

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN**



**GRADO EN INGENIERÍA EN TECNOLOGÍAS Y
SERVICIOS DE TELECOMUNICACIÓN**

TRABAJO FIN DE GRADO

**DESARROLLO DE UNA HERRAMIENTA PARA
EL DESPLIEGUE AUTOMATIZADO DE
ESCENARIOS DE RED VIRTUALIZADOS
APLICABLES A PLATAFORMAS CYBER RANGE**

SAMUEL GARCÍA SÁNCHEZ

2022

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN**



**GRADO EN INGENIERÍA EN TECNOLOGÍAS Y
SERVICIOS DE TELECOMUNICACIÓN**

TRABAJO FIN DE GRADO

**DESARROLLO DE UNA HERRAMIENTA PARA
EL DESPLIEGUE AUTOMATIZADO DE
ESCENARIOS DE RED VIRTUALIZADOS
APLICABLES A PLATAFORMAS CYBER RANGE**

**Autor:
SAMUEL GARCÍA SÁNCHEZ**

**Tutor:
MARIO SANZ RODRIGO
Departamento de Ingeniería de Sistemas Telemáticos**

2022

GRADO EN INGENIERÍA EN TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO DE FIN DE GRADO

Título: Desarrollo de una herramienta para el despliegue automatizado de escenarios de red virtualizados aplicables a plataformas Cyber Range.

Autor: D. Samuel García Sánchez.

Tutor: D. Mario Sanz Rodrigo.

Ponente: D. Manuel Álvarez-Campana Fernández-Corredor.

Departamento: Departamento de Ingeniería de Sistemas Telemáticos (DIT)

MIEMBROS DEL TRIBUNAL

Presidente:

Vocal:

Secretario:

Suplente:

Los miembros del tribunal nombrados acuerdan otorgar la calificación de:

Madrid, a de de 2022

Agradecimientos

A mi familia, en especial a mis padres, por la confianza que han depositado en mí durante esta larga etapa académica, brindándome su apoyo incondicional sobre todo en los peores momentos, que es cuando más lo necesité.

A Mario, por ayudarme y guiarme en todo momento durante la realización del trabajo.

No podía dejar de agradecer a mis amigos, por todos los buenos momentos que hemos pasado y sobre todo los que hemos sufrido, pero siempre juntos. Ha sido un proceso duro pero lo hemos conseguido.

Por último, agradecer también a todos los profesores de la escuela que han contribuido a mi desarrollo tanto profesional como personal, y que por tanto me han ayudado a ser quien soy hoy en día.

Gracias a todos.

Abstract

In this project, an analysis of the state of the art regarding tools for the deployment of IaaS (Infrastructure as a Service) is done with the aim of carrying out virtualized heterogeneous network deployments, applicable to CyberRange platforms for training and in the cybersecurity field. This deployment is focused on Terraform technology, with Cloud-based providers and Docker lightweight virtualization technology.

The main idea is, from high-level scenario descriptions (JSON, YAML, XML files...), parameterize and adapt these files to the Terraform deployment technology, avoiding the dependency associated with the physical environments where virtualized network scenarios will be deployed. The deployment refers both to the network topology, interconnections, and the provisioning of the final machines. Once the development is done, a catalog of scenarios to be virtualized automatically will be handled, which will be used to carry out cyber exercises.

The code used in this work is available in the following GitHub repository:

<https://github.com/samugs13/daerv>

Resumen

En este trabajo se lleva a cabo un análisis del estado del arte referente a herramientas para el despliegue de IaaS (Infrastructure as a Service) con el objetivo de realizar despliegues de red heterogéneos virtualizados, aplicables a plataformas CyberRange para la formación y entrenamiento en el campo de la ciberseguridad. Este despliegue se centra en la tecnología Terraform, con providers basados en Cloud y la tecnología de virtualización ligera Docker.

La idea principal es, a partir de la descripción de escenario a alto nivel (ficheros JSON, YAML, XML...), parametrizar y adaptar dichos ficheros a la tecnología de despliegue Terraform, evitando la dependencia asociada a los entornos físicos donde se desplegarán los escenarios de red virtualizados. El despliegue hace referencia tanto a la topología de red, interconexiones, como al aprovisionamiento de las máquinas finales. Una vez realizado el desarrollo, se manejará un catálogo de escenarios a virtualizar de forma automática, los cuales serán utilizados para la realización de ciberejercicios.

El código empleado en este trabajo está disponible en el siguiente repositorio de GitHub:

<https://github.com/samugs13/daerv>

Glosario

- **API:** Application Programming Interface
- **CLI:** Command Line Interface
- **ETSIT:** Escuela Técnica Superior de Ingenieros de Telecomunicación
- **FW:** Firewall
- **GCP:** Google Cloud Platform
- **GUI:** Graphical User Interface
- **HMI:** Human Machine INterface
- **IP:** Internet Protocol
- **IT:** Information Technology
- **IoT:** Internet of Things
- **LAN:** Local Area Network
- **NAT:** Network Address Translation
- **OS:** Operating System
- **OT:** Operational Technology
- **PC:** Personal Computer
- **PID:** Process Identifier
- **PLC:** Programmable Logic Controller
- **RCE:** Remote Code Execution
- **RTU:** Remote Terminal Unit
- **SCADA:** Supervisory Control And Data Acquisition
- **TFG:** Trabajo de Fin de Grado
- **VM:** Virtual Machine
- **VPC:** Virtual Private Cloud
- **VPN:** Virtual Private Network

Índice general

Agradecimientos	I
Abstract	II
Resumen	III
Glosario	IV
Índice de figuras	VIII
Índice de tablas	IX
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Estructura del documento	2
2. Estado del arte	3
2.1. Tecnologías de virtualización	3
2.1.1. Virtualización mediante hipervisor	4
2.1.2. Virtualización en contenedores	7
2.2. Tecnologías de aprovisionamiento	12
2.2.1. Aprovisionamiento estático	12
2.2.2. Aprovisionamiento dinámico	15
2.3. Tecnologías de orquestación	18
2.3.1. Docker Compose	18
2.3.2. Kubernetes	19
2.3.3. Terraform	20
2.4. Orquestación en Cloud	22
2.4.1. Google Cloud	22
3. Diseño	24
3.1. Requisitos y decisiones de diseño	24
3.2. Solución propuesta	25
3.3. Etapas de diseño	26
4. Desarrollo	27
4.1. Preparación del entorno	27

4.2. Escenarios de red	29
4.2.1. Smart Office 1	29
4.2.2. Smart Office 2	34
4.2.3. Smart Home	38
4.2.4. SCADA	42
5. Resultados y validación	46
5.1. Escenarios Smart Office	46
5.1.1. Despliegue de la infraestructura	46
5.1.2. Tests	49
5.2. Escenario Smart Home	49
5.2.1. Despliegue de la infraestructura	49
5.2.2. Tests	50
5.3. Escenario SCADA	51
5.3.1. Despliegue de la infraestructura	51
5.3.2. Tests	52
6. Conclusiones y líneas futuras	54
6.1. Conclusiones	54
6.2. Líneas futuras	55
Bibliografía	56
Anexos	58
A. Aspectos éticos, económicos, sociales y ambientales	59
B. Presupuesto económico	61
C. Ficheros de configuración del escenario Smart Office 1	64
D. Ficheros de configuración del escenario Smart Office 2	74
E. Ficheros de configuración del escenario Smart Home	87
F. Ficheros de configuración del escenario SCADA	94
G. Ficheros de aprovisionamiento	109

Índice de figuras

2.1. Tipos de hipervisor	4
2.2. Logo de KVM	5
2.3. Logo de VirtualBox	5
2.4. Logo de VMware	6
2.5. Contenedor vs VM: Estructura	8
2.6. Logo de LXC	8
2.7. Arquitectura del sistema de contenedores Docker	9
2.8. Logo de Docker	10
2.9. Ciclo de vida de los contenedores Docker	10
2.10. Docker vs LXC: Estructura	11
2.11. Creación de un contenedor Docker a partir de un Dockerfile	13
2.12. Representación de las capas de un Dockerfile	13
2.13. Logo de Vagrant	14
2.14. Comparativa de herramientas de gestión de la configuración	15
2.15. Arquitectura de funcionamiento de Chef	16
2.16. Arquitectura de funcionamiento de Ansible	17
2.17. Logo de Docker Compose	18
2.18. Arquitectura de Kubernetes	19
2.19. Logo de Terraform	21
2.20. Logo de Google Cloud	22
4.1. Selección de un proyecto en Google Cloud	27
4.2. Topología del escenario Smart Office 1	29
4.3. Implementación en GCP del escenario Smart Office 1	33
4.4. Topología del escenario Smart Office 2	34
4.5. Implementación en GCP del escenario Smart Office 2	37
4.6. Topología del escenario Smart Home	38
4.7. Implementación en GCP del escenario Smart Home	41
4.8. Topología del escenario SCADA	42
4.9. Implementación en GCP del escenario SCADA	45
5.1. Ejecución del comando <code>terraform init</code>	46
5.2. Prompt de confirmación tras la ejecución de <code>terraform apply</code>	47
5.3. Ejecución del comando <code>terraform state list</code>	47
5.4. Instancias desplegadas en Google Compute Engine - Smart Office	48
5.5. VPCs desplegadas - Smart Office	48
5.6. Reglas de FW aplicadas - Smart Office	48

5.7. Tests de conectividad - Smart Office	49
5.8. Instancias desplegadas en Google Compute Engine - Smart Home	49
5.9. VPCs desplegadas - Smart Home	49
5.10. Reglas de FW aplicadas - Smart Home	50
5.11. Tests de conectividad - Smart Home	50
5.12. Tests de aprovisionamiento - Smart Home	50
5.13. Instancias desplegadas en Google Compute Engine - SCADA	51
5.14. VPCs desplegadas - SCADA	51
5.15. Reglas de FW aplicadas - SCADA	52
5.16. Test de conectividad - SCADA	52
5.17. Test de aprovisionamiento - SCADA	53
B.1. Resumen de pagos en servicios de GCP	62

Índice de tablas

1.1. Objetivos planteados	2
3.1. Requisitos de diseño	24
3.2. Etapas de diseño	26
4.1. Estructura del escenario Smart Office 1	30
4.2. Reglas de FW del escenario Smart Office 1	31
4.3. Estructura del escenario Smart Office 2	35
4.4. Rutas del escenario Smart Office 2	36
4.5. Reglas de FW del escenario Smart Office 2	36
4.6. Estructura del escenario Smart Home	39
4.7. Reglas de FW del escenario Smart Home	40
4.8. Estructura del escenario SCADA	43
4.9. Reglas de FW del escenario SCADA	44
B.1. Horas asociadas a cada una de los subprocesos de realización del proyecto . .	61
B.2. Costes de mano de obra	62
B.3. Costes materiales	63
B.4. Costes indirectos	63
B.5. Costes totales	63

Capítulo 1

Introducción

En este capítulo se va a presentar el proyecto a fin de dar una primera impresión así como proporcionar una idea de lo que se va a ir detallando en capítulos posteriores.

1.1. Motivación

La creciente importancia que la digitalización ha alcanzado en el ámbito empresarial ha impactado directamente en las necesidades de ciberseguridad de las organizaciones. Buena parte de los ciudadanos, la mayoría de las empresas y casi la totalidad de los gobiernos son víctimas, a diario, de millones de ciberataques con un grado variable de sofisticación e impacto y, lo que es más preocupante, en su mayoría imperceptibles. La sustracción de información sensible o de datos de carácter personal, los ciberdelitos de naturaleza económica y la inutilización de sistemas militares, industriales, empresariales e incluso de infraestructuras críticas, son los principales objetivos de la gran mayoría de los ciberataques que acontecen hoy en día.

En este contexto, existe una creciente demanda de profesionales en el ámbito de la ciberseguridad por parte de gobiernos y empresas. La capacitación continua de estos profesionales es esencial para disponer de una ciberdefensa que permita establecer las medidas de seguridad apropiadas de los ciberespacios que protegen. Esta capacitación requiere de un nivel de innovación continuo únicamente proporcionado por entornos como los Cyber Range.

Un Cyber Range es una plataforma virtual que permite simular entornos operativos reales para la formación y el entrenamiento (individual o colectivo) de profesionales así como la experimentación, el testeo y la validación de nuevos conceptos, tecnologías, técnicas y tácticas de ciberseguridad y ciberdefensa.

Con el fin de hacer los Cyber Range lo más eficaces posible surge este trabajo de fin de grado. Se entiende por eficaz un Cyber Range que cumple los siguientes requisitos:

- Escenarios accesibles en tiempo y forma por los profesionales autorizados para su utilización de una forma muy sencilla.
- Escalabilidad y flexibilidad para poder responder a las necesidades de los responsables en materia de ciberseguridad y ciberdefensa en función de la naturaleza de las actividades que lleven a cabo.

- Entorno seguro que permita a los usuarios ejecutar las actividades sin poner en riesgo los sistemas en producción e información clasificada o sensible.

Estos aspectos contribuyen a una mejor formación y entrenamiento de los profesionales de seguridad, a la par que facilitan el cumplimiento de las estrategias de ciberseguridad.

1.2. Objetivos

El objetivo de este trabajo es realizar despliegues de red heterogéneos virtualizados, aplicables a plataformas Cyber Range para la formación y entrenamiento en el campo de la ciberseguridad. El trabajo se centra principalmente en el despliegue de la infraestructura así como su interconexión, y no en la configuración a fondo de todos los elementos para la realización de tests de intrusión específicos.

Para alcanzar el objetivo final, se han identificado una serie de subobjetivos, desde lo más básico hasta lo más complejo, que en conjunto permitirán obtener el resultado deseado:

Id	Objetivo del TFG
O1	Determinar los escenarios a desplegar, intentando que estos reflejen situaciones lo más realistas posible.
O2	Posibilitar un despliegue de infraestructura que sea modelable mediante variables, escalable, portable y seguro.
O3	Diseñar y configurar la comunicación entre los elementos que componen los escenarios.
O4	Desarrollar una herramienta que sirva como base para la configuración software de dichos escenarios.

Tabla 1.1: Objetivos planteados

1.3. Estructura del documento

El presente documento se secciona en capítulos. A fin de tener una visión global de las distintas fases, se muestra a continuación cada uno de ellos junto a una breve descripción:

Capítulo 1. Es la introducción al TFG, donde se proporciona una visión global y se describe la motivación de este así como los objetivos a conseguir.

Capítulo 2. Presenta el estado del arte, resumiendo las tecnologías empleadas en el proyecto y su comparación con opciones alternativas.

Capítulo 3. Diseño de la solución a implementar, basada en los requisitos identificados. Presentación de las diferentes etapas a seguir para ello.

Capítulo 4. Se desarrolla la lógica seguida para implementar cada uno de los escenarios.

Capítulo 5. Resumen de los resultados obtenidos tras la fase de desarrollo reforzado con tests que lo prueban.

Capítulo 6. Conclusiones y problemas surgidos a raíz de la realización del proyecto. Líneas futuras sobre las que trabajar.

Capítulo 2

Estado del arte

En este capítulo se van a presentar distintas tecnologías de virtualización, aprovisionamiento y despliegue aplicables al proyecto. Estas tecnologías se analizan y comparan para posteriormente seleccionar aquellas que mejor se adecuen a los objetivos que se pretenden conseguir.

2.1. Tecnologías de virtualización

Se podría decir que la virtualización es ya uno de los pilares fundamentales del mundo IT debido a las grandes ventajas que proporciona. Previo al desarrollo de las tecnologías y tipos de virtualización disponibles, es conveniente explicar en qué consiste la virtualización, que no es más que una representación mediante software de un entorno físico o recurso tecnológico, como pueden ser aplicaciones, servidores o almacenamiento. [1]

Gracias a esta tecnología, es posible contar con varios ordenadores virtuales en el mismo hardware, donde cada uno de ellos puede interactuar de forma independiente y ejecutar sistemas operativos o aplicaciones diferentes mientras comparten los recursos de una sola máquina host. Al crear varios recursos a partir de un único equipo o servidor, la virtualización mejora la escalabilidad y las cargas de trabajo, al tiempo que permite usar menos servidores y reducir el consumo de energía, los costos de infraestructura y el mantenimiento.

En función del sistema a simular, podemos encontrar diferentes categorías [2], un ejemplo es la virtualización de red, que consiste en crear redes virtuales sobre redes físicas o reproducir completamente redes físicas en software. Otro ejemplo sería la virtualización de almacenamiento, que combina varios dispositivos de almacenamiento en red, con la apariencia de una única unidad o dispositivo de almacenamiento, accesible por varios usuarios. Podríamos enumerar más tipos de virtualización, pero en lo que a este trabajo respecta vamos a centrarnos en la virtualización de software, que separa las aplicaciones del hardware y el sistema operativo, y en la que distinguimos dos subtipos: virtualización mediante hipervisor y virtualización en contenedores.

2.1.1. Virtualización mediante hipervisor

Una máquina virtual es un software que ejecuta programas o procesos como si fuera la máquina física. Es decir, se abstrae el hardware y se representa con una capa de software que proporciona una interfaz igual que el hardware, de forma que sobre ella podemos instalar uno o varios sistemas operativos invitados o *guests* distintos. Esta capa de software también se encarga de repartir y aislar los recursos del host entre las VM, de manera que el host queda protegido si falla una VM, y las VM están protegidas entre ellas. Pues bien, cuando hablamos de esta capa de software estamos hablando de lo que se conoce como hipervisor.

Como ya se ha mencionado, un hipervisor es una capa intermedia de software que permite al ordenador anfitrión prestar soporte a varias máquinas virtuales mediante el uso compartido de sus recursos. Cuando se ejecuta una instrucción en el OS invitado, el hipervisor la coje y la ejecuta en el OS anfitrión. En este proceso, el OS no diferencia entre ejecutar procesos en la máquina virtual o en la física, lo que representa plenamente el concepto de virtualización.

Dentro de los hipervisores [3], podemos distinguir dos tipos. El primero es el Tipo 1, conocido también como hipervisor nativo o *bare-metal*. Este hipervisor se ejecuta directamente sobre el hardware en lugar de un OS clásico. Todos los hipervisores necesitan algunos elementos del sistema operativo (por ejemplo, el administrador de memoria, el programador de procesos, la pila de entrada o salida [E/S], los controladores de dispositivos, entre otros) para ejecutar las máquinas virtuales. Por tanto, este hipervisor es equivalente a un OS con un poco de información adicional que le permite gestionar los OS invitados. Es muy común encontrarlos en centros de datos, por la eficiencia que supone el ahorrar una capa de software.

Los hipervisores de Tipo 2 se ejecutan sobre el OS anfitrión como una capa de software o aplicación. Están orientados a usuarios individuales que buscan ejecutar varios OS en el mismo ordenador. La ejecución de una VM sobre un hipervisor de este tipo es más lenta que en un hipervisor de Tipo 1.

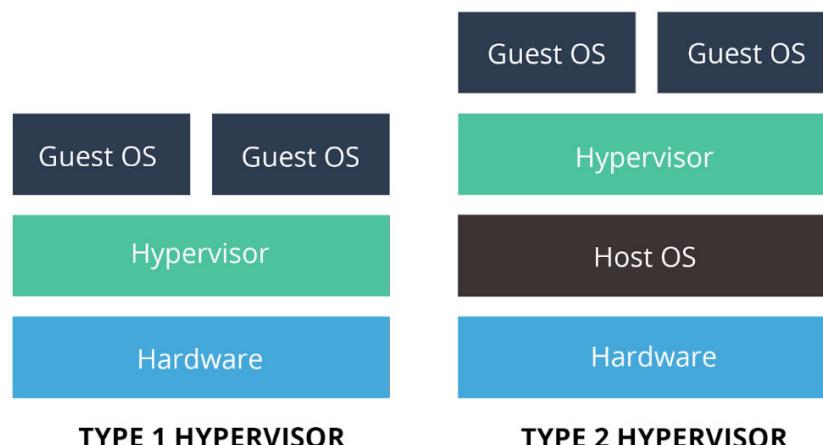


Figura 2.1: Tipos de hipervisor

A continuación se presentan algunas tecnologías que emplean este tipo de virtualización.

2.1. TECNOLOGÍAS DE VIRTUALIZACIÓN

KVM

KVM (Kernel Virtual Machine) [4] es una tecnología de virtualización open source que convierte el kernel de Linux en un hipervisor de Tipo 1 que se puede usar para la virtualización. Las KVM tienen todos los elementos necesarios de un OS porque forman parte del kernel de Linux. Cada máquina virtual se implementa como un proceso habitual de Linux. Al ser un hipervisor de Tipo 1 ofrece un mejor rendimiento.

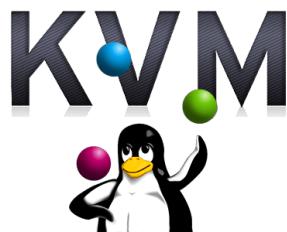


Figura 2.2: Logo de KVM

La configuración de la máquina virtual creada se almacena internamente en un fichero XML, el cual es posible editar manualmente a posteriori si se quiere hacer algún cambio. KVM nos permite disfrutar de las ventajas del software open source: no habrá restricciones en cuanto a integración, como sí puede haberlas si se usa un software propietario como VMware; y es independiente de proveedores. Es posible instalar una GUI como virt-manager, que se apoya en la biblioteca libvirt (API de virtualización estándar de Linux), para facilitar su uso.

VirtualBox

VirtualBox [5] es desarrollado por Oracle, aunque es gratuito y open source al igual que KVM. Es un hipervisor de Tipo 2, por lo que ofrece un rendimiento inferior comparado con un Tipo 1. VirtualBox permite crear y cargar máquinas virtuales de una forma muy sencilla, lo que hace que sea la alternativa elegida por muchos usuarios. Su asistente ofrece algunos valores sugeridos para tipos específicos de máquinas virtuales durante la creación de estas, pero su gestión final se produce en una configuración posterior.



Figura 2.3: Logo de VirtualBox

Una ventaja de VirtualBox son las instantáneas, que permiten tomar una imagen de la máquina virtual en un momento dado. La imagen conserva la máquina virtual, lo que permite volver a ese momento específico.

VirtualBox ofrece un soporte muy completo. Tiene versiones para Windows, Linux, Macintosh y Solaris, y puede ejecutar un amplísimo número de sistemas operativos invitados, incluidos Windows, macOS, Linux, DOS, Solaris u OpenBSD.

VMware

VMware [6] es un software comercial, lo que significa que si queremos aprovechar al máximo todas sus herramientas y configuraciones, debemos pagar por la licencia de uso. Aquí vamos a hablar de VMware Workstation Player, que es el producto gratuito de VMware para virtualización de máquinas, orientado para uso personal, doméstico y sistema educativo.



Figura 2.4: Logo de VMware

En comparación con VirtualBox, VMware Workstation Player es una experiencia más fluida y ágil, y ofrece mejor soporte y estabilidad para una amplia gama de hardware. Está disponible para Windows y Linux y también admite todo tipo de sistemas invitados. [7]

Además, VMware Workstation Player sí permite personalizar toda la configuración durante el proceso de creación de la máquina virtual. La diferencia no es mucha, pero significa que la máquina virtual está lista para ejecutarse después de finalizar el asistente, en lugar de tener que realizar más configuraciones una vez que se completa.

Por el contrario, no admite instantáneas o puntos de control. Puede suspender temporalmente el sistema operativo invitado para reanudar desde un punto específico, pero no es tan completo como la creación de un historial de imágenes para la máquina virtual.

2.1.2. Virtualización en contenedores

La virtualización basada en contenedores, también llamada virtualización del sistema operativo, es una aproximación a la virtualización en la cual la capa de virtualización se ejecuta como una aplicación en el sistema operativo.

Un contenedor [8] es un conjunto de uno o más procesos aislados del resto del sistema, que acceden sólo a los recursos que se indican. El contenedor encapsula el programa específico y las librerías, mientras que utiliza el sistema operativo del host. Podemos distinguir entre dos tipos de contenedores: [9]

- **A nivel de sistema operativo:** un sistema operativo completo se ejecuta en un espacio aislado dentro de la máquina host, compartiendo el mismo kernel.
- **A nivel de aplicación:** una aplicación o servicio, y los procesos mínimos requeridos por esa aplicación, se ejecutan en un espacio aislado dentro de la máquina host.

Al compartir el mismo kernel del sistema operativo host, un contenedor sólo puede ejecutar procesos en ese sistema operativo. Es decir, un contenedor que se ejecuta en un servidor de Linux, por ejemplo, solo puede ejecutar un sistema operativo Linux, mientras que tal y como habíamos comentado, un hipervisor emula el hardware, lo que permite que varios sistemas operativos (Windows o Linux) se ejecuten simultáneamente en un solo sistema. Además, esta compartición del núcleo hace que el nivel de aislamiento sea menor comparado con las máquinas virtuales, ya que al acceder todos los contenedores al mismo núcleo, si se explota una vulnerabilidad en el núcleo ésta afectaría a todo el sistema, incluidos todos los contenedores.

A cambio, con la virtualización basada en contenedores, no existe la sobrecarga asociada con tener a cada huésped ejecutando un sistema operativo completamente instalado. Este enfoque también puede mejorar el rendimiento porque hay un solo sistema operativo encargándose de los avisos de hardware.

Tal y como se ha mencionado previamente, los contenedores hacen uso del kernel del sistema operativo, del que cabe destacar las siguientes características necesarias para el correcto funcionamiento de este tipo de virtualización: [10]

- **Grupos de control o cgroups.** Se encargan de gestionar los recursos del ordenador asignados a un proceso o conjunto de procesos, como pueden ser el número de *slices* de tiempo de CPU asignadas a cada proceso, el límite de memoria a usar por proceso, los dispositivos de bloques de E/S... Es posible estructurar los recursos en jerarquía.
- **Namespaces.** Permiten aislar procesos entre sí. Los procesos de un contenedor se asignan a un namespace, y el sistema operativo aísla los recursos entre ese namespace y el resto. Hay diferentes tipos de namespace: de PID, que permiten mantener los mismos PIDs en diferentes contenedores; mount, para separar sistemas de ficheros; de red para aislar controladores de red...

Los contenedores, por tanto, suponen una virtualización más ligera: usan menos recursos que las máquinas virtuales, además de proporcionar mayor flexibilidad y rapidez de arranque y despliegue. Esto último se debe a que, al usar el mismo kernel que el host, sólo están iniciando procesos, algo que es casi instantáneo y que hace muy fluida la ejecución de diferentes máquinas a la vez.

Para contextualizar con el apartado 2.1.1, se muestra una imagen comparativa de la estructura de un contenedor respecto a una máquina virtual:

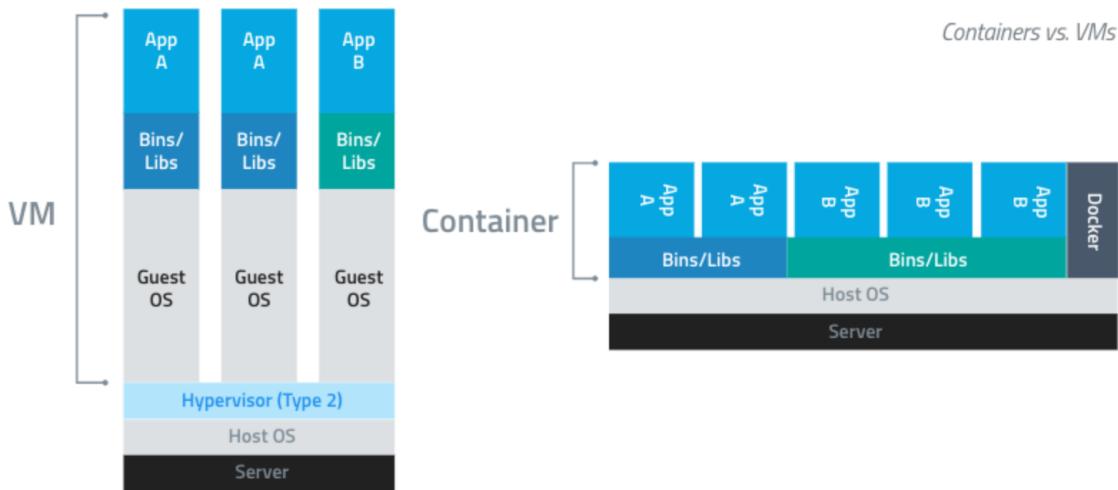


Figura 2.5: Contenedor vs VM: Estructura

LXC

LXC (Linux Containers) [11] es una plataforma de código abierto de contenedores a nivel de sistema operativo. Este tipo de contenedores hacen que un único host Linux actúe como varios host Linux. Esto es debido a que los contenedores LXC incluyen un sistema Linux prácticamente completo, similar a una VM, con su propio sistema de ficheros, espacio de red y aplicaciones. Su objetivo es recrear un entorno lo más parecido posible a una instalación de Linux, pero sin la necesidad de un kernel independiente.



Figura 2.6: Logo de LXC

Chroot es un comando UNIX que permite ejecutar un proceso bajo un directorio raíz simulado, de manera que el proceso no puede acceder a archivos fuera de ese directorio. LXC es similar a un chroot, pero ofrece mucho más aislamiento.

Para crear diferentes contenedores de sistema operativo, se emplean plantillas o templates. Las plantillas proporcionadas en LXC son scripts específicos de un sistema operativo.

LXC se suele usar junto a LXD. LXD ofrece una interfaz para gestionar contenedores LXC como si fueran máquinas virtuales, proporcionando snapshots y control de imágenes, además de otras funcionalidades que incrementan el potencial de LXC. Una de las ventajas principales de LXC es que es una tecnología sencilla de manejar.

2.1. TECNOLOGÍAS DE VIRTUALIZACIÓN

Docker

Docker [12] es uno de los proyectos más conocidos y utilizados en este tipo de virtualización. Lejos de ser un sistema operativo como tal, esta plataforma de código abierto hace uso de las funciones de aislamiento de recursos del kernel de Linux para dar lugar a contenedores independientes. Está basada en Linux, pero en los últimos años se ha producido un desarrollo y actualmente es posible su uso en Windows.

Los contenedores que proporciona Docker son a nivel de aplicación (puede ejecutar aplicaciones normales sin incluir un sistema operativo completo), y su implementación está basada en imágenes, lo que permite compartir una aplicación o un conjunto de servicios, con todas sus dependencias, en varios entornos.

Docker utiliza una arquitectura cliente-servidor. En este tipo de arquitectura, las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y quienes demandan esos recursos o servicios, que son los clientes. Un sistema de contenedores Docker se compone principalmente de 5 elementos: [13]

- **Demonio:** es el proceso principal de la plataforma.
- **Cliente:** binario que constituye la interfaz y que permite al usuario interactuar con el Demonio mediante CLI.
- **Imagen:** plantilla utilizada para crear el contenedor para la aplicación que queremos ejecutar.
- **Registros:** directorios donde se almacenan las imágenes, tanto de acceso público como privado. El más común es Docker Hub.
- **Contenedores:** carpetas donde se almacena todo lo necesario (librerías, dependencias, binarios, etc) para que la aplicación pueda ejecutarse de forma aislada.

Docker Engine es la aplicación cliente-servidor responsable de iniciar y parar los contenidos de una manera similar a como lo hace el hipervisor en una máquina virtual. A continuación se muestra una figura donde se muestran de forma gráfica las interacciones entre los ya mencionados componentes:

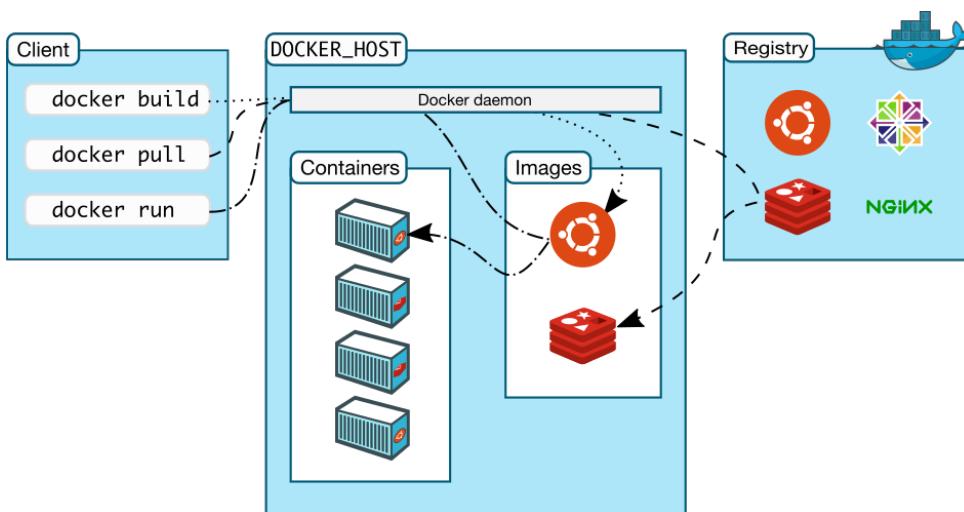


Figura 2.7: Arquitectura del sistema de contenedores Docker

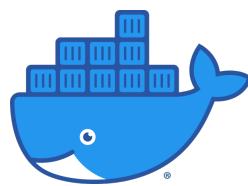


Figura 2.8: Logo de Docker

En el ciclo de vida de un contenedor Docker podemos distinguir 5 estados principales:

- **Created:** hace referencia a un contenedor que ha sido creado pero no arrancado.
- **Running/Started:** contenedor corriendo con todos sus procesos.
- **Paused:** contenedor cuyos procesos se han pausado usando la señal SIGSTOP de cgroups.
- **Stopped/Exited:** contenedor cuyos procesos se han parado. La diferencia con *paused* es que se libera la memoria que estaban usando los procesos que han sido detenidos, usando la señal SIGKILL de cgroups. El sistema de ficheros se mantiene tal y como estaba en el momento que se detuvo el contenedor.
- **Deleted/Dead:** contenedor eliminado. Es posible recuperarlo durante un periodo de tiempo determinado.

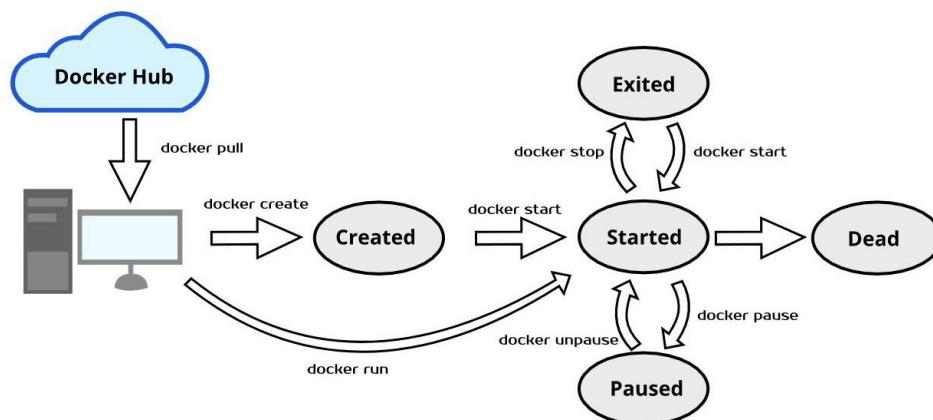


Figura 2.9: Ciclo de vida de los contenedores Docker

Como se puede apreciar en la figura, Docker permite gestionar el estado de los contenedores de forma sencilla mediante la CLI, algunas de sus órdenes más destacadas son:

```

1 $ docker pull #permite descargar una imagen de un repositorio
2 $ docker create #crea un nuevo contenedor, pero no lo arranca
3 $ docker start #arranca contenedores parados
4 $ docker run #es equivalente a create + start
5 $ docker stop #detiene los procesos corriendo en un contenedor
6 $ docker pause #pausa los procesos corriendo en un contenedor
7 $ docker unpause #reanuda la ejecucion de los procesos pausados
8 $ docker rm #elimina los procesos y el contenedor donde corrían
    
```

2.1. TECNOLOGÍAS DE VIRTUALIZACIÓN

Diferencias entre LXC y Docker

Con todo esto, podemos sacar varias conclusiones importantes acerca de estos dos tipos de contenerización, que condicionarán la elección de uno u otro en función de su uso. La principal sería que mientras que con LXC se virtualiza un sistema operativo completo, con Docker se virtualizan aplicaciones. Además, los contenedores LXC sólo permiten virtualizar entornos Linux y no se pueden portar entre máquinas, mientras que Docker permite portar entre máquinas e incluso plataformas. Esto último es algo relevante puesto que si en algún momento fuese necesario migrar a otro entorno, gracias a la portabilidad de Docker nos ahorraríamos el tener que instalar en este nuevo entorno todas aquellas aplicaciones que normalmente usemos.

Finalmente, también cabe recalcar que LXC proporciona un menor aislamiento del sistema operativo respecto a Docker. Al virtualizar sistemas operativos completos y aislados dentro del mismo host, debe haber usuario root y llamadas al propio sistema operativo (si no fuese así, estaríamos ante un sistema «recortado» porque no se podrían hacer determinadas cosas). Docker sirve para virtualizar aplicaciones dentro de un mismo host, por lo que el nivel de acceso root sí puede estar más limitado y controlado, lo que lo hace más seguro.

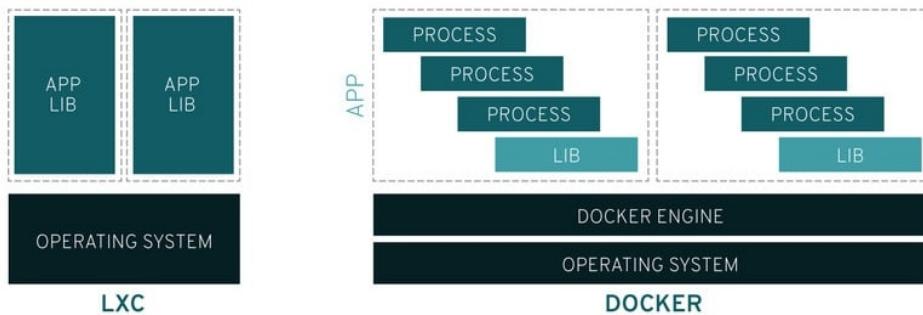


Figura 2.10: Docker vs LXC: Estructura

2.2. Tecnologías de aprovisionamiento

Antes de entrar en materia, se va a presentar la infraestructura como código, un concepto fundamental para el desarrollo de este trabajo.

La infraestructura como código (IaC) permite gestionar y preparar la infraestructura de un entorno con código, en lugar de hacerlo mediante procesos manuales. Con este tipo de infraestructura, se crean archivos de configuración que contienen las especificaciones que esta necesita, lo cual facilita la edición y la distribución de las configuraciones. Asimismo, garantiza que siempre se despliegue el mismo entorno. [14]

La IaC se puede abordar mediante dos enfoques: uno declarativo o uno imperativo:

- **Enfoque declarativo:** define el estado deseado del sistema, incluidas las propiedades que debe tener y los recursos necesarios, y la herramienta de IaC se encarga de configurarlo.
- **Enfoque imperativo:** define los comandos específicos para lograr la configuración deseada, los cuales se deben ejecutar en el orden correcto.

Las tecnologías de virtualización presentadas en el apartado anterior recibirán órdenes del orquestador de escenario (herramienta IaC que veremos más adelante) para desplegar las máquinas necesarias para formar un escenario de red determinado.

Por tanto, podemos decir que el aprovisionamiento consiste en la instalación y la configuración del software (incluido el sistema operativo y las aplicaciones) necesario para que dichas máquinas puedan desempeñar su función en el escenario de red. No es lo mismo que la configuración, aunque ambos son pasos en el proceso de implementación.

Dentro del aprovisionamiento podríamos diferenciar dos tipos, que están muy ligados a los distintos enfoques de la IaC. El primero sería un aprovisionamiento “en frío” o estático, en el que se abastece la máquina antes de arrancarla y levantar el escenario. Por otro lado tendríamos el aprovisionamiento “en caliente” o dinámico, cuya idea principal reside en lanzar órdenes, comandos o depositar archivos en una máquina ya levantada y arrancada.

2.2.1. Aprovisionamiento estático

Este tipo de aprovisionamiento tiene un enfoque imperativo, y, como ya se ha mencionado, se basa en crear de manera offline la máquina con todo lo necesario ya instalado, de forma que cuando se despliegue sólo requiera algunas pequeñas modificaciones adicionales, como podrían ser configuraciones de red. A continuación se va a detallar el uso de Docker y Vagrant para el aprovisionamiento, pero hay otras alternativas a tener en cuenta como las ISO/OVA de VirtualBox.

Docker

Habíamos visto, en el apartado de virtualización basada en contenedores, que una imagen Docker es una instantánea de la aplicación o servicio que corre en un contenedor y de su configuración y las dependencias. En otras palabras, una imagen es un archivo, compuesto por múltiples capas, que constituye una representación estática del estado de un contenedor.

2.2. TECNOLOGÍAS DE APROVISIONAMIENTO

Estas imágenes son las plantillas base desde la que partimos ya sea para crear una nueva imagen o crear nuevos contenedores para ejecutar las aplicaciones. Las imágenes pueden ser locales si se almacenan en el host, o remotas si se suben a repositorios públicos como DockerHub, donde cualquier usuario que lo desee puede hacer uso de ellas. En los repositorios se pueden encontrar tanto imágenes oficiales desde las que partir como imágenes de terceros ya modificadas.

Para crear nuestras propias imágenes, donde dotemos a nuestros contenedores del software así como de su configuración necesaria, Docker nos proporciona los Dockerfile. Un Dockerfile es un archivo de texto plano que contiene una serie de instrucciones necesarias para armar una imagen. A partir de esa imagen podemos crear varias instancias o contenedores, que integran todo lo necesario para ejecutar una aplicación.

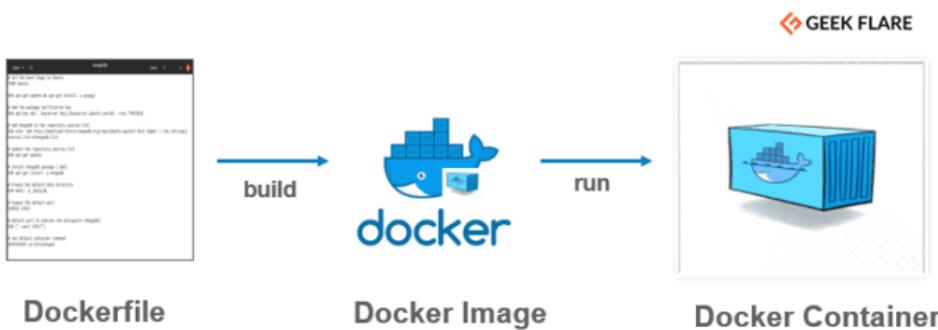


Figura 2.11: Creación de un contenedor Docker a partir de un Dockerfile

Una imagen de Docker consiste en una serie de capas de “solo-lectura”, cada una de las cuales se representa con una instrucción del Dockerfile. Las capas se amontonan unas sobre otras y cada una añade algo sobre la anterior. Cuando creamos un contenedor, que no es más que una instancia en ejecución de una imagen, estamos añadiendo una capa escribible encima de todas las demás capas de solo-lectura. Así, la estructura de una imagen Docker se podría representar como sigue:

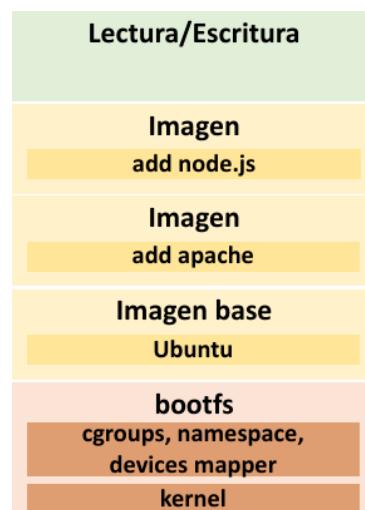


Figura 2.12: Representación de las capas de un Dockerfile

La imagen tendrá una base mínima, que han de tener todas las imágenes para el correcto funcionamiento del contenedor. Esta base incluye el kernel y algunas de sus características necesarias de las que ya hemos hablado como cgroups y namespaces. Encima de la base se representan las capas de sólo lectura, que serían las instrucciones del Dockerfile. En este ejemplo se parte de una imagen Ubuntu a la que se le instala nodejs y apache. Finalmente al crear el contenedor con la instrucción docker run se añade la última capa, correspondiente a los sistemas de ficheros de lectura/escritura, sobre el resto de capas, de forma que podemos interactuar con el contenedor.

Vagrant

Vagrant [15] es una herramienta de línea de comandos que permite crear y configurar máquinas virtuales a partir de ficheros de configuración llamados Vagrantfile.



Figura 2.13: Logo de Vagrant

Al poseer estos ficheros de configuración, que nos permiten definir los servicios a instalar así como también sus configuraciones, se centraliza toda la configuración de la VM que creamos, de forma que es posible utilizar el mismo Vagrantfile para crear una VM exactamente igual cuantas veces se quiera. Esto es algo muy ventajoso ya que nos permite ahorrar la carga de trabajo que supone el desplegar el mismo entorno una y otra vez, con la seguridad de que nuestro entorno siempre tendrá la misma configuración.

Cabe destacar que vagrant no tiene la capacidad para correr una máquina virtual sino que simplemente se encarga de definir las características con las que debe crearse esa VM y los complementos a instalar. Para poder trabajar con las máquinas virtuales es necesario la instalación de VirtualBox, Docker, Hyper-v, o la tecnología de virtualización que se desee (y sea compatible¹).

Vagrant permite realizar algunas configuraciones de red, pero no está pensado para trabajar con grandes cantidades de máquinas virtuales, para infraestructuras más complejas existen otras herramientas como Terraform que pertenece a la misma empresa HashiCorp, y que vamos a presentar posteriormente como una de las tecnologías de orquestación de escenarios.

En resumen, Vagrant es muy fácil de instalar y utilizar, permitiéndonos configurar entornos locales solo con un par de comandos. Además, nos proporciona la seguridad de que todos los entornos que creamos con el mismo Vagrantfile serán iguales. Como desventaja, es importante que todos los comandos a ejecutar no necesiten la interacción del usuario, ya que si no, va a fallar.

¹Ver en la documentación oficial [15]

2.2.2. Aprovisionamiento dinámico

El enfoque imperativo de las herramientas presentadas en el apartado anterior, así como de los tradicionales scripts en bash o python empleados para automatizar configuraciones, donde se detallan todas las instrucciones necesarias para llegar a la configuración deseada, puede desembocar en errores de ejecución.

Imaginemos un script en el que uno de los pasos para lograr la configuración deseada es crear un directorio. La primera vez que se ejecute, este script funcionará correctamente, pero, si lo volvemos a ejecutar, lo más seguro es que aparezca un error debido a que el directorio ya se creó al ejecutar el script por primera vez, y estaríamos intentando crear un directorio ya existente.

Para evitar este problema y derivados provocados por errores humanos, es muy común el empleo de un aprovisionamiento dinámico, mediante herramientas de gestión de configuración. La principal característica que presentan estas herramientas es lo que se conoce como idempotencia, que se define como la propiedad para realizar una acción determinada varias veces y aún así conseguir el mismo resultado que se obtendría si se realizase una sola vez.

	 puppet	 CHEF	 ANSIBLE
Programado en	Ruby	Ruby	Python
Arquitectura	Agente	Agente	Serverless
SW Cliente	Sí	Sí	SSH
Lenguaje	Ruby	Ruby	YAML
Configuración	Modules	Cookbooks	Playbooks
Unidades	Modules	Recipes	Tasks

Figura 2.14: Comparativa de herramientas de gestión de la configuración

Esto es debido a que las herramientas de gestión de la configuración usan un lenguaje declarativo, en el que simplemente hay que especificar el estado final deseado, y la propia herramienta llevará a cabo las acciones necesarias para llegar a él, que variarán en función del estado en el que se encuentre la máquina. De esta forma se agilizan los cambios y las implementaciones, se elimina la posibilidad de que se cometan errores humanos, y la gestión del sistema se torna predecible y ajustable. [16]

Además, estas herramientas posibilitan también tener la configuración de una infraestructura TI en forma de código, lo que da lugar a que una infraestructura sea:

- **Escalable.** La automatización del proceso de configuración ofrece la ventaja de poder aplicarlo tanto a infraestructuras pequeñas como de gran tamaño.
- **Replicable.** El código puede ser replicado y a partir de él generar la infraestructura, de manera que se adapten según las necesidades y entornos disponibles.

Actualmente existen una gran variedad de herramientas de este tipo (ver figura 2.14) con diferentes características.

Chef

El software Chef [17] es una herramienta de gestión de la configuración de código abierto que permite crear partes de una infraestructura como un servicio. Para ello hace uso de tres elementos principales: [18]

- **Recipes.** Una receta es un fichero escrito en Ruby que contiene el código que va a realizar operaciones sobre las máquinas, en este caso sería la gestión de software y configuración, aunque también podría realizarse el despliegue. Una receta puede depender o estar contenida en otra.
- **Cookbooks.** Un libro de cocina es la unidad fundamental de configuración y distribución de políticas. Define un escenario y contiene todo lo necesario para que se de ese escenario: recetas que especifican los recursos a utilizar, atributos para sobrescribir la configuración predeterminada de una máquina, ficheros estáticos para administrar archivos de configuración...
- **Knife.** Desde las estaciones de trabajo o workstations se gestionan las configuraciones antes de enviarlas al Chef Server. Para ello se emplea una herramienta de línea de comandos llamada Knife que permite interactuar mediante SSH con el servidor Chef. Con Knife se puede subir, descargar o eliminar cookbooks de Chef Server, instalar Chef Client en dispositivos (crear nodos), consultar datos acerca de nodos y cookbooks etc.

Chef es de tipo cliente-servidor, por lo que es necesaria la instalación y configuración del software Chef Client en los nodos o dispositivos (ya sean físicos, virtuales, o en la nube) que se quieran aprovisionar. Una vez instalado, todos los nodos gestionados se comunican con el servidor principal a través del uso de certificados. El elemento principal y núcleo es el llamado Chef Server, desde el cual es posible centralizar el proceso de lanzamiento y configuración de las máquinas haciendo uso de los cookbooks y recetas almacenadas en él.

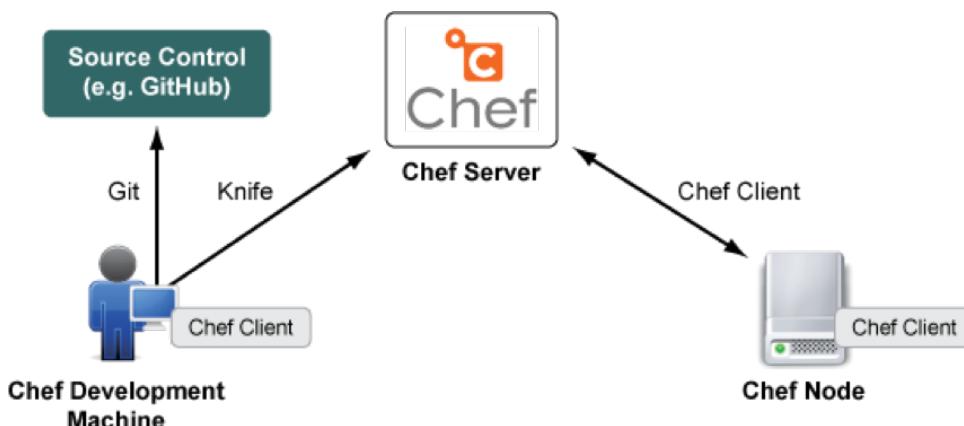


Figura 2.15: Arquitectura de funcionamiento de Chef

2.2. TECNOLOGÍAS DE APROVISIONAMIENTO

Ansible

Ansible [19], al igual que Chef, es un motor open source que automatiza los procesos para preparar la infraestructura, gestionar la configuración, implementar las aplicaciones y organizar los sistemas, entre otros procedimientos de TI.

Ansible está escrito en Python, y presenta algunas ventajas respecto a Chef. En primer lugar es serverless, lo que significa que no hay que instalar agentes ya que la configuración se lleva a cabo por SSH. Además, la configuración se realiza en YAML, un lenguaje más sencillo de aprender y usar que Ruby. Todo esto hace que la curva de aprendizaje en Ansible sea considerablemente menor que la de Chef y Puppet.

En Ansible también podemos destacar tres elementos principales:

- **Inventario.** Es un fichero YAML que contiene el grupo de máquinas sobre las que se va a realizar una acción determinada.
- **Task.** Una tarea o task es la acción específica a realizar sobre las máquinas (el estado en el que queremos que quede el sistema). Sólo es necesario declarar el estado final, ya que como se había comentado estas herramientas tienen un enfoque declarativo. Cada tarea no es más que una llamada a un módulo de Ansible.
- **Playbook.** un playbook es un fichero YAML que contiene una lista de plays o actos, siendo un play una lista de tareas o tasks que se aplican sobre un determinado conjunto de hosts (inventario).

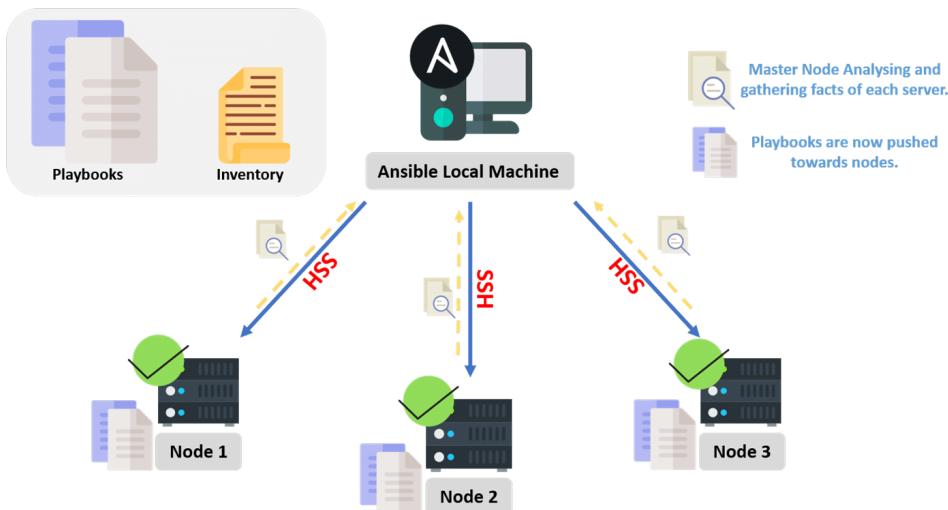


Figura 2.16: Arquitectura de funcionamiento de Ansible

La ventaja principal de Ansible frente al resto de herramientas de configuración es su facilidad de uso. Para su funcionamiento es necesario que el cliente cuente con SSH y Python, que normalmente suelen estar instaladas por defecto en los sistemas Linux.

2.3. Tecnologías de orquestación

En el pasado, la preparación de la infraestructura de TI se llevaba a cabo de forma manual e incluía la instalación de servidores físicos y la configuración del hardware según los ajustes deseados. Si se necesitaba más capacidad, se tenía que solicitar más hardware, esperar a que llegara y luego instalarlo y prepararlo. [20]

En la actualidad, la infraestructura suele definirse en el software. La virtualización y los contenedores agilizaron los procesos de preparación y eliminaron la necesidad de preparar y gestionar sistemas de hardware con frecuencia. Al igual que el aprovisionamiento de las máquinas, la preparación del escenario también puede automatizarse.

Las herramientas de orquestación de escenario son una tecnología IaC mediante la cual es posible definir vía fichero qué elementos a nivel de red/arquitectura se van a desplegar: máquinas, VLANs, recursos dedicados (como cantidad de CPU y memoria), etc.

Al presentar las tecnologías de virtualización y aprovisionamiento hablamos de despliegue, que es el hecho de levantar y configurar (por ejemplo, a nivel interconexión de red) las máquinas que van a formar ese escenario virtualizado. Por tanto, si se combina la orquestación de escenario junto con el despliegue, ambas tecnologías IaC, podremos realizar el despliegue automatizado de escenarios de red virtualizados.

2.3.1. Docker Compose

Docker Compose [21] es una herramienta para definir y ejecutar múltiples contenedores Docker de forma simultánea en un mismo host. Utiliza un archivo YAML (normalmente de nombre `docker-compose.yml`) para configurar los servicios de la aplicación. Luego, con un solo comando, crea e inicia todos los servicios desde su configuración. Si bien en algunos casos se puede decir que es un orquestador, no es del todo comparable a los orquestadores que veremos a continuación (como ECS, Kubernetes, etc) ya que pese a que puede ejecutar varios servicios distintos, no puede manejar autoscaling, downtime etc.

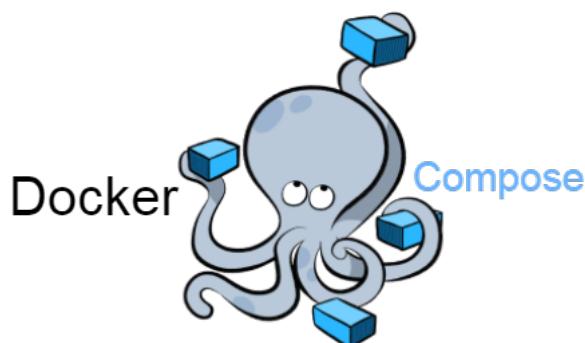


Figura 2.17: Logo de Docker Compose

El fichero Dockerfile del que hablamos en el apartado 2.2 define como crear la imagen de una aplicación o contenedor, mientras que el fichero `docker-compose.yml` nos permite vincular y configurar estos contenedores en conjunto para construir varios servicios. La forma en que ejecutamos nuestro docker compose es con la instrucción `docker-compose up`, el cual entrega instrucciones para ejecutar el contenedor según el `docker-compose.yml`.

2.3.2. Kubernetes

Docker Compose nos permite desplegar múltiples contenedores en un mismo host. Es posible que, por motivos de disponibilidad entre otros, quisieramos tener nuestros contenedores ejecutándose en diferentes hosts. Esto, entre otras cosas (como que los host que alojan estos contenedores estén en la nube), es algo que nos permite Kubernetes.

Kubernetes [22] es un orquestador open source para aplicaciones que se ejecutan en contenedores de software. La ventaja principal de usar Kubernetes es que facilita enormemente la gestión de los contenedores al encargarse del trabajo duro en el escalamiento, recuperación automática, balanceo de cargas, despliegues y mucho más.

Los elementos principales de Kubernetes son los siguientes:

- **Plano de control.** Conjunto de procesos que controlan los nodos de Kubernetes. Es donde se originan todas las asignaciones de tareas.
- **Nodo.** Máquina física o virtual que ejecuta las tareas asignadas por el plano de control.
- **Clúster.** Conjunto de nodos. Es lo que sería una implementación de Kubernetes en funcionamiento.
- **Pod.** Conjunto de contenedores que se ejecutan en el mismo nodo, compartiendo volumen y network namespace (IP, hostname, etc).
- **Controlador de replicación.** Controla la cantidad de copias idénticas de un pod que deben ejecutarse en algún lugar del clúster.
- **Servicio.** Separa las definiciones de las tareas de los pods. Los proxies de servicios de Kubernetes envían las solicitudes de servicio al pod correspondiente de forma automática, sin importar a dónde se traslade en el clúster ni si se lo reemplaza.
- **Kubelet.** Servicio que se ejecuta en los nodos y se encarga de leer los manifiestos del contenedor y de garantizar el inicio y el funcionamiento de los contenedores definidos.

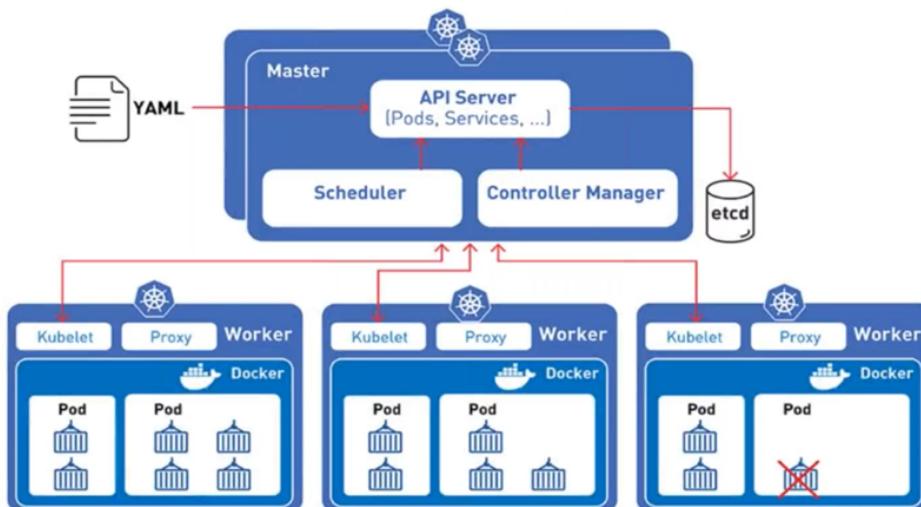


Figura 2.18: Arquitectura de Kubernetes

2.3.3. Terraform

Terraform [23] es una herramienta open-source de codificación declarativa desarrollada por HashiCorp que permite describir la infraestructura de estado final deseada para ejecutar una aplicación en local o en cloud. Esto se hace mediante ficheros de configuración en formato JSON o en la sintaxis de alto nivel HCL (HashiCorp Configuration Language) propuesta por Terraform.

Una vez especificado el estado deseado, elabora un plan para alcanzar ese estado final y lo ejecuta para suministrar la infraestructura.

La infraestructura que Terraform puede administrar incluye componentes de bajo nivel, como instancias informáticas, almacenamiento y redes, así como componentes de alto nivel como entradas de DNS, características de SaaS, etc.

Lo más común es usar un fichero para definir la arquitectura, una serie de variables para poder parametrizar el despliegue y algunas variables de salida para obtener los datos más importantes de la infraestructura una vez desplegada (por ejemplo la IP de un ELB de AWS o el endpoint de RDS).

Existen cuatro elementos principales en Terraform:

- **Providers:** definen los elementos data y resources para plataformas y herramientas específicas. Son plugins que implementan tipos de recursos y contienen todo el código necesario para autenticar y conectarse a un servicio, normalmente desde un proveedor de cloud público, en nombre del usuario. Es posible encontrar proveedores para las plataformas y servicios de cloud que se vayan a utilizar, añadirlos a la configuración y, a continuación, utilizar sus recursos para suministrar la infraestructura. Los proveedores están disponibles para casi todos los principales proveedores de cloud, oferta de SaaS, y más, desarrollados y/o soportados por la comunidad de Terraform u organizaciones individuales.
- **Resources:** definen elementos de la infraestructura que pueden ser aprovisionados, junto con sus propiedades. Una vez desplegados, proporcionan salidas que pueden utilizarse como entrada de otros elementos o como salida final de la ejecución, para mostrar al usuario.
- **Data:** recuperan información acerca del estado actual de un recurso u otro elemento y proporcionan salidas.
- **Modules:** son abstracciones para agrupar conjuntos de data y resources. Se pueden configurar entradas y salidas. Los módulos de Terraform son pequeñas configuraciones de Terraform reutilizables para varios recursos de infraestructura que se utilizan conjuntamente. Son útiles porque permiten automatizar recursos complejos con construcciones configurables y reutilizables. Escribir incluso un archivo de formato Terraform muy simple genera un módulo. Un módulo puede llamar a otros módulos, denominados módulos hijo, que permiten que la configuración de ensamblaje sea más rápida y concisa. Los módulos también se pueden llamar varias veces, ya sea dentro de la misma configuración o en configuraciones separadas.



Figura 2.19: Logo de Terraform

Hay algunas razones clave por las cuales los desarrolladores eligen utilizar Terraform sobre otras herramientas de Infraestructura como código: [24]

- **Open Source.** Terraform está respaldada por grandes comunidades de colaboradores que crean plugins para la plataforma. Independientemente del proveedor de cloud que se utilice, es fácil encontrar plugins, extensiones y soporte profesional. Esto también significa que Terraform evoluciona rápidamente y constantemente se añaden nuevas ventajas y mejoras.
- **Plataforma agnóstica.** Esto significa que puede ser utilizada con cualquier proveedor de servicios de cloud. La mayoría de las demás herramientas de IaC están diseñadas para funcionar con un único proveedor de cloud, lo que podría ser visto como una desventaja ya que si quisiéramos migrar nuestra infraestructura a otro proveedor habría que reescribir gran parte del código.
- **Infraestructura inmutable.** La mayoría de las herramientas de Infraestructura como código crean una infraestructura mutable, lo que significa que la infraestructura puede cambiar para acomodar cambios como una actualización de middleware o un nuevo servidor de almacenamiento. El peligro con la infraestructura mutable es la desviación de configuración, a medida que los cambios se acumulan, el suministro real de diferentes servidores u otros elementos de infraestructura 'se desvía' más allá de la configuración original, haciendo que los errores o problemas de rendimiento sean difíciles de diagnosticar y corregir. Terraform suministra una infraestructura inmutable, lo que significa que con cada cambio en el entorno, la configuración actual se sustituye por una nueva que aplica el cambio y se vuelve a suministrar la infraestructura. Incluso mejor, se pueden conservar las configuraciones anteriores como versiones para habilitar las reverisiones en caso necesario o si así se desea.
- **Lenguaje sencillo.** El lenguaje de configuración de alto nivel que propone HashiCorp, HCL, es muy fácil de entender ya que es muy cercano al lenguaje natural. De esta forma, la sintaxis no es un aspecto que entorpezca el aprendizaje.

2.4. Orquestación en Cloud

Los proveedores de nube [25] permiten acceder a servicios informáticos que, de otro modo, habría que gestionar por cuenta propia, tales como infraestructura (redes, bases de datos, servidores, almacenamiento, virtualización...), plataformas (sistemas operativos, middleware o entornos de ejecución) o software como aplicaciones estándares o personalizadas que ofrecen los proveedores de servicios independientes.

2.4.1. Google Cloud

Cuando hablamos de Google Cloud Platform (GCP), estamos ante todas las herramientas de Google disponibles en la nube que hasta ahora se ofrecían por separado. Este conjunto de servicios ofrecen prestaciones muy dispares; desde machine learning hasta Inteligencia artificial pasando por el big data, todo englobado bajo el paraguas del cloud computing.



Google Cloud Platform

Figura 2.20: Logo de Google Cloud

A la hora de desplegar escenarios de red virtualizados en la nube, de todos los servicios que ofrece Google Cloud los que más nos interesan son los relacionados con Computing y Networking. De entre estos, los más interesantes se detallan a continuación:

Compute Engine

Compute Engine [26] es un servicio de hosting y procesamiento que permite crear y ejecutar máquinas virtuales (también llamadas instancias) en la infraestructura de Google. Las instancias de Compute Engine pueden ejecutar las imágenes públicas de Linux y Windows Server que proporciona Google, así como imágenes personalizadas privadas que es posible crear o importar desde sistemas existentes. También pueden implementar contenedores de Docker.

Cada interfaz de red de una instancia de Compute Engine está asociada con una subred de una red de VPC única y debe tener una dirección IPv4 interna principal. Una instancia puede comunicarse con instancias en la misma red de nube privada virtual (VPC) mediante la dirección IP interna de la VM. Para la comunicación con Internet, se puede usar una dirección IP externa configurada en la instancia, pero hay que tener en cuenta que de esta forma la instancia quedaría expuesta a Internet. Si no se configura ninguna dirección externa en la instancia, se puede usar Cloud NAT, que permite que ciertos recursos sin direcciones IP externas creen conexiones salientes a Internet. Los recursos externos no pueden acceder directamente a ninguna de las instancias privadas ubicadas tras la pasarela de Cloud NAT, lo que contribuye a que las VPC de Google Cloud permanezcan aisladas y protegidas.

2.4. ORQUESTACIÓN EN CLOUD

Una etiqueta o tag es un string de caracteres que se agrega al campo de etiquetas en un recurso, como puede ser una VM. Las etiquetas permiten hacer que las reglas de firewall y las rutas se puedan aplicar a instancias de VM específicas. Una etiqueta de red solo se aplica a las Redes de VPC que se adjuntan directamente a las interfaces de red de la instancia. Es decir, en el caso de conectar dos VPC entre sí, cada VPC solo podrá ver las etiquetas de las VM cuya interfaz de red está asociada a ella. Por lo tanto, en las reglas de FW que limitan el tráfico entre dos VPC, no se puede usar como origen o destino una etiqueta de red, ya que las VPC intercambian las rutas pero no las etiquetas. En ese caso, lo mejor es usar las direcciones IP para aplicar las reglas de FW o rutas a VM específicas.

VPC

La nube privada virtual (VPC) [27] de Google Cloud ofrece funcionalidad de red a las instancias de máquinas virtuales (VM) de Compute Engine. Una red de VPC es como una red física que se virtualiza dentro de Google Cloud, compuesta por una lista de subredes virtuales regionales (subredes) en centros de datos, todas conectadas por una red de área extensa global. Las redes de VPC están aisladas de forma lógica unas de otras dentro de Google Cloud.

Cada red de VPC funciona como un firewall virtual distribuido. Las reglas de firewall permiten controlar qué paquetes pueden trasladarse a qué destinos. Cada red de VPC tiene dos reglas de firewall implícitas que bloquean todas las conexiones entrantes y permiten todas las conexiones salientes. Aunque las reglas de firewall se definen a nivel de red, las conexiones se permiten o deniegan por instancia, lo que permite restringir el intercambio de tráfico en función de la IP o de la etiqueta de una VM. Cada regla de firewall se aplica a la conexión entrante o saliente, pero no a ambas, y permite especificar origen, destino, protocolo, puertos y acción (permitir o denegar). Cuando se permite una conexión a través del firewall en cualquier dirección, también se permite el tráfico de retorno que coincide con esta conexión.

También es posible definir rutas. Las rutas indican a las instancias de VM y a la red de VPC cómo enviar el tráfico de una instancia a un destino, ya sea dentro de la red o fuera de Google Cloud. Las redes de VPC incluyen algunas rutas generadas por el sistema para enrutar el tráfico entre sus subredes y enviarlo de las instancias aptas a Internet.

Se pueden conectar dos VPC entre sí. Con el intercambio de tráfico entre redes de VPC, todas las comunicaciones se realizan mediante direcciones IP internas. Según las reglas de firewall, las instancias de VM en cada red de intercambio de tráfico pueden comunicarse entre sí sin usar direcciones IP externas. Cabe destacar que solo las redes de intercambio de tráfico directo pueden comunicarse, no se admite el intercambio de tráfico transitivo. En otras palabras, si la red de VPC N1 intercambia tráfico con las redes N2 y N3, pero estas no están conectadas directamente, la red de VPC N2 no se podrá comunicar con la N3 mediante el intercambio de tráfico entre redes de VPC. Al conectar dos VPC, las reglas de firewall (recordemos que una regla de FW se aplica a nivel de VPC) no se intercambian entre ellas. Además, como ya se ha mencionado, no es posible hacer referencia desde una VPC a etiquetas de la otra, ya que estas tampoco se intercambian.

Capítulo 3

Diseño

En este capítulo se van a detallar las tecnologías a utilizar, así como las etapas a seguir para la implementación de los escenarios de red. Para ello, se parte de un análisis de los requisitos necesarios a cumplir por el sistema tras el cual se tomarán las decisiones de diseño correspondientes.

3.1. Requisitos y decisiones de diseño

En primer lugar, se han identificado una serie de requisitos que debe cumplir el sistema. Estos se han recogido en la tabla que se muestra a continuación:

Id	Requisito
RD1	Los escenarios han de estar disponibles en cualquier momento y lugar para su utilización y han de poderse ejecutar en cualquier tipo de sistema.
RD2	El entorno debe de ser seguro, de forma que no se pongan en riesgo recursos sensibles.
RD3	Debe ser escalable, tener un tiempo de arranque reducido y ocupar el menor espacio posible.
RD4	Ha de ser portable, y debe permitir la configuración de los escenarios para adaptarse a nuevas tendencias o situaciones.
RD5	Es deseable que tanto el despliegue como la conexión y aprovisionamiento sea centralizada. Es decir, que todas las configuraciones que requiera el escenario, tanto las que se lleven a cabo en este trabajo como las futuras (configuración software), se puedan realizar desde el entorno que se va a proporcionar.

Tabla 3.1: Requisitos de diseño

Tras el análisis de tecnologías realizado en el capítulo 2, podemos determinar aquellas que se adecuan mejor a los requisitos arriba expuestos. Teniendo en cuenta los requisitos *RD1*, *RD2* y *RD3*, se ha decidido que lo mejor es realizar un despliegue en Cloud, para lo cual se ha seleccionado la plataforma Google Cloud Platform. Para orquestar el despliegue de la infraestructura en la nube, cumpliendo con los requisitos *RD3*, *RD4*, y *RD5* se usará Terraform. Finalmente, para la virtualización de los sistemas, en un principio se hará uso

3.2. SOLUCIÓN PROPUESTA

de las máquinas virtuales que proporciona Google Cloud. No obstante, en el caso de ser necesarios algunos servicios o funcionalidades concretas, se correrán contenedores Docker dentro de dichas máquinas virtuales, que como se comentó en el estado del arte permiten virtualizar gran cantidad de software de una manera muy sencilla, eficiente, ocupando poco espacio y con un tiempo de arranque muy reducido.

3.2. Solución propuesta

La solución que se propone es un despliegue en Google Cloud orquestado por Terraform y empleando la tecnología de virtualización Docker.

El hecho de realizar un despliegue en cloud permite que la infraestructura utilizada para el entrenamiento en ciberseguridad sea completamente independiente de la de la empresa o equipo de quien la usa, lo que proporciona seguridad a las organizaciones que lo usen.

Además, esta infraestructura será accesible por cualquiera que tenga autorización, sin importar donde se encuentre. No conlleva costes de operación ni mantenimiento, pues Google se encarga tanto del hardware donde se virtualizan los escenarios como de mantener actualizado el software. Por último es escalable, según se vayan necesitando recursos como almacenamiento o memoria se van asignando sin necesidad de interacción humana, y se paga sólo por los recursos que se usan, lo que supone un gran ahorro en costes.

A la hora de orquestar el despliegue, Terraform proporciona muchas ventajas. En primer lugar, permite tener en ficheros con una sintaxis muy sencilla la definición de toda la infraestructura. Sólo basta con ejecutar la orden `terraform apply` para que se despliegue todo de forma automática en la nube. Además los ficheros que describen el estado final deseado de la infraestructura permiten a su vez el uso de ficheros de variables para parametrizar el despliegue y que la configuración sea más modular y sencilla. También permite que los escenarios sean portables, ya que puede ser utilizado con cualquier proveedor de servicios de cloud. De esta forma, si se decide migrar la infraestructura a otro proveedor por cualquier motivo, no habría que reescribir todo el código, como ocurriría con el resto de herramientas de IaC diseñadas para funcionar con un único proveedor de cloud. Otro aspecto importante es que con cada cambio que se realice en el entorno, la configuración actual se sustituye por una nueva que aplica el cambio y se vuelve a suministrar la infraestructura, lo que se conoce como infraestructura mutable y que facilita su uso junto a un software de control de versiones como GitHub.

Por último, en cuanto al aprovisionamiento, Terraform permite seleccionar a la hora de definir una instancia la imagen base de esta, bien sea una imagen de las que proporciona Google o una imagen personalizada que hayamos construido y alojado en el propio Google Cloud. Además, en caso de ser necesarias configuraciones extra, Terraform también permite especificar scripts de inicio personalizados que se ejecutan cuando esta se arranca. Estos scripts pueden estar escritos en cualquier lenguaje, y pueden servir para realizar multitud de configuraciones como instalación de paquetes o modificación de ficheros. Un añadido muy interesante que proporciona Terraform es la función `templatefile`, que recibe como parámetro el path a un script y permite usarlo como una plantilla donde se especifican las variables deseadas. Es mejor verlo en un ejemplo práctico. Imaginemos que tenemos el fichero `docker.tftpl` con el siguiente contenido:

```

1 #!/bin/bash
2 docker run ${image}

```

Código 3.1: Contenido del fichero `docker.tftpl`

A la hora de definir la instancia en el fichero de configuración de Terraform, en el campo correspondiente indicamos qué valor queremos que tome esa variable, que como vemos se define con la sintaxis `${...}`:

```

1 [...] #Inicio de la definicion de la instancia
2 metadata_startup_script=templatefile("templates/docker.tftpl", {
3     image = "nginx" })
4 [...] #Fin de la definicion de la instancia

```

Código 3.2: Extracto de fichero de configuración `.tf`

De esta forma, cuando la instancia se arranque y ejecute el fichero, lo hará sustituyendo “nginx” en el lugar de la variable `image`. Es importante la extensión `.tftpl` para que Terraform interprete correctamente las variables. En el caso de no ser necesaria la parametrización del script, se pueden enviar directamente scripts en Bash, Python o el lenguaje que se desee haciendo uso de la función `file`, donde solo sería necesario especificar el path al archivo, a diferencia de la función `templatefile` que necesita también el valor que han de tomar las variables.

Como se puede observar, estas funciones proporcionan una gran flexibilidad y libertad a la hora de configurar las instancias. Y lo más importante, permiten tener un directorio con archivos de configuración y acceder a ellos desde el fichero donde se configura el despliegue de la infraestructura, de forma que se puede manejar todo de forma centralizada.

3.3. Etapas de diseño

Una vez propuesta la solución, se establecen una serie de etapas de diseño en las que se divide el proceso, siguiendo un orden secuencial que se muestra a continuación:

Id	Etapa de diseño
ED1	Definición de los escenarios a implementar.
ED2	Análisis de los escenarios, vector de ataque y conexiones necesarias entre los elementos que lo componen.
ED3	Desarrollo de los ficheros Terraform que permiten la implementación de los escenarios definidos en la nube según el estudio realizado.
ED4	Creación de herramientas que permitan arrancar contenedores Docker, así como realizar algunas pequeñas configuraciones en las instancias y que sirvan como base para la futura configuración de los escenarios.

Tabla 3.2: Etapas de diseño

Como resultado de este proceso se obtendrán escenarios de red virtualizados en la nube con todos los elementos y configuraciones necesarias para generar a partir de ellos ejercicios para la formación y entrenamiento en el campo de la ciberseguridad.

Capítulo 4

Desarrollo

En este capítulo se detalla el desarrollo del proyecto siguiendo las etapas de diseño descritas. Se comenzará por la preparación del entorno y posteriormente se describirá la implementación de los escenarios en GCP.

4.1. Preparación del entorno

Antes de comenzar, es necesario disponer de una cuenta en Google Cloud Platform, así como disponer de un proyecto creado. Para crear un proyecto, basta con seleccionar o crear uno en la página del selector de proyectos de Google Cloud.

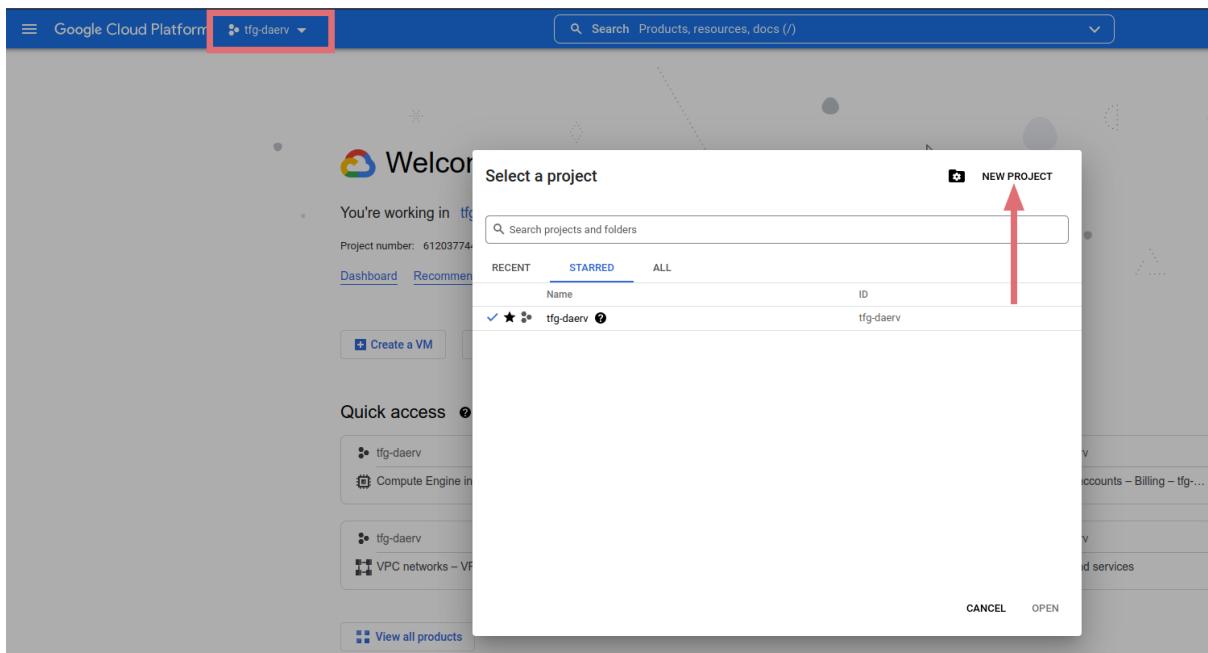


Figura 4.1: Selección de un proyecto en Google Cloud

Una vez tengamos creada una cuenta y un proyecto en ella, debemos habilitar la API de Google Compute Engine [28] para nuestro proyecto en la consola de GCP. También es necesario instalar tanto Terraform como la CLI de Google Cloud. Una vez hecho todo esto, lo primero es autenticarse con GCP. Para ello, basta con ejecutar `gcloud auth`

`application-default login` en la terminal. Esto nos dirigirá a una página donde podremos iniciar sesión con nuestra cuenta de Google para permitir el acceso a nuestros datos de GCP.

Para poder realizar peticiones desde Terraform a la API de GCP, también es necesario autenticarse para así probar que somos quien están realizando esas peticiones. Hay varias formas de realizar esta autenticación. Una de ellas es mediante las Cuentas de Servicio de Google Cloud, para la cual hay que seguir los siguientes pasos:

1. En el apartado de Cuentas de Servicio [29] de la consola de Google Cloud debemos elegir una cuenta existente, o crear una nueva. A la hora de crearla hay que tener en cuenta que es necesario asignar permisos de edición.
2. En la sección de claves, debemos generar una clave y descargarla en formato JSON, ponerle un nombre del que nos vayamos a acordar y almacenarla en un lugar seguro.
3. Para proporcionarle la clave descargada a Terraform, lo haremos mediante la variable de entorno `GOOGLE_APPLICATION_CREDENTIALS`, asignándole como valor el de la ruta al archivo con la clave ejecutando el siguiente comando:

```
1  export GOOGLE_APPLICATION_CREDENTIALS=\{\{path\}\}
```

Para que las credenciales se guarden entre sesiones, es necesario añadir esta línea a un fichero de inicio como `bash_profile` o `bashrc`. Una opción alternativa a la variable de entorno sería proporcionar a Terraform el path a la clave en la configuración del provider, dentro del fichero `main.tf`.

Por último, en el fichero `.tf` que va a contener todo el código necesario para alcanzar el estado necesario de nuestra infraestructura, hay que configurar el provider de Google. Para ello, en dicho fichero, que normalmente se llama `main.tf` añadiríamos el código mostrado en la parte inferior, y ejecutaríamos la instrucción `terraform init` en el mismo directorio donde se encuentra el fichero para que todos e configuren correctamente.

```
1 provider "google" {
2     project = var.project_id
3     region  = var.region
4     zone    = var.zone
5 }
```

En este caso, se han utilizado variables almacenadas en el fichero `variables.tf` para asignar el valor a los parámetros. Además, también cabe mencionar que por comodidad se ha empleado la misma región y zona en todos los escenarios desplegados, ya que el valor de este campo simplemente determina el centro de datos de Google en el que se despliega la infraestructura.

Una vez hechas estas configuraciones, ya podemos desarrollar nuestro código Terraform y ordenar que se despliegue la infraestructura en nuestro proyecto simplemente con el comando `terraform apply`, destruir la infraestructura desplegada con `terraform destroy` o consultar el estado de los recursos desplegados con la orden `terraform state list`, entre otras muchas opciones.

4.2. Escenarios de red

En esta sección se van a presentar los escenarios de red a simular y se va a desarrollar su implementación en Google Cloud.

4.2.1. Smart Office 1

Descripción

El escenario *Smart Office 1* simula una oficina que cuenta con una red LAN cableada Ethernet en la que se encuentran conectados los sistemas de la oficina. En dicha oficina, existe un sistema de impresión que permite a los empleados realizar impresiones desde sus ordenadores. Dicho sistema cuenta con uno o varios dispositivos basados en Linux, de los cuales uno es una impresora conectada a un servidor remoto propio de la marca en un entorno Cloud.

El atacante se encuentra conectado a una red local inalámbrica en la que que también se encuentra el PC de un empleado de la empresa. Dicho empleado está a su vez conectado a la red interna de la empresa mediante una red cableada Ethernet. El objetivo del atacante es lanzar un ataque sobre el servicio de impresión en el que, mediante la modificación del firmware, logre acceder a la red local remota. La impresora no requiere autenticación antes de la actualización de firmware, por tanto esta sería la vulnerabilidad a aprovechar por el atacante.

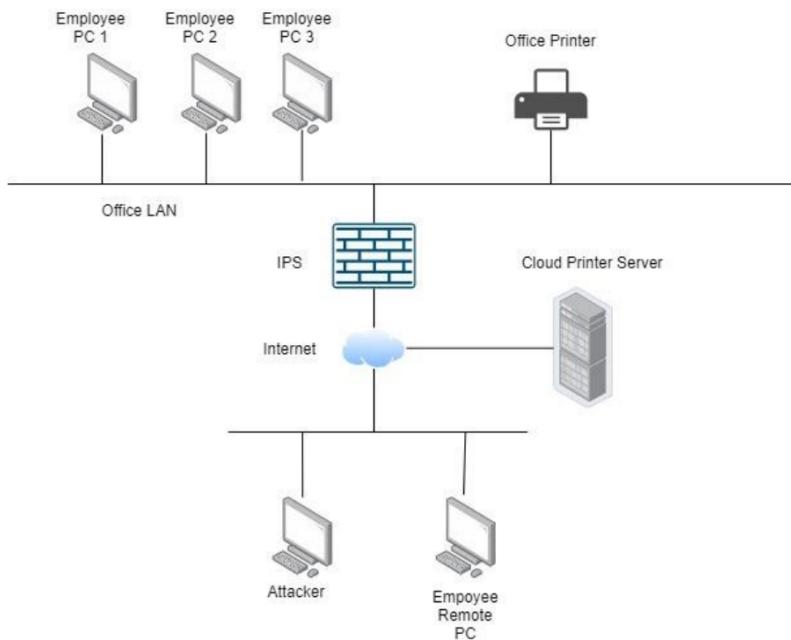


Figura 4.2: Topología del escenario Smart Office 1

El atacante deberá ganar acceso al equipo del empleado remoto, que es quien tiene conexión con la red interna de la empresa, mediante la explotación de alguno de sus servicios. Una vez hecho esto, recopilaría información referente a la impresora y solicitaría el procesamiento de un documento malicioso, cuyo contenido será enviado y embebido mediante

comandos PJL (*Printer Job Language*). La aleatoriedad de los comandos haría imposible su detección por parte de los sistemas de seguridad de la empresa, de forma una vez lleguen a la impresora víctima, esta reconocerá que el trabajo de impresión contiene una actualización de firmware válida y permitirá al atacante realizar modificaciones arbitrarias en el área de almacenamiento del firmware. En esta situación, el atacante podrá acceder al servidor de impresión, que actúa como un log descentralizado y podría contener información sensible sobre lo que haya impreso cada empleado.

Implementación

Para la implementación de este escenario se han definido 3 VPCs. Una de ellas representa la oficina interna, donde se ubican 2 instancias de Compute Engine basadas en Linux que representan equipos de empleado y la impresora. Esta red interna está segmentada, es decir, los equipos de empleado se ubican en una LAN diferente a la impresora, siendo una LAN una subred de la VPC. Otra VPC simula la red externa de la empresa, donde se localizan el equipo del atacante y el empleado remoto, ambos en la misma LAN y también basados en Linux. Finalmente, en la tercera VPC se aloja el servidor de impresión.

VPC	LAN	Rango IP	Equipos
office-internal-network	employees-lan	10.10.10.0/24	employee-pc-1 employee-pc-2
	printer-lan	10.10.20.0/24	office-printer
internal-server-network	printer-server-lan	10.10.30.0/24	cloud-printer-server
office-external-network	office-external-lan	10.10.40.0/24	employee-remote-pc attacker

Tabla 4.1: Estructura del escenario Smart Office 1

La conectividad entre las VPCs es la siguiente: se ha establecido un VPC Peering entre las redes externa e interna de la oficina, de forma que se permite la conectividad entre las direcciones IP internas de ambas, lo que simularía la conexión cableada Ethernet previamente mencionada. A su vez, también existe un VPC Peering entre la red interna de la oficina y la red donde se ubica el servidor de impresión, que no es accesible desde internet. De esta forma, los equipos de la red interna de la oficina tendrán conectividad tanto con la red externa como con la red del servidor, mientras entre la red externa y el servidor no existe dicha conectividad.

En cuanto al acceso a Internet, en cada una de las redes de la oficina se ha desplegado un Cloud Router con su respectivo Cloud NAT, lo que permite que las instancias sin direcciones IP externas que se encuentran en ambas redes puedan crear conexiones salientes a Internet. El servidor es local y por tanto no se ha configurado su acceso a internet.

Tal y como se mencionó en el estado del arte, que las instancias tengan conectividad no quiere decir que esta sea efectiva, ya que una de las reglas de FW implícitas rechaza todo el tráfico entrante. Es decir, para que sea posible el intercambio de tráfico deseado, no basta con hacer un peering entre las VPC, sino que hay que definir reglas de FW que permitan o restrinjan el tráfico que se intercambia. Para replicar este escenario, ha sido necesario crear las siguientes:

4.2. ESCENARIOS DE RED

Nombre	VPC	Dirección	Origen	Destino	Protocolos	Puertos	Acción
allow-internal	office-internal-network	INGRESS	10.10.10.0/24 10.10.20.0/24	-	TCP, UDP, ICMP	0-65535	ALLOW
allow-external	office-external-network	INGRESS	10.10.40.0/24	-	TCP, UDP, ICMP	0-65535	ALLOW
allow-internal-from-external	office-internal-network	INGRESS	employee-remote-pc (IP)	-	TCP, UDP, ICMP	0-65535	ALLOW
allow-external-from-internal	office-external-network	INGRESS	10.10.10.0/24 10.10.20.0/24	employee-remote-pc (IP)	TCP, UDP, ICMP	0-65535	ALLOW
allow-printer-from-server	office-internal-network	INGRESS	10.10.30.0/24	printer (IP)	TCP, UDP, ICMP	0-65535	ALLOW
allow-server-from-printer	internal-server-network	INGRESS	printer (IP)	10.10.30.0/24	TCP, UDP, ICMP	0-65535	ALLOW

Tabla 4.2: Reglas de FW del escenario Smart Office 1

allow-internal: esta regla se aplica a la VPC de la red interna de la oficina. Permite la entrada de tráfico de cualquier protocolo por cualquier puerto siempre y cuando la dirección IP origen se encuentre dentro de la red interna. Al no especificarse destino, aplica a toda la VPC, por lo que en resumen esta regla permite que las instancias de la red interna de la oficina intercambien tráfico entre sí.

allow-external: análogamente a allow-internal, esta regla permite que el atacante y el empleado remoto intercambien tráfico entre sí, al encontrarse en la misma LAN.

allow-internal-from-external: se aplica a la red interna de la oficina, de forma que permite la entrada del tráfico procedente únicamente del empleado remoto, ya que es quien está conectado mediante cable a la red interna. Al estar en VPCs distintas, se especifica como origen la IP del empleado remoto.

allow-external-from-internal: análogamente a la anterior, se aplica a la red externa de la oficina, y permite la entrada de tráfico procedente tanto de los empleados como de la impresora únicamente hacia empleado remoto, especificando su IP.

allow-printer-from-server: se aplica en la red interna de la oficina, permite el tráfico entrante procedente de la LAN donde se encuentra el servidor hacia la impresora.

allow-server-from-printer: análogamente a la anterior, el servidor sólo acepta tráfico de entrada procedente de la impresora.

Es necesario puntualizar algunos aspectos acerca de las reglas de FW comunes a todos los escenarios:

- Como se puede apreciar, muchas de las reglas están configuradas para permitir el tráfico de los protocolos TCP, UDP e ICMP por todos los puertos. Por supuesto esto

no sería lo óptimo, y cuando se conociese con detalle los servicios y comunicaciones que realiza cada máquina, estas reglas se podrían modificar para concretar mejor el tráfico que permiten.

- Las reglas de la tabla tienen asignada una prioridad mayor que aquellas reglas que define Google Cloud por defecto. Por tanto, todo el tráfico de entrada a cualquiera de las VPCs que no aparezca definido en la tabla será rechazado. Además, en la tabla sólo se muestran las reglas custom que ha sido necesario crear, por lo que aunque no aparezca, es efectiva la regla por defecto de Google Cloud que permite el tráfico saliente desde cualquier instancia hacia cualquier destino usando cualquier protocolo por cualquier puerto.
- A la hora de especificar los orígenes y destinos en Terraform, se hace una referencia al objeto. Es decir, no se especifica como origen un string con el rango de IPs de una subred, si no que se hace referencia al rango de IP que tenga asignada la LAN en ese momento, de forma que si esas IP cambian, no es necesario cambiar la definición de la regla. Esto se puede ver con más detalle en el Anexo C.

En cuanto al aprovisionamiento, la idea en este escenario sería crear imágenes en Google Cloud a partir de una ISO ya existente, como por ejemplo una imagen basada en Kali Linux para el atacante, o imágenes Windows de usuario para los empleados, ya que las imágenes Windows disponibles en Google Cloud son de Windows Server. Estas imágenes personalizadas se almacenarían en Google Cloud y se podría pasar como parámetro a la instancia a la hora de construirla, como también se puede ver en el Anexo C.

4.2. ESCENARIOS DE RED

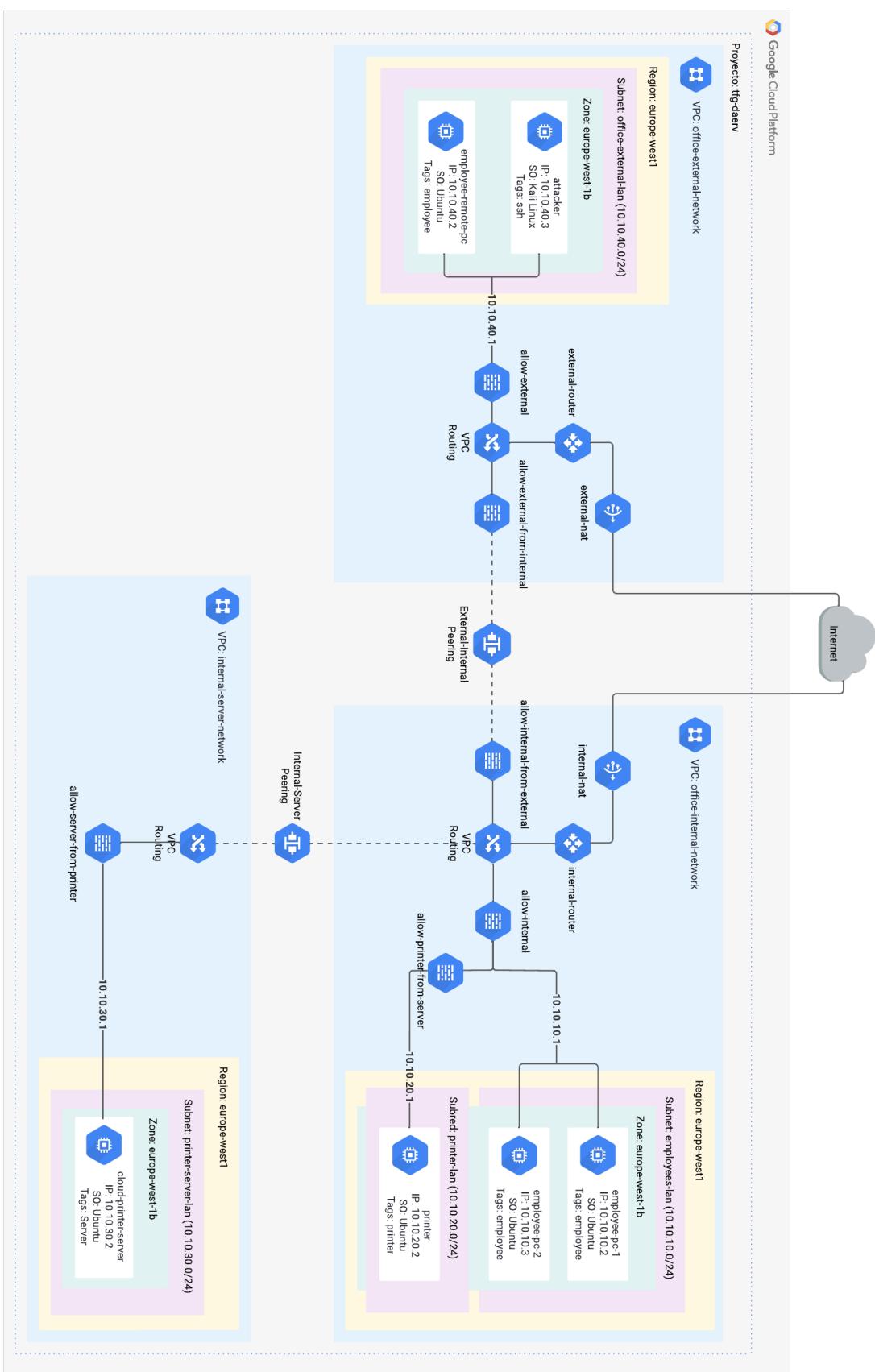


Figura 4.3: Implementación en GCP del escenario Smart Office 1

4.2.2. Smart Office 2

Descripción

El escenario *Smart Office 2* plantea una oficina que cuenta con una red LAN cableada Ethernet en la que se encuentran conectados los sistemas de la oficina. En dicha LAN, existe un sistema de *Smart Speaker* (micrófono y altavoz) que permite a los empleados hacer ciertas tareas utilizando órdenes de voz. Dicho sistema cuenta con uno o varios dispositivos basados en Linux que funcionan como clientes y que escuchan los comandos de voz en la oficina. Estos dispositivos se conectan a un hub interno que, mediante un protocolo IoT permite lanzar comandos básicos a los dispositivos (encendido, apagado, etc) y recibir mensajes de audio capturado a través de los micrófonos. Estos mensajes de audio se cifran y se envían de forma segura desde el hub a un servidor remoto localizado en un entorno Cloud.

La empresa cuenta con un servicio VPN implementado en el router de entrada a la red LAN que permite a los empleados conectarse a dicha red de forma remota. Existe una vulnerabilidad en el cliente VPN de dicho servicio que permitiría a un atacante robar las credenciales del usuario a través de sus cookies y acceder a la red local.

El atacante se encuentra conectado a una red LAN en la que también se encuentra el PC de un empleado de la empresa. En dicho PC existe una vulnerabilidad en un servicio de escritorio remoto que permite al atacante acceder a las cookies almacenadas por el cliente VPN y robarlas para suplantar al empleado y acceder a la empresa a través del servicio VPN. Una vez en la red interna, el atacante es capaz de acceder al hub Linux, que se comunica de forma insegura con los dispositivos *Smart Speaker*, siendo capaz de activarlos y desactivarlos arbitrariamente y de recuperar el audio recogido por ellos antes de que se envíe al servidor remoto sin conocimiento de los empleados en la oficina.

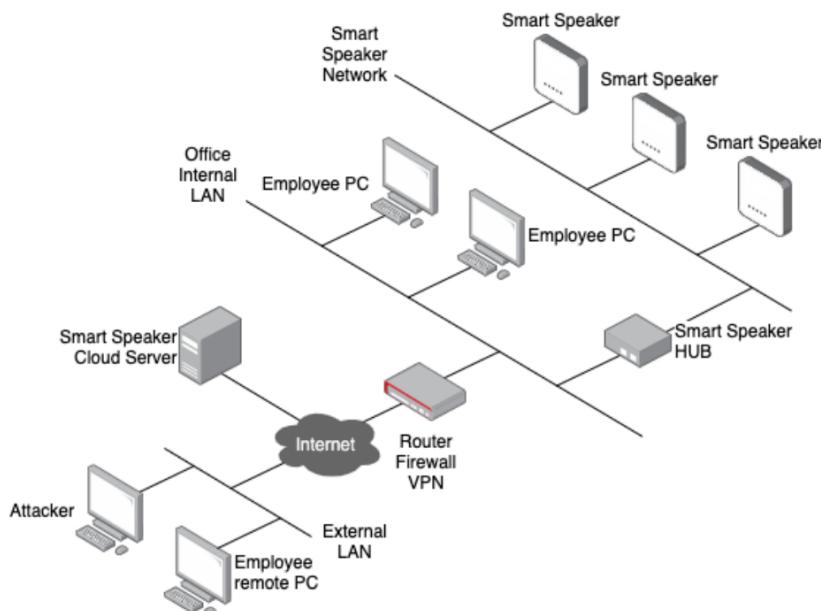


Figura 4.4: Topología del escenario Smart Office 2

4.2. ESCENARIOS DE RED

Implementación

Este escenario se ha implementado siguiendo la misma base que en el escenario *Smart Office 1*: una VPC una para cada red de oficina y otra para la red del servidor. Al igual que en el caso anterior, en la red externa se ubica un PC de empleado y el equipo del atacante y en la red del servidor se encuentra el servidor cloud donde se envía el audio recogido por los altavoces inteligentes. La red interna está segmentada en dos LAN, una de ellas con dos PC de empleado y el hub, y la otra con los altavoces inteligentes. Todas las VMs son instancias de Google Compute Engine que usan imágenes Linux, aunque lo ideal sería, como ya se ha comentado, crear una imagen personalizada de Kali Linux para el atacante y una imagen de Windows de escritorio para los empleados. Las VMs de la oficina tienen conexión a internet mediante sus respectivos Cloud Nat, mientras que el servidor no está expuesto.

VPC	LAN	Rango IP	Equipos
office-internal-network	office-internal-lan	10.10.10.0/24	employee-pc-1 employee-pc-2 smart-speaker-hub
	speakers-lan	10.10.20.0/24	smart-speaker-1 smart-speaker-2 smart-speaker-3
internal-server-network	speaker-server-lan	10.10.30.0/24	smart-speaker-cloud-server
office-external-network	office-external-lan	10.10.40.0/24	employee-remote-pc attacker

Tabla 4.3: Estructura del escenario Smart Office 2

La interconexión entre las VPC difiere del escenario anterior. Entre la red interna de la oficina y la red del servidor, se mantiene el VPC peering que permite el intercambio de tráfico entre las VPC. En cambio, con el fin de simular la conexión VPN, entre las dos redes de oficina se ha empleado el servicio de Google Cloud llamado Cloud VPN.

Cloud VPN permite la conexión segura hacia la red de Google a través de un túnel VPN IPsec. Se ha creado una puerta de enlace VPN en cada una de las VPC, además de un túnel VPN que interconecta ambas puertas de enlace. Una puerta de enlace de VPN encripta el tráfico que viaja entre las dos redes a través de la Internet pública y la otra puerta de enlace de VPN lo desencripta. De esta forma, se protegen los datos mientras viajan por Internet. Cada puerta de enlace tiene una dirección IP pública. Google Cloud ofrece dos tipos de puertas de enlace de Cloud VPN: VPN con alta disponibilidad y VPN clásica. Se han empleado puertas de enlace de alta disponibilidad, dado que proporciona un 99.99 % de esta, además de un encaminamiento más eficiente.

Pero no sólo ha bastado con crear las puertas de enlace y el túnel. Dado que no se ha entrado en la configuración de las instancias, no está implementado el cliente VPN que permite el intercambio de tráfico entre las puertas de enlace. Como solución temporal, para proporcionar conectividad entre las VPCs se han creado las siguientes rutas estáticas que permiten encaminar los paquetes entre los equipos de las redes de la oficina:

Nombre	Destino	VPC	Siguiente salto
internal-vpn-route	10.10.40.0/24	office-internal-network	internal-vpn-tunnel
external-vpn-route	10.10.10.0/24 10.10.20.0/24	office-external-network	external-vpn-tunnel

Tabla 4.4: Rutas del escenario Smart Office 2

Una vez existe la conectividad deseada entre los equipos, es necesario aceptar o denegar el tráfico entrante o saliente. Las reglas de firewall creadas se recogen a continuación:

Nombre	VPC	Dirección	Origen	Destino	Protocolos	Puertos	Acción
allow-internal	office-internal-network	INGRESS	10.10.10.0/24	-	TCP, UDP, ICMP	0-65535	ALLOW
allow-external	office-external-network	INGRESS	10.10.40.0/24	-	TCP, UDP, ICMP	0-65535	ALLOW
allow-hub-from-speakers	office-internal-network	INGRESS	speaker (tag)	speaker-hub (tag)	TCP, UDP, ICMP	0-65535	ALLOW
allow-speakers-from-hub	office-internal-network	INGRESS	speaker-hub (tag)	speaker (tag)	TCP, UDP, ICMP	0-65535	ALLOW
allow-internal-from-external	office-internal-network	INGRESS	employee-remote-pc (IP)	-	TCP, UDP, ICMP	0-65535	ALLOW
allow-external-from-internal	office-external-network	INGRESS	10.10.10.0/24	employee-remote-pc (IP)	TCP, UDP, ICMP	0-65535	ALLOW
allow-hub-from-server	office-internal-network	INGRESS	10.10.30.0/24	smart-speaker-hub (IP)	TCP, UDP, ICMP	0-65535	ALLOW
allow-server-from-hub	internal-server-network	INGRESS	smart-speaker-hub (IP)	10.10.30.0/24	TCP, UDP, ICMP	0-65535	ALLOW

Tabla 4.5: Reglas de FW del escenario Smart Office 2

Se aprecia que las reglas de FW son prácticamente iguales a las del escenario Smart Office 1. La diferencia principal radica en que en este escenario, la regla allow-internal tiene como origen sólo el rango 10.10.10.0/24, lo que permite el intercambio de tráfico únicamente entre los ordenadores de empleado y el hub. Esto se debe a que la conexión con los altavoces está restringida, de forma que sea sólo el hub quien pueda recibir y enviar tráfico desde y hacia los altavoces, de ahí la necesidad de las reglas allow-speakers-from-hub y allow-hub-from-speakers, que limitan dicho tráfico haciendo uso de las etiquetas de red de cada instancia (recordemos que, al encontrarse en la misma VPC, es más cómodo usar los tags que las direcciones IP). Por la misma razón, la regla allow-external-from-internal también admite como IP origen únicamente la de la LAN interna, puesto que al empleado externo tampoco le está permitida la conexión directa con los altavoces.

4.2. ESCENARIOS DE RED

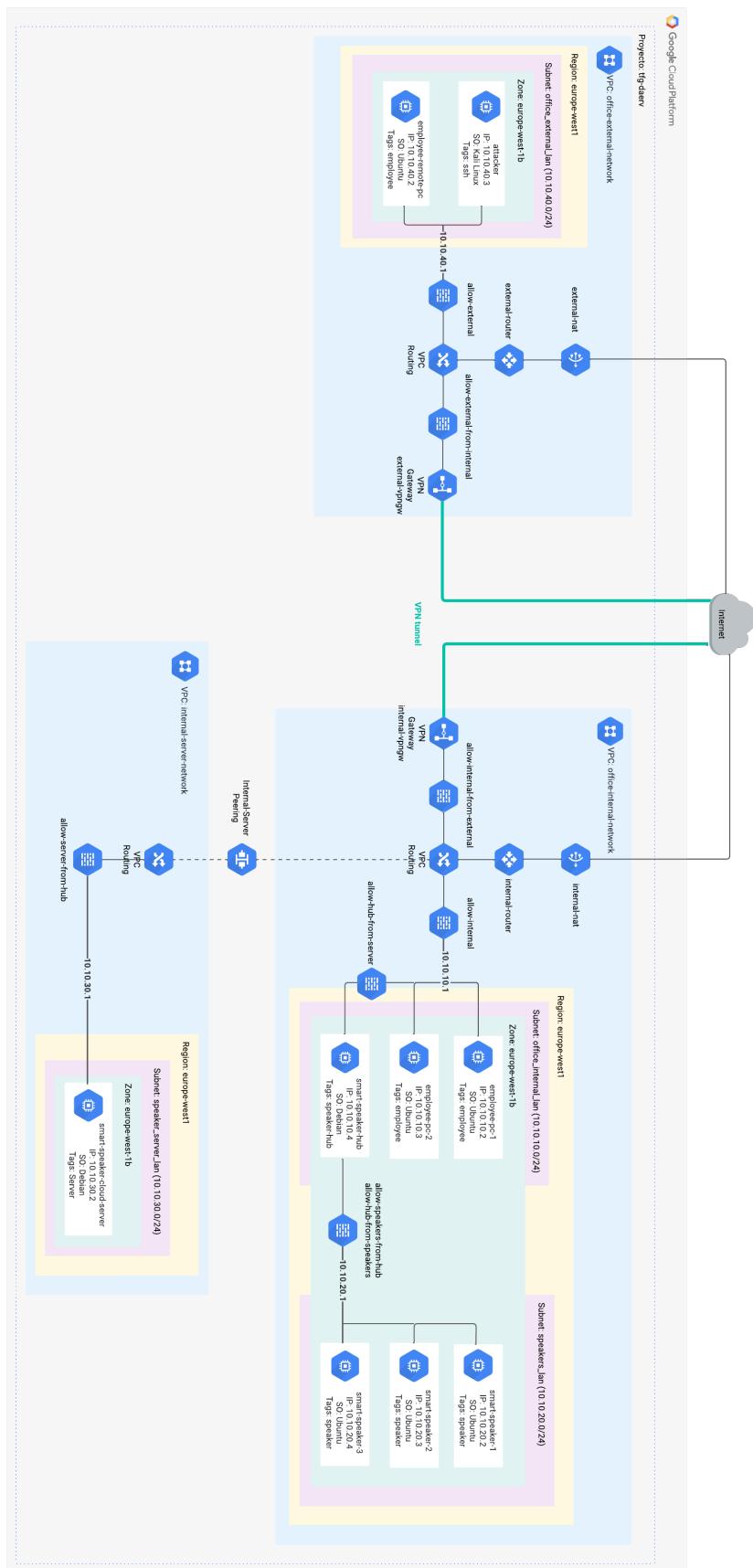


Figura 4.5: Implementación en GCP del escenario Smart Office 2

4.2.3. Smart Home

Descripción

Este escenario representa una red de tipología *Smart Home*, con un Access Point (AP) inalámbrico que ofrece conectividad a los dispositivos IoT de la casa. Además, la casa inteligente está controlada por un *Homematic Central Control Unit* (CCU2) que expone la vulnerabilidad CVE-2019-14423 en un addon del firmware, lo que permite la ejecución remota de código (RCE) de forma que atacantes remotos autenticados podrían ejecutar comandos del sistema como root de forma remota a través de una simple petición HTTP.

Además, otros dos dispositivos de la red exponen vulnerabilidades. En concreto, una bombilla inteligente (Signify Phillips Taolight Smart Wi-Fi Wiz Connected LED Bulb) permite a los usuarios remotos controlar su funcionamiento (CVE-2019-18980) y una cámara IP (Cisco Video Surveillance 8000 Series IP Cameras) podría permitir a un atacante adyacente no autenticado hacer que una cámara IP afectada se recargue (CVE-2021-1131).

El atacante tiene acceso a la red y puede explotar cualquiera de las tres vulnerabilidades mencionadas, que se explican con mayor detalle a continuación:

La versión de firmware 2.31.25 del Homematic CCU2 muestra graves fallos de seguridad que permiten a un atacante obtener acceso completo al sistema y potencialmente también a los dispositivos periféricos conectados a través de comandos de shell que utilizan indebidamente la vulnerabilidad RCE. En cuanto a la bombilla, el modelo mencionado hace uso de una API no protegida (no hay autenticación ni cifrado para utilizarla) que permite que cualquiera pueda encenderla, apagarla o cambiar su brillo de forma remota. El único requisito es que el atacante tenga acceso a la red de la bombilla. Finalmente, en las cámaras IP de la serie 8000 de Cisco Video Surveillance, una vulnerabilidad en la implementación del protocolo Cisco Discovery permite que, enviando un paquete malicioso de Cisco Discovery Protocol a una cámara afectada, el atacante pueda realizar un ataque DoS al provocar que la cámara se recargue inesperadamente.

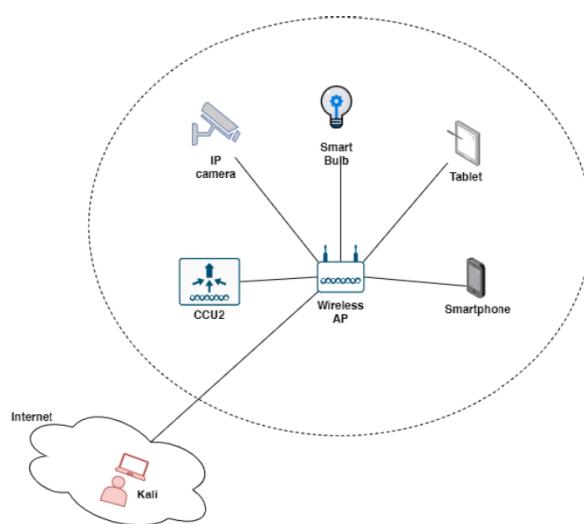


Figura 4.6: Topología del escenario Smart Home

4.2. ESCENARIOS DE RED

Implementación

Para la simulación del escenario *Smart Home* se han desplegado dos VPCs. La primera VPC representa la red local de la casa, que contiene 6 instancias Linux, que se corresponden con los dispositivos conectados a ella. La segunda VPC contiene únicamente el equipo del atacante, y su única función es la de brindarle a este acceso a la red de la casa, para lo cual se ha establecido un VPC peering entre ambas VPC. De esta forma, el atacante tiene acceso a la red local de la casa, lo que le permitirá explotar las vulnerabilidades que se han presentado anteriormente.

VPC	LAN	Rango IP	Equipos
smart-home	home-lan	10.10.10.0/24	wireless-ap, tablet, ccu2, ip-camera, smart-bulb, smartphone
attacker	attacker-lan	10.10.20.0/24	attacker

Tabla 4.6: Estructura del escenario Smart Home

En este caso, en la VPC *smart-home* no se ha empleado Cloud NAT para proporcionar acceso a internet a las instancias. A fin de que la representación de la casa sea lo más realista posible, lo que se ha hecho en esta VPC ha sido asignar una IP pública dinámica a la instancia wireless-ap (las instancias con una IP pública tienen acceso a Internet) y configurarla como un proxy, de forma que el resto de periféricos que se encuentran en la LAN de la casa accedan a internet a través de esta, sin necesidad de Cloud NAT ni de una IP pública en cada uno. De esta forma, se persigue simular el funcionamiento habitual de un router, en el que el proveedor de servicios de Internet (ISP) le asigna una IP pública dinámica que es usada por todos los dispositivos de la casa para acceder a Internet.

Para lograr esta configuración, se han creado dos template-files de Terraform. Realmente son scripts en Bash, pero se usa el formato `.tftpl` de Terraform para poder interpretar objetos de Terraform y usarlos como variables. El primero es `proxy-config.tftpl`. Este script es llamado en el proceso de creación de la instancia wireless-ap y se encarga de configurarla como un proxy de acceso a Internet. El contenido del script está disponible en el ANEXO. Básicamente se realiza una instalación de Squid y se configura el rango de direcciones IP privadas origen de las que el proxy debe aceptar conexiones para así permitir su acceso a Internet. Este rango de direcciones se le pasa como parámetro al script a la hora de invocarlo, y dicho parámetro es una referencia al rango de IPs que tenga asignada la LAN de la casa en ese momento, de modo que, si en algún momento se decidiera cambiar el rango de direcciones IP en el fichero `variables.tf`, esto no afectaría al script en absoluto.

Para aprovisionar los periféricos conectados a la LAN de la casa se ha creado el script `docker-proxy-config.tftpl`, que instala Docker en las máquinas basadas en Linux en función de la distribución elegida. El script recibe como parámetro una referencia a la IP privada de la instancia configurada como proxy. Esta IP se utiliza para definir las variables `HTTP_PROXY` y `HTTPS_PROXY`, necesarias para el acceso a Internet. Una vez hecho esto, se emplean comandos Unix para determinar con qué distribución de Linux se está tratando (Debian o Ubuntu), ya que la instalación de Docker difiere en función de cual sea. Conocida la distribución, se hace uso de las variables del proxy para acceder a Internet y

descargar Docker. Por último, una vez se ha instalado Docker, se arranca el contenedor deseado según una imagen, unos argumentos y un tag que también se le pasan como parámetro al script a la hora de invocarlo y que permiten arrancar.

Cabe mencionar que la instalación de Docker no es necesaria para todos los periféricos. En el caso del ccu2 o la bombilla es una buena opción ya que existen imágenes Docker que simulan el comportamiento de estos dispositivos y que por tanto nos permitirían configurar el escenario de la forma deseada de una forma muy sencilla gracias al script. Por el contrario, para el smartphone y la tablet, que son equipos de usuario (y no un servicio) y tendrían un papel secundario en el escenario como puede ser la generación de tráfico, lo más adecuado sería construir una imagen personalizada basada en AndroidOS y desplegar las instancias a partir de ella, indicándolo en el fichero `main.tf`.

En cuanto al intercambio de tráfico, solo ha sido necesaria la definición de dos reglas:

allow-internal: esta regla se aplica a la VPC de la casa y permite el intercambio de tráfico entre los equipos de la LAN (acepta tráfico de todos los protocolos por todos los puertos siempre y cuando la IP origen esté dentro de la LAN, y al no especificarse destino se aplica a todas las instancias de la VPC).

allow-home-from-attacker: permite la entrada de tráfico procedente de la VPC del atacante hacia la VPC de la casa. Como se puede observar, no existe la regla *allow-attacker-from-home*, ya que tal y como se comentó en el estado del arte, la regla *allow-home-from-attacker* permite el tráfico de retorno hacia el atacante asociado a ella, lo que no hace necesaria la existencia de una regla que permita que el atacante acepte tráfico procedente de la casa.

Nombre	VPC	Dirección	Origen	Destino	Protocolos	Puertos	Acción
allow-internal	smart-home	INGRESS	10.10.10.0/24	-	TCP, UDP, ICMP	0-65535	ALLOW
allow-home-from-attacker	smart-home	INGRESS	10.10.20.0/24	-	TCP, UDP, ICMP	0-65535	ALLOW

Tabla 4.7: Reglas de FW del escenario Smart Home

4.2. ESCENARIOS DE RED

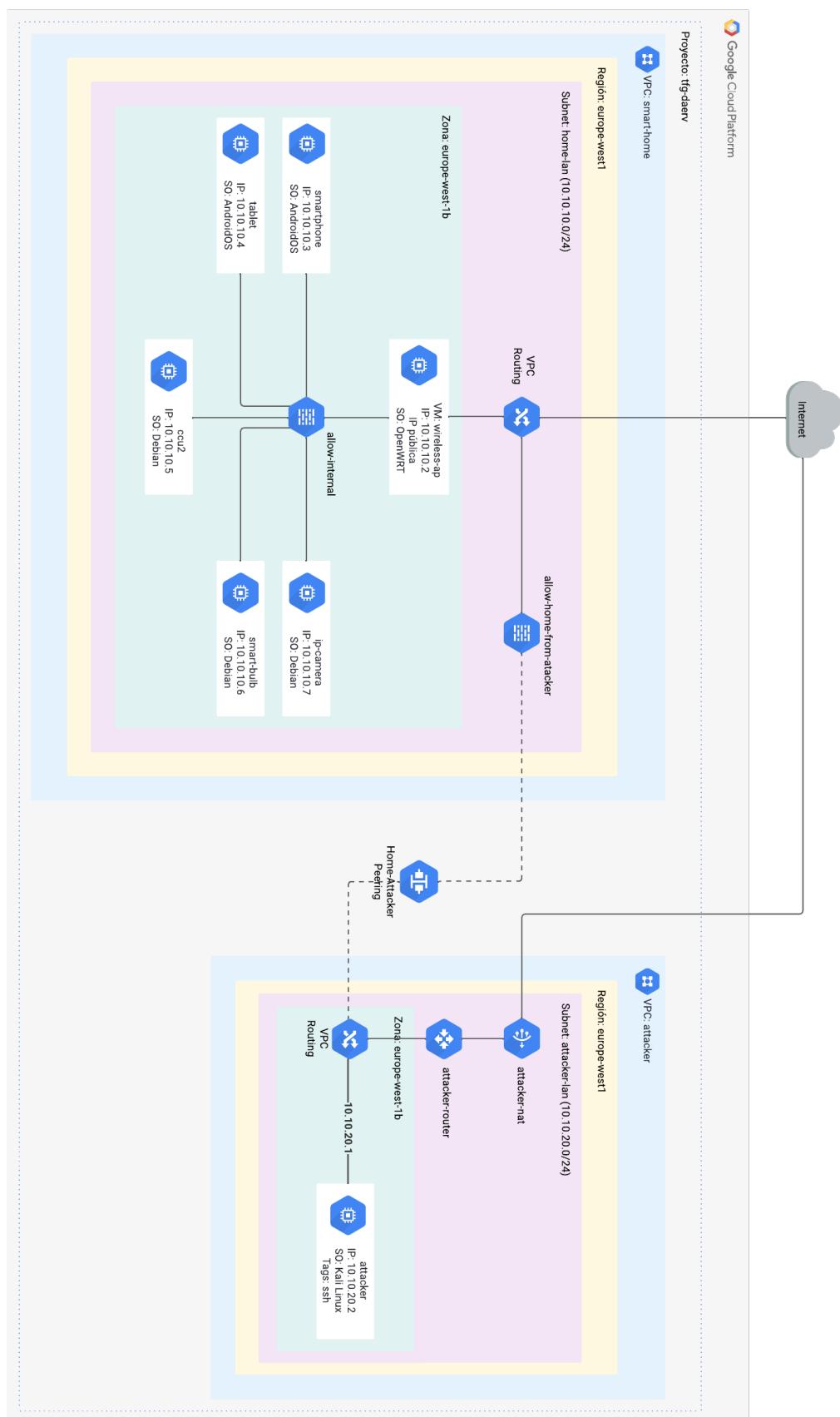


Figura 4.7: Implementación en GCP del escenario Smart Home

4.2.4. SCADA

Descripción

El sistema SCADA es una herramienta de automatización y control industrial utilizada en los procesos productivos que puede controlar, supervisar, recopilar datos, analizar datos y generar informes a distancia mediante una aplicación informática. Su principal función es la de evaluar los datos con el propósito de subsanar posibles errores. Los sistemas SCADA son cruciales para las empresas industriales, ya que ayudan a mantener la eficiencia, procesar los datos para tomar decisiones más inteligentes y comunicar los problemas del sistema para reducir el tiempo de inactividad.

Este escenario resulta de la agrupación de aplicaciones informáticas instaladas en un ordenador denominado Máster o MTU, destinado al control automático de una actividad productiva a distancia que está interconectada con otros instrumentos llamados de campo como son los autómatas programables (PLC) y las unidades terminales remotas (RTU). El HMI es la interfaz que conecta al hombre con la maquina presentando los datos del proceso ante el operario mediante un sistema de monitoreo. Además, controla la acción a desarrollar a través de una pantalla.

Figure 2: ISA95 levels applied to a SCADA architecture.

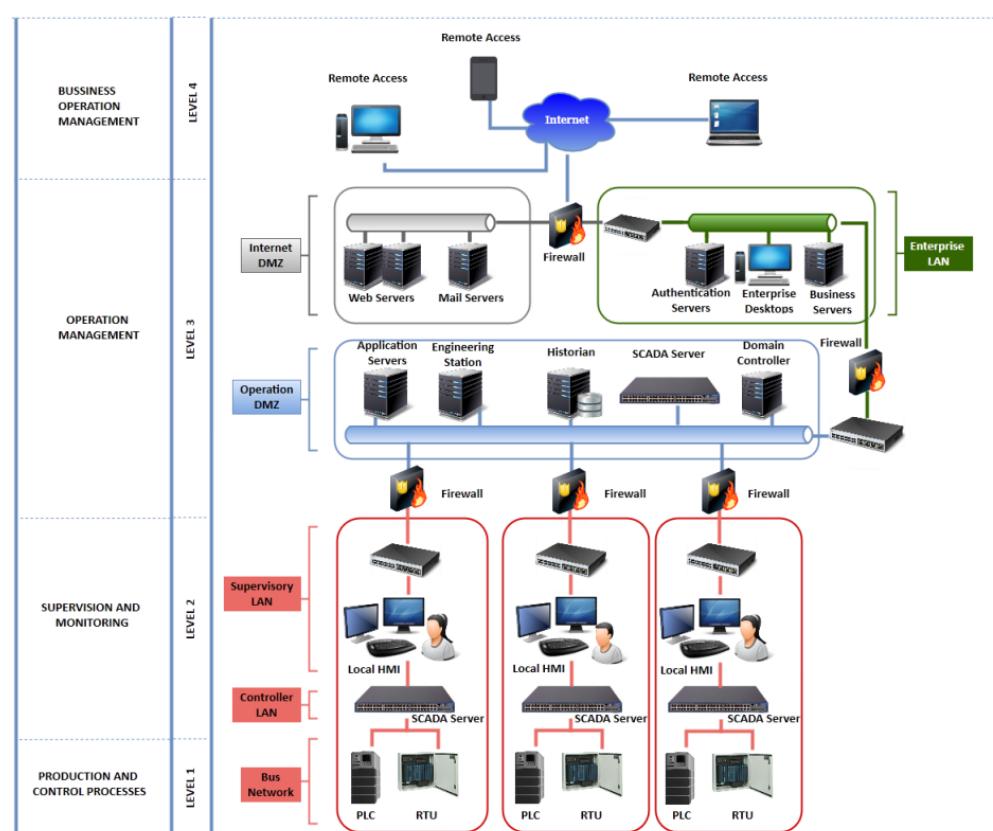


Figura 4.8: Topología del escenario SCADA

En la zona DMZ de la subred IT se encuentra un servidor web público que corre un Nginx desactualizado vulnerable a *integer overflow* (CVE-2017-7529) y que permitiría al atacante recopilar información para realizar un movimiento lateral. Una vez dentro de

4.2. ESCENARIOS DE RED

la red IT, el objetivo del atacante es llegar al HMI, que es el punto clave puesto que es quien habla con las MTU, quienes a su vez pasan los mensajes a las RTU. De esta forma, el atacante podría robar datos del SCADA server o mandar comandos para alterar el funcionamiento de los sistemas finales. La comunicación entre los equipos del bus (que no son atacables ya que no tienen conexión a internet ni son accesibles) se realiza mediante protocolos como IEC 60870-5-104, Modbus o DNP3. Cabe mencionar que existen imágenes Docker que permiten simular los protocolos Modbus y DNP3, mientras que para IEC sería necesaria una VM Windows.

Implementación

El escenario SCADA consta de una única VPC, en la que se pueden diferenciar dos grandes partes, la parte superior sería la parte IT (corporativa) y la inferior la parte OT (operativa). La VPC está segmentada en varias subredes: *internet-dmz*, *enterprise-lan*, y *operation-dmz* forman la parte correspondiente a la parte IT y -ot-lan simula la parte OT.

VPC	LAN	Rango IP	Equipos
scada	internet-dmz	10.10.10.0/24	web-server, mail-server
	enterprise-lan	10.10.20.0/24	auth-server, business-server, enterprise-desktop
	operation-dmz	10.10.30.0/24	app-server, engineering-station, historian, scada-server, domain-controller
	ot-lan	10.10.40.0/24	local-hmi-pro, scada-server-pro, plc-pro, rtu-pro, local-hmi-pre, scada-server-pre, plc-pre, rtu-pre

Tabla 4.8: Estructura del escenario SCADA

El tráfico entre todas las LAN de la VPC está limitado por reglas de FW. Además, dentro de la subred OT se distingue una red de producción y otra de preproducción, que no deben tener conexión entre sí. Para ello, dentro de estas, se han definido reglas de FW de forma que cada dispositivo solo se pueda comunicar con los de su nivel inmediatamente superior o inferior. Es decir, mientras que en las LAN de IT la comunicación entre los dispositivos que la componen es amplia, en la parte OT esta comunicación está mucho más restringida.

En cuanto al aprovisionamiento, se emplea el fichero `docker-provisioning.tftpl`. Este fichero tiene un funcionamiento igual al descrito en el escenario Smart Home, pero sin proxy, ya que esta VPC cuenta con Cloud Nat para proporcionar acceso a Internet a las instancias con dirección IP privada. Este script accede al contenido del fichero `os-release`, ubicado en el directorio `/etc/`, para determinar si el SO es Debian, Ubuntu, Centos o Fedora, instala Docker en función de dicho SO y arranca el contenedor deseado. Para arrancar el contenedor, a la hora de invocar al script se le pasan como argumentos la imagen Docker de la cual se quiere arrancar un contenedor, el tag (versión) de dicha imagen y argumentos opcionales como mapeo de puertos o un punto de entrada al contenedor.

Se detallan a continuación las reglas de firewall. Los servidores de correo y mail son públicos. Por eso las reglas *allow-web* y *allow-mail* permiten conexiones a estos desde

cualquier dirección IP a través de los puertos correspondientes, que en el caso del servidor web son el 80 y el 443 (http y https) y en el caso del servidor de correo el 465 y 587 (SMTP). El puerto 25, típico de SMTP, no se ha incluido ya que Google Cloud bloquea siempre las conexiones a este debido al riesgo de abuso. El resto de reglas hacen posible la conexión privada entre las tres subredes IT, la conexión entre la subred de operaciones y los hmi, la conexión entre cada hmi y su servidor SCADA y, finalmente, la comunicación interna entre el servidor SCADA, el PLC y el RTU. Nótese que se ha obviado la columna correspondiente a la VPC donde aplica la regla ya que sólo hay una VPC.

Nombre	Dirección	Origen	Destino	Protocolos	Puertos	Acción
allow-web	INGRESS	0.0.0.0/0	web-server (tag)	TCP	80, 443	ALLOW
allow-mail	INGRESS	0.0.0.0/0	mail-server (tag)	TCP	465, 587	ALLOW
allow-enterprise	INGRESS	10.10.10.0/24 10.10.20.0/24 10.10.30.0/24	enterprise (tag)	TCP, UDP, ICMP	0-65535	ALLOW
allow-operation	INGRESS	10.10.10.0/24 10.10.20.0/24 10.10.30.0/24	operation (tag)	TCP, UDP, ICMP	0-65535	ALLOW
allow-ops-from-hmi	INGRESS	hmi-pro (tag) hmi-pre (tag)	operation (tag)	TCP, UDP, ICMP	0-65535	ALLOW
allow-hmi-pro-from-ops	INGRESS	operation (tag)	hmi-pro (tag)	TCP, UDP, ICMP	0-65535	ALLOW
allow-hmi-pre-from-ops	INGRESS	operation (tag)	hmi-pre (tag)	TCP, UDP, ICMP	0-65535	ALLOW
allow-hmi-server-pro	INGRESS	hmi-pro (tag) scada-pro (tag)	hmi-pro (tag) scada-pro (tag)	TCP, UDP, ICMP	0-65535	ALLOW
allow-hmi-server-pre	INGRESS	hmi-pre (tag) scada-pre (tag)	hmi-pre (tag) scada-pre (tag)	TCP, UDP, ICMP	0-65535	ALLOW
allow-bus-pro	INGRESS	scada-pro (tag) plc-pro (tag) rtu-pro (tag)	scada-pro (tag) plc-pro (tag) rtu-pro (tag)	TCP, UDP, ICMP	0-65535	ALLOW
allow-bus-pre	INGRESS	scada-pre (tag) plc-pre (tag) rtu-pre (tag)	scada-pre (tag) plc-pre (tag) rtu-pre (tag)	TCP, UDP, ICMP	0-65535	ALLOW

Tabla 4.9: Reglas de FW del escenario SCADA

4.2. ESCENARIOS DE RED

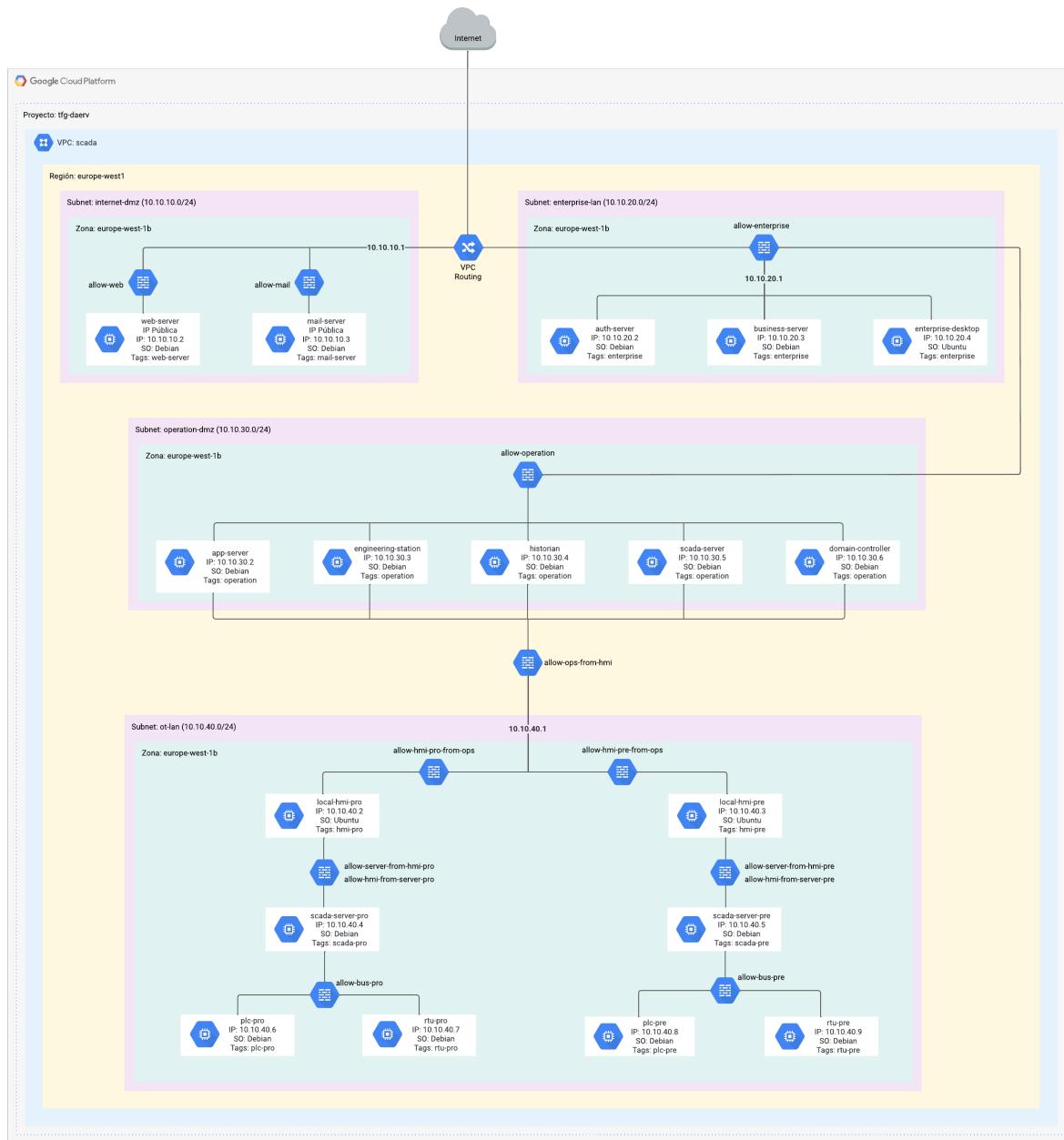


Figura 4.9: Implementación en GCP del escenario SCADA

Capítulo 5

Resultados y validación

Este capítulo presenta los resultados obtenidos tras el desarrollo del proyecto y su comprobación mediante pruebas. Para ello, para cada escenario se va a comprobar que la infraestructura se despliega correctamente y posteriormente se va a validar que la conectividad entre los equipos es la esperada.

5.1. Escenarios Smart Office

Se va a validar el despliegue de los escenarios Smart Office. Dado que la implementación es muy similar en ambos, se va a mostrar las pruebas realizadas sobre uno de ellos.

5.1.1. Despliegue de la infraestructura

Lo primero es navegar hasta el directorio en el que se encuentran los ficheros `.tf`. Una vez ahí, es necesario ejecutar la orden `terraform init` para que se configure el provider correctamente, seguida de la orden `terraform apply` para que Terraform elabore un plan para alcanzar el estado deseado. Tras ejecutar `terraform apply`, Terraform mostrará una lista de los recursos a crear, modificar o eliminar para cumplir con el estado que marcan los ficheros de configuración. Tras la lista, aparecerá un prompt en el que sólo si escribimos `yes` Terraform procede a ejecutar las acciones necesarias para el despliegue.

```
.../daerv/network-scenarios/Smart-Office-1 >>> terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/google from the dependency lock file
- Using previously-installed hashicorp/google v4.20.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

Figura 5.1: Ejecución del comando `terraform init`

5.1. ESCENARIOS SMART OFFICE

```
Plan: 31 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

Enter a value: yes

google_compute_network.office_external_network: Creating...
google_compute_network.office_internal_network: Creating...
google_compute_network.internal_server_network: Creating...
```

Figura 5.2: Prompt de confirmación tras la ejecución de `terraform apply`

Una vez se confirma la creación de los recursos, Terraform comienza a ejecutar el plan de creación de los recursos y va informando por la terminal sobre el estado de estos. Una vez se han creado todos, se nos informa y podemos ejecutar `terraform state list` para comprobar qué recursos hay desplegados:

```
google_compute_firewall.allow_external_from_external: Still creating... [20s el
google_compute_firewall.allow_server_from_printer: Still creating... [20s el
google_compute_firewall.allow_internal_from_external: Creation complete after 20s el
google_compute_firewall.allow_server_from_printer: Creation complete after 20s el

Apply complete! Resources: 31 added, 0 changed, 0 destroyed.
.../daerv/network-scenarios/Smart-Office-1 >>> terraform state list
google_compute_firewall.allow_external
google_compute_firewall.allow_external_from_internal
google_compute_firewall.allow_internal
google_compute_firewall.allow_internal_from_external
google_compute_firewall.allow_printer_from_server
google_compute_firewall.allow_server_from_printer
google_compute_firewall.allow_ssh_external
google_compute_firewall.allow_ssh_internal
google_compute_firewall.allow_ssh_server
google_compute_instance.attacker
google_compute_instance.cloud_printer_server
google_compute_instance.employee_pc_1
google_compute_instance.employee_pc_2
google_compute_instance.employee_pc_3
google_compute_instance.employee_remote_pc
google_compute_instance.office_printer
google_compute_network.internal_server_network
google_compute_network.office_external_network
google_compute_network.office_internal_network
google_compute_network_peering.peer_external_to_internal
google_compute_network_peering.peer_internal_office_to_server
google_compute_network_peering.peer_internal_server_to_office
google_compute_network_peering.peer_internal_to_external
google_compute_router.office_external_router
google_compute_router.office_internal_router
google_compute_router_nat.office_external_nat
google_compute_router_nat.office_internal_nat
google_compute_subnetwork.employees_lan
google_compute_subnetwork.office_external_lan
google_compute_subnetwork.printer_lan
google_compute_subnetwork.printer_server_lan
.../daerv/network-scenarios/Smart-Office-1 >>> |
```

Figura 5.3: Ejecución del comando `terraform state list`

También podemos comprobar en GCP el despliegue de los diferentes elementos que componen el escenario accediendo a los distintos servicios, tal y como se muestra a continuación.

Status	Name ↑	Zone	Recommendations	In use by	Internal IP
✓	attacker	europe-west1-b			10.10.40.2 (nic0)
✓	cloud-printer-server	europe-west1-b			10.10.30.2 (nic0)
✓	employee-pc-1	europe-west1-b			10.10.10.4 (nic0)
✓	employee-pc-2	europe-west1-b			10.10.10.2 (nic0)
✓	employee-pc-3	europe-west1-b			10.10.10.3 (nic0)
✓	employee-remote-pc	europe-west1-b			10.10.40.3 (nic0)
✓	office-printer	europe-west1-b			10.10.20.2 (nic0)

Figura 5.4: Instancias desplegadas en Google Compute Engine - Smart Office

Name ↑	Region	Subnets	MTU	Mode	Internal IP ranges	External IP ranges	Secondary IPv4 ranges	Gateways	Firewall Rules
default		35	1460	Auto	None				0
office-external-network		1	1460	Custom	None				3
	europe-west1	office-external-lan			10.10.40.0/24	None	None	10.10.40.1	
office-internal-network		2	1460	Custom	None				4
	europe-west1	employees-lan			10.10.10.0/24	None	None	10.10.10.1	
	europe-west1	printer-lan			10.10.20.0/24	None	None	10.10.20.1	
server-network		1	1460	Custom	None				2
	europe-west1	printer-server-lan			10.10.30.0/24	None	None	10.10.30.1	

Figura 5.5: VPCs desplegadas - Smart Office

Name	Type	Targets	Filters	Protocols/ports	Action	Priority	Network ↑
allow-external-from-internal	Ingress	employee	IP ranges: 10.10.20.0/24, 10.10.10.0/24	tcp:0-65535 udp:0-65535 icmp	Allow	1000	office-external-network
allow-external	Ingress	Apply to all	IP ranges: 10.10.40.0/24	tcp:0-65535 udp:0-65535 icmp	Allow	65534	office-external-network
allow-internal-from-external	Ingress	employee, printer	IP ranges: 10.10.40.2	tcp:0-65535 udp:0-65535 icmp	Allow	1000	office-internal-network
allow-printer-from-server	Ingress	printer	IP ranges: 10.10.30.0/24	tcp:0-65535 udp:0-65535 icmp	Allow	1000	office-internal-network
allow-internal	Ingress	Apply to all	IP ranges: 10.10.20.0/24, 10.10.10.0/24	tcp:0-65535 udp:0-65535 icmp	Allow	65534	office-internal-network
allow-server-from-printer	Ingress	server	IP ranges: 10.10.20.0/24	tcp:0-65535 udp:0-65535 icmp	Allow	1000	server-network

Figura 5.6: Reglas de FW aplicadas - Smart Office

5.2. ESCENARIO SMART HOME

5.1.2. Tests

Para comprobar que la conectividad entre los elementos es la deseada según las reglas de firewall definidas en la tabla 4.2 se han creado una serie de tests de conectividad haciendo uso del servicio Network Intelligence de GCP:

Name	Protocol	Source	Destination	Destination port	Last test time	Last packet transmission result
attacker-to-internal	tcp	attacker	office-printer	80	2022-06-21 (20:58:51)	🔴 0/50 packets delivered
external-to-server	tcp	attacker	cloud-printer-server	80	2022-06-21 (20:58:53)	🔴 0/50 packets delivered
internal-to-attacker	tcp	employee-pc-1	attacker	80	2022-06-21 (20:58:51)	🔴 0/50 packets delivered
internal-to-remote-employee	tcp	employee-pc-2	employee-remote-pc	80	2022-06-21 (20:58:51)	🟢 50/50 packets delivered
internal-to-server	tcp	employee-pc-2	cloud-printer-server	80	2022-06-21 (20:58:54)	🔴 0/50 packets delivered
printer-to-server	tcp	office-printer	cloud-printer-server	80	2022-06-21 (20:58:51)	🟢 50/50 packets delivered
remote-employee-to-internal	tcp	employee-remote-pc	employee-pc-1	80	2022-06-21 (20:58:54)	🟢 50/50 packets delivered
server-to-internal	tcp	cloud-printer-server	employee-pc-1	80	2022-06-21 (20:58:51)	🔴 0/50 packets delivered
server-to-printer	tcp	cloud-printer-server	office-printer	80	2022-06-21 (20:58:54)	🟢 50/50 packets delivered

Figura 5.7: Tests de conectividad - Smart Office

5.2. Escenario Smart Home

Se van a presentar las mismas pruebas, esta vez para el escenario Smart Home. En este apartado, así como en el siguiente, se va a obviar la parte correspondiente a los comandos de Terraform para evitar redundancia.

5.2.1. Despliegue de la infraestructura

Se muestra a continuación como la infrestructura se ha creado correctamente en GCP:

Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP
✓	attacker	europe-west1-b			10.10.20.2 (nic0)	
✓	ccu2	europe-west1-b			10.10.10.7 (nic0)	
✓	smartphone	europe-west1-b			10.10.10.2 (nic0)	
✓	tablet	europe-west1-b			10.10.10.5 (nic0)	
✓	wireless-ap	europe-west1-b			10.10.10.6 (nic0)	34.140.221.213 (nic0)

Figura 5.8: Instancias desplegadas en Google Compute Engine - Smart Home

Name ↑	Region	Subnets	MTU ?	Mode	Internal IP ranges	External IP ranges	Secondary IPv4 ranges	Gateways	Firewall Rules
▼ attacker-network		1	1460	Custom	None				1
	europe-west1	attacker-lan			10.10.20.0/24	None	None	10.10.20.1	
▶ default		35	1460	Auto	None				0
▼ smart-home		1	1460	Custom	None				3
	europe-west1	home-lan			10.10.10.0/24	None	None	10.10.10.1	

Figura 5.9: VPCs desplegadas - Smart Home

Name	Type	Targets	Filters	Protocols/ports	Action	Priority	Network	↑
allow-home-from-attacker	Ingress	Apply to all	IP ranges: 10.10.20.0/24	tcp:0-65535 udp:0-65535 icmp	Allow	1000	smart-home	
allow-internal	Ingress	Apply to all	IP ranges: 10.10.10.0/24	tcp:0-65535 udp:0-65535 icmp	Allow	1000	smart-home	

Figura 5.10: Reglas de FW aplicadas - Smart Home

5.2.2. Tests

En este escenario se han definido los siguientes tests:

Name	Protocol	Source	Destination	Destination port	Last test time	Last packet transmission result
attacker-to-home	tcp	attacker	smartphone	80	2022-06-21 (21:27:11)	🟢 50/50 packets delivered
home-to-attacker	tcp	ccu2	attacker	80	2022-06-21 (21:29:39)	🔴 0/50 packets delivered
internal	tcp	tablet	ip-camera	80	2022-06-21 (21:28:10)	🟢 50/50 packets delivered

Figura 5.11: Tests de conectividad - Smart Home

Adicionalmente, accedemos a una de las instancias de la red local de la casa a través de SSH (es necesario crear una regla de FW que lo permita) y vemos que tiene acceso a Internet a través de la IP pública de la instancia wireless-ap y que cuenta con Docker instalado, lo que verifica el funcionamiento de los scripts `proxy-config.tftpl` y `docker-proxy-provisioning.tftpl` pasados como parámetro:

```
garciasanchezsamu@ccu2:~$ which docker
/usr/bin/docker
garciasanchezsamu@ccu2:~$ curl -x http://10.10.10.6:3128 -L http://ip.me
34.140.221.213
garciasanchezsamu@ccu2:~$ █
```

Figura 5.12: Tests de aprovisionamiento - Smart Home

5.3. ESCENARIO SCADA

5.3. Escenario SCADA

Al igual que los apartados anteriores, se va a presentar el despliegue y los tests correspondientes al escenario SCADA.

5.3.1. Despliegue de la infraestructura

A continuación se muestra, al igual que para el resto de escenarios, imágenes que verifican el despliegue de la infraestructura en Google Cloud.

Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP
✓	app-server	europe-west1-b			10.10.30.6 (nic0)	
✓	auth-server	europe-west1-b			10.10.20.3 (nic0)	
✓	business-server	europe-west1-b			10.10.20.4 (nic0)	
✓	domain-controller	europe-west1-b			10.10.30.2 (nic0)	
✓	engineering-station	europe-west1-b			10.10.30.3 (nic0)	
✓	enterprise-desktop	europe-west1-b			10.10.20.2 (nic0)	
✓	historian	europe-west1-b			10.10.30.5 (nic0)	
✓	local-hmi-pre	europe-west1-b			10.10.40.9 (nic0)	
✓	local-hmi-pro	europe-west1-b			10.10.40.8 (nic0)	
✓	mail-server	europe-west1-b			10.10.10.3 (nic0)	35.205.15.8 (nic0)
✓	plc-pre	europe-west1-b			10.10.40.3 (nic0)	
✓	plc-pro	europe-west1-b			10.10.40.5 (nic0)	
✓	rtu-pre	europe-west1-b			10.10.40.4 (nic0)	
✓	rtu-pro	europe-west1-b			10.10.40.6 (nic0)	
✓	scada-server	europe-west1-b			10.10.30.4 (nic0)	
✓	scada-server-pre	europe-west1-b			10.10.40.7 (nic0)	
✓	scada-server-pro	europe-west1-b			10.10.40.2 (nic0)	
✓	web-server	europe-west1-b			10.10.10.2 (nic0)	35.195.136.46 (nic0)

Figura 5.13: Instancias desplegadas en Google Compute Engine - SCADA

5cm									
Name ↑	Region	Subnets	MTU	Mode	Internal IP ranges	External IP ranges	Secondary IPv4 ranges	Gateways	Firewall Rules
default	35	1460	Auto	None					0
scada-network	4	1460	Custom	None					13
	europe-west1	enterprise-lan			10.10.20.0/24	None	None	10.10.20.1	
	europe-west1	internet-dmz			10.10.10.0/24	None	None	10.10.10.1	
	europe-west1	operation-dmz			10.10.30.0/24	None	None	10.10.30.1	
	europe-west1	ot-lan			10.10.40.0/24	None	None	10.10.40.1	

Figura 5.14: VPCs desplegadas - SCADA

Name	Type	Targets	Filters	Protocols/ports	Action	Priority	Network ↑
allow-bus-pre	Ingress	scada-pre, rtu-pre, plc-pre	Tags: scada-pre, rtu-pre, plc-pre	tcp:0-65535 udp:0-65535 icmp	Allow	65534	scada-network
allow-bus-pro	Ingress	rtu-pro, scada-pro, plc-pro	Tags: rtu-pro, scada-pro, plc-pro	tcp:0-65535 udp:0-65535 icmp	Allow	65534	scada-network
allow-enterprise	Ingress	enterprise	IP ranges: 10.10.30.0/24, 10.10.20.0/24, 10.10.10.0/24	tcp:0-65535 udp:0-65535 icmp	Allow	65534	scada-network
allow-hmi-from-server-pre	Ingress	hmi-pre	Tags: scada-pre	tcp:0-65535 udp:0-65535 icmp	Allow	65534	scada-network
allow-hmi-from-server-pro	Ingress	hmi-pro	Tags: scada-pro	tcp:0-65535 udp:0-65535 icmp	Allow	65534	scada-network
allow-hmi-pre-from-ops	Ingress	hmi-pre	Tags: operation	tcp:0-65535 udp:0-65535 icmp	Allow	65534	scada-network
allow-hmi-pro-from-ops	Ingress	hmi-pro	Tags: operation	tcp:0-65535 udp:0-65535 icmp	Allow	65534	scada-network
allow-mail	Ingress	mail-server	IP ranges: 0.0.0.0/0	tcp:25, 465, 587	Allow	65534	scada-network
allow-operation	Ingress	operation	IP ranges: 10.10.30.0/24, 10.10.20.0/24, 10.10.10.0/24	tcp:0-65535 udp:0-65535 icmp	Allow	65534	scada-network
allow-ops-from-hmi	Ingress	operation	Tags: hmi-pro, hmi-pre	tcp:0-65535 udp:0-65535 icmp	Allow	65534	scada-network
allow-server-from-hmi-pre	Ingress	scada-pre	Tags: hmi-pre	tcp:0-65535 udp:0-65535 icmp	Allow	65534	scada-network
allow-server-from-hmi-pro	Ingress	scada-pro	Tags: hmi-pro	tcp:0-65535 udp:0-65535 icmp	Allow	65534	scada-network
allow-web	Ingress	web-server	IP ranges: 0.0.0.0/0	tcp:80, 443	Allow	65534	scada-network

Figura 5.15: Reglas de FW aplicadas - SCADA

5.3.2. Tests

Los siguientes tests de conectividad han sido definidos:

Name	Protocol	Source	Destination	Destination port	Last test time	Last packet transmission result	Last configuration analysis result
bus-to-hmi	tcp	rtu-pre	local-hmi-pre	80	2022-06-21 (22:43:54)	🔴 0/50 packets delivered	🔴 Unreachable
bus-to-scada	tcp	plc-pro	scada-server-pro	80	2022-06-21 (22:44:23)	🟢 50/50 packets delivered	🟢 Reachable
enterprise-to-operation	tcp	auth-server	domain-controller	80	2022-06-21 (22:38:02)	🟢 50/50 packets delivered	🟢 Reachable
mail-access	tcp	0.0.0.0	mail-server	587	2022-06-21 (22:23:21)	—	🟢 Reachable
operation-to-enterprise	tcp	engineering-station	business-server	80	2022-06-21 (22:38:52)	🟢 50/50 packets delivered	🟢 Reachable
ot-to-enterprise	tcp	local-hmi-pro	enterprise-desktop	80	2022-06-21 (22:40:14)	🔴 0/50 packets delivered	🔴 Unreachable
ot-to-operation	tcp	local-hmi-pro	historian	80	2022-06-21 (22:41:02)	🟢 50/50 packets delivered	🟢 Reachable
plc-to-rtu	tcp	plc-pro	rtu-pro	80	2022-06-21 (22:41:46)	🟢 50/50 packets delivered	🟢 Reachable
pre-to-pro	tcp	local-hmi-pre	local-hmi-pro	80	2022-06-21 (22:42:10)	🔴 0/50 packets delivered	🔴 Unreachable
scada-to-hmi	tcp	scada-server-pro	local-hmi-pro	80	2022-06-21 (22:45:19)	🟢 50/50 packets delivered	🟢 Reachable
web-access	tcp	0.0.0.0	web-server	80	2022-06-21 (22:22:45)	—	🟢 Reachable

Figura 5.16: Test de conectividad - SCADA

5.3. ESCENARIO SCADA

Por último se verifica el funcionamiento del script `docker-provisioning.tftpl`. Para ello accedemos a la IP pública del servidor web desde el ordenador host y comprobamos que está ejecutando el software `nginx` en el puerto 80.

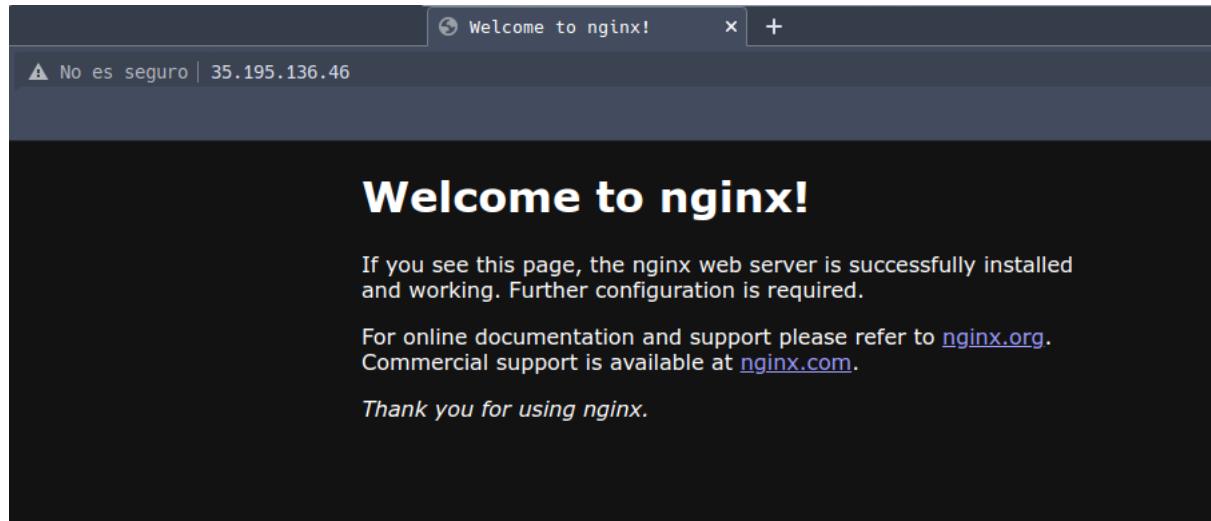


Figura 5.17: Test de aprovisionamiento - SCADA

Capítulo 6

Conclusiones y líneas futuras

Este capítulo expone las conclusiones extraídas a raíz de la realización del proyecto y las líneas futuras en las que se puede seguir trabajando.

6.1. Conclusiones

En el último año se ha experimentado un más que notable aumento del número de ciberataques y sofisticación de las amenazas conocidas. Es por esto que en la actualidad existe una gran demanda de perfiles de ciberseguridad. Este trabajo surge con la idea de hacer accesible y sencilla la práctica de ejercicios de tipo Red Team para la formación en ciberejercicios.

Con el desarrollo planteado en este trabajo de fin de grado, el cual ha alcanzado los objetivos *O1*, *O2*, *O3* y *O4* planteados en la tabla 1.1 del capítulo 1, se ha demostrado que es posible el despliegue de escenarios de red virtualizados modelables mediante variables y configurables tanto a nivel de infraestructura como de software de forma sencilla y centralizada.

El hecho de desplegar los escenarios en la nube es muy ventajoso respecto a los clásicos Cyber Range on-premises, pues los recursos se asignan bajo demanda, de forma que se paga sólo por lo que se usa y durante el tiempo que se usa, mientras que en un Cyber Range físico se ha realizado una inversión en equipos que hay que costear independientemente de si se usan o no. Además, la nube es fácilmente escalable, ya que se van asignando recursos según se van necesitando sin necesidad de interacción humana, mientras que en las instalaciones on-premises hay asociados unos costes de operación así como de tiempo en tener todo listo. El despliegue en cloud es también más seguro (es actualizado por el proveedor cloud con frecuencia, lo que conlleva también una reducción de costes en este aspecto) y configurable (permite una adaptación rápida y sencilla a cualquier topología o configuración software).

Además, el uso de Terraform permite que este despliegue se realice de forma automatizada y en cuestión de segundos. Estos aspectos, junto con la fácil sintaxis, la portabilidad entre proveedores cloud, el uso de ficheros de variables para parametrizar el despliegue y las funciones `file` y `templatefile` para ejecutar scripts de configuración personalizados

6.2. LÍNEAS FUTURAS

en las instancias, hacen de Terraform una elección idónea para la orquestación de los escenarios.

No obstante, el uso dichas tecnologías también derivó en algunos problemas, pero se encontraron soluciones alternativas, dando como resultado un sistema que cumple con los objetivos establecidos. Por ejemplo, en GCP, los scripts pasados a las instancias haciendo uso de la función `templatefile` se ejecutan como root antes de que haya ningún usuario creado ni logueado. Esto impide establecer variables de entorno en las instancias automáticamente en el arranque, de ahí la obligación de crear un segundo script para aprovisionar las instancias a través del proxy. Ambos scripts siguen la misma lógica, pero en este último, antes de cada orden fue necesario añadir el valor de la variable de entorno, que se pasaba como parámetro desde Terraform. Esto se puede apreciar con más detalle en el Anexo G.

A nivel personal, este trabajo me ha permitido adquirir conocimientos en tecnologías muy demandadas hoy en día y que son el futuro del sector por las grandes ventajas que proporcionan.

6.2. Líneas futuras

Tras el desarrollo del proyecto, en esta sección se plantean posibles mejoras y líneas sobre las que trabajar en el futuro, con la intención de mejorar en la parametrización y configuración de los escenarios para hacerlo lo más intuitivo y fácil de usar posible.

Este trabajo cubre los aspectos relacionados con el despliegue automatizado de los escenarios, así como el diseño y configuración de las conexiones entre los equipos. La principal línea a seguir sería la configuración software de dichos equipos, de forma que cumplan su papel en el escenario, obteniéndose así entornos funcionales en los que practicar tests de intrusión específicos. Dentro de esta configuración software distinguimos dos vías, una sería la configuración de servicios vulnerables mediante Docker, y la otra sería la creación de imágenes base personalizadas en GCP para los equipos de usuario.

En cuanto a la configuración de los servicios, los ficheros que se proporcionan en el directorio `template-files` permiten aprovisionar las máquinas con contenedores Docker de forma sencilla tan sólo especificando algunos parámetros. De esta forma, es posible pasar imágenes vulnerables como las disponibles en el proyecto Vulnhub [30] como argumento a la instancia de Google Cloud a la hora de construirla, teniendo así un entorno funcional en el que practicar ejercicios de Red Team.

Se plantea también como trabajo futuro la evolución de esta herramienta para que permita al usuario elegir en qué proveedor cloud desea desplegar los escenarios. Esto se conoce como capacidad multinube, y permitiría aumentar la disponibilidad y la flexibilidad, pudiendo elegir la nube que mejor se adapte a las necesidades concretas de cada escenario.

Bibliografía

- [1] VMWare. *¿En qué consiste la virtualización?* [Online]. URL: <https://www.vmware.com/es/solutions/virtualization.html>.
- [2] Microsoft. *¿Qué es virtualización?* [Online]. URL: <https://azure.microsoft.com/es-es/overview/what-is-virtualization/>.
- [3] Red Hat. *¿Qué es un hipervisor?* [Online]. URL: <https://www.redhat.com/es/topics/virtualization/what-is-a-hypervisor>.
- [4] Linux KVM. *Documentación Oficial.* [Online]. URL: <https://www.linux-kvm.org/>.
- [5] VirtualBox. *Documentación Oficial.* [Online]. URL: <https://www.virtualbox.org/>.
- [6] VMware. *Documentación Oficial.* [Online]. URL: <https://www.vmware.com/>.
- [7] MuyComputer. *¿Cuáles son las diferencias entre VirtualBox, VMWare e Hyper-V?* [Online]. URL: <https://www.muycomputer.com/2020/03/27/hipervisor-virtualbox-vmware-hyperv/>.
- [8] Microsoft. *¿Qué es un contenedor?* [Online]. URL: <https://azure.microsoft.com/es-es/overview/what-is-a-container/>.
- [9] S. K. S. “Practical LXC and LXD: Linux Containers for Virtualization and Orchestration”. En: (2017). 1st ed.
- [10] NGINX. *What Are Namespaces and cgroups, and How Do They Work?* [Online]. URL: <https://www.nginx.com/blog/what-are-namespaces-cgroups-how-do-they-work/>.
- [11] Linux Containers. *Documentación oficial.* [Online]. URL: <https://linuxcontainers.org/>.
- [12] Docker. *Documentación oficial.* [Online]. URL: <https://www.docker.com/>.
- [13] Redes Zone. *Docker y cómo funciona la virtualización de contenedores.* [Online]. URL: <https://www.redeszone.net/2016/02/24/docker-funciona-la-virtualizacion-contenedores/>.
- [14] Red Hat. *¿Qué es la infraestructura como código (IaC)?* [Online]. URL: <https://www.redhat.com/es/topics/automation/what-is-infrastructure-as-code-iac>.
- [15] Vagrant. *Documentación oficial.* [Online]. URL: <https://www.vagrantup.com/>.
- [16] Red Hat. *¿Qué es la gestión de la configuración?* [Online]. URL: <https://www.redhat.com/es/topics/automation/what-is-configuration-management>.
- [17] Progress Chef. *Documentación oficial.* [Online]. URL: <https://www.chef.io/products/chef-infra>.
- [18] Runebook. *Visión general de Chef Infra.* [Online]. URL: https://runebook.dev/es/docs/chef/chef_overview/index.

- [19] Ansible. *Documentación oficial*. [Online]. URL: <https://docs.ansible.com/>.
- [20] Red Hat. *¿Qué es la preparación?* [Online]. URL: <https://www.redhat.com/es/topics/automation/what-is-provisioning>.
- [21] Docker Compose. *Documentación oficial*. [Online]. URL: <https://docs.docker.com/compose/>.
- [22] Kubernetes. *Documentación oficial*. [Online]. URL: <https://kubernetes.io/>.
- [23] Terraform. *Documentación oficial*. [Online]. URL: <https://www.terraform.io/>.
- [24] Deloitte. *¿Qué es Terraform?* [Online]. URL: <https://www2.deloitte.com/es/es/blog/todo-tecnologia/2021/que-es-terraform.html>.
- [25] Google Cloud Platform. *Documentación oficial*. [Online]. URL: <https://cloud.google.com/>.
- [26] Google Cloud Platform. *Documentación de Compute Engine*. [Online]. URL: <https://cloud.google.com/compute/docs/instances>.
- [27] Google Cloud Platform. *Documentación sobre la nube privada virtual*. [Online]. URL: <https://cloud.google.com/vpc/docs/vpc>.
- [28] Google Cloud Platform. *Compute Engine API*. [Online]. URL: <https://console.cloud.google.com/apis/library/compute.googleapis.com>.
- [29] Google Cloud Platform. *Cuentas de servicio*. [Online]. URL: <https://console.cloud.google.com/iam-admin/serviceaccounts>.
- [30] Vulhub. *Pre-built vulnerable docker environments*. [Online]. URL: <https://github.com/vulhub/vulhub>.
- [31] COIT. *Mapa socio-profesional del titulado en Ingeniería de Telecomunicación*. [Online]. URL: https://www.coit.es/sites/default/files/informes/pdf/mapa-de-titulado-de-ingeneria-de-telecomunicacion_0.pdf.

Anexos

Apéndice A

Aspectos éticos, económicos, sociales y ambientales

En este anexo se va a realizar una descripción de los aspectos éticos, económicos, sociales y ambientales surgidos de la realización del trabajo.

Introducción

Este trabajo de fin de grado está integrado en el proyecto COBRA de la ETSIT, y su objetivo es optimizar la forma en la que se realiza actualmente el despliegue de los Cyber Range, en términos de velocidad, escalabilidad, portabilidad y seguridad.

Desde un punto de vista social, el principal colectivo afectado o asociado a la realización de este proyecto estaría estrechamente relacionado con el ámbito de la ETSIT. Considerando los planos económico, medioambiental y ético, el trabajo aporta valor en todos ellos, tal y como se va a detallar a continuación.

Impactos relacionados con el proyecto

En primer lugar, el sistema desarrollado proporcionará una mayor productividad y calidad en la formación de los alumnos que cursen asignaturas de ciberseguridad así como programas de estudios relacionados con ella. Como ya se ha comentado en secciones anteriores, el despliegue en la nube supone un gran ahorro económico, ya que se paga únicamente por los recursos que se usan y durante el tiempo que se usan, de forma que en el caso de que no hubiera nadie practicando, no habría costes asociados. Además, es un entorno seguro, pues los escenarios son desplegados en una infraestructura completamente independiente del entorno de la organización, lo cual evita cualquier conflicto ético al no haber información o recursos sensibles de por medio...

En cuanto al plano medioambiental, y estrechamente relacionado con el económico ya comentado, esta herramienta evita la compra innecesaria de dispositivos específicos para su empleo y uso. Esto se debe a que el sistema está completamente virtualizado, y por tanto es posible su despliegue completo en la nube, siendo por tanto compatible con

cualquier tipo de ordenador, con independencia de su sistema operativo. Este aspecto permite reducir la huella ambiental al reducir los residuos a largo plazo.

Adicionalmente, al estar el proyecto COBRA ligado al programa de gobierno COINCIDENTE, el sistema tendrá gran importancia de cara a la sociedad española, pues unos resultados positivos derivados del éxito del trabajo permitirían una mejor formación en los profesionales de ciberseguridad de España, lo que a su vez mejoraría la calidad de vida de las personas.

Análisis detallado de uno de los principales impactos

Tras la exposición en la sección anterior de los impactos más relevantes relacionados con el proyecto, podemos afirmar que uno de los más influyentes sería el plano económico-medioambiental asociado al despliegue en Cloud.

El hecho de poder virtualizar cualquier tipo de sistema proporciona una gran flexibilidad a la hora de definir escenarios completamente distintos entre sí y evita la compra de equipos específicamente para ello. Un Cyber Range que se despliega haciendo uso de equipos físicos corre el riesgo de que haya períodos de tiempo en los que no se use, lo cual no exime del pago de la infraestructura que se emplea. Mientras tanto, en la nube tanto el despliegue como el pago de la infraestructura es bajo demanda.

Además, respecto al uso de GCP como proveedor cloud, cabe destacar que Google no tiene emisiones de carbono desde 2007 y se comprometió a operar con energía neutra de carbono en un 100 % las 24 horas, todos los días, para el año 2030. Los centros de datos de Google, incluidos los que ejecutan los servicios de Google Cloud, también usan mucho menos energía que los centros de datos típicos. Por este motivo, el uso de Google Cloud ayuda a reducir la huella de carbono de la infraestructura IT.

Conclusiones

A lo largo del desarrollo del proyecto se han realizado decisiones sobre requisitos y objetivos a cumplir que han sido claves para el alcance de un ecosistema que favorece al desarrollo sostenible y a la mejora de la sociedad.

Es por esto que podemos llegar a la conclusión de que todas las personas y organizaciones involucradas en el proyecto obtienen un beneficio e impacto positivo como consecuencia del uso del sistema desarrollado.

Apéndice B

Presupuesto económico

Este anexo presenta una estimación de los costes asociados a la realización de este trabajo de fin de grado y que han sido necesarios para la consecución de los objetivos. Se van a tener en cuenta para ello tanto los materiales utilizados como la mano de obra, a partir de los cuales se realizará una aproximación de los costes directos e indirectos.

Costes de mano de obra

Para estimar el coste asociado a la mano de obra, en primer lugar se ha realizado una aproximación del número de horas empleadas en las diferentes etapas de realización del proyecto. Esta aproximación se muestra con detalle en la tabla inferior.

Subproceso	Horas
Estudio de las diferentes tecnologías (Terraform, Google Cloud)	90
Análisis de los escenarios (vector de ataque y conexiones necesarias entre los elementos)	40
Diseño e implementación de los escenarios en Google Cloud	130
Desarrollo de herramientas de configuración	30
Pruebas	40
Elaboración de la memoria	75
Total	405

Tabla B.1: Horas asociadas a cada una de los subprocesos de realización del proyecto

Una vez obtenida una aproximación de las horas empleadas en todo el proceso de diseño y desarrollo del sistema, se va a considerar el salario medio anual de un Ingeniero de Telecomunicación junior para estimar el coste de mano de obra. Según el informe llevado a cabo por el Colegio Oficial de Ingenieros de Telecomunicación (COIT) en su Mapa socio-profesional del titulado en Ingeniería de Telecomunicación [31], el salario medio anual de los titulados entre 18 y 24 años es de 23.553€, que con un contrato de 40h semanales y estimando 48 semanas laborales anuales resultaría en 12.27€/hora. Con este dato podemos finalmente estimar el coste de la mano de obra:

Costes de mano de obra		
Horas	€/Hora	Total
405	12.27	4.969,35€

Tabla B.2: Costes de mano de obra

Costes materiales

Para los costes materiales se tendrá en cuenta el ordenador personal con el que se ha realizado todo el desarrollo. Dicho equipo es un HP Pavilion 15-cs0012ns, que cuenta con un procesador Intel Core i7, una pantalla de 15.6z 16GB de memoria RAM como algunas de sus características principales. Este ordenador tiene un coste aproximado de 1.100€. Además, se ha calculado la amortización mensual en base a los 4 años de antigüedad que tiene y esta se ha multiplicado por el número de meses en los que se ha estado trabajando en el proyecto (Octubre-Junio) para obtener la amortización total.

También se ha de tener en cuenta que para la simulación de los escenarios fue necesario el uso de servicios de GCP que quedan fuera de la capa gratuita que ofrece Google y que por tanto tienen un coste asociado al tiempo que se han usado durante el desarrollo y las pruebas. En la imagen inferior se adjunta un reporte de los gastos en dichos servicios a lo largo del tiempo, acompañados del montante total.

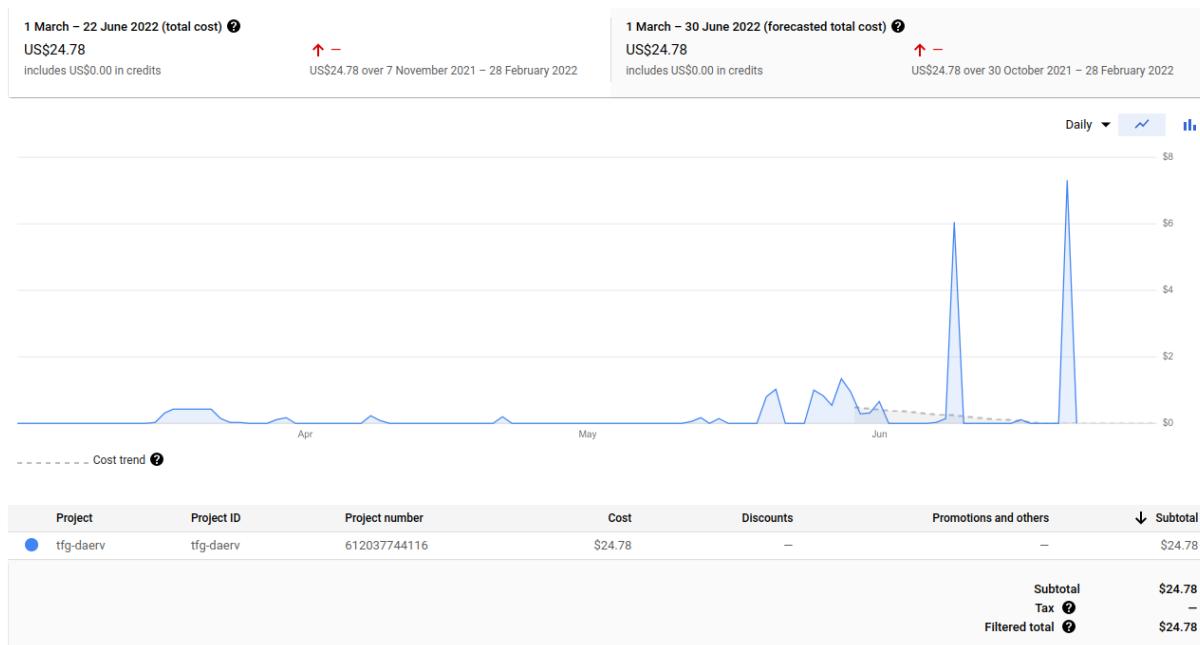


Figura B.1: Resumen de pagos en servicios de GCP

Como se puede apreciar, el gasto total ha sido de 24.78\$, que con la equivalencia actual (1€=1.06USD) serían 23,41€. Este dato nos permite cerrar los costes materiales como se puede ver a continuación.

Costes materiales					
Material	Tiempo de vida (años)	Precio	Amortización (€/mes)	Uso (meses)	Total
Ordenador portátil	4	1.100€	22.92	8	183,36€
Servicios GCP	-	23,41€	-	-	23,41€
Total					206,77€

Tabla B.3: Costes materiales

Costes indirectos

Se han estimado los costes indirectos (como pueden ser luz, agua o internet) en un 20 % sobre la suma de los costes de mano de obra y material:

Costes indirectos		
Coste directo	Estimación	Total
5.176,12€	20 %	1.035,224€

Tabla B.4: Costes indirectos

Costes totales

Para terminar, se incluyen el IVA y el beneficio industrial para obtener los costes totales asociados al proyecto:

Costes totales		
Concepto	Coste	
Costes directos	Mano de obra	4.969,35€
	Material	206,77€
	Total	5.176,12€
Costes indirectos		1.035,22€
Total antes de beneficios e impuestos		6.211,34€
Beneficio industrial (10 %)		621,13€
Total antes de impuestos		6.832,48€
IVA (21 %)		1.434,82€
Total		8.267,30€

Tabla B.5: Costes totales

Apéndice C

Ficheros de configuración del escenario Smart Office 1

Este anexo incluye todos los ficheros empleados para la configuración de la infrestructura del escenario Smart Office 1. El fichero main.tf contiene configuraciones generales, como son la configuración del provider de google, los peerings/VPN entre las VPC, o las reglas de firewall que controlan el tráfico que se intercambia entre las instancias. Adicionalmente, a fin de simplificar la lectura del código, hay un fichero .tf por cada una de las VPC, que contiene el código correspondiente a los elementos de red que la componen. Finalmente, en los ficheros variables.tf y terraform.tfvars se encuentran definidas las variables que se emplean en el resto del código.

main.tf

```
1 provider "google" {
2   project = var.project_id
3   region  = var.region
4   zone    = var.zone
5 }
6
7 resource "google_compute_network_peering" "peer_external_to_internal" {
8   name          = "peer-external-to-internal"
9   network       = google_compute_network.office_external_network.
10  self_link
11  peer_network = google_compute_network.office_internal_network.
12  self_link
13 }
14
15 resource "google_compute_network_peering" "peer_internal_to_external" {
16   name          = "peer-internal-to-external"
17   network       = google_compute_network.office_internal_network.
18   self_link
19   peer_network = google_compute_network.office_external_network.
20   self_link
21 }
22
23 resource "google_compute_network_peering" "
24   peer_internal_office_to_server" {
```

```

20      name      = "peer-internal-office-to-server"
21      network   = google_compute_network.office_internal_network.
22          self_link
23      peer_network = google_compute_network.internal_server_network.
24          self_link
25 }
26
27 resource "google_compute_network_peering" "peer_internal_server_to_office" {
28     name      = "peer-internal-server-to-office"
29     network   = google_compute_network.internal_server_network.
30         self_link
31     peer_network = google_compute_network.office_internal_network.
32         self_link
33 }
34
35 # resource "google_compute_firewall" "allow_ssh_external" {
36 #     name      = "allow-ssh-external"
37 #     network   = google_compute_network.office_external_network.self_link
38 #     project   = var.project_id
39 #     priority  = 65534
40 #     direction = "INGRESS"
41 #
42 #     allow {
43 #         protocol = "tcp"
44 #         ports    = ["22"]
45 #     }
46 #
47 #     source_ranges = ["0.0.0.0/0"]
48 #     target_tags   = ["ssh"]
49 # }
50
51
52 resource "google_compute_firewall" "allow_internal" {
53     name      = "allow-internal"
54     network   = google_compute_network.office_internal_network.self_link
55     project   = var.project_id
56     priority  = 65534
57     direction = "INGRESS"
58
59     allow {
60         protocol = "icmp"
61     }
62
63     allow {
64         protocol = "tcp"
65         ports    = ["0-65535"]
66     }
67
68     allow {
69         protocol = "udp"
70         ports    = ["0-65535"]
71     }
72
73     source_ranges = [google_compute_subnetwork.employees_lan.ip_cidr_range,
74                      google_compute_subnetwork.printer_lan.ip_cidr_range]
75 }

```

```

71
72 resource "google_compute_firewall" "allow_external" {
73   name      = "allow-external"
74   network   = google_compute_network.office_external_network.self_link
75   project   = var.project_id
76   priority  = 65534
77   direction = "INGRESS"
78
79   allow {
80     protocol = "icmp"
81   }
82
83   allow {
84     protocol = "tcp"
85     ports    = ["0-65535"]
86   }
87
88   allow {
89     protocol = "udp"
90     ports    = ["0-65535"]
91   }
92
93   source_ranges = [google_compute_subnetwork.office_external_lan.
94     ip_cidr_range]
95 }
96
96 resource "google_compute_firewall" "allow_external_from_internal" {
97   name      = "allow-external-from-internal"
98   network   = google_compute_network.office_external_network.self_link
99   project   = var.project_id
100  priority = 1000
101  direction = "INGRESS"
102
103  allow {
104    protocol = "tcp"
105    ports    = ["0-65535"]
106  }
107
108  allow {
109    protocol = "udp"
110    ports    = ["0-65535"]
111  }
112
113  allow {
114    protocol = "icmp"
115  }
116
117  source_ranges = [google_compute_subnetwork.printer_lan.ip_cidr_range,
118    google_compute_subnetwork.employees_lan.ip_cidr_range]
119  target_tags   = ["employee"]
120 }
121
121 resource "google_compute_firewall" "allow_internal_from_external" {
122   name      = "allow-internal-from-external"
123   network   = google_compute_network.office_internal_network.self_link
124   project   = var.project_id
125   priority  = 1000

```

```

126 direction = "INGRESS"
127
128 allow {
129   protocol = "tcp"
130   ports    = ["0-65535"]
131 }
132
133 allow {
134   protocol = "udp"
135   ports    = ["0-65535"]
136 }
137
138 allow {
139   protocol = "icmp"
140 }
141
142 sourceRanges = [googleComputeInstance.employeeRemotePC.
143   networkInterface[0].networkIP]
144 targetTags   = ["employee", "printer"]
145 }
146
147 resource "googleComputeFirewall" "allow_printer_from_server" {
148   name        = "allow-printer-from-server"
149   network     = googleComputeNetwork.officeInternalNetwork.selfLink
150   project     = var.project_id
151   priority    = 1000
152   direction   = "INGRESS"
153
154   allow {
155     protocol = "tcp"
156     ports    = ["0-65535"]
157   }
158
159   allow {
160     protocol = "udp"
161     ports    = ["0-65535"]
162   }
163
164   allow {
165     protocol = "icmp"
166   }
167
168   sourceRanges = [googleComputeSubnetwork.printerServerLan.
169     ipCidrRange]
170   targetTags   = ["printer"]
171 }
172
173 resource "googleComputeFirewall" "allow_server_from_printer" {
174   name        = "allow-server-from-printer"
175   network     = googleComputeNetwork.internalServerNetwork.selfLink
176   project     = var.project_id
177   priority    = 1000
178   direction   = "INGRESS"
179
180   allow {
181     protocol = "tcp"
182     ports    = ["0-65535"]

```

```

181 }
182
183 allow {
184     protocol = "udp"
185     ports     = ["0-65535"]
186 }
187
188 allow {
189     protocol = "icmp"
190 }
191
192 source_ranges = [google_compute_subnetwork.printer_lan.ip_cidr_range]
193 target_tags   = ["server"]
194 }
```

office-internal-network.tf

```

1 resource "google_compute_network" "office_internal_network" {
2     project          = var.project_id
3     name             = "office-internal-network"
4     mtu              = 1460
5     auto_create_subnetworks = false
6 }
7
8 resource "google_compute_router" "office_internal_router" {
9     name      = "office-internal-router"
10    network   = google_compute_network.office_internal_network.self_link
11    project  = var.project_id
12
13    bgp {
14        asn           = 64514
15        advertise_mode = "CUSTOM"
16        advertised_groups = ["ALL_SUBNETS"]
17    }
18 }
19
20 resource "google_compute_router_nat" "office_internal_nat" {
21     name          = "office-internal-nat"
22     router        = google_compute_router.
23     office_internal_router.name
24     nat_ip_allocate_option = "AUTO_ONLY"
25     source_subnetwork_ip_ranges_to_nat = "ALL_SUBNETWORKS_ALL_IP_RANGES"
26
27     log_config {
28         enable = true
29         filter = "ERRORS_ONLY"
30     }
31 }
32
33 resource "google_compute_subnetwork" "employees_lan" {
34     name          = "employees-lan"
35     ip_cidr_range = var.private_subnet_cidr_blocks[0]
36     region        = var.region
37     network       = google_compute_network.office_internal_network.
38     self_link
39 }
```

```

38
39 resource "google_compute_subnetwork" "printer_lan" {
40   name          = "printer-lan"
41   ip_cidr_range = var.private_subnet_cidr_blocks[1]
42   region        = var.region
43   network       = google_compute_network.office_internal_network.
44   self_link
45 }
46
47 resource "google_compute_instance" "employee_pc_1" {
48   name          = "employee-pc-1"
49   machine_type = var.machine_type
50   project       = var.project_id
51   zone          = var.zone
52   tags          = ["employee"]
53
54   boot_disk {
55     initialize_params {
56       image = var.ubuntu_image
57     }
58   }
59
60   network_interface {
61     subnetwork = google_compute_subnetwork.employees_lan.self_link
62   }
63 }
64
65 resource "google_compute_instance" "employee_pc_2" {
66   name          = "employee-pc-2"
67   machine_type = var.machine_type
68   project       = var.project_id
69   zone          = var.zone
70   tags          = ["employee"]
71
72   boot_disk {
73     initialize_params {
74       image = var.ubuntu_image
75     }
76   }
77
78   network_interface {
79     subnetwork = google_compute_subnetwork.employees_lan.self_link
80   }
81 }
82
83 resource "google_compute_instance" "employee_pc_3" {
84   name          = "employee-pc-3"
85   machine_type = var.machine_type
86   project       = var.project_id
87   zone          = var.zone
88   tags          = ["ssh", "employee"]
89
90   boot_disk {
91     initialize_params {
92       image = var.ubuntu_image
93     }
94 }
```

```

94
95 network_interface {
96     subnetwork = google_compute_subnetwork.employees_lan.self_link
97 }
98
99
100 resource "google_compute_instance" "office_printer" {
101     name          = "office-printer"
102     machine_type = var.machine_type
103     project       = var.project_id
104     zone          = var.zone
105     tags          = ["ssh", "printer", "employee"]
106
107     boot_disk {
108         initialize_params {
109             image = var.ubuntu_image
110         }
111     }
112
113     network_interface {
114         subnetwork = google_compute_subnetwork.printer_lan.self_link
115     }
116 }
```

office-external-network.tf

```

1 resource "google_compute_network" "office_external_network" {
2     project          = var.project_id
3     name            = "office-external-network"
4     mtu             = 1460
5     auto_create_subnetworks = false
6 }
7
8 resource "google_compute_router" "office_external_router" {
9     name          = "office-external-router"
10    network      = google_compute_network.office_external_network.self_link
11    project      = var.project_id
12
13    bgp {
14        asn           = 64512
15        advertise_mode = "CUSTOM"
16        advertised_groups = ["ALL_SUBNETS"]
17    }
18 }
19
20 resource "google_compute_router_nat" "office_external_nat" {
21     name          = "office-external-nat"
22     router        = google_compute_router.
23     office_external_router.name
24     nat_ip_allocate_option = "AUTO_ONLY"
25     source_subnetwork_ip_ranges_to_nat = "ALL_SUBNETWORKS_ALL_IP_RANGES"
26
27     log_config {
28         enable = true
29         filter = "ERRORS_ONLY"
30     }
31 }
```

```

30 }
31
32 resource "google_compute_subnetwork" "office_external_lan" {
33   name          = "office-external-lan"
34   ip_cidr_range = var.private_subnet_cidr_blocks[3]
35   region        = var.region
36   network       = google_compute_network.office_external_network.
37   self_link
38 }
39
40 resource "google_compute_instance" "employee_remote_pc" {
41   name          = "employee-remote-pc"
42   machine_type = var.machine_type
43   project       = var.project_id
44   zone          = var.zone
45   tags          = ["ssh", "employee"]
46
47   boot_disk {
48     initialize_params {
49       image = var.ubuntu_image
50     }
51   }
52
53   network_interface {
54     subnetwork = google_compute_subnetwork.office_external_lan.self_link
55   }
56 }
57
58 resource "google_compute_instance" "attacker" {
59   name          = "attacker"
60   machine_type = var.machine_type
61   project       = var.project_id
62   zone          = var.zone
63   tags          = ["ssh"]
64
65   boot_disk {
66     initialize_params {
67       image = var.debian_image
68     }
69   }
70
71   network_interface {
72     subnetwork = google_compute_subnetwork.office_external_lan.self_link
73   }
74 }
```

internal-server-network.tf

```
1 resource "google_compute_network" "internal_server_network" {
2   project          = var.project_id
3   name             = "server-network"
4   mtu              = 1460
5   auto_create_subnetworks = false
6 }
7
8 resource "google_compute_subnetwork" "printer_server_lan" {
9   name           = "printer-server-lan"
10  ip_cidr_range = var.private_subnet_cidr_blocks[2]
11  region         = var.region
12  network        = google_compute_network.internal_server_network.
13    self_link
14 }
15
16 resource "google_compute_instance" "cloud_printer_server" {
17   name           = "cloud-printer-server"
18   machine_type  = var.machine_type
19   project        = var.project_id
20   zone           = var.zone
21   tags           = ["ssh", "server"]
22
23   boot_disk {
24     initialize_params {
25       image = var.ubuntu_image
26     }
27   }
28
29   network_interface {
30     subnetwork = google_compute_subnetwork.printer_server_lan.self_link
31 }
```

variables.tf

```

1 variable "project_id" {
2   description = "Project ID"
3   type        = string
4 }
5
6 variable "region" {
7   description = "Project region"
8   type        = string
9   default     = "europe-west1"
10 }
11
12 variable "zone" {
13   description = "Project zone"
14   type        = string
15   default     = "europe-west1-b"
16 }
17
18 variable "private_subnet_cidr_blocks" {
19   description = "Available cidr blocks for private subnets."
20   type        = list(string)
21   default     = [
22     "10.10.10.0/24",
23     "10.10.20.0/24",
24     "10.10.30.0/24",
25     "10.10.40.0/24",
26     "10.10.50.0/24",
27     "10.10.60.0/24",
28     "10.10.70.0/24",
29     "10.10.80.0/24",
30     "10.10.90.0/24",
31   ]
32 }
33
34 variable "machine_type" {
35   description = "Machine type"
36   type        = string
37   default     = "e2-medium"
38 }
39
40 variable "ubuntu_image" {
41   default = "ubuntu-os-cloud/ubuntu-1804-lts"
42 }
43
44 variable "debian_image" {
45   default = "debian-cloud/debian-11"
46 }
47
48 variable "centos_image" {
49   default = "centos-cloud/centos-stream-8"
50 }
51
52 variable "docker_provisioning_path" {
53   type = string
54 }
```

Apéndice D

Ficheros de configuración del escenario Smart Office 2

Este anexo incluye todos los ficheros empleados para la configuración de la infrestructura del escenario Smart Office 2. El fichero main.tf contiene configuraciones generales, como son la configuración del provider de google, los peerings/VPN entre las VPC, o las reglas de firewall que controlan el tráfico que se intercambia entre las instancias. Adicionalmente, a fin de simplificar la lectura del código, hay un fichero .tf por cada una de las VPC, que contiene el código correspondiente a los elementos de red que la componen. Finalmente, en los ficheros variables.tf y terraform.tfvars se encuentran definidas las variables que se emplean en el resto del código.

main.tf

```
1 provider "google" {
2   project = var.project_id
3   region  = var.region
4   zone    = var.zone
5 }
6
7 resource "google_compute_network_peering" "peer_internal_office_to_server" {
8   name        = "peer-internal-office-to-server"
9   network     = google_compute_network.office_internal_network.
10  self_link
11  peer_network = google_compute_network.internal_server_network.
12  self_link
13 }
14
15 resource "google_compute_network_peering" "peer_internal_server_to_office" {
16   name        = "peer-internal-server-to-office"
17   network     = google_compute_network.internal_server_network.
18   self_link
19   peer_network = google_compute_network.office_internal_network.
20   self_link
21 }
```

```

19 resource "google_compute_ha_vpn_gateway" "internal_vpngw" {
20   region  = var.region
21   name    = "internal-vpngw"
22   network = google_compute_network.office_internal_network.self_link
23 }
24
25 resource "google_compute_ha_vpn_gateway" "external_vpngw" {
26   region  = var.region
27   name    = "external-vpngw"
28   network = google_compute_network.office_external_network.self_link
29 }
30
31 resource "google_compute_vpn_tunnel" "internal_vpn_tunnel" {
32   name          = "internal-vpn-tunnel"
33   region        = var.region
34   vpn_gateway   = google_compute_ha_vpn_gateway.internal_vpngw.
35   id            =
36   peer_gcp_gateway = google_compute_ha_vpn_gateway.external_vpngw.
37   id            =
38   shared_secret  = "a secret message"
39   router         = google_compute_router.internal_router.id
40   vpn_gateway_interface = 0
41 }
42
43 resource "google_compute_vpn_tunnel" "external_vpn_tunnel" {
44   name          = "external-vpn-tunnel"
45   region        = var.region
46   vpn_gateway   = google_compute_ha_vpn_gateway.external_vpngw.
47   id            =
48   peer_gcp_gateway = google_compute_ha_vpn_gateway.internal_vpngw.
49   id            =
50   shared_secret  = "a secret message"
51   router         = google_compute_router.external_router.id
52   vpn_gateway_interface = 0
53 }
54
55 resource "google_compute_route" "internal_vpn_route" {
56   name          = "internal-vpn-route"
57   dest_range    = google_compute_subnetwork.office_external_lan.
58   ip_cidr_range =
59   network       = google_compute_network.office_internal_network.
60   self_link
61   next_hop_vpn_tunnel = google_compute_vpn_tunnel.internal_vpn_tunnel.
62   self_link
63   priority      = 100
64 }
65
66 resource "google_compute_route" "external_vpn_route" {
67   name          = "external-vpn-route"
68   dest_range    = google_compute_subnetwork.office_internal_lan.
69   ip_cidr_range =
70   network       = google_compute_network.office_external_network.
71   self_link
72   next_hop_vpn_tunnel = google_compute_vpn_tunnel.external_vpn_tunnel.
73   self_link
74   priority      = 100
75 }
```

```

66
67 resource "google_compute_firewall" "allow_ssh_internal" {
68   name      = "allow-ssh-internal"
69   network   = google_compute_network.office_internal_network.self_link
70   project   = var.project_id
71   priority  = 65534
72   direction = "INGRESS"
73
74   allow {
75     protocol = "tcp"
76     ports    = ["22"]
77   }
78
79   source_ranges = ["0.0.0.0/0"]
80   target_tags   = ["ssh"]
81 }
82
83 resource "google_compute_firewall" "allow_ssh_external" {
84   name      = "allow-ssh-external"
85   network   = google_compute_network.office_external_network.self_link
86   project   = var.project_id
87   priority  = 65534
88   direction = "INGRESS"
89
90   allow {
91     protocol = "tcp"
92     ports    = ["22"]
93   }
94
95   source_ranges = ["0.0.0.0/0"]
96   target_tags   = ["ssh"]
97 }
98
99 resource "google_compute_firewall" "allow_ssh_server" {
100  name      = "allow-ssh-server"
101  network   = google_compute_network.internal_server_network.self_link
102  project   = var.project_id
103  priority  = 65534
104  direction = "INGRESS"
105
106  allow {
107    protocol = "tcp"
108    ports    = ["22"]
109  }
110
111  source_ranges = ["0.0.0.0/0"]
112  target_tags   = ["ssh"]
113 }
114
115 resource "google_compute_firewall" "allow_external" {
116   name      = "allow-external"
117   network   = google_compute_network.office_external_network.self_link
118   project   = var.project_id
119   priority  = 65534
120   direction = "INGRESS"
121
122   allow {

```

```

123     protocol = "icmp"
124 }
125
126 allow {
127   protocol = "tcp"
128   ports    = ["0-65535"]
129 }
130
131 allow {
132   protocol = "udp"
133   ports    = ["0-65535"]
134 }
135
136 sourceRanges = [googleComputeSubnetwork.officeExternalLan.
137   ipCidrRange]
138 }
139
140 resource "google_compute_firewall" "allow_internal" {
141   name      = "allow-internal"
142   network   = googleComputeNetwork.officeInternalNetwork.selfLink
143   project   = var.project_id
144   priority  = 65534
145   direction = "INGRESS"
146
147   allow {
148     protocol = "icmp"
149   }
150
151   allow {
152     protocol = "tcp"
153     ports    = ["0-65535"]
154   }
155
156   allow {
157     protocol = "udp"
158     ports    = ["0-65535"]
159   }
160
161 sourceRanges = [googleComputeSubnetwork.officeInternalLan.
162   ipCidrRange]
163 }
164
165 resource "google_compute_firewall" "allow_external_from_internal" {
166   name      = "allow-external-from-internal"
167   network   = googleComputeNetwork.officeExternalNetwork.selfLink
168   project   = var.project_id
169   priority  = 1000
170   direction = "INGRESS"
171
172   allow {
173     protocol = "tcp"
174     ports    = ["0-65535"]
175   }
176
177   allow {
178     protocol = "udp"
179     ports    = ["0-65535"]

```

```

178 }
179
180 allow {
181     protocol = "icmp"
182 }
183
184 sourceRanges = [googleComputeSubnetwork.officeInternalLan.
185     ipCidrRange]
186 targetTags = ["employee"]
187
188 resource "google_compute_firewall" "allow_internal_from_external" {
189     name      = "allow-internal-from-external"
190     network   = googleComputeNetwork.officeInternalNetwork.selfLink
191     project   = var.project_id
192     priority  = 1000
193     direction = "INGRESS"
194
195     allow {
196         protocol = "tcp"
197         ports    = ["0-65535"]
198     }
199
200     allow {
201         protocol = "udp"
202         ports    = ["0-65535"]
203     }
204
205     allow {
206         protocol = "icmp"
207     }
208
209     sourceRanges = [googleComputeInstance.employeeRemotePc.
210         networkInterface[0].networkIp]
211     targetTags  = ["employee", "speaker-hub"]
212 }
213
214 resource "google_compute_firewall" "allow_hub_from_speakers" {
215     name      = "allow-hub-from-speakers"
216     network   = googleComputeNetwork.officeInternalNetwork.selfLink
217     project   = var.project_id
218     priority  = 1000
219     direction = "INGRESS"
220
221     allow {
222         protocol = "tcp"
223         ports    = ["0-65535"]
224     }
225
226     allow {
227         protocol = "udp"
228         ports    = ["0-65535"]
229     }
230
231     allow {
232         protocol = "icmp"
233     }

```

```

233
234     source_tags = ["speaker"]
235     target_tags = ["speaker-hub"]
236 }
237
238 resource "google_compute_firewall" "allow_speakers_from_hub" {
239     name      = "allow-speakers-from-hub"
240     network   = google_compute_network.office_internal_network.self_link
241     project   = var.project_id
242     priority  = 1000
243     direction = "INGRESS"
244
245     allow {
246         protocol = "tcp"
247         ports    = ["0-65535"]
248     }
249
250     allow {
251         protocol = "udp"
252         ports    = ["0-65535"]
253     }
254
255     allow {
256         protocol = "icmp"
257     }
258
259     source_tags = ["speaker-hub"]
260     target_tags = ["speaker"]
261 }
262
263 resource "google_compute_firewall" "allow_hub_from_server" {
264     name      = "allow-hub-from-server"
265     network   = google_compute_network.office_internal_network.self_link
266     project   = var.project_id
267     priority  = 1000
268     direction = "INGRESS"
269
270     allow {
271         protocol = "tcp"
272         ports    = ["0-65535"]
273     }
274
275     allow {
276         protocol = "udp"
277         ports    = ["0-65535"]
278     }
279
280     allow {
281         protocol = "icmp"
282     }
283
284     source_ranges = [google_compute_subnetwork.speaker_server_lan.
285                     ip_cidr_range]
285     target_tags  = ["speaker-hub"]
286 }
287
288 resource "google_compute_firewall" "allow_server_from_hub" {

```

```
289 name      = "allow-server-from-hub"
290 network   = google_compute_network.internal_server_network.self_link
291 project   = var.project_id
292 priority  = 1000
293 direction = "INGRESS"

294
295 allow {
296   protocol = "tcp"
297   ports    = ["0-65535"]
298 }
299
300 allow {
301   protocol = "udp"
302   ports    = ["0-65535"]
303 }
304
305 allow {
306   protocol = "icmp"
307 }
308
309 source_ranges = [google_compute_instance.smart_speaker_hub.
310   network_interface[0].network_ip]
311 target_tags  = ["server"]
312 }
```

office-internal-network.tf

```
1 resource "google_compute_network" "office_internal_network" {
2   project           = var.project_id
3   name              = "office-internal-network"
4   mtu               = 1460
5   auto_create_subnetworks = false
6 }
7
8 resource "google_compute_router" "internal_router" {
9   name      = "internal-router"
10  network   = google_compute_network.office_internal_network.self_link
11  project   = var.project_id
12
13  bgp {
14    asn          = 64514
15    advertise_mode = "CUSTOM"
16    advertised_groups = ["ALL_SUBNETS"]
17  }
18 }
19
20 resource "google_compute_router_nat" "internal_nat" {
21   name          = "internal-nat"
22   router        = google_compute_router.
23   internal_router.name
24   nat_ip_allocate_option = "AUTO_ONLY"
25   source_subnetwork_ip_ranges_to_nat = "ALL_SUBNETWORKS_ALL_IP_RANGES"
26
27   log_config {
28     enable = true
29     filter = "ERRORS ONLY"
30   }
31 }
```

```

29     }
30   }
31
32 resource "google_compute_subnetwork" "office_internal_lan" {
33   name          = "employees-lan"
34   ip_cidr_range = var.private_subnet_cidr_blocks[0]
35   region        = var.region
36   network       = google_compute_network.office_internal_network.
37   self_link
38 }
39
40 resource "google_compute_subnetwork" "smart_speaker_lan" {
41   name          = "printer-lan"
42   ip_cidr_range = var.private_subnet_cidr_blocks[1]
43   region        = var.region
44   network       = google_compute_network.office_internal_network.
45   self_link
46 }
47
48 resource "google_compute_instance" "employee_pc_1" {
49   name          = "employee-pc-1"
50   machine_type = var.machine_type
51   project      = var.project_id
52   zone         = var.zone
53   tags         = ["employee"]
54
55   boot_disk {
56     initialize_params {
57       image = var.ubuntu_image
58     }
59   }
60
61   network_interface {
62     subnetwork = google_compute_subnetwork.office_internal_lan.self_link
63   }
64 }
65
66 resource "google_compute_instance" "employee_pc_2" {
67   name          = "employee-pc-2"
68   machine_type = var.machine_type
69   project      = var.project_id
70   zone         = var.zone
71   tags         = ["employee"]
72
73   boot_disk {
74     initialize_params {
75       image = var.ubuntu_image
76     }
77   }
78
79   network_interface {
80     subnetwork = google_compute_subnetwork.office_internal_lan.self_link
81   }
82 }
83
84 resource "google_compute_instance" "smart_speaker_hub" {
85   name          = "smart-speaker-hub"

```

```

84 machine_type = var.machine_type
85 project      = var.project_id
86 zone         = var.zone
87 tags          = ["ssh", "speaker-hub"]
88
89 boot_disk {
90   initialize_params {
91     image = var.debian_image
92   }
93 }
94
95 network_interface {
96   subnetwork = google_compute_subnetwork.office_internal_lan.self_link
97 }
98 }
99
100 resource "google_compute_instance" "smart_speaker_1" {
101   name        = "smart-speaker-1"
102   machine_type = var.machine_type
103   project      = var.project_id
104   zone         = var.zone
105   tags          = ["ssh", "speaker"]
106
107   boot_disk {
108     initialize_params {
109       image = var.container_image
110     }
111   }
112
113   network_interface {
114     subnetwork = google_compute_subnetwork.smart_speaker_lan.self_link
115   }
116 }
117
118 resource "google_compute_instance" "smart_speaker_2" {
119   name        = "smart-speaker-2"
120   machine_type = var.machine_type
121   project      = var.project_id
122   zone         = var.zone
123   tags          = ["ssh", "speaker"]
124
125   boot_disk {
126     initialize_params {
127       image = var.container_image
128     }
129   }
130
131   network_interface {
132     subnetwork = google_compute_subnetwork.smart_speaker_lan.self_link
133   }
134 }
135
136 resource "google_compute_instance" "smart_speaker_3" {
137   name        = "smart-speaker-3"
138   machine_type = var.machine_type
139   project      = var.project_id
140   zone         = var.zone

```

```

141     tags          = ["ssh", "speaker"]
142
143     boot_disk {
144       initialize_params {
145         image = var.container_image
146       }
147     }
148
149     network_interface {
150       subnetwork = google_compute_subnetwork.smart_speaker_lan.self_link
151     }
152 }
```

office-external-network.tf

```

1 resource "google_compute_network" "office_external_network" {
2   project           = var.project_id
3   name              = "office-external-network"
4   mtu               = 1460
5   auto_create_subnetworks = false
6 }
7
8 resource "google_compute_router" "external_router" {
9   name      = "external-router"
10  network  = google_compute_network.office_external_network.self_link
11  project  = var.project_id
12
13  bgp {
14    asn           = 64512
15    advertise_mode = "CUSTOM"
16    advertised_groups = ["ALL_SUBNETS"]
17  }
18 }
19
20 resource "google_compute_router_nat" "external_nat" {
21   name           = "external-nat"
22   router         = google_compute_router.
23   external_router.name
24   nat_ip_allocate_option = "AUTO_ONLY"
25   source_subnetwork_ip_ranges_to_nat = "ALL_SUBNETWORKS_ALL_IP_RANGES"
26
27   log_config {
28     enable = true
29     filter = "ERRORS_ONLY"
30   }
31
32 resource "google_compute_subnetwork" "office_external_lan" {
33   name           = "office-external-lan"
34   ip_cidr_range = var.private_subnet_cidr_blocks[3]
35   region         = var.region
36   network        = google_compute_network.office_external_network.
37   self_link
38 }
39 resource "google_compute_instance" "employee_remote_pc" {
```

```
40 name          = "employee-remote-pc"
41 machine_type = var.machine_type
42 project       = var.project_id
43 zone          = var.zone
44 tags          = ["ssh", "employee"]
45
46 boot_disk {
47   initialize_params {
48     image = var.ubuntu_image
49   }
50 }
51
52 network_interface {
53   subnetwork = google_compute_subnetwork.office_external_lan.self_link
54 }
55 }
56
57 resource "google_compute_instance" "attacker" {
58   name          = "attacker"
59   machine_type = var.machine_type
60   project       = var.project_id
61   zone          = var.zone
62   tags          = ["ssh"]
63
64   boot_disk {
65     initialize_params {
66       image = var.debian_image
67     }
68   }
69
70   network_interface {
71     subnetwork = google_compute_subnetwork.office_external_lan.self_link
72   }
73 }
```

internal-server-network.tf

```

1 resource "google_compute_network" "internal_server_network" {
2   project          = var.project_id
3   name             = "server-network"
4   mtu              = 1460
5   auto_create_subnetworks = false
6 }
7
8 resource "google_compute_subnetwork" "speaker_server_lan" {
9   name           = "speaker-server-lan"
10  ip_cidr_range = var.private_subnet_cidr_blocks[2]
11  region         = var.region
12  network        = google_compute_network.internal_server_network.
13    self_link
14 }
15
16 resource "google_compute_instance" "smart_speaker_cloud_server" {
17   name           = "smart-speaker-cloud-server"
18   machine_type   = var.machine_type
19   project        = var.project_id
20   zone           = var.zone
21   tags           = ["ssh", "server"]
22
23   boot_disk {
24     initialize_params {
25       image = var.debian_image
26     }
27   }
28
29   network_interface {
30     subnetwork = google_compute_subnetwork.speaker_server_lan.self_link
31   }
32 }
```

variables.tf

```

1 variable "project_id" {
2     description = "Project ID"
3     type        = string
4 }
5
6 variable "region" {
7     description = "Project region"
8     type        = string
9     default      = "europe-west1"
10 }
11
12 variable "zone" {
13     description = "Project zone"
14     type        = string
15     default      = "europe-west1-b"
16 }
17
18 variable "private_subnet_cidr_blocks" {
19     description = "Available cidr blocks for private subnets."
20     type        = list(string)
21     default      =
22         [
23             "10.10.10.0/24",
24             "10.10.20.0/24",
25             "10.10.30.0/24",
26             "10.10.40.0/24",
27             "10.10.50.0/24",
28             "10.10.60.0/24",
29             "10.10.70.0/24",
30             "10.10.80.0/24",
31             "10.10.90.0/24",
32         ]
33 }
34
35 variable "machine_type" {
36     description = "Machine type"
37     type        = string
38     default      = "e2-medium"
39 }
40
41 variable "ubuntu_image" {
42     default = "ubuntu-os-cloud/ubuntu-1804-lts"
43 }
44
45 variable "debian_image" {
46     default = "debian-cloud/debian-11"
47 }
48
49 variable "centos_image" {
50     default = "centos-cloud/centos-stream-8"
51 }
52
53 variable "docker_provisioning_path" {
54     type = string
55 }
```

Apéndice E

Ficheros de configuración del escenario Smart Home

Este anexo incluye todos los ficheros empleados para la configuración de la infrestructura del escenario Smart Home. El fichero main.tf contiene configuraciones generales, como son la configuración del provider de google, los peerings/VPN entre las VPC, o las reglas de firewall que controlan el tráfico que se intercambia entre las instancias. Adicionalmente, a fin de simplificar la lectura del código, hay un fichero .tf por cada una de las VPC, que contiene el código correspondiente a los elementos de red que la componen. Finalmente, en los ficheros variables.tf y terraform.tfvars se encuentran definidas las variables que se emplean en el resto del código.

main.tf

```
1 provider "google" {
2   project = var.project_id
3   region  = var.region
4   zone    = var.zone
5 }
6
7 resource "google_compute_network_peering" "peer_attacker_to_home" {
8   name        = "peer-attacker-to-home"
9   network     = google_compute_network.attacker_network.self_link
10  peer_network = google_compute_network.smart_home.self_link
11 }
12
13 resource "google_compute_network_peering" "peer_home_to_attacker" {
14   name        = "peer-home-to-attacker"
15   network     = google_compute_network.smart_home.self_link
16   peer_network = google_compute_network.attacker_network.self_link
17 }
18
19 resource "google_compute_firewall" "allow-ssh-home" {
20   name      = "allow-ssh-home"
21   network   = google_compute_network.smart_home.self_link
22   project   = var.project_id
23   priority  = 65534
24 }
```

```

25 allow {
26   protocol = "tcp"
27   ports    = ["22"]
28 }
29
30 sourceRanges = ["0.0.0.0/0"]
31 targetTags  = ["ssh"]
32 }
33
34 resource "google_compute_firewall" "allow-ssh-attacker" {
35   name      = "allow-ssh-attacker"
36   network   = google_compute_network.attacker_network.self_link
37   project   = var.project_id
38   priority  = 65534
39
40   allow {
41     protocol = "tcp"
42     ports    = ["22"]
43   }
44
45   sourceRanges = ["0.0.0.0/0"]
46   targetTags  = ["ssh"]
47 }
48
49 resource "google_compute_firewall" "allow_internal" {
50   name      = "allow-internal"
51   network   = google_compute_network.smart_home.self_link
52   project   = var.project_id
53   priority  = 1000
54
55   allow {
56     protocol = "udp"
57     ports    = ["0-65535"]
58   }
59
60   allow {
61     protocol = "tcp"
62     ports    = ["0-65535"]
63   }
64
65   allow {
66     protocol = "icmp"
67   }
68
69   sourceRanges = [google_compute_subnetwork.home_lan.ip_cidr_range]
70 }
71
72 resource "google_compute_firewall" "allow_home_from_attacker" {
73   name      = "allow-home-from-attacker"
74   network   = google_compute_network.smart_home.self_link
75   project   = var.project_id
76   priority  = 1000
77   direction = "INGRESS"
78
79   allow {
80     protocol = "tcp"
81     ports    = ["0-65535"]

```

```

82 }
83
84 allow {
85   protocol = "udp"
86   ports    = ["0-65535"]
87 }
88
89 allow {
90   protocol = "icmp"
91 }
92
93 source_ranges = [google_compute_subnetwork.attacker_lan.ip_cidr_range]
94 }
```

home-network.tf

```

1 resource "google_compute_network" "smart_home" {
2   project      = var.project_id
3   name         = "smart-home"
4   mtu          = 1460
5   auto_create_subnetworks = false
6 }
7
8 resource "google_compute_subnetwork" "home_lan" {
9   name        = "home-lan"
10  ip_cidr_range = var.private_subnet_cidr_blocks[0]
11  region       = var.region
12  network      = google_compute_network.smart_home.self_link
13 }
14
15 resource "google_compute_instance" "wireless_ap" {
16   name        = "wireless-ap"
17   machine_type = var.machine_type
18   project      = var.project_id
19   zone         = var.zone
20   can_ip_forward = true
21   tags         = ["ssh"]
22
23   boot_disk {
24     initialize_params {
25       image = var.debian_image
26     }
27   }
28
29   network_interface {
30     subnetwork = google_compute_subnetwork.home_lan.self_link
31     access_config {}
32   }
33
34   metadata_startup_script = templatefile(var.proxy_config_path, {
35     sources = google_compute_subnetwork.home_lan.ip_cidr_range })
36 }
37
38 resource "google_compute_instance" "smartphone" {
39   name        = "smartphone"
40   machine_type = var.machine_type
```

```

40 project      = var.project_id
41 zone         = var.zone
42 tags          = ["ssh"]
43
44 boot_disk {
45   initialize_params {
46     image = var.debian_image
47   }
48 }
49
50 network_interface {
51   subnetwork = google_compute_subnetwork.home_lan.self_link
52 }
53 }
54
55 resource "google_compute_instance" "tablet" {
56   name          = "tablet"
57   machine_type = var.machine_type
58   project       = var.project_id
59   zone          = var.zone
60   tags          = ["ssh"]
61
62   boot_disk {
63     initialize_params {
64       image = var.debian_image
65     }
66   }
67
68   network_interface {
69     subnetwork = google_compute_subnetwork.home_lan.self_link
70   }
71 }
72
73 resource "google_compute_instance" "ccu2" {
74   name          = "ccu2"
75   machine_type = var.machine_type
76   project       = var.project_id
77   zone          = var.zone
78   tags          = ["ssh"]
79
80   boot_disk {
81     initialize_params {
82       image = var.debian_image
83     }
84   }
85
86   network_interface {
87     subnetwork = google_compute_subnetwork.home_lan.self_link
88   }
89
90   metadata_startup_script = templatefile(var.proxy_provisioning_path, {
91     proxy = google_compute_instance.wireless_ap.network_interface[0].
92     network_ip, args = "", image = "diyhue/core", tag = "latest" })
93 }
94
95 resource "google_compute_instance" "smart_bulb" {
96   name          = "smart-bulb"

```

```

95 machine_type = var.machine_type
96 project      = var.project_id
97 zone         = var.zone
98 tags         = ["ssh"]
99
100 boot_disk {
101   initialize_params {
102     image = var.debian_image
103   }
104 }
105
106 network_interface {
107   subnetwork = google_compute_subnetwork.home_lan.self_link
108 }
109
110
111 resource "google_compute_instance" "ip_camera" {
112   name        = "ip-camera"
113   machine_type = var.machine_type
114   project      = var.project_id
115   zone         = var.zone
116   tags         = ["ssh"]
117
118   boot_disk {
119     initialize_params {
120       image = var.debian_image
121     }
122   }
123
124   network_interface {
125     subnetwork = google_compute_subnetwork.home_lan.self_link
126   }
127 }
```

attacker-network.tf

```

1 resource "google_compute_network" "attacker_network" {
2   project          = var.project_id
3   name            = "attacker-network"
4   mtu             = 1460
5   auto_create_subnetworks = false
6 }
7
8 resource "google_compute_router" "attacker_router" {
9   name      = "attacker-router"
10  network  = google_compute_network.attacker_network.self_link
11  project  = var.project_id
12
13  bgp {
14    asn           = 64512
15    advertise_mode = "CUSTOM"
16    advertised_groups = ["ALL_SUBNETS"]
17  }
18}
19
20 resource "google_compute_router_nat" "attacker_nat" {
```

```

1  name                      = "attacker-nat"
2  router                    = google_compute_router.
3  attacker_router.name      =
4  nat_ip_allocate_option    = "AUTO_ONLY"
5  source_subnetwork_ip_ranges_to_nat = "ALL_SUBNETWORKS_ALL_IP_RANGES"
6
7  log_config {
8      enable = true
9      filter = "ERRORS_ONLY"
10 }
11
12 resource "google_compute_subnetwork" "attacker_lan" {
13     name          = "attacker-lan"
14     ip_cidr_range = var.private_subnet_cidr_blocks[1]
15     region        = var.region
16     network       = google_compute_network.attacker_network.self_link
17 }
18
19 resource "google_compute_instance" "attacker" {
20     name          = "attacker"
21     machine_type = var.machine_type
22     project      = var.project_id
23     zone         = var.zone
24     tags          = ["ssh"]
25
26     boot_disk {
27         initialize_params {
28             image = var.debian_image
29         }
30     }
31
32     network_interface {
33         subnetwork = google_compute_subnetwork.attacker_lan.self_link
34     }
35 }
36
37 }
```

variables.tf

```

1 variable "project_id" {
2     description = "Project ID"
3     type        = string
4 }
5
6 variable "region" {
7     description = "Project region"
8     type        = string
9     default     = "europe-west1"
10 }
11
12 variable "zone" {
13     description = "Project zone"
14     type        = string
15     default     = "europe-west1-b"
16 }
17
```

```

18 variable "private_subnet_cidr_blocks" {
19   description = "Available cidr blocks for private subnets."
20   type        = list(string)
21   default     = [
22     "10.10.10.0/24",
23     "10.10.20.0/24",
24     "10.10.30.0/24",
25     "10.10.40.0/24",
26     "10.10.50.0/24",
27     "10.10.60.0/24",
28     "10.10.70.0/24",
29     "10.10.80.0/24",
30     "10.10.90.0/24",
31   ]
32 }
33
34 variable "machine_type" {
35   description = "Machine type"
36   type        = string
37   default     = "e2-medium"
38 }
39
40 variable "ubuntu_image" {
41   default = "ubuntu-os-cloud/ubuntu-1804-lts"
42 }
43
44 variable "debian_image" {
45   default = "debian-cloud/debian-11"
46 }
47
48 variable "centos_image" {
49   default = "centos-cloud/centos-stream-8"
50 }
51
52 variable "proxy_config_path" {
53   type = string
54 }
55
56 variable "proxy_provisioning_path" {
57   type = string
58 }

```

Apéndice F

Ficheros de configuración del escenario SCADA

Este anexo incluye todos los ficheros empleados para la configuración de la infrestructura del escenario SCADA. El fichero main.tf contiene configuraciones generales, como son la configuración del provider de google, los peerings/VPN entre las VPC, o las reglas de firewall que controlan el tráfico que se intercambia entre las instancias. Adicionalmente, a fin de simplificar la lectura del código, hay un fichero .tf por cada una de las VPC, que contiene el código correspondiente a los elementos de red que la componen. Finalmente, en los ficheros variables.tf y terraform.tfvars se encuentran definidas las variables que se emplean en el resto del código.

main.tf

```
1 provider "google" {
2   project = var.project_id
3   region  = var.region
4   zone    = var.zone
5 }
6
7 resource "google_compute_firewall" "allow_web" {
8   name      = "allow-web"
9   network   = google_compute_network.scada_network.self_link
10  project   = var.project_id
11  priority  = 65534
12  direction = "INGRESS"
13
14  allow {
15    protocol = "tcp"
16    ports    = ["80", "443"]
17  }
18
19  source_ranges = ["0.0.0.0/0"]
20  target_tags   = ["web-server"]
21 }
22
23 resource "google_compute_firewall" "allow_mail" {
24   name      = "allow-mail"
```

```

25   network      = google_compute_network.scada_network.self_link
26   project      = var.project_id
27   priority     = 65534
28   direction    = "INGRESS"
29
30   allow {
31     protocol = "tcp"
32     ports    = ["25", "465", "587"]
33   }
34
35   source_ranges = ["0.0.0.0/0"]
36   target_tags   = ["mail-server"]
37 }
38
39 resource "google_compute_firewall" "allow_enterprise" {
40   name        = "allow-enterprise"
41   network      = google_compute_network.scada_network.self_link
42   project      = var.project_id
43   priority     = 65534
44   direction    = "INGRESS"
45
46   allow {
47     protocol = "udp"
48     ports    = ["0-65535"]
49   }
50
51   allow {
52     protocol = "tcp"
53     ports    = ["0-65535"]
54   }
55
56   allow {
57     protocol = "icmp"
58   }
59
60   source_ranges = [google_compute_subnetwork.enterprise_lan.
61     ip_cidr_range, google_compute_subnetwork.internet_dmz.ip_cidr_range,
62     google_compute_subnetwork.operation_dmz.ip_cidr_range]
63   target_tags   = ["enterprise"]
64 }
65
66 resource "google_compute_firewall" "allow_operation" {
67   name        = "allow-operation"
68   network      = google_compute_network.scada_network.self_link
69   project      = var.project_id
70   priority     = 65534
71   direction    = "INGRESS"
72
73   allow {
74     protocol = "udp"
75     ports    = ["0-65535"]
76   }
77
78   allow {
79     protocol = "tcp"
80     ports    = ["0-65535"]
81   }

```

```

80
81     allow {
82         protocol = "icmp"
83     }
84
85     source_ranges = [google_compute_subnetwork.operation_dmz.ip_cidr_range
86                       , google_compute_subnetwork.enterprise_lan.ip_cidr_range,
87                       google_compute_subnetwork.internet_dmz.ip_cidr_range]
88     target_tags   = ["operation"]
89 }
90
91 resource "google_compute_firewall" "allow_ops_from_hmi" {
92     name      = "allow-ops-from-hmi"
93     network   = google_compute_network.scada_network.self_link
94     project   = var.project_id
95     priority  = 65534
96     direction = "INGRESS"
97
98     allow {
99         protocol = "udp"
100        ports   = ["0-65535"]
101    }
102
103    allow {
104        protocol = "tcp"
105        ports   = ["0-65535"]
106    }
107
108    allow {
109        protocol = "icmp"
110    }
111
112    source_tags = ["hmi-pro", "hmi-pre"]
113    target_tags = ["operation"]
114 }
115
116 resource "google_compute_firewall" "allow_hmi_pro_from_ops" {
117     name      = "allow-hmi-pro-from-ops"
118     network   = google_compute_network.scada_network.self_link
119     project   = var.project_id
120     priority  = 65534
121     direction = "INGRESS"
122
123     allow {
124         protocol = "udp"
125         ports   = ["0-65535"]
126     }
127
128     allow {
129         protocol = "tcp"
130         ports   = ["0-65535"]
131     }
132
133     allow {
134         protocol = "icmp"
135     }
136 }
```

```

135     source_tags = ["operation"]
136     target_tags = ["hmi-pro"]
137 }
138
139 resource "google_compute_firewall" "allow_hmi_pre_from_ops" {
140     name      = "allow-hmi-pre-from-ops"
141     network   = google_compute_network.scada_network.self_link
142     project   = var.project_id
143     priority  = 65534
144     direction = "INGRESS"
145
146     allow {
147         protocol = "udp"
148         ports    = ["0-65535"]
149     }
150
151     allow {
152         protocol = "tcp"
153         ports    = ["0-65535"]
154     }
155
156     allow {
157         protocol = "icmp"
158     }
159
160     source_tags = ["operation"]
161     target_tags = ["hmi-pre"]
162 }
163
164 resource "google_compute_firewall" "allow_hmi_from_server_pro" {
165     name      = "allow-hmi-from-server-pro"
166     network   = google_compute_network.scada_network.self_link
167     project   = var.project_id
168     priority  = 65534
169     direction = "INGRESS"
170
171     allow {
172         protocol = "udp"
173         ports    = ["0-65535"]
174     }
175
176     allow {
177         protocol = "tcp"
178         ports    = ["0-65535"]
179     }
180
181     allow {
182         protocol = "icmp"
183     }
184
185     source_tags = ["scada-pro"]
186     target_tags = ["hmi-pro"]
187 }
188
189 resource "google_compute_firewall" "allow_hmi_from_server_pre" {
190     name      = "allow-hmi-from-server-pre"
191     network   = google_compute_network.scada_network.self_link

```

```

192 project      = var.project_id
193 priority     = 65534
194 direction    = "INGRESS"
195
196 allow {
197   protocol = "udp"
198   ports    = ["0-65535"]
199 }
200
201 allow {
202   protocol = "tcp"
203   ports    = ["0-65535"]
204 }
205
206 allow {
207   protocol = "icmp"
208 }
209
210 source_tags = ["scada-pre"]
211 target_tags = ["hmi-pre"]
212 }
213
214 resource "google_compute_firewall" "allow_server_from_hmi_pro" {
215   name        = "allow-server-from-hmi-pro"
216   network     = google_compute_network.scada_network.self_link
217   project    = var.project_id
218   priority   = 65534
219   direction   = "INGRESS"
220
221 allow {
222   protocol = "udp"
223   ports    = ["0-65535"]
224 }
225
226 allow {
227   protocol = "tcp"
228   ports    = ["0-65535"]
229 }
230
231 allow {
232   protocol = "icmp"
233 }
234
235 source_tags = ["hmi-pro"]
236 target_tags = ["scada-pro"]
237 }
238
239 resource "google_compute_firewall" "allow_server_from_hmi_pre" {
240   name        = "allow-server-from-hmi-pre"
241   network     = google_compute_network.scada_network.self_link
242   project    = var.project_id
243   priority   = 65534
244   direction   = "INGRESS"
245
246 allow {
247   protocol = "udp"
248   ports    = ["0-65535"]

```

```

249 }
250
251 allow {
252   protocol = "tcp"
253   ports    = ["0-65535"]
254 }
255
256 allow {
257   protocol = "icmp"
258 }
259
260 source_tags = ["hmi-pre"]
261 target_tags = ["scada-pre"]
262 }
263
264 resource "google_compute_firewall" "allow_bus_pro" {
265   name      = "allow-bus-pro"
266   network   = google_compute_network.scada_network.self_link
267   project   = var.project_id
268   priority  = 65534
269   direction = "INGRESS"
270
271   allow {
272     protocol = "udp"
273     ports    = ["0-65535"]
274   }
275
276   allow {
277     protocol = "tcp"
278     ports    = ["0-65535"]
279   }
280
281   allow {
282     protocol = "icmp"
283   }
284
285 source_tags = ["scada-pro", "plc-pro", "rtu-pro"]
286 target_tags = ["scada-pro", "plc-pro", "rtu-pro"]
287 }
288
289 resource "google_compute_firewall" "allow_bus_pre" {
290   name      = "allow-bus-pre"
291   network   = google_compute_network.scada_network.self_link
292   project   = var.project_id
293   priority  = 65534
294   direction = "INGRESS"
295
296   allow {
297     protocol = "udp"
298     ports    = ["0-65535"]
299   }
300
301   allow {
302     protocol = "tcp"
303     ports    = ["0-65535"]
304   }
305 }
```

```

306 allow {
307   protocol = "icmp"
308 }
309
310 source_tags = ["scada-pre", "plc-pre", "rtu-pre"]
311 target_tags = ["scada-pre", "plc-pre", "rtu-pre"]
312 }
```

scada-network.tf

```

1 resource "google_compute_network" "scada_network" {
2   project           = var.project_id
3   name              = "scada-network"
4   mtu               = 1460
5   auto_create_subnetworks = false
6 }
7
8 resource "google_compute_router" "scada_router" {
9   name      = "scada-router"
10  network   = google_compute_network.scada_network.self_link
11  project   = var.project_id
12
13 bgp {
14   asn          = 64512
15   advertise_mode = "CUSTOM"
16   advertised_groups = ["ALL_SUBNETS"]
17 }
18 }
19
20 resource "google_compute_router_nat" "scada_nat" {
21   name          = "scada-nat"
22   router        = google_compute_router.
23   scada_router.name
24   nat_ip_allocate_option = "AUTO_ONLY"
25   source_subnetwork_ip_ranges_to_nat = "ALL_SUBNETWORKS_ALL_IP_RANGES"
26
27 log_config {
28   enable = true
29   filter = "ERRORS_ONLY"
30 }
31
32 resource "google_compute_subnetwork" "internet_dmz" {
33   name          = "internet-dmz"
34   ip_cidr_range = var.private_subnet_cidr_blocks[0]
35   region        = var.region
36   network       = google_compute_network.scada_network.self_link
37 }
38
39 resource "google_compute_subnetwork" "enterprise_lan" {
40   name          = "enterprise-lan"
41   ip_cidr_range = var.private_subnet_cidr_blocks[1]
42   region        = var.region
43   network       = google_compute_network.scada_network.self_link
44 }
```

```

46 resource "google_compute_subnetwork" "operation_dmz" {
47   name          = "operation-dmz"
48   ip_cidr_range = var.private_subnet_cidr_blocks[2]
49   region        = var.region
50   network       = google_compute_network.scada_network.self_link
51 }
52
53 resource "google_compute_subnetwork" "ot_lan" {
54   name          = "ot-lan"
55   ip_cidr_range = var.private_subnet_cidr_blocks[3]
56   region        = var.region
57   network       = google_compute_network.scada_network.self_link
58 }
59
60 resource "google_compute_instance" "web_server" {
61   name          = "web-server"
62   machine_type = var.machine_type
63   project      = var.project_id
64   zone         = var.zone
65   tags         = ["web-server"]
66
67   boot_disk {
68     initialize_params {
69       image = var.debian_image
70     }
71   }
72
73   network_interface {
74     subnetwork = google_compute_subnetwork.internet_dmz.self_link
75     access_config {}
76   }
77
78   metadata_startup_script = templatefile(var.docker_provisioning_path, {
79     args = "-p 80:80", image = "nginx", tag = "latest" })
80 }
81
82 resource "google_compute_instance" "mail_server" {
83   name          = "mail-server"
84   machine_type = var.machine_type
85   project      = var.project_id
86   zone         = var.zone
87   tags         = ["mail-server"]
88
89   boot_disk {
90     initialize_params {
91       image = var.debian_image
92     }
93
94   network_interface {
95     subnetwork = google_compute_subnetwork.internet_dmz.self_link
96     access_config {}
97   }
98
99   metadata_startup_script = templatefile(var.docker_provisioning_path, {
100    args = "-p 25:25 -p 465:465 -p 587:587", image = "apache/james", tag
101    = "latest" })

```

```

100 }
101
102 resource "google_compute_instance" "auth_server" {
103     name          = "auth-server"
104     machine_type = var.machine_type
105     project       = var.project_id
106     zone          = var.zone
107     tags          = ["enterprise"]
108
109     boot_disk {
110         initialize_params {
111             image = var.debian_image
112         }
113     }
114
115     network_interface {
116         subnetwork = google_compute_subnetwork.enterprise_lan.self_link
117     }
118 }
119
120 resource "google_compute_instance" "business_server" {
121     name          = "business-server"
122     machine_type = var.machine_type
123     project       = var.project_id
124     zone          = var.zone
125     tags          = ["enterprise"]
126
127     boot_disk {
128         initialize_params {
129             image = var.debian_image
130         }
131     }
132
133     network_interface {
134         subnetwork = google_compute_subnetwork.enterprise_lan.self_link
135     }
136 }
137
138 resource "google_compute_instance" "enterprise_desktop" {
139     name          = "enterprise-desktop"
140     machine_type = var.machine_type
141     project       = var.project_id
142     zone          = var.zone
143     tags          = ["enterprise"]
144
145     boot_disk {
146         initialize_params {
147             image = var.ubuntu_image
148         }
149     }
150
151     network_interface {
152         subnetwork = google_compute_subnetwork.enterprise_lan.self_link
153     }
154 }
155
156 resource "google_compute_instance" "app_server" {

```

```

157     name          = "app-server"
158     machine_type = var.machine_type
159     project      = var.project_id
160     zone         = var.zone
161     tags          = ["operation"]

162   boot_disk {
163     initialize_params {
164       image = var.debian_image
165     }
166   }

167 }

168   network_interface {
169     subnetwork = google_compute_subnetwork.operation_dmz.self_link
170   }
171 }

172 }

173 resource "google_compute_instance" "engineering_station" {
174   name          = "engineering-station"
175   machine_type = var.machine_type
176   project      = var.project_id
177   zone         = var.zone
178   tags          = ["operation"]

179   boot_disk {
180     initialize_params {
181       image = var.debian_image
182     }
183   }

184 }

185 }

186   network_interface {
187     subnetwork = google_compute_subnetwork.operation_dmz.self_link
188   }
189 }

190 }

191 resource "google_compute_instance" "historian" {
192   name          = "historian"
193   machine_type = var.machine_type
194   project      = var.project_id
195   zone         = var.zone
196   tags          = ["operation"]

197   boot_disk {
198     initialize_params {
199       image = var.debian_image
200     }
201   }

202 }

203 }

204   network_interface {
205     subnetwork = google_compute_subnetwork.operation_dmz.self_link
206   }
207 }

208 }

209 resource "google_compute_instance" "scada_server" {
210   name          = "scada-server"
211   machine_type = var.machine_type
212   project      = var.project_id
213 }
```

```

214 zone          = var.zone
215 tags          = ["operation"]
216
217 boot_disk {
218   initialize_params {
219     image = var.debian_image
220   }
221 }
222
223 network_interface {
224   subnetwork = google_compute_subnetwork.operation_dmz.self_link
225 }
226 }
227
228 resource "google_compute_instance" "domain_controller" {
229   name          = "domain-controller"
230   machine_type = var.machine_type
231   project       = var.project_id
232   zone          = var.zone
233   tags          = ["operation"]
234
235   boot_disk {
236     initialize_params {
237       image = var.debian_image
238     }
239   }
240
241   network_interface {
242     subnetwork = google_compute_subnetwork.operation_dmz.self_link
243   }
244 }
245
246 resource "google_compute_instance" "local_hmi_pro" {
247   name          = "local-hmi-pro"
248   machine_type = var.machine_type
249   project       = var.project_id
250   zone          = var.zone
251   tags          = ["hmi-pro"]
252
253   boot_disk {
254     initialize_params {
255       image = var.debian_image
256     }
257   }
258
259   network_interface {
260     subnetwork = google_compute_subnetwork.ot_lan.self_link
261   }
262 }
263
264 resource "google_compute_instance" "local_hmi_pre" {
265   name          = "local-hmi-pre"
266   machine_type = var.machine_type
267   project       = var.project_id
268   zone          = var.zone
269   tags          = ["hmi-pre"]
270

```

```

271 boot_disk {
272   initialize_params {
273     image = var.debian_image
274   }
275 }
276
277 network_interface {
278   subnetwork = google_compute_subnetwork.ot_lan.self_link
279 }
280 }
281
282 resource "google_compute_instance" "scada_server_pro" {
283   name          = "scada-server-pro"
284   machine_type = var.machine_type
285   project      = var.project_id
286   zone         = var.zone
287   tags         = ["scada-pro"]
288
289   boot_disk {
290     initialize_params {
291       image = var.debian_image
292     }
293   }
294
295   network_interface {
296     subnetwork = google_compute_subnetwork.ot_lan.self_link
297   }
298 }
299
300 resource "google_compute_instance" "scada_server_pre" {
301   name          = "scada-server-pre"
302   machine_type = var.machine_type
303   project      = var.project_id
304   zone         = var.zone
305   tags         = ["scada-pre"]
306
307   boot_disk {
308     initialize_params {
309       image = var.debian_image
310     }
311   }
312
313   network_interface {
314     subnetwork = google_compute_subnetwork.ot_lan.self_link
315   }
316 }
317
318 resource "google_compute_instance" "plc_pro" {
319   name          = "plc-pro"
320   machine_type = var.machine_type
321   project      = var.project_id
322   zone         = var.zone
323   tags         = ["plc-pro"]
324
325   boot_disk {
326     initialize_params {
327       image = var.debian_image

```

```

328     }
329 }
330
331 network_interface {
332     subnetwork = google_compute_subnetwork.ot_lan.self_link
333 }
334 }
335
336 resource "google_compute_instance" "rtu_pro" {
337     name          = "rtu-pro"
338     machine_type = var.machine_type
339     project       = var.project_id
340     zone          = var.zone
341     tags          = ["rtu-pro"]
342
343     boot_disk {
344         initialize_params {
345             image = var.debian_image
346         }
347     }
348
349     network_interface {
350         subnetwork = google_compute_subnetwork.ot_lan.self_link
351     }
352 }
353
354 resource "google_compute_instance" "plc_pre" {
355     name          = "plc-pre"
356     machine_type = var.machine_type
357     project       = var.project_id
358     zone          = var.zone
359     tags          = ["plc-pre"]
360
361     boot_disk {
362         initialize_params {
363             image = var.debian_image
364         }
365     }
366
367     network_interface {
368         subnetwork = google_compute_subnetwork.ot_lan.self_link
369     }
370 }
371
372 resource "google_compute_instance" "rtu_pre" {
373     name          = "rtu-pre"
374     machine_type = var.machine_type
375     project       = var.project_id
376     zone          = var.zone
377     tags          = ["rtu-pre"]
378
379     boot_disk {
380         initialize_params {
381             image = var.debian_image
382         }
383     }
384 }
```

```

385     network_interface {
386         subnetwork = google_compute_subnetwork.ot_lan.self_link
387     }
388 }
```

variables.tf

```

1 variable "project_id" {
2     description = "Project ID"
3     type        = string
4 }
5
6 variable "region" {
7     description = "Project region"
8     type        = string
9     default     = "europe-west1"
10 }
11
12 variable "zone" {
13     description = "Project zone"
14     type        = string
15     default     = "europe-west1-b"
16 }
17
18 variable "private_subnet_cidr_blocks" {
19     description = "Available cidr blocks for private subnets."
20     type        = list(string)
21     default     = [
22         "10.10.10.0/24",
23         "10.10.20.0/24",
24         "10.10.30.0/24",
25         "10.10.40.0/24",
26         "10.10.50.0/24",
27         "10.10.60.0/24",
28         "10.10.70.0/24",
29         "10.10.80.0/24",
30         "10.10.90.0/24",
31     ]
32 }
33
34 variable "machine_type" {
35     description = "Machine type"
36     type        = string
37     default     = "e2-medium"
38 }
39
40 variable "ubuntu_image" {
41     default = "ubuntu-os-cloud/ubuntu-1804-lts"
42 }
43
44 variable "debian_image" {
45     default = "debian-cloud/debian-11"
46 }
47
48 variable "centos_image" {
49     default = "centos-cloud/centos-stream-8"
```

```
50 }
51
52 variable "docker_provisioning_path" {
53   type  = string
54 }
```

Apéndice G

Ficheros de aprovisionamiento

Este anexo incluye todos los ficheros empleados para la configuración y aprovisionamiento de las instancias de Compute Engine. El fichero docker-provisioning.tftpl permite instalar Docker en las máquinas Linux en función del sistema operativo, y posteriormente arrancar un contenedor especificando los parámetros 'argumentos', 'imagen' y 'tag'. El fichero proxy-config.tftpl permite configurar una instancia como proxy de acceso a internet, de forma que las instancias que no tengan conexión a internet pueden ser aprovisionadas accediendo a través de ella, haciendo uso del fichero docker-proxy-provisioning.tftpl.

docker-provisioning.tftpl

```
1 #!/bin/bash
2
3 # Which OS are we running on?
4 OS=$(cat /etc/os-release | grep '^ID=' | cut -d "=" -f 2 | xargs)
5
6 # Docker installation depending on the OS
7 case $OS in
8
9     "debian" | "ubuntu")
10    sudo apt-get update -y
11    sudo apt-get install ca-certificates curl gnupg lsb-release -y
12
13    if [[ $OS == "debian" ]]
14    then
15        curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg
16        --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
17        echo \
18        "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/
keyrings/docker-archive-keyring.gpg] https://download.docker.com/
linux/debian \
19        $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/
20        docker.list > /dev/null
21
22    elif [[ $OS == "ubuntu" ]]
23    then
24        curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg
25        --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

```

23     echo \
24     "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/
keyrings/docker-archive-keyring.gpg] https://download.docker.com/
linux/ubuntu \
25     $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/
docker.list > /dev/null
26 fi
27
28 sudo apt-get update -y
29 sudo apt-get install docker-ce docker-ce-cli containerd.io -y
30 ;;
31
32 "centos" | "rhel")
33 sudo yum install -y yum-utils
34
35 if [[ $OS == "centos" ]]
36 then
37     sudo yum-config-manager --add-repo https://download.docker.com/
linux/centos/docker-ce.repo
38
39 elif [[ $OS == "rhel" ]]
40 then
41     sudo yum-config-manager --add-repo https://download.docker.com/
linux/rhel/docker-ce.repo
42 fi
43
44 sudo yum install -y docker-ce docker-ce-cli containerd.io
45 ;;
46
47 "fedora")
48 sudo dnf -y install dnf-plugins-core
49 sudo dnf config-manager --add-repo https://download.docker.com/linux
/fedora/docker-ce.repo
50 sudo dnf install -y docker-ce docker-ce-cli containerd.io
51 ;;
52 esac
53
54 # Start Docker daemon and run Docker container
55 sudo systemctl start docker
56 sudo docker run ${args} ${image}:${tag}

```

docker-proxy-provisioning.tftpl

```

1 #!/bin/bash
2
3 # This script is used to provision a Linux instance using the instance
4 # configured as network proxy
5 # It receives the IP address of the proxy instance as a parameter
6
7 # Give time for the proxy instance to be fully configured
8 sleep 30
9
10 # Which OS are we running on?
11 OS=$(cat /etc/os-release | grep "^ID=" | cut -d "=" -f 2 | xargs)
12 PROXY_IP="http://${proxy}:3128"
13 HTTP_PROXY="http_proxy=$PROXY_IP"
14 HTTPS_PROXY="https_proxy=$PROXY_IP"
15
16 # Docker installation depending on the OS
17 case $OS in
18     "debian" | "ubuntu")
19         sudo $HTTP_PROXY apt-get update -y
20         sudo $HTTP_PROXY apt-get install ca-certificates curl gnupg lsb-
21         release -y
22         if [[ $OS == "debian" ]]
23             then
24                 curl -x $PROXY_IP -fsSL https://download.docker.com/linux/debian/
25                 gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-
26                 keyring.gpg
27                 echo \
28                 "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/
29                 keyrings/docker-archive-keyring.gpg] https://download.docker.com/
30                 linux/debian \
31                 $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/
32                 docker.list > /dev/null
33
34         elif [[ $OS == "ubuntu" ]]
35             then
36                 curl -x $PROXY_IP -fsSL https://download.docker.com/linux/
37                 ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-
38                 keyring.gpg
39                 echo \
40                 "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/
41                 keyrings/docker-archive-keyring.gpg] https://download.docker.com/
42                 linux/ubuntu \
43                 $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/
44                 docker.list > /dev/null
45         fi
46
47         sudo $HTTP_PROXY apt-get update -y
48         sudo $HTTP_PROXY apt-get install docker-ce docker-ce-cli containerd-
49         io -y
50         ;;
51 esac
52
53 # Start Docker daemon and run Docker container

```

```
43 mkdir /etc/systemd/system/docker.service.d
44 sudo -s
45 echo "[Service]"
46 Environment=HTTP_PROXY=$PROXY_IP
47 Environment=https_PROXY=$PROXY_IP" > /etc/systemd/system/docker.service.
48 d/http-proxy.conf
49 sudo systemctl daemon-reload
50 systemctl restart docker
51 docker run ${args} ${image}:${tag}
52 exit
```

proxy-config.tftpl

```
1 #!/bin/bash
2
3 # This script sets up a Debian instance as a Squid network proxy
4 # It receives as parameter the sources from which squid proxy should
5 # accept connections
6
7 sudo apt-get install squid -y
8 sudo sed -i 's:#\(\http_access deny to_localhost\):\1:' /etc/squid/squid.
8 conf
9 sudo sed -i "1188i acl internal src ${sources}" /etc/squid/squid.conf
10 sudo sed -i "1402i http_access allow internal" /etc/squid/squid.conf
11 sudo tee -a /etc/squid/squid.conf <<'EOF'
11 acl to_metadata dst 169.254.169.254
12 http_access deny to_metadata
13 EOF
14 sudo service squid restart
```