Escola Universitària Politècnica de Mataró

Centre adscrit a:

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Grado en Ingeniería Informática

Simplificación de la gestión de red mediante el uso de APIs

Autor: Isaac Fernández Fontao

Tutor: Dr. Léonard Janer García

Primavera 2018

TecnoCampus Mataró-Maresme

Gracias a todos aquellos que han estado conmigo durante esta etapa de mi vida, en especial a mis abuelos, sin cuya ayuda esto no habría sido posible, y gracias a Aton Systems, que me han dado la oportunidad de crecer y me han acogido como uno más desde el primer momento.

Resumen

El objetivo de este proyecto es el estudio de diferentes fabricantes de equipos de red que ofrecen soluciones para la gestión de red, especialmente aquellos que proveen una API para esta función, pues su uso permite crear herramientas que faciliten la gestión de la red. A través de este estudio, se escogió al fabricante Cisco para profundizar en su funcionamiento.

Después de entender cómo trata Cisco la gestión de red, se ha creado la aplicación Crear Tenant, que permite cambiar un aspecto de la configuración de la red a través de la API, simplificando el proceso respecto a cómo sería hacerlo usando la interfaz gráfica.

Resum

L'objectiu d'aquest projecte és l'estudi de diferents fabricants d'equips de xarxa que ofereixen solucions per la gestió de xarxa, especialment aquells que proveeixen una API per aquesta funció, doncs el seu ús permet crear eines que faciliten la gestió de la xarxa. Mitjançant aquest estudi, s'ha escollit al fabricant Cisco per profunditzar en el seu funcionament.

Després d'entendre com tracta Cisco la gestió de la xarxa, s'ha creat l'aplicació Crear Tenant, que permet canviar un aspecte en la configuració de la xarxa mitjançant l'API, simplificant el procés respecte a com seria fer-ho usant la interfície gràfica.

Abstract

The objective of this project is to study different network equipment manufacturers that offer solutions for network management, especially those who provide an API for this function, since using it allows the creation of tools that facilitate the management of the network. Through this study, Cisco manufacturer was chosen to deepen in how it works.

After understanding how Cisco deals with network management, the application *Crear Tenant* has been created, which allows to change an aspect of the network's configuration through the API, simplifying the process to what it would be like using the graphic interface.

Índice

Índice de figuras **8**

Índice de tablas **10**

Glosario de términos **11**

1. Objetivos **13**

1.1. Introducción 13

1.2. Propósito 13

1.3. Finalidad 13

1.4. Objeto 14

2. Estado del arte de los fabricantes **15**

2.1. Extreme Networks 17

2.1.1. ¿Qué es NetSight? 17

2.1.2. ¿Qué ofrece NetSight? 19

2.1.3. ¿Qué ventajas tiene usar NetSight? 20

2.2. Arista 21

2.2.1. ¿Qué es CloudVision? 22

2.2.2. ¿Qué ofrece CloudVision? 23

2.2.3. ¿Qué ventajas tiene usar CloudVision? 24

2.3. Cisco 24

2.3.1. ¿Qué es ACI? 25

2.3.2. ¿Qué ofrece ACI? 26

2.3.3. ¿Qué ventajas tiene usar ACI? 27

2.4. Aerohive Networks 27

2.4.1. ¿Qué es HiveManager? 28

2.4.2. ¿Qué ofrece HiveManager? 29

2.4.3. ¿Qué ventajas tiene usar HiveManager? 29

2.5. Elección de fabricante 31

3. Introducción a las APIs **33**

4. Cisco: Nexus, ACI y sus APICs **35**

4.1. Terminología ACI 37

4.1.1. ACI Tenants 37
4.1.2. Networking Tenant 38
4.1.3. Policy Tenants 39
4.2. Programabilidad en ACI 41
4.2.1. Modelo de objetos 42
4.2.2. Objetos administrados y Árbol de información de gestión 43
4.2.3. Nombres relativos y Nombres distinguidos 46
4.3. API REST de ACI 48
4.4. ACI Toolkit 50
4.5. Cobra SDK 52
4.6. Comprobar resultados en la GUI de APIC 55
5. Aplicación Crear Tenant 57
5.1. Entendiendo Python 57
5.1.1. Uso de palabras en vez de símbolos 57
5.1.2. Identación 57
5.1.3. Comentarios 57
5.1.4. Variables 58
5.1.5. Condicionales 58
5.1.6. Bucles 58
5.1.7. Métodos 59
5.1.8. Clases 59
5.1.9. Módulos 59
5.2. ¿Qué es Kivy? 60
5.3. Desarrollo de la aplicación 61
5.3.1. Diseño de las pantallas 61
5.3.2. Cuerpo de la aplicación 68
5.4. Montaje de la aplicación 71
6. Conclusiones 73
7. Bibliografía 75

Índice de figuras

Fig. 1: Diagrama de Gartner del estado de los fabricantes en el mercado 16

Fig. 2: Logo de Extreme Networks 17

Fig. 3: Pantalla de login de NetSight 18

Fig. 4: Captura del *Dashboard* de NetSight 18

Fig. 5: Logo de Arista 22

Fig. 6: Esquema de CloudVision 22

Fig. 7: Captura del *Dashboard* de CloudVision 23

Fig. 8: Logo de Cisco 25

Fig. 9: Captura del *Dashboard* del APIC 26

Fig. 10: Logo de Aerohive Networks 28

Fig. 11: Captura del *Dashboard* de HiveManger 29

Fig. 12: Diferencias entre la arquitectura tradicional y la SDN 34

Fig. 13: Esquema de una arquitectura de red en modo ACI 36

Fig. 14: Estructura de un controlador APIC 37

Fig. 15: Estructura de un *Tenant* con un *Networking Tenant* 38

Fig. 16: Estructura de un *Tenant* con *Networking Tenant* y *Policy Tenant* 40

Fig. 17: Relación entre los elementos de un *Policy Tenant* 41

Fig. 18: Modelo de objetos de ACI 42

Fig. 19: Jerarquía de objetos en ACI 43

Fig. 20: MIT del objeto *Tenant* 44

Fig. 21: Diagrama de ejemplo de dependencias entre objetos 45

Fig. 22: Petición de inicio de sesión en Postman 49

Fig. 23: Respuesta de la petición de inicio de sesión en Postman 50

Fig. 24: Respuesta de la petición de lectura de perfiles de aplicación en Postman 50

Fig. 25: Contenido del archivo credentials.py 51

Fig. 26: Ejemplo de *script* para crear un objeto *Tenant* usando ACI Toolkit 52

Fig. 27: Ejemplo de script para crear un objeto *Tenant* usando Cobra SDK 54

Fig. 28: Visualización de los *Tenant* del APIC 55

Fig. 29: Primer diseño gráfico de las pantallas 61

Fig. 30: Código *Kv* de la pantalla de login 62

Fig. 31: Pantalla final de login 65

Fig. 32: Código *Kv* de la pantalla de creación de *Tenant* 66

Fig. 33: Pantalla final de creación de *Tenant* 67

Fig. 34: Código *Kv* del pop-up 67

Fig. 35: Pop-up de alerta 68

Fig. 36: Importaciones o Módulos 68

Fig. 37: Clases de objetos referenciados en el código *Kv* 69

Fig. 38: Clase *MyScreenManager* 70

Fig. 39: Clase *MainApp* y *main* 71

Índice de tablas

Tabla 1: Comparativa de fabricantes 31

Tabla 2: Tabla de referencia para nombrar MO 47

Tabla 3: Tipos de variables..... 58

Glosario de términos

Para ayudar a la comprensión de este proyecto y dotarlo de contexto, se encuentran recogidas en esta sección una serie de términos acerca de los que se habla sin profundizar exhaustivamente:

- **VLAN:** Acrónimo de virtual LAN, es un método para crear redes lógicas independientes dentro de una misma red física.

- **QoS (*Quality of Service*):** Es el rendimiento promedio de una red, en especial el rendimiento visto por los usuarios de la red.

- ***Dashboard*:** Interfaz desde la que un usuario puede administrar el software que se está usando.

- ***Cloud Computing*:** Paradigma que permite ofrecer servicios de computación a través de Internet.

- **IT:** Siglas de Tecnologías de la información.

- **SO:** Siglas de Sistema Operativo.

- **OVSDB:** Protocolo de gestión en entornos que utilizan redes de tipo SDN.

- **CLI (*Command-Line Interface*):** Es un método que permite a los usuarios dar instrucciones a un programa mediante una línea de texto simple.

- ***Data Centers*:** Espacio donde se concentran los recursos necesarios para el procesamiento de la información de una organización.

- ***Access Point*:** Es un dispositivo de red que interconecta equipos de comunicación inalámbricos para formar una red inalámbrica que se comunica con dispositivos móviles o tarjetas de red inalámbricas.

- **SSID:** Es una secuencia incluida en todos los paquetes de una red inalámbrica para identificarlos como parte de esa red.

- **JSON:** Acrónimo de *JavaScript Object Notation*, es un formato de texto ligero para el intercambio de datos.

- **XML:** Acrónimo de *eXtensible Markup Language*, es un metalenguaje utilizado para almacenar e intercambiar datos de forma legible.

- ***Endpoint*:** Dispositivo informático remoto que se comunica con la red a la que está conectado.

- ***Access List*:** Es una forma de determinar los permisos de acceso apropiados a un determinado objeto, permitiendo así controlar el flujo del tráfico dentro de una red.

- **BBDD:** Siglas de Bases de Datos.

- **GUI:** Siglas de Interfaz Gráfica de Usuario.

- **Script:** Programa almacenado en un fichero de texto plano.

- ***Framework*:** Es una estructura conceptual y tecnológica de asistencia definida con artefactos o módulos concretos de software, que sirve de base para la organización y desarrollo de software.

- ***Spanning-Tree*:** Protocolo de capa 2 que sirve para evitar bucles en una topología de red cuando existen enlaces redundantes.

- ***Multitouch:*** Tecnología que permite a una superficie reconocer la presencia de múltiples puntos de contacto de manera simultánea.

1. Objetivos

1.1. Introducción

A lo largo de los años que he cursado el Grado de Ingeniería Informática, a los alumnos nos han enseñado múltiples herramientas para desarrollar aplicaciones, centrándose casi exclusivamente en el ámbito de la programación. No obstante, también nos inculcaron nociones de otros campos del sector, como las redes, los sistemas operativos y seguridad, entre otros.

Gracias al tutor de este proyecto, Dr. Léonard Janer García, encontré trabajo en Aton Systems, una empresa del sector de los servicios TIC fundada en 2001 especializada en redes, comunicaciones, seguridad y arquitecturas convergentes.

En Aton Systems son especialistas en el campo de las redes, en el cuál yo no tenía mucha experiencia, pero que gracias a ellos he ganado una gran cantidad de conocimientos que me hubiese costado ganar de otra manera.

Y gracias a estos conocimientos adquiridos, nace la idea de aprender a combinar los campos de la programación y las redes, un concepto interesante que permite desde realizar cambios en la configuración de red mediante un programa hasta la automatización de procesos de gestión.

1.2. Propósito

El objetivo de este proyecto es el estudio y análisis de diferentes fabricantes de equipos de red, y hacer uso de los recursos que ofrecen para crear una herramienta que simplifique aspectos la gestión de red para hacerlo más accesible e intuitivo, usando para ello la programación.

1.3. Finalidad

Aprender sobre nuevos campos de la informática, como es la gestión de redes, y combinarlos con conceptos ya conocidos, como la programación, mediante el uso de APIs que proporciona el fabricante para poder crear herramientas que faciliten y simplifiquen la gestión de una red.

1.4. Objeto

El proyecto realizado incluirá la memoria del trabajo que contiene el análisis de los diferentes fabricantes, así como la información relacionada con el estudio en profundidad del fabricante seleccionado para hacer las pruebas. También se adjuntarán todos los scripts que se hayan usado, para poder replicar las pruebas realizadas.

2. Estado del arte de los fabricantes

En este apartado se valorarán las mejores opciones que hay en el mercado de los diferentes fabricantes con los que se trabaja en Aton Systems que ofrecen soluciones para la gestión de red, así como cuáles son las soluciones que proporcionan y cómo funcionan.

Al hacer esta valoración, se escogerá uno de los fabricantes y se procederá a trabajar con la solución que nos facilite, usando las herramientas que aporte su solución para desarrollar los scripts que permitirán la gestión de la red.

Para cada fabricante se expondrá una breve descripción acerca del mismo, así como una explicación de la herramienta de gestión que ofrecen. A esto sigue una descripción de sus características, y finaliza con una lista de las ventajas que tiene su uso.

Para justificar la elección de los fabricantes a analizar, se ha utilizado un diagrama de Gartner realizado sobre los fabricantes con mejor solución para la gestión de red, donde aparecen tres de los cuatro fabricantes seleccionados. Un diagrama de Gartner proporciona un posicionamiento competitivo gráfico de cuatro tipos de proveedores de tecnología en mercados donde el crecimiento aumenta de manera constante y la diferenciación de los fabricantes es importante. Estos tipos son:

- ***Leaders*:** Desarrollan de acuerdo a su visión de mercado y están bien posicionados para el mañana, es decir, entienden que dirección puede tomar el mercado y pueden actuar en consecuencia.
- ***Visionaries*:** Entienden hacia dónde está orientado el mercado y tienen una visión acerca de los cambios en las reglas de mercado, pero no ofrecen las mejores soluciones.
- ***Niche Players*:** Se centran en pequeños segmentos del mercado en cuestión, pero sin innovar ni superar otras soluciones.
- ***Challengers*:** Proporcionan una solución competente, pero no demuestran un entendimiento acerca de la dirección que puede tomar el mercado.

Fig. 1: Diagrama de Gartner del estado de los fabricantes en el mercado

Como se puede apreciar en el diagrama de Gartner [1] mostrado, aparece Cisco como *Leaders* en el mercado, seguido por otros dos fabricantes con los que Aton Systems trabaja de manera asidua, Extreme y Aerohive, situados como *Visionaries* en cuanto a fabricantes que ofrecen mejores soluciones de gestión de red. Debido a esto se puede extrapolar que Cisco ofrece la mejor solución para la gestión de red, ya que está clasificado en el tipo *Leaders*, asociado a aquellos que ofrecen una buena solución y que están preparados para afrontar los cambios en el paradigma de mercado.

A continuación, se procederá a analizar los cuatro fabricantes escogidos para su estudio, así como las herramientas que ofrecen para gestionar la red.

2.1. Extreme Networks

Extreme Networks [2] es un fabricante que proporciona soluciones de red basadas en software para grandes empresas y proveedores de servicios. Extreme diseña, desarrolla y fabrica equipos de infraestructura de redes cableadas e inalámbricas y desarrolla el software que permite la administración de redes, políticas, análisis, seguridad y controles de acceso a las mismas, brindando así una solución integral de extremo a extremo desde el data center hasta los *endpoints*. Este software es conocido como NetSight.

Fig. 2: Logo de Extreme Networks

2.1.1. ¿Qué es NetSight?

NetSight es un sistema de gestión que proporciona visibilidad y control de dispositivos tanto cableados como inalámbricos. La inteligencia, la automatización y la integración de su software de gestión permiten que el personal de IT optimice la eficiencia de las operaciones de red y reduzca el coste total de propiedad.

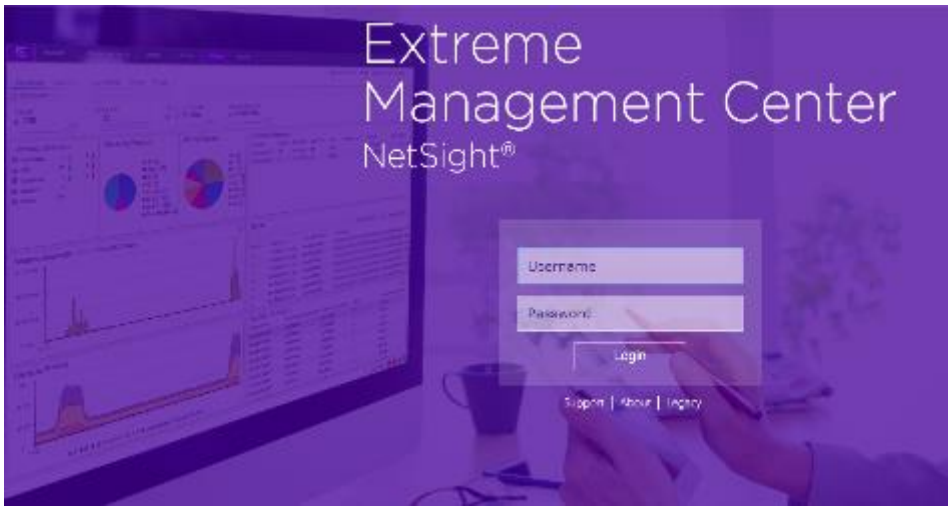

Fig. 3: Pantalla de login de NetSight

NetSight proporciona visibilidad centralizada y control individualizado de los recursos de la red empresarial gestionada de principio a fin. NetSight es distintivo para la granularidad que va más allá de los puertos y las VLAN hasta usuarios individuales, aplicaciones y protocolos. Independientemente de la cantidad de movimientos, adiciones o cambios que se produzcan en su entorno, NetSight mantiene todo a la vista y bajo control mediante controles de acceso basados en roles. Un clic puede equivaler a muchas acciones cuando se gestiona una red. NetSight puede incluso administrar más allá del switching, el enrutamiento y el hardware inalámbrico de Extreme Networks para brindar control basado en estándares de los equipos de red de otros proveedores.

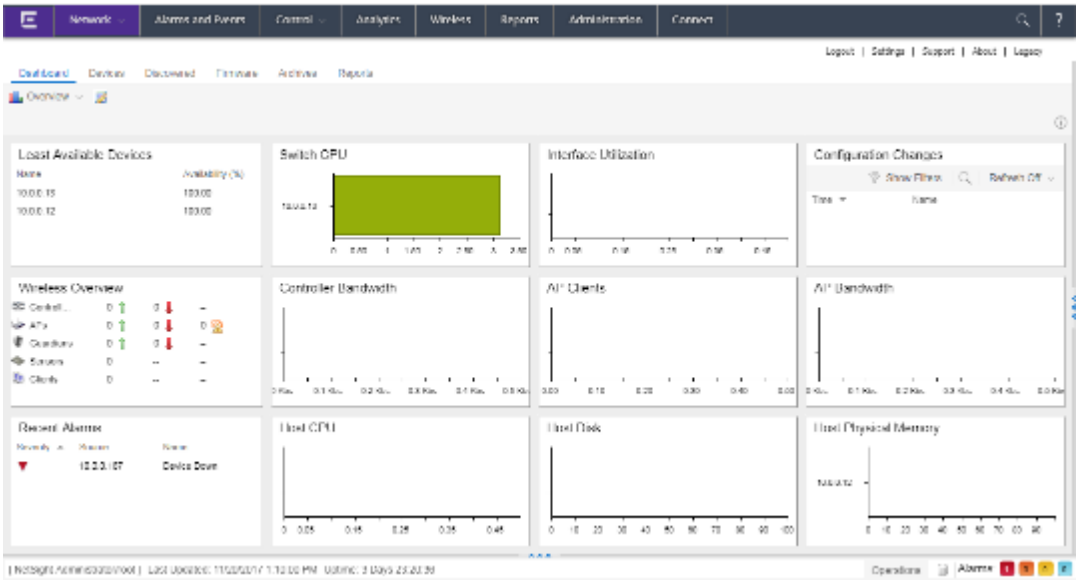

Fig. 4: Captura del *Dashboard* de NetSight

2.1.2. ¿Qué ofrece NetSight?

- **Gestión integrada de dispositivos cableados o inalámbricos:** NetSight proporciona una administración uniforme tanto de la red cableada como inalámbrica desde una única interfaz.

- **Panel de control e informes:** NetSight proporciona una interfaz gráfica de usuario para la recopilación de datos de red y la generación de informes de análisis de red.

- **API abierta:** NetSight proporciona una interfaz bidireccional abierta y segura para la integración de aplicaciones de terceros y el control de las implementaciones de redes definidas por software (SDN).

- **Administración de inventario de red:** NetSight aporta herramientas para crear una configuración de red avanzada y administración de cambios para la infraestructura cableada e inalámbrica.

- **Recolección de datos históricos y análisis de tendencias:** NetSight brinda la capacidad de evaluar datos para comprender cómo se usa la red hasta la capa 4. Con la adición de Purview, esto amplía el análisis detallado hasta la capa de aplicación (capa 7) para una visibilidad detallada de una aplicación en toda la red.

- **Red basada en políticas:** NetSight permite la creación centralizada de políticas que siguen a los usuarios y dispositivos a través de la red. Estas no están vinculadas a la red física y pueden cambiar en función del usuario, dispositivo, ubicación y tipo de conexión.

- **Identidad y control de acceso:** NetSight brinda visibilidad y control al usuario y al dispositivo.

- **Gestión inteligente de alarmas:** NetSight permite la programación de alarmas y eventos que alertan al personal de IT cuando se producen problemas dentro de la red, los cuales se borran automáticamente cuando el problema ya no existe y permite configurar entornos de prevención para que las advertencias se envíen antes de una falla real.

- **Solución de problemas:** NetSight elimina las conjeturas y facilita la resolución de problemas de los usuarios. Simplemente basta con ingresar el nombre de usuario para descubrir dónde se encuentra el usuario, dónde está conectado a la red, el tipo de dispositivo que está utilizando, y otra gran cantidad de datos.

- **Integración de red de terceros:** NetSight permite administrar dispositivos de terceros para proporcionar una imagen completa de toda la infraestructura en un entorno de red heterogéneo.

2.1.3. ¿Qué ventajas tiene usar NetSight?

La alineación del negocio

- Transformar datos de red complejos en información procesable centrada en el negocio.

- Centralizar y simplificar la definición, administración y aplicación de políticas como el acceso de invitados o dispositivos personales.

- Integración fácil con aplicaciones comerciales con redes definidas por software para lograr eficiencia operativa.

Eficiencia operacional

- Reducir el esfuerzo administrativo del personal de IT mediante la automatización de tareas rutinarias y la administración del Dashboard.

- Portal web para la gestión e integración de redes cableadas e inalámbricas.

- Fácil creación de políticas en toda la red para QoS, ancho de banda, etc.

- Permite su uso con la comodidad de un Smartphone o Tablet.

Seguridad.

- Protege los datos corporativos mediante monitoreo centralizado, control y respuesta en tiempo real.

- Mejora las inversiones existentes en seguridad de red.

- Preserva la integridad de la red LAN / WLAN con políticas unificadas.

Servicio y soporte

- Líderes de la industria en resolución de incidencias con la primera llamada y en los cuestionarios de satisfacción de los clientes.

- Servicios personalizados, que incluyen encuestas del portal web, diseño de redes, instalación y formación.

2.2. Arista

Arista [3] es un fabricante que diseña y vende switches de red multicapa pensados para soluciones de grandes centros de datos, empresas de *Cloud Computing*, computación de alto rendimiento y entornos de alta frecuencia de datos. Además, Arista ofrece soluciones para crear infraestructuras de IT y red pensando en nuevos entornos, ofreciendo una relación coste/rendimiento que permite infraestructuras rentables, sin bloqueos, con picos de rendimiento controlables y baja latencia, además de un sistema operativo robusto que permite, sin dejar de dar servicio la actualización del SO y auto-reparación de procesos del sistema que presenten algunos problemas. Arista también otorga una herramienta de gestión que permite gestionar tanto redes virtuales como físicas, llamada CloudVision.

Fig. 5: Logo de Arista

2.2.1. ¿Qué es CloudVision?

CloudVision es una solución integral que permite gestionar la carga de trabajo en toda una red, así como la automatización del flujo de trabajo, y tener visibilidad en tiempo real de todas las operaciones que tienen lugar en la red. CloudVision está basado en el SO propio de Arista (EOS), que proporciona servicios de infraestructura para transmitir, compartir y agregar los estados de funcionamiento de los equipos físicos que están conectados en la red. CloudVision ha sido diseñado para simplificar la integración de sistemas de orquestación, controladores de superposición, y otros servicios de red mediante la abstracción de la red física subyacente.

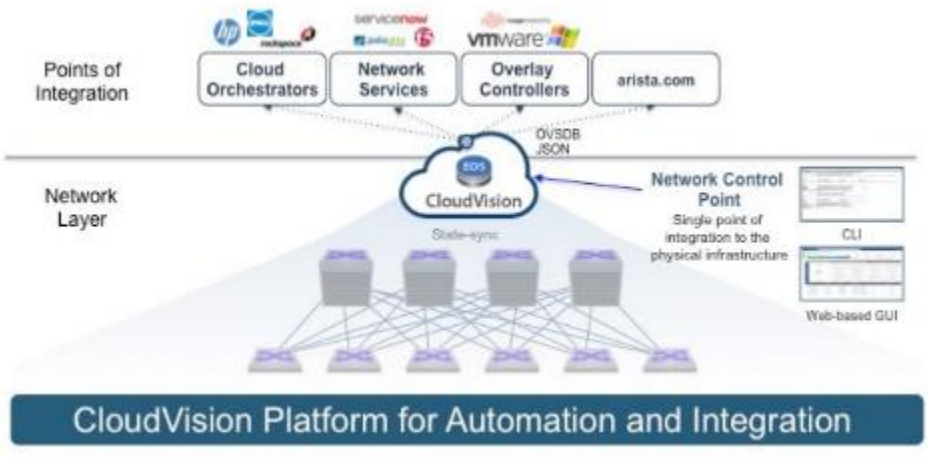

Fig. 6: Esquema de CloudVision

2.2.2. ¿Qué ofrece CloudVision?

CloudVision ofrece una interfaz web que permite visualizar los flujos de trabajo, un repositorio del estado del servidor y un motor de análisis para mantener el estado histórico de la red. La automatización del flujo de trabajo en CloudVision permite a los operadores ejecutar tareas comunes de implementación y configuración desde un solo punto de contacto visual. El portal web amplía la automatización del aprovisionamiento inicial del dispositivo para incluir la automatización de controles de cambio continuos durante el ciclo de vida operacional de los dispositivos de red. También incluye un modelo de configuración jerárquico que aprovecha la modularidad de la configuración y la reutilización de la misma para simplificar la gestión de la configuración en la red.

Fig. 7: Captura del *Dashboard* de CloudVision

La base de datos CloudVision también guarda datos históricos, incluido un historial de estado de red, configuración y versiones de software. Este estado se puede usar para entender el funcionamiento de toda la red para la verificación de control de la misma antes y después del cambio, lo que ayuda a simplificar el proceso de gestión del cambio y reducir los tiempos de ventana de mantenimiento.

La infraestructura de gestión de control de cambios permite realizar tareas avanzadas pre-integradas, así como actualizaciones de sistemas inteligentes, y la reversión se puede organizar y ejecutar en toda la red como parte de la automatización del control de cambios del portal web.

2.2.3. ¿Qué ventajas tiene usar CloudVision?

- Representación centralizada del estado de la red distribuida, que permite un único punto de integración y visibilidad y análisis en toda la red.

- Compatibilidad independiente del controlador para la organización física y virtual de la carga de trabajo a través de APIs abiertas como OVSDB, JSON y complementos de Openstack.

- Administración de configuración y administración de cambio en toda la red, incluidas las actualizaciones automáticas, la reversión de red y gráficos para el entendimiento de la red.

- Dashboard para configurar la seguridad, hacer auditorías y configurar la administración de parches de software.

- Transmisión de estado en tiempo real para telemetría y análisis de redes.

- Repositorio de estado, motores de análisis y aplicaciones de telemetría para proporcionar un gran nivel de granularidad en la supervisión en tiempo real y el estado histórico de la red para la solución de problemas.

2.3. Cisco

Cisco [4] es un fabricante que se dedica principalmente a la fabricación, venta, mantenimiento y consultoría de equipos de red. Además de desarrollar el hardware de sus equipos, Cisco también se ocupa de desarrollar su propio software de configuración y gestión de los mismos, llamado IOS.

IOS es un paquete de funciones de enrutamiento, conmutación, trabajo de Internet y telecomunicaciones que se integra estrechamente con un sistema operativo multitarea. Este SO funciona mediante un CLI (*Command-Line Interface*) que permite realizar todas las funciones de configuración de sus equipos.

Cisco también pone a disposición del usuario una herramienta de gestión de red llamada ACI (*Application Centric Infrastructure*), una solución basada en SDN, concepto del que se hablará más adelante.

Fig. 8: Logo de Cisco

2.3.1. ¿Qué es ACI?

ACI es una solución de SDN (*Software Defined Network*) que funciona de manera un poco diferente que otros competidores. En lugar de abstraer la inteligencia del hardware de red, el hardware de red de ACI está equipado con un nuevo tipo de inteligencia. Los administradores pueden crear, personalizar y duplicar políticas de red. Luego pueden instruir a la infraestructura para que siga estas reglas para aplicaciones específicas. De hecho, crea los mismos resultados que otras soluciones SDN como VMware NSX, pero la inteligencia está en el software dentro de la infraestructura en lugar de en una aplicación independiente.

ACI también incluye APIC (*Application Policy Infrastructure Controller*). A través del APIC, los administradores tienen acceso centralizado a la administración de su red. Pueden ajustar las políticas, ver el estado de la red e implementar funciones avanzadas como la QoS (*Quality of Service*) y la seguridad de múltiples usuarios. El APIC ofrece una interfaz gráfica que permite realizar toda la gestión de los recursos de la red.

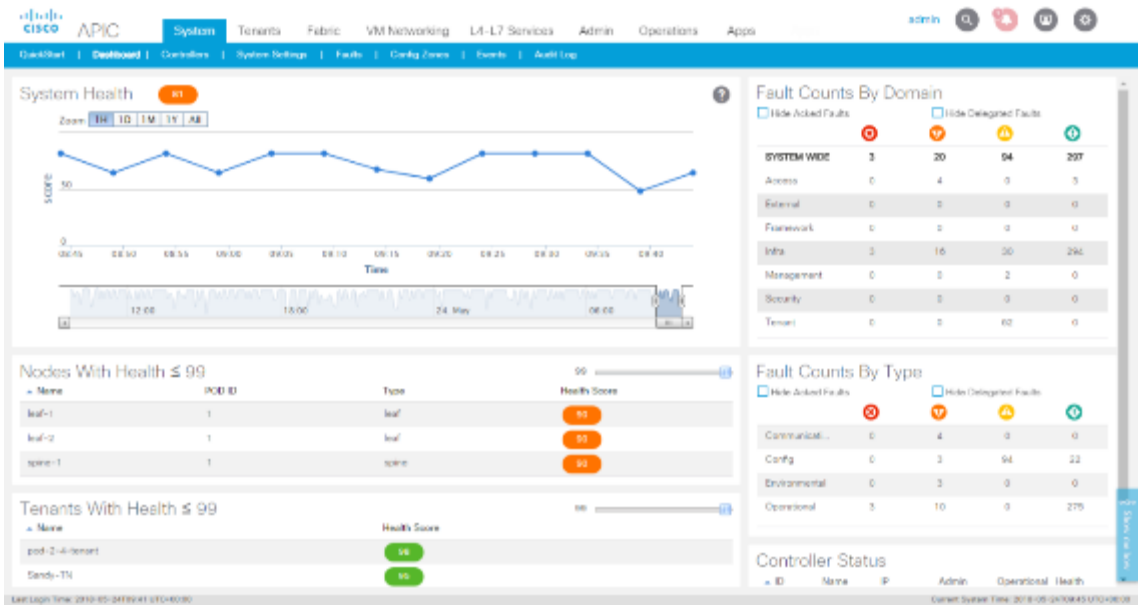

Fig. 9: Captura del *Dashboard* del APIC

2.3.2. ¿Qué ofrece ACI?

Cisco ACI ofrece un modelo operativo de transformación para las aplicaciones en la nube y los *Data Centers* de la última generación.

En el entorno de ACI, son las aplicaciones las que determinan el comportamiento de la red y no al contrario. Las descripciones y los requisitos predefinidos de las aplicaciones automatizan el aprovisionamiento de la red, los servicios de aplicaciones, las políticas de seguridad, las subredes de *Tenants* y la ubicación de la carga de trabajo. Al automatizar el aprovisionamiento de la red de aplicaciones completa, ACI contribuye a reducir los costes de IT, minimizar los errores, acelerar la implementación y a otorgar mayor agilidad a los negocios.

El modelo de ACI usa un enfoque diseñado desde la base para cubrir las exigencias de los sectores emergentes, al mismo tiempo que se mantiene una ruta de migración para las arquitecturas ya instaladas. Esto permite a las aplicaciones empresariales tradicionales y a las desarrolladas internamente ejecutarse en paralelo en una infraestructura de red que les dé cabida de un modo dinámico y escalable. Las topologías lógicas y las políticas de redes, que, tradicionalmente, han dictado el diseño

de las aplicaciones, se emplean ahora en función de las necesidades de la aplicación en cuestión. Dicho enfoque está diseñado para admitir el paso a la automatización de la administración, a las políticas definidas según el programa y a las cargas de trabajo dinámicas en cualquier dispositivo, y desde cualquier lugar.

2.3.3. ¿Qué ventajas tiene usar ACI?

- Arquitectura de sistemas que permite una vista integral de las aplicaciones, además de una vista centralizada de las mismas, y la supervisión del estado de las aplicaciones en tiempo real en todos los entornos, tanto virtuales como físicos.

- Plataforma común para administrar entornos basados en la nube, tanto físicos como virtuales.

- Rendimiento escalable que combina la flexibilidad del software con muy buenas prestaciones de hardware.

- Funcionamiento sencillo con modelos de funcionamiento, administración y políticas comunes en todos los recursos de seguridad, de red y de aplicaciones.

- APIs abiertas, así como los estándares y los elementos de código abiertos, otorgan mayor flexibilidad a los equipos de operaciones y desarrollo.

2.4. Aerohive Networks

Aerohive Networks [5] es un fabricante de equipos de red y orientado a los servicios de Cloud que ha desarrollado una arquitectura LAN sin controlador llamada *Cooperative Control*, que proporciona todo el rendimiento, disponibilidad, administración, movilidad y seguridad necesarias en una implementación de red. Esta arquitectura también permite que las redes inalámbricas sean seguras, escalables, de alto rendimiento, manejables y rentables para diferentes sectores de la industria como la educación, empresas distribuidas, comercios minoristas, salud y gubernamentales, tanto local como estatal.

Aerohive también desarrolla su propio SO, HiveOS, para sus *Access Points* y switches, un sistema de administración y gestión habilitado para el Cloud, HiveManager, herramientas de planificación Wi-Fi, aplicaciones y herramientas de administración de red para sus dispositivos.

También pone a disposición diferentes API REST abiertas para su plataforma de Cloud, *Aerohive Cloud Services*, para que los desarrolladores creen soluciones de administración y análisis de Wi-Fi.

Fig. 10: Logo de Aerohive Networks

2.4.1. ¿Qué es HiveManager?

HiveManager ha sido diseñado por Aerohive como sistema de administración y gestión de redes para la nube desde cero. El resultado es una plataforma de nueva generación para redes inalámbricas en la nube, que reduce drásticamente la complejidad operativa y el coste para los clientes y socios de la compañía.

HiveManager simplifica de manera radical la administración unificada de redes tanto cableadas como Wi-Fi, combinando flujos de trabajo de configuración optimizados, monitoreo de eventos y clientes en tiempo real, solución de problemas simplificada e integraciones de API.

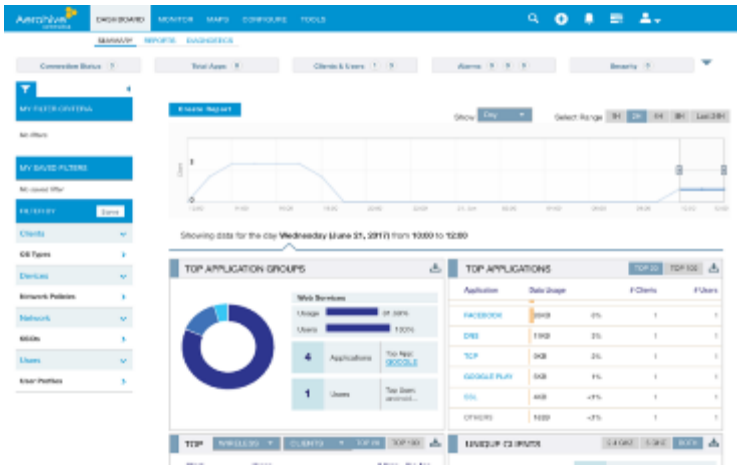

Fig. 11: Captura del *Dashboard* de HiveManger

2.4.2. ¿Qué ofrece HiveManager?

HiveManager ofrece un Dashboard intuitivo y visual con filtros contextuales personalizables para comprobar el estado de la red, así como los dispositivos que la conforman. También tiene una herramienta que permite el control de las aplicaciones que usan los dispositivos conectados a la red, y que recursos usan de la misma, además de mostrar qué usuarios usan qué aplicaciones.

HiveManager ofrece también herramientas que permiten monitorear lo que sucede y realizar informes sobre el estado de la red. HiveManager tiene una arquitectura que permite gestionar tanto redes cableadas como inalámbricas mediante políticas de red robustas, incluso si los dispositivos son de otro fabricante.

2.4.3. ¿Qué ventajas tiene usar HiveManager?

Gestión de red flexible

- Despliegue de Cloud público o privado.
- Experiencia hacia el usuario optimizada y simplificada.
- Solución de problemas simplificada.

- Inteligencia operacional.

Despliegue simplificado

- Flujo de trabajo guiado para la implementación de políticas de red.
- Asistencia de configuración de HiveManager.
- Aprovisionamiento automático para dispositivos con políticas de red y actualizaciones de firmware.

Configuración optimizada

- Plantillas preconfiguradas para switches y *Access Points* con una vista centralizada de todos los objetos de configuración.
- Edición en masa de las propiedades del dispositivo.
- La capacidad de hacer copias de seguridad y restaurar objetos, inventarios y configuraciones.
- CLI que permite acceder a un dispositivo de la red.

Gestión de políticas de red centralizada

- Clasificación del dispositivo por ubicación, tipo de sistema operativo y dirección MAC.
- Definición de la aplicación cliente y de múltiples perfiles de usuario para cada SSID de cada *Access Point*.
- Políticas de firewall basadas en tiempo y QoS.
- Políticas de firewall al nivel de las capas de red, MAC y aplicación.

Dashboard orientado al usuario

- Panel de flujo de tiempo deslizable que ofrece una vista histórica de cualquier política de red, *Access Point*, dispositivo cliente, usuario o aplicación.

- Función de búsqueda global por política de red, dirección MAC, número de serie del dispositivo, nombre de usuario o aplicación.

- Filtros para organizar la información mostrada acerca de un dispositivo cliente personalizables con ubicación, SSID, política de red, perfil de usuario y SO del dispositivo cliente.

2.5. Elección de fabricante

A modo de recapitulación, se expone en la siguiente tabla una lista de ventajas y desventajas de cada fabricante para ayudar a decidir cuál es la mejor decisión.

	Ventajas	Desventajas
Extreme	• NetSight permite la integración de dispositivos de terceros. • Permite generar informes con análisis de la red y de cómo se utiliza. • Sirve para redes cableadas o inalámbricas.	• Documentación escasa respecto a la API. • El *Dashboard* de NetSight es complicado de usar.
Arista	• *Dashboard* bastante intuitivo. • Ofrece múltiples APIs con las que interactuar con CloudVision.	• Orientado mayormente a estructuras de red basadas en la nube.
Cisco	• Gran cantidad de información acerca del uso de sus APIs, ofreciendo incluso cursos formativos gratuitos. • Proporciona entornos de prueba sobre los que probar su solución y funcionalidades. • Sirve para redes cableadas, inalámbricas y entornos basados en la nube.	• Estructura conceptual de la red diferenciada del resto, algo complicada de entender.
Aerohive	• Permite configurar en masa elementos de la red. • *Dashboard* intuitivo. • Proporciona plantillas preconfiguradas.	• Poca información acerca de sus APIs, que ofrecen pocas opciones de configuración.

Tabla 1: Comparativa de fabricantes

Después del estudio de los diferentes fabricantes, se ha decidido trabajar con la solución que proporciona Cisco, ya que se ha considerado la mejor opción debido a la gran cantidad de información que proporciona sobre su herramienta de gestión de red, así como de sus APIs, y cómo se puede interactuar con ellas.

También se ha tenido en cuenta el hecho de que se puede trabajar con entornos de prueba virtuales instalados en la nube que facilitan todo el proceso de realización de pruebas y comprobación de resultados, mientras que con otros fabricantes necesitas equipos de su marca o múltiples máquinas virtuales para realizar dichas pruebas.

3. Introducción a las APIs

Para poder hablar propiamente de qué es una API y explicar cómo funcionan, antes se debe aclarar por qué una API es útil. Igual que una interfaz gráfica facilita a un usuario el uso de un programa, una API facilita a un desarrollador el proceso de crear una aplicación que sea capaz de interactuar con software de terceros, proporcionando al desarrollador una serie de funciones que le permiten hacer uso de funciones de ese software externo sin entender exactamente cómo dicho software funciona.

De aquí se extrapola que una API (*Application Programming Interface*) es un conjunto de subrutinas, protocolos y herramientas para desarrollar aplicaciones. En esencia, es un conjunto de métodos que permiten comunicarse entre diferentes aplicaciones. Una API puede servir para desarrollar una aplicación web, pero también para interactuar con sistema operativo, con una base de datos, con librerías de software o incluso con componentes del hardware.

A pesar de los diferentes tipos de APIs que hay, uno de los más usados es la Web API. En el contexto del desarrollo web, las APIs se definen como un conjunto de peticiones HTTP, que devuelven una respuesta con los datos preguntados, normalmente en los formatos JSON o XML.

Cuando pasamos al contexto de la gestión de redes, es necesario hablar de redes tipo SDN, ya que la arquitectura tradicional de cómo construir una red hace que dicha red quede descentralizada, lo que la hace más compleja de gestionar. Debido al auge provocado por la telefonía móvil, virtualización de servidores y avances en los servicios de Cloud, ha sido necesario repensar la estructura que forma una arquitectura de red.

El modelo clásico presenta una arquitectura jerárquica, construida con diferentes rúters o switches en forma de árbol. Este diseño funcionaba en el mercado en que la computación cliente-servidor era dominante, pero una red tan estática es incapaz de desempeñar las funciones de computación dinámica y almacenamiento que requieren empresas como *Data Centers* o campus universitarios. Esto ha provocado que se idee una nueva arquitectura de red, conocida como SDN.

Una red SDN (*Software-Defined Network*) está pensada para facilitar la gestión de red, sirviéndose de un conjunto de técnicas relacionadas con el área de las redes computacionales, cuyo objetivo es facilitar la implementación e implantación de servicios de red de una manera dinámica y escalable, evitando así al administrador de la red tener que gestionar dichos servicios a bajo nivel, que se consiguió mediante la separación de los planos de control y datos, ya que la estrecha integración entre dichos planos para realizar depuraciones de los problemas de configuración o controlar el comportamiento del enrutamiento eran unas tareas desafiantes y complicadas.

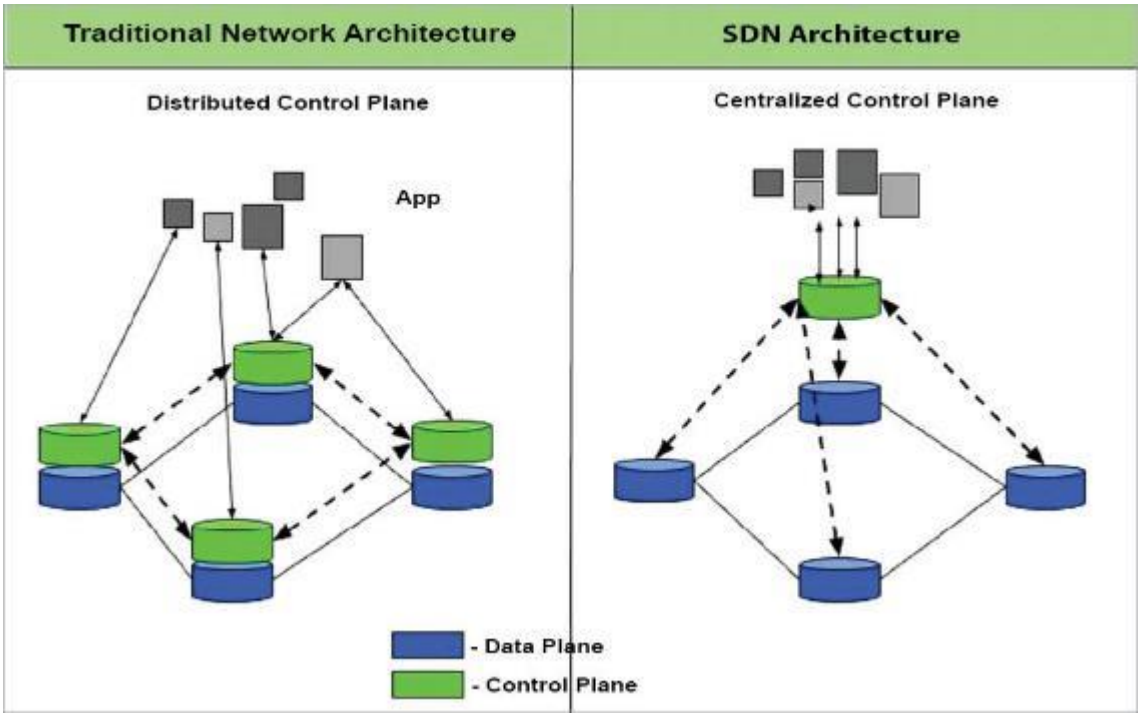

Fig. 12: Diferencias entre la arquitectura tradicional y la SDN

No obstante, este modelo no es perfecto, y entre sus desventajas destaca la seguridad, ya que aunque existen enfoques competentes, todavía está en tela de juicio, ya que dichos enfoques no convergen en una idea común acerca de cómo hacer una red segura.

4. Cisco: Nexus, ACI y sus APICs

Cisco, una de las mayores empresas dedicadas al mundo de las redes, tiene una solución de SDN, llamada ACI (*Application Centered Infrastructure*).

Usando el modo de operaciones NX-OS, los switches de Cisco utilizan un sistema operativo tradicional mejorado para proporcionar una plataforma que permite implementar una red actual, a la vez que mantiene la capacidad de aprovechar los protocolos y tecnologías de nueva generación. No obstante, este modo se sigue teniendo que gestionar a bajo nivel, lo que implica un cierto grado de conocimientos y dificultad.

De diferente manera, los switches se pueden operar en modo ACI. En este modo, toda la infraestructura es administrada de manera central por un grupo de controladores llamados APICs (*Application Policy Infrastructure Controller*).

Si bien los modos NX-OS y ACI tienen mapas de ruta y conjuntos de características independientes, la plataforma de hardware común en que se basan les brinda flexibilidad, opciones y valor. Este hardware común es la serie Nexus.

En modo ACI, los switches pueden funcionar como Spine o como Leaf, donde los Spine sirven para agregar switches Leaf, y los Leaf se utilizan como dispositivos de acceso.

Cuando un switch opera en modo ACI, se programa a través de un motor de políticas situado en el controlador APIC, a diferencia de hacerlo vía CLI como se hace en los que funcionan en modo NX-OS. Este controlador es una parte integrada de la red, y contiene perfiles que contienen las políticas para programar los switches de forma centralizada. La configuración de los equipos se mantiene en el APIC utilizando un esquema orientado a objetos, representado en XML o JSON, y se almacena en un perfil para implementar políticas impulsadas por la aplicación, basadas en la red y en la seguridad.

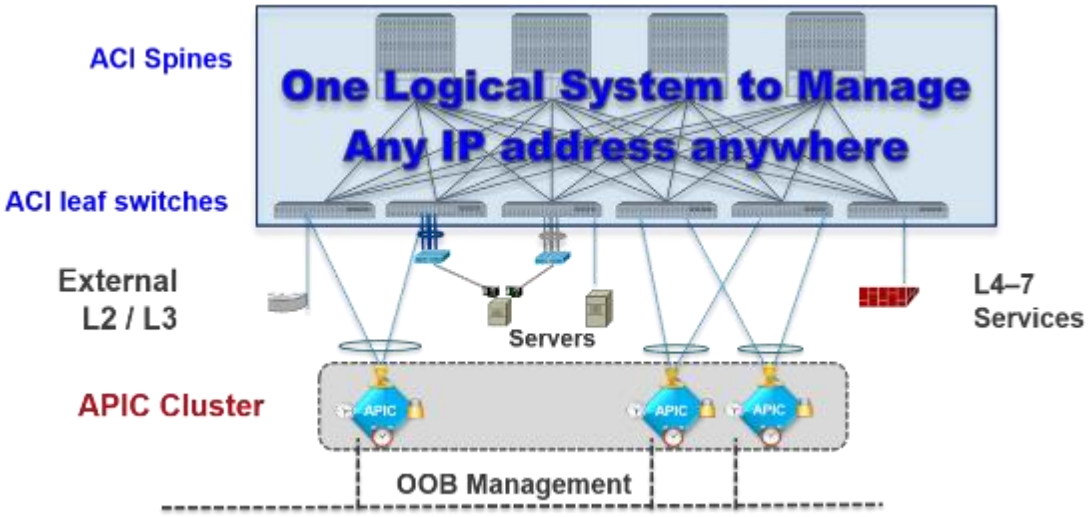

Fig. 13: Esquema de una arquitectura de red en modo ACI

Una arquitectura de red en modo ACI requiere los siguientes elementos:

- ***Leaf switches***, que proporcionan conectividad entre los dispositivos conectados. Sirven como puertas de enlace de capa 3, puntos de cumplimiento de políticas y puertas de enlace para redes externas.

- ***Border leaf switches***, que son aquellos leaf switches que se conecten a un dispositivo de red externo a la arquitectura ACI, como un firewall, un balanceador de carga, un router o switch que no opere en modo ACI.

- ***Spine switches***, cuya función principal es reenviar el tráfico entre dos leaf swithces.

- **Controladores APIC**, que proporcionan el punto de administración centralizado para configurar la arquitectura de red y monitorizan el estado de la red.

4.1. Terminología ACI

Previamente se ha mencionado que los APIC funcionan como un motor de políticas orientado a objetos. Esto permite que el administrador de la red defina los estados deseados de entramado, pero dejando la implementación al propio controlador APIC. A medida que las cargas de trabajo se mueven, el controlador reconfigura la infraestructura subyacente para garantizar que las políticas necesarias sigan vigentes para los *endpoints*. La rama del modelo de objetos en el que se definen estas políticas es el objeto *Tenant*.

4.1.1. ACI Tenants

Los *Tenants* son un objeto de nivel superior que ayuda a identificar y separar el control administrativo, los dominios de red y las políticas de aplicación. Los objetos que están un subnivel por debajo de un *Tenant* se pueden agrupar en dos categorías básicas: *Networking Tenants* y *Policy Tenants*. Estas dos categorías tienen relaciones inherentes, pero puede ser útil discutirlas por separado.

ACI tiene un *Tenant* especial llamado "*common*" que tiene habilidades únicas para compartir fácilmente sus recursos con otros *Tenants*. El *Tenant* "*common*" está diseñado para proporcionar recursos compartidos a todo el tejido de ACI, como DNS, servicios de autenticación, herramientas de seguridad, etc. También puede definir todos los objetos, contratos y filtros de *Tenant Networking* en el *Tenant* "*common*" y asociarlos con las aplicaciones en otros *Tenants*.

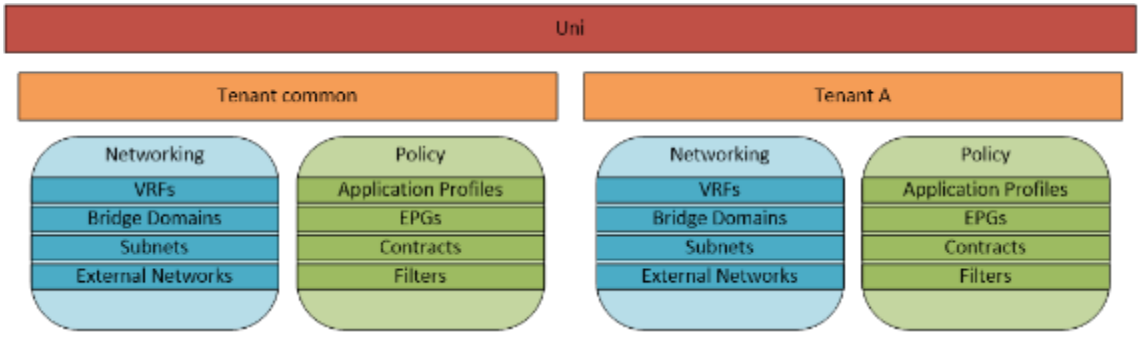

Fig. 14: Estructura de un controlador APIC

4.1.2. Networking Tenant

Los objetos *Networking Tenant* son similares a topologías de red con las que los ingenieros ya están familiarizados, y brindan conectividad de Capa 2 y Capa 3 entre distintos hosts. Los *Networking Tenants* están formados por *VRF*, *Bridge Domains*, *Subnets* y *External Networks*.

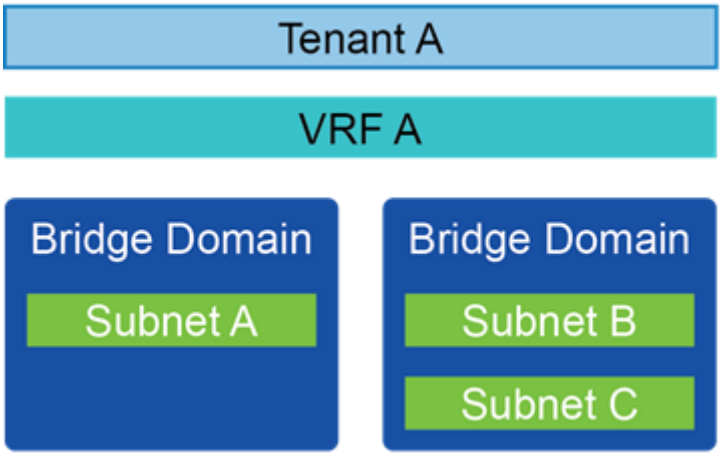

Fig. 15: Estructura de un *Tenant* con un *Networking Tenant*

- **VRF**, comúnmente conocidos como contextos y redes privadas, son tablas de enrutamiento aisladas para el *Tenant*. Un *Tenant* puede tener uno o varios VRF, o puede usar un VRF del *Tenant* "*common*".

- ***Bridge Domains***, que son los dominios de reenvío de capa 2 dentro del entramado, y definen el espacio de direcciones MAC exclusivo y los dominios de *Broadcast, Unknown Unicast* y *Multicast*. Cada *Bridge Domain* está asociado con un único VRF; no obstante, un VRF puede asociarse con muchos *Bridge Domain*. A diferencia de la forma tradicional en que se implementan VLANs, cada *Bridge Domain* puede contener múltiples subredes.

- ***Subnets***, aquellas redes de capa 3 que proporcionan espacio de IP y servicios de Gateway para que los hosts se conecten a la red. Cada subred está asociada a un único *Bridge Domain*.

- ***External Bridged Networks***, que conectan una red *Spanning-Tree*/capa 2 a la estructura ACI. Esto se suele utilizar en entornos de *Brownfield* para tener una migración fluida de una infraestructura de red tradicional a una red ACI. Un entorno *Brownfield* es un término utilizado para describir los espacios problemáticos que necesitan el desarrollo y despliegue de nuevos sistemas de software en presencia inmediata de aplicaciones o sistemas de software existentes. Esto implica que cualquier arquitectura nueva debe tener en cuenta y coexistir con el software que ya hay presente.

- ***External Routed Networks***, que crean una adyacencia de capa 3 con una red fuera de la topología ACI. Las redes externas de capa 3 admiten adyacencias usando rutas estáticas, así como los protocolos de enrutamiento BGP, OSPF y EIGRP. Las conexiones de capa 3 también tienen definidas redes que proporcionan y consumen Contratos.

4.1.3. Policy Tenants

Los objetos *Policy Tenant* están relacionados con los objetos de red, pero están más centrados en las políticas y los servicios que reciben los endpoints. Los *Policy Tenants* están formados por *Application Profiles*, *End Point Groups*, *Contracts* y *Filters*.

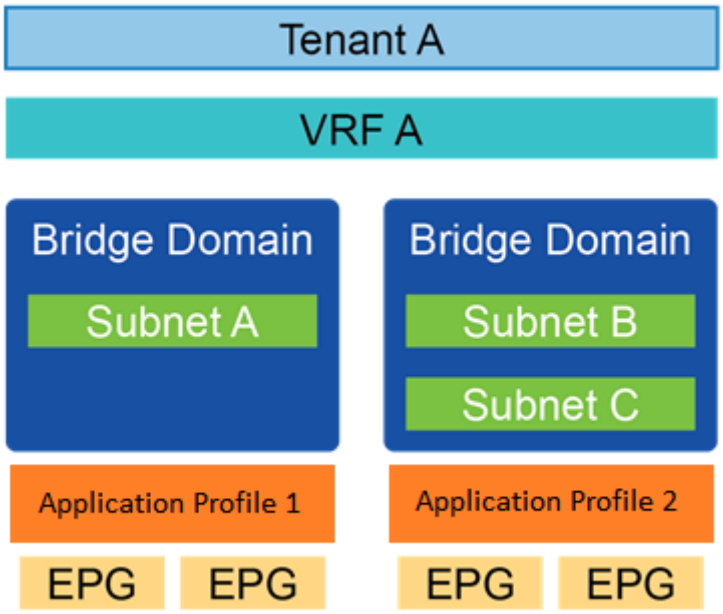

Fig. 16: Estructura de un *Tenant* con *Networking Tenant* y *Policy Tenant*

- ***Application Profiles***, que se utilizan como un identificador para aplicaciones y permite la separación de privilegios administrativos. Un *Application Profile* puede agrupar varios EPG.

- ***End Point Group* (EPG)**, que es una colección de endpoints que tienen aplicados los mismos servicios y políticas. Los EPG definen los puertos de switch, los switches virtuales y las encapsulaciones de capa 2 asociadas con un servicio de aplicación. Cada EPG solo puede estar en un solo *Bridge Domain*.

- ***Contracts***, son los que definen los servicios y políticas aplicados a los endpoints que forman un EPG. Los *Contracts* se pueden usar para redireccionar el servicio a un dispositivo de capa 4-capa 7, asignar valores de QoS y aplicar las reglas de las *Access List*. Los EPG que ofrecen los servicios proporcionan el *Contract*, mientras que los EPG que necesitan utilizar el servicio consumen el *Contract*.

- ***Filters***, que son los objetos que definen los protocolos (TCP, UDP, ICMP, etc.) y los puertos. Los objetos de tipo *Filter* pueden contener múltiples protocolos y puertos, y los *Contracts* pueden consumir múltiples *Filters*.

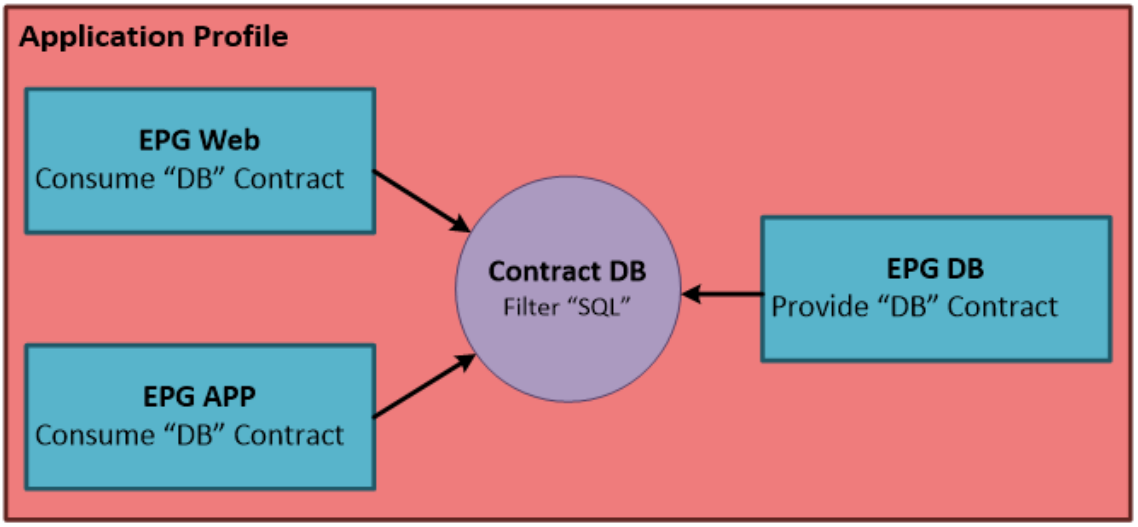

Fig. 17: Relación entre los elementos de un *Policy Tenant*

4.2. Programabilidad en ACI

Antes de explicar las opciones de programabilidad de ACI es necesario revisar el estado actual de la programabilidad de la red.

Las redes se han construido tradicionalmente con dispositivos diseñados para ser configurados y mantenidos de manera manual y de dispositivo en dispositivo. Para realizar cambios o solucionar problemas, un técnico debe conectarse a múltiples dispositivos de manera individual e introducir comandos en la consola para modificar la configuración del dispositivo. Esta solución funciona bien para entornos estáticos, pero también es más propenso a errores humanos.

Este el problema que ACI intenta resolver. ACI tiene un conjunto robusto y diverso en cuanto a opciones de programabilidad se refiere, y esto es posible gracias al Modelo de objetos de ACI.

4.2.1. Modelo de objetos

Como ya se ha explicado, ACI opera usando un modelo basado en objetos, que se usa para configurar y ver datos estadísticos. Éste modelo de objetos se puede dividir en dos categorías, modelo lógico y modelo concreto.

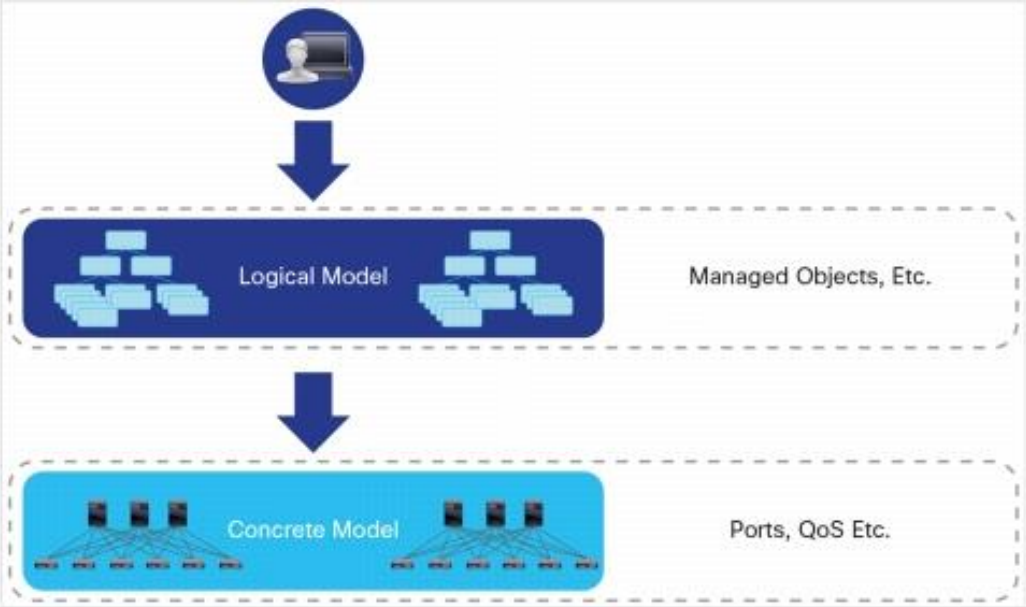

Fig. 18: Modelo de objetos de ACI

Los administradores y las herramientas de administración interactúan con el modelo lógico a través de los controladores APIC. Los cambios realizados en el modelo lógico se traspasan al modelo concreto, donde el hardware y el software se programan según sea necesario. Al separar de esta manera el modelo, se crea una única interfaz de administración donde se definen los estados deseados, en lugar de administrar la red de dispositivo en dispositivo, ayudando así a simplificar las estrategias de programación y automatización.

Dado que los administradores trabajan en el modelo lógico, se analizará cómo funciona éste modelo. El modelo lógico está organizado como una jerarquía de objetos, siendo *Root* el objeto de nivel superior.

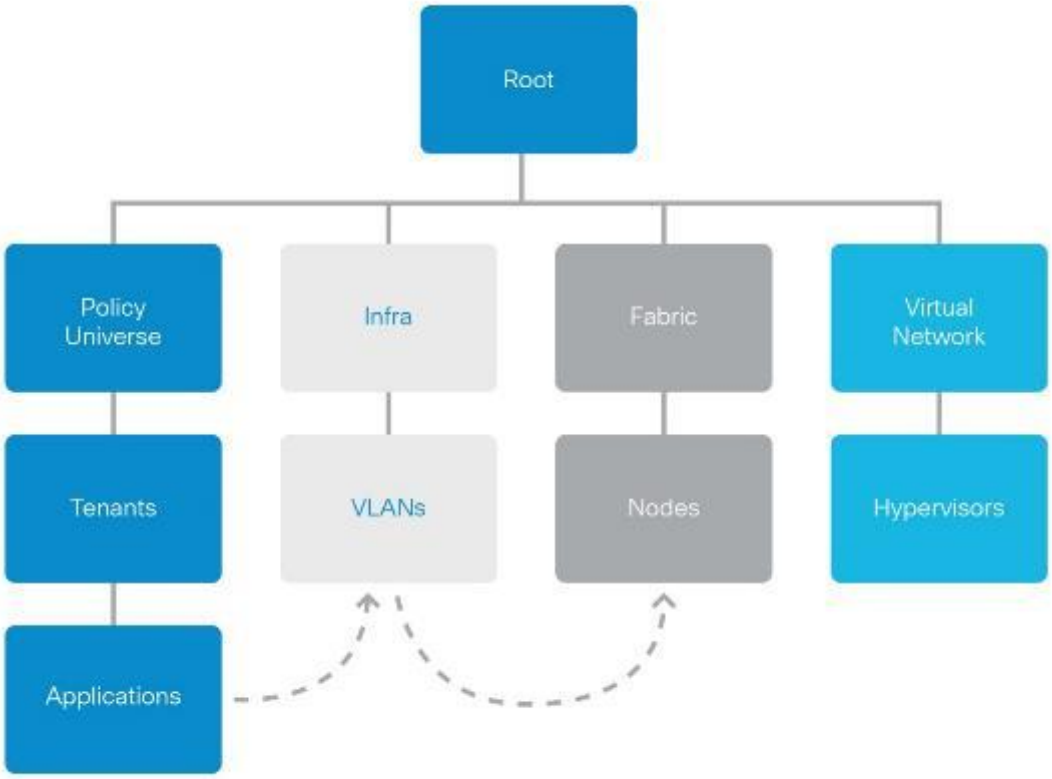

Fig. 19: Jerarquía de objetos en ACI

4.2.2. Objetos administrados y Árbol de información de gestión

Todo lo que forma parte de un tejido ACI es un objeto. Estos objetos se denominan Objetos Administrados (*Managed Objects*), MO para abreviar. Cada MO pertenece a una clase específica que determina que propiedades tiene y como funciona.

Los MO están conectados en las relaciones primarias y secundarias que forman el Árbol de información de gestión (*Management Information Tree*), MIT para abreviar. Cada MO tiene un padre, a excepción del objeto *Root*. El MIT se usa para agregar, eliminar o consultar objetos haciendo referencia a los padres del objeto hasta llegar al objeto *Root*. Los MO también pueden relaciones fuera del paradigma padre-hijo.

Las relaciones de los MO pueden ser de 1 a 1, de 1 a muchos y de muchos a muchos, como si de una BBDD se tratase. La siguiente imagen ilustra el MIT del MO *Tenant*, y resalta las diferentes formas en que los objetos pueden relacionarse.

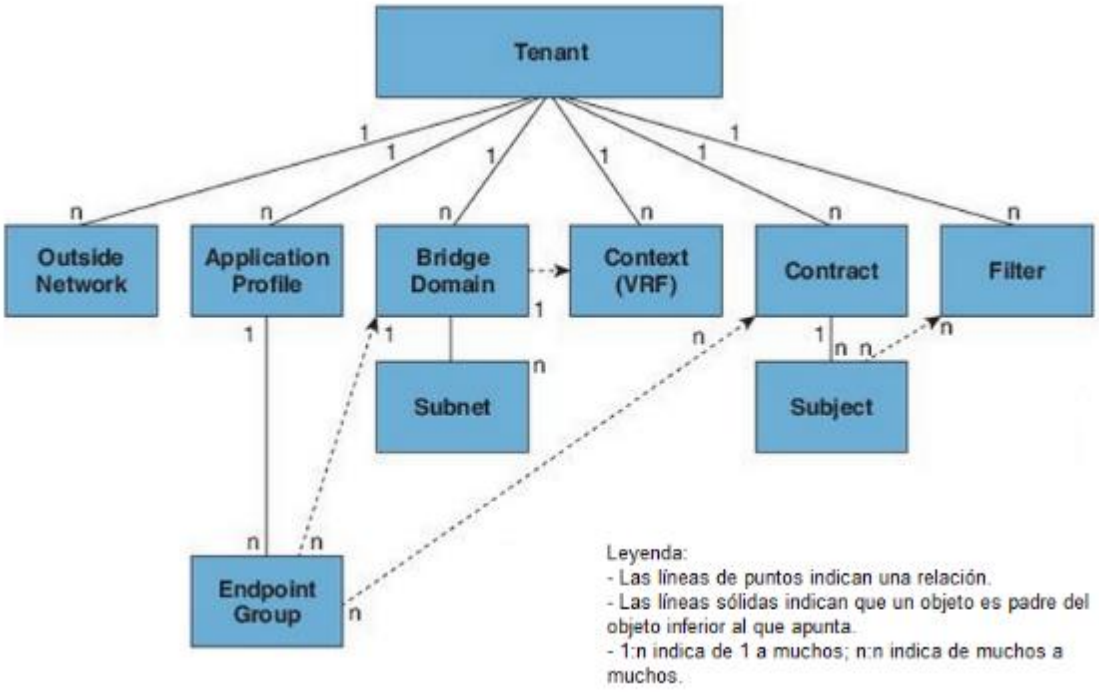

Fig. 20: MIT del objeto *Tenant*

Estas relaciones son importantes porque determinan qué recursos están disponibles para los objetos contenidos dentro de la relación. A continuación, hay un ejemplo concreto para demostrar estas dependencias.

4. Cisco: Nexus, ACI y sus APICs

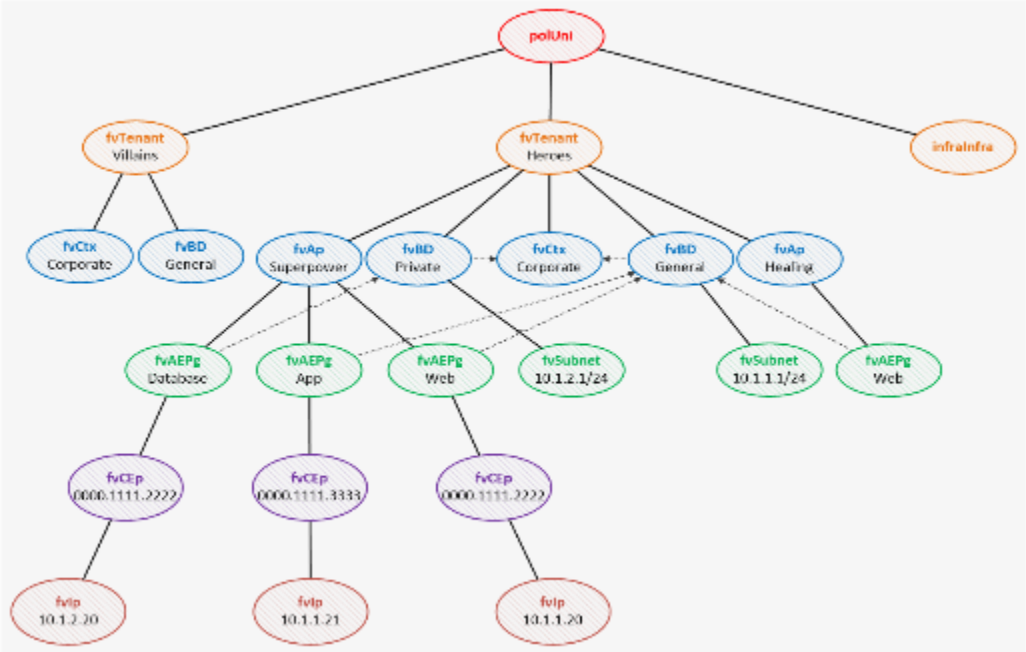

Fig. 21: Diagrama de ejemplo de dependencias entre objetos

En este diagrama se pueden observar diferentes cosas:

- Los objetos se puede identificar de manera única en relación con su objeto padre.
    - Por ejemplo, el objeto EPG (fvAEPg) Web se usa dos veces, pero pertenecen a dos objetos padres diferentes, por lo que cada uno es único.

- Todas las direcciones IP (fvip) en el *Tenant Heroes* deben ser únicas, ya que ambos *Bridge Domain* (fvBD) están relacionados con el mismo VRF (fvCtx).

- Las direcciones MAC se pueden duplicar siempre que los endpoints (fvCEp) pertenezcan a EPG relacionados con diferentes *Bridge Domain*.
    - fvAP *Superpower* > fvAEPg *Database* > fvCEp 0000.1111.2222 tiene una relación con fvBD *Private*.

    - fvAP *Superpower* > fvAEPg Web > fvCEp 0000.1111.2222 tiene una relación con fvBD *General*.

- La relación entre los EPG y los *Bridge Domain* determina que subredes (fvSubnet) pueden usar los endpoints.
    - fvAP *Superpower* > fvAEPg *Database* > fvCEp 0000.1111.2222 tiene que usar una IP en la subred 10.1.2.0/24.

    - Todos los demás EPG deben usar una IP en la subred 10.1.1.0/24.

Tres conceptos importantes a recoger después de ver estas características del MIT en cuanto a programación se refiere son:

- Es necesario referenciar los MO para acceder a los datos de configuración y referencia del mismo.

- Los objetos se acceden y se crean empezando por el objeto *Root*, y profundizando en el MIT hasta el MO deseado.

- Los programas deben tener en cuenta las relaciones entre los MO.
    - Ejemplo 1: para crear un EPG se necesitan los objetos *Tenant* y *Application Profile* para poderlos ubicar correctamente en el MIT. También se necesita un *Bridge Domain* para conectar los endpoints a las subredes y VRF correctas.

    - Ejemplo 2: para eliminar un *Bridge Domain*, antes hay que asegurarse de que ningún EPG está relacionado con el *Bridge Domain* que se quiere eliminar.

4.2.3. Nombres relativos y Nombres distinguidos

Cada MO tiene un Nombre Relativo (RN) que identifica un objeto de otros objetos secundarios del mismo padre. Aunque cada RN debe ser único dentro del mismo árbol que parte del padre del MO a nombrar, este nombre puede repetirse en otro MO de diferente padre.

Un RN empieza con un prefijo de clase y se basa en el tipo de clase del objeto. La última parte del RN es un identificador de propiedad que la clase usa para identificar un objeto, comúnmente, el nombre que se desea dar al objeto. A continuación hay una tabla con los prefijos e identificadores de propiedades más usados:

Nombre	Prefijo-Propiedad	Módulo	Clase	Clase padre	Ejemplo
Tenant	tn-name	fv	Tenant	Uni	tn-Heroes
VRF	ctx-name	fv	Ctx	Tenant	ctx-Corporate
Bridge Domain	BD-name	fv	BD	Tenant	BD-General
Subnet	subnet-ip	fv	Subnet	BD	subnet-10.1.2.1/24
Application Profile	ap-name	fv	Ap	Tenant	ap-Superpower
EPG	epg-name	fv	AEPg	Ap	epg-Database
Client Endpoint	cep-name	fv	CEp	AEPg	cep-0000.1111.2222
IP address	ip-addr	fv	Ip	CEp	ip-10.1.2.20
External L3	out-name	l3ext	Out	Tenant	out-Corporate
Filter	fit-name	vz	Filter	Tenant	fit-HTTP
Contract	brc-name	vz	BrCP	Tenant	brc-Web_Services
Contract Subject	subj-name	vz	Subj	BrCP	subj-HTTP

Tabla 2: Tabla de referencia para nombrar MO

Los RN se pueden usar para acceder a un objeto que se encuentre por debajo del objeto donde se está. Su uso más frecuente es para filtrar los resultados de las consultas mediante la API.

Los Nombres Distinguidos (DN) son una serie de RN que comienzan en el objeto padre y continúan el camino hasta el objeto en cuestión. Es el mismo concepto que la ruta de un archivo cuando quieres acceder a él mediante el CLI. Cada objeto en un mismo MIT tiene un DN único. Por ejemplo, en el diagrama mostrado en la Fig. 21, hay dos EPG que tienen el mismo RN (epg-Web), pero sus DN son diferentes:

- uni / tn-Heroes / ap-Superpower / epg-Web
- uni / tn-Heroes / ap-Healing / epg-Web

Por lo general, para construir un DN, se hace de manera inversa, empezando con el RN del objeto en cuestión, y buscando cada vez el padre. Por ejemplo, el proceso de creación del DN del MO Web, usando como *Application Profile* el que tiene de nombre *Superpower*, sería de la siguiente manera:

1. Construye el RN del objeto: **epg-Web**
2. Construye los objetos padre recursivamente y los añade delante:
    a. fvAp es la clase del objeto padre, y *Superpower* es su nombre, de manera que queda: **ap-Superpower / epg-Web**
    b. fvTenant es clase padre de fvAp-Superpower, y *Heroes* es su nombre, por lo que se añade resultando en: **tn-Heroes / ap-Superpower / epg-Web**
    c. Root es la clase padre de fvTenant: **uni / tn-Heroes / ap-Superpower / epg-Web**
3. El DN completo es **uni / tn-Heroes / ap-Superpower / epg-Web**

Los DN se utilizan para hacer solicitudes a la API para un objeto específico. Por ejemplo, si se quisiera saber datos acerca del objeto *Application Profile* Superpower, la solicitud de la API usaría el DN **único uni / tn-Heroes / ap-Superpower** como parte de la URL de la solicitud.

4.3. API REST de ACI

ACI se diseñó teniendo en cuenta la programabilidad, y se diseñó para ser configurado y mantenido a través de un controlador central a través de una API REST. Esta API es la forma en que los administradores interactúan con el modelo de objetos, lo que les permite realizar cambios, recopilar estadísticas y solucionar problemas en la estructura ACI.

La API REST utiliza peticiones HTTP para realizar las consultas o cambios, utilizando los cuatro métodos CRUD (*Create, Read, Update, Delete*):

- Crear nuevos objetos.
- Leer objetos parar ver la configuración y datos estadísticos.
- Actualizar los objetos existentes.

- Borrar objetos que no son necesarios.

A continuación se expondrán algunos de los métodos más básicos que se pueden hacer mediante la API REST. Para complementar la explicación, se usarán capturas de la aplicación Postman.

Postman es un cliente REST que se usa para interactuar con diferentes API REST, realizando peticiones a un servidor web y capturando su respuesta.

Para iniciar sesión en el APIC se ejecuta una petición de tipo POST, que envía un archivo JSON que contiene los datos del usuario que quiere hacer el login.

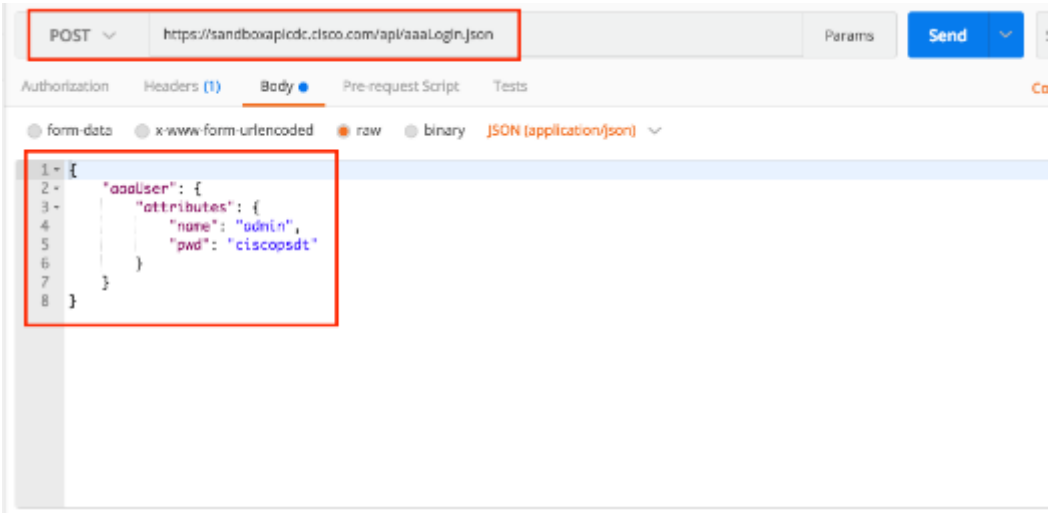

Fig. 22: Petición de inicio de sesión en Postman

Una vez hecha la petición, el cliente recibirá una respuesta del servidor. Esta respuesta contiene información acerca del estado de la conexión, y si es correcta, proporcionará una cadena de caracteres llamada *Token* que sirve como autenticación para con la API, ya que este *Token* se guardará y se enviará con cada petición. De lo contrario, sería necesario hacer un login cada vez que se quisiera hacer una petición.

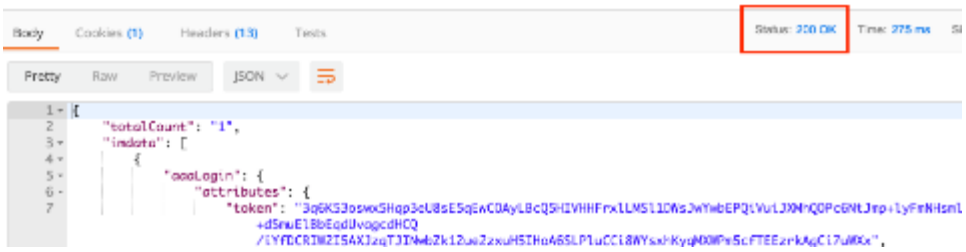

Fig. 23: Respuesta de la petición de inicio de sesión en Postman

Para realizar una petición de lectura se utiliza una petición GET. En la siguiente figura se puede ver la respuesta que recibe el cliente Postman cuando se hace una petición a la API para saber todos los perfiles de aplicación configurados en el entramado ACI.

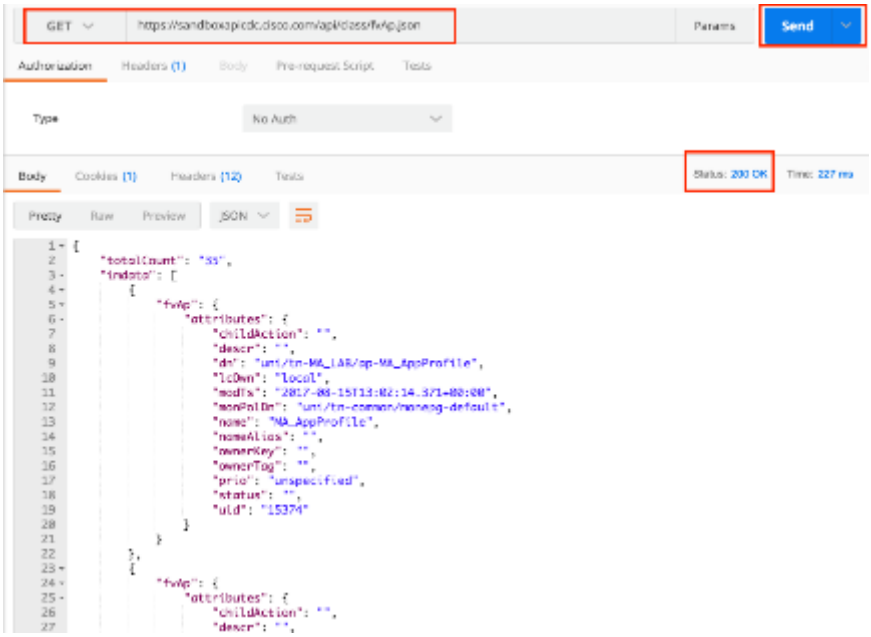

Fig. 24: Respuesta de la petición de lectura de perfiles de aplicación en Postman

El cuerpo de la respuesta enumera cada perfil de aplicación y proporciona toda la información de cada uno de sus atributos.

4.4. ACI Toolkit

Muchas personas comienzan a usar la API REST con una herramienta como Postman, ya que es una manera fácil de familiarizarse y hacer llamadas de prueba usando la API. No obstante, ACI ofrece otra herramienta, ACI Toolkit, que facilita comenzar a

interactuar con un APIC mediante programación. ACI Toolkit es un conjunto de librerías de Python que están diseñadas para interactuar con un subconjunto de los objetos más usados en el MIT (*Management Information Tree*). También ofrece una colección de scripts ya programados y probados que realizan tareas específicas.

ACI Toolkit se puede utilizar para crear, leer, actualizar y eliminar (métodos CRUD) la mayoría de los objetos dentro de la clase *Tenant*, y también funciona con las políticas de acceso a la estructura, como el encapsulamiento de capa 2, puertos físicos, canales de puertos, etc.

A continuación se pondrá un ejemplo básico que mostrará la sintaxis utilizada a la hora de programar un script que permita crear un objeto *Tenant* en el APIC usando ACI Toolkit.

A diferencia de cuando se usa un cliente REST como Postman, que proporciona una herramienta llamada *Token* para guardar la información de la sesión de un usuario, cuando se usa ACI Toolkit la información de la sesión se queda guardada en una variable, que te permite hacer múltiples operaciones mientras la sesión este activa, como crear objetos o recuperarlos del APIC.

Para crear la sesión, no obstante, se necesitan igualmente las credenciales de un usuario que pueda acceder al APIC. Estas credenciales serán las mismas para todas las operaciones que se quieran hacer, y puede que necesiten ser usadas por más de un script, por lo que la mejor manera de trabajar con ellas es guardarlas en variables en un fichero único (credentials.py) que podrán importar los scripts que requieran establecer una sesión con el APIC.

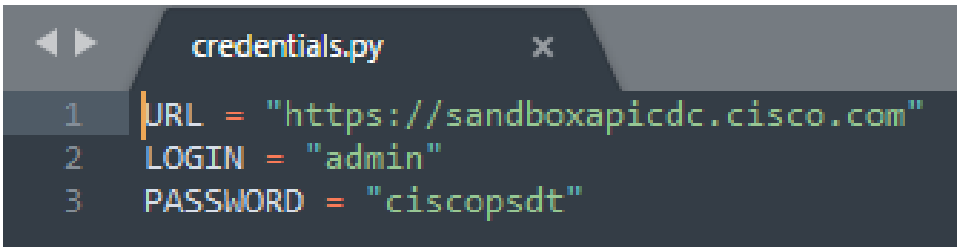

Fig. 25: Contenido del archivo credentials.py

createTenantToolkit.py

```
1  from credentials import *
2  from acitoolkit import acitoolkit
3
4  #Crear una sesion en el APIC usando las credenciales importadas.
5  sesion = acitoolkit.Session(URL, LOGIN, PASSWORD)
6
7  #Método para conectarse al APIC
8  sesion.login()
9
10 #Ahora que se tiene una conexión con el APIC, se procede a crear
11 #un objeto Tenant y guardarlo en una variable.
12 tenant = acitoolkit.Tenant("Toolkit_Tenant")
13
14 #Una vez creado el Tenant, se guarda el objeto en el APIC.
15 sesion.push_to_apic(tenant.get_url(), tenant.get_json())
```

Fig. 26: Ejemplo de *script* para crear un objeto *Tenant* usando ACI Toolkit

A pesar de que este ejemplo solo muestra cómo crear un objeto *Tenant* y guardarlo en el APIC, ACI Toolkit también se puede usar para programar scripts que crean una nueva aplicación con varios EPG (*End Point Groups*), asignar estos EPG al correcto BD (*Bridge Domain*), hacer que proporcionen y/o consuman *Contracts* y conectarlas a la infraestructura física y virtual.

Si bien ACI Toolkit es una gran herramienta para programar contra un APIC, solamente permite realizar las operaciones más comunes en el tejido ACI.

4.5. Cobra SDK

Cobra es un SDK (*Software Development Kit*) de Python que admite operaciones CRUD del tejido ACI, pero es más complejo que el ACI Toolkit, ya que es un mapeo completo del modelo de objetos.

Un mapeo completo del modelo de objetos significa:

- El MIT (*Management Information Tree*) completo está incorporado en el SDK Cobra.

- Todas las operaciones accesibles a través de la API también se pueden hacer con Cobra.

- El SDK se modela después del MIT, por lo que los nombres de las clases de Python y sus variables coincidirán con lo que hay en el MIT.

- Se puede acceder a las variables de una clase llamando al nombre de la variable de la siguiente manera "objeto.variable". Ejemplos:
    - tenant**.name:** El nombre del objeto *Tenant.*
    - endpoint**.ip:** La IP del objeto *endpoint*.

Dado que Cobra ofrece una funcionalidad completa, es más adecuado para consultas más complejas con filtros, incorporando paquetes de dispositivos de capa 4 a capa 7 para dispositivos de servicio de red externos a ACI, y creando scripts de creación de partes funcionales del tejido inicial que configuran tareas menos comunes, como SNMP, *Syslog*, políticas BGP, etc.

A continuación, al igual que con ACI Toolkit, se mostrará un script que permite crear un objeto *Tenant* y guardarlo en el APIC.

createTenantCobra.py

```python
from credentials import *
import cobra.mit.access
import cobra.mit.request
import cobra.mit.session
import cobra.model.fv
import cobra.model.pol

#Crear una sesión en el APIC usando las credenciales importadas.
auth = cobra.mit.session.LoginSession(URL, LOGIN, PASSWORD)
sesion = cobra.mit.acces.MoDirectory(auth)

#Método para conectarse al APIC
sesion.login()

#Ahora que se tiene una conexión con el APIC, se porcede a crear
#un objeto Tenant y guardarlo en una variable. No obstante, antes
#es necesario saber que padre será el objeto del Tenant. En este
#caso, el padre será Root
root = cobra.model.pol.Uni("")
tenant = cobra.model.fv.Tenant(root, "Cobra_Tenant")

#Una vez creado el Tenant, se guarda el objeto en el APIC.
config_request = cobra.mit.request.ConfigRequest()
config_request.addMo(tenant)
sesion.commit(config_request)
```

Fig. 27: Ejemplo de script para crear un objeto *Tenant* usando Cobra SDK

En el código se pueden apreciar algunas de las complejidades relacionadas con el uso de Cobra respecto al ACI Toolkit. No obstante, también tiene herramientas que lo hacen más accesible.

Debido a la mayor complejidad que tiene el uso del SDK Cobra, Cisco proporciona algunas herramientas para ayudar a los administradores.

Una de estas herramientas es Arya (*APIC Rest pYthon Adapter*), que ayuda a generar un script en Python usando las librerías de Cobra. Esto lo hace cogiendo los datos de configuración de un objeto en formato XML o JSON, y transformándolos en un script en Python que se podría usar para generar la misma configuración de ese objeto. De esta manera, se puede conseguir fácilmente una plantilla para la configuración de objetos usando Cobra.

Las otras herramientas que Cisco provee para facilitar la programabilidad del APIC se encuentran en la GUI del mismo. Esas son:

- **GUI *Download***, que permite descargarse la configuración de un objeto existente en el APIC en formato XML o JSON.

- **Visore**, un navegador de objetos que también tiene una opción de descarga XML.

- **API *Inspector***, que se utiliza para obtener los datos de configuración y URL a medida que se realizan las solicitudes a la API.

4.6. Comprobar resultados en la GUI de APIC

En la GUI del APIC se puede comprobar si los *Tenant* creados con los scripts se han creado correctamente. Para visualizar la información se deben seguir los siguientes pasos:

1. Hacer clic en la pestaña *Tenants* del menú de navegación situado en la parte superior central de la pantalla.
2. Hacer clic en la sub-pestaña *ALL TENANTS* que aparece.

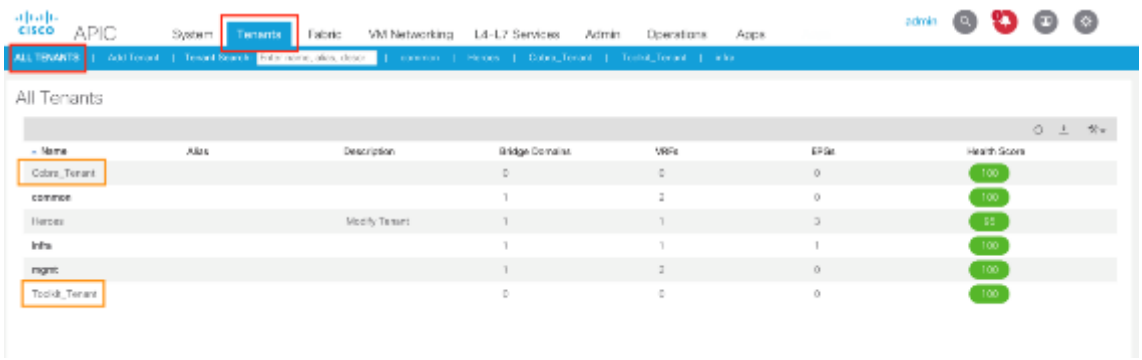

Fig. 28: Visualización de los *Tenant* del APIC

5. Aplicación Crear Tenant

En este apartado se explicarán los pasos seguidos al desarrollar una aplicación que permita la creación de un *Tenant* en el APIC, usando para ello diferentes herramientas.

La aplicación ha sido desarrollada usando Python, el *framework* Kivy y el conjunto de librerías formado por ACI Toolkit.

5.1. Entendiendo Python

Python [6] es un lenguaje de programación creado y administrado por la *Python Software Fundation*, y posee una licencia de código abierto. Gracias a este hecho, el lenguaje tiene una gran comunidad que está constantemente aportando nuevas librerías y funcionalidades de forma pública, lo que ha provocado que goce de una gran popularidad entre los desarrolladores de software.

Python es un lenguaje multiparadigma, ya que soporta programación orientada a objetos, programación imperativa y, en menor medida, programación funcional. Python es un lenguaje interpretado, usa un tipado dinámico y es multiplataforma.

Python ha sido diseñado para ser leído con facilidad, por lo que se compone de una serie de características que lo hacen posible. A continuación se expondrán los principales elementos del lenguaje.

5.1.1. Uso de palabras en vez de símbolos

Mientras otros lenguajes usan símbolos para designar operadores lógicos, Python utiliza palabras. Por ejemplo, el operador '!' de Java se escribe 'not' en Python.

5.1.2. Identación

Mientras que muchos lenguajes utilizan elementos como las llaves '{}' para declarar los bloques de código, lo que hace que no sea necesario identar el código, Python no usa estos elementos, por lo que la identación es necesaria para que al compilar el código, el interpretador sepa identificar que código forma parte de un bloque. La identación puede ser mediante espacios o tabuladores, pero no se deben mezclar.

5.1.3. Comentarios

En Python se pueden poner comentarios de dos formas, dependiendo del tamaño del comentario. Si el comentario es extenso y usa más de una línea, se utilizan triples

comillas simples para encapsular el comentario, mientras que si cabe en una línea, basta con poner delante una almohadilla '#'.

5.1.4. Variables

Las variables se definen de forma dinámica, de manera que no hay que especificar cuál es su tipo de antemano, y puede cambiar de tipo si es necesario. A continuación hay una tabla con los diferentes tipos de variables que se pueden usar en Python:

Tipo	Clase	Notas	Ejemplo
str	Cadena	Inmutable.	'Cadena'
unicode	Cadena	Versión Unicode de str	u'Cadena'
list	Secuencia	Mutable	[4.0, 'Cadena', True]
tuple	Secuencia	Inmutable	(1.0, 'Cadena', True)
set	Conjunto	Mutable, sin orden, sin duplicados	set([4.0, 'Cadena', True)]
frozenset	Conjunto	Inmutable, sin orden, sin duplicados	frozenset([4.0, 'Cadena', True)]
dict	Mapping	Grupo de pares clave:valor	{'key1':1.0, 'key2':True}
int	Número entero	Precisión fija	42
long	Número entero	Precisión arbitraria	42L
float	Número decimal	Coma flotante de doble precisión	3.1415
complex	Número complejo	Parte real y parte imaginaria	(4.5+3j)
bool	Booleano	Valor booleano verdadero o falso	True o False

Tabla 3: Tipos de variables

5.1.5. Condicionales

Las expresiones condicionales rodean bloques de código que ejecutan solo si se cumple la condición especificada. Se definen usando la palabra clave "*if*" seguida de la condición. En el caso de que haya condiciones adicionales, se definen usando "*elif*". Opcionalmente, puede haber un bloque condicional final precedido de "*else*" que se ejecuta sólo cuando las otras condiciones no se han cumplido.

5.1.6. Bucles

En Python existen dos tipos de bucle, el bucle *for* y el bucle *while*.

El bucle *for* es similar al de otros lenguajes. Consiste en recorrer un objeto iterable, como una lista, y por cada elemento del iterable ejecuta el bloque de código interno.

El bucle *while*, a diferencia, consiste en evaluar una condición. Si es verdadera, ejecutará todo el código definido en el bloque, y volverá a comprobar la condición. El proceso se repite hasta que la condición no se cumple.

5.1.7. Métodos

Los métodos son bloques de código que pueden ser llamados dentro de la misma clase, si se está usando programación orientada a objetos, o desde cualquier parte del código en caso contrario. Se definen usando la palabra clave "*def*", seguida del nombre de la función, sin usar espacios en el nombre, y de sus parámetros entre paréntesis. Los métodos que forman parte de una clase siempre tienen que tener el parámetro "*self*", ya que hace referencia a la instancia actual de la clase.

5.1.8. Clases

Las clases forman un bloque de código que representan, normalmente, a un objeto del mundo real, y que contienen variables (llamadas atributos en una clase) y métodos (llamados funciones en una clase) asociadas a las características del objeto en cuestión.

Las clases se definen usando la palabra clave "*class*", seguida del nombre de la clase, sin usar espacios en el nombre, y entre paréntesis, el nombre de la clase de la que hereda, si es el caso.

El método usado para poder instanciar una clase es "*def __init__(self)*". Es usado normalmente utilizado para definir e inicializar los atributos que contenga.

Para hacer que un atributo sea accesible desde fuera de la clase se debe declarar usando "*self.*" delante del nombre. En el caso de que un atributo no sea declarado de esta forma, no se podrá acceder a él desde fuera de la clase, por lo que si se quiere cambiar su contenido, se debería crear un método que permita realizar el cambio de valor. Esto funciona de manera similar a las declaraciones "*private*" y "*public*" de Java.

5.1.9. Módulos

Los módulos, también conocidos como librerías o APIs, son archivos que contienen multitud de métodos y clases que se pueden importar al código para realizar determinadas tareas. Un ejemplo es el módulo "*os*", que permite usar funciones del sistema operativo, o la API de ACI Toolkit, que permite comunicarse con el APIC de Cisco. Los módulos se agregan al código escribiendo "*import*" seguido del nombre del

módulo a incluir, y siempre en las primeras líneas de código. Se pueden importar solamente aquellas partes del módulo que se quieran usar utilizando la estructura "*import X from Módulo*".

5.2. ¿Qué es Kivy?

Kivy [7] es un *framework* de código libre desarrollada en Python que permite desarrollar aplicaciones móvil y diferentes aplicaciones *multitouch* con una interfaz natural de usuario. Kivy se puede ejecutar en Android, iOS, Linux, OS X y Windows, y se distribuye bajo los términos de la licencia MIT, lo que lo convierte en una librería de código libre.

Este *framework* es el principal desarrollado por la organización Kivy, en conjunto con *Python for Android*, Kivy iOS y otras librerías capaces de ser usadas en todas las plataformas disponibles.

La principal característica de Kivy es el uso del lenguaje Kivy (*Kv*), que está dedicado a describir la interfaz de usuario y sus interacciones. Está basado en los lenguajes de etiquetas, y hace posible crear de manera sencilla una interfaz de usuario a la que añadir funcionalidades. El uso de este lenguaje permite separar el diseño de la parte gráfica de la parte lógica de la aplicación.

Una aplicación desarrollada usando Kivy tiene la parte lógica escrita en Python y la parte gráfica escrita en *Kv*.

Existen dos maneras de usar el código *Kv* en una aplicación:

- **Relación nominal:** Cuando se programa en Kivy, se hace usando orientación a objetos, y la clase principal se llama usando la siguiente estructura "*nombreApp*". Al ejecutarse la aplicación, Kivy buscará un archivo con extensión "*.kv*" que se llame igual que la clase de la aplicación desarrollada, menos "*App*". Si la clase se llama "*pruebaApp*", Kivy buscará el archivo "*prueba.kv*".
- ***Builder*:** La otra manera de usar el código *Kv* es cargando directamente una cadena usando la clase *Builder* de Kivy, usando la expresión "*Builder.load_string(kv_string)*".

5.3. Desarrollo de la aplicación

Se ha desarrollado una aplicación que implementa la funcionalidad de crear un *Tenant* en el APIC de pruebas que proporciona Cisco. La aplicación consta de dos pantallas, una para establecer la conexión con el APIC mediante un login, y otra para la creación del *Tenant*.

5.3.1. Diseño de las pantallas

Antes de escribir el código que da cuerpo a las pantallas, se hizo un esbozo simple acerca de su contenido y estructura.

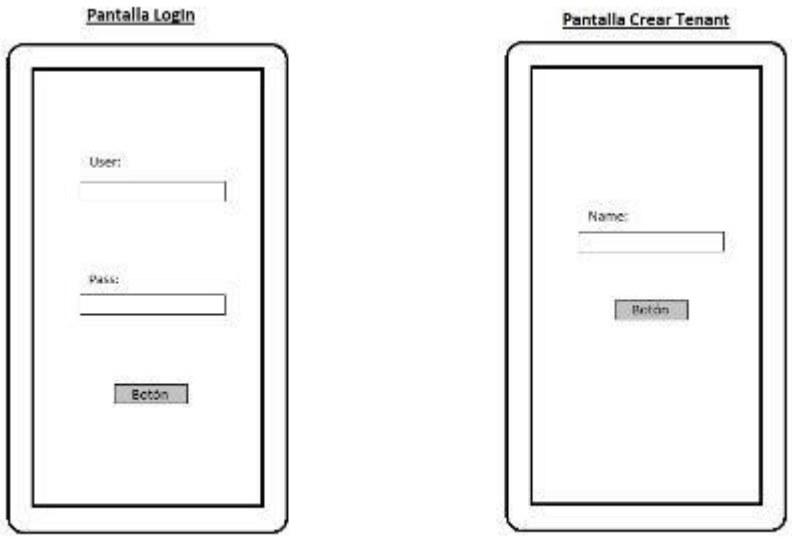

Fig. 29: Primer diseño gráfico de las pantallas

Una vez establecido el diseño de las pantallas, se procede a escribir el código, usando el lenguaje *Kv*.

A continuación se expondrá el código perteneciente a la pantalla de login.

```
<MainScreen>:
    name: "MainScreen"
    canvas:
        Rectangle:
            size: self.size
            source: "fondo.png"

    BoxLayout:
        id: login_layout
        orientation: 'vertical'
        padding: [100,230,120,350]
        spacing: 40

        BoxLayout:
            orientation: 'vertical'

            Label:
                text: 'Login'
                font_size: 20
                halign: 'center'
                text_size: root.width-20, 20
                color: 0,0,0,1

            TextInput:
                id: login
                multiline:False
                font_size: 15

        BoxLayout:
            orientation: 'vertical'
            Label:
                text: 'Password'
                halign: 'center'
                font_size: 20
                text_size: root.width-20, 20
                color: 0,0,0,1

            TextInput:
                id: password
                multiline:False
                password:True
                font_size: 15

        Button:
            text: 'Conectar'
            font_size: 20
            on_release:
                app.root.conectar("CreateTenantScreen", "left", login.text, password.text)
                login.text = ""
                password.text = ""
```

Fig. 30: Código *Kv* de la pantalla de login

Este código pertenece a la pantalla principal de la aplicación. Está escrito en *Kv* y posee algunos de los elementos gráficos que aporta Kivy. Como se puede apreciar, hay diferentes niveles de identación. Cada nivel hace referencia a un objeto diferente, y a su posición en la jerarquía. Todas las propiedades u objetos hijos deben estar en el nivel de identación que pertenezca a su padre. Los objetos que se usan en esta pantalla son:

5. Aplicación Crear Tenant

- ***MainScreen*:** Objeto de tipo *Screen*. Al ser un objeto que también se declara y usa en el código Python, está rodeado de los símbolos '< >'. Contiene las siguientes propiedades:
    - ***name*:** Nombre de la pantalla. Sirve para poder cambiar de una pantalla a otra.
    - ***canvas:*** Contiene un objeto consistente en una figura rectangular del mismo tamaño de la pantalla que contiene la imagen de fondo de la aplicación.
- ***BoxLayout*:** Objeto que sirve a modo de caja para colocar otros objetos. Estos se apilan en función de la orientación, vertical u horizontal. Tiene las siguientes propiedades:
    - ***id*:** Nombre dado al objeto para poder interaccionar con los elementos que contiene.
    - ***orientation*:** Orientación en la que se apilan los objetos.
    - ***padding*:** Relleno que hay entre los bordes y los objetos apilados.
    - ***spacing*:** Separación entre los objetos apilados.
- ***Label*:** Objeto que tiene un texto fijo que el usuario no puede cambiar. Tiene las siguientes propiedades:
    - ***text*:** Contiene el texto que se mostrará.
    - ***font_size*:** Tamaño de la fuente de texto.
    - ***halign*:** Alineación del texto.
    - ***text_size*:** Tamaño del encuadrado del texto.
    - ***color*:** Color del texto.
- ***TextInput*:** Cuadro de texto que con el que puede interactuar el usuario. Contiene las siguientes propiedades:
    - ***id*:** Nombre dado al objeto para poder interaccionar con los elementos que contiene.
    - ***multiline*:** Contiene un valor booleano (*True* o *False*).
    - ***password*:** Cambia el formato del texto de letras a '*'.
    - ***font_size*:** Tamaño de la fuente de texto.
- ***Button*:** Botón al que se le pueden asignar funciones cuando sea pulsado o despulsado. Contiene las siguientes propiedades:
    - ***text*:** Contiene el texto que mostrará el botón.

- ***font_size*:** Tamaño de la fuente del texto.
- ***on_release*:** La acción que llevará a cabo el botón una vez sea pulsado y despulsado. En el caso de querer realizar más de una acción, se debe abrir un nuevo bloque identado y usar una línea de código para cada acción. Una de las acciones que realiza este botón es una función que está declarada en la clase *ScreenManager*, un objeto que contiene las dos pantallas y permite interaccionar con ellas, así como cambiar de una a otra. La forma de llamar al método consiste en pasar por los elementos de la jerarquía hasta el objeto que implementa dicha función. En este caso, *app* es el *ScreenManager*, que se denomina de esa manera porque es el elemento principal de la aplicación. Por ejemplo, si el elemento principal de la aplicación fuera otro objeto como un *Widget*, y éste tuviese implementado el método que quiere llamar el botón, *Widget* sería *app* en la jerarquía. El segundo elemento de la jerarquía es la propia pantalla que contiene el botón, lo que se denomina en el código como *root*. Por último, se escribe el nombre del método y se le pasan los argumentos que éste necesita.

A continuación está el resultado final de la pantalla de login una vez compilado el código de la aplicación.

5. Aplicación Crear Tenant

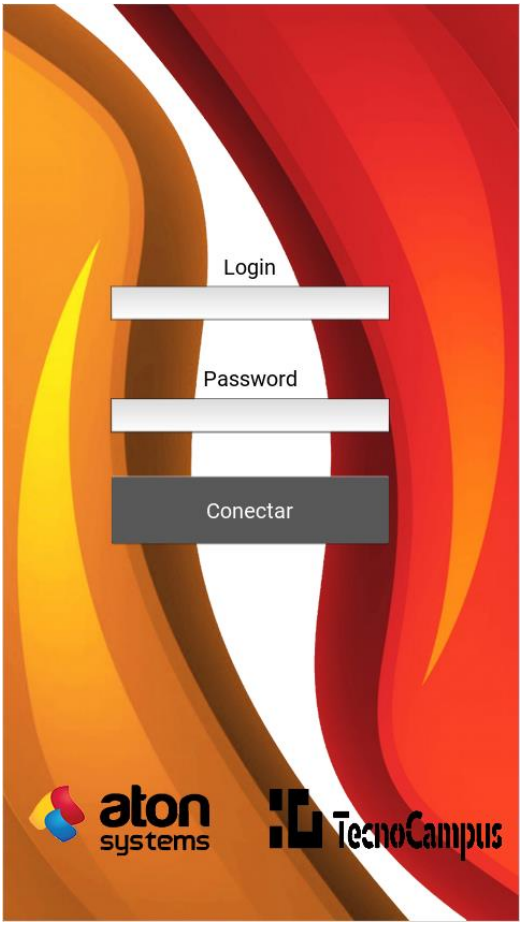

Fig. 31: Pantalla final de login

En la siguiente imagen se expondrá el código perteneciente a la pantalla de creación de *Tenant*. Esta pantalla es muy parecida a la anterior, y solo contiene un elemento nuevo respecto a su predecesora, un segundo botón que permite volver a la pantalla anterior.

```
<CreateTenantScreen>:
    name: "CreateTenantScreen"
    canvas:
        Rectangle:
            size: self.size
            source: "fondo.png"

    BoxLayout:
        id: login_layout
        orientation: 'vertical'
        padding: [100,250,120,350]
        spacing: 40

        BoxLayout:
            orientation: 'vertical'

            Label:
                text: 'Nombre del Tenant'
                font_size: 20
                halign: 'center'
                text_size: root.width-20, 20
                color: 0,0,0,1

            TextInput:
                id: nameTenant
                multiline:False
                font_size: 15

        Button:
            text: 'Crear Tenant'
            font_size: 20
            on_release: app.root.crearTenant("MainScreen", "right", nameTenant.text)

        Button:
            text: 'Cancelar'
            font_size: 20
            on_release:
                root.manager.transition = SlideTransition(direction="right")
                root.manager.current = root.manager.previous()
```

Fig. 32: Código *Kv* de la pantalla de creación de *Tenant*

En el código de esta segunda pantalla, hay un segundo botón que sirve para volver a la pantalla anterior. Este botón ha sido implementado debido a que era necesaria una forma de volver atrás sin crear un *Tenant*. Este botón asigna al *ScreenManager* que contiene las pantallas una transición, que es la animación usada al cambiar de una pantalla a otra, y posteriormente, vuelve a la pantalla anterior. Los otros botones de la aplicación también efectúan un cambio de pantalla, sin embargo éstos están ligados a condiciones concretas, por lo que se resuelven en el código Python de la aplicación.

A continuación se muestra el resultado final de la pantalla de creación de *Tenant* una vez compilado el código de la aplicación.

5. Aplicación Crear Tenant

Fig. 33: Pantalla final de creación de *Tenant*

Por último, hay otro objeto declarado en el código *Kv* que pertenece a un pop-up que aparece cuando se cumplen las condiciones necesarias.

```
<myPopup>:
    size_hint: .9, .12
    auto_dismiss: False
    Button:
        text: "Cerrar"
        on_release: root.dismiss()
```

Fig. 34: Código *Kv* del pop-up

En este pop-up solo hay un objeto, un botón para cerrar la ventana emergente. Las propiedades del pop-up son:

- ***size_hint*:** Porcentaje del tamaño usado por el padre del objeto (*ScreenManager*) en el eje de las 'x' e 'y', entre el 0 y el 1. En esta aplicación quiere decir que el pop-up ocupará un 90% de la anchura del padre y un 12% de su altitud.
- ***auto_dismiss*:** Propiedad booleana que indica si el pop-up se cierra al tocar en cualquier lugar de la pantalla situado fuera del pop-up.

A continuación se muestra un ejemplo de los diferentes pop-ups que hace saltar la aplicación una vez compilado el código de la misma.

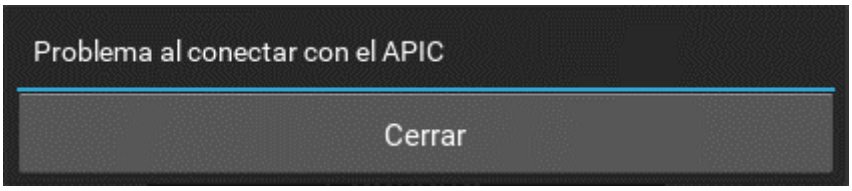

Fig. 35: Pop-up de alerta

5.3.2. Cuerpo de la aplicación

El cuerpo de la aplicación está escrito en Python, y utiliza el *framework* de Kivy y la librería de ACI *Toolkit*. Para comprobar la funcionalidad de ACI *Toolkit* y el APIC de Cisco se ha implementado solo un caso de uso, el de creación de *Tenant*. Con el *framework* Kivy, una aplicación estará en ejecución hasta que se cierre, o si se implementa un botón que permita el cierre de la aplicación. Si se cambia de aplicación, ésta se queda en estado de suspensión hasta que se vuelve a ella. Estos estados de ejecución se pueden controlar, pero en este proyecto no se ha trabajado con ellos. A continuación se explicarán las partes más importantes del código.

```
from kivy.config import Config
Config.set('graphics', 'resizable', False)
Config.set('graphics', 'width', '480')
Config.set('graphics', 'height', '854')
from kivy.app import App
from kivy.uix.screenmanager import ScreenManager, Screen, SlideTransition
from kivy.uix.popup import Popup
from kivy.uix.button import Button
from kivy.lang import Builder
from acitoolkit import acitoolkit
```

Fig. 36: Importaciones o Módulos

5. Aplicación Crear Tenant

Al inicio del código se encuentran el conjunto de módulos que permiten usar las funciones de la librería ACI *Toolkit* y el *framework* Kivy. Las cuatro primeras líneas establecen la ventana de la aplicación, basadas en este caso en un móvil con pantalla de entre 5 y 5'5 pulgadas. Las siguientes líneas importan todos los módulos y objetos usados en la aplicación, así como la librería ACI *Toolkit* que permite comunicar con el APIC.

A continuación de las importaciones, está el código *Kv* ya expuesto previamente, seguido de las clases que hacen referencia al código *Kv*.

```
class myPopup(Popup):
    pass

class MainScreen(Screen):
    pass

class CreateTenantScreen(Screen):
    pass
```

Fig. 37: Clases de objetos referenciados en el código *Kv*

Estas clases están vacías debido a que no tienen ningún atributo o función que realizar más que el ser creadas, ya que es el código *Kv* quien se encarga de dotarlas de propiedades y contenido. En Python, cuando una clase está vacía, y debido a que no hay símbolos que indican cuando empieza y acaba un bloque de texto, hace falta añadir la palabra *pass* correctamente identada para marcar el final de clase. Las clases que hacen referencia a objetos declaradas en el código *Kv* se deben llamar igual en ambas partes del código.

Seguida de estas tres clases, está la clase correspondiente al *ScreenManager*. En esta clase está implementada la conexión con el APIC y la acción de crear *Tenant*, así como el control de errores, lanzamiento de alertas y cambios de pantalla.

```
class MyScreenManager(ScreenManager):
    sesion = ""
    def conectar(self, where, direction, login, password):
        self.sesion = acitoolkit.Session("https://sandboxapicdc.cisco.com", login, password)
        login= self.sesion.login()
        if not login.ok:
            popup = myPopup(title="Problema al conectar con el APIC")
            popup.open()
            self.sesion.close()
        else:
            popup = myPopup(title="Conectado")
            popup.open()
            self.transition = SlideTransition(direction=direction)
            self.current = where


    def crearTenant(self, where, direction, nameTenant):
        tenant = acitoolkit.Tenant(nameTenant)
        guardar = self.sesion.push_to_apic(tenant.get_url(), tenant.get_json())
        if not guardar.ok:
            popup = myPopup(title="Problema al crear el Tenant")
            popup.open()
        else:
            popup = myPopup(title="Tenant creado y guardado en el APIC")
            popup.open()
            self.transition = SlideTransition(direction=direction)
            self.current = where
```

Fig. 38: Clase *MyScreenManager*

Esta clase usa la variable sesión, por defecto vacía, para guardar la sesión abierta con el APIC y poder enviarle los datos.

El primer método es llamado por el botón de la pantalla principal, y sirve para establecer la conexión con el APIC. Debido a que se ha usado el APIC de pruebas proporcionado por Cisco, se ha decidido no pedir la URL en el formulario de la aplicación, y escribirla como parámetro fijo en el código. A continuación, si la sesión se establece correctamente, se muestra un pop-up informando del éxito de la conexión y se cambia a la pantalla de creación de *Tenant*. En caso contrario, se muestra otro pop-up, y se cierra la sesión, ya que si no, se provoca un error de ACI *Toolkit* al intentar volver a llamar el método *conectar()* al pulsar el botón.

La segunda función está vinculada al botón 'Crear *Tenant*' de la pantalla de creación de *Tenant*. De manera similar al anterior método, éste llama a las funciones de ACI *Toolkit* que permiten crear un *Tenant* y enviarlo al APIC. Si la operación resulta exitosa, informa abriendo un pop-up y vuelve a la pantalla de inicio. En caso contrario, muestra la alerta y permite reintentar la operación.

Por último, se encuentra la clase encargada de inicializar todas las otras, y el bloque condicional que sirve para ejecutar la aplicación.

```
class MainApp(App):
    def build(self):
        root = MyScreenManager()
        root.add_widget(MainScreen())
        root.add_widget(CreateTenantScreen())
        return root


if __name__ == '__main__':
    MainApp().run()
```

Fig. 39: Clase *MainApp* y *main*

5.4. Montaje de la aplicación

Actualmente, las aplicaciones desarrolladas con Kivy que se quieran montar como una aplicación móvil solo puede hacerse en un entorno Linux configurado con *python-for-android*, el Android SDK y el Android NDK. Debido a la dificultad que tiene configurar las herramientas necesarias, además de la imposibilidad de hacerlo en sistemas operativos Windows o OS X, la organización Kivy proporciona una máquina virtual completamente configurada que permita el montaje de la aplicación. No obstante, debido al uso de ACI *Toolkit* y a todas las dependencias que se instalan en el sistema, no se ha podido montar la aplicación usando la máquina virtual proporcionada por Kivy para poder usar la aplicación en un Smartphone. No obstante, la aplicación si se puede ejecutar y comprobar su funcionamiento en un ordenador donde estén instalados Python, Kivy, ACI *Toolkit* y todas sus dependencias.

6. Conclusiones

Empecé a pensar en la idea de este trabajo a raíz de un proyecto que en Aton Systems ofrecimos realizar a uno de nuestros clientes. El proyecto en cuestión consistía en proporcionar una herramienta que simplificase la gestión de ciertos aspectos de la red del cliente sin tener que pasar por el *Dashboard* de NetSight, debido a lo tedioso que esto resultaba. Por ese motivo, pensaba decantarme por el fabricante Extreme Networks para trabajar con su API. No obstante, al desarrollar el proyecto, fui descubriendo la escasa documentación que había acerca de dicha API, y no me quedó más remedio que trabajar con otro fabricante, Cisco.

Con Cisco me encontré con una gran cantidad de información, además de con diferentes opciones para manipular su API, lo que me permitió unir la programación con la gestión de red. Tuve que aprender Python para trabajar con la API, pero no fue un gran reto debido a todas las nociones que aprendí durante los años que he estado cursando el grade de Ingeniería Informática, donde han hecho gran hincapié en muchos aspectos de la programación.

Por contrapartida, he tenido muchas más dificultades en lo que atañe a la gestión de red, debido a que los alumnos no tuvimos mucha materia acerca del tema, así como de sistemas, y considero que podría ser una gran mejora al plan de estudios. A pesar de ello, el hecho de trabajar en una gran empresa que se dedica al mundo de las redes y seguridad y estar rodeado de grandes profesionales que son maestros de la materia me ha permitido aprender del tema a pasos agigantados, y poder entender acerca de lo que hablo en este trabajo.

Después de haber investigado acerca del uso de las APIs y como éstas se pueden combinar con la programación para crear aplicaciones que permitan gestionar partes de una red de manera intuitiva, creo que vale la pena seguir investigando acerca del tema, y perfeccionar los resultados y conocimientos adquiridos.

Después de haber realizado las pruebas expuestas en este proyecto, he visto que podría ser útil la creación de herramientas que permitan configurar aspectos determinados de una red sin tener que pasar por sus interfaces gráficas, y hacerlo cómodamente desde un dispositivo como un Smartphone, algo que hoy en día está completamente normalizado en la vida diaria de las personas.

Finalmente, a pesar de que este trabajo de investigación ha finalizado, la idea la continuaré desarrollando, ya que me gustaría salvar los baches que me he encontrado, sobre todo el problema al montar la aplicación de prueba y poderla ejecutar en un Smartphone, problema que seguro requerirá investigar acerca del proceso de montaje de una aplicación, y qué pasa con las librerías que se usan, ya que aunque la aplicación se puede ejecutar en un ordenador, espero poder hacer que se pueda ejecutar en otros dispositivos para mayor comodidad por parte de los posibles usuarios.

7. Bibliografía

- **[1] Diagrama de Gartner**
  - https://www.gartner.com/en/research/methodologies/magic-quadrants-research

- **[2] Extreme Networks**
  - https://gtacknowledge.extremenetworks.com/
  - https://www.extremenetworks.com/
  - https://community.extremenetworks.com/extreme

- **[3] Arista**
  - https://www.arista.com/en/
  - https://www.arista.com/en/products/eos/eos-cloudvision

- **[4] Cisco**
  - https://developer.cisco.com/site/aci/
  - https://www.cisco.com/c/es_es/index.html
  - https://developer.cisco.com/site/aci/docs/apis/cobra-python/
  - https://github.com/datacenter/acitoolkit/tree/master/samples

- **[5] Aerohive Networks**
  - https://www.aerohive.com/

- **[6] Python**
  - https://www.python.org/

- **[7] Kivy**
  - https://kivy.org/
  - https://github.com/kivy/kivy/tree/master/examples/widgets
  - https://kivy.org/doc/stable/tutorials/pong.html
  - https://python-for-android.readthedocs.io/en/latest/quickstart/