UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Sistemas Informáticos y Computación

Implementación de un sistema homogéneo de autenticación y autorización en un clúster Kubernetes en el ámbito de la investigación

Trabajo Fin de Máster

Máster Universitario en Computación en la Nube y de Altas Prestaciones / Cloud and High-Performance Computing

AUTOR/A: Gaspar Aparicio, Ernesto

Tutor/a: Blanquer Espert, Ignacio

Director/a Experimental: CABALLER FERNANDEZ, MIGUEL

CURSO ACADÉMICO: 2022/2023

Vemos las cosas, no como son, sino como somos nosotros.

Kant.

Resumen

Securizar el acceso a nuestras aplicaciones es esencial con el crecimiento de la adopción de Kubernetes. A menudo, proteger microservicios supone algunos desafíos para el desarrollador que debe tener en cuenta algunas características como la utilización de proveedores de identidad (IdP) confiables o la importancia de implementar un sistema de *single-sign-on* (SSO).

En un sistema distribuido es vital la homogeneización de todos los mecanismos de autenticación, para que así, el administrador de sistemas pueda disponer de una gestión centralizada y simplificada de la concesión de permisos y la revocación de los mismos.

En este Trabajo de Fin de Máster se propone y se implementa un sistema homogéneo de autenticación y autorización en un clúster Kubernetes en el ámbito de la investigación, utilizando herramientas de código abierto como Oauth2 Proxy, kubelogin y Keycloak.

Palabras clave: Kubernetes, OpenId Connect, Keycloak, Oauth2 Proxy, AAI.

Abstract

Securing access to applications is essential as Kubernetes adoption grows. Securing microservices often poses some challenges for the developer who must take into account some features such as the use of trusted identity providers (IdP) or the importance of implementing a single-sign-on (SSO) system.

In a distributed system it is vital to homogenise all the authentication mechanisms, so that the system administrator can have a centralised and simplified management of the granting and revocation of permissions.

In this Master's Thesis we propose and implement a homogeneous authentication and authorisation system in a Kubernetes cluster in the field of research, using open source tools such as Oauth2 Proxy, kubelogin and Keycloak.

Keywords: Kubernetes, OpenId Connect, Keycloak, Oauth2 Proxy, AAI.

AGRADECIMIENTOS

Quiero agradecer a todos los profesores que me han impartido clase durante el Máster.

A mi tutor, Nacho Blanquer, por guiarme y orientarme durante este periodo.

A todo el personal del Instituto Ai2, en especial, a Paco, por darme la oportunidad de trabajar en su equipo y a Ana por su confianza, amabilidad y paciencia para enseñarme.

A mi familia por el apoyo incondicional continuo.

A mis amigos por estar ahí siempre.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

INDICE DE ANEXOS

1. INTRODUCCIÓN

1.1. Motivación

A consecuencia de mi Trabajo de Fin de Grado titulado “Despliegue de un clúster de Kubernetes altamente disponible en Google Cloud Platform”, me empecé a interesar más en el mundo de la gestión de contenedores. En mi TFG pude desplegar un clúster de Kubernetes altamente disponible con varias herramientas comparándolas entre ellas. El desarrollo de este trabajo me llevó a querer profundizar más en este sector y por ello cursé el Master Universitario en Computación en la Nube y de Altas Prestaciones.

Después de todo lo aprendido, uno de los problemas que se plantean es la protección de microservicios desplegados en un clúster de cómputo. Normalmente, estos servicios suelen estar expuestos a Internet y pueden ser una puerta de entrada para posibles ataques cibernéticos. Durante mi estancia en prácticas en el Instituto Universitario de Automática e Informática Industrial (Instituto ai2) se me propuso trabajar en un sistema que solucionara este problema.

Las infraestructuras de autorización y autenticación de identidades (AAI por su nombre en inglés) es fundamental para garantizar la seguridad en los sistemas informáticos. Este Trabajo de Fin de Master tiene como objetivo la implementación de un sistema homogéneo de autenticación y autorización de un cluster de Kubernetes en el ámbito de la investigación. Este estudio surge como respuesta a la problemática relacionada con la gestión de la seguridad en sistemas de contenedores y la necesidad de establecer un contexto que garantice la AAI en un clúster de Kubernetes.

1.2. Objetivos

El objetivo principal de este Trabajo de Fin de Master es facilitar la gestión de usuarios en un clúster de computación en un entorno científico / académico.

Los objetivos específicos del trabajo son:

1. Proponer una arquitectura y escoger las herramientas necesarias para su implementación.

2. Desplegar un servicio de gestión de identidades y autorizaciones (Keycloak) integrado con el servicio de directorio de usuarios de la UPV.

3. Desplegar un sistema de autenticación para el cliente de Kubernetes (*kubectl*) que solicite las credenciales al usuario al hacer una consulta al API server de Kubernetes.

4. Desplegar un sistema de autenticación para controlar el acceso a cualquier microservicio ejecutándose en el clúster, utilizando las mismas credenciales y sin necesitar volver a autenticarse.

1.3. Alcance

Este trabajo está orientado a proponer y llevar a cabo una arquitectura segura que, autentique a usuarios mediante servicios de proveedores confiables y gestionar el acceso de los mismos a diferentes recursos del clúster.

Este estudio se centra en la integración de un proveedor de identidades que sea compatible y que implemente el protocolo OpenId Connect (OIDC). La implementación de este protocolo se realiza en un sistema de AAI de un clúster computacional donde los recursos y servicios de este son accesibles mediante el protocolo HTTP.

La utilización de los resultados del trabajo requiere tener unos conocimientos básicos sobre Kubernetes y administración de sistemas. Es esencial conocer el manejo de los objetos de Kubernetes, implicando la creación, actualización y eliminación de *Deployments*, *Services*, *Volumes*, *Secrets*, *Pods,* etc. Además, los conocimientos básicos sobre administración de sistemas son necesarios a la hora de solucionar errores, acceder a contenedores, navegar por el sistema de archivos, etc.

Por otro lado, este trabajo se ha realizado en un entorno y con unas versiones específicas de las tecnologías que permiten la compatibilidad entre ellas y las cuales se iran mencionando durante el desarrollo de él.

1.4. Metodología

La metodología utilizada se basa en una serie de etapas que comienzan con una fase exploratoria y de formación, seguida de una etapa de diseño y pruebas de concepto, y que continua con el despliegue y configuración de los diferentes servicios. Finaliza con la validación del sistema.

A continuación, se muestra el Diagrama de Gantt correspondiente al tiempo dedicado para cada tarea del proyecto.

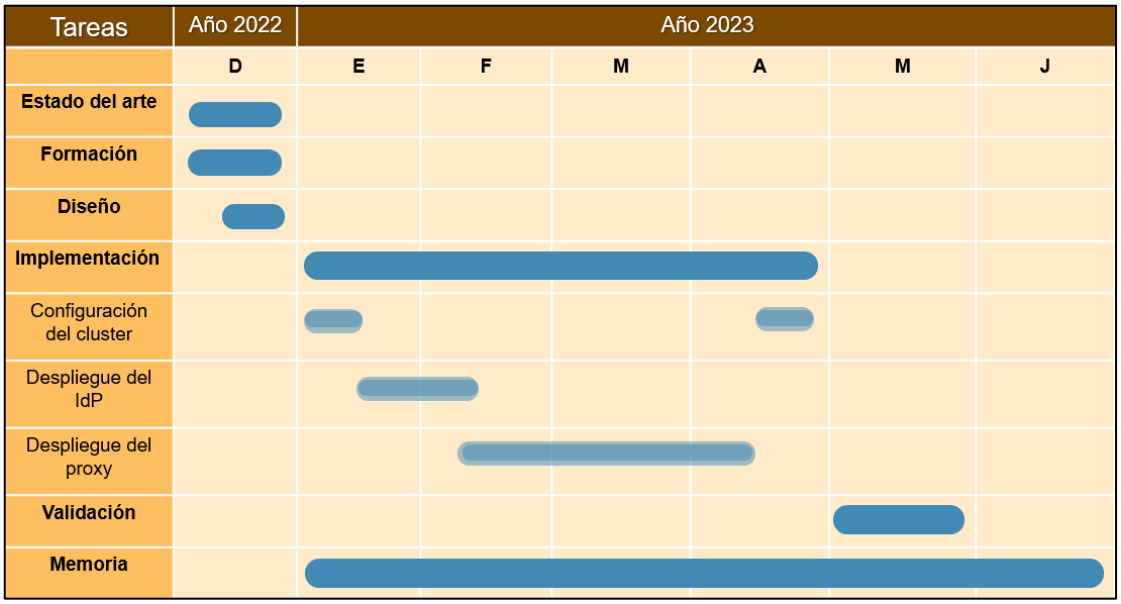

Figura 1. Diagrama de Gantt.

Las tareas se dividen en las siguientes fases:

- **Estado del arte (30d)**: durante este periodo se analiza el problema y se hizo una revisión bibliográfica y una análisis de las soluciones posibles.
- **Formación (30d)**: debido a la inexperiencia en el tema, durante el estado del arte, se realiza paralelamente un periodo de formación en el que se estuvo aprendiendo los conceptos básicos de las tecnologías y su uso.
- **Diseño (10d)**: se identifican los requisitos necesarios para realizar el diseño de la arquitectura.
- **Implementación (120d)**: el periodo de implementación se compone, de varias subtareas:
    - **Configuración del cluster (10d+10d)**: esta tarea se realiza en dos etapas: una de configuración inicial del clúster al principio de la etapa de implementación, y otra al final para terminar de integrar los componentes en el sistema.
    - **Despliegue del IdP (30d)**: esta fase engloba el despliegue y la configuración de Keycloak como IdP.
    - **Despliegue del proxy (60d)**: durante este periodo se despliega una instancia de Oauth2 Proxy y se configura para funcionar con Keycloak.
- **Validación (30d)**: esta etapa del trabajo incluye la validación de la solución implementada con varios microservicios.
- **Memoria (180d)**: durante todo el periodo del trabajo, se han ido documentando las implementaciones y configuraciones.

2. TECNOLOGÍAS RELACIONADAS Y ESTADO DEL ARTE

2.1. Descripción de las tecnologías

Todas las tecnologías utilizadas para el desarrollo de este TFM se distribuyen mediante licencias de código abierto, lo cual abre puertas a que cualquier usuario pueda reutilizar y reproducir los resultados de este trabajo.

A continuación, se definen las tecnologías utilizadas:

- Kubernetes (v1.20.7): es un orquestador de contenedores [1].

- OpenID connect (OIDC): es un protocolo de autenticación de identidades que utiliza Oauth2.0 y sirve para verificar la identidad de un usuario ante un servicio de cliente [2].

- Oauth2 Proxy (v7.2.1-debian-10-r149): es un proyecto de código abierto que actúa como un *middleware* de autenticación entre la aplicación cliente y el IdP [3].

- Oauth 2.0: es un protocolo de autorización para conceder acceso a conjuntos de recursos como API remotas o datos de usuario [4].

- Kubelogin (v1.27.0): es un *plugin* de la herramienta de línea de comandos para Kubernetes (*kubectl*) que permite la autenticación a éste nivel [5].

- Keycloak (v21.0.1): es un servicio de gestión de identidades y autorizaciones [6].

- LDAP (Protocolo Ligero de Acceso a Directorios): es un protocolo de la capa de aplicación TCP/IP que permite el acceso a un servicio de directorio ordenado y distribuido, para buscar información en un entorno de red.

2.1.1. Kubernetes

Kubernetes es una plataforma de sistema distribuido de código libre que permite automatizar el despliegue, ajustar la escala y el manejo de aplicaciones en contenedores [1].

Hoy en día, Kubernetes es el orquestador de contenedores más utilizado en el mercado, debido a sus numerosas características como la capacidad de escalado, balanceo de carga, monitorización y gestión de recursos, la tolerancia a fallos, la facilidad de implementación, la integración con otras herramientas y servicios y su facilidad de extensión.

En este trabajo, se implementa un sistema homogéneo de autenticación y autorización para un clúster de cómputo basado en Kubernetes. Como es conveniente, se hará uso de varios objetos para integrar las tecnologías mencionadas anteriormente.

Por ello, es importante definir los objetos y unidades del sistema que se van a utilizar [7]:

- **Pod:** es un grupo de uno o más contenedores, con almacenamiento y red compartidos, y unas especificaciones de cómo ejecutar los contenedores. Representa el bloque de construcción básico de Kubernetes.

- **Deployment**: es un objeto de Kubernetes que permite representar una aplicación en un clúster. Su principal ventaja es la gestión de múltiples instancias, la tolerancia a fallos y el escalado rápido.

- **Service**: es un objeto que permite que las aplicaciones desplegadas sean accesibles desde dentro y fuera del clúster. Existen distintos tipos de *Services* como:

    - ClusterIP: expone los *Pods* sólo dentro del clúster y son accesibles mediante su dirección IP y nombre de DNS.

    - NodePort: expone los *Pods* en un puerto específico de todos los nodos del clúster hacia el exterior de este.

    - LoadBalancer: permite exponer servicios al exterior utilizando un balanceador de carga de un proveedor en la nube.

    - ExternalName: realiza referencias a servicios externos fuera del clúster utilzando un nombre de DNS.

- **Ingress**: este recurso permite la exposición controlada de servicios HTTP y HTTPS externos al clúster de Kubernetes. En comparación con un *Service,* este objeto permite otorgar un *hostname* y protección TLS a un servicio ya expuesto en el clúster.

  Es necesario tener un controlador de *Ingress* que permita gestionar el enrutamiento. En el caso del presente trabajo se utilizará un NGINX Ingress Controller.

- **Namespace**: constituye un entorno dentro del espacio del clúster donde aislar y gestionar a la vez un conjunto de recursos determinados del clúster.

- **PersistentVolume (PV):** es un objeto que define un recurso de almacenamiento y permite que los datos persistan incluso cuando los *Pods* se reinician o mueren.

- **PersistentVolumeClaim (PVC):** es un recurso que define la asignación de un *Pod* a un *PV*. Es una solicitud de almacenamiento específica de un *Pod*.

- **Secret:** estos objetos permiten almacenar y administrar información confidencial como contraseñas, tokens y claves SSH.

- **Role-based access control (RBAC):**

  Kubernetes también define una serie de objetos para regular el acceso a recursos mediante roles. Los *Roles* y los *ClusterRoles* contienen reglas que representan un conjunto de permisos. Estos permisos son puramente de adición, es decir, no existen reglas para denegar acciones.

  - Role**:** establece permisos dentro de un determinado *Namespace.*

  - ClusterRole**:** establece permisos para todo el clúster.

  Estos roles, se enlazan a usuarios, grupos o procesos, mediante los *RoleBinding* o *ClusterRoleBinding.*

  - RoleBinding**:** es un objeto que representa la relación entre un *Role* y un usuario, grupo o proceso.

  - ClusterRoleBinding**:** es un objeto que representa la relación entre un *ClusterRole* y un usuario, grupo o proceso.

2.1.2. OIDC

OpenID Connect (OIDC) [2] es un protocolo de autenticación construido sobre Oauth2.0 [3]. Permite que las aplicaciones de terceros verifiquen la identidad de un usuario final utilizando sus credenciales de identificación existentes. Combina tanto la autenticación basada en *tokens* de Oauth2.0 con la información de identidad de OpenID.

OpenID Connect es un protocolo enfocado a la autenticación y añade nuevas funcionalidades que complementan a Oauth 2.0:

- Un ID *token* que permite saber quién es el usuario.
- Un nuevo *endpoint*, *userinfo*, que permite recuperar más información del usuario.
- Un conjunto de *scopes* estándar, es decir, un estándar de permisos a los que quiere acceder el cliente.
- Un conjunto de *claims* que permite obtener datos del sujeto.

Se realiza una peticion al *endpoint /token* del protocolo OIDC, el cual se utiliza para extraer el *access token*. Como se observa en la figura 2, se colorea en azul los parámetros que se solicitan en la petición y en rojo el valor que se le está asignando. En este caso, se le asigna como parámetro un usuario de prueba llamado “Prueba”.

```
$ curl -k -X POST
https://keycloak.ai2.upv.es/realms/ai2realm/protocol/open
id-connect/token -d grant_type=password -d
client_id=ai2kubernetes -d
client_secret=****************** -d username=prueba -d
password=prueba  -d scope=openid -d
response_type=id_token
```

Figura 2. Petición *endpoint /token*.

Se puede ver como la respuesta que se obtiene incluye más parámetros:

Figura 3. Respuesta petición *endpoint /token*.

Además del *access token*, que se obtiene a partir del protocolo Oauth 2.0, existe un *token* nuevo llamado *id_token*, que viene en formato JWT, un estándar que permite transmitir información de manera segura. Este *token* contiene información sobre el usuario y se puede descodificar a través de la página oficial de JWT [8].

El *id_token* tiene una estructura formada por tres partes separadas por un punto, la cuales son la cabecera, el *payload* y la firma.

- La cabecera contiene dos partes: el tipo de *token* y el algoritmo que se ha utilizado para ser encriptado (coloreado en azul).

- El *payload* contiene información y los permisos del usuario al que pertenece el token en la aplicación. Alberga información como el grupo al que pertenece, el nombre o email (coloreado en verde).

- La firma verifica que el mensaje no ha sido comprometido en ningún momento (coloreado en rojo).

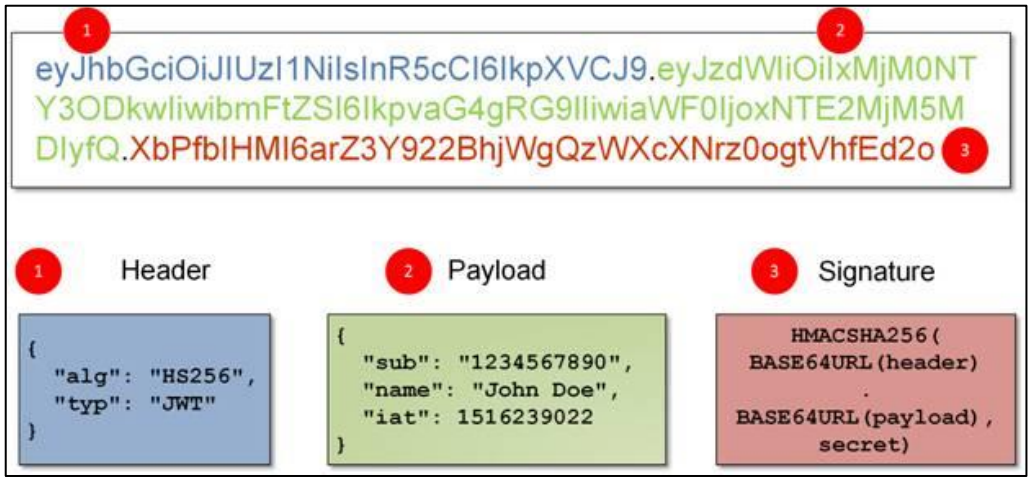

Figura 4. Partes de un *id_token*. Tomada de [9]

El *token* descodificado a través de la página web de JWT [8] contiene la siguiente información:

a) Cabecera

A continuación se detalla la información descodificada de la cabecera, donde se pueden diferenciar las dos partes mencionadas.

Figura 5. Cabecera descodificadad de un *id_token*.

b) Payload

En el *payload* se observa información más específica sobre el usuario.

Figura 6. *Payload* descodificado de un *id_token*.

c) Firma

A continuación, se muestra la firma descodificada.

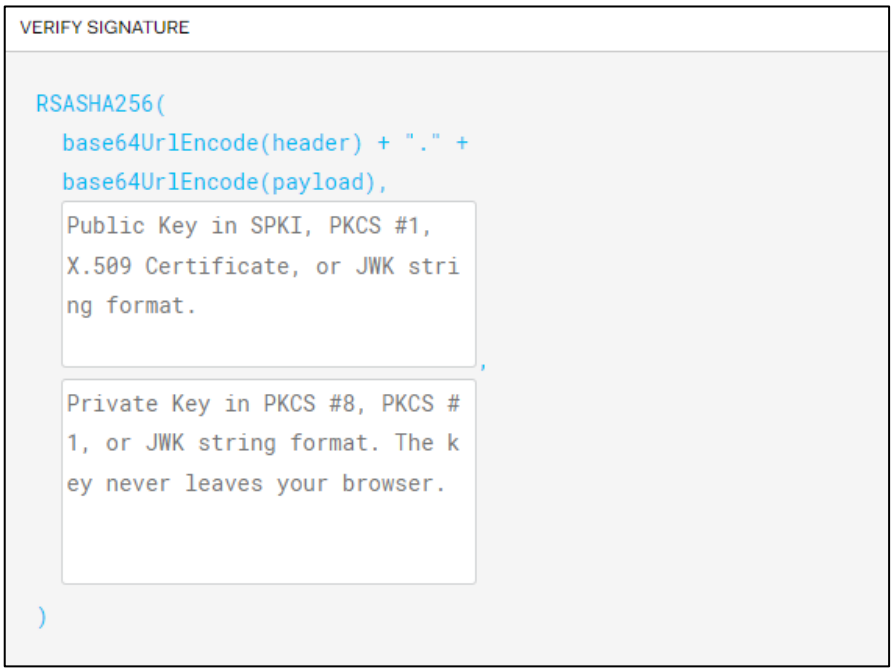

Figura 7. Firma descodificada de un *id_token*.

2.1.3. Oauth2 Proxy

Oauth2 Proxy es un *reverse proxy* que figura delante de las aplicaciones y agrega una capa adicional de seguridad al gestionar los protocolo de OIDC y Oauth 2.0 fácilmente.

Además, permite la configuración de multitud de proveedores como Google, Facebook, Github o Keycloak entre otros.

Algunos de los *endpoints* útiles que ofrece la aplicación son [10]:

- **/oauth2/userinfo:** devuelve información sobre el usuario de la sesión en formato JSON.
- **/oauth2/auth:** comprueba si el usuario está autenticado.
- **/oauth2/start:** inicia el ciclo de autenticación de Oauth.
- **/oauth2/sign_in:** devuelve la página de inicio de sesión. Si se está utilizando un proveedor para ello, devolverá la del proveedor.
- **/oauth2/sign_out:** limpia la sesión del usuario borrando la *cookie* del navegador.

Los *endpoints* que ofrece Oauth2 Proxy desempeñan funciones específicas dentro del flujo de autenticación y autorización. Estos son algunos de los *endpoints* más útiles, pero existen otros que convendrá utilizar dependendiendo de la arquitectura de la aplicación que se esté realizando.

2.1.4. Oauth 2.0

Oauth 2.0 [3] es un *framework* de autorización, que permite a las aplicaciones obtener acceso (limitado) a las cuentas de usuario de determinados servicios como Facebook, Github, Twitter y muchos más.

Consiste en delegar la autenticación de usuario al servicio que gestiona las cuentas, de modo que sea éste quien otorgue el acceso para las aplicaciones de terceros. Este estándar ofrece flujos de autenticación para aplicaciones web, aplicaciones de escritorio y móvil, entre otros.

Es importante definir la terminología que se va a utilizar para entender mejor el protocolo Oauth 2.0:

- **Recurso** (*resource*): es cualquier elemento que tenga valor y deba ser protegido, como archivos, bases de datos, directorios, funcionalidades específicas de una aplicación, etc.

- **Propietario del recurso** (*resource owner*): se refiere al usuario o aplicación que tiene el control sobre los recursos protegidos. Puede ser un usuario que posea datos personales o una aplicación que posea recursos específicos.

- **Cliente** (*client*): se refiere a la aplicación cliente que quiere acceder a datos o ejecutar alguna acción en el nombre del propietario del recurso. A fin de conseguir ello, debe contar con una autorización del usuario, y esta autorización se debe validar (a través del API del servicio).

- **Servidor de autorización** (*autorization server*): es la aplicación que conoce al propietario del recurso, donde este ya posee una cuenta en él.

- **Servidor de recursos** (*resource server*): es el servicio o API que el cliente quiere utilizar en nombre del propietario del recurso. Hay ocasiones, en las que el servidor de autorización y el servidor de recursos puede ser el mismo, en otras, puede ser que el servidor de autorización sea externo a la organización, como un servicio de terceros y que el servidor de recursos confíe en él.

- **URI de redirección** (*redirect URI*) o **función de *callback***: es la URL que el servidor de autorización utilizará para redirigir de vuelta al propietario del recurso después de otorgarle permiso al cliente.

- **Tipo de respuesta** (*response type*): se trata del tipo de respuesta que el cliente espera recibir. Puede ser *code*, *token*, *id_token* o *none*.

- **Alcance** (*scope*): son los permisos a los que quiere acceder el cliente, como por ejemplo, acceder a datos o ejecutar acciones de lectura, escritura o borrado.

- **Consentimiento** (*consent*): el servidor de autenticación obtiene el alcance de la petición del cliente y verifica con el propietario del recurso si desea o no dar esos permisos al cliente.

- **ID del cliente** (*client id*): es un identificador de caracteres que se utiliza para identificar al cliente con el servidor de autorización.

- **Secreto del cliente** (*client secret*): es una contraseña secreta que solo el cliente y el servidor de autorización conocen. Esto les permite compartir información de una manera segura y privada.

- **Código de autorización** (*authorization code*): es un código temporal de corta duración que envía el servidor de autorización al cliente y el cliente lo envía de vuelta al servidor de autorización junto al secreto del cliente a cambio de un *access token*.

- **Token de acceso** (*access token*): es una "clave" que utilizará el cliente para comunicarse con el servidor de recursos. Le otorga permisos al cliente para pedir datos o ejecutar ciertas acciones dentro del servidor.

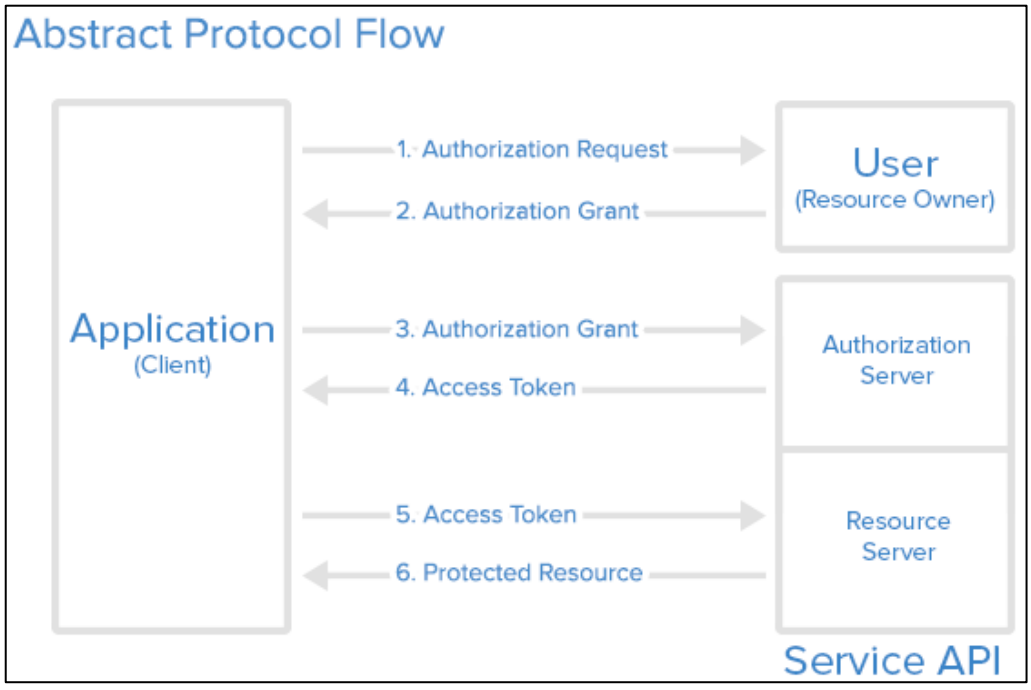

Figura 8. Flujo del protocolo Oauth 2.0. Tomada de [11]

Una vez vista la terminología basica, el flujo del protocolo tendría la siguiente estructura [3]:

1. Supongamos una aplicación cliente que solicita una autorización para acceder a los recursos de un usuario, en un servicio determinado.
2. Si el usuario autoriza esta solicitud, la aplicación recibe una *authorization grant* (concesión de autorización).
3. La aplicación cliente solicita un *access token* al servidor de autorización, demostrando que un cliente es válido y el permiso concedido anteriormente.
4. Si la identidad de la aplicación cliente se valida adecuadamente por el servicio, y la concesión de autorización es válida, el servidor de autorización emite un *access token* a la aplicación cliente. Con esto la autorización se ha completado.
5. La aplicación cliente puede presentar el *access token* recibido en el paso anterior, y "solicitar un recurso" al servidor de recursos.
6. Si el *access token* es válido, el servidor de recursos hace entrega del recurso a la aplicación.

2.1.5. Kubelogin

Kubelogin es un *plugin* de la herramienta de línea de comandos para Kubernetes (*kubectl*) que permite la autenticación a éste nivel [5]. Permite autenticarse en clústeres de Kubernetes utilizando proveedores de identidad externos a través del protocolo OIDC.

En Kubernetes existen varias formas de autenticación, una de ellas siendo la autenticación por *token*, que permite a los usuarios presentar un *token* válido para acceder al clúster.

Este *plugin* simplifica este proceso permitiendo a los usuarios autenticarse a través de un proveedor de identidades externo. Una vez autenticado el usuario, el mecanismo se encarga de extraer directamente del proveedor un *token* apropiado y de presentarselo al clúster en nombre del usuario.

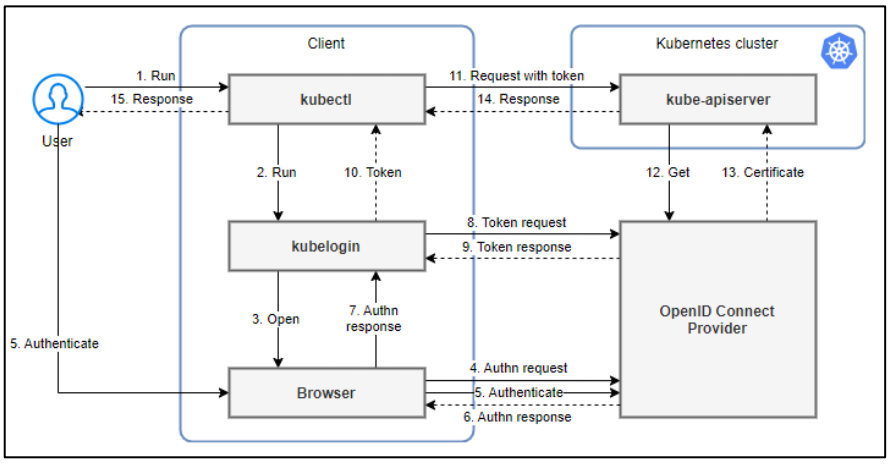

Figura 9. Diagrama de flujo kubelogin. Tomada de [5]

2.1.6. Keycloak

Keycloak [6] es un componente software de código abierto que permite el SSO con *Identity Management* y *Access Management* para aplicaciones y servicios modernos. Esta tecnología está escrita en Java y es compatible de forma predeterminada con los protocolos de federación de identidad SAML v2 y OpenID Connect (OIDC) / OAuth 2.0.

Keycloak ofrece inicio de sesión único, donde los usuarios se autentican con Keycloak directamente en vez de con aplicaciones individuales. De esta forma las aplicaciones no necesitan lidiar con formularios de inicio de sesión, autenticación o almacenamiento de usuarios. Todo esto se gestiona de forma centralizada desde la plataforma. Gracias a esto, los usuarios no tienen que volver a iniciar sesión para acceder a una aplicación diferente.

Adicionalmente, desde la consola de administración, Keycloak permite agregar inicio de sesión con IdPs de redes sociales y con IdPs OpenID Connect o SAML 2.0 existentes.

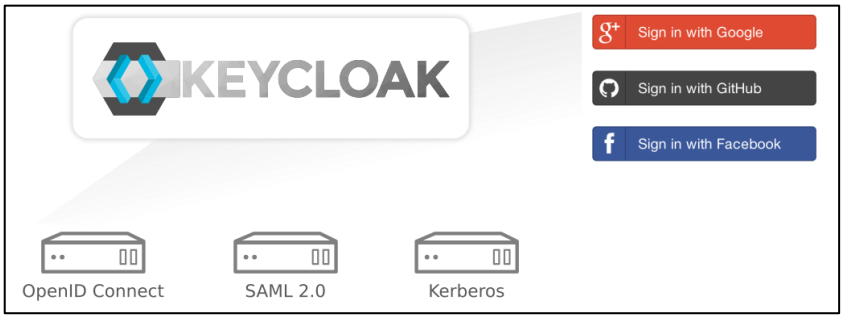

Figura 10. Inicio de sesión social. Tomada de [6].

Keycloak también tiene soporte incorporado para conectarse a servidores *LDAP* o *Active Directory* existentes. Se usará esta funcionalidad para poder conectar la base de datos de LDAP de la UPV y poder importar usuarios.

Esta tecnología define una seríe de componentes y elementos que nos ayudará a establecer nuestro sistema de gestión de usuarios y aplicaciones.

Keycloak está compuesto de *realms* (reinos o dominios administrativos), que se definen como un concepto similar a los *Namespaces* donde se puede gestionar y guardar cualquier configuración específica. De esta forma, el administrador puede tener varios *realms* con distintas configuraciones según los requisitos.

Por cada *realm*, es posible crear o importar un conjunto de usuarios (los cuales poseen nombre, usuario, direccion de correo, teléfono, etc.) y asignarles roles o agregarlos a grupos.

También se pueden crear clientes, que son entidades que pueden solicitar a Keycloak la autenticación de un usuario. En la mayoría de los casos, los clientes son aplicaciones y servicios que desean utilizar esta plataforma para protegerse y proporcionar una solución de inicio de sesión único. También, pueden ser

entidades que sólo quieren solicitar información de identidad o un *access token* para que puedan invocar de forma segura otros servicios en la red que están protegidos por Keycloak.

Existen otros componentes que sirven para lograr configuraciones más específicas como los *client scopes* que definen características o funcionalidades adicionales que pueden ser concedidas a un cliente al obtener un *access token*; o los *mappers* que se utiliza para modificar o agregar información adicional a un *token* emitido por Keycloak antes de que sea consumido por la aplicación o servicio.

2.1.7. LDAP

El protocolo LDAP es un estándar de Internet que funciona a nivel de aplicación y que proporciona acceso a la información desde distintas aplicaciones y sistemas informáticos. LDAP usa un conjunto de protocolos para acceder a directorios remotos y recuperar información [12].

LDAP está basado en el protocolo X.500 para compartir directorios. Contiene esta información de forma jerarquizada y mediante categorías para proporcionar una estructura intuitiva desde el punto de vista de la gestión por parte de los administradores.

Así mismo, los directorios se utilizan para contener información virtual de usuarios, para que otros usuarios accedan y dispongan de información acerca de los contactos que están aquí almacenados.

Las configuraciones de LDAP utilizan una estructura de árbol de directorio (DIT) estandarizada y jerarquizada para los directorios y estructuras de datos, que puede distribuirse en varios servidores.

Este protocolo tiene una serie de componentes principales [13]:

- **Organizational Unit (OU):** es un objeto de unidad organizativa, similar a un directorio de Windows. Para LDAP generalmente retiene objetos de grupo y objetos de usuario.

- **CN de grupo (Common Name):** de la misma manera en que una unidad organizativa es similar a un directorio, un objeto de grupo es similar a un archivo. El objeto de grupo contiene un atributo de miembro que es una lista de nombres distintivos que definen a los usuarios en ese grupo.

- **CN de usuario:** el objeto de usuario, también similar a un archivo, describe una persona única dentro de una estructura de LDAP. A diferencia de un grupo, un usuario no contiene una lista, en cambio, sus atributos describen a un usuario con mayor detalle que los necesarios.

- **Distinguised Name (DN)**: es la dirección única que identifica una entrada en el servicio de directorio y sigue una estructura jerárquica. Para crear un nombre distintivo, generalmente, se va desde el objeto al cual se está nombrando hasta la parte superior del árbol, de manera que, el nombre distintivo para el usuario Juan Español, sería:

CN=juanespanol,OU=Usuarios,OU=VSMGUI,DC=miempresa,DC=com

2.2. Justificación de la elección de las tecnologias

En la configuración por defecto de un clúster de Kubernetes la seguridad de los servicios se puede ver fácilmente comprometida si no se implementa una capa por encima que los proteja.

Desarrollar un flujo de autenticación para cada microservicio es una tarea que requiere mucho tiempo, es poco segura y agrega esfuerzos generales de mantenimiento. Además, también supone una molestia para el usuario final que tendrá que escribir sus credenciales de acceso cada vez que acceda a cualquier aplicación. Es de gran importancia facilitar al usuario que se autentica que exponga el menor número de veces sus claves.

Por otro lado, gestionar las identidades en el cliente también es una tarea costosa, por lo que el uso de IdPs institucionales confiables puede facilitar este proceso.

La implementación por parte del desarrollador de los servicios de autenticación supone, adicionalmente, un esfuerzo de programación y gestión que podría evitarse con un sistema coordinado y homogéneo.

Esta solución también podría implantarse para resolver la problemática que surge a la hora de descargar aplicaciones a través de gestores de aplicaciones de Kubernetes como *Helm* cuyo código está protegido bajo licencia.

En este contexto, Oauth2 Proxy ofrece una manera de descargar el flujo de trabajo de autenticación a una única instancia de *proxy* configurable, lo cual generará un conjunto de beneficios en cuanto a:

- Costes computacionales, ya que se comparte la misma instancia para proteger distintos microservicios. Teniendo siempre en cuenta que tendrá unas limitaciones y que se asegurará su escalabilidad cuando sea necesario.
- Costes de mantenimiento, ya que no hará falta programar ni mantener la autenticación para cada microservicio. También tendremos la opción de proteger servicios de cuyo codigo no somos propietarios.

La implementación de un sistema homogéneo, supone muchas ventajas frente a uno heterogéneo. El sistema homogéneo permite la centralización de la autenticación con un único proveedor y unas mismas políticas de seguridad y acceso. Esto, facilita al administrador el mantenimiento de los componentes del sistema. Asimismo, proporciona consistencia en la forma de autenticación de los usuarios en las aplicaciones finales.

Como se ha mencionado anteriormente, se destaca la importancia que tienen el uso de IdPs institucionales confiables para facilitar el proceso de verificación del cliente. Keycloak es una solución confiable, diseñada siguiendo protocolos de seguridad estándar para proporcionar una solución dinámica de inicio de sesión único. Igualmente, tiene la licencia de *Apache License Version 2.0* y una comunidad de código abierto fuerte y activa.

Keycloak ofrece un adaptador LDAP que permite conectar y sincronizar usuarios de un servidor LDAP externo. Puesto que los servicios que se van a proteger principalmente son accedidos por personal de la UPV, se integra el sistema de credenciales de la universidad para facilitar su acceso.

Al mismo tiempo se busca tener un control detallado sobre las acciones que puede realizar cada usuario, por lo que, también es importante conocer la identidad del cliente que accede. Esto sólo se consigue con la implementación de proveedores de identidad y el uso de protocolos que lo admitan.

Por otro lado, OIDC establece un marco para que los usuarios se autentiquen con sus credenciales y establecer una comunicación segura que permita el traspaso de la información entre el proveedor y la aplicación.

Por último, el uso de herramientas tan populares y extendidas como Kuberentes ofrece un abanico de tecnologías desarrolladas alrededor de esta, como los numerosos *plugins* disponibles para la interfaz de línea de comandos *kubectl.*

Aprovechando este mercado, podemos encontrar tecnologías que encajen con la arquitectura propuesta del sistema. En este caso el *plugin* kubelogin permite aportar una capa de seguridad y facilitar el sistema de autenticación y autorización en clústeres de Kubernetes a nivel de línea de comandos. De esta forma, se logra un control en todos los aspectos de las acciones que puede realizar un usuario que quiera beneficiarse de un clúster de cómputo.

2.3. Estado del arte

En Kubernetes se puede implementar la autenticación mediante varias alternativas. Existe la opcion de hacerlo mediante: un archivo estático que contenga un listado de usuarios y contraseñas, un listado de usuarios y *tokens* (aunque estas dos opciones están deprecadas desde la versión 1.19), un certificado personalizado para cada usuario que accede o el uso de proveedores de autenticación externos [14].

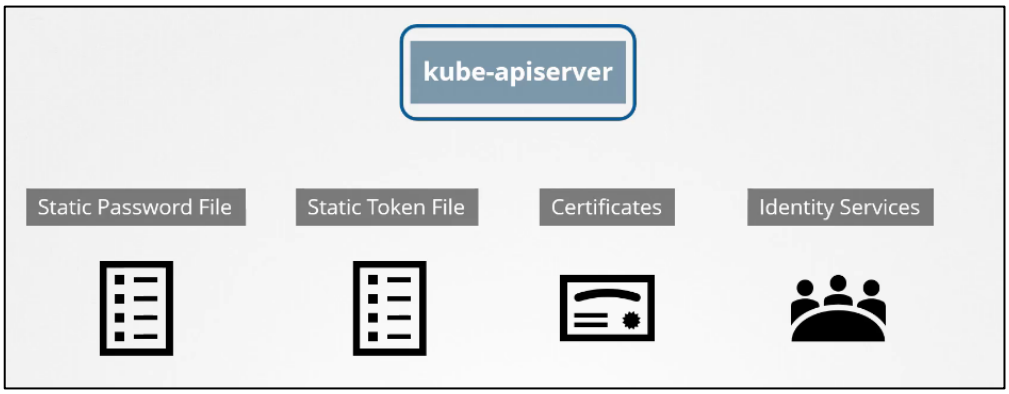

Figura 11. Tipos de autenticación Kubernetes.

En una primera opción, se trató de desarrollar una aproximación (Anexo 1) a través del uso del archivo de configuracion *config* y la creación de certificados para los usuarios. Para esta solución se crea un usuario y un contexto nuevo en el clúster. A este usuario se le establecen los permisos que debe tener y sobre qué *Namespaces*.

A pesar de que esta primera solución funcionaba correctamente ya que limitaba el acceso de los usuarios a los recursos del clúster, se acaba descartando debido a la imposibilidad de controlar el acceso a *endpoints* de HTTP. No solo eso, si no que también la creación de certificados para cada usuario presentaba una carga de trabajo sustancial y monótona para el administrador que la realizaba, aparte de ser una solución no escalable.

Por lo tanto, la propuesta que más se adecúa a la arquitectura del sistema es la de la autenticación mediante proveedores de autenticación externos mediante el uso del protocolo OIDC.

Configurando el cluster de Kubernetes con el protocolo OIDC, estableciendo Keycloak como gestor de autorizaciones e identidades y utilizando las políticas de RBAC de Kubernetes, seremos capaces no solo de gestionar la autenticación de los usuarios, que podrán hacerlo a través de las credenciales de la UPV, si no también de la autorización a realizar sobre ciertos recursos del clúster.

3. ANÁLISIS Y DISEÑO

3.1. Escenarios de Usuario

Se plantean tres escenarios posibles en el sistema de cara al usuario:

1. El acceso vía HTTP a un microservicio protegido en el que se le concede o deniega el acceso al usuario.

2. El acceso, también vía HTTP, a un microservicio protegido en el que se le concede o deniega el acceso al usuario. Además, dentro de la aplicación, el usuario tiene capacidad de interacción con Kubernetes que será controlada a través de RBAC.

3. Acceso vía interfaz de línea de comandos a los recursos de Kubernetes que será controlado a través de RBAC.

3.2. Análisis de requisitos

El análisis de requisitos es necesario para analizar las necesidades y limitaciones de un sistema *software*.

Estos requisitos se dividen principalmente en dos:

- Requisitos funcionales: aquellos que describen las interacciones entre el sistema y su ambiente, en forma independiente a su implementación. El ambiente incluye al usuario y cualquier otro sistema externo con el cual interactúe el sistema.

- Requisitos no funcionales: describen atributos sólo del sistema o del ambiente del sistema, que no están relacionados directamente con los requistos funcionales. Los requisitos no funcionales incluyen restricciones cuantitativas, como el tiempo de respuesta o precisión, tipo de platafoma, etc.

3.2.1. Requisitos funcionales

Requisito	Descripción	Dependencias
RF 1.0	Integración con credenciales de la universidad: permitir a los usuarios acceder con su nombre de usuario y contraseña de la UPV.	
RF 2.0	Gestión de permisos: el sistema debe tener la capacidad de gestionar los permisos y roles de los usuarios.	
RF 3.0	Gestión de usuarios: el sistema debe tener la capacidad de permitir la creación, edición y eliminación de usuarios.	
RF 4.0	Acceso: si las credenciales no son válidas, el sistema no debe permitir el acceso.	RF 1.0
RF 4.1	Si las credenciales son válidas, y el usuario tiene los permisos para acceder al recurso que solicita, el sistema debe permitir el acceso.	RF 1.0, RF 2.0, RF 3.0
RF 4.2	Si las credenciales son válidas, pero el usuario no tiene los permisos para acceder al recurso que solicita, el sistema no debe permitir el acceso.	RF 1.0, RF 2.0, RF 3.0

3.2.2. Requisitos no funcionales

Requisito	Descripción	Dependencias
RNF 1.0	El sistema debe garantizar la disponibilidad evitando interrupciones y minimizando el tiempo de inactividad no planificado.	
RNF 2.0	El sistema debe tener la capacidad de gestionar errores y fallos de manera robusta.	
RNF 3.0	Tiempo de respuesta aceptables no superiores a 2 minutos.	RNF 1.0
RNF 4.0	El sistema debe ser capaz de escalar su número de réplicas según sea demandado.	
RNF 5.0	Garantizar la protección de los datos de los usuarios del sistema.	

3.3. Arquitectura y casos de uso

3.3.1. Arquitectura

La arquitectura del sistema desarrollado se define mediante el siguiente diagrama:

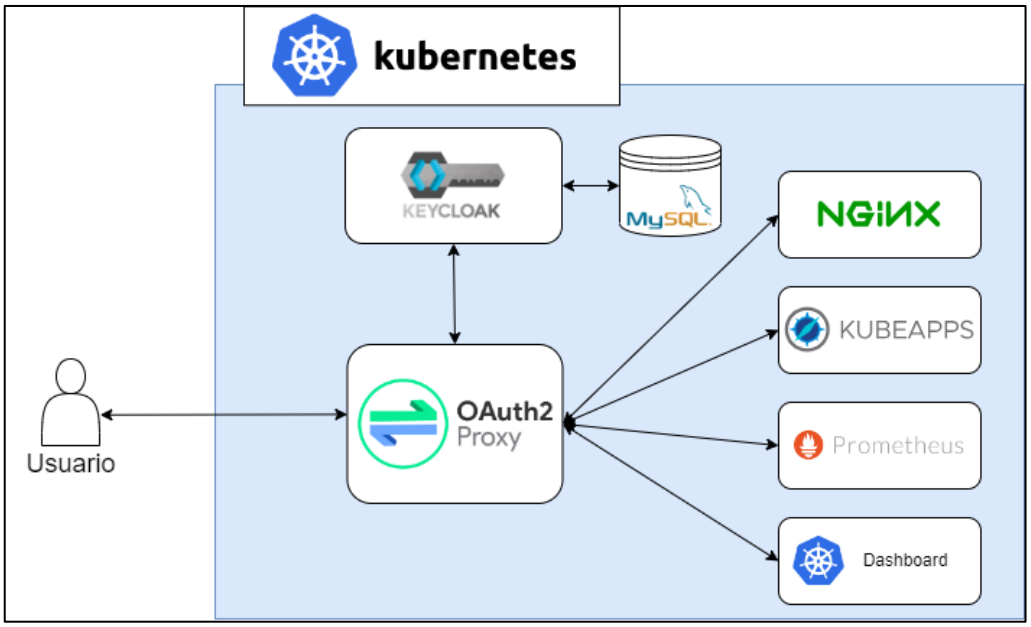

Figura 12. Diagrama de la arquitectura del sistema.

Como se observa, existen varios microservicios que se comunican entre ellos, principalmente es la instancia de Oauth2 Proxy la que hace de intermediario entre el usuario y los microservicios. Esta instancia intercepta la petición del usuario a una aplicación cualquiera y primero comprueba sus credenciales con Keycloak, que persiste sus datos en una base de datos de MySQL.

Todos los microservicios han sido desplegados dentro del mismo clúster de cómputo, que cuenta, con un total de 18 nodos, siguiendo una arquitectura de tolerancia a fallos multimaestro con 3 nodos maestros y 15 nodos trabajadores.

3.3.2. Casos de uso

Los casos de uso en un sistema informático sirven para describir los requisitos funcionales de este. Representan interacciones entre actores y el sistema en cuestión y permiten implementar los escenarios de usuario descritos en el apartado 3.1.

Se han definido 4 casos de uso que podrían suceder en el sistema:

Caso 1. Dar de alta un usuario en la plataforma.

En primer lugar, el administrador de sistemas tiene la capacidad de añadir a usuarios bajo el dominio “Alumno” (coloreado en rosa en la figura 12) o “UPVNET” (coloreado en azul en la figura 12) en el LDAP de la UPV bajo la unidad de la organización del Instituto ai2.

La estructura de árbol es la siguiente:

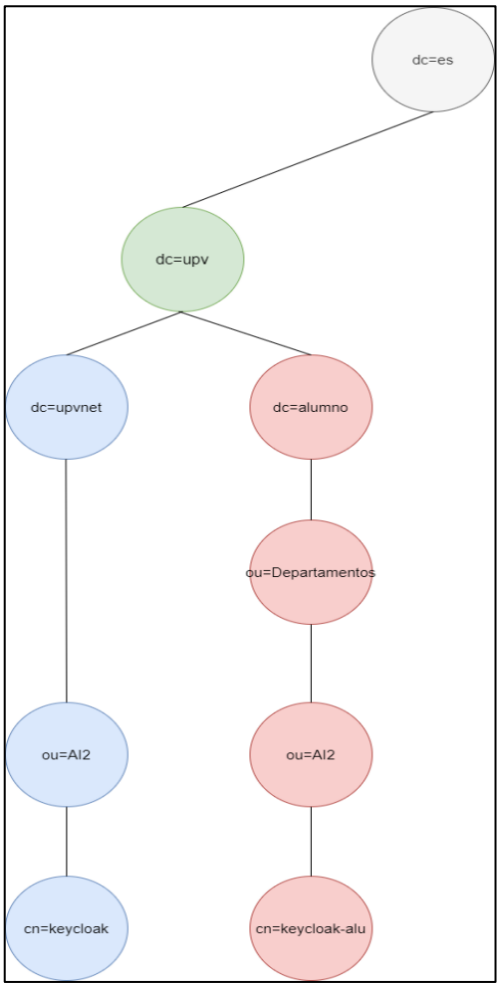

Figura 13. Árbol LDAP UPV ai2.

Dentro de la OU, el administrador de sistemas es capaz de crear un grupo dentro de cada dominio, en este caso se han creado dos grupos por cada dominio de usuarios:

- Grupo “keycloak”, para organizar al conjunto de usuarios bajo el dominio “UPVNET”.

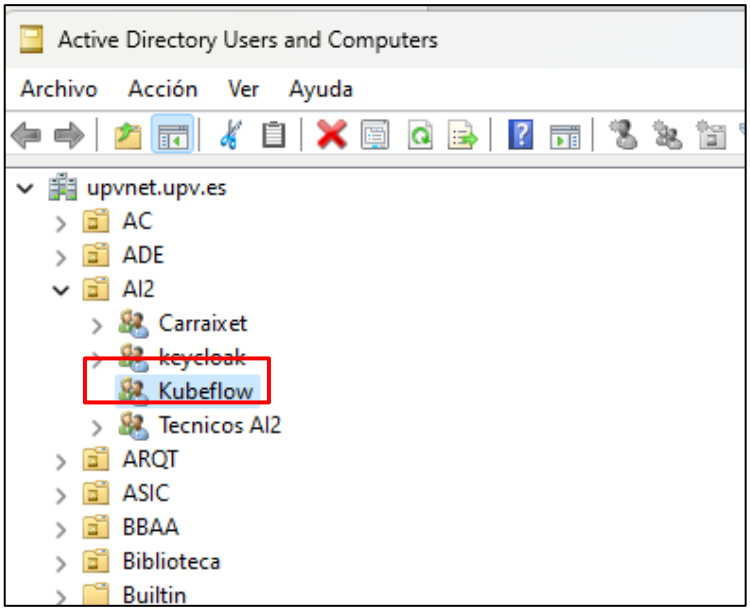

Figura 14. Grupo “keycloak” bajo el dominio de “UPVNET”.

- Grupo “keycloak-alu”, para organizar al conjunto de usuarios bajo el dominio “Alumno”.

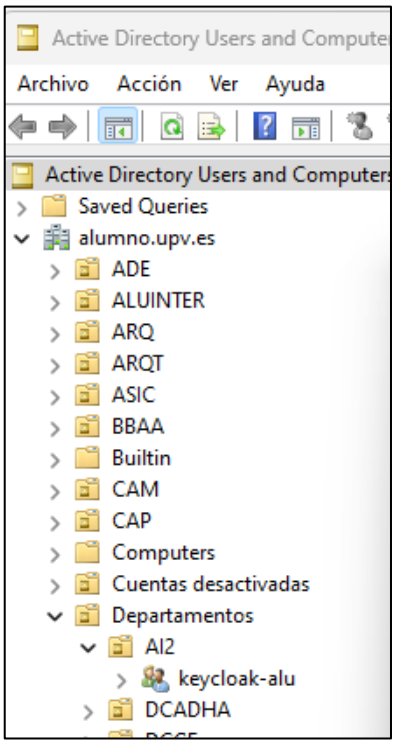

Figura 15. Grupo “keycloak-alu” bajo el dominio “Alumno”.

Por cada nuevo usuario que se quiera dar de alta a la plataforma de Keycloak, primero es necesario averiguar su dominio. Una vez sabido si pertenece al dominio “UPVNET” o “Alumno”, se le agrega al CN correspondiente (“keycloak” o “keycloak-alu”).

Una vez agregado al CN correspondiente, se sincroniza la plataforma de Keycloak para que actualice el listado de usuarios con los recién agregados.

Dentro de Keycloak, se selecciona el usuario y se le agrega a los grupos que corresponda. Por cada aplicación del clúster que se quiere proteger, existe un grupo en Keycloak de usuarios a los que se les concederá el acceso a la aplicación.

Por ejemplo, para la aplicación del Dashboard de Kubernetes, existe un grupo llamado “apps-dashboard” al cual el usuario que desee acceder deberá pertenecer.

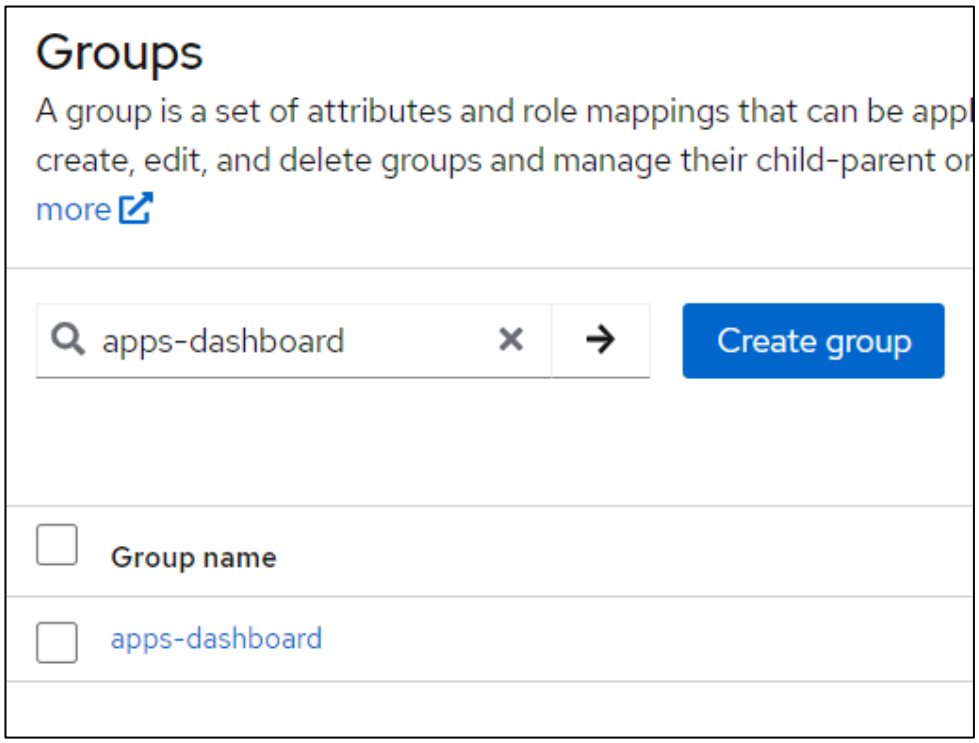

Figura 16. Listado del grupo “apps-dashboard”.

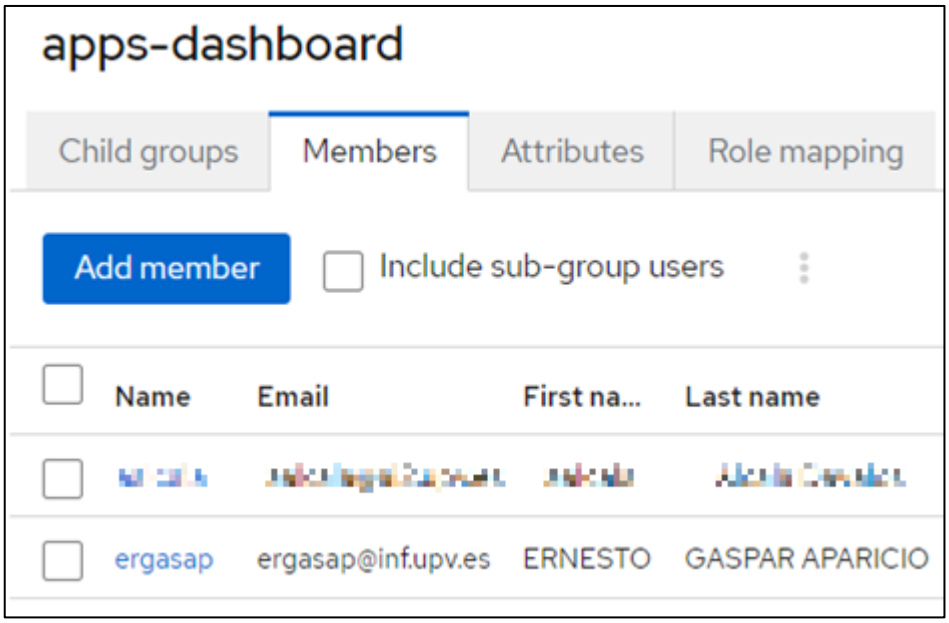

Figura 17. Listado de usuarios pertenecientes al grupo “apps-dashboard”.

También deberá pertenecer a un segundo grupo, según los permisos que se le quieran otorgar dentro de la aplicación que accede. Se ha propuesto un modelo en el que se le conceden permisos de administrador para el *Namespace* “ns-usuario” y permiso restringido para cualquier otro tipo de acciones sobre cualquier *Namespace* que no sea el suyo (explicación más detallada del modelo en el apartado 4.4.).

Por ejemplo, agregaríamos el usuario “Prueba” al grupo “ns-prueba” que tendrá permisos de administrador sobre el *Namespace* “prueba”:

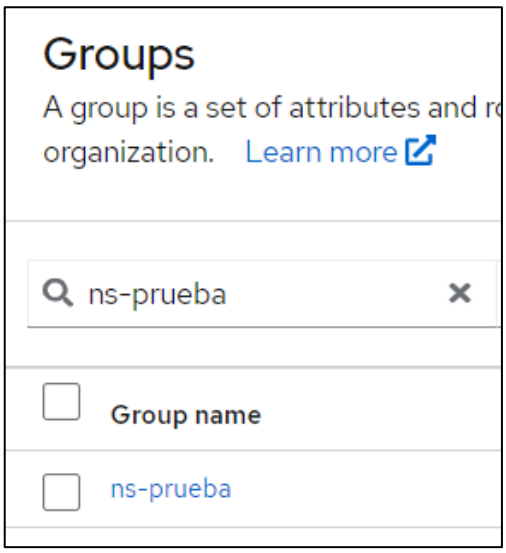

Figura 18. Listado de grupo “ns-prueba”.

El dar de alta a un usuario en el sistema pasa primero por agregarlo al grupo correspondiente en el LDAP. Una vez que aparezca en Keycloak, se agrega el usuario a los grupos de las aplicaciones que tenga acceso, además de agregarlo al grupo correspondiente que hará que obtenga ciertos permisos dentro de las estas aplicaciones.

Caso 2. Usuario con permisos que accede a la aplicación sin estar autenticado.

A continuación, se describe el flujo de autenticación para un usuario que pertenece al grupo con los permisos correspondientes para acceder a una aplicación y que además, este usuario no ha sido autenticado previamente.

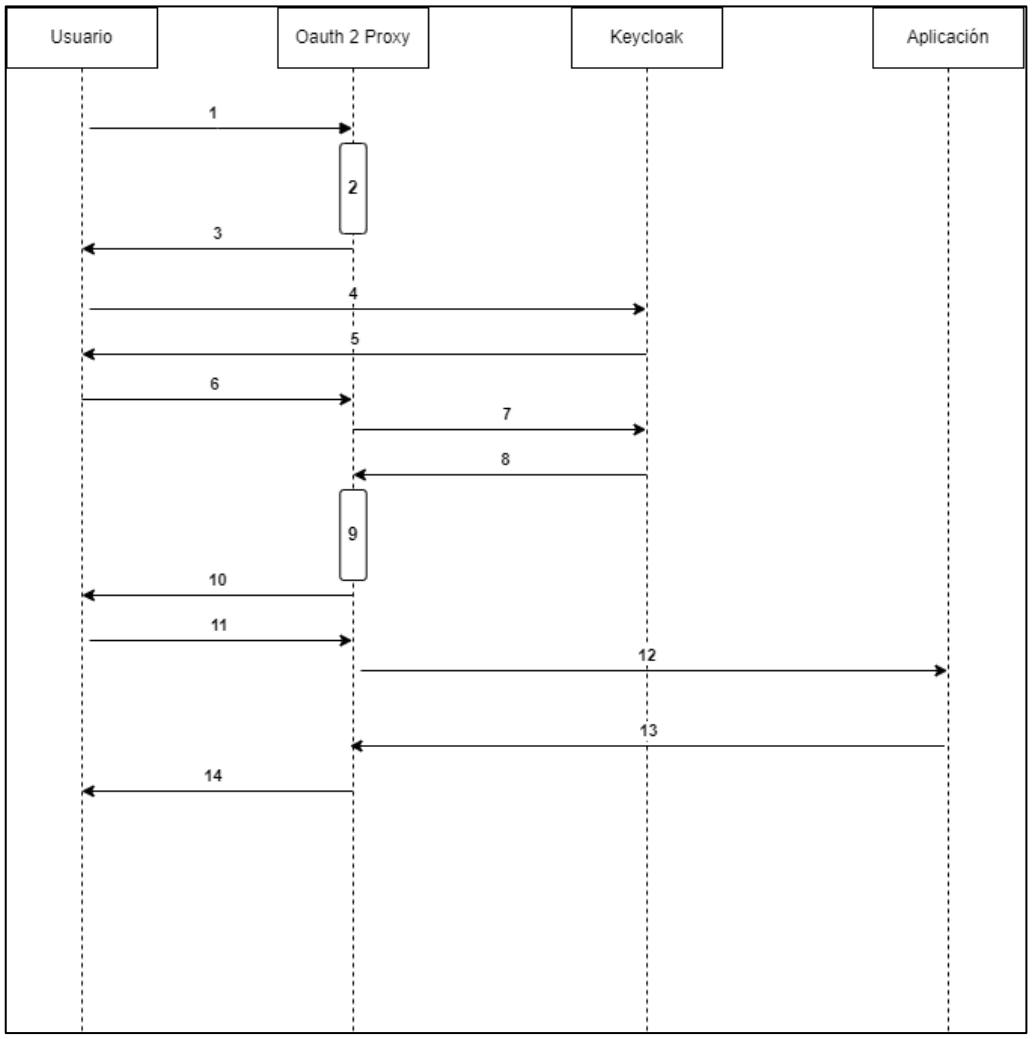

Figura 19. Diagrama de secuencia para el Caso 2.

1. El usuario realiza una petición a la aplicación.
2. Se comprueba la *cookie*, como no existe, se redirige el usuario al servidor de autorización (instancia de Keycloak).
3. El servidor de autorización devuelve el formulario de inicio de sesión.
4. El usuario envía el formulario con sus credenciales de inicio de sesión.

5. Se redirige al usuario a la instancia de Oauth2 Proxy junto al código de autorización (*auth code*).
6. Se envía el código de autorización.
7. Se intercambia el códido de autorización por un *token*.
8. La instancia de Keycloak devuelve el *id_token* y *access token* correspondiente al usuario.
9. La instancia de Oauth2 Proxy guarda el *token* en la sesión y genera una *cookie*.
10. Establece el nombre de la *cookie*.
11. Se vuelve a enviar la petición a la aplicación.
12. Se comprueba la *cookie* y se redirecciona el cliente a la aplicación.
13. Respuesta de la aplicación.
14. Se redirige la respuesta al cliente.

Caso 3. Usuario con permisos que accede a la aplicación estando autenticado.

A continuación, se describe el flujo de autenticación para un usuario que pertenece al grupo con los permisos correspondientes para acceder a una aplicación y que además, este usuario ya ha sido autenticado previamente.

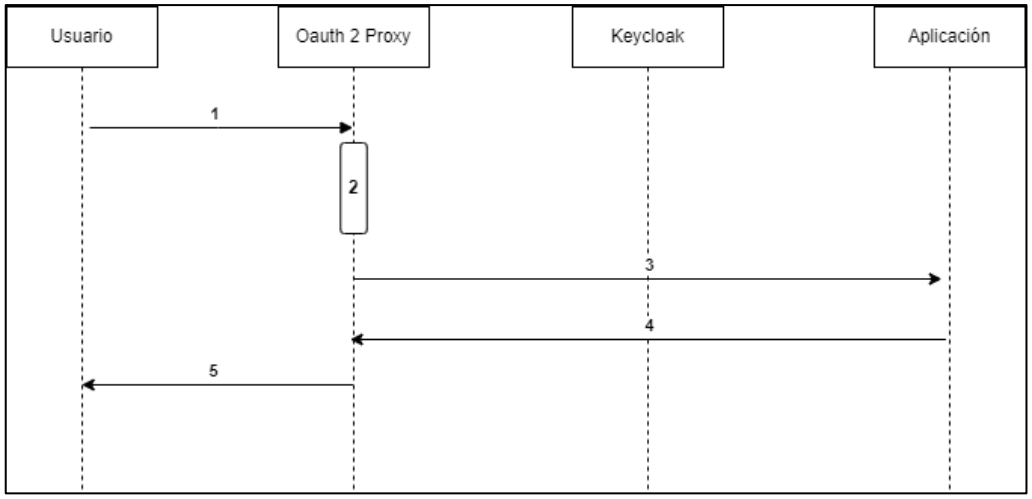

Figura 20. Diagrama de secuencia para el Caso 3.

1. El usuario realiza una petición a la aplicación
2. Se comprueba la *cookie*, como existe, se va a redirigir el usuario a la aplicación.
3. Se redirecciona el cliente a la aplicación.
4. Respuesta de la aplicación
5. Se redirige la respuesta al cliente.

Caso 4. Usuario sin permisos que accede a la aplicación sin estar autenticado.

A continuación, se describe el caso de uso para un usuario que no tiene permisos en la aplicación y quiere acceder a esta sin estar autenticado.

Este caso sigue el mismo diagrama de secuencia que para el caso 2 (figura 19) con el mismo flujo de mensajes y peticiones. La diferencia radica en el paso nº 13, donde la aplicación al comprobar los permisos del grupo al que pertenece el usuario, identifica que este no posee privilegios para acceder a ella, por lo que se le deniega el acceso en la petición de respuesta al cliente (paso nº 14).

Caso 5. Usuario sin permisos que accede a la aplicación sin estar autenticado y sin existir en la base de datos.

Por último, se describe el caso de uso para un usuario que quiere acceder a la aplicación, no está autenticado y no existe en la base datos, por lo que es el servidor de autorización el que le deniega el acceso directamente al cliente.

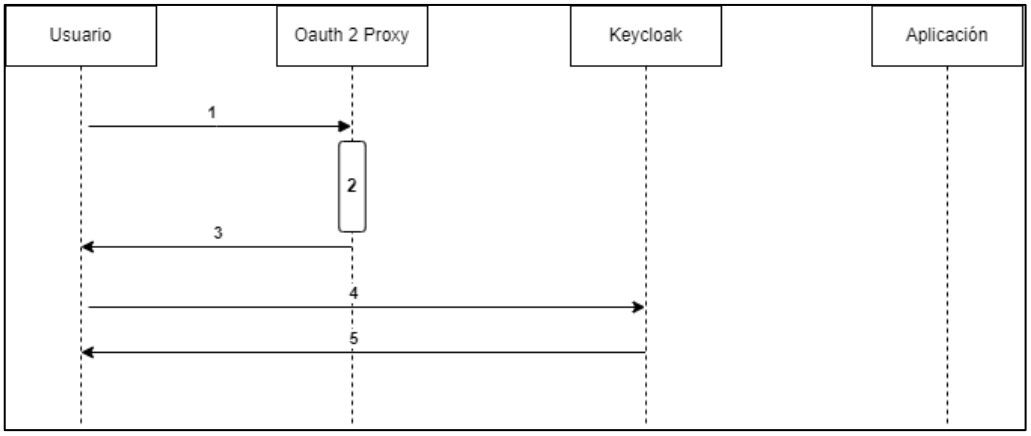

Figura 21. Diagrama de secuencia para Caso 5.

1. El usuario realiza una petición a la aplicación
2. Se comprueba la *cookie*, como no existe, se redirige el usuario al servidor de autorización.
3. Se devuelve la página de inicio de sesión del servidor de autorización.
4. El usuario envía el formulario con sus credenciales de inicio de sesión.
5. El servidor de autorización comprueba las credenciales y devuelve una respuesta de denegación de acceso porque no existen las credenciales en la base de datos.

4. IMPLEMENTACIÓN

En esta sección del trabajo, se lleva acabo el desarrollo de la solución propuesta.

En primer lugar, se instala la instancia de Keycloak, se crea un cliente, se realiza la conexión con el protocolo LDAP de la UPV y se configura el servicio API server de Kubernetes para que utilice ese cliente mediante el protocolo OIDC.

En segundo lugar, se despliega una instancia de Oauth2 Proxy, se configura para que funcione con Keycloak y se crean los roles necesarios en Kubernetes.

Por último, se instala el *plugin* kubelogin y se configura para que funcione con Keycloak.

4.1. Keycloak

4.1.1. Despliegue de la instancia

El despliegue de la instancia de Keycloak en el clúster se ha realizado de la siguiente forma:

1. Creación del *Namespace* "keycloak".

Primero se crea el *Namespace* "keycloak", donde se crearán todos los objetos de Kubernetes necesarios para la instalación.

Crear un *Namespace* dedicado para la aplicación es importante, ya que va a permitir tener un control de todos los componentes de la aplicación en un mismo lugar, por lo que a nivel administrativo es beneficioso. También es importante porque permitirá aislar la aplicación mejorando la seguridad y evitando interferencias con otras aplicaciones dentro del clúster.

2. Despliegue de la base de datos MySQL.

A continuación, se crea una base de datos para persistir toda la información de Keycloak, en este caso se ha utilizado una de MySQL, ya que ofrece una base de datos relacional muy ligera y con un mantenimiento muy extendido, aunque existen otras opciones también válidas como PostgreSQL.

Se crea un *Secret* que contenga la contraseña de acceso a la base de datos, un *PV* y un *PVC* para indicarle donde guardar la información en el disco, un *Deployment* donde se indican configuraciones varias del despliegue como la imagen y el montaje del *PVC*. Por último, se expone el despliegue con un *Service* de tipo *ClusterIP* ya que no necesita ser accedido desde fuera del clúster.

3. Creación de la base de datos.

Se accede al *Pod* lanzado por el *Deployment* y se crea la base de datos:

```
$ mysql -u root -p ******
$ create database keycloak
```

4. Despliegue de la instancia de Keycloak.

Se configura un *Deployment* de Keycloak indicando la imagen y algunas variables de entorno. En estas variables se tiene que indicar los datos de la base de datos que se acaba de crear para que Keycloak se pueda conectar y empezar a persistir sus datos ahí (Anexo 2).

Una vez creado el *Deployment*, se configura un *Service* para exponer el despliegue, de tipo *NodePort* (Anexo 3).

Finalmente, se lanza un *Ingress* que permitirá el acceso desde el exterior a la instancia de Keycloak (Anexo 4).

4.1.2. Configuración en la plataforma

A continuación, se detallan algunas configuraciones iniciales necesarias que se deben realizar dentro de la instancia de Keycloak desplegada. Todas las figuras que se detallan a continuación han sido tomadas de la interfaz web de la consola de administrador de la instancia desplegada de Keycloak [15].

1. Crear Realm.

En primer lugar, se crea un nuevo *Realm.* Al hacer esto podemos aislar configuraciones de una aplicación a otra.

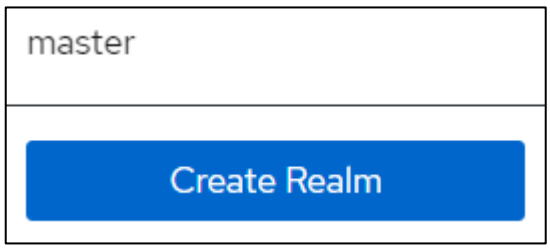

Figura 22. Botón crear *Realm*.

2. Conexión de Keycloak a LDAP.

Se accede al apartado de federación de usuario (*user federation*). Desde aquí existe la capacidad de agregar y sincronizar usuarios desde fuentes externas al servidor de Keycloak.

Por lo tanto, creamos una nueva fuente que importe los usuarios desde el LDAP de la UPV.

Como se ha mencionado anteriormente (punto 3.2.2. Caso 1), la jerarquía de LDAP obliga al administrador a que, si quiere importar usuarios de dos dominios distintos, tenga que tratarlos como dos DC diferentes. Es por esto, por lo que se deben crear dos fuentes distintas en Keycloak también.

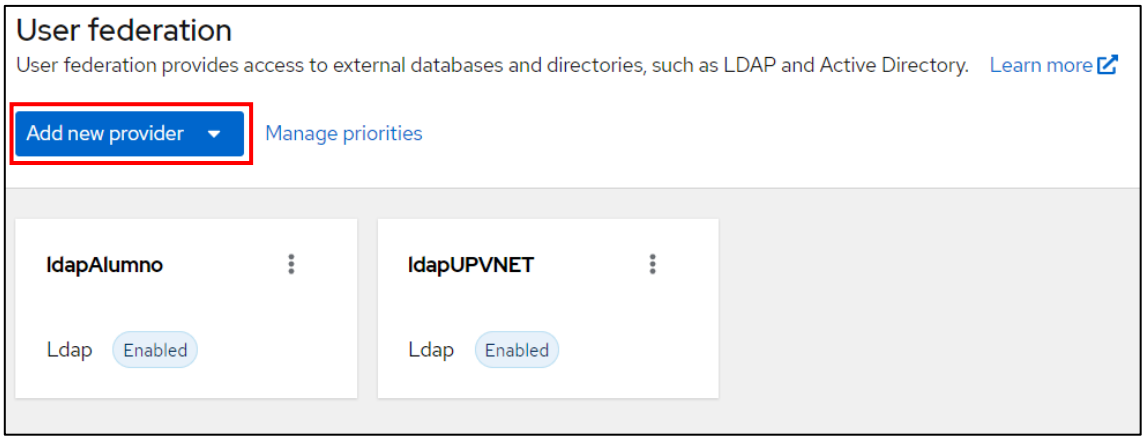

Figura 23. Apartado de federación de usuario.

La primera de ellas importa usuarios del dominio “UPVNET”:

```
(objectClass=person)(memberOf=CN=keycloak,OU=AI2,DC=upvnet,DC=upv,DC=es)
```

La segunda de ellas importa usuarios del dominio “Alumno”:

```
(objectClass=person)(memberOf=CN=keycloak-alu,OU=AI2,OU=Departamentos,DC=alumno,DC=upv,DC=es
```

3. Importación de los usuarios del LDAP y comprobación.

A continuación, se sincroniza la base de datos de usuarios de Keycloak para que importe los usuarios que casen con los filtros de LDAP configurados anteriormente.

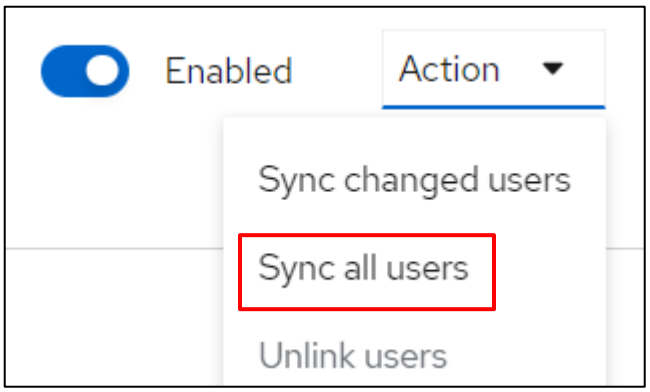

Figura 24. Botón sincronización de usuarios.

Así mismo, se agregan los usuarios a los grupos correspondientes y, si no existen los grupos, se crean.

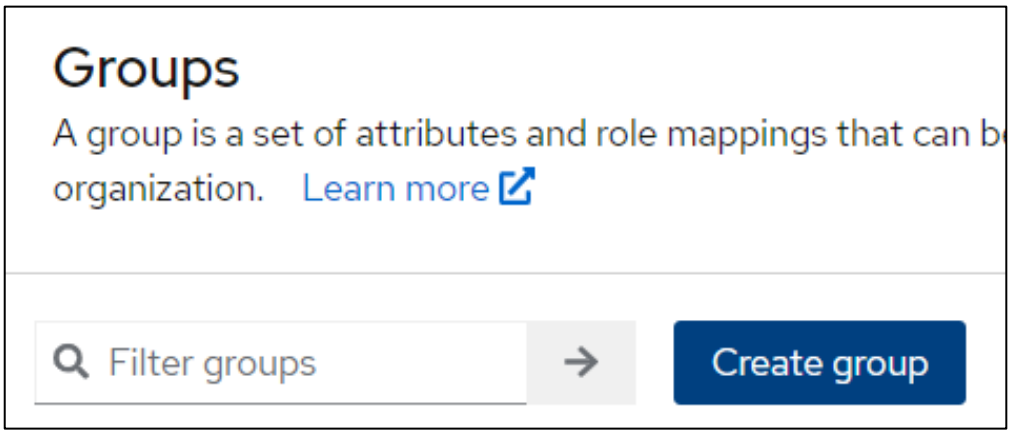

Figura 25. Botón crear grupo.

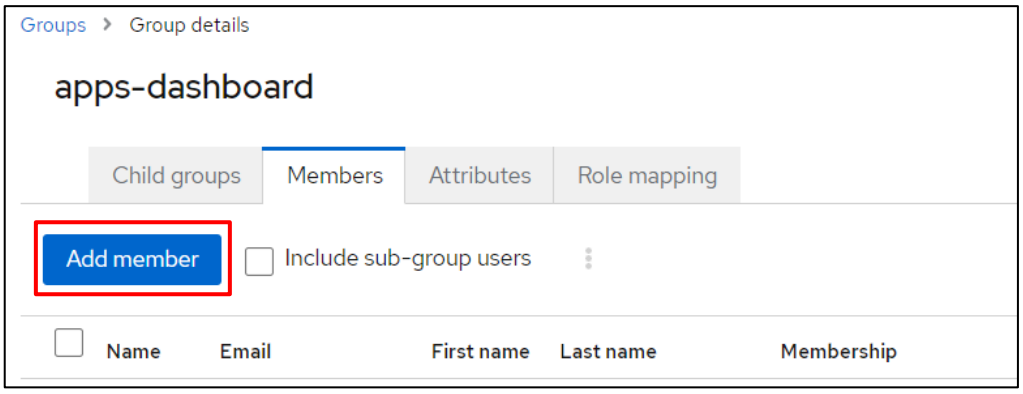

Figura 26. Botón agregar usuarios a grupo.

4. Crear cliente.

Después, se crea un cliente que representará nuestro clúster de Kubernetes. Posteriormente, se modificará y se importará el cliente en el manifiesto del API server para que, de esta forma, nuestro clúster se configure correctamente con el protocolo OIDC y permita reconocer los *tokens* de Keycloak.

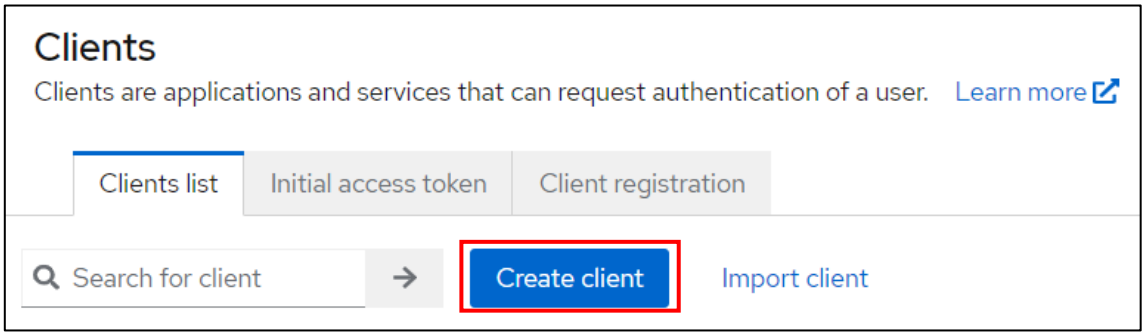

Figura 27. Botón crear cliente.

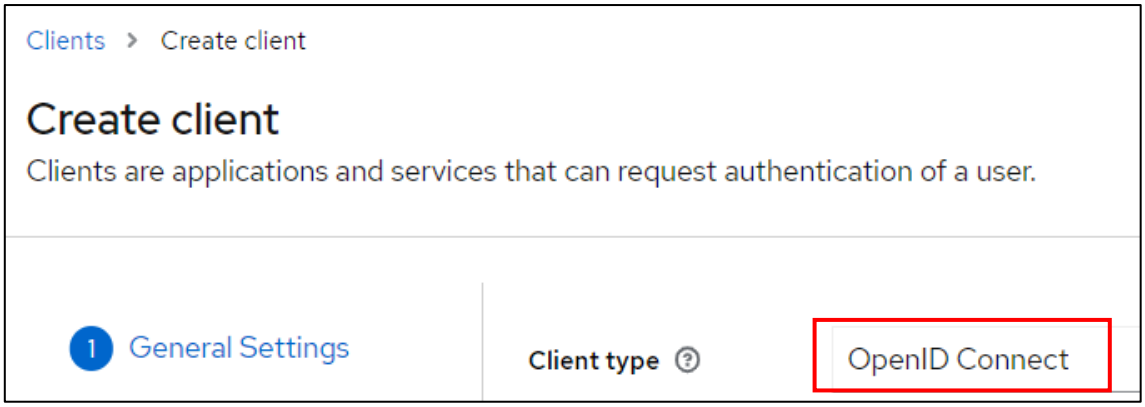

Figura 28. Selección del tipo de cliente.

El cliente debe ser de tipo OIDC y además, dentro de él se crea un *client scope* (normalmente ya hay uno dedicado) y un *mapper* de tipo *group membership* llamado “groups”.

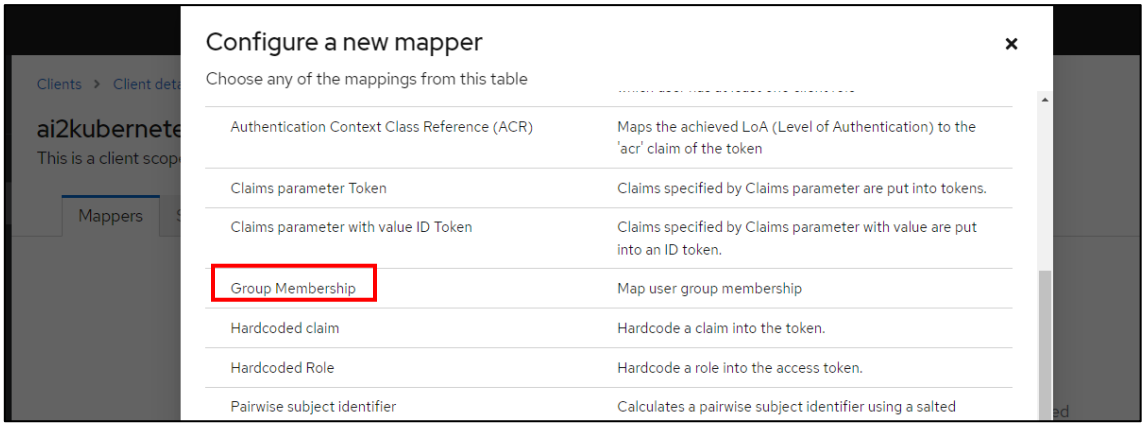

Figura 29. Selección de nuevo *mapper*.

4.1.3. Configuración en Kubernetes

Kubernetes ofrece la posibilidad de configurar el servidor con un proveedor de identidades.

En este apartado se especifican las configuraciones que hacen falta dentro de nuestro clúster de Kuberentes para que el API server pueda confiar en los *tokens* expedidos por Keycloak.

A este manifiesto, habrá que añadirle algunos parámetros nuevos como:

- **Oidc-issuer**: donde se especifica la dirección de la instancia de nuestro proveedor de identidades.

- **Oidc-client**: donde se indica el nombre del cliente creado en Keycloak anteriormente.

- **Oidc-username-claim**: donde se detalla qué campo del *id_token* deberá utilizar el clúster como nombre de usuario.

- **Oidc-groups-claim**: donde se especifica el nombre del *claim* (petición) que contiene la información de los grupos o roles a los que pertenece un usuario en Keycloak.

- **Oidc-ca-file**: aquí se debe especificar la ruta del archivo de la autoridad certificadora de Keycloak para que así el clúster confíe en la instancia de nuestro proveedor.

A continuación, se modifica el manifiesto del API server que se encuentra en */etc/kubernetes/manifests/kube-apiserver.yaml*, y se añaden las opciones anteriores:

```
--oidc-issuer-
url=https://keycloak.ai2.upv.es/realms/ai2realm
--oidc-client-id=ai2kubernetes
--oidc-username-claim=preferred_username
--oidc-groups-claim=groups
--oidc-ca-file=/etc/kubernetes/ssl/cadGEANT.pem
```

De esta forma queda habilitada la autenticación mediante el protocolo OpenID Connect en el clúster de Kubernetes.

4.2. Oauth2 Proxy

Una vez configurado el API server del clúster para atender al protocolo OIDC, el siguiente paso es instalar y configurar la instancia de Oauth2 Proxy.

Existe la posibilidad de realizar la instalación de la instancia por sí sola mediante *Helm.* Debido a la popularidad de la tecnología, hay aplicaciones que ya integran una instancia de Oauth2 Proxy en su instalación como es el caso de *Kubeapps*.

Kubeapps [16], es una plataforma de gestión de aplicaciones en Kubernetes que ofrece un interfaz intuitiva y amigable. Como se quiere instalar el servicio, se aprovecha la instancia de Oauth2 Proxy que se despliega para integrarla en la solución final.

4.2.1. Configuración en el proveedor.

En primer lugar es necesario crear un segundo cliente en Keycloak para esta aplicación [17].

De la misma forma que para el cliente anterior, se accede al apartado de clientes (*clients*) y se agrega uno nuevo con los siguientes parámetros:

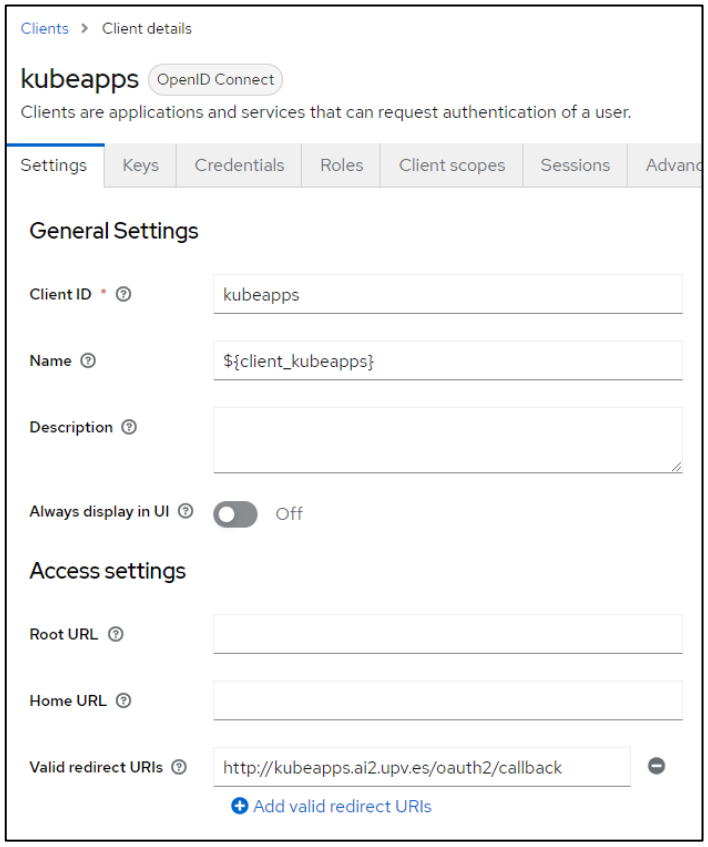

Figura 30.a. Configuración del cliente de *Kubeapps*.

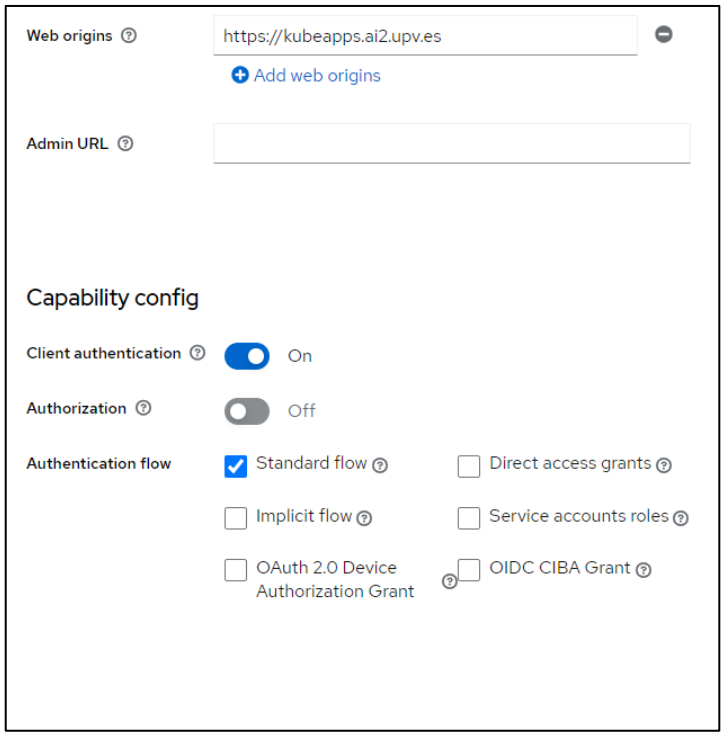

Figura 30.b. Configuración del cliente de *Kubeapps*.

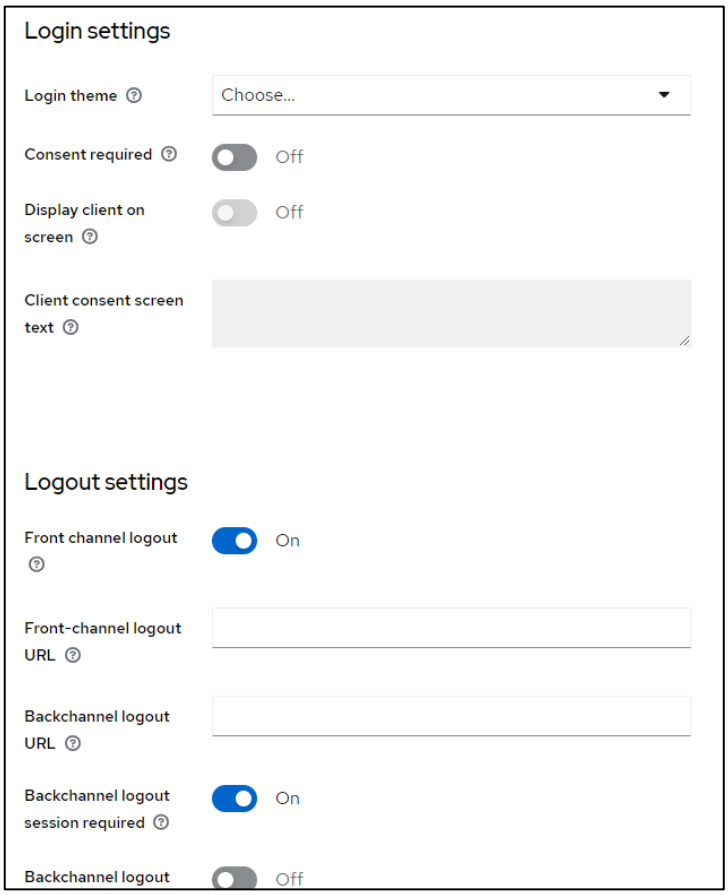

Figura 30.c. Configuración del cliente de *Kubeapps*.

En el apartado de *Valid redirect URIs* y *Valid post logout redirect URIs* se añade el *hostname* de la instancia de *Kubeapps*. Por último, hay que asegurarse de nuevo que en el apartado *Client Scopes* estén habilitados los *scopes* email y *groups*, y adicionalemnte, se añade el *scope* para el cliente del clúster creado anteriormente.

4.2.2. Despliegue de la instancia

Una vez creado el cliente en *Keycloak*, el siguiente paso es proceder a la instalación de la instancia.

La instalación de *Kubeapps* (v8.1.8) se ha realizado mediante *Helm* (v3.5.4), y para ello, primero es necesario agregar el repositorio de *Bitnami*:

```
$ helm repo add bitnami https://charts.bitnami.com/bitnami
```

Y posteriormente instalarlo en su correspondiente *Namespace*, pasándole un fichero *values.yaml* (cuyo contenido se detalla en el Anexo 5) como parámetro al comando:

```
$ helm install kubeapps --namespace kubeapps
bitnami/kubeapps --version 8.1.8 --values ./values.yaml
```

Es importante destacar que en el parámetro *client id* del fichero *values.yaml* se tiene que añadir el nombre del cliente de *Kubeapps* creado anteriormente.

Adicionalmente, en el parámetro *oidc-issuer-url* se añadirá la URL de nuestra instancia de *Keycloak*.

Y por último, se crea un objeto de *Ingress* para la instancia de *Kubeapps*, donde se le agrega un nombre de *host* adecuado, además de la configuración del secreto para obtener cifrado TLS.

```
ingress:
  enabled: true
  ingressClassName: "nginx"
  hostname: kubeapps.ai2.upv.es
  tls: true
  extraTls:
    - hosts:
        - kubeapps.ai2.upv.es
      secretName: kubeapps-https-cert
  annotations:
    nginx.ingress.kubernetes.io/proxy-read-timeout: "600"
```

Figura 31. Regla de *Ingress* del fichero *values.yaml* (Anexo 5).

4.3. Kubelogin

Antes de realizar la instalación del *plugin*, primero es importante contar con un fichero *config* dentro de la carpeta */.kube.* Este fichero es utilizado por la interfaz de línea de comandos *kubectl* para obtener información sobre cómo conectarse al clúster. Contiene detalles de configuración como la dirección del clúster, las credenciales de autenticación y otros ajustes relevantes.

La instalación del *plugin* de kubelogin (v1.27.0) se ha realizado a través de *Krew* (v0.4.3), un manager de *plugins* para la interfaz de línea de comandos *kubectl.*

Primero, hay que instalar *Krew*. Siguiendo la documentación oficial, se instala el gestor y se exporta la variable de entorno [18].

Una vez instalado *Krew*, se instala *kubelogin* siguiendo la documentación del repositorio oficial [5].

```
$ kubectl krew install oidc-login
```

Después de instalar la herramienta se configura utilizando el comando *setup*. Para ello se ejecuta el comando indicando la dirección del proveedor, el *client id*, el *client secret*, y la ruta del archivo con la clave de la CA correspondiente de Keycloak (azul los parámetros que se solicitan en la petición y en rojo el valor que se le está asignando):

```
$ kubectl oidc-login setup
--oidc-issuer-url=https://keycloak.ai2.upv.es/realms/ai2realm
--oidc-client-id=ai2kubernetes
--oidc-client-secret=****************
--certificate-authority .kube/cadGEANT.pem
```

Al ejecutar el comando se abrirá el navegador y pedirá las credenciales para autenticar al usuario que está realizando la instalación.

Posteriormente, se solicita que se ejecuten algunos comandos extra para terminar de configurar el cliente, como configurar el API server para que admita el protocolo OIDC (realizado en apartado 4.1.3).

Entre estos comandos, se destaca la creación de un contexto para poder comunicarse con el clúster:

```
$ kubectl config set-credentials oidc
--exec-api-version=client.authentication.k8s.io/v1beta1
--exec-command=kubectl
--exec-arg=oidc-login
--exec-arg=get-token
--exec-arg=--oidc-issuer-
url=https://keycloak.ai2.upv.es/realms/ai2realm
--exec-arg=--oidc-client-id=ai2kubernetes
--exec-arg=--oidc-client-secret=****************
--exec-arg=--certificate-authority=.kube/cadGEANT.pem
```

Seguidamente, se establece el contexto creado como contexto actual:

```
$ kubectl config set-context --current --user=oidc
```

Finalmente, se pueden realizar consultas al clúster con normalidad. Si la *cookie* expira, en la siguiente consulta se le solicitará al usuario introducir sus credenciales de nuevo.

De esta forma, se integra un AAI a nivel de línea de comandos en un clúster de Kubernetes.

4.4. Establecimiento del modelo de control de acceso

Una vez desplegada la instancia del proveedor de identidad y del *proxy* inverso, es momento de establecer un modelo de control de acceso mediante el uso de RBAC de Kubernetes.

Como se ha explicado anteriormente (apartado 2.1.1) Kubernetes ofrece numerosos objetos para gestionar y controlar los privilegios de los usuarios que acceden a un clúster, pero es importante establecer un esquema que organice a los usuarios en roles y establecer permisos sobre esos roles.

En el problema planteado en este trabajo, se quiere gestionar el acceso de los usuarios a las aplicaciones y además establcer un control sobre a qué recursos tienen acceso estos usuarios dentro de las aplicaciones.

En vez de establecer roles sobre usuarios, se establecen roles sobre grupos a los que pertenecerán los usuarios.

Se plantea el siguiente modelo:

1. Grupo control de acceso

Se crea un grupo para gestionar el acceso controlado a las aplicaciones cuya nomenclatura es "apps-<nombre_de_la_aplicación>". Por ende, existirá un grupo para cada aplicación del clúster.

Por ejemplo:

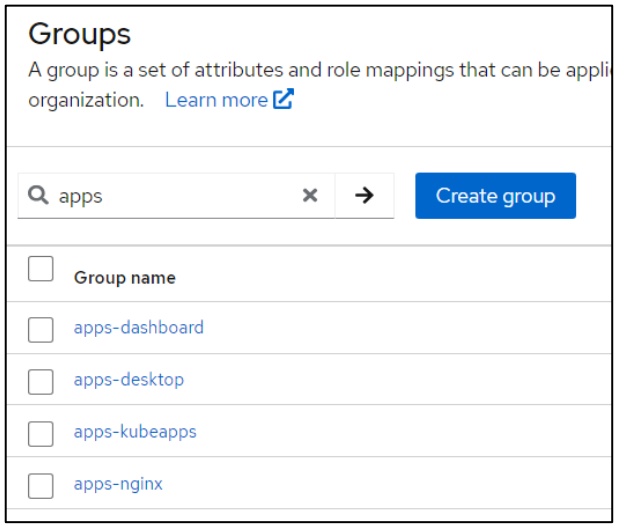

Figura 32. Listado de grupos de control de acceso a aplicaciones.

Como se observa en la figura 32, existe un grupo para cada aplicación. El grupo "apps-dashboard" gestiona el acceso de los usuarios a la aplicación del Dashboard de Kubernetes, el grupo "apps-kubeapps" gestiona el acceso de los usuarios a la aplicación de *Kubeapps* etc.

Si el usuario que quiere acceder a la aplicación no pertenece al grupo correspondiente, se le denegará el acceso.

Para que los grupos que gestionan el acceso tengan efectividad en el sistema, es necesario rellenar algunos campos de los objetos de *Ingress* que exponen las aplicaciones del clúster.

En la siguiente figura, se muestra la regla de *Ingress* para la aplicación del Dashboard de Kubernetes.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: kubernetes-dashboard-ingress-kubeapps
  namespace: kube-system
  annotations:
        nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
        nginx.ingress.kubernetes.io/rewrite-target: /$2
        nginx.ingress.kubernetes.io/auth-url: "https://kubeapps.ai2.upv.es/oauth2/auth?allowed_groups=apps-dashboard"
        nginx.ingress.kubernetes.io/auth-signin: https://kubeapps.ai2.upv.es/oauth2/start
        nginx.ingress.kubernetes.io/auth-response-headers: authorization
        nginx.ingress.kubernetes.io/proxy-buffer-size: "16k"
        nginx.ingress.kubernetes.io/configuration-snippet: rewrite ^(/dashboard3)$ $1/ redirect;
        nginx.ingress.kubernetes.io/default-backend: ingress-nginx-defaultbackend
        nginx.ingress.kubernetes.io/custom-http-errors: 403,503


spec:
  ingressClassName: nginx
  tls:
  - hosts:
      - kb-cluster.ai2.upv.es
    secretName: kb-cluster-secret
  rules:
  - host: kb-cluster.ai2.upv.es
    http:
      paths:
      - path: /dashboard3(/|$)(.*)
        pathType: Prefix
        backend:
          service:
            name: kubernetes-dashboard
            port:
              number: 443
```

Figura 33. Regla de *Ingress* del Dashboard de Kubernetes.

En el apartado de *annotations* se permite añadir configuraciones extra al controlador de *Ingress*. Entre todos estos campos se destaca el uso del campo *auth-url* y *auth-signin*.

```yaml
nginx.ingress.kubernetes.io/auth-url: "https://kubeapps.ai2.upv.es/oauth2/auth?allowed_groups=apps-dashboard"
nginx.ingress.kubernetes.io/auth-signin: https://kubeapps.ai2.upv.es/oauth2/start
```

Figura 34. Anotaciones de autenticación en la regla de *Ingress*.

En el campo *auth-signin* se establece el *endpoint* de la instancia de Oauth2 Proxy que da comienzo al protocolo una vez llega una petición. En este caso el *endpoint* es “/oauth2/start”.

El campo *auth-url* se utiliza para especificar la URL de autenticación. Como se observa en la URL establecida para el ejemplo, se le añade al final al parámetro *allowed_groups* el nombre del grupo al que se permitirá el acceso. Como se trata de la regla de *Ingress* para la aplicación del Dashboard de Kubernetes en ese campo se escribe el nombre del grupo que correspondiente a esa aplicación, “apps-dashboard”.

Es de esta forma como Kubernetes permite establecer un control de acceso a los servicios seleccionados.

2. Grupo de gestión de permisos

Se crea un grupo para gestionar los recursos a los que puede acceder el usuario dentro del clúster y qué permisos tendrá dentro de las aplicaciones a las que accede. Su nomenclatura será "ns-<nombre_privilegios>", siendo "nombre_privilegios" un nombre que permita describir los privilegios que obtendrá el usuario al que accede.

Por ejemplo:

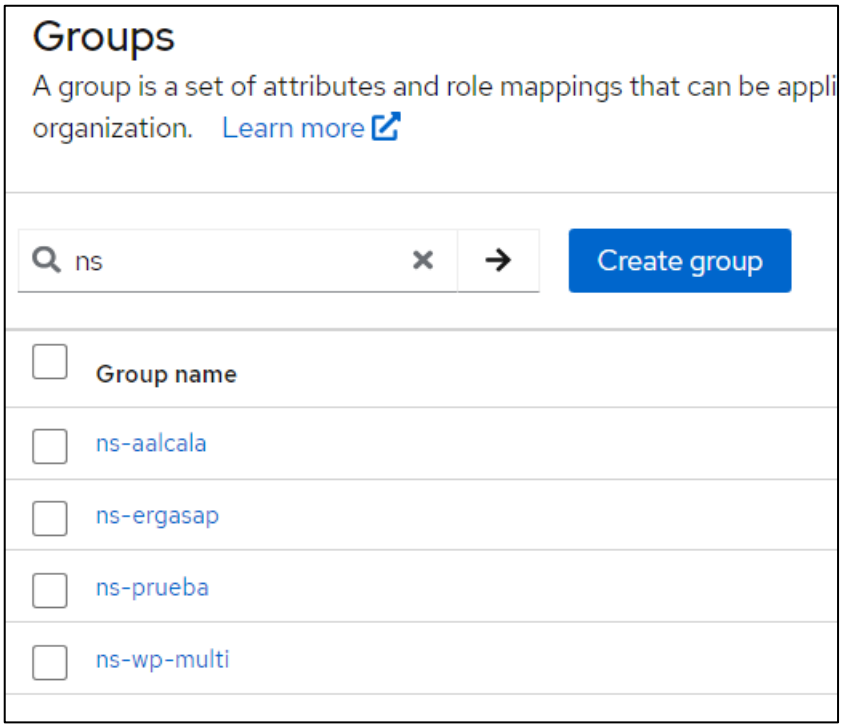

Figura 35. Listado de grupos de gestión de permisos de usuarios.

El grupo "ns-wp-multi" define un grupo cuyos permisos serán de creación, modificación y eliminación de objetos solamente en el *Namespace* llamado "wp-multi".

Para que los grupos que gestionan los recursos a los que acceden los usuarios tengan validez en el sistema, es necesario aplicar los mecanismos de RBAC que ofrece Kubernetes.

Si lo que se quiere es restringir un usuario a un cierto *Namespace* es necesario crear un *RoleBinding* que representará la relación entre *ClusterRole* y *Grupo*.

```
kind: RoleBinding
metadata:
  name: ns-wp-multi-rolebinding
  namespace: wp-multi
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: ns-wp-multi
```

Figura 36. Manifiesto del *RoleBinding*.

En el apartado *roleRef* se establece el rol que se quiere aplicar. En este caso, se trata de un rol de tipo *ClusterRole* llamado *cluster-admin.* El rol *cluster-admin* es uno de los roles que ya vienen predefinidos en la creación del clúster y define permisos de administrador.

Seguidamente, en el apartado *subjects,* se indica cual es el sujeto al que se le aplicará el rol. En este contexto, se aplica el rol sobre el grupo *ns-wp-multi* que se había definido previamente en Keycloak.

Por último, para especificar que los usuarios pertenecientes al grupo *ns-wp-multi* van a tener permisos de *cluster-admin* sobre el *Namespace* “wp-multi” se debe indicar este en el apartado de *metadata* del manifiesto.

De este modo es como se logra una gestión de los permisos de los usuarios en Kubernetes.

5. VALIDACIÓN DE LOS ESCENARIOS

En este apartado se comprueba y se evalúa la solución propuesta sobre un clúster de Kubernetes en un entorno de producción real. El objetivo de esta sección es demostrar la efectividad y viabilidad de la arquitectura definida.

Como se ha mencionado anteriormente, se han definido 3 escenarios posibles que tienen lugar en el sistema. Es importante validar los escenarios para garantizar que el sistema cumple con los requisitos propuestos y así, poder identificar y corregir errores.

Para cada escenario se acceden a los microservicios mediante el usuario de prueba “Prueba”. Se inicia la sesión a través del siguiente formulario:

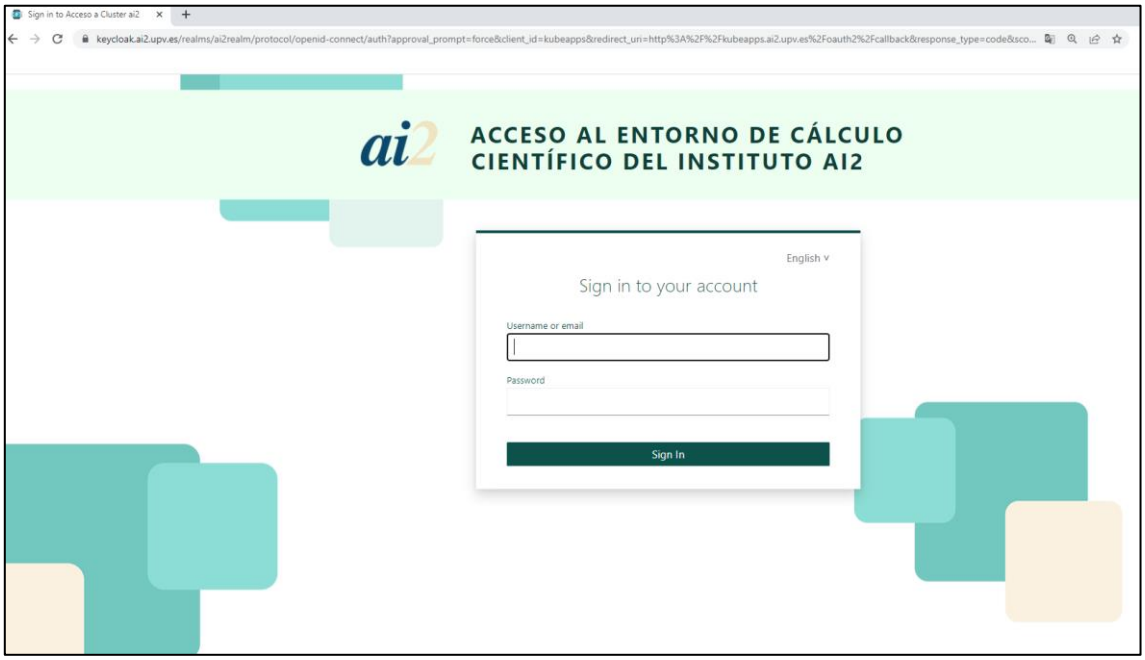

Figura 37. Página de inicio de sesión.

5.1. Acceso vía HTTP a microservicios

Este escenario define un acceso vía HTTP a un microservicio protegido en el que no se necesita que se limiten los recursos a los que se acceden del clúster, como puede ser una página web estática.

Para ello, al intentar acceder al *endpoint* del microservicio, se le solicitan las credenciales al usuario (figura 37). Si pertenece al grupo con permisos de acceso, se le otorga; en caso contrario, se le deniega el acceso.

En este caso, este servicio que ofrece una página web de NGINX es accesible solo para los usuarios pertenecientes al grupo “apps-nginx” (figura 38).

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app-ingress
  namespace: default
  annotations:
        nginx.ingress.kubernetes.io/rewrite-target: /$1
        nginx.ingress.kubernetes.io/auth-url: https://kubeapps.ai2.upv.es/oauth2/auth?allowed_groups=apps-nginx
        nginx.ingress.kubernetes.io/auth-signin: https://kubeapps.ai2.upv.es/oauth2/start
        nginx.ingress.kubernetes.io/auth-response-headers: Authorization
        nginx.ingress.kubernetes.io/default-backend: ingress-nginx-defaultbackend
        nginx.ingress.kubernetes.io/custom-http-errors: 403,503

spec:
  ingressClassName: nginx

  rules:
  - host: kb-cluster.ai2.upv.es
    http:
      paths:
      - path: /nginx/*
        pathType: Prefix
        backend:
          service:
            name: nginx-app
            port:
              number: 80
```

Figura 38. *Ingress* del microservicio NGINX.

Si el usuario “Prueba” pertenece al grupo “apps-nginx” se muestra la página de bienvenida (figura 39). De lo contrario se le deniega el acceso devolviendo una respuesta con error 403 (figura 40).

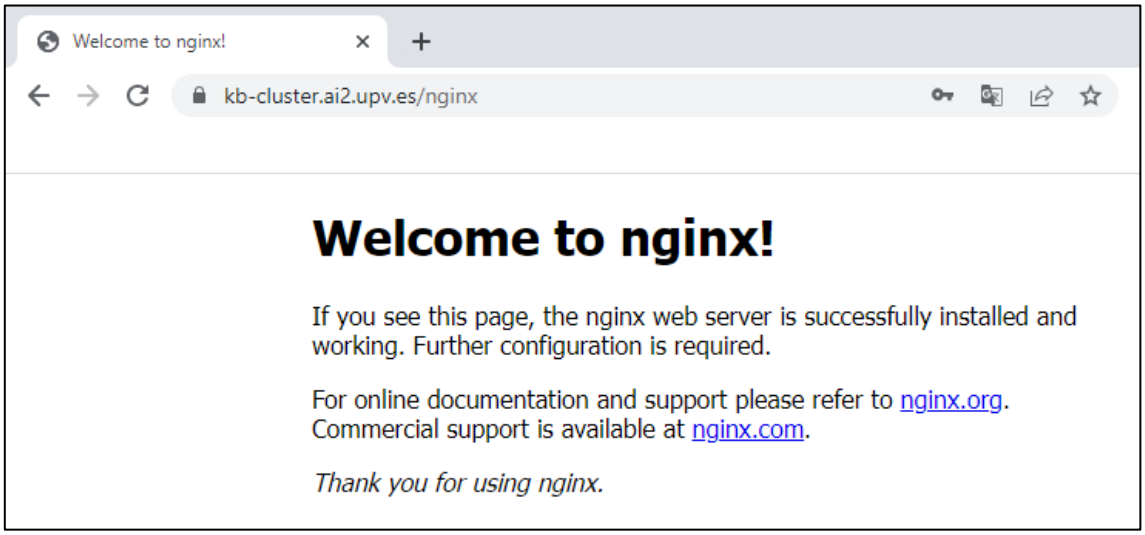

Figura 39. Página de bienvenida.

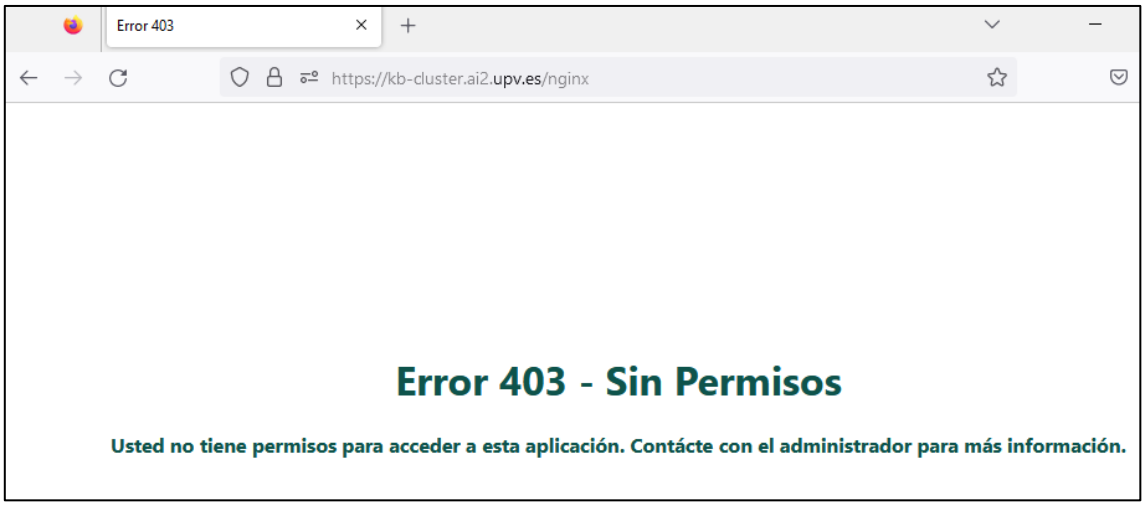

Figura 40. Página de denegación de acceso (Error 403).

5.2. Acceso vía HTTP a microservicio con interacción con Kubernetes

Este escenario define un acceso vía HTTP a un microservicio protegido, que al igual que en el primer escenario, dependiendo del grupo al que pertenezca se le permitirá el acceso o no. Adicionalmente, el usuario podrá interaccionar con el clúster desde la aplicación web, por lo tanto, se limitan sus permisos mediante el uso de RBAC de Kubernetes.

En este caso, este servicio ofrece acceso a la aplicación del Dashboard de Kubernetes que será accesible solo para los usuarios pertenecientes al grupo "apps-dashboard" (figura 41).

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: kubernetes-dashboard-ingress-kubeapps
  namespace: kube-system
  annotations:
        nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
        nginx.ingress.kubernetes.io/rewrite-target: /$2
        nginx.ingress.kubernetes.io/auth-url: "https://kubeapps.ai2.upv.es/oauth2/auth?allowed_groups=apps-dashboard"
        nginx.ingress.kubernetes.io/auth-signin: https://kubeapps.ai2.upv.es/oauth2/start
        nginx.ingress.kubernetes.io/auth-response-headers: authorization
        nginx.ingress.kubernetes.io/proxy-buffer-size: "16k"
        nginx.ingress.kubernetes.io/configuration-snippet: rewrite ^(/dashboard3)$ $1/ redirect;
        nginx.ingress.kubernetes.io/default-backend: ingress-nginx-defaultbackend
        nginx.ingress.kubernetes.io/custom-http-errors: 403,503


spec:
  ingressClassName: nginx
  tls:
  - hosts:
      - kb-cluster.ai2.upv.es
    secretName: kb-cluster-secret
  rules:
  - host: kb-cluster.ai2.upv.es
    http:
      paths:
      - path: /dashboard3(/|$)(.*)
        pathType: Prefix
        backend:
          service:
            name: kubernetes-dashboard
            port:
              number: 443
```

Figura 41. *Ingress* del microservicio Dasboard de Kubernetes.

Si el usuario “Prueba” pertenece al grupo “apps-dashboard” se muestra la página principal (figura 42). De lo contrario se le deniega el acceso devolviendo una respuesta con error 403 (figura 40).

Una vez iniciada la sesión correctamente, dependiendo de los permisos del usuario, será capaz de ver, modificar o eliminar ciertos recursos del clúster.

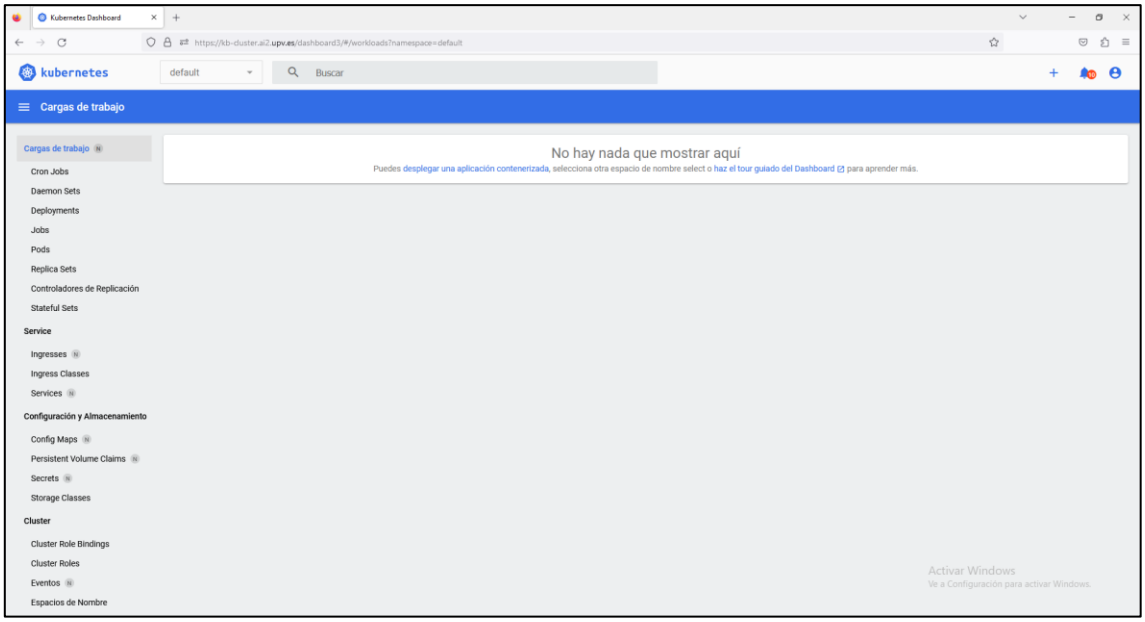

Figura 42. Página principal del Dashboard de Kubernetes.

Para este ejemplo el usuario “Prueba” también pertenece al grupo “ns-wp-multi” cuyos permisos son de creación, modificación y eliminación de objetos en el *Namespace* “wp-multi”. Tendrá acceso restringido al resto de *Namespaces* por lo que todos estos permisos se verán reflejados en la aplicación principal.

Si se accede al Namespace “wp-multi” en la aplicación, se pueden observar los objetos despegados:

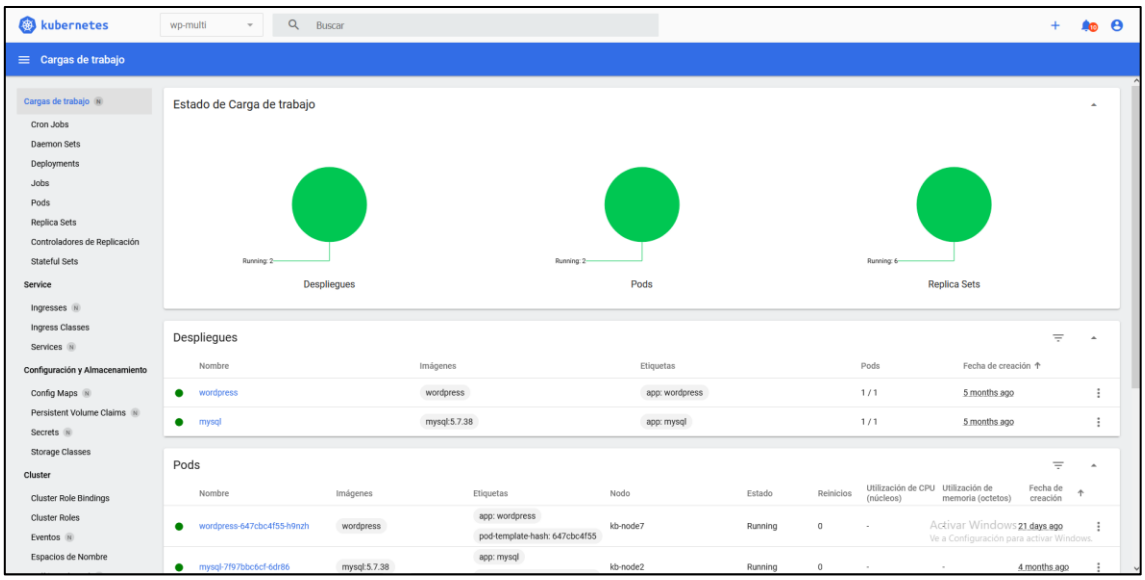

Figura 43. Pestaña de cargas de trabajo para el *Namespace* “wp-multi”.

En cambio, si se accede a cualquier otro *Namespace*, por ejemplo al *Namespace* “default”, la aplicación restringirá al usuario y mostrará los siguientes mensajes de error:

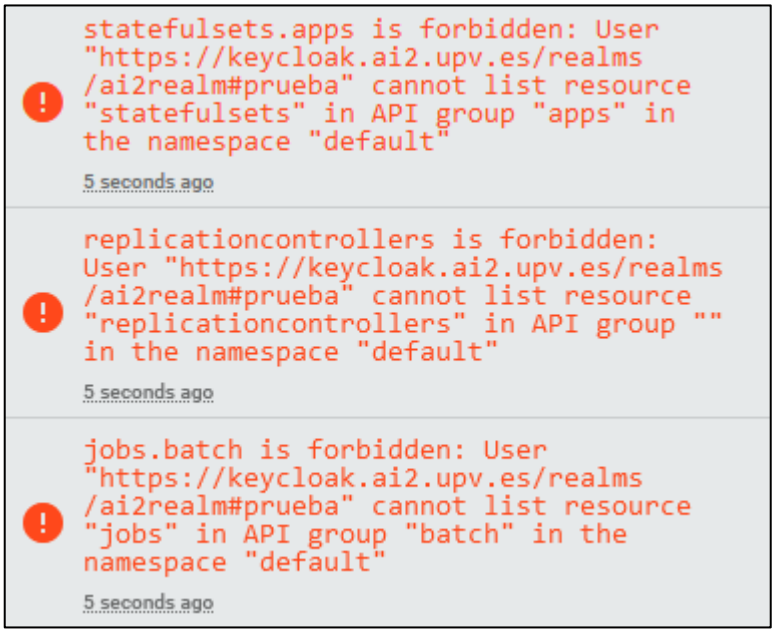

Figura 44. Error en el *Namespace* “default”.

5.3. Acceso a recursos del clúster por línea de comandos

Se considera un último escenario, en el que un usuario desea interactuar con el clúster mediante la interfaz de línea de comandos *kubectl*. Este usuario debe tener el fichero de configuracion *kubeconfig* configurado con el usuario *oidc* y, opcionalmente, en el contexto por defecto debe encontrarse este usuario (apartado 4.3).

Cuando un usuario intenta hacer una consulta al clúster, el *plugin* de kubelogin, se encargará de abrir una ventana en el navegador preguntándole a este por sus credenciales.

Al hilo del caso anterior, se autentica en la plataforma con el usuario “Prueba”. En este caso la consulta que se intenta hacer es la de obtener el listado de *Deployments* del *Namespace* “wp-multi”.

Al ejecutar el comando “kubectl get deployments -n wp-multi” aparece el formulario de inicio de sesión que solicita las credenciales (figura 45):

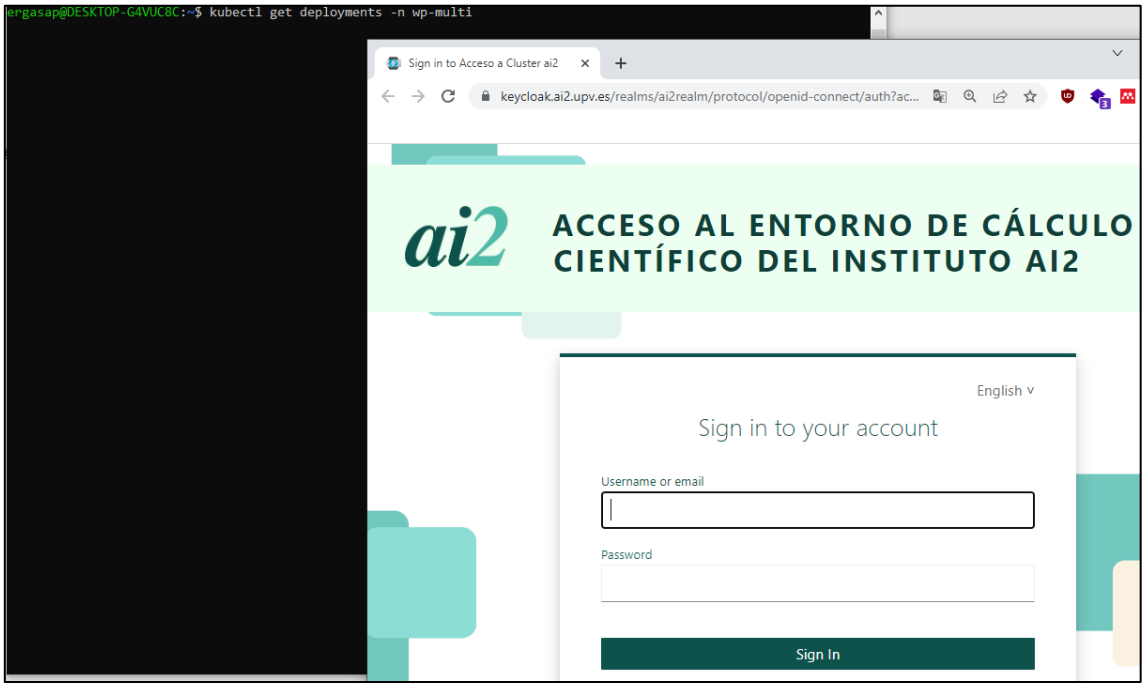

Figura 45. Ventana de inicio de sesión.

Una vez inciada la sesión, el API server responderá a la petición, aceptándola o denegándola, según los permisos que tenga el grupo al que pertenece el usuario. Como ya se ha explicado, el usuario “Prueba” tiene permisos totales sobre este *Namespace*, por lo que el API server responde sin problemas (figura 46):

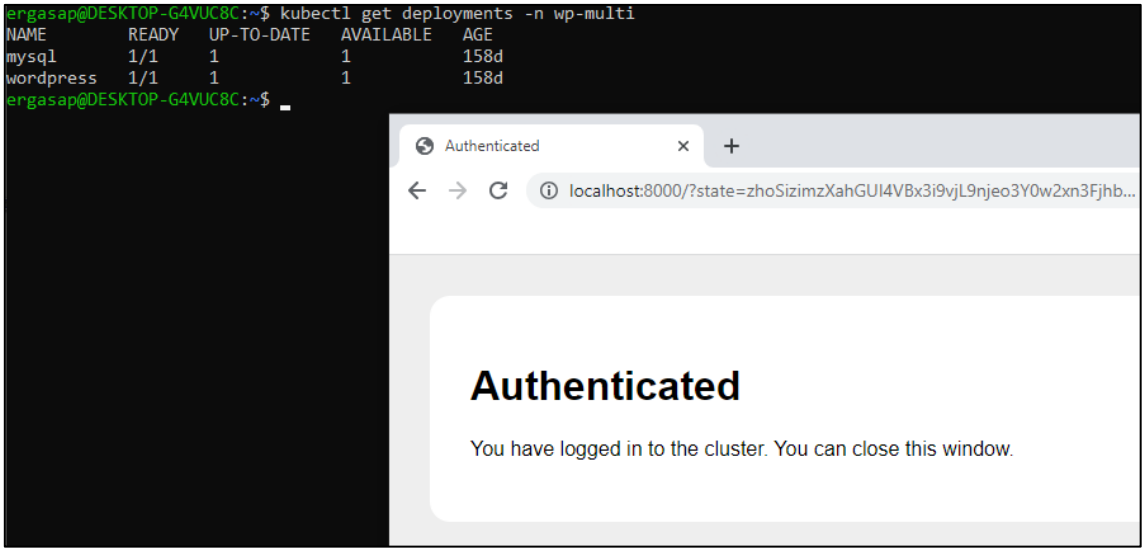

Figura 46. Respuesta aceptada.

Si se intenta acceder a información sobre cualquier otro *Namespace,* como por ejemplo el *Namespace* “default”, el API server comprueba las credenciales y deniega el acceso mostrando el siguiente error (figura 47):

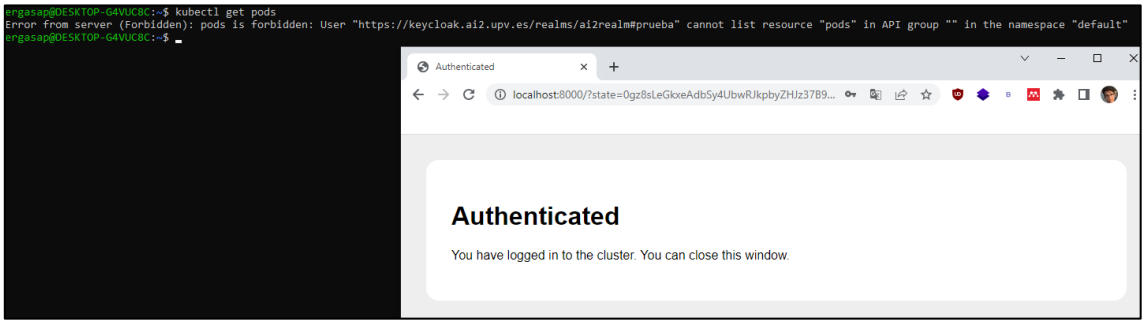

Figura 47. Respuesta denegada.

6. DISCUSIÓN

En este apartado se compara este trabajo con otras soluciones existentes para la configuración de la autenticación y autorización en Kubernetes utilizando servidores externos.

En el trabajo de S. López [19], se implementa una arquitectura elástica basada en contenedores en la nube. En cuanto a la autorización de infraestructuras federadas, el autor crea una herramienta llamada *kube-authorizer* [20]. Con ella se automatiza la creación de un espacio de trabajo y de políticas de RBAC dentro del clúster para usuarios nuevos autenticados a través de OIDC. También, menciona que la herramienta de *kube-authorizer* puede utilizarse en conjunto con un *proxy* de OIDC. En su trabajo menciona *oidc-proxy* [21].

En este TFM, no se automatiza la creación de espacios de trabajo y de políticas de RBAC de la misma manera, se hace de forma manual. Pero, sí que se implementa el protocolo OIDC para autenticar usuarios. En este trabajo también se hace uso de un *proxy* que se encarga de la autenticación. En este caso se usa Oauth2 Proxy.

En el estudio de A. Téllez [22], se crea un clúster de Kubernetes sobre una serie de dispositivos de bajo coste llamados Raspberry Pi, a diferencia de este TFM en donde el clúster que se crea es de mayor potencia computacional y coste.

En cambio, ambos estudios presentan similitudes en cuanto a la configuración del sistema de autenticación y autorización. En este sentido los usuarios pueden acceder por medio de las credenciales de la universidad, y con una gestión de permisos que aseguren que los usuarios autenticados estén asignados a los grupos que correspondan. Estos grupos, a su vez, se corresponden con roles en Kubernetes que controlan los permisos de lo que podrán hacer en el clúster.

Por otro lado, la implementación de A.Téllez gestiona la conexión con la universidad a través de una aplicación OIDC de la UNED; mientras que en este trabajo se gestiona la conexión con la universidad mediante un proveedor de identidades (Keycloak) que además permite la gestión de usuarios.

7. CONCLUSIONES Y TRABAJO FUTURO

7.1. Conclusiones

El presente trabajo ha realizado las siguientes aportaciones:

1. Se ha propuesto una arquitectura para la autenticación y autorización de usuarios en Kubernetes y se han escogido herramientas estándar para su implementación (Keycloak, Oauth 2 Proxy y kubelogin*)*.

2. Se ha desplegado un servicio de gestión de identidades y autorizaciones (Keycloak) y se ha integrado la base de datos de usuarios de la UPV.

3. Se ha desplegado un sistema de autenticación para el cliente de Kubernetes (*kubectl*) que solicita las credenciales al usuario al hacer una consulta al Kubernetes API server.

4. Se ha desplegado un sistema de autenticación para controlar el acceso a cualquier microservicio desplegado en el clúster, utilizando las mismas credenciales y sin necesidad de volver a autenticarse.

5. Se han cumplido los requisitos indicados, se han validado los tres escenarios propuestos y se ha desplegado el servicio en producción.

6. Se ha descrito detalladamente la configuración e instalación del sistema para que otro desarrollador pueda implantarlo en su arquitectura.

7.2. Trabajo futuro

Debido a que el alcance del TFM es limitado en el tiempo y que se han identificado funcionalidades adicionales, como trabajo futuro se propone:

a) Mejorar la usabilidad utilizando *scripts* configurables que faciliten la puesta en marcha de algunas soluciones.

b) Realizar la puesta en práctica o despliegue de un sistema de *Log Out* de Keycloak para microservicios.

c) Implementar la instalación de Oauth2 Proxy de forma independiente a la aplicación de *Kubeapps.*

8. BIBLIOGRAFÍA

[1] «Documentación de Kubernetes | Kubernetes». https://kubernetes.io/es/docs/home/ (accedido 20 de junio de 2023).

[2] «AB/Connect Working Group - OpenID Foundation». https://openid.net/connect/ (accedido 20 de junio de 2023).

[3] «OAuth 2.0 — OAuth». https://oauth.net/2/ (accedido 20 de junio de 2023).

[4] «¿Qué es OAuth 2.0 y para qué sirve? - Auth0». https://auth0.com/es/intro-to-iam/what-is-oauth-2 (accedido 20 de junio de 2023).

[5] «int128/kubelogin: kubectl plugin for Kubernetes OpenID Connect authentication (kubectl oidc-login)». https://github.com/int128/kubelogin (accedido 20 de junio de 2023).

[6] «Documentation - Keycloak». https://www.keycloak.org/documentation (accedido 20 de junio de 2023).

[7] «Conceptos sobre Kubernetes». https://kubernetes.io/es/docs/concepts/ (accedido 13 de julio de 2023).

[8] «JWT». https://jwt.io/ (accedido 3 de julio de 2023).

[9] «Partes de un id_token». https://me.zhuoyue.me/2021/07/accesstoken-vs-id-token-vs-refresh-token-what-why-when/ (accedido 3 de julio de 2023).

[10] «Endpoints de Oauth 2 Proxy».

[11] «An Introduction to Oauth 2». https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2 (accedido 3 de julio de 2023).

[12] «LDAP: ¿Qué es y para qué se utiliza este protocolo?» https://www.profesionalreview.com/2019/01/05/ldap/ (accedido 7 de julio de 2023).

[13] «Conceptos fundamentales de seguridad de LDAP». https://docs.oracle.com/cd/E80557_01/VSMGU/ldap001.htm (accedido 7 de julio de 2023).

[14] «Alternativas de autenticación en Kubernetes», Accedido: 7 de julio de 2023. [En línea]. Disponible en: https://kubernetes.io/docs/reference/access-authn-authz/authentication/

[15] «Instancia de Keycloak del ai2».

[16] «Documentación Kubeapps». https://kubeapps.dev/ (accedido 5 de julio de 2023).

[17] «Configuración de Keycloak para Oauth 2 Proxy». https://kubeapps.dev/docs/latest/howto/oidc/oauth2oidc-keycloak/#keycloak-configuration (accedido 5 de julio de 2023).

[18] «Instalación de Krew». https://krew.sigs.k8s.io/docs/user-guide/setup/install/ (accedido 14 de julio de 2023).

[19] S. López Huguet, «Elastic, Interoperable and Container-based Cloud Infrastructures for High Performance Computing», Universitat Politècnica de València, Valencia (Spain), 2021. doi: 10.4995/Thesis/10251/172327.

[20] «Repositorio del código fuente de kube-authorizer». Accedido: 11 de julio de 2023. [En línea]. Disponible en: https://gitlab.com/primageproject/kube-authorizer

[21] «Repositorio del código fuente de oidc-proxy». Accedido: 11 de julio de 2023. [En línea]. Disponible en: https://gitlab.com/primageproject/oidc-proxy

[22] Álvaro. Téllez Rodríguez, «Reconfiguración dinámica de redes de laboratorios. Autenticación y monitorización con Kubernetes», 2020.

9. ANEXOS

Anexo 1. Creación de un usuario en Kubernetes y configuración de su acceso a cierto *Namespace*, mediante el uso de certificados.

Se plantea un caso para un usuario *employee* que pertenece al grupo *employees y* que solamente tendrá acceso, a nivel de *kubectl*, al *Namespace office* donde tendrá permisos de administrador.

Primeramente generamos la clave privada:

```
$ openssl genrsa -out employee.key 2048
```

Luego, creamos el *Certificate Sign Request* (*CSR*) apartir de la clave privada:

```
$ openssl req -new -key employee.key -out employee.csr -
subj "/CN=employee/O=employees"
```

Firmamos el CSR con los certificados del cluster:

```
$ openssl x509 -req -in employee.csr -CA
/etc/kubernetes/pki/ca.crt -CAkey
/etc/kubernetes/pki/ca.key -CAcreateserial -out
employee.crt -days 500
```

En este momento tendremos generados tres ficheros: *employee.key*, *employee.csr* y *employee.crt*.

A continuación, estableceremos este nuevo usuario en el fichero de configuración:

```
$ kubectl config set-credentials employee –client-
certificate=/home/.kube/employee.crt  --client-
key=/home/.kubeemployee.key
```

Por último, establecemos un contexto nuevo para el usuario recién creado y lo limitamos al *Namespace* que queramos que acceda, en este caso *office*:

```
$ kubectl config set-context employee-context --
cluster=cluster.local --namespace=office --user=employee
```

El fichero *config* tendrá que ser algo parecido a esto:

```
kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority-data: DATA+OMITTED
    server: https://172.16.162.50:6443
  name: cluster.local
contexts:
- context:
    cluster: cluster.local
    namespace: office
    user: employee
  name: employee-context
current-context: employee-context
kind: Config
preferences: {}
users:
- name: employee
  user:
    client-certificate: /home/.kube/employee.crt
    client-key: /home/.kube/employee.key
```

Una vez ya hemos configurado el clúster, ahora tendremos que establecer qué permisos queremos que tenga el usuario *employee* en el *Namespace* “office”. Para ello haremos uso de las políticas de RBAC que ofrece Kubernetes.

Para establecer permisos para un usuario específico utilizaremos un *RoleBinding* para enlazar el *User employee* con el Role que describe los permisos. Para ello se aplica un manifiesto con la siguiente estructura:

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  namespace: office
  name: employee-role
rules:
- apiGroups: ["*"]
  resources: ["*"]
  verbs: ["*"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1beta1
metadata:
  name: employee-binding
  namespace: office
subjects:
- kind: User
  name: employee
  apiGroup: ""
roleRef:
  kind: Role
  name: employee-role
  apiGroup: ""
```

Hasta aquí la implementación. Se ha creado un usuario nuevo, se ha configurado el usuario con Kubernetes y se le han establecido los permisos que nosotros hemos elegido para dicho usuario. Se puede probar la implementación con:

```
$ kubectl get pods
```

Este comando nos dejará listar el *Namespace office* (si hemos establecido el contexto por defecto a “employee-context”).

```
$ kubectl get pods ---all-namespaces
```

Este otro comando en cambio no nos dejará efectuarlo, ya que no tenemos permisos de listado para el resto de *Namespace*.

Anexo 2. *Deployment* Keycloak con MySQL.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: keycloak
  namespace: keycloak
spec:
  replicas: 1
  selector:
     matchLabels:
     app: keycloak
  template:
     metadata:
     labels:
     app: keycloak
     spec:
     containers:
     - name: keycloak
          image: quay.io/keycloak/keycloak:21.0.1
          args: ["start-dev”]
          env:
          - name: DB_VENDOR
          value: mysql
          - name: DB_ADDR
          value: mysql-service.keycloak
          - name: DB_DATABASE
          value: keycloak
          - name: DB_PORT
          value: "3306"
          - name: DB_USER
          value: root
          - name: DB_PASSWORD
          valueFrom:
               secretKeyRef:
               name: mysql-secret
               key: password
          - name: KEYCLOAK_ADMIN
          value: "admin"
          - name: KEYCLOAK_ADMIN_PASSWORD
          value: "admin"
          - name: KC_PROXY
          value: "edge"
          - name: PROXY_ADDRESS_FORWARDING
          value: "true"
          - name: KEYCLOAK HOSTNAME
          value: keycloak.192.168.49.2.nip.io
          - name: KEYCLOAK FRONTEND URL
          value: https://keycloak.192.168.49.2.nip.io/auth
          ports:
          - containerPort: 8080
```

Anexo 3. Service para exponer *Deployment* de Keycloak.

```
apiVersion: v1
kind: Service
metadata:
  name: keycloak
  labels:
     app: keycloak
spec:
  ports:
  - name: http
     port: 8080
     targetPort: 8080
  selector:
     app: keycloak
  type: NodePort
```

Anexo 4. *Ingress* NGINX para Keycloak.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: keycloak
  namespace: keycloak
spec:
  tls:
     - hosts:
     - KEYCLOAK_HOST
  rules:
  - host: KEYCLOAK_HOST
     http:
     paths:
     - path: /
     pathType: Prefix
     backend:
          service:
          name: keycloak
          port:
               number: 8080
```

Anexo 5. Contenido del fichero *values.yaml* para la instalación de *Kubeapps*.

```
authProxy:
  enabled: true
  skipKubeappsLoginPage: false
  provider: oidc
  clientID: kubeapps
  clientSecret: BbGYNLLrjhw2w396Q3UPCkPp0Va31huI
  ## python -c 'import os,base64; print
base64.urlsafe_b64encode(os.urandom(16))'
  cookieSecret: 3vwp-RTdGe8Jo9WvnLyL-
XCfQIPBTgFCGRRb6hypdtE=
  emailDomain: "*"
  extraFlags:
    - --ssl-insecure-skip-verify
    - --cookie-secure=false
    - --cookie-domain=.ai2.upv.es
    - --whitelist-domain=.ai2.upv.es
    - --set-authorization-header=true
    - --set-xauthrequest=true
    - --scope=openid email groups
    - --oidc-issuer-
url=https://keycloak.ai2.upv.es/realms/ai2realm
apprepository:
    initialReposProxy:
      enabled: true
      httpProxy: "http://proxy.upv.es:8080"
      httpsProxy: "http://proxy.upv.es:8080"
      #noProxy:
ingress:
  enabled: true
  ingressClassName: "nginx"
  hostname: kubeapps.ai2.upv.es
  tls: true
```

```
  extraTls:
    - hosts:
        - kubeapps.ai2.upv.es
      secretName: kubeapps-https-cert
  annotations:
    nginx.ingress.kubernetes.io/proxy-read-timeout: "600"
```

GLOSARIO DE ACRÓNIMOS

AAI: Autorization and authentication of identities (autorización y autenticación de identidades).

API: Aplication Programming Interface (Interfaz de programación de aplicaciones).

CN: Common Name (Nombre Común).

DC: Domain Component (Componente de Dominio).

DIT: Directory Information Tree (Árbol de Información de Directorio).

DN: Distinguised Name (Nombre Distintivo).

IdP: Identity Provider (Proveedor de Identidad).

JWT: JSON Web Token.

LDAP: Lightweight Directory Access Protocol (Protocolo Ligero de Acceso a Directorios).

OIDC: OpenId Connect.

OU: Organizational Unit (Unidad Organizativa).

RBAC: Role-based access control.

SSO: Single Sign On (Inicio de Sesión Único).

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**	X			
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades**.				X
ODS 11. **Ciudades y comunidades sostenibles**.				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Este Trabajo de Fin de Master propone e implementa un sistema homogéneo de autenticación y autorización en un clúster Kubernetes en el ámbito de la investigación, utilizando herramientas de código abierto como Oauth2 Proxy, kubelogin y Keycloak. Este estudio surge como respuesta a la problemática relacionada con la gestión de la seguridad en sistemas de contenedores y la necesidad de establecer un contexto que garantice la AAI en un clúster de Kubernetes.

Hoy en día todas las empresas que se dedican al sector TI dedican gran parte de su presupuesto en políticas de ciberseguridad que aseguren la prevención de posibles ataques cibernéticos. Es por esto por lo que es importante emplear herramientas y tecnologías seguras para consolidar estas prácticas. Este TFM se relaciona directamente con un nivel “Alto” con el ODS nº 8 Trabajo decente y crecimiento económico, debido al valor de implementar un sistema de autenticación y autorización de microservicios como primera barrera para prevenir ciberamenazas que puedan incurrir en costes muy elevados para las empresas.

Adicionalmente, se relaciona con un nivel “Alto” con el ODS nº 9 Industria, innovación e infraestructura. Debido al crecimiento de la informática y, más concretamente, de los sistemas de contenedores en la nube, es importante la creación de arquitecturas e infraestructuras que hagan frente a la demanda de estas tecnologías. En este estudio se propone una arquitectura para un clúster de Kubernetes con herramientas punteras de código abierto, que permitirán satisfacer los requisitos de resiliencia de un sistema informático en la nube.

El resto de los ODS no se relacionan en ninguna medida con este estudio.