
# Guía de configuración técnica

# Control de versiones

| Versión | Fecha de modificación | Responsable | Aprobador | Resumen de cambios |
|---|---|---|---|---|
| 1.0 | 22/09/2023 | Anjana Producto | Anjana Producto | Creación del documento |
| 1.1 | 31/01/2024 | Anjana Producto | Anjana Producto | Aclaración sobre el uso de archivos de configuración YAML |
| 1.2 | 16/02/2024 | Anjana Producto | Anjana Producto | Especificada configuración para la caché de Apache |

# Introducción

El presente documento es una guía para configurar, de manera correcta, Anjana en la organización. En él, se detallan las posibles configuraciones de cada microservicio, la manera de encriptar información sensible y también detalles del frontal de la aplicación.

# Configuración Backend

## General para todos los microservicios de Anjana

- Los entornos Anjana normalmente usan alias como hecateserver, edusaserver, … por lo que es necesario añadirlos en /etc/hosts.

```
127.0.0.1 indexservice zkservice s3service s3servicenode1 rdbservice ldapservice
127.0.0.1 hecateserver edusaserver zeusserver hermesserver viatorserver
viatorservernode1 minervaserver kernoserver totserver portunoserver
127.0.0.1 totpluginaqtivaserver totpluginawsglueserver totpluginawsiamserver
totpluginawss3server totpluginazureadserver totpluginazurefilesserver
totpluginazurestorageserver totplugingcpbigqueryserver totplugingcpiamserver
127.0.0.1 totplugingcpstorageserver totpluginhdfsserver totpluginhiveserver
totpluginjdbcserver totpluginjdbcdenodoserver totpluginjdbcoracleserver
totpluginjdbcredshiftserver totpluginjdbcsqlserverserver totpluginldapserver
totpluginpowerbiserver totplugintableauserver
```

Pero pueden ser añadidos directamente los nombres de los host, o del servicio en Kubernetes, y no editar /etc/hosts.

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://back1.anjanadata.com:50761/eureka
```

- Todos los microservicios de Anjana tienen por defecto un tamaño de de cabeceras HTTP de 2KB, este valor puede ser modificado cambiando esta propiedad en el fichero de propiedades.

```
server:
  max-http-header-size: 20000
```

- La url de validación de token en cada microservicio es http://<zeus_hostname>:<zeus_port>/internal/v1/auth/validate-token.

```
security:
  oauth2:
    resource:
      tokenInfoUri:
http://zeusserver:8088/internal/v1/auth/validate-token
```

- Configuración de Hecate Server en cada microservicio

Para el caso de Kubernetes se debe completar con el nombre del servicio o balanceador de carga.

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://hecateserver:50761/eureka
```

- Es posible activar las estadísticas de Hibernate para visualizar la información que ofrece con la siguiente configuración.

Es importante destacar que la generación de las estadísticas puede afectar al rendimiento por lo que debe estar activa solo de forma puntual.

```
jpa:
  properties:
    hibernate:
      generate_statistics: true
```

- Si se desea lanzar el servicio para que obtenga la configuración de Edusa es necesario añadir lo siguiente al comando:

```
--spring.config.import=optional:configserver:http://<edusa_server>:8888
--spring.profiles.active=default --spring.cloud.config.failFast=true
```

- Todos los módulos de Anjana incluyen un fichero de propiedades en formato YAML donde se configuran, entre otras, parámetros de la aplicación. Anjana provee los microservicios con una configuración por defecto. Esa configuración se puede sobreescribir con otros valores. Estas modificaciones podrán ser cargadas si el YAML se encuentra dentro de los ficheros de Edusa o si en los argumentos de arranque de la aplicación se añade:

```
--spring.profiles.active=default --spring.cloud.config.enabled=false
--spring.cloud.config.failFast=false --spring.config.additional-location=<ruta
al fichero de propiedades>
```

Nota: en la propiedad 'additional-location' se debe de poner la ruta al fichero, sin el fichero para que se complemente la configuración por defecto con la que se quiera añadir; en otro caso se sustituirá toda la configuración, incluso si no se pone valor.

- Para Hermes, Kerno, Minerva, Portuno y Zeus se puede configurar una lista de urls que puede utilizar Swagger para hacer llamadas REST a la API en su yml. Esta configuración es opcional

por lo que, en caso de no necesitar Swagger, se puede obviar. Estas urls deberán apuntar al proxy de viator para el correcto funcionamiento de Swagger, si no se incluye, Swagger genera una url válida por sí mismo.

```
swagger:
  server:
    url:
      - http://viatorserver/gateway
      - http://<anjana_host>/gateway
      - http://hostalias/gateway
```

- Si se quiere configurar claves (o propiedades de cualquier tipo) de manera que no estén en texto plano si no encriptadas, se puede hacer colocando la propiedad encriptada en Base64 entre comillas simples y concatenando delante {cipher}. De la siguiente manera

```
propiedad: '{cipher}asdf8976sdfa'
```

- Colector de basura: Todos los servicios que estén utilizando el jdk8 tendrán que tener el colector de basura G1 para manejar de forma más eficiente la memoria:

```
-XX:+UseG1GC -XX:+UseStringDeduplication
```

## Configuración de servicios para cargar propiedades en un servicio externo

Se ha añadido la opción de configurar los servicios para que obtengan propiedades desde un servicio establecido por configuración, en el caso mostrado a continuación es Portuno.

Para especificar esta url tenemos 3 posibilidades de configuración:

- Incluido en el fichero de configuración (distinto al application.yaml).

```
anjana:
  properties:
    wait: 10
    retry: 3
    url:
      - http://portunoserver:8998/internal/v2/appconf?prefix=hermes
```

- Incluido como variable de entorno. Si Anjana se ejecuta en una máquina Linux no se podrá utilizar este método ya que no se permite el "." como carácter en el nombre de la variable.
- Incluido en el descriptor de servicio como propiedad -D de java (-Danjana.properties.url[0]=http://portunoserver:8998/internal/v2/appconf?prefix=hermes). Ojo que esta propiedad va antes de la especificación de la aplicación jar.

```
ExecStart=/usr/lib/jvm/java-8-openjdk-amd64/bin/java -XX:+UseG1GC
-XX:+UseStringDeduplication -Xmx4G
-javaagent:/opt/common/xjar-agent-hibernate.jar -jar
-Danjana.properties.url[0]=http://portunoserver:8998/internal/v2/appconf?prefix=
```

```
hermes /opt/hermes/hermes.jar  --spring.cloud.config.uri=http://edusaserver:8888
--spring.profiles.active=default --spring.cloud.config.failFast=true
```

## Configuración de logs

Los microservicios están configurados por defecto para que den su salida a consola estándar para que los log puedan ser gestionados por la utilidad de logs del sistema, normalmente en entornos de máquina virtual se usará syslog, rsyslog y jornalctl para consumirmos o configurar su tratamiento.

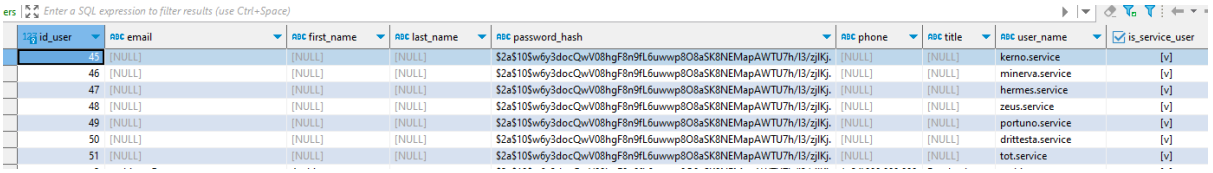
Todos ellos pueden ser configurados mediante el fichero yaml de cada uno siguiendo las prácticas estándar de Spring Boot.

Ejemplo de usos comunes:

```
logging:
  pattern:
    console: "%clr(%d{yyyy-MM-dd HH:mm:ss.SSS}){faint} [HERMES]
%clr(${LOG_LEVEL_PATTERN:%5p}) %clr(${PID:- }){magenta}
%clr(---){faint} %clr([%15.15t]){faint} %clr(%-40.40logger{39}){cyan}
%clr(:){faint} %m%n${LOG_EXCEPTION_CONVERSION_WORD:%wEx}"
  level:
    root: INFO
    com.anjana: DEBUG
```

## Securización de endpoints internos

Se ha incorporado la securización de los endpoints internos mediante la utilización de tokens de servicio. Para ello se han creado unos usuarios en la tabla zeus.users que están marcados como de servicio (aunque no se usa para nada más ese flag ahora mismo) .



| id_user | email | first_name | last_name | password_hash | phone | title | user_name | is_service_user |
|---|---|---|---|---|---|---|---|---|
| 45 | [NULL] | [NULL] | [NULL] | $2a$10$w6y3docQwV08hgF8n9fL6uwwp8O8aSK8NEMapAWTU7h/I3/zjIKj. | [NULL] | [NULL] | kerno.service | [v] |
| 46 | [NULL] | [NULL] | [NULL] | $2a$10$w6y3docQwV08hgF8n9fL6uwwp8O8aSK8NEMapAWTU7h/I3/zjIKj. | [NULL] | [NULL] | minerva.service | [v] |
| 47 | [NULL] | [NULL] | [NULL] | $2a$10$w6y3docQwV08hgF8n9fL6uwwp8O8aSK8NEMapAWTU7h/I3/zjIKj. | [NULL] | [NULL] | hermes.service | [v] |
| 48 | [NULL] | [NULL] | [NULL] | $2a$10$w6y3docQwV08hgF8n9fL6uwwp8O8aSK8NEMapAWTU7h/I3/zjIKj. | [NULL] | [NULL] | zeus.service | [v] |
| 49 | [NULL] | [NULL] | [NULL] | $2a$10$w6y3docQwV08hgF8n9fL6uwwp8O8aSK8NEMapAWTU7h/I3/zjIKj. | [NULL] | [NULL] | portuno.service | [v] |
| 50 | [NULL] | [NULL] | [NULL] | $2a$10$w6y3docQwV08hgF8n9fL6uwwp8O8aSK8NEMapAWTU7h/I3/zjIKj. | [NULL] | [NULL] | drittesta.service | [v] |
| 51 | [NULL] | [NULL] | [NULL] | $2a$10$w6y3docQwV08hgF8n9fL6uwwp8O8aSK8NEMapAWTU7h/I3/zjIKj. | [NULL] | [NULL] | tot.service | [v] |

Los distintos servicios necesitan tener el usuario y contraseña que van a utilizar para obtener el token de sistema y están definidos en el yaml de cada microservicio con las siguientes propiedades codificadas en base64:

```
anjana:
  client_id: <client_id>
  secret: <client_password>
```

El login es similar a un usuario normal por lo que el client_id y secret se corresponde con usuario y contraseña codificados en base 64.

# Edusa

Edusa es un servicio de configuración central que se puede configurar para servir configuraciones desde ficheros locales, desde un repositorio git, desde un Vault, … .

Tiene que ser lanzado con el argumento `--spring.config.additional-location=<path a directorio de fichero de propiedades>` para seleccionar el repositorio de propiedades (Filesystem, Git o Vault). Es importante que termine en "/" cuando se trata de un directorio, procediendo a leer los ficheros de configuración existentes.

Un ejemplo sería `--spring.config.additional-location=file:/opt/data/edusa/conf/`

En caso de establecer diferentes repositorios, se deberá establecer un orden mediante la propiedad `spring.cloud.config.server.<repositorio>.order`

## Repositorio Filesystem

- Solo es necesario ajustar la carpeta donde se encuentran los ficheros de configuración de los microservicios de Anjana

Ejemplo /opt/data/configrepo/configserver-localfiles.yaml:

```
spring:
  profiles:
    active: native
  cloud:
    config:
      server:
        native:
         search-locations:
         - file:/opt/data/configrepo
         - file:/opt/data/configrepo/{application}
         - file:/opt/data/configrepo/{application}/{profile}
```

## Repositorio Git

Se puede indicar que rama se va a usar en default-label

Ejemplo /opt/data/configrepo/configserver-gitfiles.yaml :

```
spring:
  cloud:
    config:
      server:
        git:
          uri: git@bitbucket.org:anjanadatacom/config-local.git
          default-label: develop
          skipSslValidation: true
          timeout: 10
          clone-on-start: true
          force-pull: true
          searchPaths: '{application}'
          ignoreLocalSshSettings: true
          privateKey: |
            -----BEGIN RSA PRIVATE KEY-----
                              .......
            -----END RSA PRIVATE KEY-----
```

## Repositorio Vault

Se permite la utilización de repositorios Vault mediante la herramienta de HashiCorp's Vault

Ejemplo:

```
spring:
  profiles:
    active: vault
  cloud:
    config:
      server:
         vault:
            token: ******
            kv-version: 2
            host: 0.0.0.0
            port: 8200
            authentication: TOKEN
```

Para su correcta configuración con su tecnología Vault es recomendable acceder al apartado de instalación del documento "Anjana Data - Config Vault" o a la documentación oficial correspondiente.

## Secrets Manager AWS

Permite guardar los parámetros sensibles de la configuración del microservicio en el servicio Secret Manager de AWS y recuperarla mediante llamada por API. Cada microservicio que se conecte al Secret Manager para recuperar el secreto no podrá utilizar Edusa como proveedor de configuración.

El microservicio deberá disponer de un AWS_ACCESS_KEY_ID y AWS_SECRET_ACCESS_KEY con el que identificarse o asignar un role de AWS si se trata de una EC2.

Añadir al yaml de configuración del microservicio la siguiente configuración:

```
aws:
  secretsmanager:
    enabled: true
    region: eu-central-1
```

En el comando de arranque del microservicio hay que quitar las propiedades:

```
--spring.config.import=configserver:http://edusaserver:8888
```

```
--spring.cloud.config.failFast=true
```

Dentro del AWS Secret Manager los secretos se guardan con el prefijo '/secret', acompañado del nombre del microservicio y del perfil de spring separado por '_' (configuración por defecto de AWS Secret Manager).

Por ejemplo para configurar Kerno con el profile default el secreto será /secret/kerno_default y su valor serán las propiedades que se requieran para que funcione.

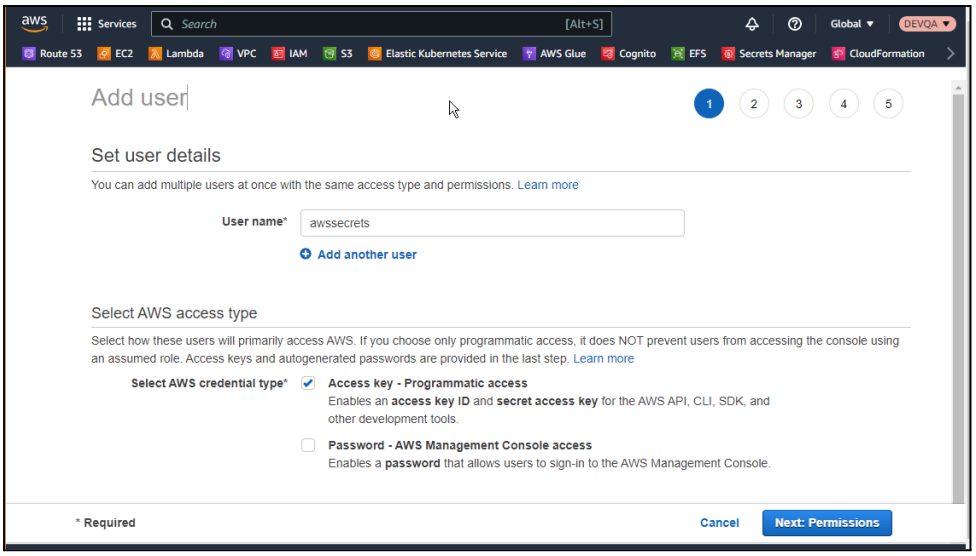
Se debe de crear una política para poder añadirla a la cuenta de servicio que podrá acceder al AWS Secrets Manager. Para ello, se va a crear una política como la siguiente

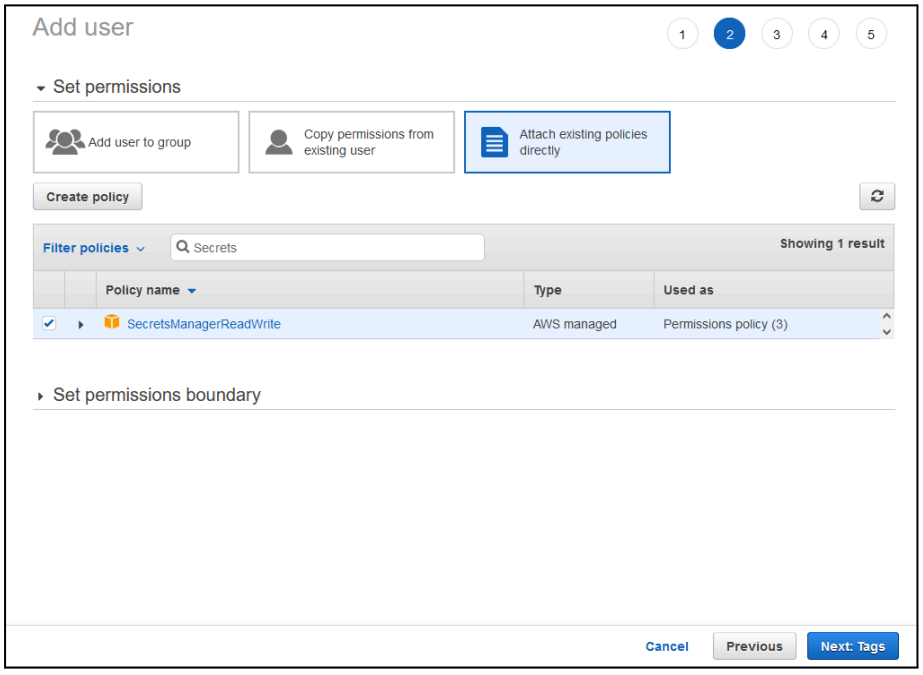
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
            "Resource":
"arn:aws:secretsmanager:eu-central-1:985175497294:secret:*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "secretsmanager:ListSecrets",
            "Resource": "*"
        }
    ]
}
```

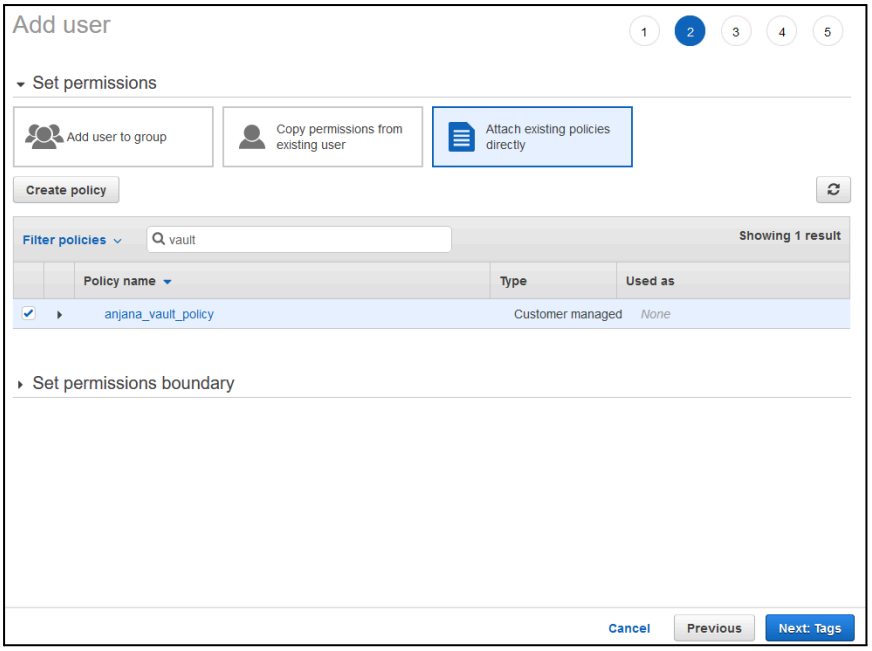
Se puede hacer también que únicamente tenga permiso para leer un secret, añadiendo el arn del secret en el resource de la política. Se añadiría de la siguiente manera.

```
{
       "Version": "2012-10-17",
       "Statement": [
       {
              "Sid": "VisualEditor0",
              "Effect": "Allow",
              "Action": "secretsmanager:GetSecretValue",
              "Resource":
"arn:aws:secretsmanager:eu-central-1:985175497294:secret:/secret/hermes_default-
fozbRl"
       },
       {
              "Sid": "VisualEditor1",
              "Effect": "Allow",
              "Action": "secretsmanager:ListSecrets",
              "Resource": "*"
       }
       ]
}
```

Una vez creada la política, se debe crear un usuario de IAM desde el menú de usuarios. En el asistente de creación, se le debe asignar las dos políticas que se encuentran en la foto de abajo.



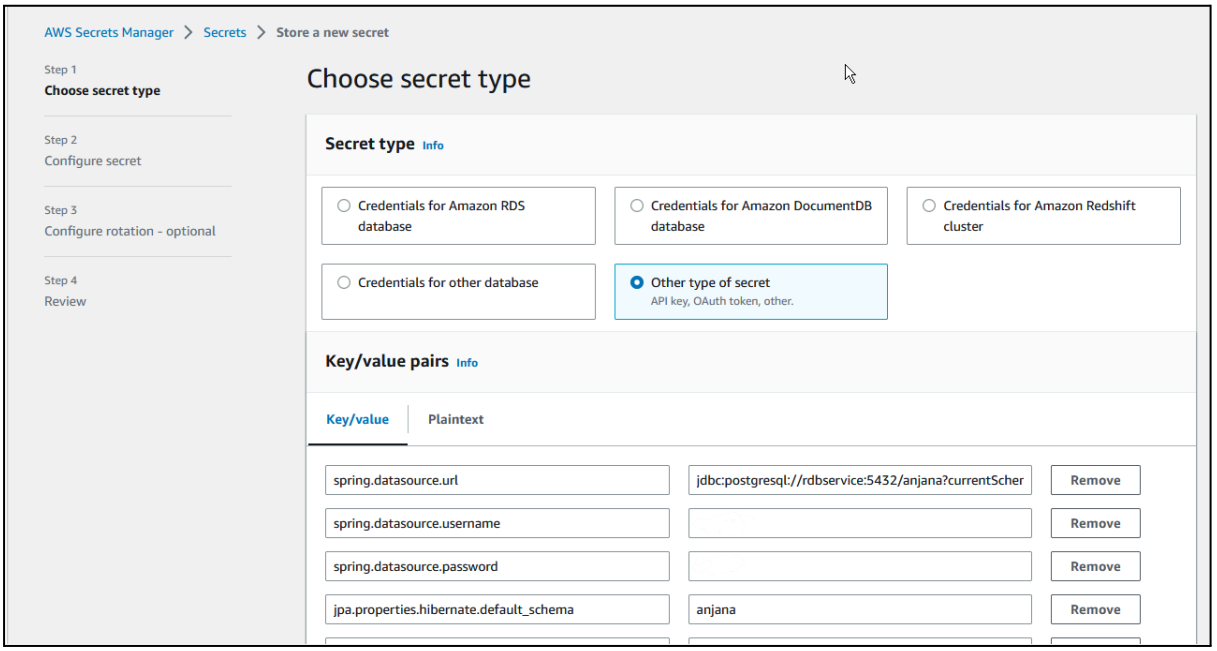





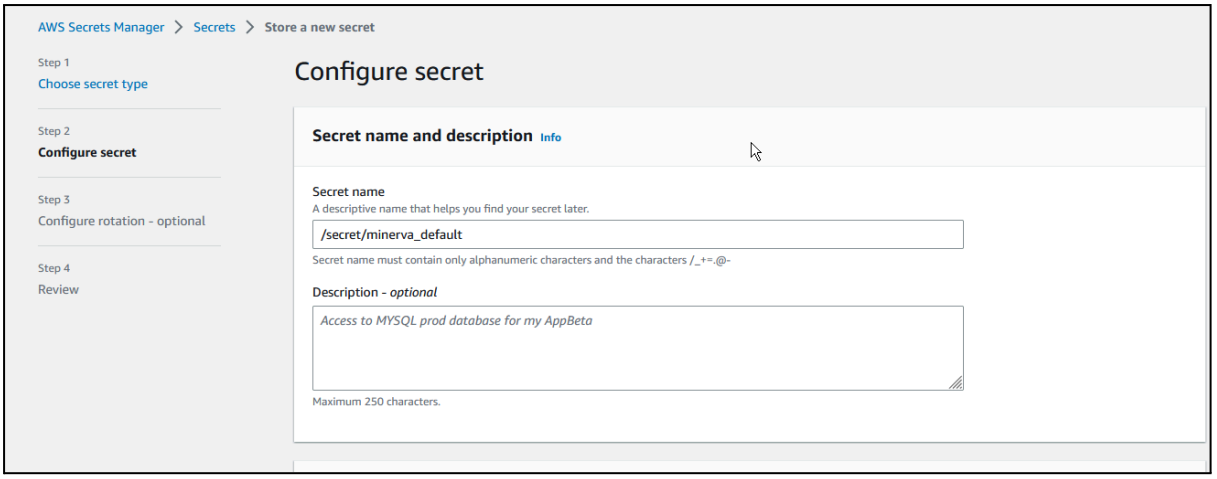

Una vez creado el usuario, se recuperan el accesskey y el secretkey que deben ser guardados en el gestor de contraseñas pertinente.

A continuación se crea un vault desde el secrets manager.

En la primera pantalla hay que seleccionar “Other type of secret” y rellenar los key/value con TODAS las propiedades de la configuración del microservicio tal y como se muestra en la imagen.




En la siguiente pantalla se debe completar el nombre del secret, que debe llevar la nomenclatura /secret/<microservicio>_<perfil> . Un ejemplo es el siguiente




Si todo está correcto, en el paso 4 se puede pulsar en el botón naranja “Store”.

Ahora, se debe rellenar la configuración del microservicio con la región en la cual se encuentra el vault.

Para que el microservicio tenga acceso al servicio Secrets Manager de AWS se pueden poner las credenciales en variables de entorno o darle un role a la máquina si es una EC2 de AWS

- Se crea la carpeta con el nombre del servicio /etc/systemd/system/xxxxx.service.d y dentro un archivo del tipo env.conf (root con permisos 600) para que solo el microservicio en concreto tenga acceso a ese archivo env.conf. Se completa de la siguiente manera:

```
root@dev44:~# cat /etc/systemd/system/zeus.service.d/env.conf
[Service]
Environment=AWS_ACCESS_KEY_ID=AKI[redacted]
Environment=AWS_SECRET_ACCESS_KEY=Udi61h[redacted]JMl7h1CQRm
```

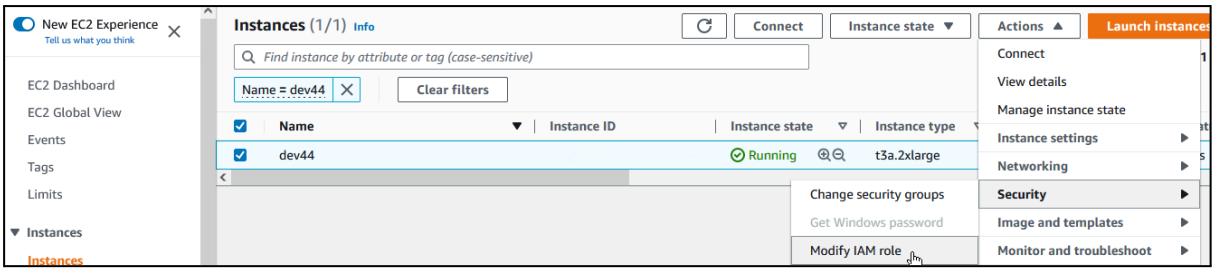
- Se puede evitar la creación del usuario con sus claves y políticas, asignando directamente a la máquina un rol. (Esto solo es posible si es una EC2 del cloud de AWS)

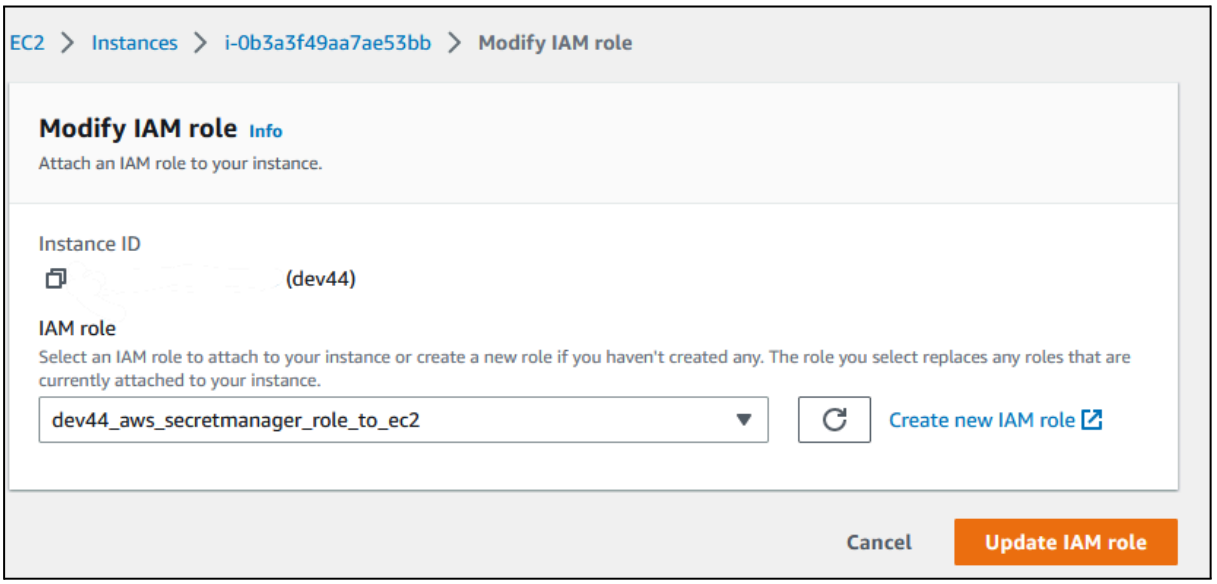
  Para ello, se debe crear un rol con las siguientes políticas y la política predefinida por Amazon con el nombre SecretsManagerReadWrite.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": "secretsmanager:GetSecretValue",
                                                    "Resource":
"arn:aws:secretsmanager:eu-central-1:985556747294:secret:*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
```

```
            "Action": "secretsmanager:ListSecrets",
            "Resource": "*"
        }
    ]
}
```

Una vez creado el rol, seleccionamos la instancia, y le asignamos el rol de la siguiente manera




Una vez realizado esto, sin necesidad de access key o secret key, va a poder acceder la máquina al propio secret manager.

## Azure Key Vault

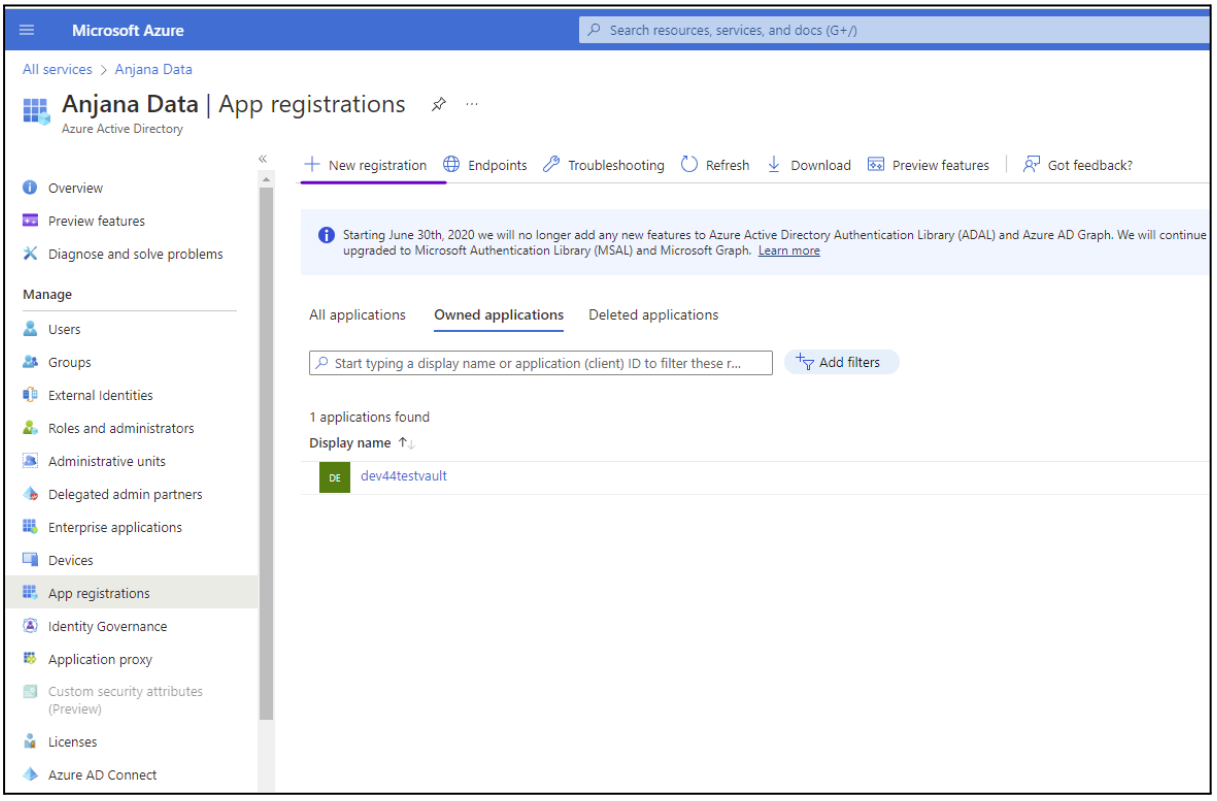
Permite guardar las propiedades sensibles de la configuración del microservicio en Azure Key Vault y recuperarla mediante llamada por api. Para más información consulte la documentación oficial: Azure Key Vault

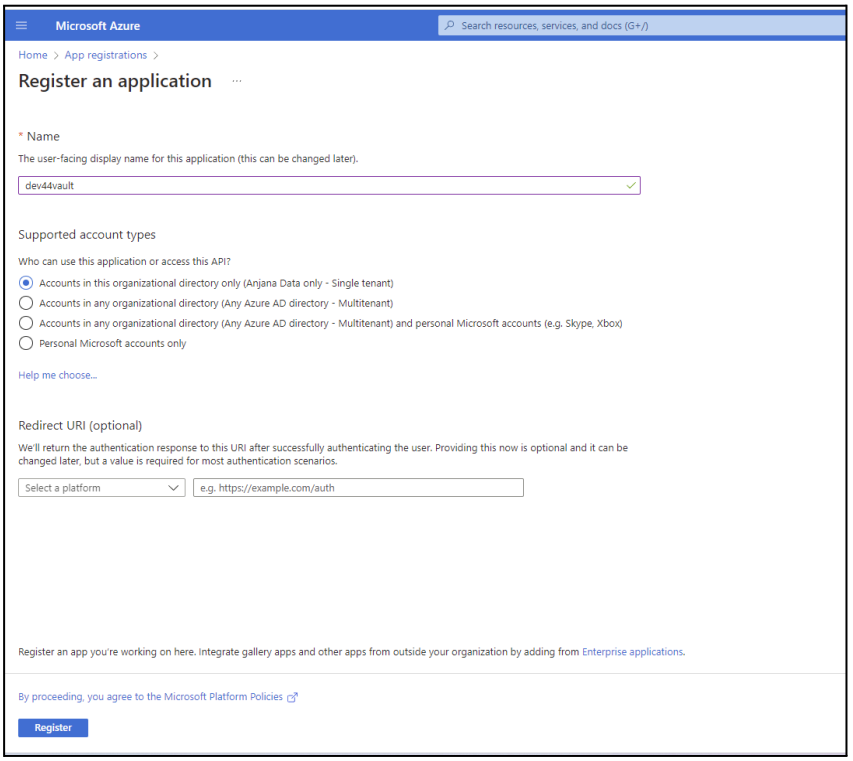
Puntos a tener en cuenta:

- Se debe configurar en cada microservicio que se desee conectar con el Vault, incluidos los plugin. **Salvo en Hermes, que no es posible**.
- A la hora de configurar secretos, se debe tener en cuenta que Azure no permite incluir "/" en el nombre del secreto.
- Se debe configurar las propiedades siguientes en los microservicios deseados con la conexión al Vault de Azure:
    - La configuración de spring.cloud.config.enabled debe estar deshabilitada.
    - La propiedad de secret-keys indica qué secretos del vault se quieren usar (si son varios se usa una lista separada por comas), si no se especifica la propiedad, se traerá todos los secretos del Vault.
    - El secret-key deben coincidir con el nombre del secreto creado en el Vault y se utilizará en las propiedades con el mismo nombre, de tal manera que haga referencia al secret del Vault, por ejemplo:
        - `passwordDatabase: ${secretKeyName}`

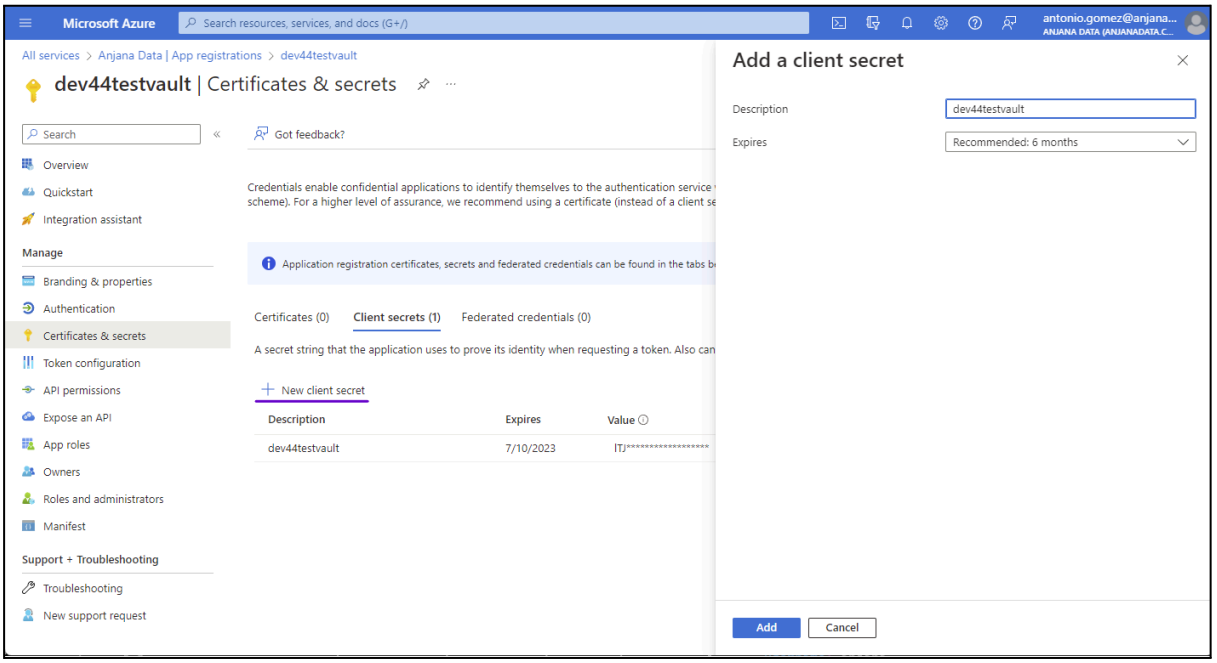
Para la configuración de este método el primer paso sería levantar la infraestructura necesaria:

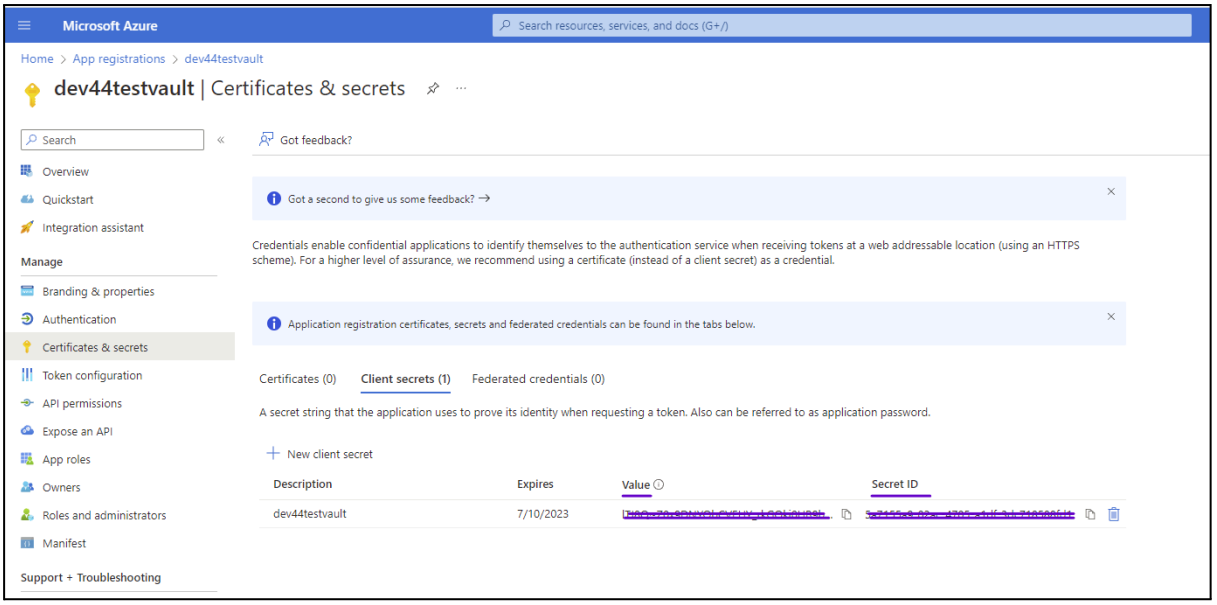
- Se registra la aplicación que va a hacer de cuenta de servicio para acceder al vault



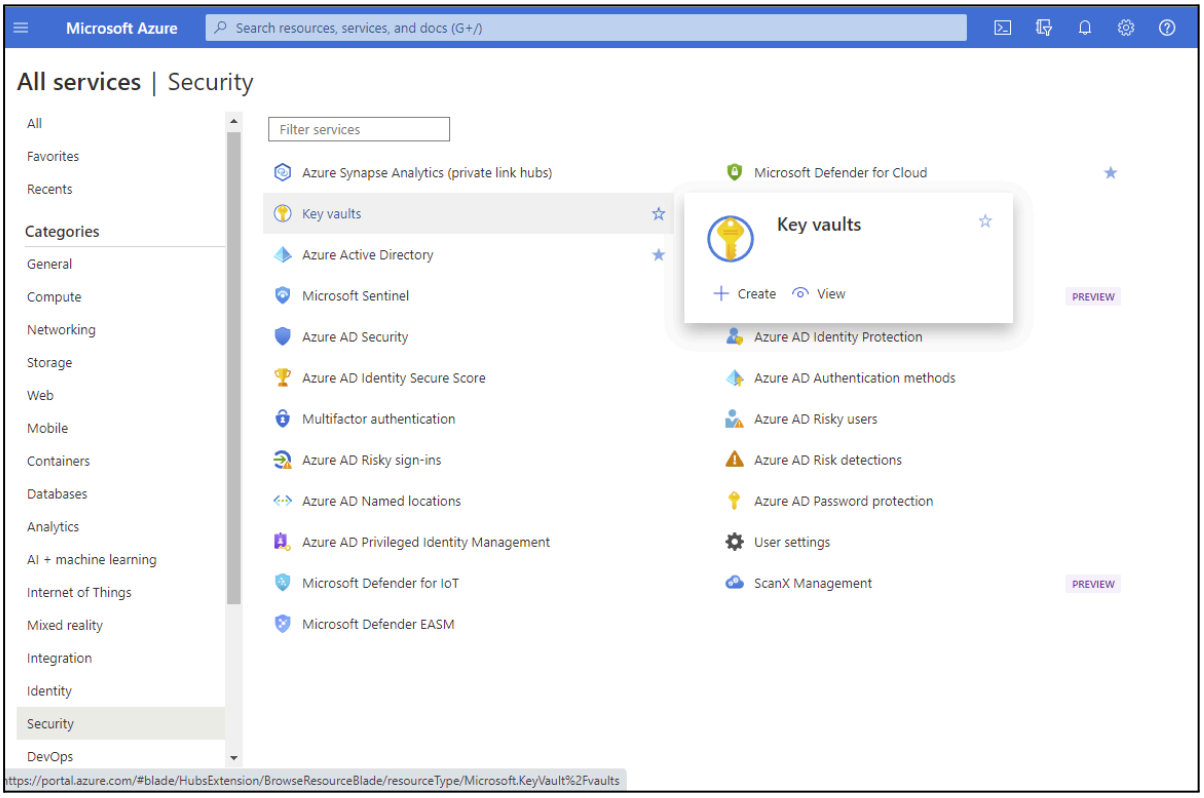


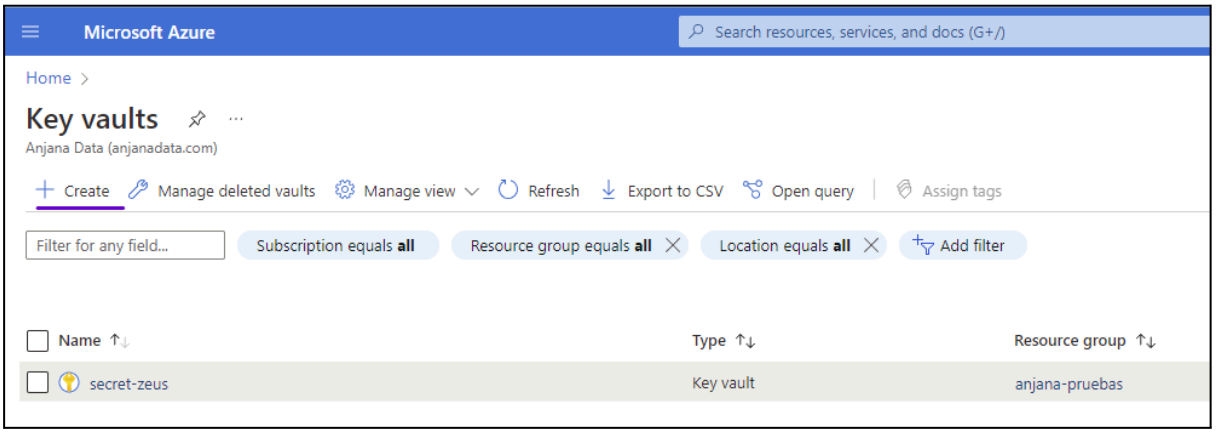

- A continuación se crean los access y secret keys que posteriormente se usarán para la configuración.

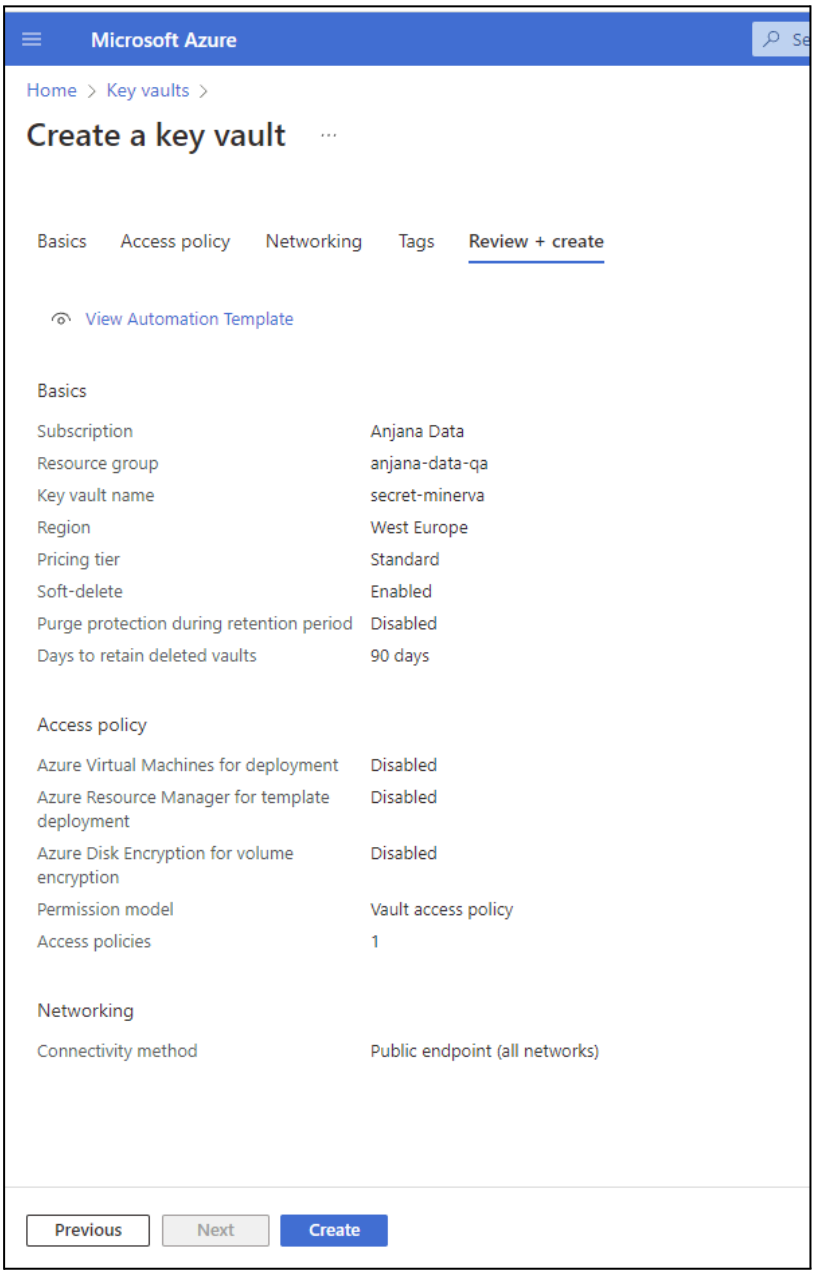




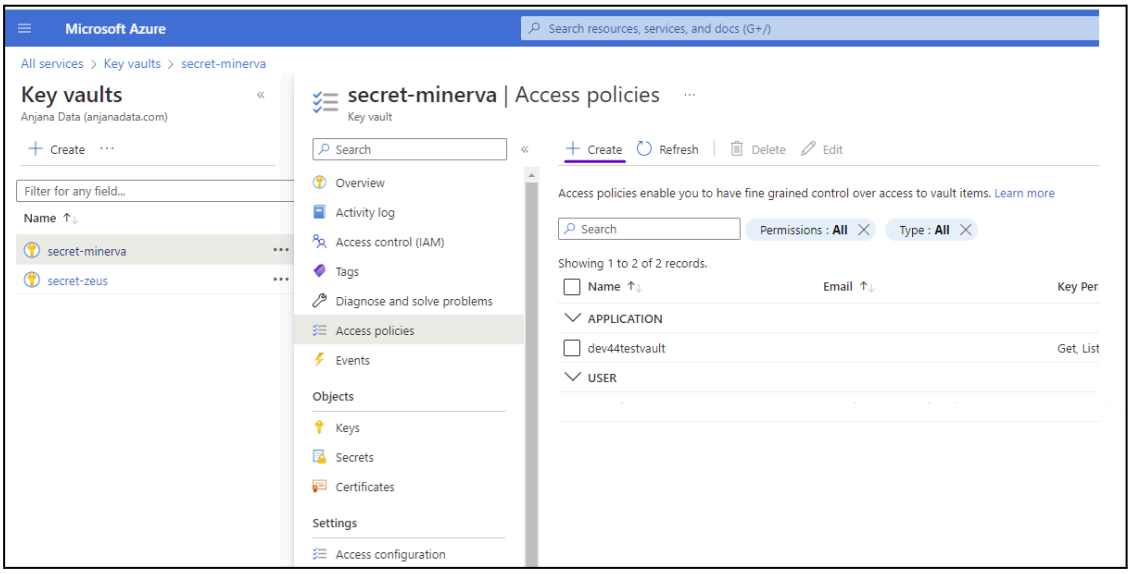

- Después habrá que crear el vault correspondiente al microservicio

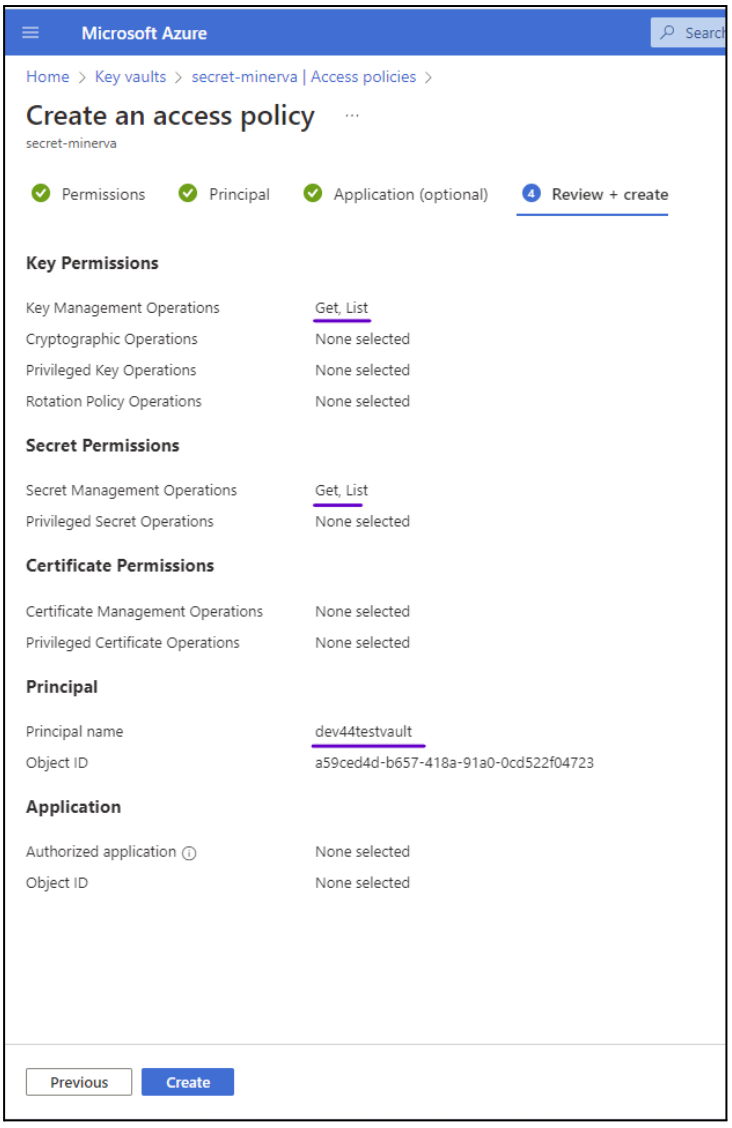



Microsoft Azure
Search resources, services, and docs (G+/)
Home
Key vaults
Anjana Data (anjanadata.com)
Create
Manage deleted vaults
Manage view
Refresh
Export to CSV
Open query
Assign tags
Filter for any field...
Subscription equals all
Resource group equals all
Location equals all
Add filter
Name
Type
Resource group
secret-zeus
Key vault
anjana-pruebas

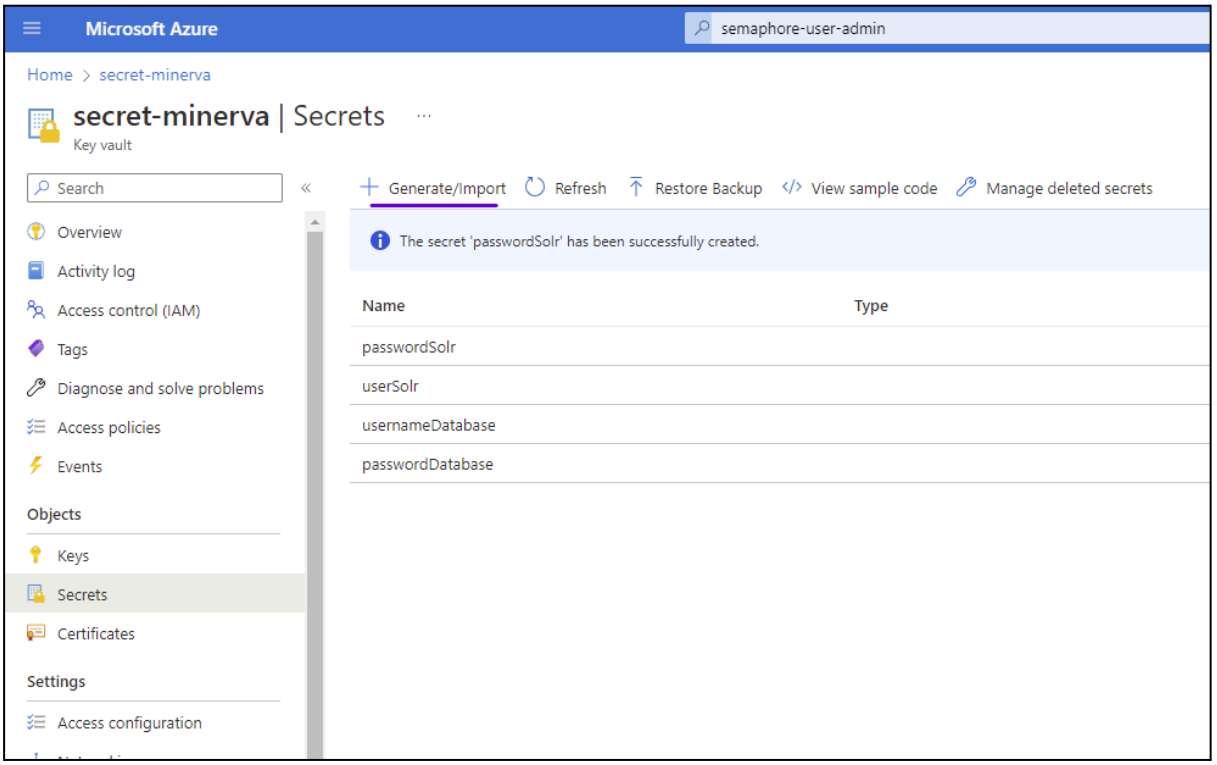

Microsoft Azure
Home
Key vaults
Create a key vault
Basics
Access policy
Networking
Tags
Review + create
View Automation Template
Basics
Subscription
Anjana Data
Resource group
anjana-data-qa
Key vault name
secret-minerva
Region
West Europe
Pricing tier
Standard
Soft-delete
Enabled
Purge protection during retention period
Disabled
Days to retain deleted vaults
90 days
Access policy
Azure Virtual Machines for deployment
Disabled
Azure Resource Manager for template deployment
Disabled
Azure Disk Encryption for volume encryption
Disabled
Permission model
Vault access policy
Access policies
1
Networking
Connectivity method
Public endpoint (all networks)
Previous
Next
Create

- Una vez creado el vault se define la política de acceso



- En este último apartado se crean los secrets a almacenar.




- Después se ajusta la configuración del microservicio que se quiere conectar al vault con las credenciales obtenidas en pasos anteriores y el resto necesario. Estas propiedades se añaden al comienzo del archivo yml de configuración que se esté usando:

```
spring:
  cloud:
    config:
      enabled: false
    azure:
      keyvault:
        secret:
          property-source-enabled: true
          property-sources:
            - credential:
                client-id: <client ID>
                client-secret: <client key>
              endpoint: <Azure-Key-Vault-endpoint>
              case-sensitive: true
              secret-keys: <secret name1>, <secret name2>
              profile:
                tenant-id: <tenant ID>
```

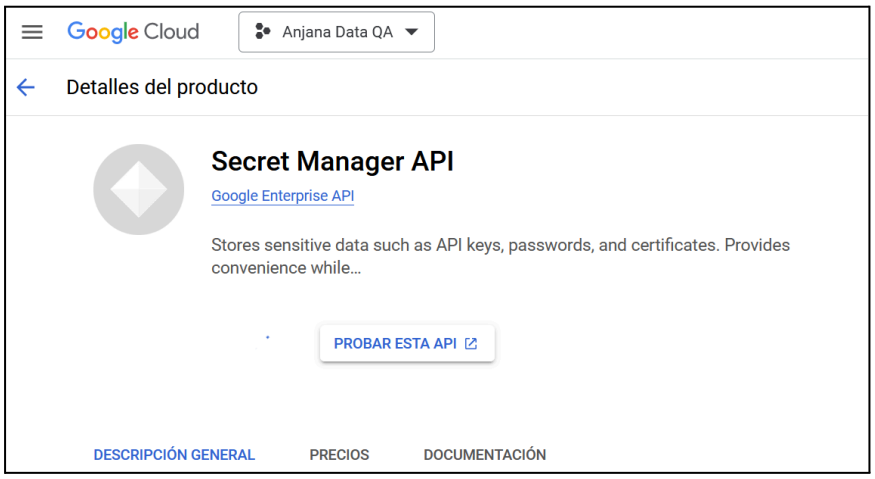
## GCP Secret Management

Permite guardar la configuración sensible en GCP Secret Management y recuperarla mediante llamada por api. Para más información consulte la documentación oficial: GCP Secret Management.

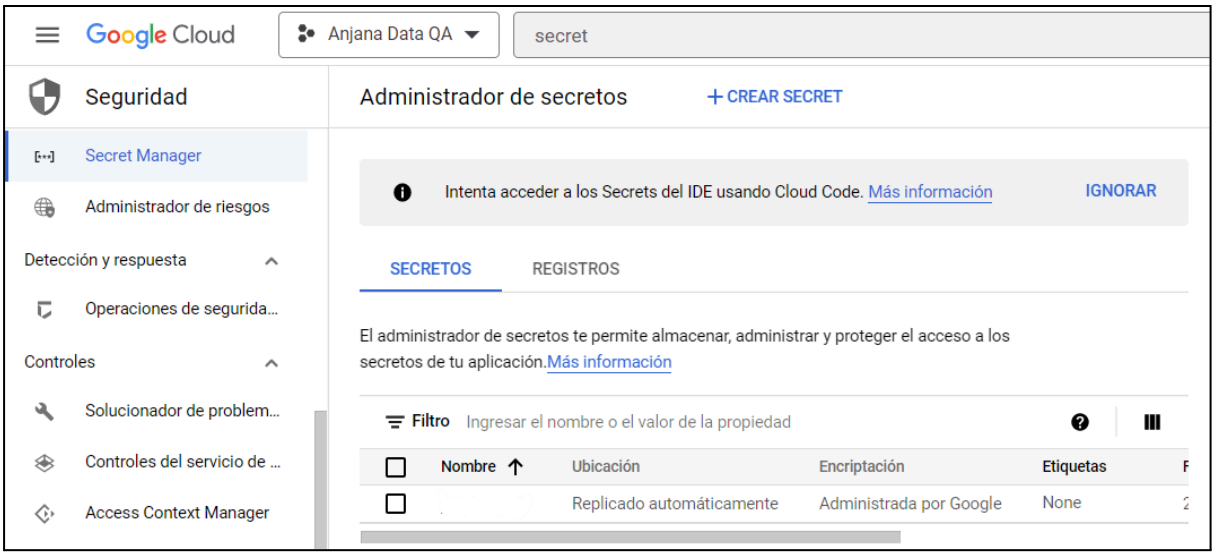
- Se debe configurar en cada microservicio que se desee conectar con el Vault.

- Por cada microservicio se debe crear un proyecto en GCP Secret Management
- Los valores de las variables se leerán con el siguiente formato `${sm://<variable>}` siendo `sm://` el prefijo, se puede omitir con la propiedad `spring.cloud.gcp.secretmanager.secret-name-prefix=sm://`

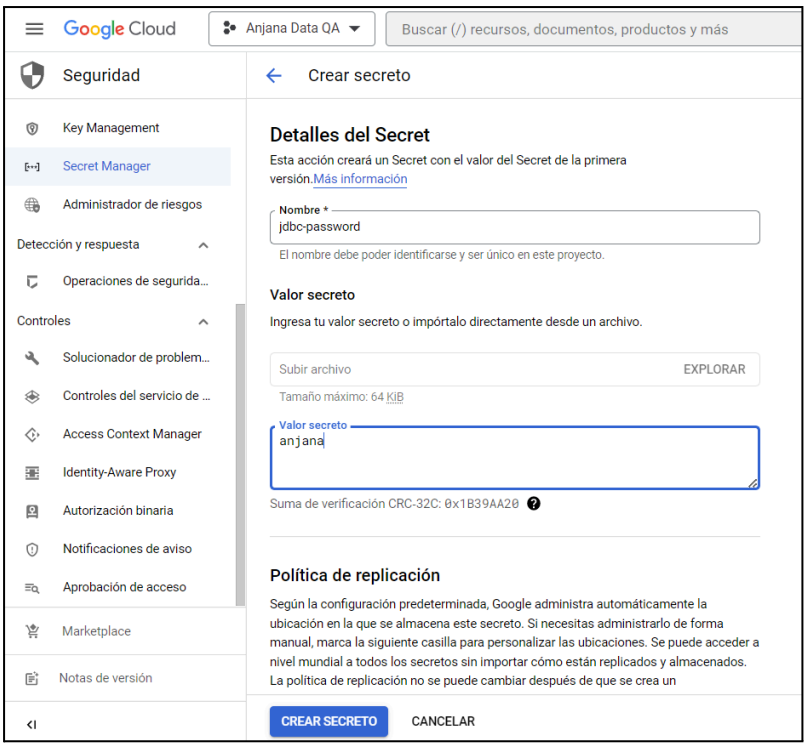
Se debe acceder a GCP y crear el secret manager. Para ello primero se debe habilitar la API si no está habilitada.



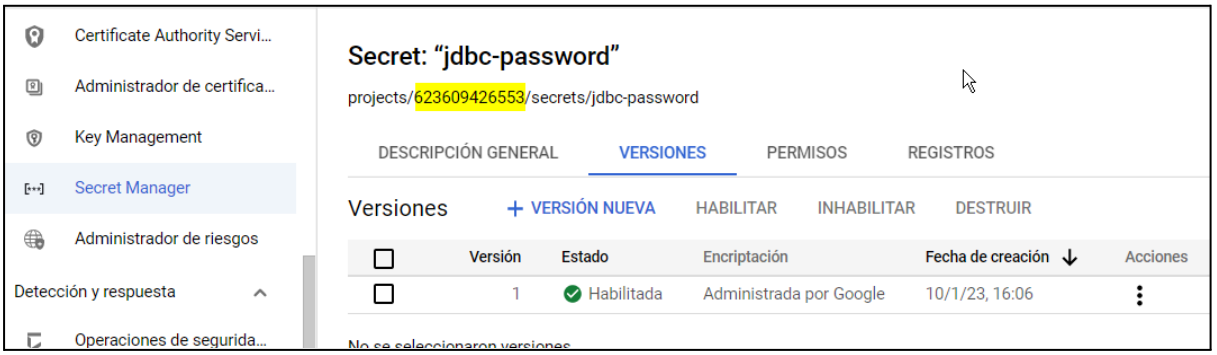

Una vez habilitada, al buscar en el buscador secret manager, debe salir el siguiente menú. Para crear uno nuevo, se selecciona crear secret.



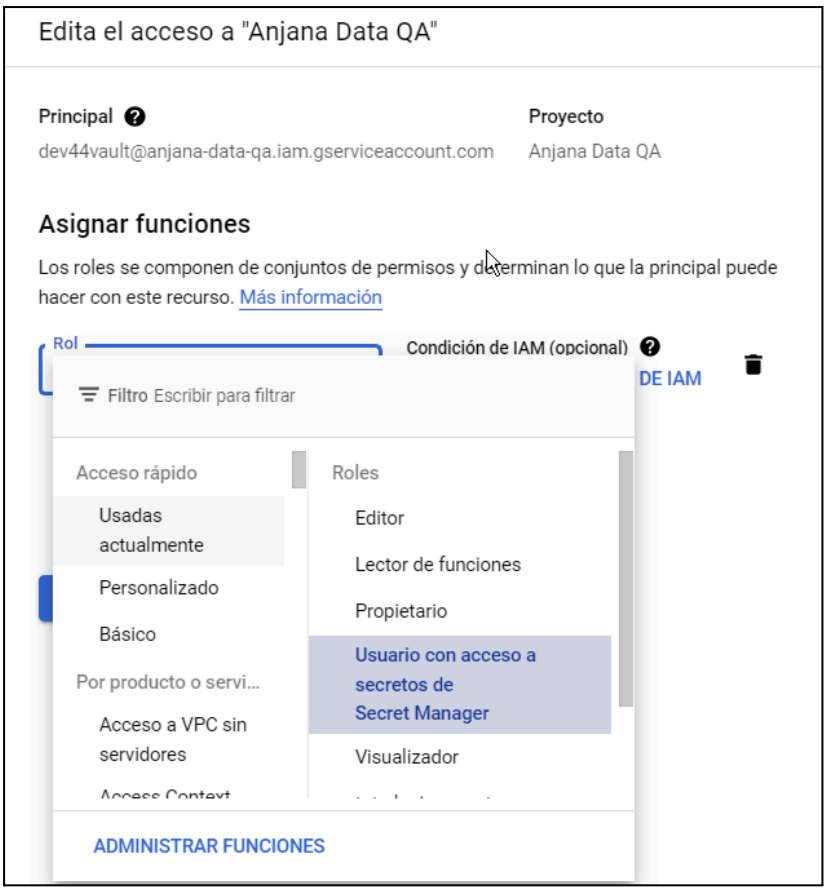

Como nombre, se usará la nomenclatura que se desee. En este caso, se creará el jdbc-password. En valor secreto se debe poner el valor el cual después tiene que poner en la configuración.



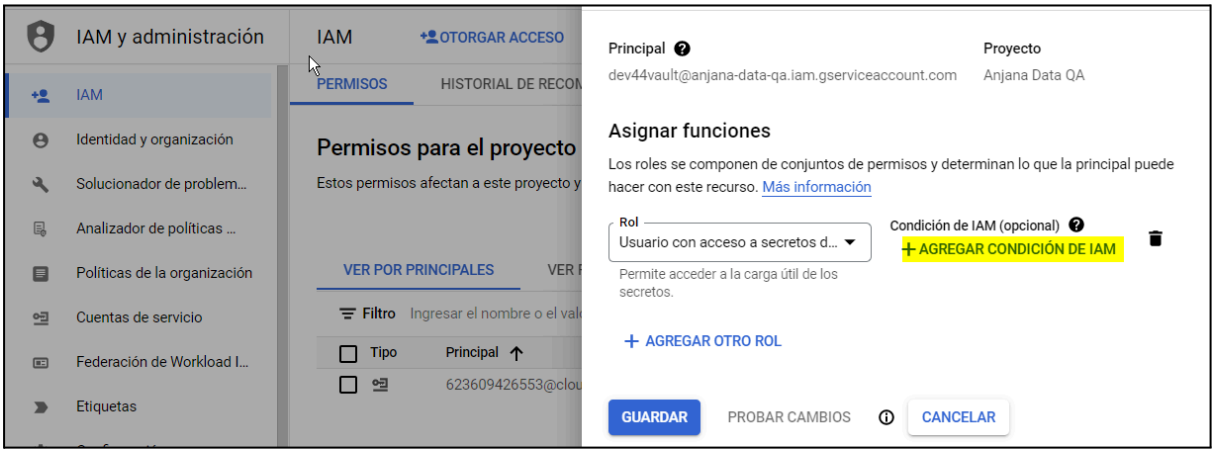

Una vez creado, es necesario copiar el siguiente código, ya que es el ID del secreto que se ha creado.




Ahora se creará la cuenta de servicio, y se le asignará el rol de Usuario con acceso a secretos de Secret Manager.



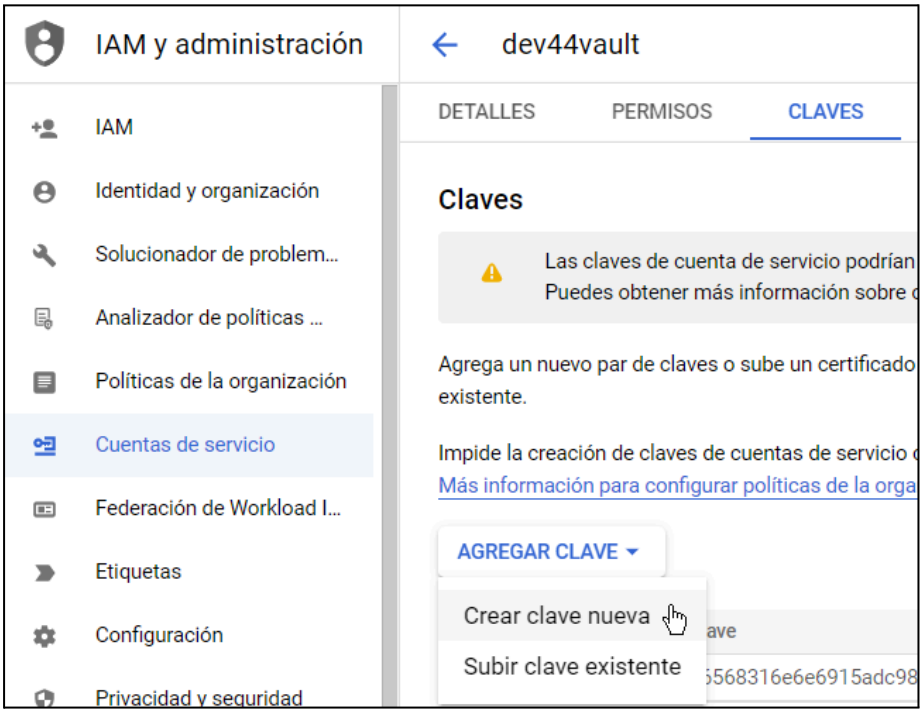

En esta misma pantalla es necesario crear una condición para que solo se lea el secret que se desee. Para ello se selecciona “Agregar condición de IAM”.




Una vez dentro se debe completar de la siguiente manera. El campo valor debe ser la ruta completa del secreto, el cual es la siguiente:

| projects/<id_secreto>/secrets/<nombre_secreto>/versions/latest |
|---|


Para este usuario se deben crear unas claves. Para ello, en el usuario, se selecciona el menú claves y se pulsa en “Agregar clave” para crear una nueva.




Cuando se cree, se debe descargar como json para poderlo añadir donde se encuentra el jar del microservicio que necesite del secreto.

Una vez realizado todo lo anterior, se añade el siguiente código en el archivo de configuración del microservicio que se desee, cambiando el project-id y la localización del archivo.

```
spring.cloud.gcp:
 core:
   enabled: true
 secretmanager:
   enabled: true
      project-id:  623609xx6553  # Código del proyecto gcp secret
management
```

```
   credentials:
          location:  file:*******.json  # Fichero que contiene las
credenciales gcp
```

También, es necesario cambiar el campo que se desee sustituir por una variable con el formato ${sm://<nombre_secret_gcp}. Un ejemplo es el siguiente:

```
spring:
  datasource:
    #If schema changes, change hibernate.default_schema few lines after
    url: jdbc:postgresql://rdbservice:5432/anjana?currentSchema=zeus
    username:
    password: ${sm://jdbc-password}
```

## HA config

Para configurar HA en Edusa, se tiene que modificar el comando de arranque de java en cada microservicio de Anjana añadiendo en el parámetro spring.uri todas las URLs de los servidores donde Edusa está escuchando. No es posible añadir autopurgado de servidores de configuración caídois en esta versión.

```
[ec2-user@ip-10-150-100-245 ~]$ sudo cat /etc/systemd/system/hecate.service
[Unit]
Description=Anjana hecate server
Requires=network.target
After=network.target  edusa.service

[Service]
Environment=HOSTNAME=localhost
Environment=JAVA_HOME=/usr/lib/jvm/jre-1.8.0-openjdk
Environment=HECATE_REPLICAS=http://10.150.100.245:50761/eureka,http://10.150.100.211:50761/eureka
WorkingDirectory=/opt/hecate
ExecStart=/opt/hecate/hecatelauncher /usr/lib/jvm/jre-1.8.0-openjdk/bin/java -XX:+UseG1GC -XX:+UseStringDeduplication -Xmx256
aagent:/opt/common/xjar-agent-hibernate.jar -jar /opt/hecate/hecate.jar --spring.config.import=optional:configserver:http://1
100.245:8888,optional:configserver:http://10.150.100.211:8888 --spring.profiles.active=default --spring.cloud.config.failFast=

# COPY THE NEXT FLAG IN ExecStart AFTER java TO ACTIVATE DEBUG
# -agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=0.0.0.0:5053

User=anjana


Type=simple
Restart=on-failure
RestartSec=30

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

## Variables de entorno para ocultar credenciales

Se pueden usar variables de entorno para ocultar información sensible en los yamls de configuración por ejemplo.

- Se crea la carpeta con el nombre del servicio /etc/systemd/system/xxxxx.service.d y dentro un archivo del tipo env.conf (root con permisos 600) de esta forma solo el microservicio tiene acceso a ese archivo env.conf.

```
root@dev44:~# cat /etc/systemd/system/zeus.service.d/env.conf
[Service]
Environment=AWS_ACCESS_KEY_ID=AKIA6K56CQQPAIJP73O4
Environment=AWS_SECRET_ACCESS_KEY=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
```

```
[Service]
Environment=<KEY>=<VALUE>
Environment=<KEY2>=<VALUE2>
```

- En el yaml de configuración del microservicio se puede poner entre llaves y con un simbolo dolar previo ${KEY}

```
spring:
  datasource:
    username: anjana
    password: ${BBDD_PASSWORD}
    ...
```

# Hecate

- ${HECATE_REPLICAS} en el descriptor de servicio como variable de entorno o directamente en el fichero de configuración. Puede ser él mismo, otro nodo o un array de hecateservers
    - http://hecateserver:50761/eureka
    - http://hecateserver1:50761/eureka, http://hecateserver2:50762/eureka

Ejemplo (no necesario si se setea el valor HECATE_REPLICAS) :

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://hecateserver:50761/eureka
```

## HA config

Para configurar HA en Hecate, se tiene que modificar los microservicios de Anjana que se registran en Hecate (Kerno, Minerva, Hermes, Viator, Portuno, Tot, Drittesta y Zeus), cambiando el valor de eureka.client.serviceUrl.defaultZone. En el descriptor de servicio de Hecate, se tiene que cambiar el valor de la variable HECATE_REPLICAS, como se describe en el apartado anterior.

Ejemplo:

```
eureka:
  client:
    serviceUrl:
      defaultZone: http://back1.anjanadata.com:50761/eureka,http://back2.anjanadata.com:50761/eureka
```

Lo mismo pero configurado en el descriptor de servicio:

```
[Unit]
Description=Anjana hecate server
Requires=network.target
After=network.target  edusa.service

[Service]
Environment=HOSTNAME=hecateserver
Environment=HECATE_REPLICAS=http://back1.anjanadata.com:50761/eureka,http://back2.anjanadata.com:50761/eureka
WorkingDirectory=/opt/hecate
ExecStart=java -Xmx256m -javaagent:/opt/common/xjar-agent-hibernate.jar -jar /opt/hecate/hecate.jar --spring.cloud
=http://back1.anjanadata.com:8888,http://back2.anjanadata.com:8888 --spring.profiles.active=default --spring.cloud
lFast=true
User=anjana
Type=simple
```

## Drittesta, Hermes, Kerno, Minerva, Portuno, Tot, Zeus, Viator

Tienen las propiedades de configuración técnica explicadas en su propio ejemplo en el artefacto de YAMLs, para poder copiar y editar. A continuación se detallan, de estos, los módulos con configuración especial:

### Zeus

La parte de seguridad tiene dos partes (security.authentication y security.authorization) :

- security.authentication.ldap.user-authentication: autenticación del usuario sobre LDAP. USER_PASSWORD, USER_PASSWORD_ENC y USER_CONNECTION son los valores permitidos.
- security.authentication.ldap.url: URL y puerto de LDAP.
- security.authentication.ldap.base-dn: DN base del esquema de LDAP.
- security.authentication.ldap.user-search-filter: filtro para la búsqueda de usuarios por el atributo seleccionado en LDAP.
- security.authentication.ldap.user-structural-class: clase estructural de LDAP para los usuarios.
- security.authentication.ldap.user-search-attribute: atributo para la búsqueda de usuarios en LDAP.
- security.authentication.ldap.name-search-attribute: nombre de la propiedad que corresponde con el atributo nombre del usuario en LDAP, por defecto: `givenName`
- security.authentication.ldap.surname-search-attribute: nombre de la propiedad que corresponde con el atributo apellido del usuario en los usuarios de LDAP, por defecto: `sn`
- security.authentication.ldap.employee-search-attribute: nombre de la propiedad que corresponde con el atributo número de empleado en LDAP, por defecto: *employeeNumber*
- security.authentication.ldap.title-search-attribute: nombre de la propiedad que corresponde con el atributo título del usuario en LDAP, por defecto: `title`
- security.authentication.ldap.phone-search-attribute: nombre de la propiedad que corresponde con el atributo teléfono del usuario en LDAP, por defecto: `telephoneNumber`
- security.authentication.ldap.connection-user-dn: DN del usuario administrador de LDAP que se va a usar para establecer la conexión.
- security.authentication.ldap.connection-user-password:contraseña del usuario administrador de LDAP.

- security.session.expiredtimeminutes: tiempo de expiración de la sesión en minutos.
- security.jwt.expiration-timeout-minutes: tiempo de expiración del token en minutos.
- security.jwt.secret-key: clave para codificación del token.
- security.authentication.oidc.providers: agrupación de providers a los que Zeus tendrá acceso.
- security.authentication.oidc.providers.<name>: clave para darle nombre a cada una de las conexiones.
- security.authentication.oidc.providers.<name>.name: nombre para mostrar en el Portal y Portuno.
- security.authentication.oidc.providers.<name>.authorize-url: URL del proveedor para autorizar usuarios de Anjana Portal. (Utiliza variables en la URL).
- security.authentication.oidc.providers.<name>.authorize-url-portuno: URL del proveedor para autorizar usuarios de Portuno. (Utiliza variables en la URL).
- security.authentication.oidc.providers.<name>.token-url: URL propia del proveedor para gestionar la creación de tokens.
- security.authentication.oidc.providers.<name>.scopes: scopes de autenticación propios del proveedor.
- security.authentication.oidc.providers.<name>.client-id: identificador de autenticación en el proveedor.
- security.authentication.oidc.providers.<name>.client-secret: secreto de autenticación en el proveedor.
- security.authentication.oidc.providers.<name>.client-authentication-method: método de autenticación propio del proveedor.
- security.authentication.oidc.providers.<name>.redirect-uri: URI a la que tiene que redirigir el navegador cuando se produce un login exitoso en un proveedor externo de autenticación en el portal.
- security.authentication.oidc.providers.<name>.redirect-uri-portuno: URI a la que tiene que redirigir el navegador cuando se produce un login exitoso en un proveedor externo de autenticación en el portal administrativo (Portuno).
- security.authentication.oidc.providers.<name>.username-claim: campo donde se encuentra el nombre de usuario en el proveedor.
- security.authentication.oidc.providers.<name>.type: tipo de proveedor.
- security.authentication.oidc.providers.{proveedor}.workflowType: IMPLICIT
    - Con valor IMPLICIT seguirá este flujo definido por OAuth2 (único flujo previo a la v4.5).
    - Con valor AUTHORIZATION_CODE seguirá este flujo definido por OAuth2
        - En caso de poner este valor se necesitan configurar dos propiedades más a la misma altura de workflowType:
            - authorizeServer: url del endpoint del servidor de autorización que devuelve información del usuario en base a un 'authorization code'
                - AWS: `https://<your-user-pool-domain>/oauth2/userInfo` Doc AWS
                - GCP: Doc GCP
                - Azure: https://graph.microsoft.com/oidc/userinfo Doc Azure
                - Auth0: https://<auth-server>/userinfo Doc Auth0
            - userNameProperty: propiedad donde se encuentra el nombre del usuario en el endpoint anteriormente mencionado. Suele coincidir con la propiedad usernameClaim ya configurada para el proveedor.

    - AWS: username
    - GCP: email
    - Azure: email
    - Auth0:email
- security.authorization.ldap.url: URL y puerto de LDAP.
- security.authorization.ldap.base-dn: DN base del esquema de LDAP.
- security.authorization.ldap.connection-user-dn: DN del usuario administrador de LDAP que se va a usar para establecer la conexión.
- security.authorization.ldap.connection-user-password:contraseña del usuario administrador de LDAP.
- security.authorization.ldap.user-structural-class: clase estructural de LDAP para los usuarios.
- security.authorization.ldap.user-search-attribute: atributo para la búsqueda de usuarios en LDAP.
- security.authorization.ldap.group-structural-class: clase estructural de LDAP para los grupos.
- security.authorization.ldap.group-search-attribute: atributo para la búsqueda de grupos en LDAP.
- security.authorization.ldap.member-of-patch-filter: filtro para especificar la membresía de un usuario en un grupo.
- security.authorization.ldap.membership-user-group: atributo de pertenencia de un usuario en un grupo.
- security.authorization.ldap.root-ous: unidades organizativas raíz cuyo nombre se ha de excluir en la composición del nombre de la lista de unidades organizativas en su recuperación.
- security.authorization.ldap.ou-structural-class: clase estructural de LDAP para las unidades organizativas.
- security.authorization.ldap.ou-search-attribute: atributo para la búsqueda de unidades organizativas en LDAP.
- security.authorization.{providerType}.enabled: flag para indicar que se habilita la autorización en este tipo de provider.
- security.authorization.{providerType}.providers.{providerName}.groupOrgUnitSeparator: propiedad que permite configurar el separador de unidades organizativas.
- security.authorization.{providerType}.providers.{providerName}.roleOrgUnitSeparator: propiedad que permite configurar el separador de roles.
- security.authorization.{providerType}.providers.{providerName}.groupPrefix: propiedad que permite poder configurar el borrado de un prefijo al obtener las unidades organizativas en en el proovedor. Si no se incluye, por defecto no borrará nada.

Las integraciones de autorización y autenticación se detallan en los documentos específicos para tal fin:

- Anjana Data 4.x - DS - Integración Azure
- Anjana Data 4.x - DS - Integración AWS
- Anjana Data 4.x - DS - Integración GCP
- Anjana Data 4.x - DS - Integración LDAP
- Anjana Data 4.x - DS - Integración OKTA

## Drittesta

### Generación de claves de encriptación

El NIST recomienda usar claves de como mínimo 2048 bits, que son las que proporciona Anjana. Aún así, si el cliente lo desea, podrá generar un nuevo par de claves RSA para incrementar la seguridad de la encriptación hasta 4096 bits creando las claves de forma manual; de este modo el cliente será el creador y validador de la seguridad y unicidad de las claves. Una vez generadas, la pública la deberá proporcionar a Anjana para configurar su instalación. La privada se establecerá en el proyecto de **Drittesta** con la nueva propiedad del yaml **anjana.privateKey** teniendo en cuenta que:

- Representa la clave privada del cliente para la comunicación entre los servicios de licencia.
- Será una propiedad obligatoria.
- Podrá ser generada por el cliente de la siguiente manera:
    - Con el endpoint habilitado en **Drittesta** para ello (clave de 2048 bits):

```
curl --location --request GET
'http://{{host}}:{{port}}/api/license/keys'
```

Donde {{host}} es la dirección de la máquina y {{port}} el puerto de **Drittesta.** La respuesta es un json con el siguiente formato:

```
{
  "publicKey": clave pública,
  "privateKey": clave privada
}
```

    - O bien,

En una consola UNIX ejecutar las instrucciones:

- ssh-keygen -m PKCS8 -t rsa -b 4096
- openssl rsa -in nombreClave -pubout

La primera genera las claves en un directorio local. Descripción de los parámetros:

- **-m** indica el tipo de algoritmo que es obligatorio que sea PKCS8
- **-t** indica el tipo de clave que es RSA obligatoriamente.
- **-b** indica el tamaño de la clave. El cliente puede configurar el tamaño que desee. Los tamaños más comunes son 1024, 2048, 3072 o 4096 bits, siendo 4096 el tamaño máximo.

La segunda instrucción devuelve la clave pública en el formato correcto para usarse en Anjana. El parámetro “nombreClave” será el nombre de la clave que hemos dado en el paso anterior.

## Tot

La comunicación entre Tot y los plugins se realiza de forma cifrada, encriptando el mensaje que se envía y se recibe. Por ello, es necesario informar de cuál es la clave privada de Tot y de la clave pública de los plugins (usarán todos la misma).

- tot.privateKey: Clave privada de Tot

- tot.plugin.publicKey: Clave pública de los plugins
- anjana.async.pool.maxPoolSize: Tamaño máximo del pool de hilos (valor por defecto 3)
- anjana.async.pool.corePoolSize: Tamaño de cores usado (valor por defecto 3)
- anjana.async.pool.queueCapacity: Máximo nº de hilos en cola (valor por defecto 1500)

# Tot plugins

En cada plugin hay que añadir la configuración de claves para cifrar la comunicación con Tot.

- totplugin.encryption.privateKey: Clave privada del plugin
- totplugin.encryption.tot.publicKey: Clave pública de Tot

Para generar un nuevo par de claves válido se recomienda usar los siguientes comandos. El primer comando usa el algoritmo RSA con codificación PKCS8 para generar las claves.

```
ssh-keygen -m PKCS8 -t rsa
```

Es importante no introducir ningún texto cuando solicite una passphrase, solo pulsar Enter, para que no devuelva la propia clave encriptada.

El comando anterior generará dos archivos key y key.pub (key siendo el nombre introducido para el fichero de salida). El archivo key.pub contendrá la clave pública y el archivo key la privada. Para obtener la clave pública en el mismo formato que la privada, se debe introducir el siguiente comando, que formateará la clave pública y la dejará lista en el nuevo fichero keyPublic, para usarla en los plugins.

```
openssl rsa -in key2 -pubout -out keyPublic
```

En cada plugin existe configuración para poder registrarte en múltiples Tots y usarlo como proxy para registrarse en eureka.

totplugin.server.urls: Lista de urls de los Tot en los que el plugin se registraría

eureka.client.serviceUrl.defaultZone: Se debe poner alguno de los Tot que se configuraron en la propiedad de arriba.

Para el registro en tot se requiere indicar qué funcionalidades ofrece y sobre que tripletas, dicha configuración se incluye en la variable totplugin.aris

Además de la configuración común, cada plugin tiene un documento para desplegar y configurar (con ejemplos de configuración).

# Apache2

## Cache

Para activar el caché del navegador y que el frontend no tenga que recargar cada vez todos los archivos Javascript y Html es necesario introducir, tanto en la configuración SSL como la no SSL, el siguiente código:

```
<filesMatch "\.(html|htm|js|css)$">
   FileETag MTime Size
   <ifModule mod_headers.c>
      Header unset Cache-Control
      Header unset Pragma
      Header unset Expires
      Header set Cache-Control "public, max-age=43200, must-revalidate"
   </ifModule>
</filesMatch>
```

Se determina una vida útil para ese caché de 12h (**43200**) que es lo máximo que se recomienda desde Anjana Data.

Si, por el contrario, se necesita desactivar el caché permanentemente, la configuración debe ser la siguiente:

```
<filesMatch "\.(html|htm|js|css)$">
   FileETag None
   <ifModule mod_headers.c>
      Header unset ETag
      Header set Cache-Control "max-age=0, no-cache, no-store, must-revalidate"
      Header set Pragma "no-cache"
      Header set Expires "Wed, 11 Jan 1984 05:00:00 GMT"
   </ifModule>
 </filesMatch>
```

## HA config

Para configurar HA en Apache, se tiene que modificar las entradas Location en la configuración de apache2, añadiendo las correspondientes IP/DNS de destino para los nodos existentes.

Por ejemplo, para añadir nodos de minio, se necesita cambiar el siguiente ProxyPass.

```
<Proxy balancer://cdnbackends>
    BalancerMember http://s3servicenode1:9000/cdn
    BalancerMember http://s3servicenode2:9000/cdn

    ProxySet lbmethod=bytraffic
</Proxy>
<Location /cdn>
  ProxyPass balancer://cdnbackends
</Location>
```

Lo mismo para Viator

```
<Proxy balancer://gatewaybackends>
      BalancerMember http://viatorservernode1:8085
      BalancerMember http://viatorservernode2:8085

      ProxySet lbmethod=bytraffic
</Proxy>
<Location /gateway>
      ProxyPass balancer://gatewaybackends
</Location>
```

Igual para Portuno

```
<Proxy balancer://portunobackends>
      BalancerMember http://portunoserver1:8998/portuno
      BalancerMember http://portunoserver2:8998/portuno

      ProxySet lbmethod=bytraffic
</Proxy>
<Location /portuno>
      ProxyPass balancer://portunobackends
</Location>
```

Este bloque de configuración de Apache permite acceder a Swagger a través del puerto 80 o 443 <anjana_url>/swagger/swagger-ui.html. Este cambio es necesario en los archivos de configuración anjana.conf y anjana_ssl.conf. Sin esta configuración Swagger sólo es accesible a través del puerto de Viator <anjana_url>:8085/swagger/swagger-ui.html

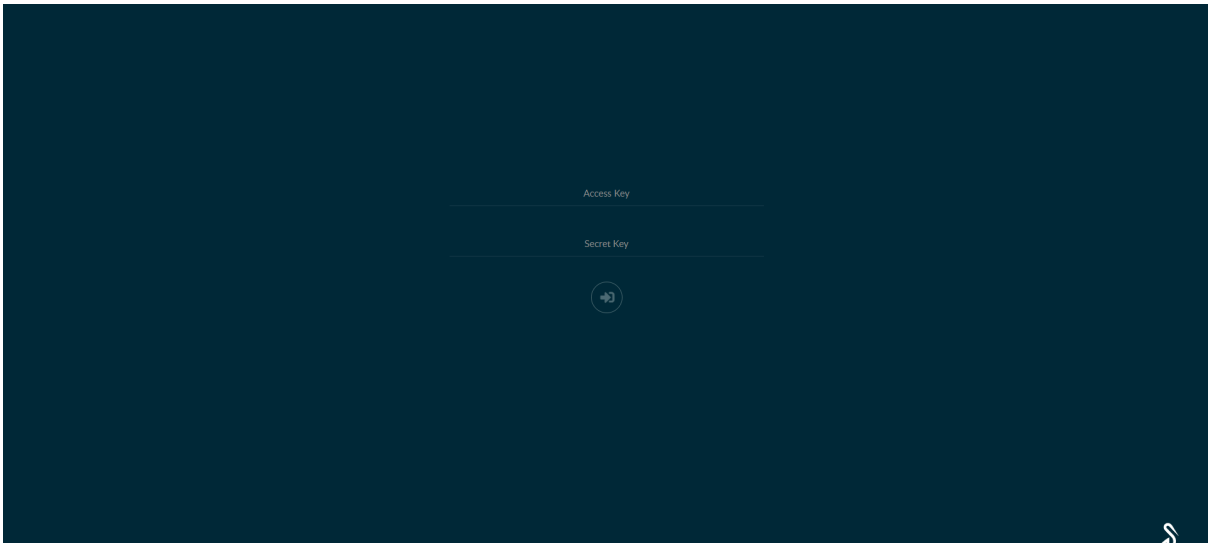
```
<Proxy balancer://swaggerbackends>
     BalancerMember http://viatorservernode1:8085/swagger
     ProxySet lbmethod=bytraffic
</Proxy>
<Location "/swagger">
     ProxyPass "balancer://swaggerbackends"
     ProxyPassReverse "balancer://swaggerbackends"
</Location>

<Proxy balancer://swaggerapibackends>
     BalancerMember http://viatorservernode1:8085/v3/api-docs
     ProxySet lbmethod=bytraffic
</Proxy>
<Location "/v3/api-docs">
     ProxyPass "balancer://swaggerapibackends"
     ProxyPassReverse "balancer://swaggerapibackends"
</Location>
```

# Configuración Frontend

## Minio

### ¿Qué es Minio?

Anjana usa almacenamiento de ficheros en servidores cloud compatible con Amazon S3. Como almacén de datos, Minio permite guardar datos no estructurados como fotos, videos, backups o imágenes de contenedores.



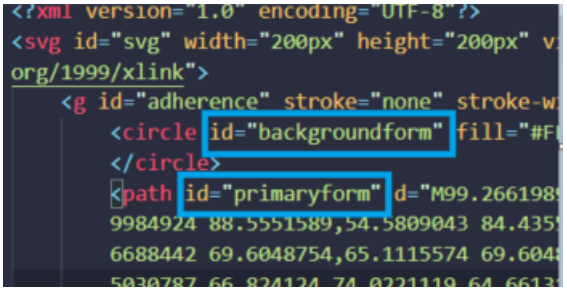

Minio funciona usando contenedores llamados **buckets.**

## Estructura de Minio básica en Anjana Data

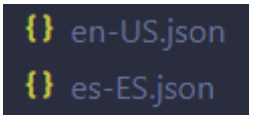
Para el correcto funcionamiento de Anjana es necesario crear los siguientes buckets:

- **imports** -> Para almacenar los ficheros excel para importación masiva de objetos
- **dsa** -> Para almacenar los contratos de los DSA y cualquiera de los ficheros adjuntos a las entidades o relaciones de Anjana.
- **cdn** -> Para almacenar las imágenes de la aplicación y los ficheros de traducción de los idiomas utilizados en la misma.
    - images
        - attribute-icons: Para almacenar los iconos propios del core de la aplicación (no personalizables).
        - languages-icons: Iconos de idiomas
        - subtype-icons: Para almacenar los iconos de subtipo personalizables. Estos iconos están hechos con una sola forma fundida (primaryform) y tiene un borde que será el que indica el estado del objeto en la pantalla de linaje.

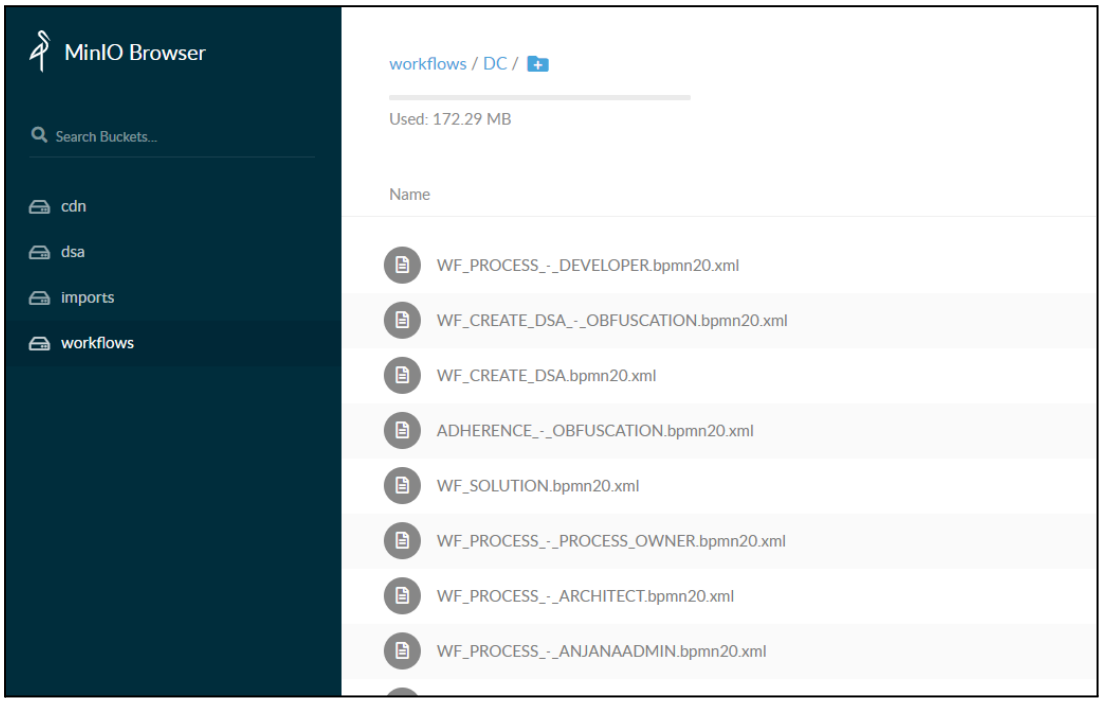
          Para que esto funcione, el círculo de fondo que llevan los iconos de subtipo tiene el id=”backgroundForm”, Y la forma del icono es un path con el id=”primaryForm”, sin estos ids no funcionarán los iconos en el linaje.




    - i18n -> Donde se guardan los ficheros que se generan, desde Portuno, con las traducciones para permitir en Anjana el multiidioma




- **workflows** -> Para almacenar los xml correspondientes a la configuración de los workflows que se van a ejecutar en Anjana. Este bucket contiene dos buckets: BG y DC




- **textarea ->** Para almacenar las imágenes que se suben en los atributos del tipo ENRICHED_TEXT_AREA.

## CDN - Precarga estática

Tienen que ser precargados los ficheros estáticos en el bucket CDN para que la aplicación funcione correctamente (iconos, imágenes, ficheros de traducción, etc).

```
wget
https://<user>:<pass>@releases.anjanadata.org/repository/releasesraw/com/anjana/
cdn/<version>/cdn-<version>.tar

tar -xzpvf cdn-<version>.tar -C <path de la carpeta cdn>
```

## Notas

Desde Anjana v.4.2.7, se usa una nueva versión de Minio, desde la cual la consola ha sido desenlazada de la API y es necesario añadir un vhost en el fichero /opt/apache/conf/httpd.conf para tener acceso.

```
<VirtualHost *:8081>
    ServerAdmin admin@test.com
    ServerName test.com
    ServerAlias www.test.com

  <Proxy balancer://miniobackends>
        BalancerMember http://s3servicenode1:9001/
        ProxySet lbmethod=bytraffic
  </Proxy>
  <Location />
        ProxyPass http://s3service_node1:9001/
  </Location>
</VirtualHost>
```