UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Gestión de Infraestructura en la Nube: Herramientas, Complejidad y Despliegue

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Tornal Hidalgo, Adrián

Tutor/a: Valderas Aranda, Pedro José

CURSO ACADÉMICO: 2023/2024

Resumen

Este trabajo aborda la gestión y optimización de infraestructuras en la nube, con un enfoque particular en la mejora de una infraestructura en un entorno bancario.
En primer lugar, se presenta un análisis teórico de la computación en la nube, destacando sus beneficios, como la flexibilidad y escalabilidad, así como los desafíos que plantea, especialmente por su extensión y complejidad, así como en la gestión eficiente de recursos. Se estudian las principales tecnologías y servicios que soportan la computación en la nube, con un énfasis en Amazon Web Services (AWS), dada su predominancia en el sector.

La automatización es uno de los pilares clave en la gestión moderna de infraestructuras en la nube, y en este contexto, se explora el uso de herramientas como Terraform. Esta tecnología permite definir y gestionar entornos en la nube de manera codificada, lo que facilita la replicación de infraestructuras, mejora la coherencia en el despliegue de recursos y reduce la posibilidad de errores humanos.

El trabajo no solo se queda en la teoría, sino que se aplica este conocimiento en un caso práctico centrado en la mejora de una infraestructura bancaria ya existente en AWS. La infraestructura original presentaba diversas deficiencias, principalmente en áreas críticas como la seguridad, la capacidad de escalado y la automatización de procesos. A través de la utilización de Terraform y otras herramientas, se implementan mejoras significativas que optimizan la gestión de recursos, refuerzan la seguridad y preparan la infraestructura para futuras expansiones.

Este caso práctico demuestra cómo la aplicación de buenas prácticas en la gestión de la nube, junto con una automatización eficaz, puede transformar una infraestructura existente en un entorno mucho más seguro, eficiente y preparado para las demandas futuras del negocio. Las mejoras introducidas fueron validadas mediante pruebas exhaustivas, confirmando que no solo se cumplieron los objetivos planteados, sino que se superaron las expectativas en términos de fiabilidad y cumplimiento normativo.

Palabras clave: Cloud computing, infraestructura en la nube, AWS, gestión de recursos, infraestructura como código, automatización, Terraform, buenas prácticas, seguridad, networking

---

Resum

Aquest treball aborda la gestió i optimització d'infraestructures en el núvol, amb un enfocament particular en la millora d'una infraestructura en un entorn bancari. En primer lloc, es presenta una anàlisi teòrica de la computació en el núvol, destacant els seus beneficis, com ara la flexibilitat i escalabilitat, així com els desafiaments que planteja, especialment per la seua complexitat i l'eficient gestió dels recursos. S'estudien les principals tecnologies i serveis que suporten la computació en el núvol, amb un èmfasi en Amazon Web Services (AWS), donada la seua predominança en el sector.

L'automatització és un dels pilars clau en la gestió moderna d'infraestructures en el núvol, i en aquest context, s'explora l'ús d'eines com Terraform. Aquesta tecnologia permet definir i gestionar entorns en el núvol de manera codificada, la qual cosa facilita la replicació d'infraestructures, millora la coherència en el desplegament de recursos i redueix la possibilitat d'errors humans.

El treball no es queda en la teoria, sinó que aquest coneixement s'aplica en un cas pràctic centrat en la millora d'una infraestructura bancària ja existent en AWS. L'estructura original presentava diverses deficiències, principalment en àrees crítiques com la seguretat, la capacitat d'escalat i l'automatització de processos. A través de l'ús de Terraform i altres eines, s'implementen millores significatives que optimitzen la gestió de recursos, reforcen la seguretat i preparen la infraestructura per a futures expansions.

Aquest cas pràctic demostra com l'aplicació de bones pràctiques en la gestió del núvol, juntament amb una automatització eficaç, pot transformar una infraestructura existent en un entorn molt més segur, eficient i preparat per a les demandes futures del negoci. Les millores introduïdes van ser validades mitjançant proves exhaustives, confirmant que no només es van complir els objectius plantejats, sinó que es van superar les expectatives en termes de fiabilitat i compliment normatiu.

Paraules clau: Computació en el núvol, Infraestructura al núvol, AWS, Gestió de recursos, Infraestructura com a codi, Automatització, Terraform, Bones pràctiques, Seguretat, Xarxes

Abstract

This work addresses the management and optimization of cloud infrastructures, with a particular focus on the improvement of an infrastructure in a banking environment. First, a theoretical analysis of cloud computing is presented, highlighting its benefits, such as flexibility and scalability, as well as the challenges it poses, especially due to its complexity and the efficient management of resources. The main technologies and services that support cloud computing are studied, with an emphasis on Amazon Web Services (AWS), given its dominance in the sector.

Automation is one of the key pillars in modern cloud infrastructure management, and in this context, the use of tools such as Terraform is explored. This technology allows cloud environments to be defined and managed as code, which facilitates the replication of infrastructures, improves consistency in resource deployment, and reduces the likelihood of human errors.

The work does not remain theoretical; instead, this knowledge is applied in a practical case focused on improving an existing banking infrastructure in AWS. The original infrastructure had several deficiencies, mainly in critical areas such as security, scalability, and process automation. Through the use of Terraform and other tools, significant improvements are implemented to optimize resource management, reinforce security, and prepare the infrastructure for future expansions.

This practical case demonstrates how the application of best practices in cloud management, along with effective automation, can transform an existing infrastructure into a much more secure, efficient environment, ready to meet the future demands of the business. The improvements introduced were validated through extensive testing, confirming that not only were the objectives met, but expectations were exceeded in terms of reliability and regulatory compliance.

Key words: Cloud computing, Cloud infrastructure, AWS, Resource management, Infrastructure as code, Automation, Terraform, Best practices, Security, Networking

Índice general

Apéndices

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

Actualmente la ingeniería en la nube tiene presencia en la gran mayoría de empresas de diversos sectores, desde la industria textil hasta el sector bancario ya que permite establecer infraestructuras escalables y flexibles en la nube para soportar aplicaciones y servicios críticos.

La nube se basa en recursos informáticos tales como servidores, bases de datos, recursos de almacenamiento, redes, etc. a los que se pueden acceder desde internet y se pueden manejar y replicar fácilmente. Teniendo definido el concepto de "la nube" en informática, la infraestructura en la Nube es una disciplina centrada en diseñar, administrar y mantener servicios y sistemas tecnológicos basados en la nube.

A lo largo del proyecto se van a exponer una variedad de servicios expuestos en la nube centrándonos en los relacionados con la infraestructura de las redes, se va a proceder a su definición, su comparación unos con otros y su uso, para así poder mostrar la complejidad de interactuar con la nube y la gran utilidad de esta. Se mostrará la complejidad y los retos de trabajar con la nube en la actualidad, los puntos a tener en cuenta para dar soluciones eficientes, con alta disponibilidad y eficaces también en cuanto a la economía.

Una vez explicados los recursos y servicios que se proporcionan en la nube se va a redactar una explicación sobre herramientas de gestión de los recursos y servicios anteriormente explicados, los proveedores existentes, dando paso a Amazon con AWS, uno de los proveedores de servicios en la nube más grandes existentes. Un punto clave en este apartado es la infraestructura como código, es la herramienta utilizada para desarrollar la infraestructura en la nube a partir de código, para ello se utilizan los mismos recursos que los usados visualmente en los portales de los proveedores, pero a partir de nombres predefinidos. La herramienta para producir infraestructura como código más utilizada, y en la cual nos vamos a centrar es Terraform, desarrollada por HashiCorp y utilizada en este caso junto al editor de código Visual Studio (desarrollado por Microsoft).

Tras tener todos los puntos claramente definidos se dará paso a un análisis de un caso sobre una infraestructura en la nube, los recursos y servicios que se utilizan, todos ellos servidos a través de AWS y desplegados mediante la herramienta de Terraform, se explicará su funcionamiento y como se comunican entre ellos. Además habrá un análisis de herramientas útiles para mejorar los despliegues de infraestructura y que dan soporte a este.

1.1 Motivación

A lo largo de los últimos años la informática ha seguido el crecimiento exponencial que lleva experimentando desde hace más de una década, y uno de los puntos más importantes es la migración de servicios que se daban en puestos físicos hacia la nube. Esto incluye la migración de hardware físico como son los servidores, routers, bases de datos, sistemas de almacenamiento, etc. a hardware virtualizado en máquinas, en servidores que contienen todo lo necesario para que las redes informáticas funcionen, y con ellos, los servicios que se proporcionaban en los datacenters físicos de las empresas, se pueden proporcionar de la misma manera en estos centros de datos virtualizados.

Este es un aspecto de la informática crucial hoy en día para todos los servicios que utilizamos, está detrás de todas las aplicaciones que usamos a diario y de todas las páginas web que visitamos, y teniendo en cuenta que una buena infraestructura en la nube, desde una vista empresarial, muestra más puntos a favor que en contra, es importante tener bien claros los puntos a definir en esta y tener unos conocimientos extensos sobre las opciones que se muestran y cómo utilizarlas, por lo que es importante que las empresas tengan personal cualificado en este aspecto si quieren hacer uso de este tipo de infraestructuras en sus proyectos.

A lo largo de las asignaturas que se cursan en la carrera de Ingeniería Informática se tratan muchos puntos sobre la infraestructura de las redes informáticas, aumentando más estos conocimientos y ahondando en ellos si se cursa la rama relacionada con estas redes informáticas. Pero, así como se tratan los componentes y funcionamientos de este apartado de la informática, no se explica como es el espectro en la actualidad, no se llega a mencionar el gran uso de la nube para proveer infraestructura en el ámbito empresarial ni la importancia de este aspecto.

Una vez acabé de cursar las asignaturas proporcionadas por el grado, empecé las prácticas en DISID CORPORATION SL, y comencé a interactuar con el mundo de "cloud engineering", y con ello entendí de la importancia del sector, así como de su complejidad, ya que aun con la realización de varios cursos, todos centrados en la nube por AWS, tenía el control de muy pocos sistemas de los existentes. Posteriormente entré en un proyecto relacionado con la realización de una infraestructura en la nube, donde teníamos que realizar una infraestructura desde cero, con pocas indicaciones, y teniendo en cuenta una estructura anterior mal realizada sin seguir ningún tipo de buenas prácticas, lo que dificultó todos los pasos que se pedían, y fue en ese momento donde surgió la idea de realizar el TFG acerca de la complejidad de la nube y los servicios que ofrece para mostrar cómo de importante es tener los conocimientos necesarios para poder interactuar con todos los recursos y saber utilizarlos desde un primer momento en los proyectos, así como mostrar de forma práctica esto mediante la explicación y redacción del propio proyecto.

1.2 Objetivos

El objetivo principal que aborda el trabajo es el de tratar de analizar y comprender la complejidad de la ingeniería en la nube, así como identificar y abordar los desafíos que esta presenta. Para ello se busca presentar una visión general de las herramientas y tecnologías existentes, centrando la atención en AWS, la plataforma de ingeniería en la nube proporcionada por Amazon) y Terraform como herramienta principal de infraestructura como código (concepto que se explicará más adelante), así como la utilidad y necesidad de estas herramientas en proyectos empresariales.

Además se tratará de explicar un despliegue utilizando varios servicios y herramientas proporcionadas por AWS, mejorando una estructura de las redes de un servicio bancario para representarla en la nube, para mostrar la creación de los recursos de la nube mediante las herramientas anteriormente explicadas.

1.3 Estructura de la memoria

El trabajo va a constar de diferentes apartados, tras exponer el estado actual de la informática en el ámbito de la ingeniería en la nube se ahondará en todo lo relacionado con el networking en este campo. Se tratará de dar a entender cuál es el cambio producido entre tener infraestructura de redes o en físico y tener esto mismo en la nube.

Tras explicar como es el networking y su estructura en la nube, se procederá a realizar una explicación de todos los componentes, recursos y servicios existentes para poder mostrar lo extenso del sector, así como las herramientas con las que se tratan estos.
Siempre se va a tratar de poner el foco en networking, pero también se hablará de componentes que se utilizan en extensión a las redes y que cumplen un papel importante como son los componentes de seguridad o de cómputo.
Todos estos componentes se explicarán desde la vista de AWS, proveedor más utilizado para la nube de cualquier empresa, pero también se mostrará como varían a lo largo de los proveedores, así como una comparación entre estos.

Tras esto, se expondrá la infraestructura como código, donde se explicará cómo se compone, como se trata, la forma de utilizar Terraform para crear recursos en la nube, más específicamente en la nube proporcionada por AWS, proveedor de Amazon.

Tras tener claro la utilidad y el uso de la infraestructura como código se realizará una breve exposición de herramientas que complementan el uso de esta, sobre todo herramientas de ayuda a la hora de realizar infraestructuras que sigan las buenas prácticas y de control de lo que se está creando.

El siguiente paso se basará en explicar las buenas prácticas a la hora de crear infraestructura en AWS.

Al tener toda la estructura explicada, los componentes junto a la forma de tratarlos, se procederá a explicar un caso de uso real sobre una infraestructura en la nube para un banco.
Se tratará el caso de uso desde la perspectiva de mejorar una infraestructura existente para aumentar en seguridad y eficiencia, se expondrán los recursos y servicios a utilizar, cómo se crean mediante Terraform para su uso en AWS, y cómo quedaría la foto final tras tener todo desplegado y por último mostrarán pruebas del buen funcionamiento de la infraestructura.

Por último se concluirá el trabajo tratando las conclusiones de este en un apartado relacionado con mostrar un resumen de lo mostrado a lo largo del proyecto, así como mi opinión personal acerca de este y una valoración sobre las posibles mejoras o cambios que se podrían realizar.

CAPÍTULO 2
Estado de la cuestión

El National Institute of Standards and Technology (NIST) define la computación en la nube como[1]:

"Un modelo que permite el acceso bajo demanda a través de la Red a un conjunto compartido de recursos de computación configurables (redes, servidores, almacenamiento, aplicaciones y servicios) que se pueden aprovisionar rápidamente con el mínimo esfuerzo de gestión o interacción del proveedor del servicio"

Este conjunto de servicios posee una serie de características, así como 3 modelos de servicio y 4 modelos de despliegue. Estos modelos son la base de todos los esquemas que se producen en la ingeniería en la nube, a partir de la combinacion de los sistemas de servicio junto a los modelos de despliegue se consigue formar cualquier infraestructura, y según las características del proyecto se escogerán unos servicios mediante unos modelos de despliegue[1].

2.1 Modelos de servicio

Con el paso de los años, la ingeniería en la nube ha crecido de forma exponencial, innvoando en los servicios que ofrece, hasta llegar a mostrar tres modelos típicos de servicio. Estos modelos de servicio dentro de la nube son los conocidos como IaaS, SaaS y PaaS, compuestos por características diferentes y con utilidades propias que pueden llegar a combinarse.
Pero fuera de estos modelos de servicio, actualmente se hace uso de otro modelo de servicio basado en contenedores (CaaS), modelo que se encuentra en crecimiento pero que actualmente ya cuenta con una sólid base de usuarios utilizándolo.

Figura 2.1: Modelos iaas, saas y paas

A continuación se va a realizar una explicación de cada uno de ellos, y que recursos de AWS se pueden encasillar en estos modelos.

2.1.1. IaaS

IaaS o infraestructura es un modelo que proporciona la infraestructura virtual y el hardware bruto, lo cual permite crear, gestionar y destruir almacenamiento y máquinas virtuales a través de un servicio basado en la web.[4]
El modelo IaaS es el resultado de la evolución de los servidores privados virtuales, que ya se conocen desde hace muchos años. El proveedor de IaaS ofrece al cliente un servidor virtual junto con una o más CPU que ejecutan varias opciones de sistemas operativos.[5]
El usuario final no gestiona ni controla la infraestructura principal de la nube, pero puede tener el control sobre el SO, almacenamiento y aplicaciones desplegadas.

Toda la infraestructura que se despliega en la nube mediante el formato IaaS puede escalar o desescalar según las necesidades del cliente.
Otra ventaja es que toda esta infraestructura se obtiene en apenas unos segundos o como minutos, ya que se encuentra en la nube y muestra por otra parte, una disponibilidad casi total en la mayoría de los casos.

IaaS en AWS

El modelo IaaS comprende unos recursos específicos que son comunes en todos los proveedores, pero tienen características, configuraciones y nombres diferentes según estos.

IaaS es la base de cualquier infraestructura en la nube, por lo que la mayoria de servicios que se ofrecen son de este tipo.

Centrándonos en AWS (ya que es el proveedor usado para el caso práctico) algunos de los recursos que ofrece como IaaS son las máquinas virtuales EC2, que son máquinas de cómputo potentes, con varias opciones de configuración que van asociadas a un pricing en concreto, de forma que pagas por la cantidad de recursos que usas así como el tiempo que están activas las máquinas. A parte de ofrecer capacidad de cómputo ofrece capacidad de almacenamiento como IaaS, ya que tiene servicios como S3, o Elastic Block Storage. [9] Además en relación a las redes y a la infraestructura como tal en la nube, AWS ofrece el modelo de VPC para crear comunicaciones entre los recursos, que también se entiende como IaaS por su capacidad de configuración según las preferencias del usuario.

Estos son solo algunos de los recursos que se encuentran en AWS como infraestructura como servicio, ya que en casi su completitud los proveedores de servicio en la nube se componen de este tipo de servicios.

2.1.2. PaaS

PaaS (Plataforma como Servicio) proporciona el marco necesario para crear, probar, desplegar, gestionar y actualizar productos de software. Utiliza la misma infraestructura básica que IaaS (Infraestructura como Servicio), pero también incluye los sistemas operativos, middleware, herramientas de desarrollo y sistemas de gestión de bases de datos necesarios para crear aplicaciones de software.[6]
El cliente no tiene control sobre la infraestructura de la nube, como servidores, redes, almacenamiento o sistemas operativos, mientras que si tiene control sobre las aplicaciones desplegadas y sus configuraciones.

Este tipo de servicio ofrece algunas ventajas ya que es más fácil de controlar, de configurar y de escalar ya que depende del proveedor, así como el control de costes.
Pero esto mismo se puede considerar una desventaja según quien requiera el uso del servicio, ya que el no poder controlar según que aspectos puede ser perjudicial para expertos que prefieren tener el control sobre donde despliegan el software.

PaaS en AWS

La mayoría de proveedores de la nube ofrecen diferentes opciones de servicios PaaS en su catálogo, y por tanto lo mismo sucede con AWS

Algunos de estos servicios son:

- EC2 Elastic Beanstalk
  Como PaaS relacionada con el cómputo. En este servicio se puede implementar y administrar aplicaciones rápidamente en la nube de AWS sin tener que preocuparse por la infraestructura que las ejecuta. Se despliega de manera autónoma una infraestructura mediante máquinas EC2 que da soporte a la aplicación desplegada[10]

- AWS Lambda
  Otra PaaS relacionada con el cómputo. En este caso se despliega código directamente sobre la función lambda, y se ejecuta sin un servidor detrás. Permite una gran cantidad de lenguajes y puede interactuar con otros servicios de AWS.

- RDS
  En este caso es una PaaS de almacenamiento, ya que ofrece bases de datos relacionales autogestionadas. Sin necesidad de configurar la base de datos se disponen de manera correcta al subirlos.

De esta forma AWS ofrece la capacidad de utilizar servicios que serían complejos de gestionar de una manera sencilla para su uso, donde el cliente solo se preocupa en desplegar sus aplicaciones o datos.

2.1.3. SaaS

Es un modelo que permite a los clientes usar y alquilar las aplicaciones del proveedor sin tener que instalarlas en sus propios PC [4]. Esto significa que las aplicaciones con licencia proporcionadas a los clientes se ejecutan en la infraestructura de la nube a través de interfaces de cliente como Google Chrome, Internet Explorer y muchos otros. De esta forma el cliente solo se preocupa por utilizar la aplicación, sin tener la responsabilidad de mantener ni el código ni la infraestructura donde corre esta.

La mayor ventaja es la falta de responsabilidad sobre cualquier problema en la aplicación ya que el cliente solo está para su uso, así como reduce el coste de la aplicación ya que solo se paga por usar esta.

SaaS en AWS

AWS como tal no ofrece servicios de SaaS como si puede hacerlo Google con su servicio de email (gmail). Pero mediante sus recursos se pueden crear servicios SaaS personalizadas de terceros [11]. AWS ofrece AWS SaaS Factory, donde experimentar con SaaS y creaciones de SaaS de terceros, y además pone su Marketplace al público para subir soluciones SaaS y que puedan ser utilizadas.

2.1.4. CaaS

Un contenedor trata de una forma de virtualización y cómputo, ya que no hace uso de una máquina virtual. Los contenedores proporcionan una virtualización dentro del kernel del sistema, dando mayor portabilidad, eficiencia, capacidad de uso y otras tantas ventajas en comparación con las típicas máquinas virtuales. [12]

Es por ello que tras el auge del uso de contenedores y tecnologías como docker (crear y ejecutar contenedores) y kubernetes (orquestar contenedores), se ha establecido una nueva forma de servicio en la nube relacionada con estos llamada contenedores como servicio o CaaS, los cuales se usan con la estructura de la imagen 2.2.

Container as a Service (CaaS) aborda el problema de las aplicaciones desarrolladas en un entorno PaaS específico, cuyas ejecuciones están restringidas a las especificaciones de dicho entorno PaaS. Al adoptar CaaS, la aplicación se vuelve completamente independiente de las restricciones del entorno PaaS, eliminando así las dependencias. Esto permite que las aplicaciones basadas en contenedores sean portátiles y se ejecuten en cualquier entorno de ejecución. [12]

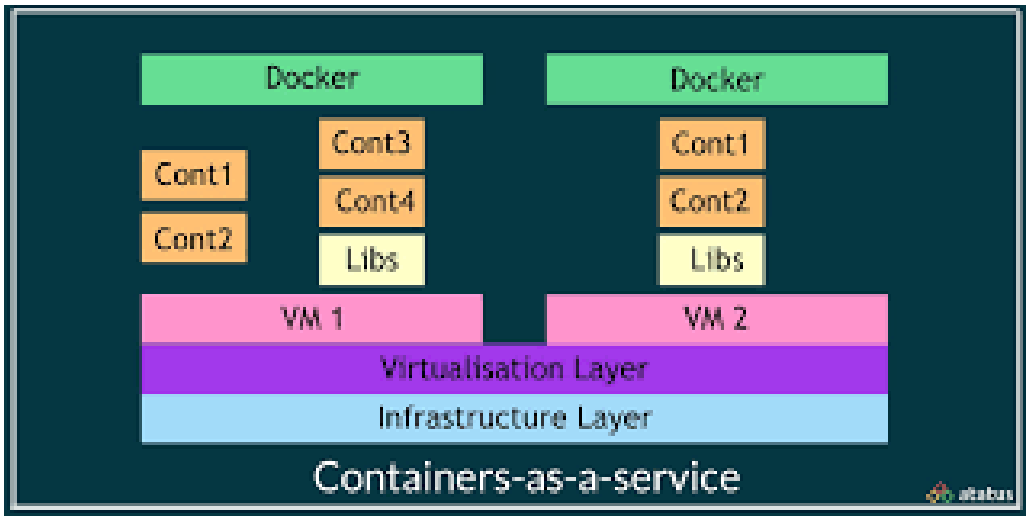

Figura 2.2: Contenedores como servicio

CaaS en AWS

El uso de contenedores como servicio ha crecido hasta el punto en el que la mayoría de proveedores en la nube tienen un sistema propio de contenedores, y AWS es uno de ellos.
Amazon tiene su sistema ECS insertado en AWS, mediante el cual se pueden gestionar contenedores en la nube compatibles con Docker. No se va a ahondar más en las características de estos sistemas ya que tienen una gran complejidad, recursos propios, servicios propios y formas de coordinación propias.

Aparte de ECS Amazon dispone de el servicio EKS, un servicio administrado de kubernetes mediante el cual gestionar clústers de kubernetes formados por contenedores y otros recursos.

Actualmente los despliegues mediante kubernetes y contenedores están sustituyendo a despliegues con máquinas virtuales de tipo IaaS, ya que ganan en velocidades, eficiencia y económicamente, la única desventajas es la comlejitud de los sistemas a desplegar.

2.2 Modelos de despliegue

Los diferentes modelos de despliegue en la nube han surgido debido a la creciente demanda de la computación en la nube. La computación en la nube opera bajo un modelo de pago por uso, proporcionando servicios de computación online según las necesidades del usuario.

El despliegue en la nube describe cómo se implementa y se aloja una plataforma en la nube, y quién tiene acceso a ella. Todos los modelos de despliegue en la nube se basan en la virtualización del poder de cómputo de los servidores, dividiendo este poder en aplicaciones segmentadas y controladas por software que ofrecen capacidades de procesamiento y almacenamiento.

Las nubes difieren en características como capacidades de almacenamiento, sistemas de facturación y métodos de provisión de servicios. Debido a estas diferencias, elegir el modelo de despliegue adecuado puede ser complicado, ya que los usuarios deben considerar cuál se adapta mejor a sus requisitos específicos.

A continuación se va a realizar un análisis de los diferentes modelos de despliegues en la nube, explicando sus diferencias y cuál se acopla más a las necesidades en según que casos.

2.2.1. Nube privada

Se entiende como nube privada a aquella red privada dentro de una empresa, manejada direcrtamente por ella y a la cual solo pueden acceder una cantidad fija de empleados de la propia empresa.

Esta nube privada puede estar alojada dentro de los propios servidores de la empresa, o que se aloje en un proveedor externo como podría ser AWS. La segunda opción ayuda a manejar el hardware de la nube, ya que es proporcionado por el propio proveedor.

Un ejemplo de recursos en la nube privada puede ser VMWare de AWS, servidores virtualizados pertencientes al proveedor. Con AWS puedes crear redes aisladas mediante el servicio de VPC, ya que, aunque pertenezca a un conjunto de redes públicas, puede ser utilizado de forma aislada para la empresa que lo requiera.

2.2.2. Nube pública

La nube pública trata de un tipo de nube la cual puede ser accedida por clientes a través de un servicio externo vía interner, y sirve generalmente para páginas web, compartición de archivos y para almacenar datos no restringidos. [6]

Este tipo de red se utiliza a través de proveedores que tienen una gran cantidad de servidores distribuidos a lo largo de los países. Estas redes suelen contar recursos y servicios comerciales que estan estandarizados y aprovisionados dinámicamente, con tecnologías a través de internet y con un pago por uso.[7]

Un ejemplo de tipo de nube pública es la Elastic Compute Cloud de AWS, con máquinas de cómputo públicas que pueden ser utilizadas por los clientes. Son recursos gestionados por el proveedor, por lo que este paso de control se gestiona a partir de este proveedor dando al cliente otras responsabilidades.

2.2.3. Nube híbrida

La nube híbrida trata de una combinación de nube privada y nube pública, donde, al menos, se encuentra una nube de cada tipo.

En este tipo de nubes se transportan datos entre nubes, de forma que se gestionan datos tanto interna como externamente. Este tipo de nubes son las más utilizadas empresarialmente, ya que permite interactuar con proveedores externos y las nubes de estos, y además permite guardar datos de la forma más segura posible en las redes privadas del cliente. [8]

Un ejemplo de formación de una nube híbrida puede ser una empresa con datos sobre recursos humanos, que son menos importantes de proteger en una nube pública como una nube de AWS, y a la vez, tiene datos bancarios en una nube privada propia.

2.2.4. Nube comunitaria

Una nube comunitaria tiene como característica principal la compartición de estas por varias organizaciones o conjunto de usuarios que tienen unos mismos requisitos, finalidades, necesidades o políticas. [6]

El término nube comunitaria se usa para darle un nombre de marketing, ya que realmente se trata de una nube privada, pública o híbrida que se reutiliza por varios clientes.

Un ejemplo de nube comunitaria podría ser una red de hospitales que utiliza la misma red para el almacenamiento y compartición de datos. Estas redes podrían formarse como un conjunto de VPC de AWS conectadas mediante peering y distribuidas mediante AWS Organizations.

2.3 Modelos utilizados en el caso práctico y comparación con otros casos

En el caso práctico que se expone al final de este documento se ha hecho uso de unos modelos de servicio junto a unos modelos concretos de despliegue para llegar a la solución expuesta.

2.3.1. Modelos de servicios en el caso práctico

En cuanto al tipo de servicios ofrecidos en la solución, se epxonen tanto IaaS como PaaS.

En relación a la infraestructura como servicio en el proyecto se hace un uso de recursos como las VPC que pertenecen a este modelo de servicio, ya que se ofrece la infraestructura mediante el hardware y el cliente se encarga de personalizar las características que desea.
Además dentro de cada VPC se encuentran recursos como Nat Gateways o VPC peerings que también siguen la definición de IaaS.

Por otra parte en relación a los PaaS se encuentran recursos como Elastic Beanstalk, AWS Lambda y Load Balancers, que siguen la estructura de este tipo, ya que ofrecen una plataforma entera gestionada por AWS y el cliente solo se encarga de su despliegue y uso.

Otros modelos de servicio posibles

Así como en el proyecto expuesto se muestran servicios de tipo IaaS y PaaS, existen otros proyectos en la actualidad que no solo se desarrollan con recursos de este tipo, ya que, centrándonos en AWS existe otro tipo de modelo de servicio anteriormente definido, el conocido como CaaS.

Un ejemplo de uso de CaaS es la creación de una infraestructura mediante el uso de AWS ECS, mediante la contenerización de wordpress para tener una imagen de una página web en un contenedor administrada por docker.
[13] De esta forma se obtiene las imágenes de la página web, administradas por docker, y se despliegan utilizando primero ECR para mantener el registro en AWS, y posteriormente se usa ECS para administrar las imágenes y darles un punto de despliegue y conexión. Para este ejemplo en el que me baso, y que he obtenido a través de la referencia [13], se utiliza un despliegue mediante EC2 a partir de ECS.

Esta sería una adición a los modelos de servicio utilizados en este trabajo, un tipo de servicio que no se utiliza en mi caso, y que existe actualmente como posibilidad en la nube.

2.3.2. Modelos de despliegue en el caso práctico

En el caso que se expone en el apartado final de este trabajo se utiliza una nube híbrida, donde predomina el uso de nube privada ya que la mayor parte de la solución propuesta se basa en la creación de una red privada única dedicada al cliente, donde se suma una serie de redes públicas dentro de una VPC de AWS con recursos a los cuales se conectarán los clientes.

Al deberse a la red de un banco todos los recursos relacionados con datos bancarios de clientes guardados en bases de datos, así como las aplicaciones que los tratan, se encontrarán en subredes privadas, lo que genera una red de comunicación interna al cliente que solicitó la infraestructura.
Pero por otra parte existe una serie de recursos como un servicio de url, o ciertos puntos de conexión a APIs de bancos externos que se consideran externos, lo que genera esta red pública que crea una red híbrida. Este aspecto de red pública es externo a las acciones que se realizan en el proyecto que presento, pero que se debe tener en cuenta por la naturaleza de este, con esto me refiero a que no se va a desplegar ningún aspecto relacionado con una red pública a lo largo del proyecto, pero que en un futuro estos componentes existirán en el proyecto.
Es por esto que la conclusión acerca del modo de despliegue en el caso que presento es un despliegue mediante una red híbrida con predominancia de la red privada.

Otros modelos de servicio posibles

El modo de despliegue de red híbrido es uno de los modos más empleados en el aspecto empresarial, pero como ya se ha expuesto anteriormente se puede hacer uso de una sola red privada, pública o de redes comunitarias.

El uso de una red privada o una red pública podría considerarse como una distinción de las redes que considero en mi proyecto o en el proyecto explicado para CaaS en un aprtado anterior [13] donde se expone una aplicación web mediante ECS. En este último caso de la aplicación web se generaría una nube pública ya que los recursos mostrados en la web son públicos a los clientes, solo en caso de tener un sistema de login existiría una parte privada para no estar estos datos personales expuestos a todo Internet.

Y en cuanto a una red privada podría ser la consideración de solo el apartado que desgloso en mi explicación acerca de mi caso práctico, donde solo se tiene en cuenta el apartado de TEST donde se realizarán pruebas internas, de forma que toda la red de la aplicación bancaria será privada.

Por otra parte tenemos una red completamente diferente, una red comunitaria, que se usa hoy en día para organizaciones que comparten recursos y propósitos como he explicado en el apartado 2.2.4. Un ejemplo de este tipo de despliegue sería el que usa un conjunto de universidades, un sistema de salud de un país o el gobierno de un país.[14]

Un ejemplo en concreto de un despliegue que se puede hacer mediante community cloud sería para una red de sanidad de un país como he nombrado anteriormente. De esta manera existe un proyecto producido en Pakistán donde se expone como se podría desplegar una red mediante una nube comunitaría para el sistema de salud del país. [15] En este proyecto citado se expone como a partir de los datos de los hospitales del país sería beneficioso crear una red comunitaria para todos estos, aumentando en velocidad y seguridad, a partir de proveedores como IBM, donde almacenar y compartir datos entre todos los hospitales.

Este ejemplo es aplicable a los diversos campos que he nombrado anteriormente, ya que todos los sectores donde se traten datos compartidos, es posible utilizar esta solución de redes comunitarias.

CAPÍTULO 3
Metodología

La metodología a la hora de desarrollar un proyecto se entiende como una unión de técnicas, métodos y procedimientos a seguir durante el desarrollo de un proyecto para obtener resultados favorables al final del desarrollo. También puede incluir competencias estratégicas de la empresa para alinear los objetivos del negocio con los resultados del proyecto. Las metodologías son útiles ya que sirven para facilitar las tareas de planificación y tareas de control y seguimiento, mejorar la relación costo/beneficio, optimizar el uso de recursos, facilitar la evaluación de resultados y cumplimiento de los objetivos, mejorar la comunicación, optimizar las fases del proyecto, mejorar la calidad del producto final, entre otros. A la hora de aplicar metodologías para realizar proyectos en informática, existen dos enfoques ampliamente conocidos y utilizados, el enfoque en cascada y el enfoque ágil, los cuales se discutiran a continuación.

3.1 Enfoque en cascada

Pertenece a un tipo de enfoque tradicional, utilizado desde 1970 por Winston W. Royce. Este enfoque se caracteriza por representar las fases del proyecto de manera secuencial y lineal, de forma que se comienza la siguiente fase una vez la anterior esté completada íntegramente.

Las fases que componen este enfoque se desglosan como:

- Requisitos: Definición y documentación detallada de todos los requisitos del proyecto.
- Diseño: Creación de la arquitectura del sistema y diseño detallado.
- Implementación: Desarrollo y programación del sistema.
- Pruebas: Verificación y validación del sistema desarrollado.
- Mantenimiento: Corrección de errores y mejoras continuas después de la entrega.[2]

Como ventajas se encuentra su facilidad de administración ya que cada fase tiene una entrega fija y de revisión, además tiene una fácil planificación y programación ya que todo esto se realiza en la primera fase y se queda fijo en todo el ciclo de vida del proyecto. Como desventajas se encuentra que es un enfoque lineal y rígido, sin opción a cambios en caso de que sucedan hechos inesperados, y que en caso de que los requisitos no se definan a la perfección desde un principio puede haber problemas a lo largo del desarrollo del proyecto. Por estas razones este enfoque es para proyectos de menor tamaño, que tienen unos requisitos fijos definidos desde el principio, como pueden ser proyectos internos de

una empresa donde el cliente no puede variar los requisitos o añadir funcionalidades una vez comenzado el proyecto.

3.2 Enfoque ágil

Este tipo de enfoque surgió a partir de 2001 por trabajadores de tecnologías de la información, a partir de la creación del manifiesto ágil. El manifiesta consta de una serie de principios y valores, los cuales sumados entre sí resultan en un cambio de mentalidad, en una nueva forma de organización para realizar proyectos.[3] Este enfoque tiene su base en el uso de fases dinámicas llamadas "sprints", con una duración definida y con entregas continuas definidas al comienzo de cada sprint, cuyas características y requisitos están definidos con unos mínimos, pero sujetos a cambios. Mediante este enfoque se busca una mayor flexibilidad, colaboración, retroalimentación y respuesta rápida a los cambios que se puedan generar ya sea en el proyecto o en los requisitos por parte del cliente. Los valores de las metodologías ágiles se sustentan en la agilidad, transparencia y eficiencia.

Dentro del enfoque ágil se encuentran varias ramas, de las cuales se va a proceder a realizar un comentario acerca de las dos más utilizadas:

3.2.1. Scrum

Mediante el uso de los sprints, la metodología se enfoca en la entrega incremental de valor y en la mejora continua a través de varias ceremonias:

- Sprint Planning: Planificación del trabajo a realizar en el sprint.
- Daily Scrum: Reuniones diarias para sincronizar las actividades del equipo.
- Sprint Review: Revisión del trabajo completado al final del sprint.
- Sprint Retrospective: Reflexión sobre el sprint para identificar áreas de mejora.

Con este tipo de metodología ágil se crea un pan a medio plazo, de manera que, se saltará de un sprint a otro teniendo en cuenta el anterior, ya que si existe desarrollo en proceso y añadido sin ejecutar, una vez haya acabado el sprint, se unirá al desarrollo del sprint siguiente modificando la situación de este, así como la dedicación en horas. Al acabar el sprint se presenta el momento del proyecto con todo el equipo. Todos estas características se muestran en la imagen 3.1. Ofrece una gran adaptación a cualquier problema emergente, así como un marco de revisión constante.

Figura 3.1: Metodología Scrum

3.2.2. Kanban

Kanban es una rama de la metodología ágil utilizada para mostrar una visual del flujo de trabajo y de las tareas correspondientes con el proyecto. Sitúa tiempos máximos por tarea, límites de tiempo así como una organización por niveles según las tareas (padre e hijos) mediante diferentes columnas según el estado de la tarea (to do, in progress, done).

3.3 Metodología utilizada en el caso práctico

Para este proyecto la metodología escogida es una metodología ágil que combina el uso de Scrum y Kanban, añadiendo también el uso de GIT para el control de versiones. Para ello se utiliza la herramienta JIRA, software de gestión de proyectos desarrollada y distribuida por ATLASSIAN.

Al recibir el caso del cliente se forma un plan con varios sprints llamados fases de desarrollo, mostrados en la imagen 3.3. Estos sprints pertenecen al aspecto de Scrum, ya que se desarrollan con tareas internas a cada sprint, que, en caso de no abordarlas todas en ese límite de tiempo (5 semanas como máximo por sprint) pasan a tomar partido en el sprint siguiente, remodelando este último para cumplir con los tiempos de entrega.

Tras la recopilación de requisitos se divide el trabajo por lotes, según el equipo de trabajo, por lo que en este caso solo se trabaja en el lote B, ya que trata de la primera parte del desarrollo relacionada con crear la infraestructura necesaria para instalaciones y despliegues posteriores. Este lote por tanto tiene una duración de las 5 semanas del sprint, de mayo a junio, donde suceden las diferentes tareas del sprint o fase 2, que supone la creación de toda la red. Para tener más claro la distribución de tareas dentro de un sprint se hizo uso de un tablero kanban proporcionado por la propia herramienta de JIRA, el mostrado en la figura 3.2. Este tablero consta de las siguientes columnas:

- Por hacer: Tareas asignadas, que no han sido comenzadas
- En curso: Tareas comenzadas, pero sin haber acabado
- Bloqueado: Tareas que se encuentran en desarrollo pero no pueden seguir avanzando hasta que otra tarea, o un avance en el proyecto, las desbloquee

- Validación: Tareas completadas a falta de ser comprobadas y validadas

- Pruebas: Tareas relacionadas con realizar pruebas a despliegues de otras tareas

- Hecho: Tareas finalizadas y validadas

De esta manera las tareas van sucediéndose y completándose según preferencia o necesidad, de forma que, a la vez que se van realizando, van asignándose horas de cómputo dentro del proyecto, para una vez acabado, tener el desglose de horas usadas por tarea dentro de todo el proyecto. Estas horas en Jira se muestra dentro de la propia tarea, como en la imagen 3.2.

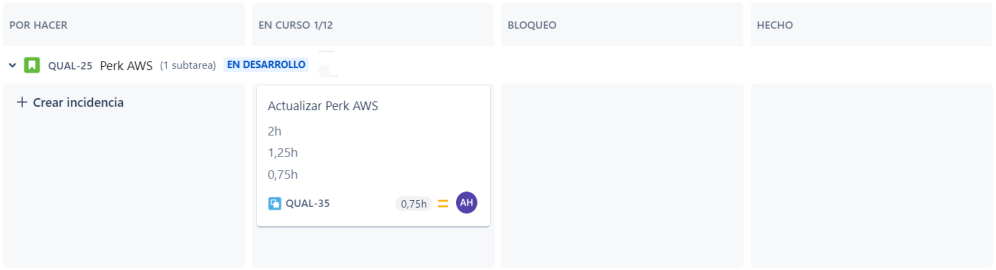

Figura 3.2: Tarea sobre actualizar un perk de AWS

Para el caso de este proyecto se realizaron reuniones diarias con los responsables para aportar el seguimiento diario así como dar ayuda a problemas que iban surgiendo mediante el paso de los días y con ello de las tareas.
De esta misma forma se pactó una reunión semanal con el responsable del proyecto por parte del cliente para mostrar avances y feedback acerca del desarrollo.

Al final del sprint se produjo una reunión y puesta en común del estado del proyecto, así como una presentación de tareas que no habían sido completadas en el plazo estimado para poder solucionar estas diferencias de horas en el sprint siguiente, moviéndose a la columna en curso para el siguiente sprint, para que, pudieran finalizar en la columna Hecho, donde debían acabar todas las tareas.

Además, durante las semanas, el gestor del proyecto por parte del cliente fue solicitando cambios acerca de algunos despliegues como el uso de un tipo de entrada a la red u otra, lo que llevaba a realizar cambios en las tareas y las horas de estas. Un ejemplo de este suceso ocurre en la tercera semana de producción al añadirse una nueva funcionalidad de conexión entre redes de AWS y una entrada por un punto de entrada de una VPC, lo que alargó una de las tareas del segundo sprint que acabó en el tercero. Esto fue posible gracias a la metodología ágil que se usó desde un principio.
Todo el planteamiento de reuniones y feedback con el cliente sucede por el hecho de ser un proyecto empresarial, ya que es necesario tener un plan desde el principio mediante las metodologías propuestas así como validar los avances con el cliente.

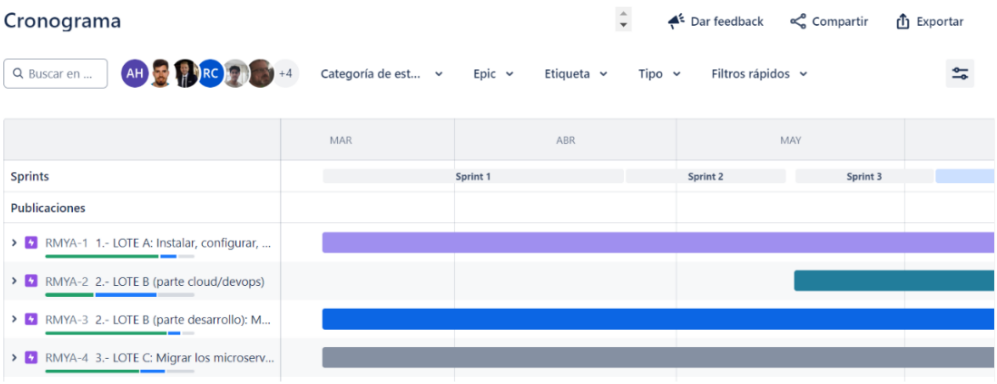

Figura 3.3: Ejemplo de los sprints de un proyecto

CAPÍTULO 4
Networking

En el aspecto de la ingeniería en la nube, tal y como se ha explicado en el apartado introductorio, se encuentran varios campos relacionados con la informática como la seguridad, computación, machine learning, hosteos de aplicaciones, etc. Pero para que se pueda dar esto, debe existir una infraestructura de redes de comunicaciones detrás de todos los componentes, una arquitectura de red que sea suficiente para los requisitos pedidos por quien solicita el proyecto, que también proporcione la seguridad necesaria y que a su vez ofrezca capacidades de una buena eficiencia y eficacia a la hora de transmitir los datos entre los recursos que se sitúen en ella. Este apartado basado en pensar, crear y desplegar una infraestructura de red donde se posarán los componentes finales de un sistema, es el apartado de redes, también conocido como "networking".

La creación de una arquitectura es clave para cualquier proyecto basado en la nube, y, así como se podrían definir una cantidad fija de componentes que componen el machine learning (IA) en un proveedor en la nube, no es tan exacto este número a la hora de hablar de la redes en la nube. Las redes tienen unos componentes básicos, donde se forma la red, pero para crear accesos o manejar quien o que puede acceder a la red, encontramos componentes que podríamos asignar a un conjunto de recursos diferentes a los considerados de la red. Para construir una arquitectura compleja y totalmente funcional, los criterios solicitados por el cliente que requiere la arquitectura, es necesario el uso de otros componentes, generalmente de computación y seguridad, para securizar las redes, controlar los accesos a estas y realizar formas de entrada. Toda la creación de las redes en la nube depende del proveedor escogido, ya que existe una extensa lista de estos:

- AWS, provisto por Amazon, el mayor de todos, con más servicios y estadísticamente el más utilizado. El seleccionado para exponer todos los servicios del proyecto.
- Oracle cloud
- Azure
- IBM Cloud
- Google Cloud (GCP)
- Red Hat

Cada uno de estos proveedores da acceso a una cantidad de recursos semejantes, pero todos ellos poseen ventajas y desventajas, por lo que según el proyecto puede ser que uno de ellos termine siendo favorable frente al resto. Desde el propio AWS se muestra una diferenciación en grupos a la hora de conjuntar los servicios de la nube según sus usos y características. A continuación se muestra la imagen 4.1 que resume a la perfección esta

diferenciación por grupos los diferentes grupos existentes como el de Networking, el relacionado con la integración de aplicaciones, los compuestos por inteligencia artificial o análisis de datos.

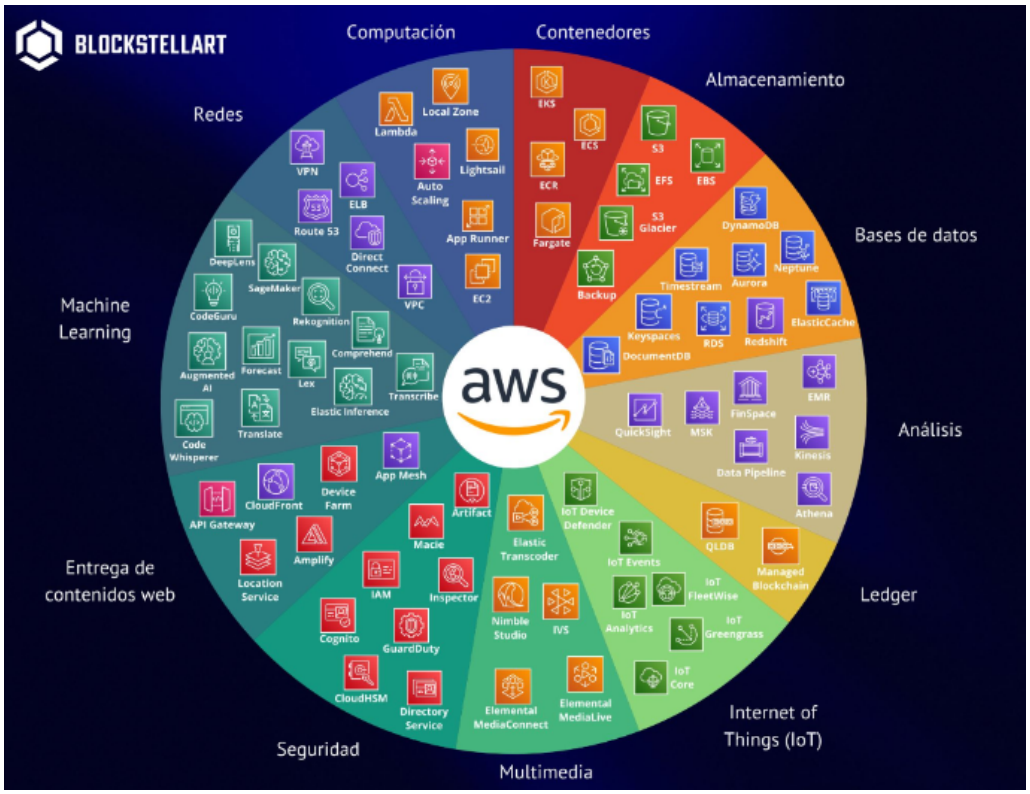

Figura 4.1: Recursos de AWS por grupos

4.1 Recursos Networking en AWS

Como se puede observar AWS tiene una gran cantidad de categorías de servicios, y dentro de estas categorías tenemos multitud de estos, por lo que es una tarea muy compleja el poder tener el conocimiento de todos ellos.

Por esto mismo, y por la complejidad de explicar cada uno de ellos, se va a centrar la explicación en los servicios de las redes y entrega de contenido o "networking", y los recursos más comúnmente utilizados en conjunto a estos servicios, así como los recursos más comunes en cualquier arquitectura, es decir, elementos como las máquinas virtuales, que siempre estarán presentes en proyectos donde se involucre una creación de una red en la nube.

Para poder entender cómo funciona la red de AWS, primero debe explicarse cómo se conforma la red en sí, como se estructura Amazon en el mundo con su servicio de AWS. Amazon tiene desplegados múltiples centros de datos alrededor del todo el mundo, construyendo una disponibilidad de 33 regiones diferentes, desde Estados Unidos hasta Osaka pasando por toda Europa. Dentro de estas regiones, llamadas por acrónimos dentro de AWS (eu-west-1, us-east-1), se encuentran zonas de disponibilidad (nombradas como AZ en el proveedor), es decir, zonas donde se sitúan centros de datos distanciados entre sí, donde se provee de los recursos necesarios para dar todo tipo de soluciones a la nube. Estas zonas varían según la región, ya que, por ejemplo se asocian 4 zonas a Oeste de Estados Unidos, mientras que a la región Oeste de Europa se le proporcionan 3 zonas de disponibilidad. [16]

Además, existen zonas locales, que tratan de puntos donde se proporcionan bases de datos, almacenamiento y otros servicios a los usuarios finales, para proporcionar respuestas de milisegundos si es necesario.[16]

Teniendo en cuenta cómo se estructura AWS, a continuación se va a realizar una explicación de todos los componentes que forman la categoría de networking, cada uno con su respectiva explicación para mostrar el uso real de ellos.
Para la redacción que sigue, se ha llevado a cabo una revisión exhaustiva del curso de KodeKloud sobre Amazon Web Services, específicamente el curso preparatorio para la certificación .ªWS Solutions Architect Associate". Este curso, disponible en [17], proporciona una explicación detallada de numerosos recursos y conceptos clave que se presentan a continuación. Muchas de las ideas y definiciones expuestas en esta sección se han adquirido a partir de dicho curso en línea.

4.1.1. VPC

Una VPC (Virtual Private Cloud) trata de una red virtual privada mediante la cual se controlan las comunicaciones entre componentes y es específica de una región de AWS. Al crear una VPC, se selecciona la región donde estará ubicada, por ejemplo, us-east-2. No es posible extender una VPC más allá de la región en la que se crea.

Inicialmente, una VPC no tiene comunicación con otras VPC en la misma región. Por motivos de seguridad, las VPC se encuentran aisladas y es necesario otorgar permisos explícitos para habilitar la comunicación entre ellas. Cada VPC tiene asignado un bloque CIDR que define el rango de direcciones IP que pueden utilizar los recursos dentro de la VPC. Estos bloques CIDR tienen un tamaño que varía entre /16 y /28 de direcciones IP, pero se pueden ajustar dentro de este rango.

Por ejemplo, si el CIDR es 192.168.0.0/16, el rango de direcciones IP será de 192.168.0.0 a 192.168.255.255. Además del bloque primario de direcciones IPv4, se pueden agregar bloques secundarios de direcciones IPv4, así como bloques de direcciones IPv6 con un tamaño de CIDR de /56, con un límite máximo de 5 bloques adicionales, aunque este límite es ajustable.

VPC predeterminada

Se crea automáticamente al abrir una cuenta en AWS, una por cada región. Estas VPCs vienen con una configuración predeterminada que incluye la conexión a internet para los recursos, pero son básicamente VPCs configuradas de manera sencilla. Tienen un bloque CIDR /16, que siempre es 172.31.0.0/16. Para cada zona de disponibilidad habilitada (por ejemplo, us-east-1a o us-east-1b), hay una subnet predeterminada /20. La VPC predeterminada incluye un gateway a internet, a través del cual el tráfico se dirige a 0.0.0.0/0, proporcionando así acceso a internet para los recursos de la VPC y permitiendo la comunicación con dispositivos externos. Además, viene con un grupo de seguridad por defecto que gestiona la salida de datos y una lista de control de acceso a la red (NACL) que regula el tráfico de entrada y salida a internet.

VPC personalizada

Son VPCs creadas por el usuario, donde se especifica la configuración completa, incluyendo el CIDR, la configuración de la red y cualquier otra configuración necesaria. Al crear una VPC personalizada, se selecciona la región deseada y se elige entre crear solo la VPC o también incluir subredes y tablas de enrutamiento. Se proporciona un nombre

para la VPC, se selecciona el bloque CIDR, se indica si se desea utilizar IPv6, y se pueden añadir etiquetas opcionales. Una vez completados estos pasos, la VPC personalizada está lista para ser creada.

4.1.2. Componentes dentro de una VPC

Muchos de los componentes que forman el campo de Networking en AWS van unidos a una VPC, ya que se especifican dentro de ella y tienen una utilidad sobre la comunicación dentro o fuera de esta. Estos componente son:

Subnets

Las subnets son grupos de direcciones IP proporcionadas dentro de una VPC, por lo que este grupo de direcciones dependerá del bloque CIDR anteriormente asignado a la VPC donde se encuentren. Las subredes se encuentran situadas dentro de una zona habilitada en la región, y como característica de AWS puede haber más de una subnet por AZ (Availability zone), pero una subred solo puede estar asignada a una AZ. Otras características de este componente de una VPC es que las subnets no pueden tener IPs compartidas entre ellas, los grupos de IPs tienen que ser distantes entre las diferentes subnets de una misma VPC. Por otra parte las subnets pueden configurarse como públicas o privadas, para que acepten o no tráfico desde el exterior, pero para ello es necesario proveer otros componentes, unas puertas de salida al exterior ya que las subnets están preconfiguradas para que sean privadas.

Routing

Todas las subnets vienen creadas con una tabla de enrutamiento ya que todas ellas tienen una interfaz de un router designado, debido a que las VPCs están configuradas con un router por defecto. Esta interfaz del router tiene una tabla de enrutamiento predefinida (aceptando todo el tráfico dentro de la VPC) y configurable, para asignar las reglas que sean necesarias a la hora de permitir o no las comunicaciones.

Internet Gateway

Son puertas de enlace con internet para permitir la salida al exterior de una subnet. Esta puerta se crea en una vpc y se asigna la dirección de este recurso a la tabla de enrutamiento de una subnet para dirigir el tráfico saliente hacia una IP que no esté especificada en la tabla de enrutamiento, hacia esa puerta y así poder salir a internet. Si esta puerta no está creada, no habrá comunicación entre internet y los recursos que se encuentren en las subnets de una VPC. Como última necesidad a la hora de permitir la comunicación con el exterior de un recurso de una subnet es asignarle una IP pública, ya que si no se provee de una IP pública, no se podrá conectar con Internet.

Nat Gateway

Son otro tipo de puertas semejantes a las Internet Gateway, pero en este caso usadas para poder comunicar componentes que se encuentren en una subnet privada con Internet, las cuales ofrecen la ventaja de la imposibilidad de acceder desde internet a estos componentes. La estructura que se sigue con estas puertas de enlace se basa en crearlas en una subred pública, y asignar la dirección IP generada a la tabla de enrutamiento de

la subred privada, para que el tráfico saliente a internet se redirija a esta puerta, pase a la subred pública, y de aquí se transmita a la puerta de enlace con internet.

DNS

Cabe destacar la existencia de un nombre de dominio para cada IP privada asignada a cada recurso. Además se puede utilizar servicios de AWS para dar nombres de dominio a las IP públicas así como resolverlos.

4.1.3. Elastic IP

Cuando asignamos una IP pública a una instancia EC2 ubicada en una subred pública, esta IP puede cambiar cuando reiniciamos la instancia, ya que no está reservada. Este cambio puede causar problemas en la aplicación, ya que la dirección IP cambia y, por lo tanto, también cambia la llamada a la máquina. Esto requeriría modificar el código, el backend o las llamadas de los usuarios cada vez que reiniciemos la instancia.

Para solucionar este problema, AWS proporciona la Elastic IP address, que es una dirección IPv4 estática. Esta IP puede asociarse a cualquier recurso que elijamos, como por ejemplo, un servidor EC2, y permanecerá reservada para nuestro uso durante el tiempo que la tengamos asignada. De esta manera, la IP no cambiará cada vez que reiniciemos o movamos la instancia a otro host.

Además, podemos configurar grupos de seguridad para esta IP elástica, de manera que se apliquen a cualquier recurso al que esté asociada.

La Elastic IP es muy útil para realizar tareas de mantenimiento en un servidor. Por ejemplo, podemos cambiar la IP a otro servidor sin perder la conexión con los usuarios, ya que la IP está asociada a un servidor que proporciona el mismo servicio a los usuarios, como una aplicación o un backend.

Es importante tener en cuenta que si asociamos más de una IP elástica a una instancia, se generará un cargo por hora por cada IP adicional asociada. Del mismo modo, si tenemos una IP elástica pero no está asociada a ninguna instancia, también se generará un cargo por hora.

Las IP elásticas son específicas de una región y solo pueden estar asociadas con recursos dentro de esa región, sin posibilidad de salir de ella. Estas direcciones provienen de un pool de direcciones IPv4 proporcionadas por Amazon, y podemos elegir solicitar una IP específica.

4.1.4. Security groups y NACLs

Estos dos elementos actúan como firewalls a la hora de regular las comunicaciones entre un recurso y otro externo, no van asociados a las subnets, sino a los recursos que se crean dentro de esta, como podría ser una máquina virtual. Por una parte tenemos los grupos de seguridad, que vienen definidos en los recursos denegando todo tipo de comunicaciones, por lo que será necesario aplicar una configuración o asociar un grupo de seguridad ya definido, para que se acepte el traspaso de datos a partir de las reglas especificadas. Estos grupos de seguridad actúan como stateful firewall, ya que la regla de seguridad que se aplica en la entrada de datos se da por aplicada en la salida si el firewall detecta que proviene del flujo de entrada aplicado en la regla de entrada. En cuanto a la configuración de Security Groups en AWS, se tratan reglas donde se define el nombre, el id de la regla que se pondrá solo, la versión IP del tráfico que trata la regla, el tipo

de tráfico (HTTP, SSH, etc.), el protocolo (TCP, UDP, etc.) el puerto en el que actúa como firewall el puerto al que deja entrar el paquete si es inbound o salir si es outbound, y la fuente o destino de los datos, según sea regla de entrada o salida.

Por otra parte tenemos los "Network Access Control List", que también actúan como firewall pero del tipo stateless, ya que no siguen en flujo de la información y hay que especificar tanto reglas de entrada como de salida aunque sea para el mismo caso con la misma IP. Si no hay ninguna regla de este tipo creada no sucede nada con las comunicaciones, ya que en este caso se trata de aceptar o denegar casos de comunicaciones entre recursos. Estas reglas se configuran a partir de un número el cual define la prioridad, teniendo mayor prioridad las de menor numeración. Y la configuración que tienen son con los campos: tipo, protocolo, port range y source en inbound destination en outbound, y si permiten o deniegan el tráfico.

4.1.5. Load Balancers

Es uno de los componentes claves en toda infraestructura de la nube, ya que da dos de las ventajas de desplegar infraestructura en la nube: trabajos en paralelo por la replicación y alta disponibilidad. Un load balancer es un componente que se sitúa enfrente de un conjunto de componentes del mismo tipo para realizar dos acciones: dividir las cargas de trabajo en ese conjunto de componentes según la disponibilidad de estos, y asociar una IP al Load Balancer para conectarse con este y no tener caídas de conexión, ya que se envía la solicitud al Load Balancer y llega al componente que esté disponible. Existen 3 tipos de load balancers:

- Classic Load Balancer, el clásico y casi en desuso
- Application Load Balancer: Para peticiones HTTP/HTTPS
- Network Load Balancer: Para peticiones TCP y UDP

En AWS están los Elastic Load Balancer, donde se sitúa un Load Balancer detrás de una VPC, y se crean interfaces de este en todas las zonas habilitadas, para poder distribuir la carga entre instancias de un recurso de diferentes AZ. Además mediante el cross-zone balancing de AWS, si las instancias no son equitativas entre las zonas, se distribuye la carga de forma equivalente entre las zonas. Como última característica, se puede personalizar la conexión con estos load balancers, especificando si han de ser públicos o privados.

4.1.6. VPN

Es un recurso habitual, tanto en datacenters físicos como en la nube, ya que es uno de los métodos de interconexión más utilizados. En este caso se usa para conectar servidores on-premise con recursos en la nube. El traspaso de datos funciona a partir de una puerta VPN en la parte de la nube y una customer gateway en la parte del cliente, y el túnel creado encripta los datos enviados, ya que el traspaso se realiza a través de internet.

4.1.7. Direct Connect

Otra forma de conectar tu servidor on-premise con el servicio en la nube, pero en este caso es un medio físico, no se traspasa por internet la información, por lo que ganas en velocidad, pero los datos no van encriptados.

4.1.8. VPC Peering

Se trata de conectar dos VPC de AWS. Se crea una conexión mediante las IPs de estas, y habrá que añadir la regla para el envío de datos de una VPC a otra en las tablas de enrutamiento. No permite conocer VPC al conectar unas con otras, es decir, si conectamos 3 VPC, la primera no puede pasar por la segunda para llegar a la tercera.

4.1.9. Transit Gateway

Actúa igual que VPC peering, pero se usa para conexiones de una gran cantidad de VPCs, y poder ganar escalabilidad y no crear múltiples conexiones entre VPC cuando crezca la red. Se sitúa en medio de un número de VPC y las conecta a todas, la información pasa a través de la puerta y esta la redirige a la VPC designada.

4.1.10. Private Link

Trata de dar servicio a recursos privados como puede ser un EC2 para que puedan conectarse con servicios como S3 sin tener que usar una conexión a internet total, ni pasar por internet. Se crea un link privado de conexión, también puede usarse para conectar con otras VPCs.

4.1.11. Cloud Front

Amazon CloudFront es un servicio de distribución de contenido que acelera la entrega tanto de contenido estático como dinámico en todo el mundo, utilizando ubicaciones de borde o edge locations. Estas ubicaciones, presentes en todas las regiones de AWS, permiten replicar servicios y reducir las latencias. Cuando un usuario envía una solicitud de un recurso, CloudFront responde desde la edge location más cercana. La arquitectura de este sistema se compone de los siguientes elementos:

El origen del servicio son los recursos que deben ser replicados, como S3, EC2, u otros, así como los datos que contienen. CloudFront envía los datos desde el origen hasta las edge locations. Una vez almacenados en estas ubicaciones, los usuarios pueden recibir las solicitudes directamente, sin necesidad de acceder a la ubicación original. El servicio se configura para recoger los objetos deseados desde el origen y asignarles un nombre de dominio para solicitarlos. Este nombre de dominio se traduce en las edge locations donde residen los datos.

Cuando un usuario realiza una solicitud, CloudFront responde con el contenido si está almacenado en la edge location, utilizando el nombre de dominio. Si la edge location no tiene el dato solicitado, lo obtiene del origen y lo entrega al usuario. Posteriormente, este contenido se almacena en la ubicación para una entrega más rápida en el futuro.

4.1.12. Global Accelerator

Global Accelerator es un servicio diseñado para resolver problemas como la alta latencia al acceder a servicios a través de internet. Utiliza ubicaciones de borde (edge locations) de AWS para establecer una conexión directa y rápida con los servicios.

Cuando se utiliza Global Accelerator, las edge locations establecen una conexión directa con el servicio a través de la red de AWS (AWS Backbone Network), en lugar de utilizar una conexión a través de internet. Esta red dedicada garantiza conexiones mucho

más rápidas que las que se obtendrían a través de un proveedor de servicios de internet convencional.

4.1.13. Route 53

Route 53 es un servicio de DNS (Sistema de Nombres de Dominio) controlado por AWS. Actúa como un registrador de dominios, permitiendo registrar nombres de dominio como, por ejemplo, Kodekloud.com.

Es un servicio global que no está limitado a una región específica de AWS.

Cuando se crea un dominio en Route 53, se crea una "hosted zone"(zona alojada), que se encuentra en 4 servidores dedicados reservados para ese dominio específico.

Route 53 permite tanto registrar nuevos dominios como transferir dominios existentes.

El servicio verifica la disponibilidad de un nombre de dominio y ofrece diferentes opciones con diversos precios. También ofrece la posibilidad de renovar automáticamente el dominio o desactivar esta opción.

En la "hosted zone.ªparecen los servidores DNS proporcionados por AWS.

Una vez creado el dominio, podemos configurar registros DNS, asignándoles una dirección IP, como la de un servidor web, para asociar el nombre de dominio con la página web correspondiente.

Además, Route 53 ofrece la función de alias, que permite asociar registros DNS con otros recursos de AWS, como por ejemplo, un balanceador de carga de Elastic Load Balancing (ELB) o un bucket de Amazon S3. Esto permite configurar fácilmente la resolución de nombres para servicios de AWS sin la necesidad de especificar direcciones IP.

4.1.14. API Gateway

Amazon API Gateway es un servicio que permite centralizar y gestionar múltiples servicios de backend en un único punto de acceso. Se utiliza para la creación, despliegue y gestión de APIs que conectan con servicios de AWS.

Este servicio facilita el control de la API al incorporar un sistema de control de versiones y proporcionar herramientas para el despliegue de la misma.

Además, API Gateway ofrece capacidades de transformación de mensajes tanto en las peticiones como en las respuestas, lo que permite adaptar los datos según sea necesario.

En cuanto a la seguridad, API Gateway ofrece una amplia gama de controles. Permite controlar el acceso al endpoint de la API y asegurar la propia API mediante diferentes mecanismos de seguridad.

También proporciona la capacidad de establecer límites de velocidad (rate limits) sin necesidad de modificar el código del backend, así como políticas de estrangulamiento para evitar cuellos de botella y garantizar el funcionamiento óptimo del sistema.

4.1.15. Componentes utilizados en el caso práctico

Estos componentes son los establecidos en la categoría de networking según el propio Amazon. A la hora de construir la infraestructura que se desplegará en el último punto del trabajo, para realizar una solución a una mejora en un sistema en la nube para un banco, se van a utilizar los siguientes elementos de los anteriormente explicados:

- VPN, incluyendo los siguientes componentes:
  - Subnets
  - Internet Gateway
  - Nat Gateway
  - Routing
- Elastic IP
- Security Groups
- Application Load Balancer
- VPN
- VPC peering

Pero además, se desplegarán recursos que se encuentran en otras categorías de las nombradas anteriormente:

- Almacenamiento
  - Aurora MySQL RDS
    Aurora es una base de datos relacional donde se divide la información por cientos de nodos, y proporciona escalabilidad, replicación mediante un clúster de Aurora y durabilidad mediante controles de salud continuos
- Computación
  - EC2
  - Elastic Beanstalk
- Monitorización
  - CloudWatch

Todas estas herramientas y recursos se encuentran dentro de AWS, y se va a trabajar con ellos en el apartado donde se explicará el modelo de diseño propuesto.

4.2 Proveedores

En este punto se ha adquirido una visión general de la abundancia de recursos y sistemas que proporciona amazon, y la utilidad de varios de ellos.

Tras esto se expone a continuación una tabla comparativa entre 4 de los proveedores más importantes en el mundo de la ingeniería en la nube:

Aspecto	**AWS**	**Azure**	**Google Cloud**	**IBM Cloud**
Computación	EC2, ECS, Lambda	VMs, AKS, Functions, Logic Apps	Compute Engine, GKE, Cloud Functions	VMs, Kubernetes, Cloud Functions
Almacenamiento	S3, EBS, RDS	Blob Storage, Disk Storage, SQL Database	Cloud Storage, Persistent Disk, Cloud SQL	Object Storage, Block Storage, Db2

Aspecto	AWS	Azure	Google Cloud	IBM Cloud
Redes	VPC, CloudFront, Route 53	VNet, CDN, Traffic Manager	VPC, Cloud CDN, Cloud DNS	VPC, CDN, DNS Services
Precios	Pago por uso, Savings Plans	Pago por uso, suscripción, Reserved Instances	Pago por uso, suscripción, Committed Use Contracts	Pago por uso, suscripción, Committed Use Contracts
Disponibilidad Global	25 regiones, 81 zonas	60+ regiones, múltiples zonas	24 regiones, 73 zonas	18 regiones, múltiples zonas
Herramientas de Gestión	AWS Management Console, CloudFormation, CloudWatch	Azure Portal, ARM, Azure Monitor	GCP Console, Deployment Manager, Stackdriver	IBM Cloud Console, Terraform, Monitoring
Seguridad	ISO 27001, SOC 1/2/3, HIPAA	ISO 27001, SOC 1/2/3, HIPAA	ISO 27001, SOC 1/2/3, HIPAA	ISO 27001, SOC 1/2/3, HIPAA
Soporte	Básico, Developer, Business, Enterprise	Básico, Developer, Standard, Professional Direct	Básico, Silver, Gold, Platinum	Básico, Avanzado, Premium
Comunidad y Documentación	Amplia documentación, foros, programas de certificación	Documentación extensa, foros, Microsoft Learn	Documentación extensa, foros, Google Cloud Training	Documentación extensa, foros, IBM Skills Gateway
Empuje empresarial, crecimiento	Usado por empresas desde 2006, continuo crecimiento con enfoque empresarial	Desde 2010. Principalmente para integrar productos de Microsoft, posteriormente productos propios	Nace en 2008 como impulsor de startups y desarrolladores, hasta 2015 no empieza su expansión hacia el mercado empresarial	A partir de 2007 empieza a comercializarse mediante su base sólida de productos, pero no ha evolucionado tanto como sus competidores

Tabla 4.1: Comparativa de servicios en la nube

Como se observa en la tabla, todos los proveedores, o los más grandes como son estos 4, proveen servicios de todo tipo, y a la hora de compararlos pueden parecer que son semejantes o incluso iguales, pero siempre se encuentran diferencias para seleccionar uno u otro según el caso.

En la tabla se muestra como todos ellos tienen capacidad suficientes de cómputo y de creación de redes mediante diferentes tipos de servicios, pero a la hora de mostrar una disponibilidad superior, Amazon con el uso de cloudfront supera al resto, ya que parte con ventaja al disponer de la mayor cantidad de zonas de disponibilidad entre los proveedores de cloud.
Esta es una ventaja de AWS, la gran cantidad de infraestructura y puntos de apoyo a la red que posee en todo el mundo, tiene la mayor capacidad de servidores, con ello ofrece la mayor disponibilidad posible para cualquier cliente de cualquier país, por ello es el proveedor puntero, sobre todo, a la hora de crear soluciones de redes en la nube.

Otra fortaleza que presenta amazon es que dispone de la mayor capacidad de cómputo manejado[18], sin el uso de servidores detrás, mediante su sistema de AWS Lambda,

un sistema semejante a los sistemas functions de Google, Azure e IBM, pero superior en cuanto a potencia y capacidad de integración, ya que corre junto a una multitud de servicios de Amazon, puede desplegarse junto a triggers de otros servicios y tiene posibilidad de usarse en cualquier lenguaje de programación.
Todo esto a su vez tiene una alta complejidad, se muestran muchos recursos, y la curva de aprendizaje no es sencilla, ya que puede ser abrumador encontrarte con tanta posibilidad tanto de recursos como de configuración al iniciarte en el mundo de la nube con AWS[18], y, aunque ayuda bastante con su extensa documentación, es una de las partes débiles de este proveedor.

Por otra parte cabe destacar la potencia de Azure a la hora de integrarse con productos de Microsoft[19], ya que muchas empresas hacen uso de este tipo de productos (Microsoft 365 con todos sus servicios o la propìa ERP de Microsoft) día a día y en multitud de casos y usar Azure como plataforma de cloud ayuda para potenciar el uso de estos mismos.
Además en este 2024 ha desarrollado una muy buena base de sistema de integración de aplicaciones mediante el uso de Logic Apps, usado para crear integraciones de aplicaciones por microservicios mediante APIs, por lo que también destaca en este aspecto sobre el resto de sus competidores.

Google Cloud está en continuo crecimiento, parte de una base menor de servicios ya que no se impulsó significativamente en el mercado empresarial hasta después de hacerlo el resto de competidores, por lo que no abarca tantos servicios como si lo puede hacer Amazon o Azure, pero la decisión de centrarse en las artes más crecientes de este sector como son el Machine Learning o el big data[19], hace que proporcione los sistemas más modernos y potentes del mercado en estos aspectos.

Y por último está descrito en la tabla IBM, que se podría describir como el más tradicional de ellos, ya que es el más antiguo, y, aunque no ha evolucionado tan exponencialmente como AWS o Google Cloud, IBM ofrece con su plataforma de servicios en la nube soluciones robustas, sobre todo, en aspectos de integraciones con sistemas legacy (sistemas en desuso que siguen usándose dentro de una empresa) que son bastante importante para algunas empresas o sectores que deciden no variar su sistemas con el paso de los años.
Y, aunque parezca contradictorio por la tradicionalidad de la base de sus sistemas, IBM destaca en sus servicios de IA y análisis, ya que ha desarrollado este mercado al máximo, pero a costa de dejar atrás todo tipo de avances en cuanto a cantidad de recursos o de sistemas de redes ya que posee la menor cantidad de zonas de disponibilidad de los 4 proveedores comparados.

Por lo tanto, ya se ha visto como AWS destaca por su gran cantidad de servicios, capacidad de cómputo y de opciones a la hora de crear soluciones de networking, y teniendo visto el proyecto ofrecido por el cliente, así como que su base estaba desarrollado en este proveedor, fue por ello que se eligió seguir con Amazon, pero existen otros proveedores que ofrecen servicios de alta calidad, y que tienen sus puntos fuertes para destacar y poder ser utilizados en otros proyectos tal y como se ha expuesto.

CAPÍTULO 5
Infraestructura como código

La infraestructura como código (IaC) tiene como objetivo dar un método para poder trabajar con los componentes provistos por los diferentes proveedores de servicios en la nube a través de código, y evitar el uso de portales web u otros procesos manuales[20] para crear, actualizar y borrar los diferentes recursos utilizados.

Para trabajar con la infraestructura mediante código se hace uso de una serie de archivos de configuración, donde se preparan los recursos a desplegar, configurando todas las opciones que brinda el proveedor para ese servicio.

La infraestructura como código, al basarse en archivos de configuración, tiene una gran ventaja sobre la creación de recurso vía procesos manuales, ya que estos archivos, frecuentemente, se encuentran en repositorios de Git, lo que facilita el control de versiones de todo el código utilizado, por lo que da la opción de controlar las versiones de los despliegues, dando cierta tolerancia a fallos en despliegues y una forma sencilla y rápida de solucionar estos.

Otra de las ventajas que se obtienen a la hora de desplegar infraestructura mediante código es la capacidad de modulación de la implementación, es decir, la capacidad de dividir por grupos de recursos de la nube la implementación, para poder segmentar los despliegues o en la mayoría de los casos, poder utilizar módulos con archivos de configuración para diferentes entornos, y poder reutilizar estos módulos, solo cambiando las variables necesarias en la configuración de los recursos a desplegar.

También es posible, ya que se trata de trabajo con código, la automatización de este. Esta es una de las mayores ventajas que se encuentran a la hora de crear despliegues mediante código, ya que es posible crear configuraciones de recursos o de grupos de estos para que, por ejemplo, cambiando un número en una variable anteriormente configurada, se cree más de una instancia del recurso configurado, lo que facilita la reutilización del código para cualquier entorno dentro de un proyecto, o las actualizaciones según las necesidades que vayan surgiendo. Esta automatización es clave a la hora de desarrollar proyectos mediante IaC, pero es un paso opcional, no es necesario crear despliegues automatizados, aunque sí recomendado.

La infraestructura como código muestra dos enfoques diferentes

5.1 Enfoque declarativo

A través de este enfoque la configuración de los recursos queda al mando del desarrollador, el sistema te da las opciones a configurar, el programador las configura según las opciones dadas y según las necesidades, y este sistema se encarga de realizar el desplie-

gue en el proveedor seleccionado.[20] Un punto fuerte de este enfoque es la capacidad de detallar el estado de los recursos mediante una lista que puede ser interpretada por el programador para resolver dudas o desmontar la infraestructura, además al tener este control de estado, a la hora de aplicar cambios en los recursos, el sistema es capaz de implementarlos sin necesidad de ninguna acción humana.

5.2 Enfoque imperativo

Mediante el enfoque imperativo el sistema proporciona una serie de comandos y su orden de ejecución para realizar la creación de recursos y servicios. Esta solución no vierte un detallado del estado de los recursos, por lo que a la hora de realizar modificaciones en este estado deberán hacerse mediante los comandos, no lo realizará el sistema por sí mismo.[20]

5.3 Terraform

La infraestructura como código puede ser desplegada mediante multitud de herramientas[20]:

- Chef
- Red Hat
- Puppet
- AWS Cloud Formation
- Terraform

Como se ha explicado hay dos enfoques en la IaC, pero, comúnmente ambos enfoques son posibles en las herramientas de manejo de infraestructura como código, por lo que asumiendo que el enfoque declarativo ofrece más ventajas que el imperativo por el control de estados, generalmente este es más utilizado.

Posteriormente, como ya se ha comentado, se mostrará un despliegue en AWS mediante el uso de esta técnica, por lo que se ha usado una herramienta de las anteriormente nombradas para ello. Esta herramienta es Terraform.

Terraform es un software de código abierto, utilizado para la creación y gestión de infraestructura como código desarrollado por HashiCorp, que utiliza un enfoque puramente declarativo a la hora de gestionar la creación de recursos.[21]

Terraform, al ser de código abierto, proporciona la capacidad de incluir multitud de plugins para el manejo del código.

Por otra parte Terraform puede funcionar con cualquier proveedor, tiene la capacidad de controlar despliegues de código en cualquier nube de cualquier proveedor de servicios.

Otra ventaja de Terraform es que incluye un control de versiones en la infraestructura que se despliega, y además, en cada despliegue de esta, se sustituye la anterior, facilitando la detección y corrección de errores.[21]

Terraform permite validar infraestructuras a partir del código creado; realizar un plan antes de la creación para comprobar cómo va a ser el despliegue y prevenir errores; obtener el plan para visualizar el estado de los recursos y cómo serán sus características a la

hora de la creación; desplegar la infraestructura en el proveedor seleccionado; actualizar la infraestructura desplegada anteriormente, o infraestructura existente en la nube y destruir la infraestructura anteriormente desplegada para corregir errores o desplegar otra diferente.

5.3.1. Prerrequisitos sobre el uso de Terraform

Los prerrequisitos para poder realizar cualquier despliegue en Terraform son los siguiente:

1. Debe descargarse en binario de terraform desde la propia página de HashiCorp, e incluir el exe en una variable de entorno, para así poder ejecutar cualquier comando de terraform desde el fichero que sea necesario

2. Antes de poder desplegar en la nube es necesario configurar un proveedor de servicios mediante un archivo provider.tf, así como disponer de un main.tf que contenga la configuración de la infraestructura a desplegar, o en el caso de disponer de módulos, la configuración de las variables necesarias en cada módulo

5.3.2. Comandos Terraform

Una vez realizados los prerrequisitos se debe seguir una serie de comandos para poder realizar un despliegue en un proveedor, y si es necesario, borrar los recursos del despliegue:

- terraform init
  En el fichero donde se encuentre el main.tf y la configuración del proveedor, es necesaria la ejecución de este comando, para inicializar el proyecto, configurando el proveedor seleccionado así como la configuración o los módulos que se desplegarán posteriormente

- terraform plan
  Se hará uso de este comando para comprobar el buen funcionamiento de la estructura y la buena disposición y configuración de los recursos, si el plan muestra errores el despliegue no podrá ejecutarse. Pero, por otra parte, aunque el plan no muestre errores, podemos encontrarnos con otros problemas al intentar el despliegue, por lo que el plan no asegura que todo lo que se vaya a desplegar funcione correctamente.

- terraform apply
  A la hora de hacer un apply, terraform crea un plan de forma anterior al despliegue, y una vez el plan está validado por la herramienta realiza, o intenta realizar, el despliegue en el proveedor escogido.

- terraform destroy
  Es el comando utilizado para el borrado de los recursos y el ambiente de desarrollo, es utilizado en casos donde sea necesario borrar toda la configuración o en entornos de desarrollo o pruebas para no incurrir en gastos.

Aparte de estos comandos, es posible el uso de otros distintos para opciones como formatear el código y que obtenga un formato correcto y común para todo el proyecto mediante "terraform fmt" o el uso de "terraform get" para obtener módulos enteros remotos y configurarlos al gusto del programador.

5.3.3. Terraform junto a AWS

En el caso expuesto en el proyecto se va a utilizar Terraform con AWS, para ello el proveedor se configurará estableciéndolo a AWS, y proporcionando un usuario y una clave. Para ello se utiliza la configuración que se muestra a continuación:

Figura 5.1: Proveedor configurado en Terraform

Como se observa en la imagen 5.1, se proporciona un proveedor amazon así una versión de hashicorp para el propio proveedor, necesario para que Terraform conozca los recursos existentes y los pueda manejar a partir de la configuración creada.

Además se provee un usuario y contraseña y la región donde se desplegarán los recursos, pero estas variables no deberían estar expuestas en ninguna parte del código por la propia seguridad de la cuenta de AWS. Es por ello que para proteger estas credenciales se encuentran varias opciones como crear variables de entorno, usar archivos de configuración, o como es el caso más común y el caso del proyecto, usar lo conocido como Terraform Cloud.

5.3.4. Terraform Cloud

Terraform cloud se configura como se muestra en la parte superior de la imagen 5.1, especificando la organización de la nube de Terraform donde se va a desplegar, el hostname y el workspace.

El funcionamiento de Terraform Cloud se basa en comunicarse con el propio código desplegado en Terraform para asignar las variables en la nube, y, a la hora de desplegar, obtener estos valores para la configuración del código en los archivos en local.

En Terraform Cloud, para proteger los datos sensibles, es posible incluir variables como sensibles, de forma que los datos que contienen la variable son opacos a cualquiera que no sea administrador de la nube, de manera que, a la hora de compartir código con quien sea necesario, las variables en el código por una parte no se verán reflejadas ya que estarán en la nube, y además las variables que representen un valor sensible en ningún momento serán visibles aunque se tenga acceso a la nube de terraform.

A la hora de configurar los recursos, terraform puede manejar los de cualquier tipo de proveedor, de esta manera a partir de nomenclaturas específicas para el recurso requerido se crea la configuración de este.

5.3.5. Ejemplo de configuración Terraform

A continuación se va a ilustrar un ejemplo de una red virtual en AWS, como sería su especificación en Terraform mediante el editor de Visual Studio Code:

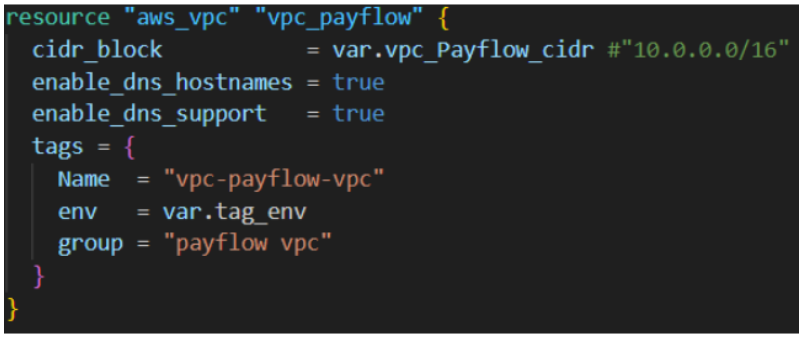

Figura 5.2: Ejemplo de configuración de una VPC en Terraform

Como se observa en la imagen 5.2, el recurso se define a partir de una nomenclatura específica, mediante una estructura común a cualquier proveedor, ya que siempre un recurso se define mediante la palabra "resource", seguida del nombre específico del recurso, y con un nombre propio asignado por parte del desarrollador. Dentro del recurso se especifican características en formato clave = valor, asignando así todas las características necesarias para la creación del recurso.

Los nombres de los recursos se encuentran todos documentados en la página oficial de Terraform Registry [25], en el apartado dedicado a AWS. Esta página ofrece todos los recursos que se pueden crear desde código, así como un ejemplo de la creación, los campos requeridos para la configuración del recurso y los campos que son opcionales.

5.3.6. Estructura de Terraform

Tal y como se ha comentado anteriormente, el desarrollo de infraestructura en terraform puede ser mediante módulos, a continuación se va a realizar una explicación más extensa acerca de este tipo de desarrollo, ya que será el que se va a seguir en el proyecto posteriormente presentado.

Al interactuar con módulos en terraform el fichero donde se encuentran los archivos se divide en varias partes:

- **Fichero Raíz**
  - `README`
    - Donde se explicará la configuración necesaria. Se trata de un manual de uso del código para facilitar su uso y el despliegue para quien no lo haya desarrollado.
  - `.gitignore`
    - Utilizado para, a la hora de desplegar los archivos en git, obviar aquellos archivos como `terraform.tfstate` que se crean a la hora de inicializar el

entorno de terraform y no son necesarios, sólo cuando se vaya a realizar el despliegue.
    - Además, se deberán obviar archivos que contengan configuraciones del método de despliegue para evitar transferir archivos de gran tamaño, lo que da problemas al subirlo a git.

- **Módulos**
  - `Output.tf`
    - Sirve para sacar valores definidos en el módulo como puede ser la ID de una subnet para que, en el main final, se puedan asociar a valores de campos especificados en recursos de módulos diferentes.
  - `Variables.tf`
    - Sirve para especificar las variables que se van a definir posteriormente. Se crean las variables y se les da un tipo.
  - `Main.tf`
    - Es donde se crea la especificación de los recursos, usando variables en los campos dentro de los recursos para que se modifique el valor según el entorno de desarrollo especificado.
  - `*.tf`
    - Son ficheros de terraform diferentes a `main.tf`, utilizados si se quiere dividir un módulo en grupos de recursos. Un ejemplo claro es cuando en `networking` se configura en `main.tf` la red y subredes, y en otro fichero como podría ser `bastion.tf` se configura una entrada a la red. El `*` actúa de wildcard, es decir, es posible asignar cualquier nombre.

- **Entornos de Desarrollo (dev, pre, pro...)**
  - Esto muestra los entornos de desarrollo del proyecto, que muestra cómo el proyecto evoluciona según el entorno. Por ejemplo, `dev` significa *development* o desarrollo en español, ya que en este apartado se harán las configuraciones necesarias para el momento en el que el proyecto está en desarrollo y no está preparado para la producción.
  - `dev` suele ser un entorno donde se realizan pruebas acerca del funcionamiento de la estructura, pero abaratando costes ya que no es un entorno que se vaya a utilizar realmente. Así como este entorno se usa para ese caso, el resto van evolucionando hasta llegar al entorno de producción donde se desplegará la solución final.
  - El uso de esta diferenciación es clave, ya que, a partir de un código generado en los módulos se pueden especificar los valores de las variables creadas según sea necesario, y de esta forma, crear configuraciones diferentes para diferentes entornos a partir de un mismo código.
  - En los entornos para seguir unas buenas prácticas, un prototipo de estructura a seguir mediante los ficheros es la siguiente:
    - `variables.tf`
      - Fichero que contendrá todas las variables necesarias que se encuentran especificadas en los módulos. Se asignan las variables sin darle ningún valor ni tipo, ya que el tipo se asigna a partir del tipo creado en el fichero de variables de los módulos cargados en el entorno.
    - `terraform.tfvars`

        - ◇ Este es el fichero donde se definen los valores de las variables descritas en el archivo anterior. Se asignan con formato clave = valor. En este archivo es necesario definir cualquier valor para todas las variables ya que, sino, a la hora del despliegue se pedirá establecer por la consola un valor.
        - ◇ A la hora de crear infraestructuras dinámicas, donde se pueda ejecutar más de una vez el mismo módulo (un ejemplo sería desplegar 3 bastion a partir de un módulo `bastion.tf`), las variables se asignarán como vacías en `terraform.tfvars`, y a continuación en un `main.tf` se les dará el valor diferente para cada una de las creaciones del mismo módulo.
    - `provider.tf`
        - ◇ A partir de este fichero se configura la llamada al proveedor que se va a utilizar, y, si es el caso, la comunicación con Terraform Cloud. Un ejemplo de este fichero será el visto en la imagen x.
    - `main.tf`
        - ◇ En el archivo `main` se configuran los módulos a utilizar a través de un código del tipo `source`. Se especifica de dónde provienen los datos de los módulos estableciendo la ruta desde donde se encuentra el `main.tf`, y una vez hecha la conexión se definen las variables.
        - ◇ En la definición de las variables se requiere el uso de todas las especificadas en `variables.tf` de cada módulo, y en caso de que estén definidas en `terraform.tfvars` se hace una llamada a la variable, pero si están vacías ya que se asignan dinámicamente por el uso de más de una instancia del mismo módulo, se definen directamente en la especificación de la variable en el archivo `main`.
        - ◇ Es aquí donde se podrá hacer llamadas a variables especificadas en los outputs de los módulos, para asignar valores creados de forma dinámica en el despliegue en módulos diferentes a donde se especifica el recurso.

- **Módulo Global**

    - Este módulo contendrá la misma estructura de los tres ficheros `output`, `main` y `variables`, pero se usará de manera global a todos los entornos.
    - En este fichero global se definirán características que se comparten para todos los entornos como pueden ser variables de tags de grupos, o la misma VPC que se vaya a usar si no varía según entorno.

5.3.7. Nomenclatura

A la hora de nombrar los recursos que proporciona Terraform existe un convenio sobre las buenas formas de los nombres, por lo que hay que seguir unas normas para tratar de crear un código con la mayor calidad posible.

Algunos de estas normas, o las más importantes, son las siguientes:[24]

1. En los nombres de los recursos, fuentes de datos, variables, salidas u outputs usar guiones bajos "_".
2. Crear los nombres preferiblemente siempre en minúsculas.
3. Usar sustantivos singulares para los nombres

4. Para los nombres dentro de los recursos, es decir, los nombres como tal que se verá en el despliegue, se utilizará guiones ".
5. Incluir count o for each al principio en el recurso, como primer argumento.
6. Siempre que sea posible incluir tags, y si es necesario incluir depends on para mostrar el orden de creación de los recursos.
7. Especificar los tipos en las variables

5.4 Visual Studio Code

Terraform al ser un software mediante el cuál definir y desplegar infraestructuras, se hace uso de un editor de código, en el caso del proyecto, se hace uso de Visual Studio. Visual Studio Code es un editor de código fuente desarrollado por Microsoft, conocido por su ligereza, extensibilidad y potente funcionalidad.[22] Es ampliamente utilizado para el desarrollo de software y la gestión de configuraciones en diversos lenguajes y herramientas, incluyendo Terraform. Visual Studio soporta una gran cantidad de lenguajes y permite el uso de extensiones con él, lo que facilita la edición de código desde la plataforma.

Para configurar Visual Studio con Terraform es necesaria su instalación en el sistema mediante un archivo ejecutable, y tras ello con una simple instalación de una extensión de HashiCorp Terraform es posible editar código de este tipo, y obtener ayuda a la hora de desarrollar los recursos.

Visual Studio proporciona terminales de varios tipos, de forma que es posible ejecutar los comandos de Terraform anteriormente expuestos desde el editor de código directamente, facilitando así el despliegue en los proveedores de la nube, ya que además existen extensiones de conexión con la mayoría de proveedores y edición de recursos de estos en este editor de código.

5.4.1. Plugins

Al utilizar Terraform en Visual Studio Code (VS Code), es posible aprovechar diversos plugins que facilitan los despliegues mediante Infraestructura como Código (IaC). Existen extensiones tanto para Terraform como para otros sistemas de IaC, muchas de las cuales son compatibles con la mayoría de proveedores de servicios en la nube.

A continuación se presenta una lista de plugins y herramientas complementarias que se han utilizado en el proyecto para lograr distintos objetivos, como validar las buenas prácticas de IaC, comparar el despliegue en la nube con el definido en el código, y observar el coste de los recursos utilizados en Terraform.

Driftctl

GitHub - `snyk/driftctl`: Detecta, rastrea y alerta sobre desviaciones en la infraestructura en comparación con la infraestructura definida en IaC. Esta herramienta ofrece las siguientes ventajas:

- Escaneo de proveedores de nube y recursos para detectar desviaciones con el código IaC.
- Análisis de diferencias y alertas sobre desviaciones y recursos no deseados.

- Permite ignorar recursos específicos.
- Soporte para múltiples formatos de salida.
- Gestión de conformidad, comprobando el cumplimiento de los estándares de seguridad y políticas establecidas.
- Aplicación de correcciones a desviaciones no establecidas.

Forma de instalación:

- **En macOS:**

```
$ curl -L https://github.com/snyk/driftctl/releases/latest/download/
    driftctl_darwin_amd64 -o driftctl
$ chmod +x driftctl
$ sudo mv driftctl /usr/local/bin/
```

- Para comparar la infraestructura entre el código y lo desplegado, es necesario instalar AWS CLI y configurar las credenciales de AWS en el sistema mediante 'aws configure'.

Infracost

Shift FinOps Left With Infracost: Esta herramienta permite estimar el gasto de la infraestructura antes de desplegarla.

- Estimación de costos al cambiar el código.
- Flujos de aprobación basados en límites presupuestarios establecidos.
- Comparación de cambios de código con las mejores prácticas y recomendaciones para opciones más económicas.
- Integración directa en flujos de trabajo de GitHub y GitLab, y con Jira para gestión de productos.
- Exportación diaria de información para integrarse en paneles de control de FinOps y Business Intelligence.

Opciones de precios:

- **Gratis:** Para ingenieros individuales, incluye soporte para desglose y comparación de costos, integración con GitHub y GitLab, y API de precios en la nube.
- **500€/mes:** Para 10 ingenieros, incluye panel de control de costos, política de etiquetado, política de FinOps, comprobaciones automáticas de presupuestos y flujos de trabajo de aprobación, integración con Jira, informes detallados, auditorías, y precios personalizados.

Forma de instalación:

- Instala la extensión de Infracost en VS Code.
- Conecta la extensión a tu cuenta de Infracost.

- (Opcional) Configura la conexión a CI/CD para visualizar costos en los pull requests.
- Crea un archivo de configuración `infracost.yaml` para detectar los costos del proyecto.
- Descarga el ejecutable de Infracost.
- Asigna la ruta del ejecutable en el PATH para utilizar los comandos de Infracost.

OPA

Open Policy Agent (OPA): Herramienta que compara los planes de Terraform con políticas predefinidas en lenguaje `.rego` para validar la conformidad.

Forma de instalación:

- Descarga el ejecutable de OPA correspondiente a tu sistema operativo.
- Asigna la ruta del ejecutable en el PATH.
- (Opcional) Instala la extensión de OPA en VS Code para ayuda en la redacción de políticas.

Uso:

- Ejecuta Terraform y guarda el plan:

```
$ terraform init
$ terraform plan --out plan.binary
```

- Convierte el plan a JSON:

```
$ terraform show -json plan.binary > plan.json
```

- Redacta y aplica la política utilizando OPA:

```
$ opa exec --decision terraform/analysis/authz --bundle policy/ plan.
    json
```

Terrascan

Terrascan: Herramienta para detectar violaciones de cumplimiento y seguridad en la infraestructura como código. Trabaja en conjunto con OPA para crear políticas personalizadas.

- Escaneo de IaC en busca de configuraciones incorrectas.
- Supervisión de la infraestructura en la nube para detectar cambios que introduzcan desviaciones.
- Detección de vulnerabilidades y violaciones de cumplimiento.
- Ejecución local o integración con CI/CD.

Forma de instalación:

- **En macOS mediante ejecutable:**

```
$ curl -L "$(curl -s https://api.github.com/repos/tenable/terrascan/
    releases/latest | grep -o -E "https://.+?_Darwin_x86_64.tar.gz")"
    > terrascan.tar.gz
$ tar -xf terrascan.tar.gz terrascan && rm terrascan.tar.gz
$ install terrascan /usr/local/bin && rm terrascan
$ terrascan scan
```

- **Instalación con Homebrew:**

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew
    /install/HEAD/install.sh)"
$ brew install terrascan
$ terrascan scan
```

Políticas: Se pueden crear políticas usando Rego, y hay repositorios con políticas predefinidas y personalizables.

Conclusión: Tras haber definido los componentes de AWS, explicado en detalle los aspectos relacionados con networking y seleccionado los recursos a utilizar, se procederá a describir las buenas prácticas sobre el despliegue de networking en AWS

CAPÍTULO 6
Buenas Prácticas en Networking para AWS

A la hora de crear infraestructura en AWS o en cualquier proveedor cloud, no existe un convenio estricto de pasos a seguir o acciones siempre necesarias, ya que cada caso es diferente y requiere recursos o condiciones de los servicios distintas. Por esta razón, no se puede llegar a un consenso exacto de cómo realizar siempre una red en la nube. Sin embargo, existen prácticas comunes o "buenas prácticas" que ayudan a crear sistemas seguros, eficientes y altamente disponibles. A continuación, se detallan estas recomendaciones a seguir:

6.1 Diseño de la Red

- **Planificación de VPCs y Subnets:** Es fundamental dividir la red en varias VPCs (Virtual Private Clouds) para segmentar los recursos de manera lógica, mejorando así la seguridad y la gestión. Dentro de cada VPC, se deben crear subnets públicas y privadas según las necesidades de acceso a Internet de los recursos.
  - **Subnets públicas:** Estas subnets se utilizan para recursos que necesitan acceso directo a Internet, como servidores web en caso de que sean necesarios.
  - **Subnets privadas:** Estas subnets se destinan a recursos internos que no requieren exposición directa a Internet, como bases de datos, servidores de aplicaciones o máquinas de cómputo para servidores backend, APIs o instancias de control y administración.
- **Zonas de Disponibilidad:** Para aumentar la resiliencia y disponibilidad, es recomendable distribuir las subnets a través de múltiples zonas de disponibilidad (AZs). [23]
- **Gateways y NAT:** La utilización de Internet Gateways permite que los recursos en subnets públicas accedan a Internet. Además, los NAT Gateways permiten que los recursos en subnets privadas tengan acceso saliente a Internet sin ser accesibles desde el exterior. [23]

6.2 Seguridad de la Red

- **Grupos de Seguridad y Listas de Control de Acceso (ACLs):** Es esencial configurar grupos de seguridad y ACLs de red para controlar el tráfico entrante y saliente. [23]

  - **Grupos de Seguridad:** Actúan como firewalls virtuales a nivel de instancia, permitiendo definir reglas detalladas de tráfico.
  - **ACLs:** Operan a nivel de subnet, proporcionando una capa adicional de control sobre el tráfico de red.

- **VPNs y Peering de VPC:** Para la comunicación segura con redes locales, se deben utilizar conexiones VPN. El peering de VPCs es útil para la comunicación eficiente y segura entre diferentes VPCs.

6.3 Monitorización y Gestión

- **Logs y Monitorización:** Habilitar el registro de logs es crucial para monitorear el tráfico de red y los accesos. AWS CloudWatch y VPC Flow Logs son herramientas valiosas para obtener visibilidad sobre el tráfico de red y detectar posibles problemas de seguridad o rendimiento. [23]

- **Automatización y Terraform:** La utilización de herramientas de infraestructura como código (IaC) como Terraform es esencial para automatizar el despliegue y la gestión de la infraestructura de red. Esto garantiza consistencia, facilita la reproducción de entornos y permite el control de versiones.

- **Escaneo de Seguridad:** Implementar herramientas como Terrascan permite detectar violaciones de seguridad y cumplimiento en la infraestructura como código antes de desplegarla, mitigando riesgos y asegurando el cumplimiento de las mejores prácticas de seguridad.

6.4 Mejores Prácticas Específicas de AWS

- **Identidad y Acceso:** La gestión segura del acceso a los recursos de AWS a través de AWS IAM (Identity and Access Management) es crucial. Se deben definir políticas detalladas que sigan el principio de privilegios mínimos. [23]

- **Sistemas de Protección de Datos:** Implementar sistemas de encriptación para los datos en tránsito y en reposo. Utilizar AWS KMS (Key Management Service) para gestionar claves de encriptación de manera segura o AWS Certificate Manager para validar certificados por AWS o para gestionar certificados propios.

La implementación de estas buenas prácticas mejora la seguridad, eficiencia y gestión de la infraestructura de red en AWS. Siguiendo estas recomendaciones, se asegura que la infraestructura no solo cumple con los estándares de la industria, sino que también es escalable y fácil de gestionar a medida que crecen las necesidades del proyecto.

CAPÍTULO 7
Caso práctico

El caso práctico de este proyecto trata de recopilar todos los aspectos explicados de forma práctica, llevar la teoría explicada anteriormente en cada uno de los apartados a un uso real y práctico, y explicar todo el proceso.

Para realizar cualquier proyecto se realizan unas fases previas de análisis de los requisitos, una fase de producción técnica y una última fase de pruebas de rendimiento. Este proyecto se ha realizado de la misma forma, ya que se trata en esencia de un caso de uso empresarial, por lo que se van a detallar todos los pasos dados durante la realización del caso técnico.

El proyecto fue desarrollado durante mi estancia en prácticas en la empresa DISID, se realizó para un cliente que solicitó una consultoría sobre su infraestructura en cloud y necesitaba un principio de solución para problemas que no sabía acotar desde producción, por lo que se ofreció una solución para TEST escalable a producción en un futuro.

7.1 Estructura original a mejorar

Este despliegue surge tras la necesidad de una mejora de una infraestructura bancaria. Tras un análisis de la estructura utilizada, surgieron una serie de problemas críticos para el negocio.

Antes de actualizar la infraestructura tras la realización del proyecto, la estructura bancaria estaba desarrollada en AWS, mediante un método de creación de los recursos directamente desde el portal de Amazon.

Para poder realizar el análisis, se obtuvo antes la estructura existente desplegada por la empresa bancaria. En el caso del proyecto, se va a mostrar cómo era la estructura anterior a la propuesta, y se va a ahondar más en la parte de networking, ya que es donde se produjeron los cambios. Esta explicación es necesaria para, a la hora de explicar las fallas del despliegue anterior al propuesto, tener un contexto de dónde se obtienen estos problemas.

La infraestructura original proporcionada por el cliente se representa mediante la figura 7.1, diseñada en draw.io, herramienta para creación de diagramas.

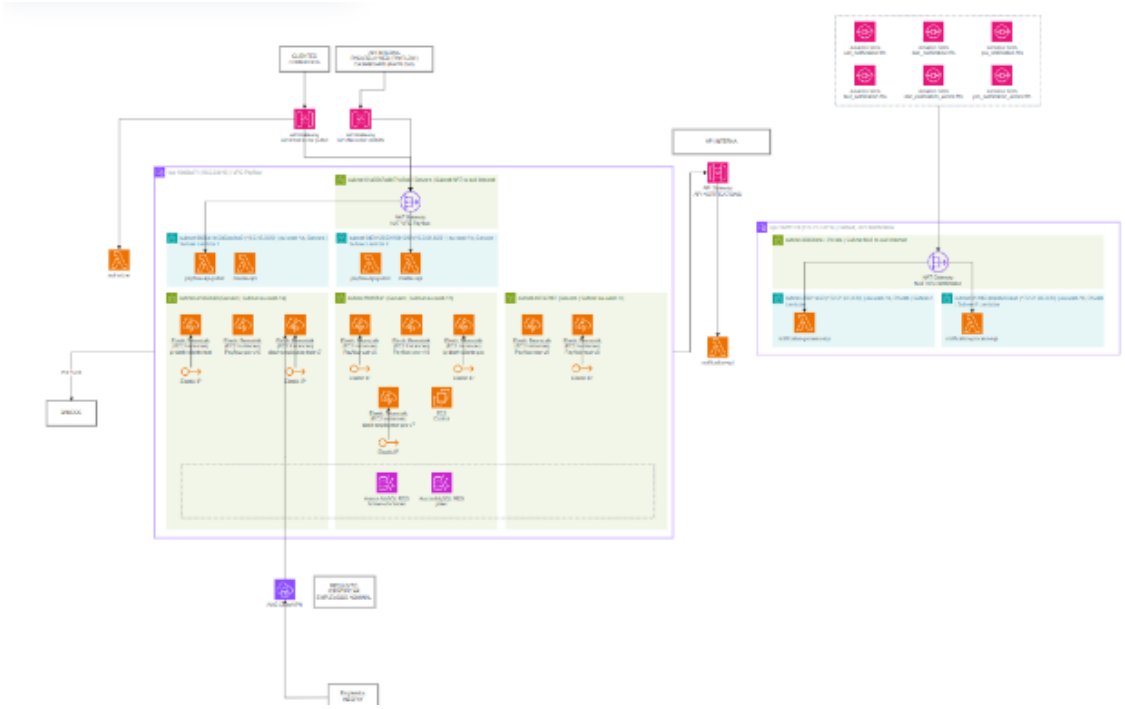

Figura 7.1: Arquitectura original del cliente

Se trataba de 3 VPC, pero en el diagrama solo salen representadas 2 que son las que tenían todos los recursos dentro de sí. La red restante solo contenía recursos relacionados a la seguridad anti-malware y, desde el principio del proyecto, fue desechada por las siguientes razones:

- La protección relacionada con el networking iba a crearse de forma intrínseca en los recursos de las VPC, ya que para proteger estos basta con crear reglas de firewall suficientes en la red.
- La protección relacionada con apps, bases de datos y otros recursos externos a networking sería creada posteriormente de forma externa a una VPC de Amazon, ya que el objetivo de este proyecto abarcaba la creación de la red de soporte al sistema final. Además, no se iba a usar en producción, por lo que no era necesaria esta protección.

Por todo ello, desde el cliente solo llegó esta información, la relevante a nuestro proyecto.

En el diagrama original 7.1, estas dos VPC tenían los nombres *"VPC Payflow"* y *"VPC Notification"*. La primera de ellas contenía los recursos suficientes para servir un flujo entero de pagos, desde que un cliente se conecte hasta realizar una transacción o visualizar su dinero en la cuenta. Y la segunda se basaba en transmitir comunicaciones con los clientes que realizaban operaciones en el banco, todo manejado mediante notificaciones por SMS o Email. Para poder tener esta funcionalidad implementada era necesario el uso de una gran cantidad de recursos de AWS, pero además una buena infraestructura donde alojar todas estas operaciones.

A la hora de realizar la explicación acerca de la estructura original proporcionada se va a usar como base una tabla con la definición de los componentes existentes en ambas VPC:

Aspecto	VPC Payflow	VPC Notification
Subnets	4 subnets públicas, 2 subnets privadas	1 subnet pública, 2 subnets privadas
Internet Gateway	1 Internet Gateway	1 Internet Gateway
Nat Gateway	1 Nat Gateway	1 Nat Gateway
VPN	1 Client VPN	No existente
Logs	No existe recopilación de logs para ningún elemento	No existe recopilación de logs para ningún elemento
Seguridad	Entrada por VPN a una de las subnets públicas, el resto sin seguridad adicional	Sin seguridad adicional
Zonas habilitadas	3 Zonas habilitadas de eu-west usadas en 3 subnets públicas, la cuarta se repite con eu-west-1a	Eu-west-1a y eu-west-1b para las privadas, Eu-west-1a para la subnet pública, Eu-west-1a y eu-west-1b para las privadas

Tabla 7.1: Definición de los componentes existentes en ambas VPC

La creación de la red original comprende los elementos provistos en la tabla 7.1, pero dentro de las subredes se encuentran otra cantidad amplia de componentes relacionados directamente con la funcionalidad del banco. Estas herramientas se dividen en 3 grupos:

- **Máquinas de cómputo EC2 elastic beanstalk** usadas como servidores web o servidores para uso de otros componentes. Se encuentran solo en 3 subnets públicas de VPC Payflow, y la mayoría de ellas consta de una IP elástica.

- **Funciones sin servidor lambda**, usadas para funciones concretas de la aplicación bancaria, y, sobre todo, para conectar funcionalidades de AWS, por ejemplo, recibir mensajes de una cola SQS y realizar acciones en el banco. Se encuentran en las subnets privadas de ambas VPC.

- **Bases de datos**, donde se encuentra información sensible de los clientes y del banco, replicadas en las zonas habilitadas de las subnets privadas de VPC Payflow.

Por tanto, la foto de la estructura queda con 2 VPC relevantes sin el uso de logs o seguridad extra a la provista por AWS en la creación de los recursos, con subnets públicas y privadas, cuyos recursos se encuentran repartidos por estas subnets, comunicadas con internet mediante las puertas de internet y puertas NAT para las subnets privadas, distribuyendo los recursos como bases de datos o máquinas de servidores a lo largo de las subnets públicas, así como las funciones en las subredes privadas.

7.2 Problemáticas de la infraestructura a resolver

El análisis de la infraestructura original reveló varias áreas que requieren mejoras significativas para alinearse con las mejores prácticas de AWS y asegurar una operación más eficiente y segura. A continuación, se detallan los principales problemas identificados y las razones por las que deben ser abordados.

Tras el análisis de la infraestructura original proporcionada se revelaron varias áreas con puntos débiles significantes los cuales se necesitaban mejorar para seguir las mejores prácticas de redes en la nube y de AWS específicamente, asegurando así una operación más eficiente y segura. A continuación, se detallan los principales problemas encontrados y las razones por las que deben ser abordados:

1. **Seguridad inadecuada en acceso a recursos:**
   - La infraestructura proporcionada permitía la entrada a una subnet pública de una VPC mediante una VPN para empleados del cliente, pero para el resto de subnets públicas no había implementadas mayores formas de seguridad que las implementadas y gestionadas por AWS, como las reglas de firewall predefinidas por el proveedor.
     Por esto mismo, al tener solo algunas reglas de firewall predefinidas y algo de control sobre las acciones a realizar en los recursos mediante roles de IAM predefinidos, si un atacante llega a acceder a la red (algo que puede pasar por el poco cuidado con las reglas de red) podrá realizar multitud de operaciones con otros recursos, ya que por ejemplo, las bases de datos no constaban de roles de IAM y se encontraban en subredes públicas poco seguras.
   - Por otra parte en la antigua arquitectura se usa SSH para establecer una conexión a la Ec2 Bastión y desde ahí se conectaban a otras máquinas virtuales en caso de requerirlo, de esta forma el realizar auditorías sobre algunas máquinas virtuales y procesos se vuelve muy complicado ya que solo hay acceso a métricas y logs de la máquina bastión por el acceso mediante ssh.
2. **Falta de recopilación de Logs**
   - No hay recopilación de VPC logs, por lo que no se obtendrán logs de la información que pase a través de las subredes de la VPC, ni entrante ni saliente de toda la VPC. Si que se encuentran algunos logs en las instancias dentro de las máquinas de cómputo ec2 elastic beanstalk, pero debía de implementarse los logs para toda la red dentro de la VPC ya que son necesarios para detectar problemas de rendimientos, incidencias en la seguridad, cumplimiento de normativas y otros tantos aspectos cruciales en una infraestructura de red.
3. **Uso Inadecuado de Subnets Públicas y Privadas**
   - En la infraestructura original existe una multitud de subnets públicas con máquina de cómputo que no deberían estar abiertas en una subnet pública como es el caso de la EC2 de control, o como es el caso de algunas EC2 beanstalk que no son servidores web. Esto mismo sucede con las bases de datos, se encuentran en una subnet pública. Esta forma de alojar los recursos va en contra de las buenas prácticas ofrecidas tanto por la nube en general como por AWS directamente, ya que alojar recursos con información sensible como puede ser la máquina de administración y control o las bases de datos aumenta el riesgo de exposición a ataques y accesos no autorizados.
4. **Falta de Redundancia y Resiliencia**

- Las subnets están divididas de manera no uniforme a lo largo de las 2 VPC, teniendo en la primera 4 públicas por lo que se repite una subnet en una zona de disponibilidad y solo 2 privadas desaprovechando la existencia de la tercera zona de disponibilidad en este caso. De la misma manera en la VPC Notification solo se encuentra 1 pública por lo que no existía ningún tipo de replicación y 2 privadas, sin llegar a utilizar todos los recursos que ofrece AWS. Esto produce dos efectos negativos:
  - Por una parte la repetición de una subnet en una zona habilitada supone menos tolerancia a fallos y resiliencia por parte de los recursos que estén en esas subnets, ya que si deja de funcionar la zona dejan de funcionar dos subnets enteras sin replicación.
  - Por otra parte no están bien utilizadas las zonas habilitadas, ya que los recursos de las subnets privadas no aprovechan la replicación en todas las zonas habilitadas, quitando así parte de una de las ventajas que ofrece la nube, la replicación, disponibilidad y eficiencia de los recursos por poder situarse en varias zonas de una región.

5. **Implementación Deficiente de Servicios de Protección y Anti-Malware**

   - Al tener distribuidos los recursos anti-malware y de protección en una VPC aparte, muchos recursos no contaban con seguridad propia, lo que podía causar desastres en caso de recibir algún ataque en un recurso desprotegido.

6. **Falta de conectividad y comunicación**

   - Las VPC creadas no tenían ningún tipo de comunicación entre sí, se basaban en mostrar recursos aislados dentro de la VPC, comunicándose mediante otros sistemas externos si era necesario obtener algo externo a la red interna de AWS, de manera que entre recursos de VPC Payflow y VPC Notification, no existía comunicación ninguna. Esto es negativo ya que siempre debe existir una vía de comunicación segura entre las redes privadas de AWS siguiendo las buenas prácticas de creación de una red en la nube.

7. **Falta de automatización**

   - Todos los recursos estaban desplegados desde la propia consola de AWS, y solo hacían uso de 1 entorno, donde dividían el uso de los recursos (test, dev, pro) según el nombre de los recursos, pero sin cambiar de entorno. Esto genera dos problemas; por una parte el no poseer más de un entorno puede generar problemas sobre el uso de recursos, por errores humanos puede ocurrir que un recurso realice funcionalidades de un entorno que no debería lo que puede llegar a afectar en los recursos de producción y con esto a los clientes; y por otra parte el usar la consola de AWS para el despliegue de los recursos genera una falta de automatización lo que no sigue las buenas prácticas, ya que no hay forma de reutilizar recursos o realizar pruebas sin tener que depender de la creación de estos desde cero.

Abordar estos problemas no solo mejorará la seguridad y eficiencia de la infraestructura, sino que también asegurará que la solución sea escalable y sostenible a largo plazo. La implementación de estas mejoras permitirá a la organización aprovechar al máximo las capacidades de AWS, garantizando un entorno más seguro, robusto y preparado para futuras expansiones.

7.3 Plan de acción

Tras realizar el análisis de los defectos existentes en la infraestructura original de la plataforma bancaria se procedió a realizar un plan de acción donde configurar las tareas para la primera fase del desarrollo del proyecto, donde se iba a proceder a realizar la nueva infraestructura de nueva, desde una perspectiva de los puntos de mejora. Para esto se hizo un estudio y uso de la metodología ágil con uso de Scrum y Kanban mediante Jira, tal y como se explica en el apartado 3.3.
Se obtuvo un desglose de las tareas a realizar, de las horas a ejercer por tarea, así como una descripción en cada tarea donde se mostraba que había que realizar.

De esta manera lo que se buscaba realizar de forma básica (a falta de peticiones extra durante el desarrollo por parte del cliente) era lo mostrado en la tabla 7.2. En la tabla se muestra la tarea padre o tarea épica de esta fase del proyecto, que es la arquitectura de red, junto a tareas dentro de esta épica, que a su vez se distribuyen en subtareas con horas asignadas en cada una. Se muestran también los entornos de actuación para poder tener una automatización de los recursos, reutilización y entorno de pruebas donde poder desplegar antes de enviar a producción. Además se muestra la estimación de horas, que se alargó una vez comenzado el proyecto. Estas tareas y subtareas pertenecen al primer sprint del proyecto, que es el único considerado en este trabajo, ya que los sprints siguientes están relacionados con el desarrollo de aplicaciones dentro de la infraestructura, lo cual no se aborda en este proyecto.

Épica	Tareas	Entornos	Estimación horas
Arquitectura	Desarrollo y despliegue de red en los diferentes entornos (+VPC Flow Logs)	TEST, SAN	32
Arquitectura	Configuración de peering, VPN	TEST, SAN	24
Arquitectura	Configuración de roles, políticas y grupos de seguridad	TEST, SAN	8

Tabla 7.2: Desglose de tareas y estimación de horas para la primera fase del proyecto

Mediante esta definición base se hizo un desglose de las subtareas a realizar por cada tarea así como los objetivos finales una vez acabado el sprint 1.

7.3.1. Desarrollo y despliegue de red en los diferentes entornos

Tareas

- **Diseño de la red VPC (Virtual Private Cloud):**
  - Crear una estrategia de diseño de subredes que apoye la separación de recursos y el aislamiento de entornos y aplicativos junto a las organizaciones establecidas.
  - Planificar regiones y zonas para la alta disponibilidad y la tolerancia a fallos.
  - Definir el espacio de direcciones IP y rutas así como la privatización de los recursos adecuados.
- **Desarrollo de la infraestructura como código con Terraform:**

  - Escribir los módulos de Terraform para la creación de VPC, subredes, gateways de Internet, y demás recursos de red.
  - Implementar las prácticas de IaC (Infraestructura como Código) para el versionado y la reutilización del código.
  - Probar el código en un entorno no productivo (TEST).

- **Implementación de VPC Flow Logs (Terraform):**
  - Configurar VPC Flow Logs para capturar información sobre el tráfico de la red.
  - Definir los roles y políticas de IAM necesarias para permitir la creación y gestión de Flow Logs.
  - Configurar el destino de los logs (Amazon S3 o CloudWatch Logs).

Objetivos y propósitos

- **Seguridad Mejorada:** Una VPC bien diseñada protege los recursos de AWS del acceso no autorizado.
- **Automatización y Estandarización:** Terraform permite implementar infraestructura como código, lo que facilita la gestión y reduce los errores humanos.
- **Visibilidad y Troubleshooting:** Con VPC Flow Logs, se puede obtener visibilidad detallada del tráfico de la red, lo que facilita la detección de problemas y la respuesta a incidentes de seguridad.

7.3.2. Configuración Peering, VPN client y bastion EC2

Tareas

- **Establecimiento de Peering de VPC:**
  - Analizar los requisitos de conectividad entre las VPCs para configurar el peering.
  - Crear y configurar las solicitudes de peering y las rutas necesarias en las tablas de rutas de las VPCs involucradas.
- **Configuración de la conexión VPN:**
  - Diseñar y configurar la conexión VPN para permitir la comunicación segura entre la infraestructura de AWS y las ubicaciones on-premise o diferentes nubes.
  - Implementar el túnel VPN y configurar los parámetros de cifrado y el intercambio de claves.
  - Realizar pruebas para asegurar la estabilidad y la seguridad de la conexión VPN.
- **Configuración del bastion EC2 y acceso por SSM:**
  - Configurar la máquina EC2.
  - Crear los puntos de conexión por session manager de AWS.
  - Realizar pruebas de entrada a la máquina EC2.

Objetivos y propósitos

- **Interconectividad:** Permite a diferentes partes de la infraestructura del cliente comunicarse de manera segura y confiable.
- **Acceso Remoto Seguro:** Las conexiones VPN cifradas protegen los datos sensibles del cliente cuando se accede a la nube desde ubicaciones remotas.
- **Flexibilidad y Escalabilidad:** Facilita la expansión de la red y la integración con otros servicios sin comprometer la seguridad.
- **Seguridad y control de sesiones:** El uso del EC2 Bastion mediante session manager mejora la seguridad de la conexión al no abrir puertos y además se manejan mejores auditorías ya que controla todas las sesiones creadas en la máquina.

7.3.3. Configuración de roles, políticas y grupos de seguridad

Tareas

- **Análisis de requisitos de seguridad y cumplimiento:**
  - Identificar los requerimientos de seguridad de la organización que deben ser aplicados.
- **Desarrollo de seguridad**
  - Crear, durante la creación de los recursos de networking, toda la seguridad necesaria en estos mediante grupos de seguridad, listas de control de acceso y roles.

Objetivos y propósitos

- **Control de acceso a los recursos** mediante los roles creados y asignados para cada uno de los recursos.
- **Seguridad de comunicación entre recursos** por las tablas de enrutamiento y las políticas de roles.
- **Mitigar ataques de forma sencilla** disminuyendo las posibilidades de comunicación al máximo entre los recursos, aislando las partes críticas del sistema de los accesos a la red.

Con esto se define un plan de acción sobre las necesidades del proyecto, cómo actuar, qué tareas realizar y una estimación de tiempos, así como los objetivos a cumplir una vez realizada cada una de las tareas.

Cabe destacar que la primera y segunda tarea si que van en orden de ejecución, pero la tercera de ellas se produce entre la realización de las otras, ya que los componentes que se configuran en esta última tarea deben crearse a la vez que se configuran los componentes de las otras tareas.

7.4 Implementación solución

Con el análisis de la infraestructura original realizado, y un plan de acción predefinido se da paso al inicio del proyecto, a la implementación de la nueva infraestructura. Este proceso comienza con la creación en AWS de una zona de lanzamiento:

Una Landing Zone es un entorno correctamente diseñado con varias cuentas, además de escalable y seguro. Se establece como punto de partida para la implementación de cargas de trabajo y aplicaciones en AWS, proporcionando una base sólida que cumple con las mejores prácticas de seguridad, cumplimiento y eficiencia operativa.

Siguiendo las prácticas recomendadas de AWS en este ámbito, se aíslan los recursos y las cargas de trabajo en varias cuentas de AWS para reducir el impacto.

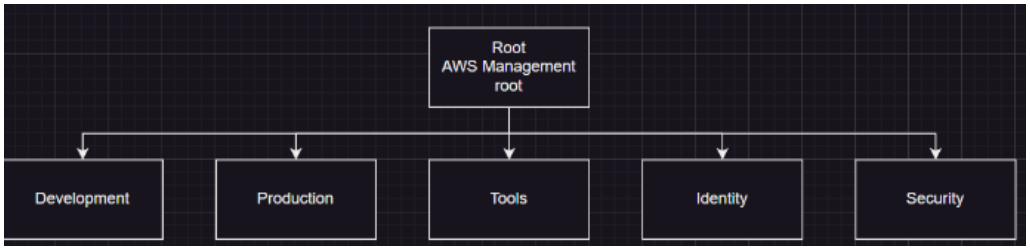

Figura 7.2: Cuentas de AWS por tipos de tareas

De esta forma mediante la distribución vista en la figura 7.2 se consigue una diferenciación según 5 unidades organizativas con cuentas dentro de estas unidades. Tras este primer paso, una vez creadas las cuentas dentro de la unidad organizativa "Development" que serán "UAT", "TEST" y "SAN", se pasa a trabajar con la de "TEST" y "SAN" las asignadas para la fase 1 del proyecto y la fase desarrollada en este trabajo.

La primera tarea a desarrollar trata del Desarrollo y despliegue de red en los diferentes entornos junto a los logs de las VPC. En esta tarea tal y como está descrito en el plan de acción se debe realizar una estrategia de infraestructura según los requisitos, las problemáticas a resolver y las buenas prácticas de AWS se construye la tabla siguiente acerca de la nueva arquitectura en la nube, comparándola con la arquitectura antigua:

Recursos	**Antigua Arquitectura**	**Nueva Arquitectura**
VPC	2 VPC	2 VPC (VPC Payflow, VPC Notification)
Subnets Payflow	6 subnets públicas	6 privadas + 3 públicas
Subnets Notification	1 subnet pública + 2 privadas	3 subnets públicas + 3 privadas
Internet Gateway	2 IGW, una para cada VPC	2 IGW, una para cada VPC
Nat Gateway	2 Nat Gateway, 1 en una subnet pública por cada VPC	6 Nat Gateway, 1 para cada subnet pública

Tabla 7.3: Comparación entre la antigua y la nueva arquitectura

Una vez especificados los recursos que se quieren utilizar en la nueva infraestructura se dio otro paso más para tener una representación visual más fiel que una tabla con los recursos, de manera que se utilizó la herramienta Brainboard.
Mediante el uso de brainboard se dispuso cada uno de los elementos que se querían desplegar de manera general, sin entrar en muchos detalles, pero dividiendo las zonas habilitadas de forma vertical, y mostrando como se configurarían las subredes dentro de las VPC, así como todas las puertas conectadas en las diferentes subredes como se ve en la imagen 7.3.

De esta manera la arquitectura original cumple con una mejor división de las subredes en cada VPC, ya que para cada VPC la asignación de las subredes será acorde a los recursos que se insertan en ella, de forma que tendremos una división de redes privadas y públicas acorde a las necesidades y a los recursos de la red.

Una vez hecho este desglose sobre la nueva arquitectura, y validada la calidad de la nueva creación, se procede a implementarla mediante la infraestructura como código.
Para ello el primer paso es desglosar la estructura de archivos en Visual Studio Code, la cual se irá completando con el paso de las tareas.

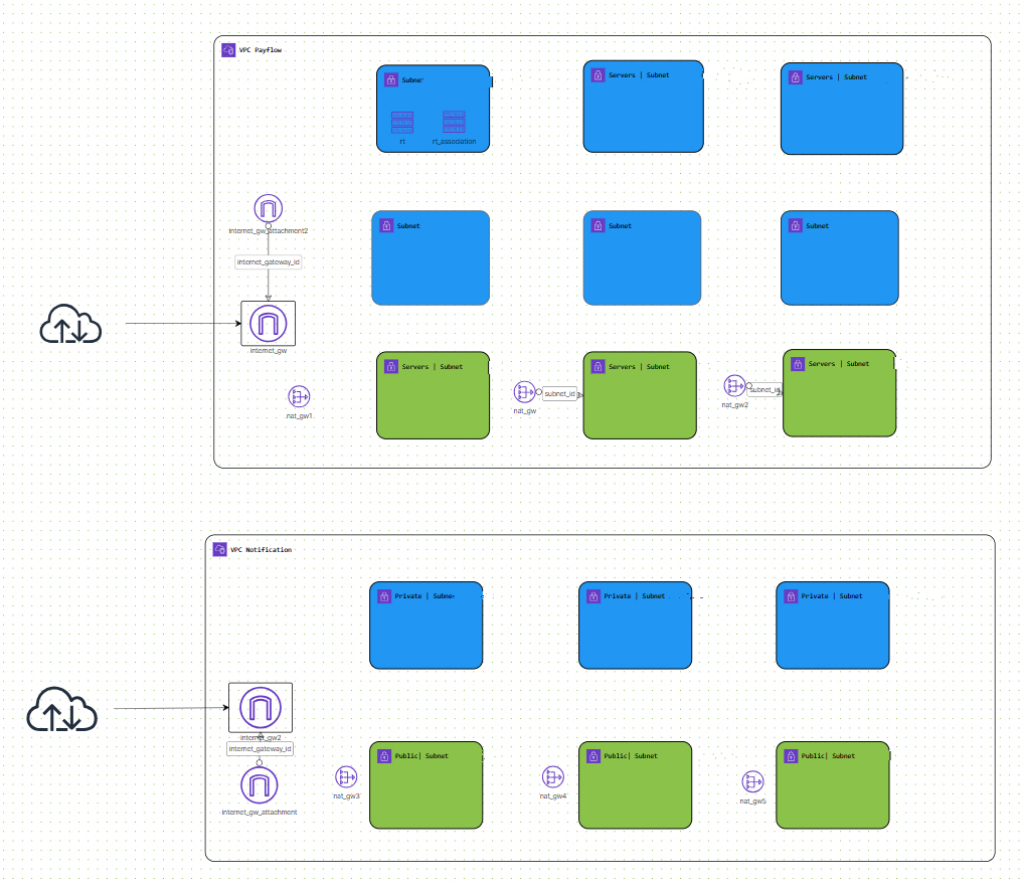

Figura 7.3: Diagrama mediante BrainBoard

7.4.1. Creación de la estructura de archivos en la solución

Ya que en el proyecto se va a trabajar en el entorno TEST y SAN, todo el despliegue de código se produce en TEST, y se reutiliza si es necesario en el entorno SAN, por esto mismo en la estructura de archivos se va a crear solo el directorio TEST.
Aparte de la estructura de TEST tenemos otro directorio modules, y uno global, así como dos archivos extra: uno README.md para dar información del despliegue al cliente (en

el caso particular de este proyecto, el archivo existe, pero no contiene información ya que el despliegue no lo ha hecho el cliente), y un .gitignore ya que está la carpeta conectada a un gitlab de la propia empresa.
A continuación vamos a explicar que contiene cada uno de los directorios dentro del directorio padre del proyecto:

Directorio global

Dentro de global encontramos el uso de main.tf y variables.tf, ya que este directorio tiene como único uso el asignar valores globales a todos los archivos del proyecto, de forma que tiene que obviarse su repetición.

Global será muy útil cuando se despliegue infraestructura en otro entorno como en SAN, ya que variables como regionreferente a la region se despliega dentro de AWS (eu-west-1, us-west-1, etc.), o "tag_group"para asignar los tags de los grupos de recursos como "networking.º "databases"sin tener que estar duplicando esta información por cada entorno.

Directorio networking

En este directorio, que se encuentra dentro de los módulos, se dispone toda la información referente a la configuración, y tras obtener la infraestructura deseada, se hizo la siguiente distinción de archivos dentro de "modules":

- main.tf
  En este archivo main se encuentra la configuración general de toda la red, donde se despliegan los recursos principales de esta.
- bastion.tf
  En este archivo se hará despliegue de un bastión de EC2 con acceso mediante session manager para tener conexión interna con toda la red. La máquina proporciona un punto de entrada a los empleados del cliente, se trata de una petición expresa del cliente que entra dentro de los requisitos de la infraestructura.
- flow_logs.tf
  Para desplegar los logs tal y como se representa en la tercera tarea del plan de acción se hará uso de un archivo de terraform. Aparte de este archivo, en algunos recursos como el bastion de EC2, se activan los logs en la propia configuración.
- vpc_peering.tf
  Para realizar la configuración de la conexión entre las VPC y la VPN se hará uso de dos archivos de configuración, este es el primero de ellos, donde se crea un peering mediante puntos de conexión en ambas VPC.
- vpn.tf
  En este segundo archivo relacionado con la segunda fase del proyecto se crea una VPN client para dar acceso a los clientes. Más adelante se comentará como este componente se despliega pero no es utilizado en el entorno TEST finalmente.
- variables.tf
  Tal y como queda explicado en el apartado relacionado con este tipo de ficheros en la sección 5.3.6, aquí se insertarán el nombre de las variables junto a su tipo, sin darle un valor concreto, ya que este se obtendrá del valor dado en otro archivo en relación al entorno donde se despliegue.

Dentro de estos archivos, obviando el relacionado con las variables, se harán configuraciones de los recursos de AWS, y a partir de estas se realizará el despliegue posterior en el proveedor asignado.

Directorio TEST

Dentro de este directorio TEST tendremos el siguiente desglose de archivos:

- main.tf
  En main se relacionará el valor de las variables asignadas en terraform.tfvars con las variables del archivo variables.tf, todo esto dentro de un módulo, en este caso al solo existir el módulo de networking será en este donde se asignarán las variables de forma única, que conectarán con las variables creadas en este propio módulo donde se creará la configuración de los recursos de networking a partir de estas.

- provider.tf
  En este archivo se encontrará lo referente al proveedor de servicios, en este caso AWS, donde encontraremos una conexión con Terraform Cloud para mostrar que donde se asignan las variables protegidas como las claves de conexión y entrada de la cuenta de AWS del cliente. Además se especifica que se hará contacto con AWS a la hora de desplegar infraestructura para que el propio software tenga esto en cuenta al desplegar los recursos

- terraform.tfvars
  En el archivo de tfvars, tal y como se explica en la sección 5.3.6, se asigna un valor a las variables, en este caso el valor que deben adoptar en el entorno TEST

- variables.tf
  Por último se encuentra el archivo de variables de TEST. En este caso se encuentran las mismas que el módulo de Networking, ya que solo existe este y es necesario para crear la asociación.

7.4.2. Configuración de recursos de red en Terraform

Tras crear la estructura de archivos, y tener una primera visión del diagrama, se utiliza el código obtenido en la herramienta BrainBoard para dar una base al archivo main.tf.

A partir de este código se comienza la generación de configuración concreta para la arquitectura establecida por código. El primer paso es especificar en que región se va a escoger, y teniendo en cuenta que se trata de la mejora de un proyecto existente se reutiliza la misma región que la anteriormente utilizada, en este caso la región del oeste de europa, llamada eu-west-1.

Creación de VPC

Tras escoger la región, se trata de configurar tanto las VPC como las Subnets que componen el proyecto, asignando las variables necesarias para poder dividir por entornos en un futuro, de forma que se le asigna a las VPC una variable en relación al bloque CIDR a cada una, se asigna a true el soporte para dns, y por último se asigna un nombre así como una variable para el grupo y el entorno donde se despliega el recurso.

El uso de tags para el entorno, el grupo de recursos al que pertenece cada recurso y el uso de un nombre para cada recurso es común en la configuración de todos ellos, ya que

con ello aumentamos la calidad del código siguiendo las buenas prácticas mencionadas anteriormente en el apartado 5.3.7.

Creación de subnets

Tras configurar las VPC Payflow y VPC Notification se configuran las subredes asociadas a estas, tanto las privadas como las públicas.
Para VPC Payflow se crean las 9 subredes mencionadas anteriormente en la tabla 7.3, 6 privadas tal y como se especificaba, 3 para datos y 3 para computación, las cuales se diferencian mediante el propio nombre, en el cual además se especifica la zona habilitada. Un ejemplo del nombre de una de estas subredes sería **vpc-cliente-payflow-subnet-compute-private-a**, y en el caso del nombre del recurso en terraform igual pero con barra baja, para seguir las buenas prácticas.
Por otra parte, un ejemplo de como se usa Terraform para la automatización del código en caso de cambios se muestra a la hora de crear las subnets, ya que para especificar las zonas habilitadas se escoge un valor de un array de tres strings para cada una de las subnets, ya que si fuera necesario cambiar la región y con ello las zonas habilitadas a usar, se podría modificar este array con las zonas habilitadas que fuesen necesarias, y con este cambio las subnets se crearían en otras zonas sin cambiar nada del código, solo el valor de una variable.

En cuanto a la creación de las subnets públicas se configuran 3, 1 por cada zona habilitada, apuntando la característica de pública en el nombre y con un argumento que no se configura en las privadas, este es aplicar una ip pública en el lanzamiento, lo cual será necesario para el posterior despliegue de recursos de computación u otro tipo de recursos públicos en estas subredes si fuera necesario, y, al no gastar recursos económicos, se configura esta posibilidad en todas las subredes públicas.
Estas subnets contienen una variable dedicada a cada uno de los bloques CIDR que las componen, y se deberá tener en cuenta

Para VPC Notification se configuran 3 subnets de cada tipo, con una configuración semejante a la de las otras subredes anteriores. En estas subredes la única diferencia será el nombre, ya que en cada subred se especifica a que VPC van conectadas.

Tras tener las VPC y sus subnets creadas, se da paso a los elementos internos a estas.

Creación de Nat Gateways

El primer elemento creado serán las puertas Nat, las cuales se asocian a las subredes públicas, de forma que tenemos la total creación de 6 puertas Nat.

Para generar estas puertas Nat se hace uso de un recurso específico de Terraform .ªws_nat_gateway", donde se debe especificar la subnet donde se inserta la puerta Nat y una ip elástica, ya que es necesario para crear estas puertas Nat.
Esta ip elástica se crea a partir de otro recurso de Terraform para crear estas IPs, de forma que se especificará una variable con la cantidad de ip elásticas necesarias y se hará uso de çount"para crear la cantidad de ip elásticas según el valor que se le asigne a la variable.

De esta forma se crean las 6 Nat Gateways necesarias para la red especificada.

Creación de Internet Gateways

La siguiente puerta que se crea en la infraestructura es la relacionada con la conexión a internet de la red o Internet Gateway. Esta puerta no se asigna a una subred concreta,

sino que se enlaca con la VPC en la que se crea, de forma que tendremos la creación de dos puertas de conexión a internet en ambas VPC.
En estos recursos solo se asigna el valor de la VPC en la que se crean, aparte de los tags generales que se encuentran en todos los recursos creados.

Creación de tablas de enrutamiento

Para que los datos dentro de las subredes y de la propia red virtual puedan pasar a través de las puertas hay que crear unas rutas para ordenar por donde deben pasar estos.

Esta creación de tablas de enrutamiento se explica en la tercera tarea del proyecto, pero por la naturaleza del proyecto y por seguir la creación de la red paso a paso se crea en este punto, después de tener los elementos generales de la red definidos, tal y como se explica anteriormente en la propia explicación del plan de acción.7.3.3

De esta forma es necesaria de crear tablas de enrutamiento, las cuales se generan a partir de dos elementos de terraform, uno para crear la tabla en una VPC y otro elemento para asociar las tablas a una subred dentro de la VPC.

La tabla de enrutamiento se crea mediante .ªws_route_table"donde se configura la VPC donde se inserta y las rutas que se asignan a esta tabla.
Como tenemos unas subredes privadas y otras públicas dentro de las VPC, se crean dos tipos de tablas:

- Para la conexión a internet desde las subredes públicas se asigna la ruta hacia las Internet Gateway permitiendo el paso a un bloque CIDR como variable, que obtendrá el valor en este caso de todo el tráfico o "0.0.0.0/0"
- Para la conexión a internet desde las subredes privadas se asigna la ruta hacia las Nat Gateway permitiendo el paso a un bloque CIDR como variable, que obtendrá también el valor de todo el tráfico o "0.0.0.0/0"

Al disponer de las tablas creadas es necesario realizar la asociación de estas con una subred, por ello se crea el recurso .ªws_route_table_association", donde se selecciona la tabla deseada y se asigna a la subred que toca.

Con estos dos elementos tenemos la creación de tablas de enrutamiento para la trnasmisión de datos dentro de las VPC hacia internet.

7.4.3. Configuración de flow logs

Una vez esta creada la infraestructura de la red con todos los componentes necesario para darle estabilidad y conexión, se da por cerrada la segunda fase de la primera tarea seleccionada en el plan de accion 7.3.1.
Con ello se da paso a la siguiente fase, la creación y recopilación de logs dentro de la red.

Primero se debe escoger entre CloudWatch o S3 para almacenar los logs generados, y tras ello se pasa a la implementación de este apartado en Terraform.

Elección de CloudWatch

Para la elección del almacenamiento se tuvo en cuenta dos opciones, almacenar en s3 o en CloudWatch, cada una con sus ventajas y desventajas.

Tras analizar las dos opciones se llega a la conclusión de que el uso de CloudWatch es mas relevante para el caso del proyecto, ya que ofrece posibilidad de crear alarmas

y alertas sobre las métricas que se obtengan o sobre los valores de los logs, ofrece una forma más fácil de visualizar estos así como analizarlos.
En cambio s3 nos ofrece un beneficio en costo, posibilidades de almacenamiento e integración con otros sistemas externos, pero, ya que la diferencia de precios para los logs que se solicitan son mínimos por la baja generación de estos, y además solo se va a usar AWS para la consulta y análisis de los logs, queda en evidencia que las ventajas del uso de CloudWatch son cuantiosamente mayores al uso de s3 para el caso de uso en el que se está trabajando.

Creación de flow logs y roles

Tras tener la elección de CloudWatch clara se procede a crear los recursos necesarios para obtener y almacenar los logs.

Todos los recursos relacionados con estos logs se crean en el archivo "flow_logs", donde se tiene un total de 5 recursos diferentes necesarios para generar y guardar los logs de las dos VPC creadas en "main.tf".
La parte relacionada directamente con los flow logs se compone de dos recursos; uno estrictamente creado para tener un grupo de logs donde poner los logs a recoger de las VPC; y el segundo recurso es el que crea la posibilidad de recoger estos logs y almacenarlos. Este segundo recurso es .ªws_flow_log", donde se define de que VPC se obtenien los logs, el tipo de tráfico que se va a recoger, el grupo donde se van a guardar los logs (será uno de los grupos creado en el recurso anterior), y un rol asociado a estos logs. Este recurso se debe crear siempre una vez creados los grupos, y es por ello que se hace uso de la función "depends_on"de Terraform, para definir el orden de ejecución.

Como he comentado uno de los atributos es asociarle un rol al recurso de los logs, y es por ello que en este punto se tiene que definir el primer rol, con sus políticas, del proyecto.
Este apartado es otro de los definidos en la tercera tarea del proyeco, pero al tener la obligación de ir creando estos recursos conforme se crean el resto por la necesidad de rellenar los atributos de estos.
Para definir un rol en aws hace falta un total de tres recursos, ya que se dividen las tareas en los siguientes puntos:

1. Creación del rol
   El primer paso es crear el rol como tal, donde se especifica el nombre y que servicio puede asumirlo. Estando en el apartado de los flow logs, se especifica que el rol podrá ser asumido por el servicio de flow logs de AWS. En caso de no concretar que servicio puede asumir el rol, este no podrá ser asumido por ningún servicio de AWS.

2. Creación de la política
   El segundo recurso que se crea es la política que se quiere establecer en el rol. En este caso la política establecida acepta la creación, actualización y descripción de logs.

3. Adjuntar políticas al rol
   En este último recurso se asocian las políticas establecidas al rol creado. En el proyecto se debe usar la función depends_on para establecer que antes de asociar las políticas se deben de haber creado el rol y las propias políticas.

Tras la creación de estos tres recursos ya es posible visualizar los logs en el grupo de CloudWatch creado una vez se haga el despliegue de todo el código.

Este sería el último paso de la tarea 1, divido todo en unas 32 horas y 3 fases. Se da paso con esto a la tarea 2.

7.4.4. Configuración de peering

Como comienzo de la segunda tarea, algo más breve, se prepara la conexión entre las dos VPC.

Para esto se usa el archivo vpc_peering, donde se desarrolla un total de tres recursos, un recurso central y 2 grupos de seguridad.

El recurso central del peering es una puerta de enlace entre ambas VPC. Este recurso se configura con el nombre .ªws_vpc_peering_connection", donde se asignan las dos VPC mediante su id.

El tener esta puerta creada no es suficiente para poder conectar entre las VPC, ya que es necesario generar una ruta de conexión a la puerta de peering, así como aceptar esta ruta mediante una política en un grupo de seguridad.

Para establecer la ruta de conexión se ha configurado una ruta adicional a cada una de las tablas de enrutamiento creadas en el apartado 7.4.2. Esta ruta añadirá la capacidad de transmitir datos hacía la VPC en la cual no está creada a través de la puerta de enlace entre VPC creada anteriormente. De esta forma las subredes creadas en la VPC Payflow podrán enviar datos a la VPC Notification y viceversa a través de la puerta de transmisión.

Además, como última configuración, para evitar bloqueos se crea un grupo de seguridad que añade una regla de entrada y otra de salida a cada una de las redes virtuales donde se permite el envío y recibimiento de datos desde y hasta cualquier puerto mediante cualquier protocolo a través del bloque CIDR de la VPC opuesta.

Al configurar todos estos elementos se consigue la posibilidad de comunicarse desde los componentes de una VPC con los de la otra red.

Cabe explicar en este punto que durante el proyecto se crean más grupos de seguridad, pero a la hora de desplegar estos grupos se complementan utilizando las reglas añadidas en los diferentes grupos para la misma VPC. La creación de varios grupos de seguridad se debe al uso de módulos en la arquitectura, es preferible dividirlo de esta forma y poder decidir el modificar un grupo, eliminarlo o mantenerlo y con ello las reglas que lo componen.

7.4.5. Configuración de VPN

La segunda sección referente a la segunda parte de la implementación del proyecto trata de la creación de una VPN de entrada a la red para los clientes.

Este recurso estaba contemplado al inicio del proyecto, pero como se explicará en un apartado posterior, tras realizarlo e incluso desplegarlo el cliente lo rechazó, por lo que es un elemento que se tuvo en cuenta durante todo el proyecto pero que en la implementación final no estuvo contemplado.

Creación de recursos VPN

Para la creación de la VPN para clientes se configura el archivo "vpn.tf". Dentro de este archivo se realiza la configuración de mas de diez recursos.

Todos estos recursos no son directamente utilizados para crear la VPN, pero si son necesarios para poder darle uso, y una parte esencial es la creación y validación de un

certificado para cifrar la conexión en la VPN y que a la vez el traspaso de datos sea posible desde el exterior hacia la red privada y viceversa. Por ello se utilizan los siguientes servicios:

- Route 53
  Una de las secciones de la configuración es la formada por el servicio de nombres de AWS, Route53. A partir de este servicio se crea una zona de nombres de dominio con un recursos .ªws_route53_zone"la cual solo se crea dándole un nombre, donde se ingresará un registro mediante el recurso .ªws_route53_record". Para este segundo recurso es necesario un nombre, un registro, el tipo de registro y la zona donde se ingresa. Los primeros tres atributos en este caso no son variables, son valores que se obtienen de un certificado existente en AWS, ya que esta zona de nombres de dominio utiliza para validar un certificado, con su nombre y el registro de este, ya que con el nombre de dominio, se utiliza el registrado en el certificado, y se valida la capacidad de usar este en la VPN. Es por ello que el siguiente apartado de la VPN está relacionada con los certificados.

- Certificate Manager
  A partir de este servicio se obtendrá un certificado y se validará mediante el recurso .ªws_acm_certificate_validation". Se obtendrá un certificado ya creado mediante el uso de "dataz el arn de este porque el cliente en un principio disponía de un certificado generado nuevo para la nueva VPN, por lo que se coge este certificado y se pasa en la zona de route53, con su arn y seleccionando la zona para su validación.

Una vez realizada la sección en relación al DNS junto al certificado de la VPN, se comienza la creación de los grupos de seguridad con las políticas para permitir el traspaso de datos entre las IP de las VPC y la VPN. Para ello se utiliza de nuevo el recurso de creación de grupos de seguridad, pero en este caso las reglas de permisión son las siguientes:

- En cuanto a las reglas de entrada se permite el uso del protocolo UDP con el puerto 443, por lo que se permite la conexión desde cualquier IP que cumpla con ese puerto y protocolo y que vaya desde la VPN a la VPC de AWS anteriormente configurada.

- En la regla de salida se permite cualquier tipo de salida, de puerto y protocolo, en dirección a la VPN para poder pasar información desde cualquier sector de la VPC a esta VPN de clientes.

Tras tener el grupo de seguridad configurado y el certificado validado se da paso a la creación de la VPN con todos sus recursos que la componen:

- aws_ec2_client_vpn_endpoint
  Mediante este recurso se crea el punto de entrada del tunel en la red de AWS. Se establecen varios atributos, entre ellos: el bloque de direcciones IP de la VPN para hacer contacto con estas direcciones y permitir el paso de datos de ellas; el certificado anteriormente generado y validado para atuenticar al usuario y cifrar la conexión; el activado de logs en la propia VPN; y la configuración relacionada con la descripción y nombre del recurso, valores de grupo y entorno de despliegue.

- aws_ec2_client_vpn_network_association
  Este recurso se basa en una asociación entre el endpoint VPN y una subnet o VPC completa. En este caso se asocia el endpoint con tres subredes de VPC Payflow, las

tres subredes privadas relacionadas con la computación. Para ello se asigna el id de cada subnet con el id del endpoint creado, por lo que se hace uso del recurso en tres ocasiones, una para cada subnet.

- aws_ec2_client_vpn_authorization_rule
  Al igual que se asocia el punto de conexión VPN con las subredes, se debe crear una regla que atorice el paso de datos por la VPN por parte de estas. Este es el recurso de aprobación de la conexión por parte de la VPN, ya que se acepta el tráfico del bloque cidr de la subnet, y se asocia esta regla con la VPN creada. Este recurso se usa tres veces, una por cada subred.

Tras crear estos recursos se puede desplegar la VPN de conexión çlient-to-site", llamada así porque conecta un cliente (un dispositivo único) con una red remota generalmente corporativo, y con los recursos de esta.

En este punto quedan configurados todos los componentes, mediante variables asignadas para la configuración de estos . Estas variables se insertan en el archivo variables.tf, asignándoles a todas ellas el tipo que son.
Tras configurar estos componentes con las variables se da paso a la creación del bastion EC2, pero antes se va a explicar el porque del no uso de la VPN en la entrega del proyecto al cliente.

VPN descartada en TEST

Tras realizar toda la estructura de la VPN obtuve un problema a la hora de desplegarla, relacionado con el certificado. Al estar tratando con un cliente se hizo la solicitud del certificado, el usado o uno nuevo, pero el propio cliente no llegó a ofrecer este certificado. Por esto mismo, tras una reunión se llegó a la conclusión de dejar hecha la VPN pero no utilizarla, ya que el cliente optó por mantener en uso la VPN que tenía creada con el antiguo certificado, el cual ya tenían validado y en uso. De esta forma se aplicaron las horas al proyecto a la tarea relacionada con este apartado, pero se dejó el despliegue de la VPN apartado en el entorno TEST, con vistas a futuro de desplegarlo con un nuevo certificado en DEV, pero en un sprint diferente.

7.4.6. Configuración de EC2 Bastion y conexión mediante SSM

La última fase de la segunda tarea trata de configurar una instancia EC2 que actúe de punto de entrada a la red de forma interna. Esta EC2 recibe el nombre de bastion porque trata de un servidor de acceso protegido e interno a la red.

Para poder efectuar esta seguridad y que sea todo el acceso de forma interna se descarta el uso de ssh para la entrada al servidor y se utiliza session manager como punto de acceso.

Es por ello que esta configuración se realiza mediante diferentes secciones dentro del archivo "bastion.tf":

Si se desea ver un ejemplo de la configuración en terraform, se encuentra esta misma en el anexo B.

Grupo de seguridad y rol

Al igual que en la VPN, se hace uso del recurso .ªws_security_group"para crear un grupo de seguridad, en el cual se permite el tráfico de entrada y salida de la VPC Payflow para los puertos 443 y el protocolo TCP.

Aparte del grupo de seguridad es necesario también configurar un rol de AWS el cual permita ser asumido por el sistema de EC2, y contenga políticas para interactúar con todos los recursos internos a la VPC, así como los servicios de session manager perteneciente a system manager, para que sea posible entrar al servidor mediante session manager.

Creación del servidor bastion EC2

Tras configurar los recursos relacionados con la seguridad y políticas, se procede a crear el código referente a la máquina EC2.

Para ello se hace uso de un recurso .ªws_instance". Para crear la instancia se selecciona la subnet donde se sitúa, el tipo de instancia que va a ser (cantidad de recursos que utilizará), y hace uso de un perfil de instancia y de una ami para el sistema operativo a utilizar.
Para estos dos últimos elementos es necesario crear dos recursos, uno relacionado con la ami donde, mediante el uso de filtros y del número de ami, se selecciona la el software existente en AWS a utilizar. Y como segundo recurso se hace el uso de .ªws_iam_instance_profile.en el que se asigna a un perfil el rol creado en el recurso anterior.

Con estos elementos queda creada la instancia de EC2, pero sin configurar el acceso por session manager.

Acceso mediante session manager de SSM

Para permitir el acceso mediante el servicio de session manager proporcionado por system manager es necesario crear tres endpoints dentro de la VPC.

Estos endpoints se crean dentro de VPC Payflow, mediante el recurso .ªws_vpc_endpoint".
En este último recurso para la configuración del servidor se establece en los tres existentes, la VPC donde se crea, la subnet que es donde está creada la máquina EC2, el grupo de seguridad para permitir el intercambio de datos entre la VPC y la máquina, el tipo de servicio que es de interfaz en todos ellos, y el nombre del servicio, que varía según el servicio que proporcione AWS para cada endpoint.
Los tres servicios tienen diferentes nombres, porque están relacionados con servicios de diferente tipo. Estos tres son un endpoint para el servicio de EC2, uno para SSM y otro para los mensajes de SSM. Para poder establecerlos es necesario obtener la url entera del nombre de estos servicios, y sin ellos no es posible establecer la entrada por session manager a la máquina virtual.

Tras crear este último recurso se quedan todos los componentes del proyecto configurados, y se da paso al directorio de los entornos donde se establecen las variables según donde se despliegue la red.

7.4.7. Configuración del entorno TEST

Como último paso previo al despliegue queda la configuración del directorio TEST, donde se configuran las variables con los valores deseados para los módulos creados, en este caso, para el único módulo existente.

Al existir un único módulo el archivo de variables será igual al de variables del módulo, pero obviando la definición del tipo de estas, ya que esto se define siempre en el archivo referente al módulo.
Tras tener las variables definidas se configura el valor de estas en terraform.tfvars, llamado así porque la extensión tfvars hace que a la hora de aplicar el apply de terraform se utilice el archivo de forma directa sin tener que definirlo en ningún otro momento. Algunos de los valores definidos que sirven como ejemplo se encuentran en el anexo B.

Una vez asignados los valores, en el archivo main.tf se asignan las variables con un valor dentro del módulo seleccionado. De forma que el valor se escoge de la variable, que suma su valor de tfvars, y se asigna a una variable de las existentes en el módulo. En el caso del proyecto esta tarea es muy sencilla ya que las variables del módulo y del entorno son las mismas, por lo que básicamente se asigna una variable del entorno con otra variable con el mismo nombre, pero esta segunda esta creada en el directorio del entorno.

Con esta definición creada se da paso al despliegue de toda la infraestructura.

7.5 Despliegue de la solución

Para realizar el despliegue se hace uso de la consola que proporciona el propio Visual Studio Code. Como primer paso, obviando que se debe tener instalado terraform tal y como se define en el apartado 5.3.1, se debe ejecutar terraform init. Tras ejecutar el comando se crea el entorno de terraform, y se inicializa el módulo de networking, el entorno de TEST y se conecta con AWS y la cuenta asignada. En caso de insertar nuevos archivos o cambiar el proveedor o la cuenta de este se deberá realizar de nuevo "terraform init".

El siguiente paso es ejecutar el plan, para comprobar la buena configuración de la implementación, así como la posibilidad de despliegue. Tras ejecutar se obtiene por pantalla un JSON con la configuración.

Tras ejecutar el plan se hace uso de uno de los plugins utilizados en el proyecto, se hace uso de Terrascan. Este apartado se explica más adelante en el apartado de las pruebas 8.2.2.

Una vez hecho "terraform planz la prueba con Terrascan, se da paso a realizar el despliegue mediante el comando "terraform apply", el cual ejecuta un plan si no se le pasa por consola, y pide una aprobación mediante un prompt, el cual al acpetarlo comienza el despliegue en la cuenta de AWS.

Una vez acabado el despliegue se obtiene el total de recursos desplegados y ya se puede utilizar toda la infraestructura ya que esta desplegada en la cuenta del cliente de AWS. Aparece un error a la hora de desplegar relacionado con las IPs elásticas, ya que el límite esta preestablecido en 6, por lo que no es posible crear más de esas 6 ip elásticas y con ello solo es posible crear 6 Nat Gateways a la hora de desplegar. Para entornos más avanzados de producción se solicitará a Amazon la ampliación de este límite, pero para TEST se configuró solo con estas 6 IP elásticas.

Llegados a este punto se da el despliegue como realizado, pero falta un último punto por acotar, ya que con el trabajo de despliegue hecho se puede realizar un diagrama final de como está la estructura que se va a entregar al cliente, tanto para tener una visual de todo el proyecto como para que el cliente pueda validar que esta infraestructura es cliente.

Por estas razones se trabajó con draw.io de nuevo para realiza un diagrama tal y como se realizó al principio del proyecto con la estructura original (imagen 7.1, pero con la infraestructura final. Este diagrama se muestra en la imagen 7.4.

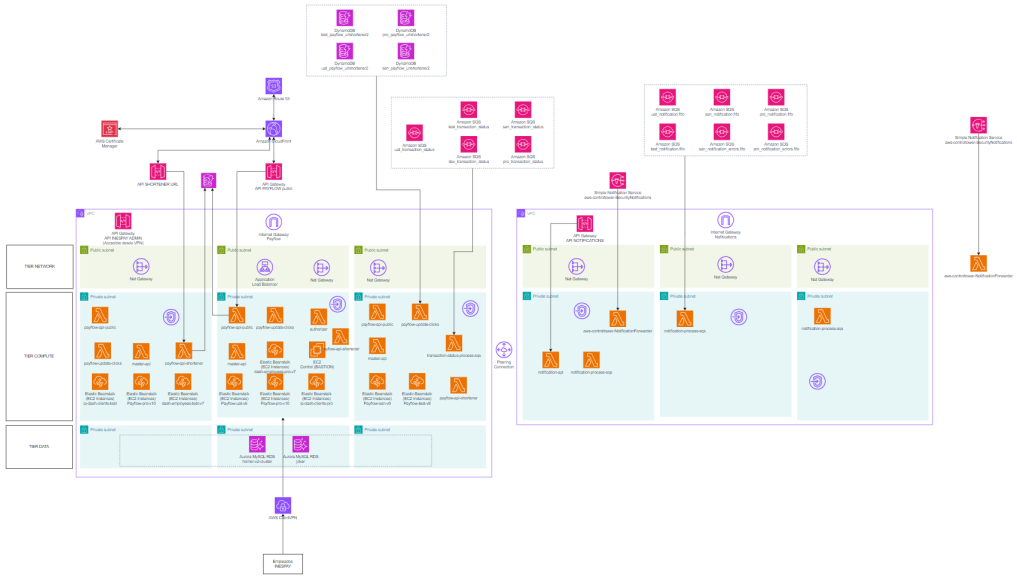

Figura 7.4: Infraestructura final desplegada en AWS

En esta imagen se ve toda la nueva estructura desplegada, con las vpc, las subnets y todos sus recursos dentro, pero cabe destacar que muchos de los recursos que se observan no están, en este momento, configurados ni desplegados, como lo son muchas de las máquinas de Elastic Beanstalk y todos los recursos relacionados con bases de datos, APIs, etc.
El incluir todos los componentes en la figura para que el propio cliente pueda visualizar como será al completo el despliegue, relacionandose con la nueva red creada en AWS.

Tras confirmar que la red y la infraestructura general es correcta mediante el diagrama 7.4, se da paso a las pruebas de validación de todo lo desplegado.

CAPÍTULO 8
Pruebas

En este punto podría darse por finalizado el proyecto, pero las tareas aun tienen horas por imputar. Esto sucede porque las pruebas entran dentro de las horas contempladas para las tareas, por lo que hasta que no se valide el entorno creado mediante algunas pruebas, no se puede cerrar el sprint del proyecto en el que se trabaja.

Para poder validar el buen funcionamiento de la red creada se tratan diferentes apartados, y de forma secuencial se van ejecutando las pruebas posibles hasta llegar a la conclusión de que todos los componentes, así como las conexiones entre ellos, están bien configurados.

8.1 Pruebas en la consola de AWS

Para comenzar con las pruebas, el primer paso trata de comprobar que todos los componentes desplegados mediante código existen en la cuenta de AWS, están disponibles y activos.

Para ello se accede a la consola de aws mediante un navegador de internet, con el usuario y contraseña de la cuenta proporcionada por el cliente para el entorno de TEST.

8.1.1. Configuración recursos de red correcta

Al acceder al panel de control de AWS, se accede al recurso VPC y en el panel inicial del recurso se desglosan los recursos de la configuración de la red tal y como se observa en la figura 8.1. Un punto importante es como se muestra que hay una interconexión entre VPC, que es el peering anteriormente creado en Terraform, por lo que se comprueba que la comunicación entre las VPC está correctamente creada.

Figura 8.1: Panel inicial de recursos en VPC

Tras esto podemos ver que todos los recursos relacionados con la construcción de la red se han generado correctamente, pero además se comprueba que la conexión entre los componentes dentro de la VPC sea correcta. Para ello se utiliza una opción de AWS donde se muestra un mapa de conexiones como se ve en las imágenes 8.2 y 8.3, donde se muestra la conexión entre los componentes de las dos VPC creadas.

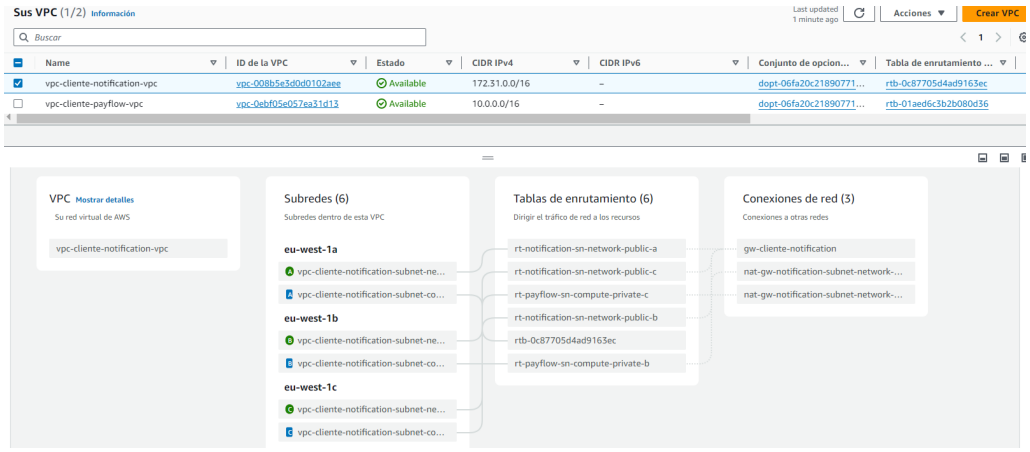

Figura 8.2: Conexiones VPC Notification

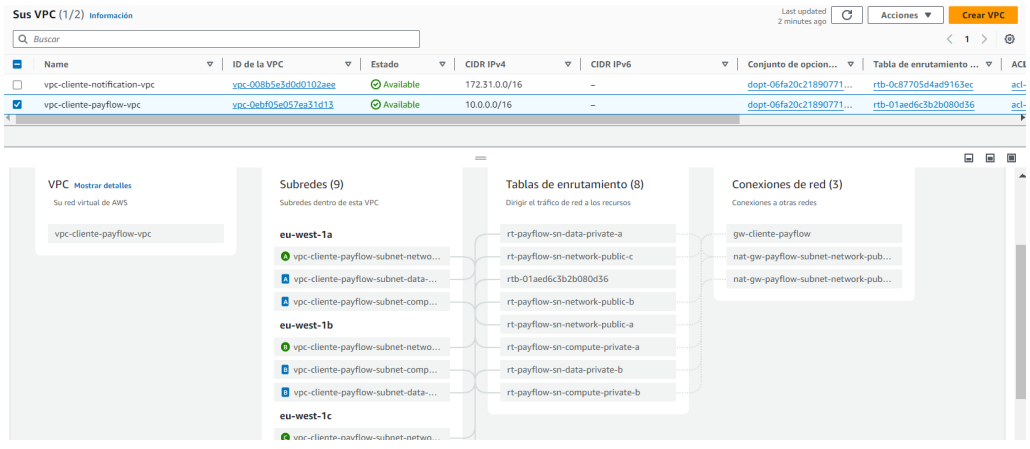

Figura 8.3: Conexiones VPC Payflow

Tras estas comprobaciones visuales sobre la buena creación de la red, se da paso a visualizar en el propio panel de AWS que el grupo de registros de flow logs está creado correctamente y existen estos logs en el grupo.

8.1.2. Configuración de Flow Logs correcta

Para ello se accede al recurso CloudWatch y al seleccionar el apartado de los grupos de registros aparecener, tal y como se ha configurado en el archivo de Terraform, dos grupos de registros, uno para cada VPC, con los nombres que muestra la imagen 8.4.

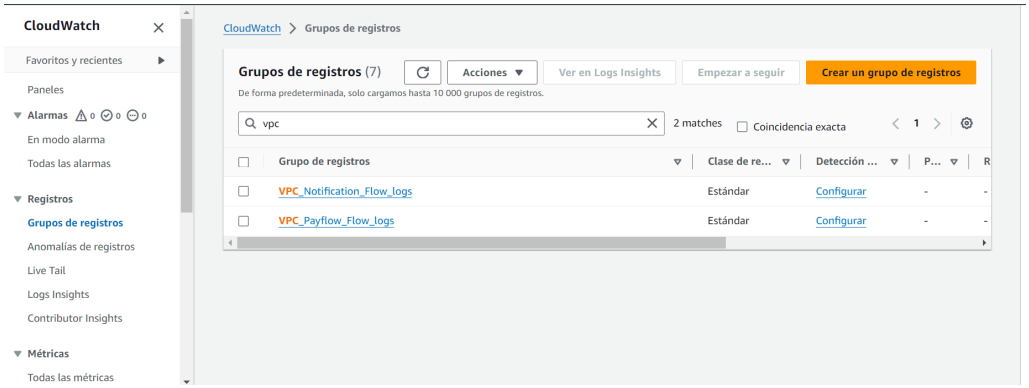

Figura 8.4: Grupos de registros VPC flow logs

Además si se accede a uno de los grupos, por ejemplo el de VPC Payflow, se encuentran todos los streams de logs, que representan diferentes interfaces de red. Y en caso de acceder se encuentran los logs de las conexiones que suceden dentro de las interfaces que se ven en la imagen 8.5.

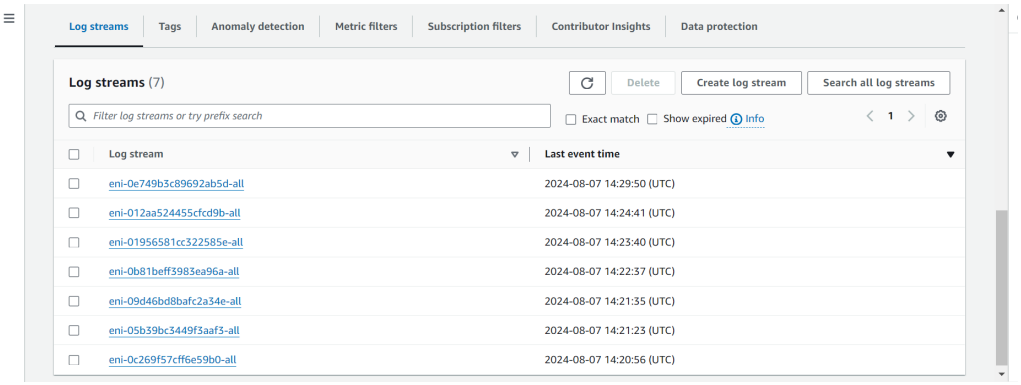

Figura 8.5: Grupos de logs por interfaces de red

8.1.3. Configuración de EC2 Bastion correcta

Para comprobar el buen funcionamiento del EC2 Bastion se procede a acceder al servicio de EC2, y se tratará de entrar en la máquina que aparece creada. Si la conexión está bien realizada, será posible acceder por SSM, y con ello tendríamos la prueba de que tanto los endpoints, como la propia máquina, poseen una estructura correcta. Para ello se accede a la máquina que aparece con el estado Running"seleccionandola y pulsando al botón Çonnect"que aparece en la esquina superior derecha de la imagen 8.6.
Una vez dentro del apartado de conexiones de EC2 se clicka en el apartado de Session Manager, y tal y se prueba a conectar mediante el boton Çonnect"que en este caso aparece

en la esquina inferior derecha en la imagen 8.7. Y con ello se consigue acceder a la máquina EC2 a través de Session Manager de forma privada en la red de AWS, tal y como se observa en la imagen 8.8.

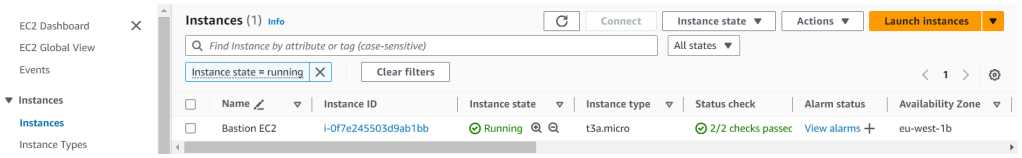

Figura 8.6: EC2 Running

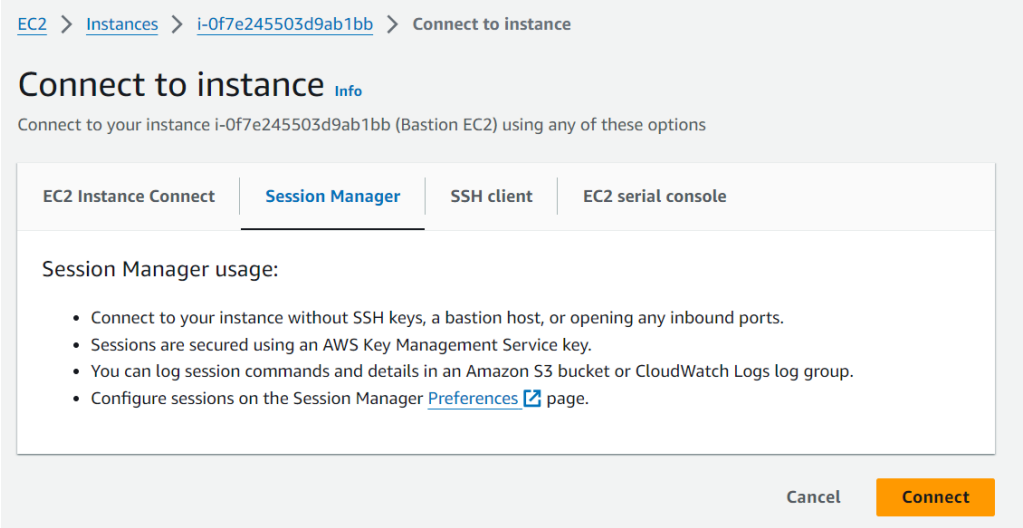

Figura 8.7: Conectar con EC2 via Session Manager

Figura 8.8: Conexión establecida con EC2

Con estas pruebas podemos asegurar que la máquina EC2 que actúa como Bastion está correctamente configurada y en funcionamiento.

8.1.4. Conexiones internas y externas correctas

Para comprobar que la conexión entre las subredes mediante las tablas de ruta definidas es correcta, basta con, desde la propia máquina EC2 que se estaba utilizando en el apartado anterior, hacer ping al exterior, por ejemplo, a google.com. De esta forma, si la conexión es exitosa como se muestra en la figura 8.9, significa que la conexión entre la puerta Nat y la subnet privada donde se encuentra la EC2 es correcta, la tabla de enrutamiento es correcta, y lo mismo sucede con la conexión entre la subnet pública y la puerta de conexión a internet, ya que esta es la ruta de la conexión desde que sale de la máquina hasta que conecta con Google.
Si esta ruta es correcta, se deduce que el resto de rutas de las subnets de ambas VPC lo son porque tienen la misma configuración.

Por otra parte cabe destacar que en AWS la conexión entre subredes de una misma VPC esta permitida de forma predeterminada en las tablas de enrutamiento y grupos de seguridad, por lo que esta conexión se considera correcta desde el principio.

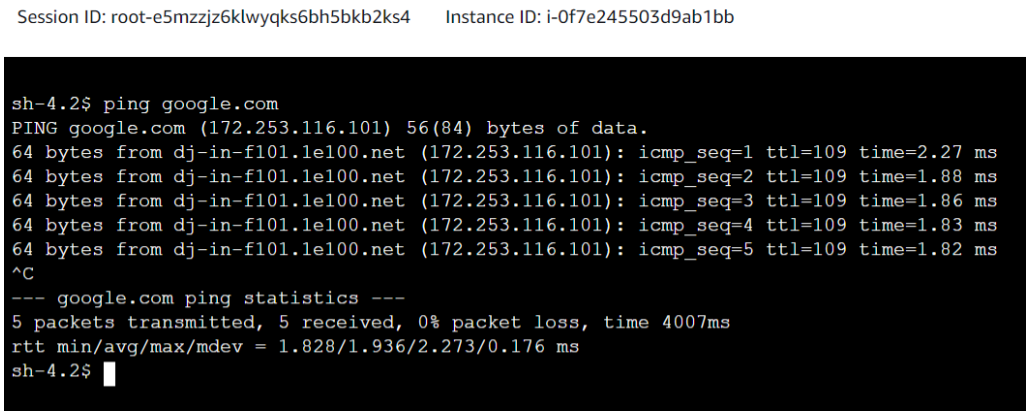

Figura 8.9: Conexión exitosa con Internet desde subnet privada

Por último es necesario comprobar que hay conexión entre elementos de las dos VPC, y que el peering configurado funciona. Para ello es necesario crear un recurso en una subnet (en este caso la única que contiene la Nat asignada para la salida de datos en Internet, ya que en esta VPC solo existe 1 para las subnets privadas por la falta de la ampliación en la capacidad de IPs elásticas, explicado en 7.5) de VPC Notification al cual conectar desde la EC2, para ello se ha creado una máquina virtual sencilla desde la plataforma de AWS solo para realizar esta prueba. Una vez creada se obtiene la IP privada en el menu inferior que se muestra en la imagen 8.10.
Una vez obtenida la IP, entramos de nueva en Session Manager de EC2 para realizar un ping, y observar en la figura 8.11 como los paquetes alcanzan destino, por lo que el peering entre VPC está correctamente configurado y es funcional.

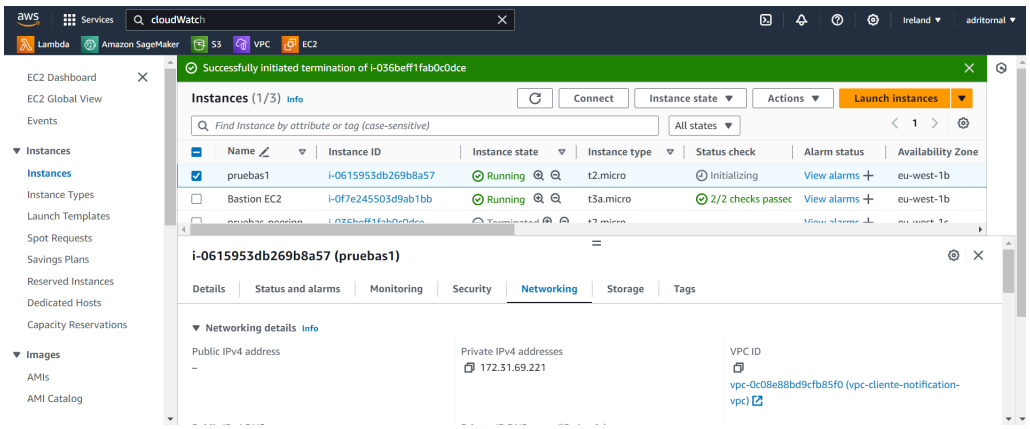

Figura 8.10: Máquina EC2 para pruebas de conexión entre VPCs

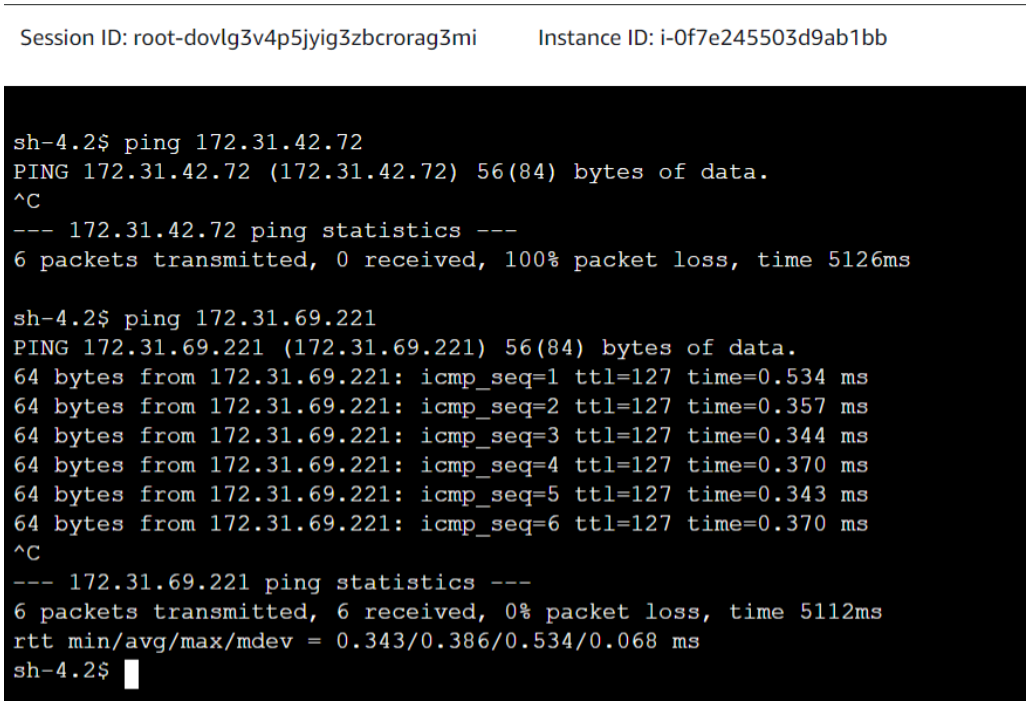

Figura 8.11: Conexión entre VPCs exitosa

Como las conexiones son exitosas en todos sus modos, podemos afirmar que los grupos de seguridad (configurados para permitir el acceso mínimo y necesario) y las tablas de enrutamiento están correctamente creados y funcionan como deben.

Llegados a este punto se han realizado pruebas de la creación y conexión de los componentes de la arquitectura, se ha realizado la comprobación de que los grupos de registros para flow logs están creados y recogen logs de las interfaces de red, se ha comprobado la creación del bastion EC2 y el posible acceso mediante Session Manager, y por último se ha realizado una prueba de conexión entre las subredes privadas a Inetrnet pasando por las puertas Nat y de Internet, así como la conexión entre las dos VPC creadas a través del peering.

8.2 Pruebas sobre la IaC

Las últimas pruebas realizadas tienen relación con la infraestructura como código y el despliegue de esta.

Se ha hecho uso de algunos de los plugins explicados en 5.4.1. Concretamente se ha hecho uso de OPA y Terrascan.

8.2.1. Pruebas con OPA

En el proyecto se ha optado por el uso de OPA para pasar una política personalizada relacionada con el nombre de los grupos de registros y de entorno, asegurando que todos los recursos tienen estas etiquetas definidas y que la definición cumple las buenas prácticas.

Para pasar la prueba es necesario obtener el plan conforme indica el apartado 5.4.1, y con este y la política creada existen dos formas de pasar la prueba tal y como se muestra en la imagen 8.12.

Figura 8.12: Pruebas con la política de OPA

Estas son las dos formas de pasar la política, la primera más compleja donde te saca un resultado en json, y la segunda forma se obtiene añadiendo "–format pretty" donde solo se observa el true o false según acepte el código o no según la política que pasamos.

Tal y como se observa el resultado es true, lo que indica que la prueba se ha superado satisfactoriamente y podemos validar el buen uso de las etiquetas y de las buenas prácticas a la hora de crear estas.

8.2.2. Pruebas con Terrascan

Terrascan se ha usado al final del proyecto antes de despliegue para comprobar que se siguen las buenas prácticas a lo largo de toda la infraestructura como código.

Para ello se hace uso de "scan"la funcionalidad más completa de Terrascan, donde se pasan 142 políticas al código.

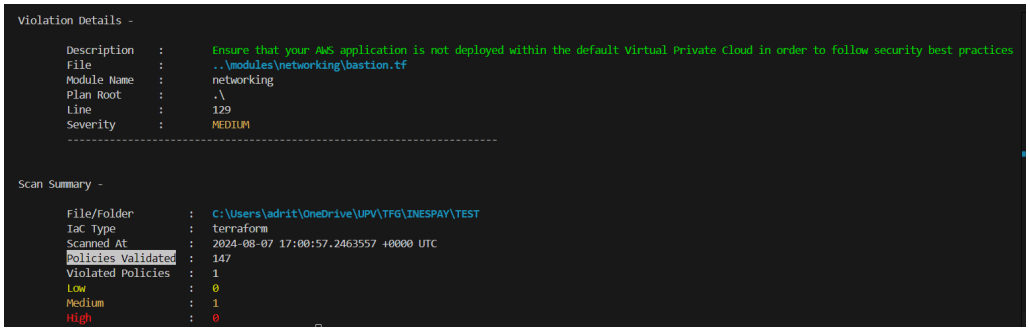

Figura 8.13: Pruebas con Terrascan

Tras ejecutar scan se obtiene el resultado mostrado en 8.13, donde se observa que solo se incumple 1 de todas las políticas existentes. Este fallo se debe a que la subred asociada está pasada mediante una variable y terrascan no detecta que la variable pertenece a una subred creada en una VPC diferente a la preconfigurada en una cuenta de AWS, por lo que no se puede considerar un fallo crítico, y además se obvia a la hora de comprobar la buena creación de infraestructura mediante código.

Tras esta última prueba se da por cerrado el proyecto, con todos los recursos desplegados, y con un buen conjunto de pruebas realizado para asegurar que toda la estructura desplegada cumple con unos requisitos, es funcional y cumple con las exigencias del cliente.

CAPÍTULO 9
Conclusiones

Este Trabajo de Fin de Grado ha permitido llevar a cabo una implementación eficaz de una infraestructura en la nube utilizando AWS como plataforma principal, en conjunto con Terraform como herramienta de automatización para la gestión de la infraestructura. El objetivo principal fue mejorar una infraestructura preexistente, optimizándola mediante el uso de buenas prácticas de la nube y asegurando un despliegue automatizado y eficiente.

Uno de los aspectos más destacados de este proyecto ha sido la complejidad inherente a la gestión de infraestructuras en la nube. AWS, con su amplia variedad de servicios y configuraciones, exige una planificación cuidadosa y una implementación precisa para garantizar que los recursos desplegados sean escalables, seguros y eficientes. La utilización de Terraform ha sido clave para garantizar la consistencia en el despliegue de recursos, permitiendo una automatización que no solo facilita la gestión, sino que también asegura la repetibilidad y el control sobre los entornos creados.

A lo largo del desarrollo, se ha hecho uso de herramientas adicionales que garantizan que la infraestructura cumpla con los estándares de seguridad, lo que ha sido un aspecto fundamental para asegurar que los entornos sean adecuados para su despliegue en producción. Además, este enfoque ha permitido una mayor flexibilidad y ha facilitado la implementación de mejoras de manera ágil y controlada.

Es importante destacar que durante el desarrollo del proyecto he hecho uso de conocimientos adquiridos a lo largo de la carrera, ya que, aunque se trate de un apartado de la informática el cual no se explica concretamente en ninguno de los cuatro años de grado, se compone de conocimientos que si se exponen en este grado. Muchos de estos aprendizajes provienen de asignaturas como RCO, donde te enseñan a trabajar con redes de computadores, entender protocolos, relaciones entre dispositivos y como se identifican estos en la red, todo ello necesario para poder desplegar infraestructura en la nube. También han sido utiles los conocimientos adquiridos en IAP o DEW, ya que estas asignaturas dan la visión suficiente para interactuar con apis, necesario para realizar despliegues desde Terraform por ejemplo. Por último asignaturas como TSR, donde te muestran que es una estructura escalable o los principios de escalabilidad y disponibilidad son muy importantes para entender como funcionan las estructuras en la nube.

En resumen, este proyecto ha demostrado la importancia de una infraestructura bien diseñada y automatizada en la nube. La combinación de las capacidades de AWS con herramientas como Terraform no solo facilita el despliegue eficiente de recursos, sino que también garantiza que la infraestructura pueda crecer y adaptarse a las necesidades cambiantes, manteniendo la seguridad y la estabilidad como prioridades fundamentales. Este TFG ha sentado una base sólida para la gestión de infraestructuras cloud en entornos

empresariales, reflejando las mejores prácticas de automatización y optimización en la nube.

CAPÍTULO 10
Trabajos futuros

Tras la implementación inicial y la mejora de la infraestructura bancaria existente en AWS utilizando Terraform, los próximos pasos se centrarán en la creación y configuración de los recursos adicionales necesarios para completar la infraestructura completa mostrada en la figura 7.4, con un enfoque particular en la seguridad, automatización y cumplimiento normativo. Este proceso incluirá la provisión de bases de datos, instancias EC2, la integración de varios servicios de AWS, y la implementación de políticas de seguridad y monitoreo continuo para garantizar un entorno robusto, seguro y escalable. A continuación, se detallan los pasos específicos que se llevarán a cabo:

1. Creación de Bases de Datos:
   - Bases de datos para las aplicaciones web: Configurar bases de datos relacionales en AWS RDS (Amazon Relational Database Service) para alojar los datos críticos de las aplicaciones web. Estas bases de datos estarán configuradas con opciones de alta disponibilidad, replicación y cifrado para asegurar la continuidad del servicio y la protección de datos.
   - Bases de datos privadas para otros recursos: Provisión de bases de datos adicionales para almacenar información sensible o de uso interno, alojadas en subredes privadas y configuradas con políticas de acceso estrictas para maximizar la seguridad.
2. Implementación de Instancias EC2 mediante Elastic Beanstalk:

   Desplegar instancias EC2 utilizando AWS Elastic Beanstalk, lo que permitirá gestionar automáticamente la capacidad de las instancias, balancear la carga, escalar y supervisar la salud de las aplicaciones desplegadas. Elastic Beanstalk facilitará también la integración con otros servicios de AWS, optimizando el rendimiento y la administración de la infraestructura.
3. Configuración de API Gateway:

   Implementar AWS API Gateway para gestionar las solicitudes de API de manera segura y eficiente. Este servicio permitirá crear, publicar, mantener, monitorizar y proteger APIs a cualquier escala, y facilitará la interacción entre diferentes servicios y componentes de la infraestructura.
4. Integración de SNS y SQS:

   Configurar Amazon Simple Notification Service (SNS) para gestionar las notificaciones del sistema y Amazon Simple Queue Service (SQS) para el manejo de colas de mensajes entre servicios. Estos servicios permitirán la comunicación asincrónica

entre diferentes partes de la aplicación, mejorando la escalabilidad y resiliencia del sistema.

5. Configuración de Route 53:

   Utilizar Amazon Route 53 para gestionar el DNS de la infraestructura, asegurando una resolución de nombres rápida y confiable. Route 53 también se integrará con otros servicios de AWS para ofrecer capacidades avanzadas de enrutamiento, como el balanceo de carga geográfico y la conmutación por error (failover).

6. Automatización y Pruebas:

   Continuar utilizando Terraform para automatizar la creación y gestión de estos nuevos recursos, asegurando que la infraestructura como código (IaC) mantenga la coherencia y reduzca la posibilidad de errores manuales. Se desarrollarán módulos reutilizables de Terraform para cada componente de infraestructura, facilitando futuras expansiones y cambios. Se llevarán a cabo pruebas exhaustivas para validar que la infraestructura cumple con los requisitos de rendimiento, seguridad y escalabilidad definidos por el cliente.

7. Monitoreo y Optimización Continua:

   Implementar servicios de monitoreo como AWS CloudWatch, AWS CloudTrail, y AWS Config para supervisar el rendimiento, registrar actividades, y asegurar la conformidad con políticas establecidas. Se configurarán alertas automatizadas para detectar y resolver problemas de seguridad o rendimiento en tiempo real, asegurando así la alta disponibilidad y seguridad de la infraestructura.

Estos pasos adicionales completarán la creación de una infraestructura robusta y escalable en la nube, alineada con las mejores prácticas de AWS y preparada para futuras expansiones y mejoras. La combinación de servicios como RDS, EC2, API Gateway, SNS, SQS y Route 53, junto con la automatización mediante Terraform y un enfoque en la seguridad y el monitoreo continuo, permitirá optimizar la gestión de recursos y asegurar una alta disponibilidad y seguridad para todas las aplicaciones y servicios desplegados.

Bibliografía

[1] Peter Mell y Timothy Grance. The NIST Definition of Cloud Computing. *NIST Special Publication 800-145*, National Institute of Standards and Technology, Gaithersburg, MD, September 2011. https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf

[2] John Doe y Jane Smith. Metodologías para el desarrollo de proyectos. *Universidad Católica de Colombia*, 2020. https://repository.unicatolica.edu.co/bitstream/handle/20.500.12237/2037/ART%C3%8DCULO_METODOLOG%C3%8DAS_PARA_DESARROLLO_PROYECTOS.pdf?sequence=1&isAllowed=y

[3] Sebastián Hadida y Fernando Troilo. LA AGILIDAD EN LAS ORGANIZACIONES: TRABAJO COMPARATIVO ENTRE METODOLOGÍAS ÁGILES Y DE CASCADA EN UN CONTEXTO DE AMBIGÜEDAD Y TRANSFORMACIÓN DIGITAL . *Econstor Discussion Papers*, UNIVERSIDAD DEL CEMA, Buenos Aires, Argentina, 2020. https://www.econstor.eu/bitstream/10419/238381/1/756.pdf

[4] Mohammad Ubaidullah Bokhari; Qahtan Makki Shallal y Yahya Kord Tamandani. Cloud computing service models: A comparative study. *Institute of Electrical and Electronics Engineers (IEEE)*, 2016 3rd International Conference on Computing for Sustainable Global Development, New Delhi, India, 2016. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7724392

[5] Dimpi Rani y Rajiv Kumar Ranjan A Comparative Study of SaaS PaaS and IaaS in Cloud Computingvol. 4, no. 6 pp. *ijarcsse*, Arni University, Indora Kangra, India, 2014

[6] Hiral B. Patel y Nirali Kansara. Cloud Computing Deployment Models: A Comparative Study. *International Journal of Innovative Research in Computer Science and Technology (IJIRCST)*, Institute of Computer Studies Ganpat University, Kherva India, 2021. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3832832

[7] Mahyar Amini , Nazli Sadat Safavi, Seyyed Mojtaba Dashti Khavidaki y Azam Abdollahzadegan Type Of Cloud Computing (Public And Private) That Transform The Organization More Effectively. *International Journal of Engineering Research and Technology (IJERT)*, Faculty of Computing, Universiti Teknologi, Johor Bahru, Malaysia 2013. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2270660

[8] Sumit Goyal Public vs Private vs Hybrid vs Community-Cloud Computing: A Critical Review *MECS* , I.J. Computer Network and Information Security, New Delhi, India, 2014. https://www.mecs-press.org/ijcnis/ijcnis-v6-n3/IJCNIS-V6-N3-3.pdf

[9] Chris Tozzi. Opciones de IaaS vs. PaaS en AWS, Azure y Google Cloud Platform. *c*omputerweekly, 2019. https://www.computerweekly.com/es/consejo/Opciones-de-IaaS-vs-PaaS-en-AWS-Azure-y-Google-Cloud-Platform

[10] AWS Elastic Beanstalk: Guía para desarrolladores. *A*WS Documentation, 2024. https://docs.aws.amazon.com/es_es/elasticbeanstalk/latest/dg/Welcome.html

[11] ¿Qué es el software como servicio (SaaS)? *A*WS Documentation, 2024. https://aws.amazon.com/es/what-is/SaaS/

[12] Mohamed K. Hussein , Mohamed H. Mousa y Mohamed A. Alqarni A placement architecture for a container as a service (CaaS) in a cloud environment. *Journal of Cloud Computing: Advances, Systems and Applications*, 1Faculty of Computers and Informatics, Suez Canal University, Ismailia, Egypt 2019. https://link.springer.com/article/10.1186/s13677-019-0131-1

[13] MÓNICA JIMÉNEZ CANDELA. OPTIMIZACIÓN DE INFRAESTRUCTURA CLOUD: MIGRACIÓN DE EC2 A UN ENTORNO ECS EN AWS. *Trabajo de fin de grado, Universitat Politècnica de Catalunya*, 2023. https://upcommons.upc.edu/bitstream/handle/2117/407998/183560.pdf?sequence=2&isAllowed=y

[14] Kingson Jebaraj. What is Community Cloud? Benefits and Examples with Use Cases. *knowledgehut*, 2023. https://www.knowledgehut.com/blog/cloud-computing/cloud-computing-for-community

[15] Imran Ahmed Khan. Implementation of Community Cloud Computing Infrastructure in Pakistani Healthcare Organizations. *MS Thesis In Master of Science in Project Management*, COMSATS Institute of Information Technology Virtual Campus, Pakistan, 2014. https://www.knowledgehut.com/blog/cloud-computing/cloud-computing-for-community

[16] Infraestructura global de AWS. *A*mazon Documentation, 2024 https://aws.amazon.com/es/about-aws/global-infrastructure/

[17] AWS Training by KodeKloud. AWS Certified Solutions Architect – Associate (SAA-C03). *AWS Certified Solutions Architect – Associate (SAA-C03).* KodeKloud courses, 2024. https://kodekloud.com/courses/aws-saa.

[18] Pranav Lahitkar TOP 3 Cloud service providers comparison. *M*aster of Computer Applications, 2024 https://www.linkedin.com/pulse/top-3-cloud-service-providers-comparison-pranav-lahitkar-tqzof/?trackingId=GufEr5KYRVKB2vZv%2FZy7gQ%3D%3D

[19] Búho Guardián. AWS vs Azure vs Google: ¿Cuál es mejor para un entorno en la nube?. *n*umber8, blog, 2021. https://www.knowledgehut.com/blog/cloud-computing/cloud-computing-for-community

[20] ¿Qué es la infraestructura como código (IaC)? *R*edHat, accedido el 20/07/2024. https://www.redhat.com/es/topics/automation/what-is-infrastructure-as-code-iac?pfe-g1uzw0xmw=productos

[21] ¿Qué es Terraform? *I*BM, accedido el 21/07/2024. https://www.ibm.com/es-es/topics/terraform#:~:text=Terraform%20es%20una%20herramienta%20de%20infraestructura%20como%20c%C3%B3digo%20de%20mayor,nubes%20p%C3%BAblicas%20y%20nubes%20privadas.

[22] Frankier Flores Qué es Visual Studio Code y qué ventajas ofrece *O*penWebinars, 2022. https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/#:~:text=Visual%20Studio%20Code%20es%20un,y%20conectar%20con%20otros%20servicios.

[23] Prácticas recomendadas de seguridad de la VPC *A*WS Documentation, accedido el 15/07/2024. https://docs.aws.amazon.com/es_es/vpc/latest/userguide/vpc-security-best-practices.html

[24] Convención del nombrado *T*erraform Best Practices, accedido el 17/07/2024. https://docs.aws.amazon.com/es_es/vpc/latest/userguide/vpc-security-best-practices.html

[25] Terraform Registry *T*erraform Registry, accedido el 09/08/2024. https://registry.terraform.io/

APÉNDICE A

Relación del trabajo con los objetivos de desarrollo sostenible de la agenda 2030

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

A.1 Descripción de la alineación del TFG/TFM con los ODS con un grado de relación más alto

En este Trabajo de Fin de Grado (TFG) sobre la optimización y gestión de infraestructuras en la nube utilizando herramientas avanzadas como AWS y Terraform, se puede observar una relación significativa con varios Objetivos de Desarrollo Sostenible (ODS) de la Organización de las Naciones Unidas. A través del enfoque del proyecto, se contribuye a la consecución de algunos de estos objetivos globales, especialmente en los ámbitos de crecimiento económico, innovación tecnológica, producción responsable y acción climática.

En primer lugar, el TFG está estrechamente relacionado con el **ODS 8**, *Trabajo decente y crecimiento económico*. Este objetivo promueve el crecimiento económico sostenido, inclusivo y sostenible, así como el empleo productivo y el trabajo decente para todos. Al mejorar la eficiencia operativa de las infraestructuras en la nube y reducir costos en entornos empresariales como el sector bancario, el trabajo fomenta un entorno más estable y seguro para los trabajadores. La implementación de soluciones de automatización y la optimización de recursos no solo mejoran la productividad, sino que también reducen los riesgos asociados con la gestión manual de infraestructuras, lo que se traduce en condiciones de trabajo más seguras y decentes.

Además, el proyecto contribuye significativamente al **ODS 9**, *Industria, innovación e infraestructuras*. Este objetivo se centra en la construcción de infraestructuras resilientes, la promoción de la industrialización inclusiva y sostenible, y el fomento de la innovación. A través del uso de tecnologías avanzadas como AWS y la metodología de infraestructura como código (IaC) con Terraform, este TFG impulsa la modernización de las infraestructuras digitales, permitiendo que las organizaciones adopten prácticas más ágiles, seguras y escalables. Esta capacidad de innovar y mejorar las infraestructuras tecnológicas refuerza la resiliencia de las empresas frente a los cambios del entorno digital y promueve un desarrollo industrial más sostenible.

Por otra parte, el TFG también toca aspectos del **ODS 12**, *Producción y consumo responsables*, al promover la eficiencia en el uso de recursos tecnológicos. Aunque no es el enfoque central del trabajo, la mejora en la gestión de infraestructuras en la nube tiene el potencial de reducir el consumo energético de los centros de datos, minimizando así el impacto ambiental asociado con las operaciones de TI. La optimización de la infraestructura permite un uso más eficiente de los recursos digitales, lo que contribuye a patrones de consumo más sostenibles y responsables.

Finalmente, el proyecto se alinea de manera indirecta con el **ODS 13**, *Acción por el clima*, al fomentar prácticas que pueden reducir la huella de carbono de las operaciones de TI. La utilización de infraestructuras más eficientes y el uso de la nube para optimizar la gestión de recursos permiten disminuir el consumo de energía y, por lo tanto, las emisiones de gases de efecto invernadero. Este enfoque no solo ayuda a mejorar la eficiencia operativa, sino que también apoya los esfuerzos globales para mitigar el cambio climático, promoviendo una infraestructura tecnológica más sostenible y consciente del medio ambiente.

En resumen, este TFG, a través de la mejora y optimización de infraestructuras en la nube, contribuye a varios ODS relevantes al fomentar el crecimiento económico sostenible, impulsar la innovación tecnológica, promover un uso responsable de los recursos y apoyar la acción climática. Estos elementos no solo refuerzan la importancia del proyecto en el contexto de la ingeniería en la nube, sino que también subrayan su relevancia para el desarrollo sostenible global.

APÉNDICE B
Configuración del sistema

En este anexo se va a mostrar parte del código de la configuración del sistema, mostrando la configuración de cada uno de los tipos de archivos que componen el código de terraform: archivos de configuración de la red, de creación de variables, de establecimiento de los valores de las variables y de creación de módulos.

B.1 Archivos de configuración de la red (VPN y bastionEC2)

El primer archivo de configuración está relacionado con una VPN, sus subredes y las puertas de enlace

```
resource "aws_vpc" "vpc_payflow" {
  cidr_block = var.vpc_Payflow_cidr #"10.0.0.0/16"
  enable_dns_hostnames = true
  enable_dns_support = true
  tags = {
    Name = "vpc-payflow-vpc"
    env = var.tag_env
    group = "payflow vpc"
  }
}

resource "aws_subnet" "subnet_payflow_private_eu_west_1a" {
  vpc_id            = aws_vpc.vpc_payflow.id
  cidr_block        = var.Subnet_payflow_private_eu-west-1a_cidr #"10.0.80.0/20
      "
  availability_zone = var.az[0]

  tags = {
    Name = "vpc-payflow-subnet-private-eu-west-1a"
    env = var.tag_env
    group = "payflow subnet"
  }
}

resource "aws_subnet" "servers_subnet_lambda_2_subnet_db" {
  vpc_id            = aws_vpc.vpc_payflow.id
  cidr_block        = var.server_subnet_lambda2_cidr #"10.0.64.0/20"
  availability_zone = var.az[1]

  tags = {
    Name = "vpc-payflow-subnet-lambda2-eu-west-1b"
    env = var.tag_env
    group = "payflow subnet"
  }
}
```

```
resource "aws_internet_gateway" "internet_gw_payflow" {
  vpc_id = aws_vpc.vpc_payflow.id

  tags = {
    Name = "gw-payflow"
    env = var.tag_env
    group = "payflow internet gateway"
  }
}
resource "aws_eip" "eip" {
  count = var.eip_count
  domain = "vpc"
}

resource "aws_nat_gateway" "nat_gw_subnet_servers_1a" {
  subnet_id     = aws_subnet.servers_subnet_eu_west_1a.id
  allocation_id = aws_eip.eip[0].id

  tags = {
    Name = "nat-payflow-public-eu-west-1a"
    group = "payflow nat gateway"
  }
}
```

Otro ejemplo de configuración es la creación de un bastion, con los roles y políticas así como los recursos necesarios.

```
resource "aws_security_group" "sg_ec2_bastion" {
  name   = var.sg_ec2_bastion_name
  vpc_id = aws_vpc.vpc_payflow.id

  ingress {
    from_port = var.port_tcp
    protocol  = var.protocol_tcp
    to_port   = var.port_tcp
    cidr_blocks = [aws_vpc.vpc_payflow.cidr_block]
    description = "Incoming ssm connection"
  }

  ingress {
    from_port   = var.port_tcp
    to_port     = var.port_tcp
    protocol    = var.protocol_tcp
    cidr_blocks = [var.cidr_block_allow_all]
    ipv6_cidr_blocks  = [var.cidr_block_allow_all_ipv6]
  }

  ingress {
    from_port   = var.port_80
    to_port     = var.port_80
    protocol    = var.protocol_tcp
    cidr_blocks = [var.cidr_block_allow_all]
    ipv6_cidr_blocks  = [var.cidr_block_allow_all_ipv6]
  }

  egress {
    from_port = var.port_all
    protocol  = var.protocol_all
    to_port   = var.port_all
    cidr_blocks = [var.cidr_block_allow_all]
  }
}

resource "aws_iam_role" "ec2_bastion_role" {
```

```
38   name = "ec2_bastion_role"
39
40   assume_role_policy = jsonencode({
41     "Version" : "2012-10-17",
42     "Statement" : [
43       {
44         "Effect" : "Allow",
45         "Principal" : {
46           "Service" : "ec2.amazonaws.com"
47         },
48         "Action" : "sts:AssumeRole"
49       }
50     ]
51   })
52 }
53
54 resource "aws_iam_policy" "ssm_bastion_policy" {
55   name        = "SsmBastionPolicy"
56   description = "Policy allowing access to instances in Elastic Beanstalk
       environment"
57   policy = jsonencode({
58     "Version" : "2012-10-17",
59     "Statement" : [
60       {
61         "Effect" : "Allow",
62         "Action" : [
63           "cloudwatch:PutMetricData",
64           "ds:CreateComputer",
65           "ds:DescribeDirectories",
66           "ec2:DescribeInstanceStatus",
67           "logs:*",
68           "ssm:*",
69           "ec2messages:*"
70         ],
71         "Resource" : "*"
72       },
73       {
74         "Effect" : "Allow",
75         "Action" : "iam:CreateServiceLinkedRole",
76         "Resource" : "arn:aws:iam::*:role/aws-service-role/ssm.amazonaws.com/
             AWSServiceRoleForAmazonSSM*",
77         "Condition" : {
78           "StringLike" : {
79             "iam:AWSServiceName" : "ssm.amazonaws.com"
80           }
81         }
82       },
83       {
84         "Effect" : "Allow",
85         "Action" : [
86           "iam:DeleteServiceLinkedRole",
87           "iam:GetServiceLinkedRoleDeletionStatus"
88         ],
89         "Resource" : "arn:aws:iam::*:role/aws-service-role/ssm.amazonaws.com/
             AWSServiceRoleForAmazonSSM*"
90       },
91       {
92         "Effect" : "Allow",
93         "Action" : [
94           "ssmmessages:CreateControlChannel",
95           "ssmmessages:CreateDataChannel",
96           "ssmmessages:OpenControlChannel",
97           "ssmmessages:OpenDataChannel"
98         ],
```

```
99          "Resource" : "*"
100       }
101     ]
102   })
103 }
104
105 resource "aws_iam_policy_attachment" "bastion_policy_role_attachment" {
106   name = "bastion_role_policy_att"
107   policy_arn = aws_iam_policy.ssm_bastion_policy.arn
108   roles = [ aws_iam_role.ec2_bastion_role.name ]
109 }
110
111 resource "aws_iam_instance_profile" "instance_profile_bastion" {
112   name = "ec2_bastion_profile"
113   role = aws_iam_role.ec2_bastion_role.name
114 }
115
116 data "aws_ami" "amazon_linux" {
117   most_recent = true
118
119   filter {
120     name   = "name"
121     values = ["amzn2-ami-hvm-*"]
122   }
123
124   filter {
125     name   = "virtualization-type"
126     values = ["hvm"]
127   }
128
129   owners = ["137112412989"] # Amazon
130 }
131
132
133
134 resource "aws_instance" "ec2_bastion" {
135   ami                         = data.aws_ami.amazon_linux.id
136   instance_type               = var.instance_type_bastion
137   subnet_id                   = aws_subnet.subnet_apps_2.id
138   security_groups             = [aws_security_group.sg_ec2_bastion.id]
139   iam_instance_profile        = aws_iam_instance_profile.
        instance_profile_bastion.name
140   tags = {
141     Name = "Bastion EC2"
142   }
143 }
```

B.2 Archivos de creación de variables

Otro tipo de archivo es donde se establece la creación de variables (variables.tf). Trata de una de las configuraciones más sencillas y entendibles, por lo que con un simple ejemplo es suficiente para ilustrar como es esta composición.

```
1 variable "env" {
2   type = string
3 }
4
5 variable "tag_env" {
6   type = string
7 }
8
```

```
variable "vpc_Payflow_cidr" {
  type = string
}

variable "az" {
  type = list(string)
}

variable "Subnet_payflow_private_eu-west-1a_cidr" {
  type = string
}
```

B.3 Archivos de establecimiento de valores de las variables

Estos son los archivos denominados terraform.tfvars, y tienen la siguiente composición

```
vpc_Payflow_cidr                        = "10.0.0.0/16"
Subnet_payflow_private_eu-west-1a_cidr  = "10.0.80.0/20"
server_subnet_lambda2_cidr              = "10.0.64.0/20"
eip_count = 6
cidr_block_allow_all = "0.0.0.0/0"
instance_type_bastion = "t3a.micro"
sg_ec2_bastion_name = "Security Group EC2 Bastion"
port_80 = "80"
cidr_block_allow_all_ipv6 = "::/0"
```

B.4 Archivos de creación de los módulos

Este es el archivo nombrado main.tf, donde se hace la creación del módulo a partir de todo lo configurado en el resto de archivos o "módulos".

```
module "networking" {
  source = "../modules/networking"

  env = var.env
  tag_env = var.tag_env
  vpc_Payflow_cidr = var.vpc_Payflow_cidr
  Subnet_payflow_private_eu-west-1a_cidr = var.Subnet_payflow_private_eu-west-1
      a_cidr
  server_subnet_lambda2_cidr = var.server_subnet_lambda2_cidr
  sg_beanstalk_name = var.sg_beanstalk_name
  service_name = var.service_name
  service_name_messages = var.service_name_messages
  service_name_messages_ec2 = var.service_name_messages_ec2
  cidr_block_allow_all_ipv6 = var.cidr_block_allow_all_ipv6
}
```