Trabajo de Fin de Grado

Sistemas de Identidad Federados en Openstack

Federated Identity Systems in Openstack

José Joaquín Escobar Gómez

La Laguna, 5 de septiembre de 2017

D. **Juan Carlos Pérez Darias**, con N.I.F. 45.441.625-L profesor Titular de Universidad adscrito al Departamento de Física de la Universidad de La Laguna, como tutor

D. **Enol Fernández del Castillo**, con N.I.F. 54.047.212-H Doctor en Ingeniería Informática y especialista en Cloud Computing en la Fundación EGI(Holanda), como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Sistemas de Identidad Federados en Openstack.*"

ha sido realizada bajo su dirección por D. **José Joaquín Escobar Gómez**, con N.I.F. 79.196.800-A

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 5 de septiembre de 2017

Agradecimientos

A mi tutor Juan Carlos, por brindarme todo su apoyo y experiencia en el desarrollo de este trabajo y estar siempre atento a mis dudas y problemas.

A mi cotutor Enol por su gran paciencia en las explicaciones, la ayuda y dedicación prestada.

A mi familia, putativa y biológica, por todo su apoyo en las decisiones que he tomado y amarme incondicionalmente.

A mis amigos de grado por estar en las buenas y en las malas, por todas las risas y los cafés compartidos.

A mi equipo de natación, Bentacu Laguna, por despejarme la mente en el agua y hacerme siempre sonreir en los entrenamientos.

Licencia

Resumen

El objetivo de este Trabajo de Fin de Grado ha sido el desarrollo de una librería que permita la gestión desatendida de una infraestructura en la nube basada en OpenStack usando autenticación con OpenID Connect. La librería de gestión de identidad Keystone de OpenStack posee distintos métodos de autenticación, de los cuales se partió como base para la creación de un nuevo plug-in, siguiendo la misma estructura y pautas.

Todo el desarrollo del proyecto se ha realizado en sistemas operativos Linux, utilizando como servidor Apache e instalando OpenStack como infraestructura de cloud computing.

El plug-in creado se ha realizado en el lenguaje de programación Python, siguiendo la misma estructura de los plug-ins existentes en Keystone, se realizaron los test unitarios necesarios para comprobar el correcto funcionamiento de éste.

Se instaló y configuró en el servidor la infraestructura de computación en la nube OpenStack, a la cual se le hicieron las peticiones desde el cliente. Las peticiones realizadas con el cliente de OpenStack fueron realizadas con el plug-in desarrollado, utilizando como parámetros principales las credenciales otorgadas por el proveedor de identidad.

En el servidor se instaló y configuró un módulo Apache que permite la comunicación con el proveedor de identidad, facilitando el uso del protocolo OpenID Connect, para realizar las peticiones pertinentes al proveedor de identidad.

Los flujos de comunicación entre el servidor y el proveedor de identidad se realizan utilizando las operaciones básicas de los sistemas REST (Representational State Transfer- Transferencia de Estado Representacional), ya que es el estándar más utilizado y eficiente en los desarrollos de APIs para servicios ofrecidos en Internet.

Palabras clave: Sistema Federado, Identidad Federada,Identidad Digital, OpenStack, OpenID Connect, OAuth2.0, Keystone.

Abstract

The main objective of the this Final Degree Project has been the development of a library that allows the unattended management of a cloud-based infrastructure based on OpenStack using OpenID Connect Authentication. The identity management library Keystone of OpenStack has different methods of authentication which were used as the basis for the develop of a new plug-in, following the same structure and guidelines.

The entire development of the project has been done in Linux operating systems, using Apache server and using OpenStack as a cloud computing infrastructure.

The plug-in developed has been made in Python programming language, following the same structure of the existing Keystone plug-ins, and were made the necessary unit tests to verify its functioning.

The cloud computing infrastructure based on Openstack was installed and configured, to which the request were made from the client. The requests made with the OpenStack's client were made with the developed plug-in using as parameters the credentials granted by the identity provider.

In the server was installed and configured an Apache module that allows communication with the identity provider, facilitating the use of OpenID Connect's protocol to make the relevant requests to the identity provider.

Communications flows between the server and the identity provider were performed using REST-like (Representational State Transfer) operations which is the most used and efficient standard in the development of Internet services.

Keywords: *Federated Identity,Digital Identity OpenStack, OpenID Connect, OAuth2.0, Keystone.*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Antecedentes y estado actual

Internet es hoy el eje central del negocio y relaciones personales, y cada día más y más organizaciones ofrecen a los usuarios servicios desplegados en la nube, esto requiere que los usuarios poseean para cada proveedor de servicio, *service provider* (SP), una identidad vinculada a las credenciales y a la información personal, ya sean con usuario y contraseña, claves de acceso, u otro mecanismo de autenticación.

Para muchos de los servicios ofrecidos, los usuarios por lo general poseen una cuenta de usuario y su correspondiente contraseña, esto requiere que los usuarios recuerden cada una de estas cuentas, fomentado que la seguridad de las contraseñas sea debil, ya que los usuarios establecen contraseñas básicas y fáciles de recordar, favoreciendo potenciales brechas, como comprometer la seguridad de los datos del usuario o robo de identidad.

Por lo general cada proveedor de servicios almacena y gestiona las identidades digitales, permitiendo tener políticas de seguridad propias, para mejorar la experiencia al usuario final y para aumentar los beneficios propios.

Anteriormente a la aparición de los sistemas de identidad, predominaba la creencia de que la gestión online de las identidades eran problematicas porque cada proveedor de servicio gestionaba un conjunto de identidades, por lo general una identidad para cada servicio ofrecido, creando potenciales puntos de fallo, y que también los usuarios tenían la sensacion de no tener el control sobre sus datos personales, y la poca confianza que se le otorgaba a a los proveedores, acarreando posible robos de identidad ó violaciones de privacidad [9].

Los sistemas de gestión de identidad se desarrollaron con el objetivo de evitar los problemas antes mencionados, así como ofrecer al usuario la confianza

sobre estos sistemas. La seguridad en los sistemas de gestión de identidad se incrementó con la introducción de un nuevo elemento, el proveedor de identidad,*identity provider* (IdP), en el cual recae toda la confianza.

La gestión de la identidad, así como las acciones de autenticación de los usuarios, son controlados por el proveedor de identidad, el cual posee mecanismos de gestión de los datos personales del usuario, como lo es la regulación de qué información puede ser compartida y cual no, y también posee mecanismos que involucran las confirmaciones de autenticación, *authentication assertions*, sobre los datos de los usuarios. Al recibir alguna petición de autenticación de parte de algún programa en nombre de un usuario, el proveedor de servicio es el que toma la decisión sobre si autorizar al usuario usando los datos que acompañan a las confirmaciones de autenticación.

El rápido crecimiento del comercio digital, ha fomentado la necesidad de evolución de los sistemas de gestión de identidad, en los sistemas de gestión de identidad federados, que tienen como objetivo principal mejorar la usabilidad del usuario final, mejorar la privacidad, y distribuir la gestión de los usuarios mediante la federación de identidades entre entidades de mutua confianza [18].

En los sistemas de gestión de identidad federados, es necesario tener una base tecnológica que permita la interoperabilidad y que la proporción entre coste y eficiencia sea lo más nivelado posible, es por esto que los servicios web son un buen candidato para suplir estas necesidades, ya que posee la capacidad de comunicar y componer distintas aplicaciones comerciales sobre redes distribuidas y heterogéneas [36].

Cloud Computing se refiere a el conjunto de aplicaciones, plataformas e infraestructura que ofrece servicios a través de una red, que por lo general es Internet, siendo el IaaS (*Infrastructure as a Service*) el modelo más básico de servicio en la nube. Principalmente estas soluciones ofrecen un gran catálogo de recursos informáticos virtualizados, reduciendo los costes de inversión, mantenimiento y gestión. El objetivo principal de estas soluciones es la de ofrecer una gestión de los recursos de forma eficiente y eficaz así como de escalabilidad y adaptabilidad.

Las tecnologías de código libre para el desarrollo de servicios en la nube, como OpenNebula, Eucalyptus, OpenShift, Cloud Foundry y OpenStack, cada vez van ganando más importancia, ya que permiten la oportunidad de investigar, desarrollar,implementar, innovar y desplegar nuevos servicios.

OpenStack es una arquitectura para *Cloud Computing open source* eficiente, escalable y adaptativo para soluciones IaaS (*Infrastructure as a Service*) tanto

públicas como privadas, así como también enfocada en aprovechar al máximo los recursos físicos, en un entorno profesional altamente competitivo o en entornos de menor envergadura [33].

1.2. Objetivo

A continuación se expondrán los objetivos principales del proyecto, que han sido la guía de su desarollo.

1.2.1. Objetivo general

El objetivo general del proyecto es diseñar e implementar un plug-in que permita la gestión desatendida de una infraestructura en la nube basada en OpenStack mediante el uso de la autenticación por medio de OpenID Connect.

1.2.2. Objetivos específicos

Para conseguir este objetivo se plantean una serie de objetivos específicos:

- Realizar un análisis de la librería Keystone de OpenStack y sobre los métodos de autenticación que utiliza, así como el funcionamiento de OpenID Connect.
- Desarrollar y desplegar un plug-in para Keystone, que utilice el Refresh Token, y las credenciales del usuario para obtener el Access Token, y con éste último realizar la introspección de la información del usuario al proveedor de identidad.
- Instalar y configurar el servidor OpenStack en el IAAS del Centro de Cálculo.
- Instalar y configurar el modulo OIDC para Apache.
- Comprobar el correcto funcionamiento del plug-in mediante pruebas en un entorno local y en un entorno real externo.
- Redacción de la documentación de todo el desarrollo del proyecto.

1.3. Plan de trabajo

Para la consecución de los objetivos se ha planificado el siguiente programa de actividades:

Tarea 0. Planificación.

La primera tarea consiste en la planificación del TFG, estableciendo reuniones periódicas con el tutor y cotutor. Se intenta coordinar las reuniones con el avance de las demás tareas propuestas, es decir, que a medida que se van realizando las tareas se realizarán reuniones para exponer los resultados obtenidos, así como también aclarar dudas que se pudiesen encontrar.

Tarea 1. Formación.

Para empezar con el TFG, debido al amplio abanico de tecnologías involucradas ,es imprescindible adquirir ciertos conocimientos básicos necesarios para el proyecto. Esta tarea consiste en una investigación teórica de los procesos, protocolos, herramientas y tecnologías a utilizar durante el desarrollo del mismo.

Tarea 2. Pruebas básicas.

Una vez se han adquirido los conocimientos básicos necesarios, se procede a realizar un primer contacto con estas herramientas y tecnologías, realizando pruebas y analizando las respuestas obtenidas. Las pruebas se realizarán utilizando la librería oauth2.py y el OAuth Playground de Google.

Tarea 3. Ingeniería Inversa.

Ya familiarizado con las herramientas y tecnologías, se continúa con analizar la librería Keystone de Openstack, para comprender el funcionamiento de los plug-ins de autenticación implementados, comprender cómo funcionan, cómo son llamados, los parámetros requeridos, etc, para posteriormente implementar un plug-in nuevo.

Tarea 4. Ampliación de la librería.

Se busca que la librería permita utilizar el Refresh Token para la obtención del Access Token, para lo cual se diseñará e implementará un plug-in que lo

permita, éste será probado en un entorno local, donde se simulará la respuesta del servidor a la llamada del plug-in. El plug-in se desplegará en Pypi.

Tarea 5. Implementación servidor.

La implementación del servidor OpenStack se llevará acabo en la plataforma IaaS de los servicios TIC de la Universidad de La Laguna. Instalado el servidor OpenStack, se procederá a la instalación de los módulos apache necesarios que permitirán gestionar las soluciones OpenID Connect, se configurarán estos módulos, así como la configuración de la federación en OpenStack. Para la configuración de los módulos de apache obtendrán las credenciales necesarioas, registrando el servidor OpenStack en el proveedor de identidad, que en el desarrollo del proyecto será Google.

Tarea 6. Peticiones al servidor.

Con la configuración del servidor, se procede a realizar peticiones desde el cliente hacia el servidor implementado. Con estas peticiones se comprobará el correcto funcionamiento del servidor, se comprobarán los resultados obtenidos y se corregirán los errores que se presenten.

Tarea 7. Documentación.

La fase de documentación consiste en la recopilación de los resultados obtenidos, al igual de exponer todo el proceso de desarrollo del proyecto. Para esta tarea se utilizarán las herramientas LaTex y BibTeX.

1.4. Estructura de la memoría

La documentación se encuentra dividida en cinco capítulos principales, ordenados de la siguiente manera:

- En el capítulo II se describen las herramientas y tecnologías utilizadas en el proyecto.

- En el capítulo III se explican los diversos pasos realizados para la consecución del proyecto.
- En los capítulos IV y V sobre las concluciones del proyecto y las futuras líneas de desarrollo.
- En el capítulo VI se muestra el presupuesto estimado, con precios de mercado actuales.

Capítulo 2

Herramientas y tecnologías

En el capítulo anterior se ha descrito el objetivo general del proyecto, sobre realizar una ampliación de la librería de autenticación de openstack para permitir la gestión desatendida, para llevar a cabo los objetivos planteados, se utilizaron las siguientes herramientas y tecnologías.

2.1. Identidad Federada

Una de las grandes ventajas que proporcina la gestión de la identidad federada es la capacidad de reconocer la seguridad y aprovechar las identidades que los usuarios poseen entre entidades de confianza y sobre todo la capacidad de compartir las identidades de forma segura entre el círculo de confianza de las entidades, evitando que los usuarios deban proporcionar sus credenciales que lo identifican, como su cuenta y contraseña cuando requieran acceder a los recursos ofrecidos por algún proveedor de servicio.

Además la federación de identidad proporciona la ventaja de que los usuarios puedan gestionar su información personal, seleccionando la información que se quiere compartir, ya sea su perfil u otros datos que le pertenezcan, entre las entidades de confianza. El principal objetivo es que el usuario deberá proporcionar la debida autorización ó consentimiento a las entidades, proveedores de servicio u organizaciones que requieran del acceso a su identidad.

En el proyecto se utiliza una infraestructura en la nube, OpenStack, ver la sección 2.3, la cual tiene como módulo de identidad principal llamado Keystone,ver la sección 2.3.1, que fue desarrollado buscando dar soporte a la identidad federada, dándole a los administradores la capacidad de aprovechar la solución existente de identidad federada, así como también brindar la posibilidad de que los usuarios utilicen sus credenciales y aprovechar la capacidad de inicio se sesión único, *single sign on* (SSO). La idea principal es la de utilizar un proveedor de identidad ajeno a la organización, para poder centrarse fundamentalmente

en la autorización y prestar un mejor servicio, gracias a que toda la gestión y mantenimiento de las credenciales de los usuarios es realizada po el IdP.

La federación de los recursos está ganando gran valor como mecanismo enfocado a la interoperabilidad de recursos construidos con distintas tecnologías a través de infraestructuras independientes. Al federar los recursos se consigue un mecanismo interoperable que trabaja en un entorno que tiende a seguir un único modelo de colaboración.

Analicemos algunos conceptos básicos.

2.1.1. Autenticación

La autenticación es el proceso en el cual un sistema verifica que una entidad es quien dice ser, presentando información que el usuario tiene conocimiento , por lo general es necesario presentar una cuenta y contraseña, otros métodos pueden ser presentando algo que el usuario tiene, como hardware, tarjetas con chips integrados, ó presentando alguna lectura biométrica del usuario, como el iris o huella dactilar [11].

En la sección 2.3.1, se presentan los métodos de autenticación de la librería Keystone de OpenStack.

2.1.2. Autorización

La autorización se refiere a los procesos de concesión de privilegios a una entidad. Una vez se verifica que una entidad es quien dice ser(Autenticación), los derechos, los privilegios y todas las acciones permisibles se determinan mediante la autorización, es decir, la autorización es la especificación de las políticas de acceso que se traducen en reglas que el sistema puede usar para decidir si aprobar o denegar peticiones de acceso [2].

2.1.3. Identidad digital

Con la globalización del comercio y el avance de los servicios en Internet, fue necesario la representación de los usuarios del mundo real en el mundo digital. La identidad digital usualmente está comprendida en una cuenta personal del usuario que contiene un conjunto limitado de atributos que caracterizan a dicha persona [31].

2.1.4. Identidad basada en Claims

Uno de los objetivos principales de la identidad basada en claims, es la de mejorar la experiencia digitial, así como la de asignar los recursos digitales basados en claims realizados por una entidad a otra. Una entidad puede ser una organización, una persona física, el gobierno, una página web, un servicio web ó un dispositivo. Se puede decir que un claim es simplemente algo que una entidad dice sobre sí misma o sobre otra entidad. Este claim puede ser, por ejemplo, sobre información básica de la entidad, los grupos a los que pertenece, o sobre los privilegios que ésta posee [37].

La identidad basada en claims es el método de representación de la información y atributos sobre una entidad, donde un proveedor de identidad puede emitir claims dependiendo de los estándares que utilice, para que sean usados por los proveedores de servicio. Los claims son solicitados por un proveedor de servicio y que unos valores empaquetados en tokens de seguridad, que son emitidos por un servicio de tokens de seguridad,*Secutiry token service* (STS), STS es el encargado de contruir, firmar y emitir tokens de seguridad [5].

Las aplicaciones y servicios basados en claims, donde los claims son usados como una representación sobre una entidad para realizar una petición de acceso, son llamados *Relying party*(RP) *applications*, es decir una aplicación que es dependiente de los claims. Un RP puede ser llamado de igual forma como, *"claims aware application"* o *"claims-based application"*.Una aplicación RP utiliza los tokens emitidos por el *Secutiry token service* (STS) y extrae los claims de estos para usarlos en tareas relacionadas con la identidad federada [6].

2.2. Protocolos

En esta sección se describirán los protocolos más relevantes empleados en el proyecto.

2.2.1. OAuth 2.0

El *OAuth 2.0 authorization framework* [17] lo define como el protocolo basado en web que otorga a los usuarios el permiso de acceso a sus recursos, datos o servicios alojados en otro RP. OAuth 2.0 es usado frecuentemente también para autenticar al usuario mediante el gestor de identidad del IdP, como por ejemplo *single sign on* (SSO).

La autorización y el *single sign on* (SSO), se han implementado como una solución a los servicios web a través de los años, siendo OAuth 2.0 uno de los frameworks más populares. Actualmente OAuth 2.0 es utilizado en distin-

tos proveedores de identidad (IdP) como Facebook, Google, Yahoo, Microsoft, Amazon, Github, LinkedIn y Dropbox. Con la implementación de este protocolo en organizaciones tan importantes en la era digital, se ha permitido que gran cantidad de usuarios accedan a un grandísimo catálogo de RPs o de compartir sus datos con éstos, convirtiendo a OAuth uno de los protocolos más usados en *single sign on* (SSO) en la web.

Para que un RP pueda interactuar con un IdP, es necesario registar de antemano el RP en el IdP, esta operación es realizada, por lo general, de forma manual. Durante el registro del RP, el IdP asigna y proporcina dos credenciales al RP, una pública y opcionalmente una privada, *OAuth client id* y *OAuth client secret*, respectivamente. Hay que decir que en el estándar OAuth, el término *client* representa al RP. En caso de que el RP deba autenticarse, deberá proporcionar el *client secret* al IdP. También en el momento del registro, es necesario presentar al IdP, una o más direcciones finales de retorno *redirection endpoint* URIs pertenecientes al RP.

Las interacciones entre el usuario y su consola o navegador web, el RP y el IdP pueden realizarse en distintos flujos o *grant types*, a continuación se describen los cuatro flujos [17]:

- *Authorization code grant*, cuando el usuario intenta autorizar un acceso a sus datos en un IdP por parte de un RP o si desea acceder a un RP, el RP primero redirecciona al usuario hacia el IdP, donde el usuario deberá autenticarse usando las credenciales del IdP, donde posteriormente es redirigido al RP acompañado de un código de autorización emitido por el IdP. Ahora es cuando el RP puede comunicarse con el IdP con el código de autorización, y con las credenciales (*OAuth client id* y *OAuth client secret*) y recibir un token de acceso *Access Token*, el cual el RP puede utilizar como credencial de acceso a los datos protegidos del usuario en el IdP.

- *Implicit grant*, este flujo es muy similar al anterior, pero en vez de proporcionar un código de autorización, el IdP entrega directamente el *access token* al RP, usando el navegador web del usuario, es decir que en vez de crear un código de autorización, el IdP emite el *access token* de inmediato y realiza una redirección al *redirection endpoint* del RP con el *access token* dentro de la URI. Para obtener el *access token*, es necesario utilizar un código JavaScript, el cual es enviado en la respuesta al *redirection endpoint* del RP. Al obtenerlo, el RP ya puede hacer uso del *access token* para ser autenticado o autorizado.

- *Resource owner password*, en este flujo, el usuario proporciona sus credenciales del IdP directamente a un RP, el RP se autentica en el IdP en

nombre del usuario y obtener el *access token*. Este método está pensado para ser implementado cuando existe un alto grado de confianza hacia los RPs, también por aplicaciones con privilegios especiales o cuando los dos flujos anteriores no son posibles de implementarse.

- *client credentials grant*, a diferencia de los tres flujos anteriores, este método funciona sin la interacción del usuario, en cambio es iniciado por un RP con el fin de obtener el *access token* y poder acceder a los recursos del RP en un IdP.

OAuth 2.0 ofrece a los usuarios la posibilidad de acceder a recursos protegidos a través de la obtención de un *access token*, el cual es un string que representa una autorización de acceso emitida al cliente [17], en vez de utilizar las credenciales del usuario directamente. El *access token* representa el alcance que tiene el permiso otorgado, la duración y otros atributos concedidos por la autorización concedida.

El *access token* provee de una abstracción que sustituye otras estructuras de autorización, como el usuario y contraseña, por un token único que es legible por el servidor de recursos, *resource server*. Lo cual permite la emisión de *access tokens* válidos por periodo de tiempo limitado.

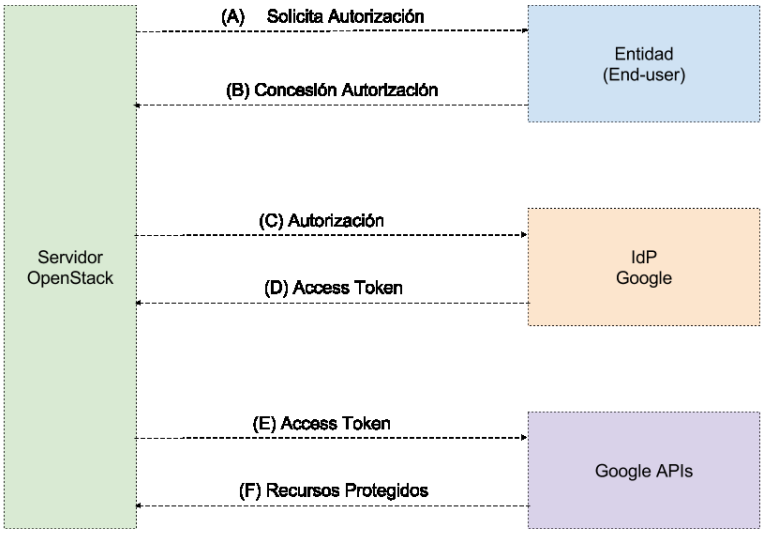

Figura 2.1: Flujo abstracto del protocolo OAuth 2.0

El protocolo OAuth 2.0 define cuatro roles principales:

- Propietario de los recursos, es la entidad(end-user) capaz de conceder el acceso a los recursos protegidos.
- Servidor de recursos, es el servidor de alojamiento de los recursos protegidos de la entidad.
- Cliente, es la aplicación,proceso o servicio que realiza peticiones en nombre de una entidad con su autorización, para acceder a sus recursos protegidos.
- Servidor de autorización, es el servidor emisor de los *access tokens* al cliente despues de realizar la autenticación de la entidad y obtener su autorización de acceso.

El flujo abstracto de OAuth 2.0 mostrado en la Figura 2.1, muestra de forma general las interacciones entre el cliente(OpenStack), el servidor de recursos(Google API), el servidor de autorización(IdP Google) y el propietario de los recursos(Entidad end-user) [19]. (A)El cliente solicita autorización a la entidad(end-user),(B) la entidad le da la autorización al cliente para realizar acciones a su nombre, (C) con la autorización de la entidad el cliente puede ir al IdP y solicitar un *access tokens*, (D) el IdP comprueba la autorización de la entidad y proporciona en caso asertivo el *access tokens*, (E) con el *access tokens* el cliente puede realizar peticiones en nombre de la entidad(end-user) a recursos protegidos de ésta y (F) el servidor que posee los recursos protegidos le responde con la información solicitada.

El *access token* es posible obtenerlo de igual forma usando un *refresh token*. Los *Refresh Tokens* son credenciales emitidas al cliente por el servidor de autorización y son usados para la obtención de un nuevo *access token*, ya que este último, como ya se ha mencionado anteriormente, expira con el tiempo, o también cuando ha dejado de ser válido [19]. Emitir un *refresh token* es una credencial ofrecida por el servidor de autorización de forma opcional, y en caso de ser emitido, se presentará en la misma solicitud de *access token*, en el paso (D) de la Figura 2.1. Hay que resaltar que a diferencia del *access token*, el *refresh token* está pensado para ser usado sólo con los servidores de autorización, como se puede observar en la Figura 2.2 .

Existen dos escenarios posibles cuando el cliente solicita acceso a información de la entidad(end-user) en su nombre respecto al *access token*, el primero es que el *access token* se encuentre válido al realizar las peticiones y el segundo que el *access token* sea inválido.

Como se puede observar en la Figura2.2, (A) el cliente autorizado para realizar peticiones en nombre de la entidad solicita un *access token*, (B) el IdP

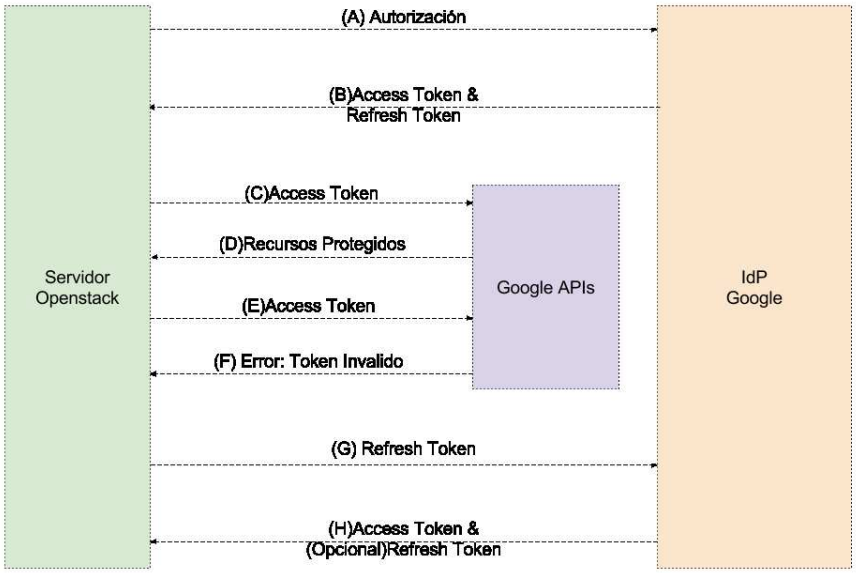

Figura 2.2: Obtención del Access Token con el Refresh Token

le responde con el *access token* y con un *refresh token*. (C) El cliente realiza peticiones al servidor de recursos con el *access token* solicitando información de la entidad(end-user), (D) se comprueba el *access token* y si es válido se responde a la solicitud. (E) El cliente realiza tantas veces los pasos (C) y (D) requiera, mientras el *access token* sea válido, en caso contrario, (F) se rechazan las peticiones del cliente y se debe de utilizar el *refresh token* para refrescar el *access token*, (G) el cliente envía el *refresh token* al IdP y si es correcto, el IdP le responde al cliente con un nuevo *access token* y opcionalmente acompañado de un *refresh token*.

2.2.2. OpenID Connect

El principal pilar del protocolo de autenticación *OpenID Connect* es OAuth 2.0, que es usado actualmente y soportado por compañias como Google, Microsoft, PayPal, GitHub, entre otros. *OpenID Connect* es actualmente soportado por grandes organizaciones, lo cual le otorga un futuro comprometedor en años próximos. *OpenID Connect* se basa en OAuth 2.0, y proporciona una interfaz *user-friendly*, fácil de usar y eficiente para realizar la autenticación de los usuarios y adicionalmente ofrece características opcionales como el descubrimiento dinámico de un IdP y el registro de una RP, así como tareas de firmado y cifrado de mensajes [7].

OpenID Connect se puede definir como una capa de identificación, autenticación sobre el protocolo OAuth 2.0. El cual permite a los clientes verificar

la identidad de una entidad basado en la autenticación realizada por un servidor de autenticación, así mismo permite la obtención de información protegida de una entidad, realizando operaciones *REST-like* y de forma que permita la interoperabilidad entre sistemas [8].

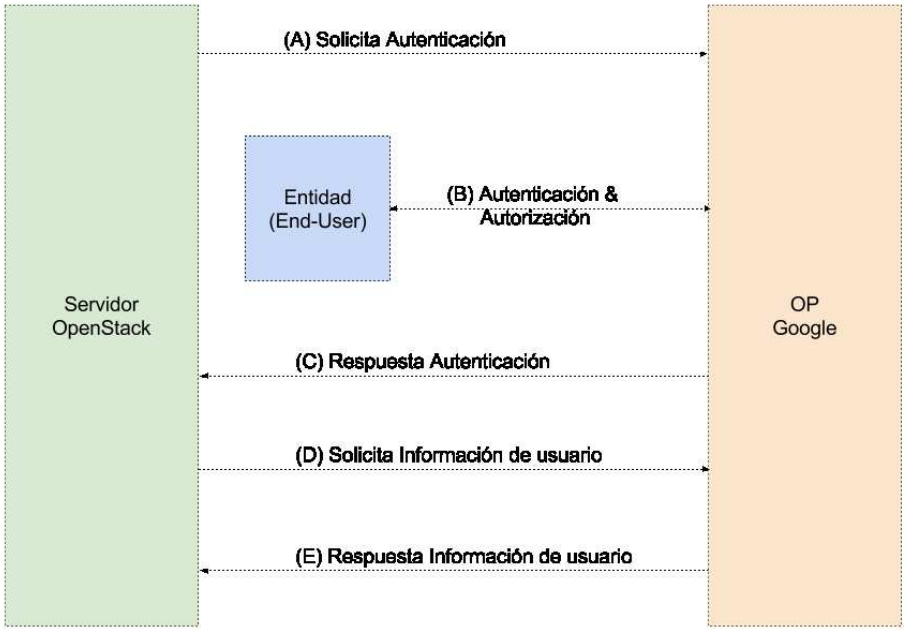

Figura 2.3: Abstracción flujo del protocolo OpenID Connect

Como se puede apreciar en la Figura 2.3, OpenID Connect, realiza diversos pasos para conseguir los claims de una entidad(*End-User*). (A) El cliente o RP, realiza una petición de autenticación al proveedor OpenID, *OpenID Provider* (OP), luego (B) el OP autentica a la entidad(*End-User*) y obtiene de él la autorización. (C) El OP responde con un *ID Token* y por lo general va acompañado de un *access token*. (D) El RP o cliente envía una solicitud con el *access token* al *UserInfo Endpoint*. (E) El *UserInfo Endpoint* responde con los claims, a los que tiene autorizado acceder, de la entidad(*End-User*) [8].

La estructura de dato principal en el protocolo OpenID Connect es el *ID Token*, el cual es un token de seguridad que contiene los claims de una entidad, y está representado como un *JSON Web Token* (JWT). Un JWT es un formato de representación compacta de los claims, diseñado para ser usado en entornos limitados como lo son las cabeceras de autorización HTTP y como parámetro de consulta URI. Los claims se codifican para ser transmitidos como un objeto JSON el cual es usado como carga útil de una estructura *JSON Web Signature*(JWS) [15] o una estructura *JSON Web Encryption* (JWE) [14], permitiendo

que los claims puedan ser firmados digitalmente y/o cifrados [16].

El *ID Token* debe ir siempre firmado usando un JWS [15], y opcionalmente, ser cifrado usando un JWE [15], proporcionando autenticación, integridad, no repudio y opcionalmente confidencialidad, si se cifra y para ser cifrado primero debe de estar firmado usando JWS.

Figura 2.4: Ejemplo JWT codificado y decodificado

JWT está dividido por tres secciones, separados por ".", que son: cabecera, carga útil y firma, ver Figura 2.4. La cabecera contiene un JSON que define el tipo de token y cómo se debe tratar. La sección de la carga útil incluye los claims y los datos, ofrecidos por el emisor del token, sobre el usuario y/o aplicación. La sección de firma es usada para la verificar la autenticidad del emisor del JWT y asegurarse que no se han modificado los datos en el recorrido. Para más detalles sobre estas tres secciones, así como funcionan, se puede ver en las especificaciones de JWT [16].

A continuación se muestran los claims usados dentro del *ID Token* en los

flujos del protocolo OAuth 2.0 usados por OpenID Connect [16]:

- iss (*Issuer Identifier for the Issuer of the response*), es un case sensitive URL, que emplea el esquema HTTPS y contiene el host, esquema y opcionalmente un número de puerto y la ruta de componentes.
- sub (*Subject Identifier*),identificador único del emisor para una entidad.
- aud (*Audience(s)*), a quién va dirigido el *ID Token*, deberá contener la credencial OAuth 2.0 cliend_id del RP.
- exp (*Expiration time*), fecha y hora límite en el cual el *ID Token* es válido.
- iat (*Issued At Time*), fecha y hora en el cual el *ID Token* fue emitido.
- auth_time, fecha y hora en la cual la entidad(*End-User*) se autenticó.
- nonse, atributo usado para asociar la sesión del cliente con un *ID Token.*
- acr (*Authentication Context Class Reference*), especifica el contexto de autenticación que el acr realizó satisfactoriamente.
- amr (*Authentication Methods References*), contiene los identificadores para los métodos de autenticación usados en la autenticación.
- azp (*Authorized party*), la parte a la cual el *ID Token* le fue emitido.

El *ID Token* puede contener otros claims, como el nombre, apellido, nombre completo, perfil, fotografía, email, genero, etc.

OpenID Connect realiza la autenticación para dar accesso a un *End-User* o para comprobar que ya se encuentra dentro del sistema, retornando el resultado de la autenticación realizada por el servidor al cliente de forma segura, para que el cliente confie en él, es por esta razón que el cliente se llama la parte de confianza,*Relying Party*(RP). El resultado, como ya se ha comentado, es devuelto en un *ID Token.* La autenticación puede realizarse siguiendo uno de los tres flujos que se mencionan a continuación [8]:

- *Authorization Code Flow*, retorna un código de autorización al cliente, el cual puede intercambiarlo directamente por un *ID Token* y un *Access Token.* Este método proporciona la ventaja de evitar exponer algún token a un tercero. El servidor de autorización también puede autenticar al cliente antes de realizar el intercambio del código por el *Access Token.* Es importante decir que este método es apropiado de ser implementado en clientes donde se puede mantener un *Client Secret*, entre las partes implicadas. Los tokens son devueltos desde el *Token Endpoint.* El *response_type* al momento de realizar la solicitud de autenticación tiene que ser "*code*".

- *Implicit Flow*, es usado por lo general por clientes implementados en un navegador web que utiliza un lenguaje de scripting,el *ID Token* y el *Access Token* son devueltos al cliente directamente. A diferencia del *Authorization Code Flow*, no se realiza ninguna operación de autenticación del cliente por parte del servidor de autorización. Los tokens son devueltos desde el *Authorization Endpoint*. El *response_type* al momento de realizar la solicitud de autenticación tiene que estar con alguno de estos valores "*id_token*"ó "*id_token token*".

- *Hybrid Flow*, cuando es usado este método, algunos tokens pueden ser devueltos desde el *Token Endpoint* y otras veces devueltos por el *Authorization Endpoint*. El *response_type* al momento de realizar la solicitud de autenticación tiene que estar con alguno de estos valores "*code id_token*", "*code token*"ó "*code id_token token*".

En la documentación de OpenId Connect [8] se puede ver todo el funcionamiento de los métodos de autenticación con más detalle, así como todos los que no se mencionan en este documento.

Para finalizar la explicación del protocolo, es necesario explicar el como 'Refrescar' un *Access Token* mediante el uso del *Refresh Token*. Para refrescar un *Access Token*, el cliente deberá autenticarse en el *Token Endpoint* usando algún método de autenticación registrado para su *cliend_id*. La petición es enviada mediante una llamada HTTP POST al *Token Endpoint*. El servidor de autenticación deberá de validar el *Refresh Token*, verificando que fue emitido para el cliente que lo solicita y deberá comprobar que el cliente se ha autenticado, se puede ver en la Figura 2.2.

2.3. OpenStack

OpenStack está compuesto por diversos módulos, y ya que es de código abierto, cualquier persona puede contribuir a su evolución, añadiendo componentes adicionales para mejorar y suplir sus necesidades. La implementación de OpenStack en un entorno competitivo, proporciona un entorno de desarrollo ágil, escalable y eficiente, ofreciendo los distintos servicios básicos de una infraestructura en la nube, como lo son los servicios de almacenamiento, computación, comunicaciones y automatización de servicios. Cada uno de estos servicios puede escalar individualmente y con gran adaptabilidad por estar constituido por estandares y APIs de código abierto [32].

Proporciona servicios de almacenamiento y computación similares a los de Amazon EBS, Amazon Elastic Compute Cloud y Amazon S3. OpenStack esta

completamente desarrollado en Python, publicado bajo la licencia Apache 2.0, soporta máquinas virtuales basadas en núcleo(KVM), monitores de máquinas virtuales VMware y Xen.

Los componentes en el desarrollo de OpenStack son varios, pero se pueden identificar nueve componentes claves, que hacen parte del núcleo de OpenStack [26]:

- Nova, es el motor principal de computación detras de OpenStack. Es el que permite desplegar y gestionar un gran número de máquinas virtuales y otras instancias relacionadas con tareas de computación.
- Swift, es el sistema de almacenamiento de ficheros y objetos. Construido para la escalabilidad y almacenamiento de gran cantidad de archivos, donde los desarrolladores delegan la tarea de gestionar en donde se situan los objetos y datos a OpenStack,y además decide cual es la mejor manera de respaldar los objetos en caso de algún fallo de una máquina o por pérdida de conexión. Los ficheros y objetos almacenados poseen un identificador que los relaciona, permitiendo gestionarlos con éste.
- Horizon, es la implementación gráfica de OpenStack. Proporciona de una interfaz a los servicios de OpenStack, donde los usuarios pueden acceder a todos los componentes individualmente, a través de una interfaz gráfica que permite a los usuarios realizar la administración y gestión de los recursos.
- Neutron, es un SDN (*Software Defined Networking*) enfocado en prestar *networking-as-a-service* (NaaS) en entornos virtuales de computación. Proporciona la capacidad de *networking* en OpenStack, desarrollado para la comunicación rápida y eficiente entre los componentes.
- Glance, es el servicio de plantillas de imágenes para las máquinas virtuales. Permite usar las imágenes para la creación de nuevas instancias de máquinas virtuales.
- Cinder, es el componente de almacenamiento de bloques, donde para acceder a los datos se accede a una ubicación específica a una unidad de disco. Es un escenario más tradicional, que se usa en situaciones en las que se requiere un rápido acceso a los datos.
- Ceilometer, permite la recolección eficiente de datos producidos por los servicios de OpenStack, para normalizarlos y transformarlos. Provee de servicios de telemetría, permitiendo contabilizar el uso que los usuarios realizan a cada servicio utilizado.

- Heat, es el componente que articula OpenStack, ayuda a gestionar la infraestructura necesaria para que un servicio en la nube funcione. Permite a los desarrolladores almacenar en un fichero todos los requerimientos y recursos necesarios para una aplicación.

- Keystone, es el componente que proporciona del servicio de identidad a OpenStack. Permite, en general, realizar las tareas de autorización y autenticación necesarios para el uso de los servicios ofrecidos por OpenStack.

2.3.1. Keystone

En de gran importancia explicar con más detalle el componente de identidad de OpenStack, Keystone, ya que el desarrollo de este proyecto está enfocado a ser utilizado por este componente, utilizando las librerías que tiene implementadas.

Keystone es un servicio de OpenStack que proporciona mecanismos de autenticación a los usuarios a través de una API, a los distintos servicios y componentes que ofrece OpenStack [29], así como a cuales recursos tiene autorizado acceder. En general su función principal es la de proporcionar seguridad y control de acceso a los recursos en OpenStack.

Con el desarrollo y crecimiento de la identidad federada, Keystone ha evolucionado para en dar soporte a la federación de identidad. Para comprender como Keystone proporciona seguridad y control de acceso a los servicios de OpenStack, se procede a continuación a explicar sobre las características principales del núcleo de Keystone, Identidad *Identity*, Autenticación*Authentication* y Autorización*Access Management(Authorization)* [30]. A pesar de que en la sección 2.1 se explicaron estas definiciones, aquí se realizará en el contexto Keystone.

Identidad - *Identity*

La identidad se refiere a la identificación de una entidad que intenta acceder a los recursos de OpenStack, se representa por lo general como un usuario, el cual puede estar almacenado de forma local,en la base de datos de Keystone, o de forma externa por medio de un IdP. Para poder identificar a los usuarios un proveedor externo es necesario que Keystone pueda extraer la identidad del usuario del IdP.

Autenticación - *Authentication*

Como se habla en la sección 2.1.1, la autenticación es el procedimiento de comprobación de la identidad de una entidad. En Keystone las contraseñas son tratadas como información que debe de siempre estar segura y protegida en

todo momento. Para esto, Keystone crea un token al inicio de la autenticación, con el cual el usuario podrá realizar posteriores autenticaciones, reduciendo la exposición y uso de la contraseña. Los Tokens tienen un tiempo límite de uso para asegurar el sistema en caso de que los tokens queden comprometidos.Los servicios de OpenStack dependen de los tokens emitidos por Keystone para su acceso.

Autorización - *Access Management(Authorization)*

Una vez el usuario se ha autenticado correctamente y se ha generado el Keystone Token, empiezan los procesos de gestión de acceso, es decir los procesos autorización2.1.2, donde el sistema determina las políticas de acceso que posee el usuario a los recursos de OpenStack. Para esto, Keystone mapea a los usuarios y grupos a proyectos o dominios, asociando a un usuario a un rol específico al proyecto o dominio, y verifica que el usuario posee los permisos para gestionár los recursos a los cuales desea acceder.

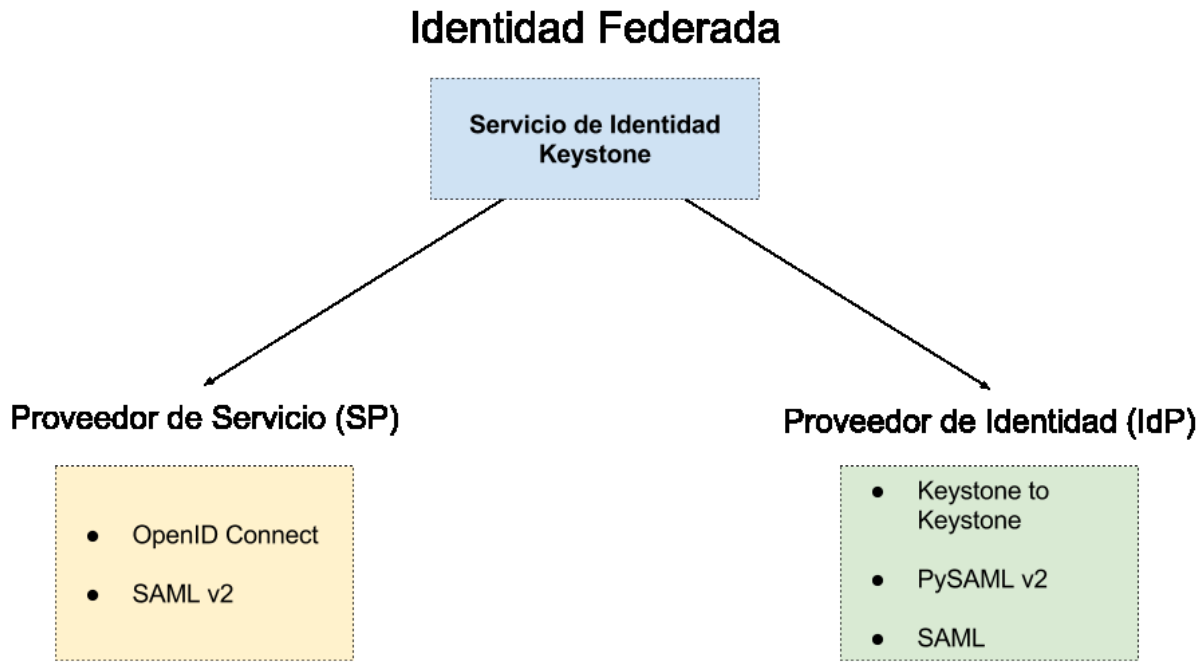

Figura 2.5: Federación de Keystone

El servicio de identidad Keystone se puede usar de dos[1] formas distintas para la federación de identidad [30],como se muestra en la Figura 2.5:

- Keystone como un SP: utilizando identidadas solicitadas a un proveedor de identidad externo, ya sean por medio de claims OpenID Connect o por medio de aserciones SAML[2].
- Keystone como un IdP: satisfaciendo las solitudes de autenticación a nombre de un proveedor de servicio.

En desarrollo del proyecto, se utilizó el servicio de identidad Keystone como un proveedor de servicio(SP), se activó Keystone como un SP configurado para aceptar las aserciones de un IdP Externo mediante los protocolos soportados. Este procedimiento se explicará en el capítulo 3.

Modelo de autorización

Keystone proporciona un modelo de autorización basado en Dominios, el cual proporciona una gran flexibilidad de gestión de los usuarios y de la gestión del acceso a los recursos. En Keystone un dominio es una conjunto de usuarios y proyectos que existen en un entorno OpenStack, donde los usuarios pueden tener roles asignados a un proyecto y asignar el nivel de autorización correspondiente. Esto permite que existan usuarios con el nivel de gestión a nivel de administrador, permitiendo gestionar los recursos de un determinado dominio, a diferencia de un administrador de la nube que puede gestionar los recursos de la infraestructura y entornos en general.

Mapping

En la federación de identidad, los usuarios en Keystone son efímeros, es decir que se evita almacenar datos sobre el usuario dentro de la base de datos, para eliminar las tareas de gestión de usuarios en Keystone. Para dar un nivel de autorización a usuarios efímeros, existe el mecanismo llamado Mapping, que es el encargado dar a las entidades externas un sentido en Keystone, es decir de asignar a los usuarios efímeros la debida autorización en la nube para realizar alguna tarea y también de se asegura de tener los datos suficientes del usuario que permita dar valores a los atributos del token a generar [34].

[1]Keystone permite también que un Keystone actue como un SP y que utilice identidades de otro Keystone actando como un IdP

[2]SAML es un estándar basado en XML usado para la federación de identidad, el documento resultante contiene los atributos del usuario y es llamado un Assertion

Para conseguir esta tarea, Keystone utiliza la información de los claims para realizar el mapeo y otorgar el acceso a un recurso específico a un usuario efímero, es necesario configurar en Keystone las reglas de mappping y asignarlas a un proveedor de identidad y un protocolo específico. En el capítulo 3, se mostrará como crear reglas de mapeo y su asignación a un IdP.

Realización de la autenticación federada

Para obtener el Keystone Token, es necesario seguir unos pasos, a continuación se describen los pasos necesarios para su obtención [30]:

1. Es necesario realizar una autenticación externa, a través de un IdP y obtener un *unscoped token* en Keystone una vez se ha verificado la identidad del usuario. Para su obtención, el usuario debe de acceder a través de una URL de token federada, dedicada y protegida, indicando el IdP y el protocolo de uso. La URL posee el siguiente formato:

```
/identity/v3/OS\-FEDERATION/identity\providers/{idp\id}/protocols/{protocol\id}/
    auth
```

   El usuario es redirigido a la página de autenticación de su IdP y se le pedirán las credenciales. Despues de una correcta autenticación el usuario será redirigido al Endpoint del SP.

2. Determinar las políticas de acceso a los recursos que posee el usuario. El usuario puede realizar solicitudes a la lista de proyectos y dominios a los cuales tiene acceso. El usuario puede ver la lista de proyectos a los cuales está autorizado a acceder, con la siguiente llamada: *GET /auth/projects*

   Obteniendo como por ejemplo esta respuesta:


```
{
  "projects":
  [{
    "domain_id": "37ef61",
    "enabled": true,
    "id": "12d706",
    "links":
    {
      "self": "http://example.com/identity/v3/projects/12d706"
    },
      "name": "a project name"
   },
   {
     "domain_id": "37ef61",
     "enabled": true,
     "id": "9ca0eb",
     "links":
     {
     "self": "http://example.com/identity/v3/projects/9ca0eb"
     },
        "name": "another project"
    }],
    "links":
```

```
    {
      "self": "http://example.com/identity/v3/OS-FEDERATION/projects",
      "previous": null,
      "next": null
    }
  }
```

3. Obtener un *scoped token*, un usuario federado puede solicitar uno, solicitandolo utilizando el *unscoped token*. Para su obtención se deberá especificar ya sea el id o nombre de un proyecto o dominio al cual se tiene acceso.

   A continuación se muestra una solicitud de ejemplo de un *unscoped token*, que contiene la identidad del usuario y el alcance de la información:

```
{
  "auth": {
    "identity": {
      "methods": [
      "token"
      ],
      "token": {
        "id": "--federated-token-id--"
      }
      },
      "scope": {
        "project": {
          "id": "263fd9"
        }
      }
    }
  }
```

   Al intercambiar el *unscoped token* por un *scoped token* se obtiene una respuesta a la solicitud de autenticación, el *token ID* en la cabecera y en el cuerpo un token conteniendo los métodos,roles, usuario, alcance, catálogo e información sobre la emisión y fecha de expiración, a continuación se muestra un ejemplo de respuesta:

```
{
  "token": {
    "methods": [
        "token"
    ],
    "roles": [
      {
        "id": "36a8989f52b24872a7f0c59828ab2a26",
        "name": "admin"
      }
    ],
    "expires_at": "2014-08-06T13:43:43.367202Z",
    "project": {
      "domain": {
        "id": "1789d1",
        "links": {
          "self": "http://example.com/identity/v3/domains/1789d1"
        },
        "name": "example.com"
      },
```

```
        "id": "263fd9",
        "links": {
          "self": "http://example.com/identity/v3/projects/263fd9"
        },
        "name": "project-x"
      },
      "catalog": [
        {
          "endpoints": [
            {
              "id": "39dc322ce86c4111b4f06c2eeae0841b",
              "interface": "public",
              "region": "RegionOne",
              "url": "http://example.com/identity"
            },
            {
              "id": "ec642f27474842e78bf059f6c48f4e99",
              "interface": "internal",
              "region": "RegionOne",
              "url": "http://example.com/identity"
            },
            {
              "id": "c609fc430175452290b62a4242e8a7e8",
              "interface": "admin",
              "region": "RegionOne",
              "url": "http://example.com/identity"
            }
          ],
          "id": "266c2aa381ea46df81bb05ddb02bd14a",
          "name": "keystone",
          "type": "identity"
        }
      ],
      "user": {
        "domain": {
            "id": "Federated"
        },
        "id": "username%40example.com",
        "name": "username@example.com",
        "OS-FEDERATION": {
          "identity_provider": "ACME",
          "protocol": "SAML",
          "groups": [
              {"id": "abc123"},
              {"id": "bcd234"}
          ]
        }
      },
        "issued_at": "2014-08-06T12:43:43.367288Z"
      }
}
```


Una vez el usuario es autenticado , recibe el token generado, que le autoriza y le da acceso a los servicios de OpenStack. Por defecto el token tiene un tiempo máximo de uso de una hora y se recomienda establecer el tiempo de expiración según el tiempo que tardan en completarse de las tareas que el usuario requiera hacer. En la sección3.6.2 se muestra un ejemplo del keystone token generado.

Es conveniente tener presente la fecha de expiración del token al recibirlo, ya que si se emplea para realizar tareas que requieran mucho tiempo de ejecución, con un token con tiempo de expiración corto, es posible que el token caduque cuando los procesos del usuario siguen en ejecución, pudiendo causar problemas y pérdida de datos al usuario.

Capítulo 3

Desarrollo del proyecto

En este capítulo se explicarán las fases de como se ha desarrollado el proyecto, desde las primeras fases de análisis de la teoría, pasando por la fase de pruebas iniciales de herramientas para conocer su funcionamiento, siguiendo con la fase de desarrollo y despliegue del plug-in, el levantamiento de un servidor OpenStack con su debida configuración y finalmente con las pruebas y resultados.

3.1. Formación

Una de las primeras tareas fundamentales para el desarollo del proyecto fue la formación para adquirr los conocimientos básicos. En esta fase se estudió la base teórica de las herramientas y tecnologías a utilizar en el proyecto.

Se realizó una investigación sobre los protocolos a utilizar. El objetivo principal de esta tarea fue la de conocer el funcionamiento de los protocolos, adquirir la base teórica, la cual se puede observar en el capítulo 2, principalmente en OAuth 2.0(sección 2.2.1) y OpenID Connect(sección 2.2.2).

3.2. Pruebas básicas

Una vez realizada la fase de investigación, se procedió a la fase de primer contacto con herramientas que permitieron ver el funcionamiento de los protocolos estudiados. Se utilizó en primer lugar, la librería oauth2.py [13], la cual es una librería creada por Google como herramienta para la autenticación de Gmail.

Como se ha comentado en la sección 2.2.1, es necesario registrarse en el IdP, el IdP utilizado fue Google. Se procedió al registro de una aplicación OAuth en Google y se obtuvieron las credenciales, el *OAuth client id* y el *OAuth client secret*. A continuación se describen brevemente los pasos para la obtención de estas credenciales, para más detalles ver Setting up OAuth 2.0:

1. Se debe poseer una cuenta activa de Google.
2. Ir a la consola de desarrollo de APIs Google.
3. Seleccionar un proyecto, o crear uno nuevo.
4. Seleccionar APIs y servicios.
5. Abrir la pestaña de Credenciales.
6. Añadir una Credencial nueva seleccionando ID de cliente OAuth.
7. Seleccionar el tipo de aplicación,en este caso sería Otro.
8. Crear ID de cliente.

Un vez se han creado las credenciales , Google devuelve las credenciales esperadas, en este caso las credenciales creadas son para el uso de una entidad(End-user), a continuación se muestra un ejemplo de estas credenciales:

```
{"installed":
  {
      "client_id":"123456789123-1a2b3c4d5e6f7g8h9i1a2b3c4d5e6f7g8.apps.googleusercontent.com
          ",
    "client_secret":"pWJHSIa0NtXIXtZoaF7m1234",
      "project_id":"proyecto-id",
    "redirect_uris":["urn:ietf:wg:oauth:2.0:oob","http://localhost"],
      "auth_uri":"https://accounts.google.com/o/oauth2/auth",
      "token_uri":"https://accounts.google.com/o/oauth2/token",
      "auth_provider_x509_cert_url":"https://www.googleapis.com/oauth2/v1/certs",
  }
}
```

Obtenidas las credenciales OAuth, se procedió a realizar las pruebas con la librería oauth.py, para esto, se descargó desde GitHub [13] y se procedió a realizar la petición por los OAuth tokens, especificando en el campo scope, el alcance que tendrá el *access token* a los recursos protegidos,a continuación se muestra el comando comando:

```
$ python oauth2.py
  --generate_oauth2_token
  --user alu0100837094@ull.edu.es
  --client_id=918394529313-apk3gu0m4aqld5qd7c
    0ldkli1vtn77lv.apps.googleusercontent.com
  --client_secret=0bRV9U8I7trXkWmj7FAYrqTE
  --scope 'openid profile email'
```

Al ejecutar el comando anterior, Google presenta una URL donde solicita autorizar el token, se aceptan las condiciones y Google devuelve un código de

verificación que se debe ingresar en la ejecución del script, cuando ha terminado la ejecución del script y Google ha validado al usuario y al código, se obtienen los tokens. Google proporciona un *access token* con su respectivo *refresh token*, el *access token* puede ser usado hasta su expiración y el *refresh token* se puede usar indefinidamente, por esto, se debe de almacenar para su futuro reuso, a continuación se muestra un ejemplo de la salida del script:

```
To authorize token, visit this url and follow the directions:
https://accounts.google.com/o/oauth2/auth?123456789123-1a2b3c4d5e6f7g8h9i1a2b3c4d5e6f7g8.apps
    .googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&response_type=
    code&scope=https%3A%2F%2Fmail.google.com%2F
Enter verification code: 7/UpKNDGOK9QbqqMSqIh5zF_SRkr_glPrLr957FsK2GjI


  {
   "access_token" : "ya29.GlucBAEFs_YKy7GIcEzsdQW
                     TTDLbD7u07THjS7xxOzJmYKxBMLs
                     RRQYTXzz4dIu2pKx3BSQm1L4zjhB
                     yc1Pw8pRfTJz5UK-o0ZfUrQE_QDR
                     SSizu3vVYXAZz5XwT",
   "expires_in" : 3600,
   "refresh_token" : "1/ExMuDCyGcqaTVXUIGoyp5ML2ZwV
                     e0Q6JMGtd66uoyQJNDz4GSMDTi3fncKq4eEZB",
   "token_type" :    "Bearer"
 }
```

Obtenidos los OAuth tokens, se procedió a utilizar el *refresh token*, para refrescar el *access token*, a continuación se muestra un ejemplo el proceso:

```
  $ python oauth2.py
   --client_id=123456789123-1a2b3c4d5e6f7g8h9i1a2b3c4d5e6f7g8.apps.googleusercontent.com
   --client_secret=pWJHSIa0NtXIXtZoaF7m1234
   --refresh_token=123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJ


   Access Token: ya29.GlvtAznwcYa47QihOzfcAUVTenCWfD6JS
   0ivKuFZP0CgYpGJi7flj-6z7Ynbeg7GDnYbM
   f7QsPmH9pCvhso7eaLU98bXVvysf
   OpMFkQzi0BMuBehMDID89XZPtwP
   Access Token Expiration Seconds: 3600
```

Aparte de la librería oaut2.py, también se utilizó la herramienta de Google, *OAuth 2.0 Playground*, la cual es un *sandbox* que permite realizar pruebas con las distintas APIs que Google soporta de forma visual con el protocolo OAuth 2.0 [12].

Como se puede ver en la Figura 3.1, se muestra un ejemplo del flujo de obtención de los tokens de OAuth 2.0 al IdP Google. (A) El cliente solicita a Google un *access token* en nombre del usuario, (B) el IdP Google necesita que el usuario ingrese sus credenciales y de su consentimiento a que un servicio de un cliente acceda a su información, (C)el IdP Google responde con un código de autorización, el cual es utilizado (D) por el cliente para solicitar un *access token*, (E) el IdP Google responde con el *access token* y su *refresh token*, una

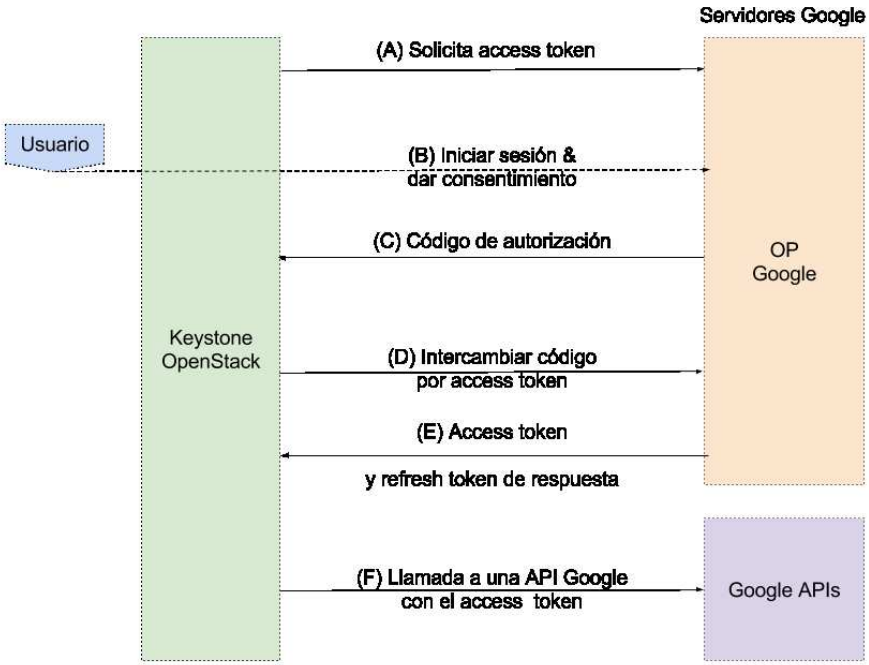

Figura 3.1: Obtención Tokens OAuth 2.0 de Google

vez el cliente posee el *access token*, puede usarlo (F) para solicitar a una API Google información del usuario.

3.3. Ingeniería Inversa

Entendido el funcionamiento de los protocolos y realizadas las pruebas con las herramientas, se procedió al análisis del código de Keystone. Se empezó leyendo la documentación [30], entendiendo su funcionamiento, sobre todo examinando principalmente los métodos de autenticación implementados. Se procedió a descargar la librería Keystoneauth desde GitHub [27], la cual es el paquete que contiene todas las herramientas que permiten la autenticación en una infraestructura en la nube basada en OpenStack.

Se encontró que esta librería contiene los plug-ins de autenticación, como por password, token y plug-ins basados en sistemas federados y se identificó que existen plug-ins en versión 2 y otros en versión 3. A través de la documentación se comprobó que los de versión V3 son los actuales y que las futuras líneas de desarrollo deben seguir en ésta. La librería utiliza una sesión para mantener las configuraciones de los clientes durante los procesos de peticiones, está basado en la librería de peticiones de Python.

Para analizar la estructura del código de Keystoneauth, se utilizó la herramienta Pylint [21], que contiene una opción, usando PyReverse [22], de crear diagramas UML a partir de código Python.Se extrajeron distintos diagramas UML utilizando distintas parámetros y ver los distintos UML que generaba, analizando la estructura de clases y herencia.

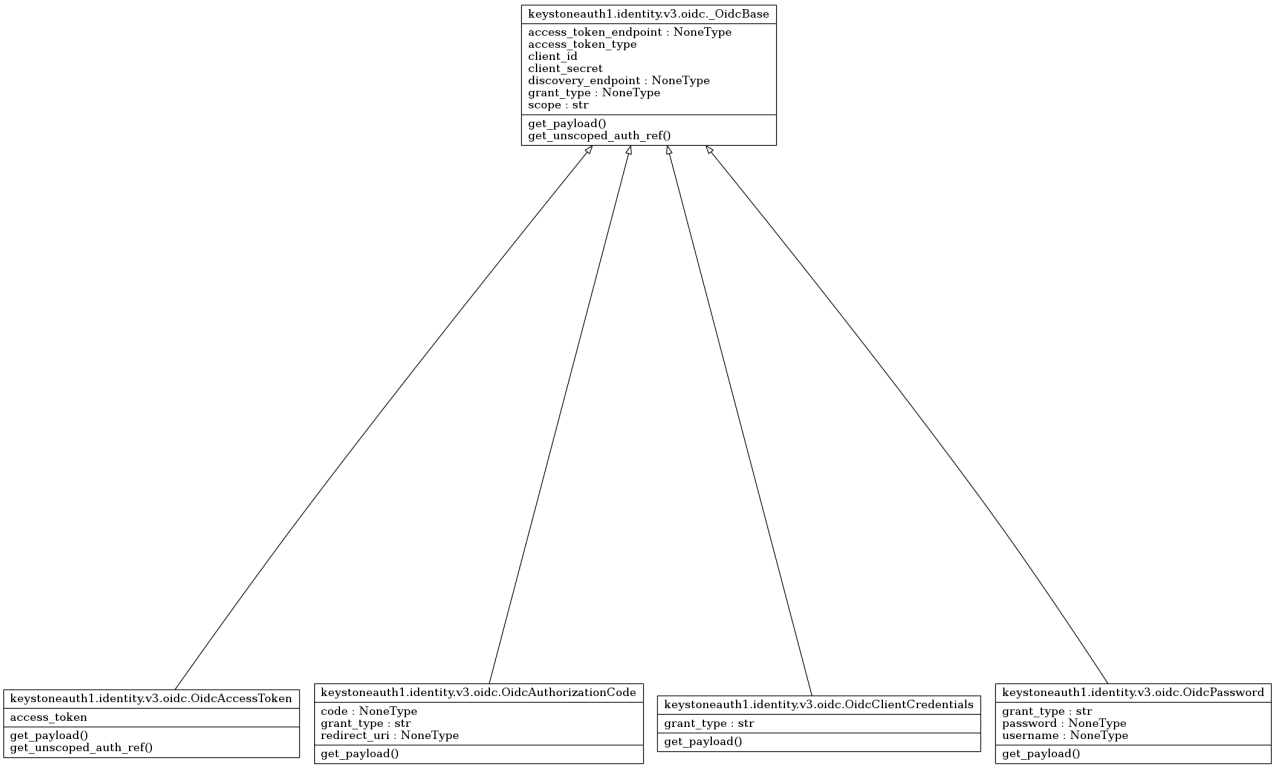

Figura 3.2: Estructura de clases OpenID Connect de Keystoneauth

Un fragmento de un diagrama UML generado,es la Figura3.2 que muestra la estructura que siguen los plug-ins implementados que utilizan el protocolo OpenID Connect.

Se analizó el código de los plug-ins, estudiando sus componentes, viendo cómo estaban implementados, así como entendiendo el funcionamiento de cada uno, con los atributos que cada uno usa. Se prestó atención especial a la herencia, analizando la herencia profunda que la librería posee.

3.4. Implementación plug-in

3.4.1. Desarrollo en Keystone

Entendido el funcionamiento de los plug-ins que tiene implementados la librería Keystoneauth, se procedió a la implementación del plug-in, el cual debe de utilizar el *refresh token* para la obtención del *access token* y con este último

realizar la petición al IdP para la obtención de las claims de usuario.Se explican en este apartado cómo se realizó el plug-ing, los problemas obtenidos y las soluciones encontradas.

Se identificaron los ficheros principales para la implementación del código, los cuales fueron los siguientes:

keystoneauth1/identity/v3/oidc.py

En este fichero se implementó la clase *OidcRefreshToken*, con su constructor y dos métodos:
El primer método,*get_payload*, es el encargado de con el *refresh token* crear la carga útil para la petición que se realiza al IdP. El segundo método, *get_unscoped_auth_ref*, llama al método para la creación de la carga útil y realizar la petición a un método heredado para intercambiar las credenciales por un nuevo *access token* que lo hace al *access token endpoint* del *OpenID Connect Provider*(OP).

Obtenido el *access token*, se procede a la solicitud de un *keystone token*. El *keystone token* se solicita a un método hereado, que tiene como parámetro el *access token*, en la cabecera de un *Authorization: Bearer* que es enviado a una URL protegida, esto activa al *OpenID Connect Provider*(OP) para empezar a realizar una instronspección y que devuelva información del usuario en forma de claims de OpenID Connect. Los claims son enviados a Keystone en forma de variables de entorno en la sesión del usuario.

keystoneauth1/loading/_plugins/identity/v3.py

En este fichero se programó una clase llamada *OpenIDConnectRefreshToken*, en la cual se indica que el código implementado en el fichero antes mencionado es un plug-in, es decir, se carga como un plug-in más de Keystone y se indican cuales son las opciones que éste requiere para su funcionamiento.

keystoneauth1/loading/_plugins/__init__.py

En este fichero se configuró la importación del plug-in para permitir la importación de éste dentro de la librería.

3.4.2. Despliegue

En principio se tenía planeado desplegar el plug-in desarrollado utilizando el sistema de revisión de código usado por OpenStack, Gerrit. Este sistema está implementado para la revisión del código nuevo en los proyectos de OpenStack, donde se revisa línea a línea el código y se revisará si el código es aceptado para ser añadido al código base de un proyecto [23], permitiendo mantener la

calidad del código, así como también su sustentación por parte de la comunidad OpenStack.

Durante el desarrollo del proyecto se decidió en vez de utilizar el sistema Gerrit, implementar y desplegar el plug-in como un paquete Python disponible en PyPI. Para esto se procedió a la creación de un paquete Python desde cero, utilizando la documentación [35] [10] sobre el empaquetado y despliegue de proyectos, dando como resultado la creación del paquete *keystoneauth-oidc-refreshtoken*.

3.4.3. keystoneauth-oidc-refreshtoken

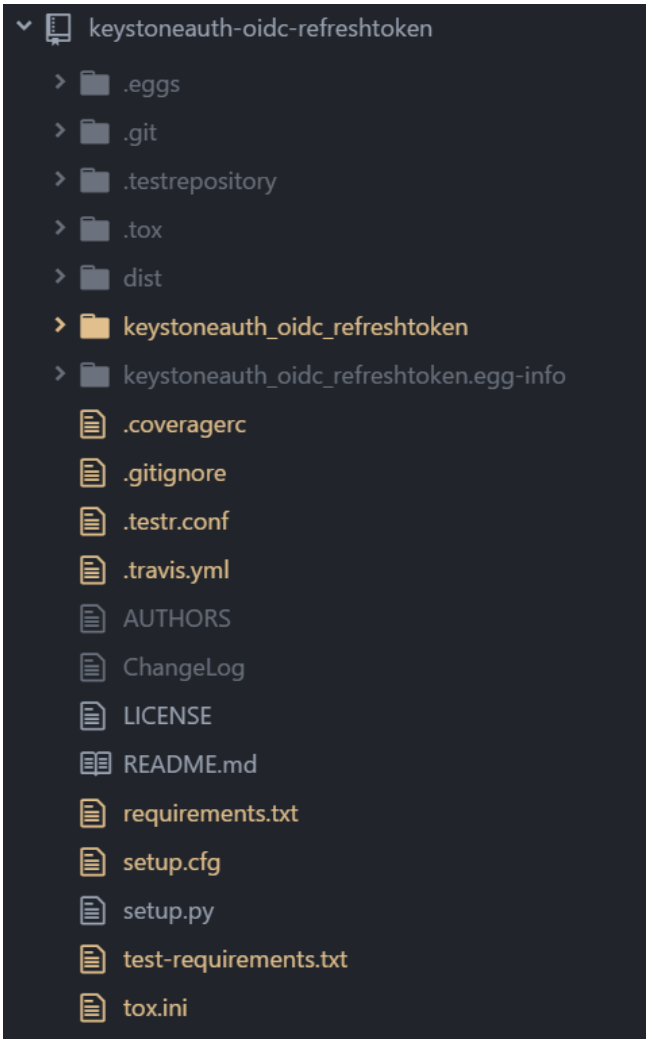

Figura 3.3: Estructura del paquete

El paquete se empezó por la creación de la estructura básica necesaria. La estructura final de éste se puede ver en la Figura 3.3, donde el código principal

se encuentra dentro del directorio *keystoneauth_oidc_refreshtoken*. La configuración del paquete se encuentra dentro del fichero *setup.cfg*, ya que es llamado utilizando la herramienta setuptools [3], usado para registrar el paquete, construir el paquete y desplegarlo en pypi.

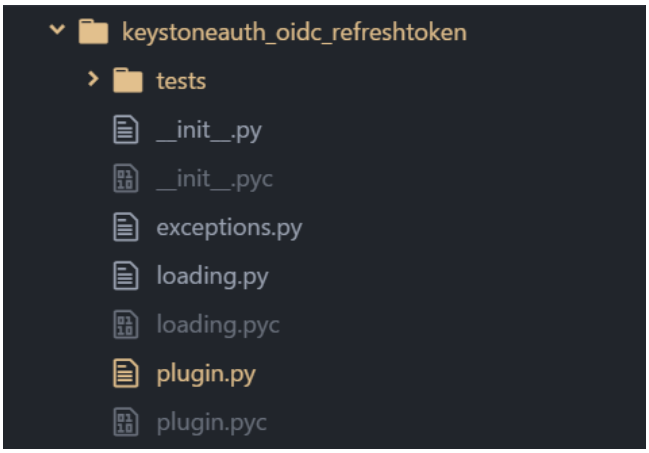

Figura 3.4: Contenido directorio principal

Dentro de este directorio, Figura3.4, se encuentran los ficheros principales del plug-in, *plugin.py* y *loading.py*, en estos ficheros se implementó el código descrito en la sección3.4.1.

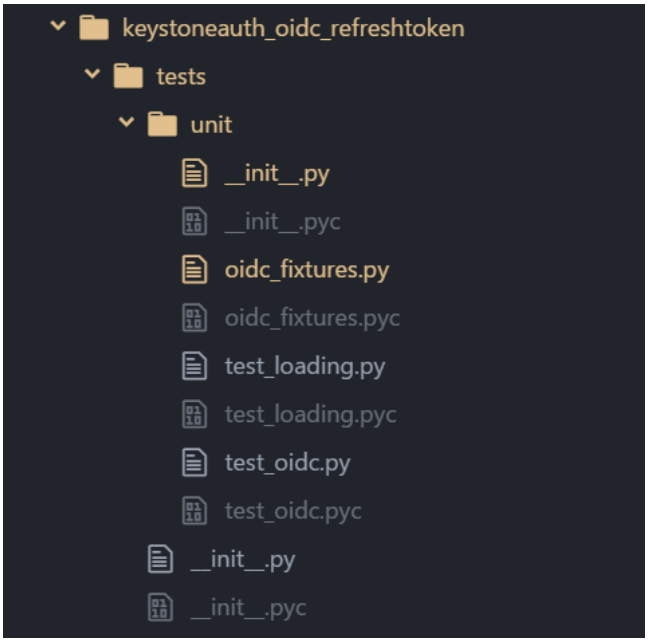

Figura 3.5: Directorio tests unitarios

Como se puede observar, hay un directorio llamado tests, Figura 3.5, el cual contiene los tests unitarios necesarios para probar el correcto funcionamiento del código implementado. Los test unitarios se realizaron utilizando la herramienta Tox [20].

Tox se utilizó para automatizar la tareas de comprobar que el paquete se instalara de forma correcta, así como de correr los test y el de correr el PEP8, permitiendo tener un código Python estandarizado. La automatización con Tox se complementó utilizando la herramienta de integración continua Travis [4], la cual ejecuta Tox al realizar un commit en el repositorio Github del plug-in.

Al comprobar que los tests se pasaban con éxito y que se cumplía con PEP8, se procedió al despliegue del paquete. El paquete puede ser encontrado en pypi en el siguiente enlace, keystoneauth-oidc-refreshtoken 0.1.0.

3.4.4. Pruebas con Flask

Al desplegar el plug-in como paquete, se procedió a la fase de pruebas con una simulación de respuesta de un servidor OpenStack. Para la realización de las pruebas se instalaron los paquetes necesarios en un entorno virtual y se procedió a la instalación del plug-in.

En la plataforma IaaS de los servicios TIC de la Universidad de La Laguna, se montó una máquina Ubuntu Linux 16.04, en el cual se instaló un Framework para Python llamado Flask [1]. Se configuró Flask para devolver un ejemplo de *scoped token* como el mostrado en la sección 2.3.1, al recibir una petición a la URL protegida que se configuró previamente en el Framework. Se comprobó el correcto funcionamiento del plug-in al comprobar que se obtenía el *Access Token* con el *Refresh Token* y que se realizaba la petición al URL protegida.

En la sección 3.5, se mostrarán las salidas pertinentes, en un entorno de producción, a la creación del *keystone token*, por eso se ha evitado mostrar en esta sección.

3.5. Servidor OpenStack

Comprobado el correcto funcionamiento del plug-in desplegado, se procedió al montaje de la infraestructura en la nube OpenStack en la plataforma IaaS de los servicios TIC de la universidad, a continuación se describen los pasos seguidos.

3.5.1. Instalación

El servicio de identidad Keystone, corre sobre Apache, por lo que se procedió a la instalación de Apache:

```
$ sudo apt-get install apache2
```

Una vez instalado el servidor Apache, se procedió a la intalación de los módulos que dan soporte OpenID Connect a Apache, libjansson4, libhiredis0.13, libcurl3, mod_auth_openidc y libcjose0:

```
$ sudo apt-get install libjansson4 libhiredis0.13 libcurl3

$ wget https://github.com/pingidentity/mod\_auth\_openidc/
releases/download/v2.2.0/libcjose0\_0.4.1-1.wily.1\_amd64.deb
$ sudo dpkg -i libcjose0_0.4.1-1.wily.1_amd64.deb

$ wget https://github.com/pingidentity/mod_auth_openidc/
releases/download/v2.2.0/libapache2-mod-auth-openidc\_2.2.0-1.wily.1\_amd64.deb
$ sudo dpkg -i libapache2-mod-auth-openidc_2.2.0-1.wily.1_amd64.deb
```

Al finalizar la instalación, se habilitó el módulo Apache:

```
$ sudo a2enmod auth_openidc
```

Se clonó el repositorio DevStack que contiene una serie de scripts para facilitar el levantamiento de OpenStack, es necesario tener instalado git, python-pip y curl:

```
$ sudo apt-get install python-pip git curl vim -y
$ git clone https://github.com/openstack-dev/devstack
```

Al tener un acceso limitado en la plataforma IaaS , tocó modificar el fichero stackrc de DevStack, ya que no se poseían permisos al puerto del protocolo GIT, se modificó la línea 259 para usar HTTPS:

```
259 GIT_BASE=${GIT_BASE:-https://git.openstack.org}
```

Se creó un fichero llamado local.conf con la siguiente configuración:

```
$ cd devstack
$ git checkout stable/newton
$ cat >> localrc << EOF
> RECLONE=yes
> ENABLED_SERVICES=key,g-api,g-reg,n-api,n-crt,n-obj,n-cpu,n-net,n-cond,cinder,c-sch,c-api,c
    -vol,n-sch,n-cauth,horizon,mysql,rabbit
> SERVICE_TOKEN=openstack
> ADMIN_PASSWORD=openstack
```

```
> MYSQL_PASSWORD=openstack
> RABBIT_PASSWORD=openstack
> SERVICE_PASSWORD=openstack
> LOGFILE=/opt/stack/logs/stack.sh.log
> LIBS_FROM_GIT=python-keystoneclient,python-openstackclient
> EOF
```

Finalmente se procedió a la ejecución del script de DevStack para la instalación, hay que ejecutar el script utilizando un usuario no-root pero que sí tenga "sudo"habilidado, se utilizó el usuario übuntu"que viene por defecto en la máquina:

```
./stack.sh
```

El proceso de instalación suele tardar entre 30-40 minutos.

3.5.2. Configuración

Finalizada la instalación, se configuraron en el fichero */etc/keystone/keystone.conf*, las opciones de federación, en las secciones [auth] y [federation]. Se añadió la dirección de acceso a la interfaz gráfica proporcionada por Horizon, ver sección 2.3 sobre los componentes de OpenStack, que será el portal de acceso web a los usuarios y se añade el protocolo oidc como método de autenticación :

```
  $fqdn=alu0100837094.iaas.ull.es/dashboard/auth/websso/

$ sed -i "s/#methods = external,password,token,oauth1/methods =
  external,password,token,oidc/g" /etc/keystone/keystone.conf
$ sed -i "s/#oauth1 = keystone.auth.plugins.oauth1.OAuth/oidc =
  keystone.auth.plugins.mapped.Mapped/g" /etc/keystone/keystone.conf
$ sed -i "s/#remote_id_attribute = <None>/remote_id_attribute =
  HTTP_OIDC_ISS/g" /etc/keystone/keystone.conf
$ sed -i "s/#trusted_dashboard =/trusted_dashboard =
  http:\/\/$fqdn/g" /etc/keystone/keystone.conf
```

Antes de la creación de un dominio, ver sección 2.3.1 sobre el modelo de autorización Keystone, se configuraron unas variables de entorno:

```
$ export OS_IDENTITY_API_VERSION=3v
$ export OS_PASSWORD=openstack
$ export OS_AUTH_URL=alu0100837094.iaas.ull.es/dashboard/auth/websso/
$ export OS_USERNAME=admin
$ export OS_TENANT_NAME=admin
$ export OS_USER_DOMAIN_ID=default
$ export OS_PROJECT_DOMAIN_ID=default
```

A continuación se creó el dominio y el proyecto :

```
$ openstack domain create google_domain
$ openstack project create google_project
```

A continuación se creó un grupo para que los usuarios se puedan autenticar de forma federada y puedan ser mapeados dentro de un grupo y se asignó el grupo al proyecto creado:

```
$ openstack group create google_users
$ openstack role add member --group google_users --project google_project
```

Se creó un fichero JSON que contiene las políticas de acceso que se describieron en la sección 2.3.1 y se introduce el id del grupo creado anteriormente, a continuación se muestran las reglas de mapeo del proyecto:

```
$ rm -rf mapping.google.json
$ cat >> mapping.google.json << EOF
  [
  {
    "local": [
      {
        "user":
        {
          "name": "{0}"
        },
        "group":
        {
          "name": "google_users",
          "id":  "22415bd84e5a40a78bbdab927cd77f"
        }
      }
    ],
    "remote": [
      {
        "type": "REMOTE_USER"
      },
      {
        "type": "HTTP_OIDC_ISS",
        "any_one_of":
        [
          "https://accounts.google.com"
        ]
      }
    ]
  }
]
EOF
$ sed -i "s/replace_group_id/$group_id/g" mapping.google.json
```

Para implementar las reglas del JSON anterior, se asignó a un proveedor de identidad , que en el proyecto es Google y se le asignó el protocolo de federación oidc, a continuación se muestra la asignación de las reglas de mapeo del proyecto:

```
$ openstack identity provider create google
  --remote-id https://accounts.google.com
$ openstack mapping create google_mapping
  --rules mapping.google.json
$ openstack federation protocol create oidc
  --identity-provider google --mapping google_mapping
```

Se configuró el fichero */etc/apache2/sites-available/keystone-wsgi-public.conf*. Previamente se obtuvieron las credenciales de Google, ver sección3.2, y al ge-

nerar las credenciales se obtuvo un client_id y client_secret para el servidor. Al momento de realizar el registro del servidor se deben introducir las direcciones de redirección protegidas como se muestra en el fichero. También es necesaria la activación de las APIs Google a las cuales se desea obtener los claims. Esencialmente lo que se hace en el fichero es configurar las URL protegidas y dar valores a las propiedades de OpenID Connect, así como definir el proveedor de los servicios de identidad OpenID Connect.

A continuación se muestra el fichero:

```
ProxyPass "/identity" "unix:/var/run/uwsgi/keystone-wsgi-public.socket|uwsgi://uwsgi-uds-
    keystone-wsgi-public/" retry=0

# Configure OIDC
OIDCClaimPrefix "OIDC-"
OIDCResponseType "code token id_token"
OIDCScope "openid email profile"
OIDCProviderMetadataURL https://accounts.google.com/.well-known/openid-configuration
OIDCClientID 918394529313-7v9put0bol4b66nmdsas0o60ru4c9g66.apps.googleusercontent.com
OIDCClientSecret 18OQr-8Y0wmsVyk2WnSSpGum
OIDCCryptoPassphrase openstack

OIDCRedirectURI http://alu0100837094.iaas.ull.es/identity/v3/OS-FEDERATION/websso/oidc/
    redirect
OIDCRedirectURI http://alu0100837094.iaas.ull.es/identity/v3/OS-FEDERATION/
    identity_providers/google/protocols/oidc/auth/redirect
OIDCAuthRequestParams prompt=none

OIDCOAuthClientID 918394529313-7v9put0bol4b66nmdsas0o60ru4c9g66.apps.googleusercontent.com
OIDCOAuthClientSecret 18OQr-8Y0wmsVyk2WnSSpGum

OIDCOAuthIntrospectionEndpoint https://www.googleapis.com/oauth2/v1/tokeninfo
OIDCOAuthIntrospectionTokenParamName access_token
OIDCOAuthRemoteUserClaim email

<Location ~ "/identity/v3/OS-FEDERATION/identity_providers/google/protocols/oidc/auth">
  AuthType oauth20
  Require valid-user
  SetEnv HTTP_OIDC_ISS https://accounts.google.com
  LogLevel debug
</Location>

#<Location ~ "/identity/v3/OS-FEDERATION/identity_providers/google/protocols/oidc/auth">
#  AuthType openid-connect
#  Require valid-user
#  LogLevel debug
#</Location>

#<Location ~ "/identity/v3/auth/OS-FEDERATION/websso/oidc">
#       AuthType openid-connect
#       Require valid-user
#       LogLevel debug
#</Location>
```

En el JSON anterior se puede ver los parámetros del módulo OpenID Connect de Apache, en el cual es configurado primero el protocolo OpenID Connect y posteriormente el protocolo OAuth 2.0 y se configuró la URL protegida, especificando el tipo de autenticación a emplear. Como se puede observar, el método de autenticación utilizada es oauth20, esto se debe a que al realizar las peticiones al proveedor de identidad utilizando openid-connect, se obtenía una redirección por parte del IdP para que la entidad(end-user) se autenticara y autorizara al

cliente como si de un SSO se tratara. Al ser la gestión desatendida el objetivo del proyecto, se decantó por el primer método.El código comentado pertenece a la configuración SSO, ver sección 3.6.1 .

Keystone permite habilitar el SSO, pero se debe configurar manualmente ya que no viene habilitado por defecto. En el proyecto se utilizó el SSO para comprobar el correcto funcionamiento del servidor, ver sección 3.6.1. Para esto movió la plantilla de sso_call_template.html al directorio Keystone y se configuró el servidor para que se puedan realizar autenticaciones por medio Google usando el SSO, se modificó el fichero */opt/stack/horizon/openstack_dashboard /local/local_settings.py*, habilitando el servicio SSO especificando que el método de autenticación es OpenId Connect, a continuación se muestran los comandos usados :

```
$ cp /opt/stack/keystone/etc/sso_callback_template.html /etc/keystone/

$ sed -i "s/^OPENSTACK_KEYSTONE_URL=.*/OPENSTACK_KEYSTONE_URL
      = \"http:\/\/alu0100837094.iaas.ull.es\/identity\/v3\"/g"
      /opt/stack/horizon/openstack_dashboard/local/local_settings.py

$ cat >> /opt/stack/horizon/openstack_dashboard/local/local_settings.py << EOF
    OPENSTACK_API_VERSIONS = {
     "identity": 3
    }

    WEBSSO_ENABLED = True
    WEBSSO_CHOICES = (
    ("credentials", _("Keystone Credentials")),
    ("oidc", _("OpenID Connect"))
    )

    WEBSSO_INITIAL_CHOICE = "oidc"
  EOF
$ cp  /opt/stack/horizon/openstack_dashboard/local/local_settings.py
      /opt/stack/horizon/openstack_dashboard/local/enabled/
```

Para finalizar se reinició el servidor apache:

```
$ sudo service apache2 restart
```

3.6. Pruebas y resultados

Al finalizar la configuración del servidor OpenStack y principalmente la configuración de la federación con OpenID Connect, se efectuaron pruebas con el plug-in desarrollado, sección 3.4.3, para comprobar la correcta configuración del servidor.

3.6.1. SSO - Single Sign On

SSO es uno de las características más solicitadas en OpenStack, pero es necesario su configuración para su implementación, al no estar en los servicios por defecto en Keystone. La configuración se puede observar en la sección 3.5.2. Para probar la correcta instalación y configuración del servidor OpenStack,así como también del servicio de identidad Keystone, se procedió a realizar el acceso con la cuenta institucional de la ULL.

Como se puede ver en el fichero de configuración SSO las últimas líneas del código están comentadas, ya que en el proyecto el objetivo es realizar la autenticación por medio de la consola en vez de la interfaz web y si se dejan sin comentar, al realizar la llamada con el plug-in desarollado el sistema intentará realizar la autenticación por la interfaz web.

En las Figura 3.6 se puede observar que se el Dashboard posee la opción SSO con OpenID Connect solicita autenticarse con una cuenta Google.

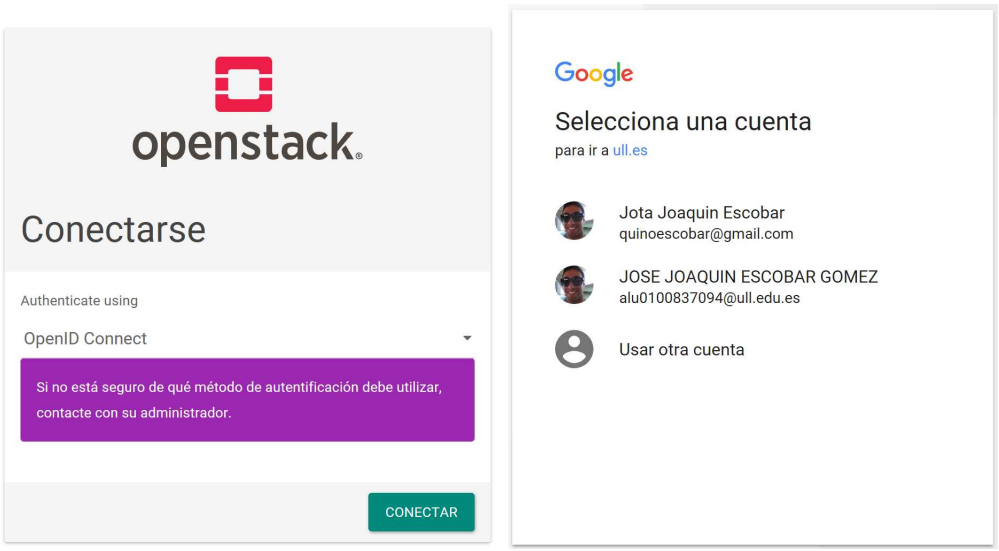

Figura 3.6: Web SSO en OpenStack

Al ser una autenticación por medio de la federación de la identidad, el usuario para OpenStack es efímero, ya que se usa la identidad del usuario proveniente del IdP. En la Figura 3.7, en la esquina superior derecha se puede apreciar que la sesión se ha iniciado usando la identidad del *OpenID Connect Provider*(OP).

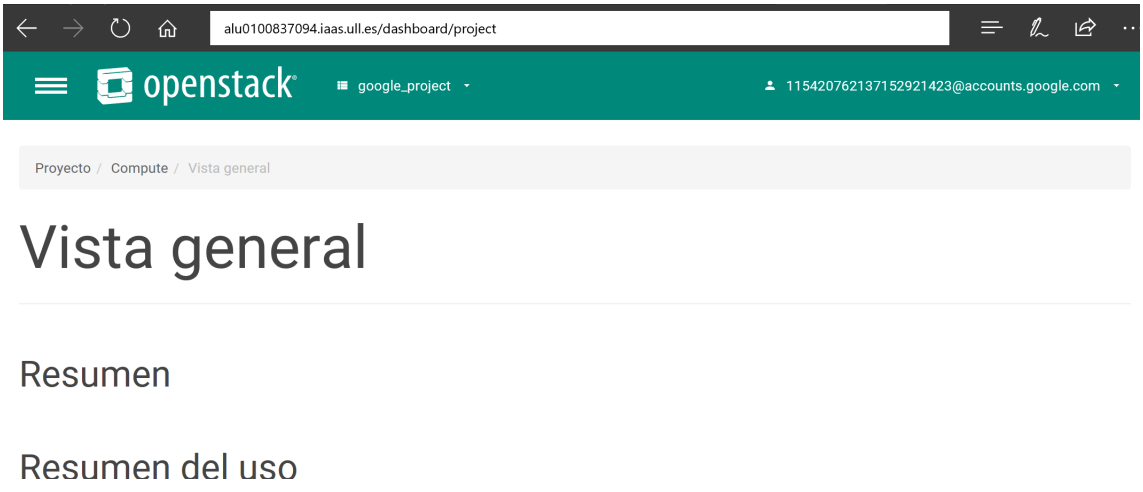

Figura 3.7: Usuario autenticado dentro del Dashboard de OpenStack

3.6.2. Pruebas con OpenStack

Como se ha descrito en la sección 1.2, el objetivo principal del proyecto es la gestión desatendida de OpenStack utilizando la identidad federada, para esto el usuario necesita obtener un *keystone token*.

Se creó un entorno virtual, instalando el cliente de OpenStack, *OpenStackClient* (OSC) [25], el cliente de la API de identidad de OpenStack *Keystoneclient* [24] y el plug-in desarrollado keystoneauth-oidc-refreshtoken 0.1.0.

```
$ virtualenv OSC_env
$ source OSC_env/bin/activate

$ pip install python-openstackclient
$ pip install python-keystoneclient
$ pip install keystoneauth-oidc-refreshtoken
```

Instalados los paquetes, se realizó la llamada del plug-in keystoneauth-oidc-refreshtoken especificando el método de autenticación a utilizar, el protocolo, el IdP, las credenciales OAuth 2.0, el Endpoint del access token y la URL para la autenticación:

```
$ openstack --os-auth-type v3oidcrefreshtoken
  --os-auth-url http://alu0100837094.iaas.ull.es/identity/v3
  --os-refresh-token 1/sp7hglEgjThbyRjjKFq22MAH8kzUYAbcIkiRIokxGUg
  --os-client-secret 0bRV9U8I7trXkWmj7FAYrqTE
  --os-client-id 918394529313-apk3gu0m4aqld5qd7c0ldkli1vtn77lv.apps.googleusercontent.com
  --os-protocol oidc
  --os-identity-provider google
  --os-access-token-endpoint https://www.googleapis.com/oauth2/v4/token
  token issue
  --debug
```

Con la opción –debug, se puede obtener por consola, las ejecuciones que realiza el comando ejecutado, resaltando las peticiones al IdP solicitando refrescar el *Access Token*, obteniendo una respuesta como la que se muestra a a continuación:

```
REQ: curl -g -i -X POST https://www.googleapis.com/oauth2/v4/token
-H "User-Agent: osc-lib/1.6.0 keystoneauth1/2.21.0
python-requests/2.18.1 CPython/2.7.13" -d
'{'grant_type': 'refresh_token',
'refresh_token': u'1/sp7hglEgjThbyRjjKFq22MAH8kzUYAbcIkiRIokxGUg'}'

Starting new HTTPS connection (1): www.googleapis.com

https://www.googleapis.com:443 "POST /oauth2/v4/token HTTP/1.1" 200 None
RESP: [200]
RESP BODY:

{
"access_token": "ya29.Gl28BGR3Z-NhWN6PlzvfGThnoFd9yHFJJvsObOW8JJ_
                 MmFlOiZLs5EYhfeqdGRw0ldJCPe-K6PeAYBeLmuvMC
                 FOcI8LAPMM-gmAsyC8W3iGlEHCMRDvhEtpyl8nJ7Ts",
"token_type":   "Bearer",
"expires_in":   3600,
"id_token":    "eyJhbGciOiJSUzI1NiIsImtpZCI6ImM3OTc2ZTVmYTk
                 0MjM5ZmFlOWI4NjY3MTgxMDIxMWNlZWQ0NjhkMTYifQ.
                                    .
                                    .
                                    .
                 0b69Frip8gtGOARBM_g30ln55vleAyzqcvBx2zqceJ2r
                 yEWlCxCquK9OMLraaW3AeQFyz145GK0Pwl_XmcDEF32h"
}
```

Se puede observar en el JSON anterior la respuesta del IdP Google a la petición de refrescar el *Access Token* utilizando el *Refresh Token* mediante el plug-in implementado, también se puede ver que el IdP también responde con un JWT que contiene claims del usuario. Aunque Google en este caso responda con un Id Token que contiene claims del usuario al realizar la peticion, no quiere decir que otros IdP tengan el mismo comportamiento, por esta razón se implementó en el plug-in el mecanismo para la obtención de los claims del usuario a través del *Access Token*.

Obtenido el *Access Token*, es utilizado para obtener el *Keystone Token* por medio de la solicitud a una URL protegida al servidor OpenStack (el SP), el cual es el encargado de realizar la introspección al IdP Google para obtener los claims del usuario al que pertenecen las credenciales usadas a través de un ID Token y generar el *Keystone Token* al usuario, como se muestra a continuación:

```
REQ: curl -g -i -X POST http://alu0100837094.iaas.ull.es/identity
/v3/OS-FEDERATION/identity_providers/google/protocols/oidc/auth
-H "Authorization: {SHA1}9b014d79380aeb6139ed726d87d9065d27afd3c2"
-H "User-Agent: osc-lib/1.6.0 keystoneauth1/2.21.0
python-requests/2.18.1 CPython/2.7.13"

Starting new HTTP connection (1): alu0100837094.iaas.ull.es
http://alu0100837094.iaas.ull.es:80
"POST /identity/v3/OS-FEDERATION/identity_providers/google
/protocols/oidc/auth HTTP/1.1" 201 444
RESP: [201]
```

```
RESP BODY:
{
  "token":
  {
    "issued_at": "2017-08-09T16:34:04.000000Z",
    "audit_ids": ["ZdBL3pQfQrGbjmJUx6xHdA"],
    "methods": ["oidc"],
    "expires_at": "2017-08-09T17:34:04.000000Z",
    "user":
    {"OS-FEDERATION":
      {
        "identity_provider":
        {
          "id": "google"
        },
        "protocol":
        {
          "id": "oidc"
        },
        "groups":
        [
          {
            "name": "google_users",
            "id": "22415bd84e5a40a78bbdab927cd77fb1"
          }
        ]
      },
      "domain":
      {
        "id":
        "Federated",
        "name":
        "Federated"
      },
      "id": "4fa102af6c5d4e09bdb3ede5550f8eae",
      "name": "alu0100837094@ull.edu.es"
    }
  }
}

POST call to None for
http://alu0100837094.iaas.ull.es/identity/v3/OS-FEDERATION/
identity_providers/google/protocols/oidc/auth
used request id req-00695787-f4bc-4e89-bd9c-8355e3d8e0ca

<Response [201]>

+---------+-----------------------------------------------------+
| Field   | Value                                               |
+---------+-----------------------------------------------------+
| expires | 2017-08-09T17:34:04+0000                            |
| id      | gAAAAABZrJoXmcndPmkdqdpw8QhAA0HJbnhRJSgNKbVC        |
|         | n_k3mYAJzyb_PgV8yvyQFrv9f8YbZU-HPuRcES0IQqJY        |
|         | Q-YDDrDzHDEQqS-d1OJk860sx6K-XjdVbyHmsF5jHrqC        |
|         | hDbxeIhLOqXsqQMgApKYIn8DJFIr7E3knQRwTJL1akEY        |
|         | s6p46sLZpknSi-MYIbjN9F5TyH7v                        |
| user_id | 4fa102af6c5d4e09bdb3ede5550f8eae                    |
+---------+-----------------------------------------------------+
clean_up IssueToken:
END return value: 0
```

Como se explicó en la sección 2.3.1, sobre el mecanismo de mapeo que realiza Keystone, se puede observar en el JSON que se ha mapeado al usuario usando las reglas de mapeo asociadas al IdP Google y al protocolo oidc y se puede ver que se le asignó el grupo especificado al dominio creado para la federación de identidades de Google en OpenStack.

Al terminar la ejecución, al usuario le es retornado los datos del token genera-

do según sus políticas de acceso, es decir después del mapping. El usuario puede hacer uso del id del token para utilizar los servicios de OpenStack a los cuales tiene autorización de acceso. Hay que resaltar que aunque los tokens tienen una fecha de expiración, el id del token es persistente en el backend.

El keystone token generado contiene toda la información que se muestra en el JSON anterior. Toda esta información Keystone la firmada usando un CMS (*cryptographic message syntax*) y una vez firmado se comprime.

3.7. Problemas y soluciones

Durante el desarrollo del proyecto se tuvieron algunos problemas, los cuales se mencionan a continuacion.

Funcionamiento de Keystone

Gracias al volumen de código que posee la librería Keystone, se tuvo el problema de comprender el funcionamiento del mismo. Principalmente se intentó comprender porqué habían métodos programados en dos versiones distintas y cómo era su funcionamiento. Se encontró la solución al problema realizando diagrámas UML, ver sección 3.3, con los cuales se logró entender la herencia existente de los componentes de la librería.

Instalación OpenStack

Al principio del proyecto se utilizaban los servidores del Instituto de Física de Cantabria (IFCA) para realizar el montaje de los servidores y realizar las pruebas, uno de los problemas que se tuvo fue que durante el desarrollo, el IFCA sufrió un apagón, afectando los servidores desplegados, provocando que fueran inaccesibles. Al intentar reiniciar los servidores se obtuvieron mensajes de error. Los servidores estuvieron fuera de servicio y en el tiempo en que los recuperaban, se empezó a montar el servidor desde cero en la plataforma IaaS de los servicios TIC de la Universidad de La Laguna.

Uno de los problemas obtenidos al instalar el servidor OpenStack fue la configuración del mismo, ya que el proyecto OpenStack está en constante evolución, existe mucha documentación sobre cada versión desplegada, es decir que entre versiones publicadas hay multitud de diferencias, provocando que la configuración se realizara mal, al seguir documentación de otras versiones.

IaaS

Al intentar montar el servidor en el IaaS se tuvieron problemas con permisos y puertos a los que se denegaba el acceso, se solucionó con la ayuda del nivel de administrador que posee el tutor del proyecto.

Para acceder al IaaS es necesario estar conectado a la red institucional de la ULL, pero en una fase del proyecto que se realizaba fuera de la universidad, no era posible acceder a éste. La solución fue conectarse mediante VPN.

Obtención de los claims

Al realizar las pruebas una vez implementado el plug-in y configurado el servidor, se obtuvieron denegaciones de acceso, se realizó un análisis de los ficheros de registro de actividad de Apache, Keystone y Horizon, para determinar la causa del problema. De esta forma se solucionaron los problemas, viendo en los logs qué era lo que pasada.

Uno de estos problemas fue difícil de detectar. Se estaba usando un Refresh Token que no poseía los permisos de acceso necesario para acceder a los recursos solicitados por el servidor OpenStack,es decir que estaban llegando datos insuficientes al servidor lo cual daba como consecuencia la denegación de acceso. Se solucionó obteniendo el Refresh Token con el alcance suficiente para obtener la información deseada.

Capítulo 4

Conclusiones y líneas futuras

4.1. Conclusiones

Para facilitar la gestión de las identidades se propone el uso de identidades Federadas, permitiendo por una parte, que los usuarios tengan el control de la información que desean compartir con un cliente, estableciendo el nivel de privacidad y por otra parte la creación de un entorno común entre las distintas tecnologías, como base para favorecer la interoperabilidad.

En la gestión de identidad convencional, la autenticación es por lo general tratada como si fuesen un conjunto de listas definidas, pero cuando el número de usuarios aumenta, la gestión de identidad se vuelve más difícil y sería contradictorio para una infraestructura como servicio (IaaS) que tiene como objetivo principal proporcionar una gestión de los recursos de forma eficiente y eficaz, tenga un servicio que se convierte en una carga con el tiempo y es poco escalable.

Con el proyecto se ha proporcionado la opción de que los usuarios de una infraestructura en la nube basada en OpenStack puedan adquirir un token emitido por el servicio de identidad Keystone, que permite ser usado en tareas de gestión de los servicios a los que tienen acceso de forma desatendida, favoreciendo y fomentando la automatización de tareas de gestión en general.

Finalmente, el desarrollo de este proyecto ha seguido los estándares Open Source, permitiendo que se pueda utilizar en su totalidad, para la modificación y evolución del mismo o ser usado en otros proyectos, cumpliendo con la licencia.

4.2. Líneas futuras

El trabajo futuro de este proyecto está enfocado a la mejora de las procesos de obtención de claims del usuario en el IdP, en el proyecto sólo se ha implementado para utilizar a Google como proveedor de identidad, y cada proveedor de identidad posee distintos mecanismos para la introspección, así como el contenido de los claims depende de un IdP a otro.

Una propuesta para mejorar el proyecto es su implementación en un entorno real, permitiendo realizar pruebas de uso con usuarios reales, ya que la retroalimentación es fundamental para el desarrollo de cualquier producto, proporcionando distintos puntos de vista y favoreciendo el crecimiento del proyecto.

Otra propuesta sería la creación de tareas básicas de gestión, para comprobar que la gestión de los servicios funciona como debe utilizando el keystone token. También una propuesta interesante a futuro es la implementación en la misma infraestructura en la nube, Keystone como SP y Keystone como IdP, a esta configuración se la conoce como Keystone 2 Keystone(K2K).

Capítulo 5

Summary and Conclusions

In order to facilitate the identity management it is proposed the use of Federated identities, allowing users to control the data they want to share with a client, tough establishing a level of privacy and, on the other hand, creating a common environment between the different technologies as a basis for promoting interoperability.

In the traditional identity management, authentication is usually treated as if it were a set of lists, but when the number of user increases, the identity management becomes more difficult and would be contradictory for an IaaS platform,to have a service that becomes a burden over time and is hard to escalate, since it aims to provide an effectively and effectly resource management.

This project has provided the option for users of an OpenStack cloud-based infrastructure, to acquire an issued token by the Keystone identity service, which allows the user to use it in management tasks of services to which it has authorization, in an unattended way, promoting and fostering the automation of management tasks in general.

Finally, the development of this project has followed the Open Source standards, allowing his use in its totality for modifications and improvements thereof or to be used in other projects, fulfilling the licence.

Capítulo 6

Presupuesto

Este capítulo describe el coste de cada una de las tareas de desarollo del proyecto, en un presupuesto estimado con los precios de mercado actuales.

6.1. Presupuesto

Para la realización de este proyecto, el desarrollo ha sido dividido en una serie tareas, las cuales han sido detalladas en la sección1.3. En el desarrollo del proyecto se emplearon una cantidad de horas para cada una de las tareas, como se puede ver en la Tabla 6.1

Estipulando que el precio de mercado para cada hora de trabajo de un ingeniero informático es de 16 Euros, el presupuesto del proyecto se muestra en la tabla siguiente:

Tareas de desarrollo	**Número de horas**
Tarea 0. Planificación	8
Tarea 1. Formación	10
Tarea 2. Pruebas básicas	20
Tarea 3. Ingeniería Inversa	30
Tarea 4. Ampliación de la librería	100
Tarea 5. Implementación servidor	50
Tarea 6. Peticiones al servidor	40
Tarea 7. Documentación	20
Total horas de trabajo	**278**
Equipo informático	500 €
Presupuesto total	**4948 €**

Tabla 6.1: Prespuesto y horas invertidas

Bibliografía

[1] A.Ronacher and contributors. Flask. http://flask.pocoo.org/.

[2] J. Audun. A consistent definition of authorization. *University of Oslo*, 2017.

[3] Python Packaging Authority. Setuptools' documentation. https://setuptools.readthedocs.io/en/latest/.

[4] Travis CI. Getting started with Travis. https://docs.travis-ci.com/user/getting-started/.

[5] Microsoft Corporation. Claims-based identity and concepts in SharePoint. http://msdn.microsoft.com/en-us/library/ee534975.aspx.

[6] Microsoft Corporation. Relying Party Microsoft Developer Network. https://msdn.microsoft.com/en-us/library/ee748466.aspx.

[7] R. Küsters D. Fett and G. Schmitz. A comprehensive formal security analysis of oauth 2.0. *University of Germany*, 2016.

[8] Openid dot net. OpenID Connect Core 1.0. http://openid.net/specs/openid-connect-core-1_0.html#RFC6750.

[9] F.B Schneider E. Birrel. *Federated Identity Management Systems: A privacy-Bases Characterization.* Cornell University, 2013.

[10] Python Software Foundation. Packaging and Distributing Projects. https://packaging.python.org/tutorials/distributing-packages/#packaging-and-distributing-projects.

[11] A. Ganti. Plan 9 authentication in linux. *Google Inc.*, 2008.

[12] Google. Google Playground. https://developers.google.com/oauthplayground/.

[13] Google. Gmail OAuth tools - OAuth2.py, 2012. https://github.com/google/gmail-oauth2-tools/blob/master/python/oauth2.py.

[14] JOSE Working Group. JSON Web Encryption (JWE), 2015. https://tools.ietf.org/html/draft-ietf-jose-json-web-encryption-40.

[15] JOSE Working Group. JSON Web Signature (JWS), 2015. https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-41.

[16] OAuth Working Group. JSON Web Token (JWT), 2014. https://tools.ietf.org/html/draft-ietf-oauth-json-web-token-32.

[17] D. Hardt. Rfc6749 – the oauth 2.0 authorization framework. *IETF.*, 2012.

[18] G. Ahn J. Lam. Managing privacy preferences for federated identity management. *University of North Carolina at Charlotte*, pages 28–29, 2005.

[19] M. Jones. The oauth 2.0 authorization framework: Bearer token usage. *Internet Engineering Task Force*, 2012.

[20] Holger Krekel and Others. The tox automation project. https://tox.readthedocs.io/en/latest/.

[21] Logilab. Pylint. https://pylint.readthedocs.io/en/latest/.

[22] Logilab. Pyreverse : UML Diagrams for Python. https://www.logilab.org/blogentry/6883.

[23] OpenStack. Gerrit Code Review - A Quick Introduction. https:review.openstack.org/Documentation/intro-quick.html.

[24] OpenStack. KeystoneClient. https://github.com/openstack/python-keystoneclient.

[25] OpenStack. OpenStackClient. https://github.com/openstack/python-openstackclient/tree/master/openstackclient.

[26] OpenStack. What is OpenStack? https://www.openstack.org/software/.

[27] OpenStack. keystoneauth, 2015. https://github.com/openstack/keystoneauth.

[28] OpenStack. Federated Identity in Keystone, 2017. https://docs.openstack.org/developer/keystone/federation/federated_identity.html.

[29] OpenStack. Identity V3, 2017. https://developer.openstack.org/api-ref/identity/v3/index.html.

[30] OpenStack. Keystone, the OpenStack Identity Service, 2017. https://docs.openstack.org/keystone/latest/.

[31] M. Gupta R. Sharman, S. Das Smith. *Digital identity and access management*. Information Science Reference, 2012.

[32] Redhat. What is OpenStack? https://www.redhat.com/en/topics/openstack.

[33] T. Rosado and J. Bernardino. An overview of openstack architecture. *Polytechnic Institute of Coimbra - ISEC – Coimbra Institute of Engineering*, 2014.

[34] H.Ñash S. Martinelli and B.Topol. *Identity, Authentication, and Access Management in OpenStack Implementing and Deploying Keystone, OpenStack's Identity Service*. O'Reilly, 2011.

[35] S. Torborg. How To Package Your Python Code. https://python-packaging.readthedocs.io/en/latest/.

[36] W3C. Web services description language (WSDL) v 1.1. Technical report, 2001.

[37] D. Baier V. Bertocci K. Brown S. Densmore E. Pace M. Woloski. *A guide to claims-based identity and access control*. Microsoft Corporation, 2011.