos en <https://openpadi.local/api/documentos>:



```
JSON Datos sin procesar Cabeceras
Guardar Copiar Contraer todo Expandir todo Filtrar JSON
▼ 0:
  titulo: "Manuscrito DB 1 (Startup)"
  fecha: "2024-01-01"
  contenido: "Contenido del manuscrito 1..."
  id: 1
  minio_object_key: null
▼ 1:
  titulo: "Pergamino DB 2 (Startup)"
  fecha: "2024-02-15"
  contenido: "Contenido del pergamo 2..."
  id: 2
  minio_object_key: null
```

5. Validación final de la Integración completa del *frontend* con Keycloak y la API

El próximo paso es asegurarnos de que el *frontend*, además de utilizar Keycloak para autenticar usuarios, se encarga de que la API valide los tokens para las operaciones que lo requieran, como la subida de documentos.

Crearemos un documento de prueba:

```
feliperobleslopez@debian-base-gui:~$ touch mandarina.txt
```

Y en la interfaz de OpenPaDi lo subiremos:

Crear Nuevo Documento

Título:

Mandarina

Fecha (YYYY-MM-DD):

06 / 06 / 2025



Contenido:

Archivo de prueba para verificar la integración de la API

Archivo (PDF, Imagen, etc.):

Examinar... mandarina.txt

[Crear Documento](#)

Debe aparecernos en la lista de documentos:

Documentos Existentes

Manuscrito DB 1 (Startup) (ID: 1)

Fecha: 2024-01-01

Contenido: Contenido del manuscrito 1...

Pergamino DB 2 (Startup) (ID: 2)

Fecha: 2024-02-15

Contenido: Contenido del pergamo 2...

Mandarina (ID: 3)

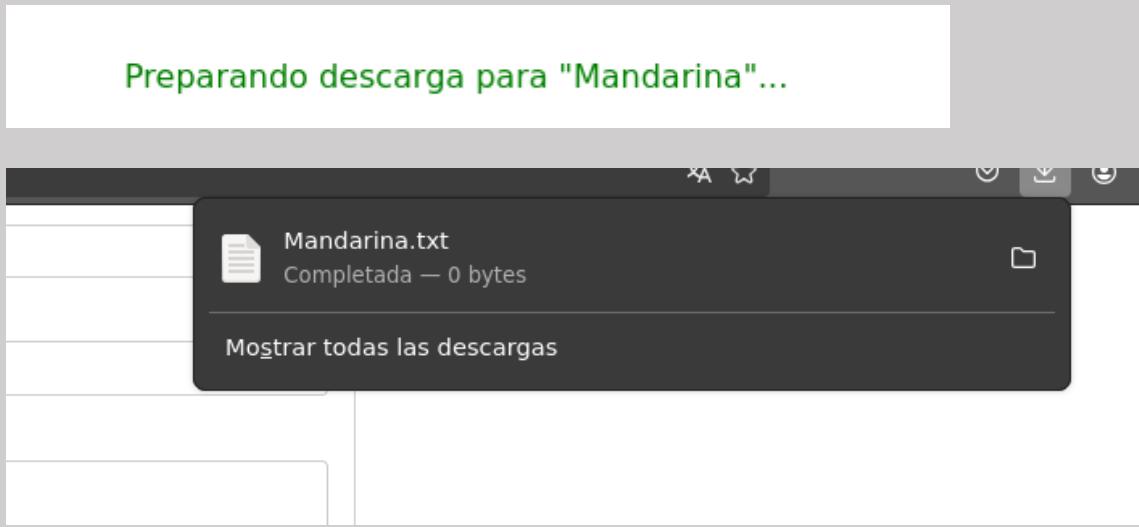
Fecha: 2025-06-06

Contenido: Archivo de prueba para verificar la integración de la API

[Ver/Descargar Archivo](#)

Si le damos a *Descargar* nos debe dejar bajarlo²²:

²² Es posible que la descarga se bloqué porque el navegador no confíe. Debemos darle a *Permitir*.



Si miramos la consola del navegador, en la pestaña red, podemos ver que en las peticiones GET se está utilizando el token en la cabecera *Authorization*:

Est	Mé	Domi...	Archivo	Inicia...	Tip	Trans...	Tar	Cabezas	Cookies	Solicitud	Respuesta	Tiempos	Traza de la pila
?	GET	au...	step1.html	subd...	htm	NS_B...	2,3						
200	GET	op...	favicon.png	Favic...	png	cach...	1,5						
200	GET	au...	step2.html	step...	htm	1,13 ...	686						
200	GET	au...	login-status-iframe.htm	subd...	htm	4,77 ...	4,3						
204	GET	au...	init?client_id=openpa...	login...	plai	232 B	0 B						
302	GET	au...	auth?client_id=openp...	subd...	htm	2,98 ...	220						
200	GET	op...	silent-check-sso.html	subd...	htm	449 B	220						
200	POS	au...	token	cuEr...	j...	4,32 ...	3,8						
200	GET	au...	account	cuEr...	j...	1,81 ...	1,3						
200	GET	op...	documentos	2.BiD...	j...	597 B	455						

Cabezas Cookies Solicitud Respuesta Tiempos Traza de la pila

date: Mon, 09 Jun 2025 08:38:31 GMT
server: unicorn
X-Firefox-Spdy: h2

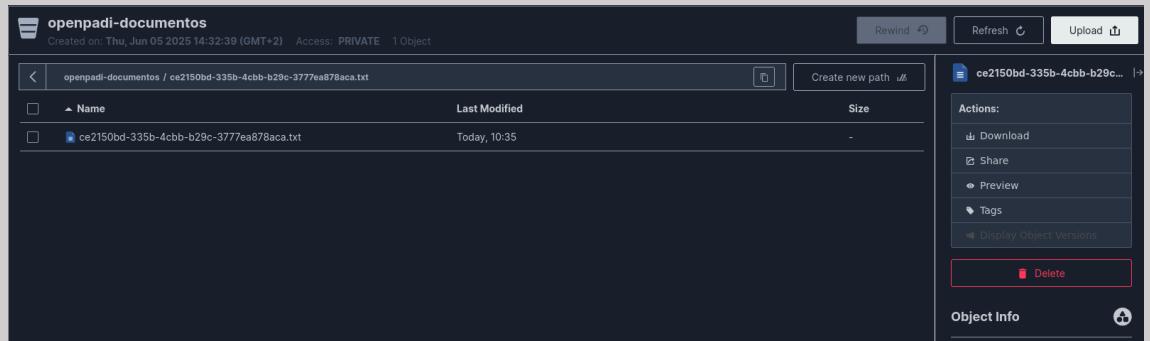
Accept: application/json, text/plain, */*
Accept-Encoding: gzip, deflate, br, zstd
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cClgOiAiSldUiIwiaZIkIA6ICjLcFh3Rk9fRINy1lYTxxLTQ5OWltOWRIYS04MzQ0NDRkOWIxYjIiLCJpc3MIOijodHRwczovL2F1dGgub3BibrW5wYWRpLWZyb250ZW5kliwbm9uY2UiOiiODYxZG11y05MTZmLTQ5NGQtODNINI04MtYWlsQG1haWwuY29tln0.io8gnu05f1JRlQM-9HtnG0AAAAmD6FC5iu5jMznDHoci_PwXneFSHnBkmSwndgves4wf7NgZUx8HDPGzZUeALEPzbdw3TxYGdxF7BT2Y8fyNn4zgjwfbVxIEv

Connection: keep-alive

Podemos, igualmente, verificar que el documento que hemos subido se encuentra tanto en MinIO como referenciado en PostgreSQL:

- Para MinIO entraremos en <http://192.168.1.14:9001>²³:

²³ Credenciales: *openpadiadmin* | *abc123..*



- Para PostgreSQL ejecutaremos la consulta “*SELECT * FROM documentos;*” en el servidor *op-db-primary*:

<code>id</code>	<code>titulo</code>	<code>fecha</code>	<code>contenido</code>	<code>minio_object_key</code>
1	Manuscrito DB 1 (Startup)	2024-01-01	Contenido del manuscrito 1...	
2	Pergamino DB 2 (Startup)	2024-02-15	Contenido del pergámido 2...	
3	Mandarina	2025-06-06	Archivo de prueba para verificar la integración de la API	<code>ce2150bd-335b-4cbb-b29c-3777ea878aca.txt</code>

Con esto tenemos listo todos los servicios necesarios para la Inraestructura PaaS de OpenPaDi, correctamente configurados e integrados entre sí.