Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Computación

Infraestructura como código para laboratorios de Ciberseguridad

Autora: Celia Martín Hernández

Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención en Ingeniería de Computación

Infraestructura como código para laboratorios de Ciberseguridad

Autora: Celia Martín Hernández

Tutor: Blas Torregrosa García

Con cada desafío se fortalece el camino hacia nuestros sueños.

Agradecimientos

A mis padres y toda mi familia, quienes han sido mi motivación constante a lo largo de estos años desafiantes pero gratificantes. Su apoyo incondicional ha sido fundamental para seguir adelante sin rendirme y alcanzar este logro significativo en mi vida académica.

A mis amigos, quienes han enriquecido esta etapa con momentos memorables y un apoyo inigualable. Gracias a ellos, esta travesía ha sido muy bonita y llena de aprendizajes compartidos.

También quiero expresar mi agradecimiento a mi tutor, quien me ha brindado la oportunidad de explotar un tema que me apasiona profundamente. Su apoyo ha sido crucial para el desarrollo de este trabajo y me inspira a seguir formándonme en este campo con el objetivo de poder realizar más proyectos como este en un futuro.

Resumen
El uso del Cloud permite disponer de una infraestructura en la nube, eliminando la necesidad de utilizar recursos On-Premises. En este trabajo de fin de grado, se ha diseñado, desarrollado y desplegado una infraestructura en la nube de Azure utilizando la metodología de infraestructura como código mediante Terraform. Esta herramienta permite automatizar y estandarizar la configuración y despliegue de los recursos necesarios, garantizando su consistencia y minimizando los errores humanos. Además, la integración con Azure DevOps ha sido crucial para implementar un proceso de despliegue automatizado mediante pipelines, lo que asegura que los recursos se desplieguen de manera eficiente y repetible.

La infraestructura desarrollada está específicamente diseñada para llevar a cabo pruebas de ciberseguridad, constituyendo un escenario ideal para laboratorios educativos. Esta infraestructura se compone de diversos componentes que han sido configurados siguiendo los requisitos previamente analizados, permitiendo así la ejecución y evaluación de diversas técnicas de ataque. La máquina virtual Kali Linux actúa como la entidad atacante, mientras que la aplicación alojada en la otra máquina virtual es el objetivo de los ataques. Entre las técnicas de ataque que se pueden realizar se incluye la inyección SQL, permitiendo al usuario poner en práctica sus conocimientos en un entorno controlado y seguro.

Abstract
The use of the Cloud allows to have an infrastructure in the cloud, without the need to use On-Premises resources. This paper, an infrastructure has been designed, developed and deployed in the Azure cloud using the methodology of infrastructure as code through Terraform. This tool allows automating and standarizing the configuration and deployment of the necessary resources, ensuring consistency and minimizing the human errors. In addition, integration with Azure DevOps has been the key to implement an automated deployment process using pipelines, ensuring that resources are deployed in an efficient and repeatable process.

The infrastructure developed is specifically designed to perform cybersecurity testing, making it an ideal scenario for educational labs. This infrastructure is composed of several components that have been configured following the previously analyzed requirements, thus allowing the execution and evaluation of various attack techniques. The virtual machine Kali Linux acts as the attacking entity, while the application hosted on the another virtual machine is the target of the attacks. The attack techniques that can be performed include SQL injection, allowing the user to put his knowledge into practice in a controlled and secure environment.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Contexto

Hace unos años, la práctica predominante en las empresas era almacenar su información utilizando su propia infraestructura local, conocida como *On-Premises*[1]. Esto implicaba instalar el software en servidores o dispositivos locales dentro de las instalaciones de la empresa. De esta manera, los usuarios tenían acceso a toda la información almacenada en estos servidores desde dentro de la red corporativa.

En el año 1960, John McCarthy [2], ante el problema que experimentaban las empresas a la hora de intentar obtener la información desde distintos puntos de acceso, introdujo la computación compartida. En los años 90 surgió la empresa Salesforce[3], fundada por el exejecutivo de Oracle Marc Benioff[4], que desarrolló una plataforma *Customer Relationship Management* (CRM)[5]. Este sistema ofrece a los usuarios, a través de una página web, múltiples aplicaciones empresariales. A partir de ese momento, empresas como Amazon[6] comenzaron a impulsar la computación en la nube. Actualmente los proveedores más populares son: Amazon Web Services (AWS)[7], Azure[8] y Google Cloud[9].

El almacenamiento en la nube, también conocido como *Cloud*[10], es un modelo de computación bajo demanda que permite a los usuarios almacenar y transferir datos y archivos a través de internet a proveedores externos. A pesar de la creciente popularidad del Cloud, muchas empresas en la actualidad mantienen almacenados los datos en sus propios centros de datos, lo que ha dado lugar al surgimiento del modelo híbrido[11] combinando el almacenamiento *On-Premises* con el almacenamiento en la nube. En el modelo híbrido, los usuarios tienen la capacidad de conectar recursos basados en la nube a sus infraestructuras locales existentes o integrar recursos de la nube con aplicaciones heredadas en sus sistemas internos. Esto proporciona a las empresas una mayor flexibilidad para adaptarse a sus necesidades específicas y aprovechar lo mejor de ambos enfoques de almacenamiento de datos.

Las principales ventajas de usar el Cloud Computing[12] con respecto al almacenamiento *On-Premises* son las siguientes:

- Pagar por lo que se consume: el usuario paga por el consumo real de los recursos que utiliza.
- Menor gasto de capital (CAPEX [13]): de esta forma se evita gastar dinero en la ejecución y mantenimiento de los centros de datos.
- Escalabilidad: evita predecir la capacidad de la infraestructura. De esta forma se aumentan o reducen los recursos bajo demanda.

- Economía de escala: consiste en agrupar varias empresas para obtener beneficios financieros por volumen, lo que se traduce en la obtención de menores costes.
- Velocidad y agilidad: las aplicaciones se pueden desarrollar e implementar en pocos minutos.
- Globalización de forma instantánea: permite implementar aplicaciones para clientes en cualquier parte del mundo en cuestión de minutos.

1.2 Objetivo

El objetivo principal de este trabajo de fin de grado consiste en desarrollar mediante la Infraestructura como Código (IaC)[14], un escenario para la práctica de ciberseguridad. Para implementar esta metodología, se ha seleccionado la herramienta Terraform[15]. Mediante Terraform, se describe toda la infraestructura necesaria en los respectivos archivos de configuración, facilitando así la reproducibilidad y el control de versiones. Una vez que la infraestructura ha sido definida, se procede a desplegar todos los recursos en la nube. Para este despliegue, se ha escogido el proveedor de Azure, ya que proporciona un entorno seguro y flexible, permitiendo así una implementación eficiente y eficaz del escenario de ciberseguridad planteado.

El escenario desarrollado consta de dos partes diferenciadas. La primera hace referencia a la parte atacante, conocido como *"red team"*, desde donde se van a llevar a cabo los ataques hacia la aplicación vulnerable. Esta sección consta de una máquina virtual con el sistema operativo Linux[16], donde tras su creación, se instala el sistema operativo de código abierto especializado en seguridad, Kali Linux[17]. La segunda parte simula la infraestructura de una empresa, que actúa como la parte atacada, *"blue team"*. Esta infraestructura incluye un servidor web que aloja una aplicación vulnerable y un servidor SQL que gestiona una base de datos. Adicionalmente, para añadir una capa de seguridad extra, se va a configurar un Firewall que protege la conexión entre Internet y la red interna de la empresa, reforzando la defensa del servidor que aloja la aplicación.

1.3 Metodología

Para el desarrollo de este trabajo de fin de grado se ha escogido la metodología Agile[18], que permite gestionar los proyectos de forma iterativa e incremental, gracias a la entrega continua de tareas. Esto ayuda al equipo del proyecto a adaptarse rápidamente a cambios que se originen en los requisitos, ofreciendo una mayor satisfacción al cliente. Además, la metodología Agile fomenta la comunicación entre los distintos miembros del equipo, haciendo que estos estén alineados y enfocados en los objetivos del proyecto.

1.4 Estructura de la memoria

A continuación, se explica de forma breve cada uno de los capítulos que forman parte de este proyecto:

- **Capítulo 1.** Introducción. En este capítulo se explica el proyecto ha desarrollar, el contexto, el objetivo, la metodología empleada y la estructura de este trabajo de fin de grado.
- **Capítulo 2.** Estado del arte. En este capítulo se describen las herramientas empleadas durante el desarrollo de la infraestructura, así como las utilizadas para el almacenamiento y la automatización.

- **Capítulo 3.** Planificación del proyecto. Este capítulo aborda en detalle la planificación inicial y final del proyecto, junto con las tareas y el presupuesto asociado.

- **Capítulo 4.** Escenario propuesto. En este capítulo se detalla todo lo necesario para el desarrollo de la infraestructura, los prerrequisitos, la infraestructura diseñada, una descripción de la infraestructura, el testeo y la automatización.

- **Capítulo 5.** Laboratorio de Ciberseguridad. En este capítulo se muestran los posibles ataques que se pueden realizar a la aplicación desplegada, el listado de puertos abiertos y vulnerabilidades de SQL Injection.

- **Capítulo 6.** Conclusiones y trabajo futuro. En este capítulo se detallan las conclusiones de este trabajo, así como posibles mejoras en la infraestructura para futuros trabajos.

- **Bibliografía.** En este capítulo se presentan las referencias bibliográficas empleadas para el desarrollo de este trabajo de fin de grado.

- **Apéndice A.** Código de Terraform. En este apéndice se incluye el código de Terraform desarrollado para la infraestructura diseñada y el código de las pruebas unitarias.

- **Apéndice B.** Glosario de términos. En este apéndice se definen cada uno de los recursos que conforman la infraestructura.

Capítulo 2

Estado del arte

En el actual panorama tecnológico, la agilidad y eficiencia son claves para el éxito de las empresas. La Infraestructura como Código (IaC) permite definir y administrar la infraestructura utilizando lenguajes de programación y herramientas especializadas. Una de las herramientas más populares para la IaC es Terraform, un orquestador[1] de infraestructura multiplataforma que permite la creación y gestión de recursos en la nube. En este caso, el proveedor de Cloud que se ha escogido para desarrollar este trabajo de fin de grado ha sido Azure.

2.1 Infraestructura como código

La infraestructura como código, *Infraestructure as Code*, permite automatizar la gestión de la infraestructura tanto de los centros de datos como de la computación en la nube (*Cloud Computing*). Para ello, se definen las características de los servicios de la infraestructura como si fueran datos, pudiéndose manejar de forma dinámica dentro de los scripts donde se programan.

Esto ha supuesto muchos beneficios a la hora de construir una infraestructura en comparación con los enfoques tradicionales. Las principales ventajas son:

- Automatización y consistencia: con la IaC se pueden crear scripts o archivos de configuración que describen la infraestructura. Esto permite la automatización de tareas y la gestión de la infraestructura, lo que reduce los posibles errores humanos y asegura la consistencia del entorno.

- Eficiencia y agilidad: gracias a la IaC, se lleva a cabo una implementación rápida y reproducible de la infraestructura. En lugar de realizar las configuraciones de forma manual, se utiliza el código para describir y definir los recursos deseados para la infraestructura.

- Control de versiones y seguimiento de cambios: la IaC permite utilizar herramientas de control de versiones como Git[19]. Gracias a este tipo de herramientas, los usuarios pueden realizar un seguimiento de los cambios realizados en el código. Esto permite una mejor colaboración entre los distintos equipos que trabajan simultáneamente en la misma infraestructura y ofrece la posibilidad de revertir a versiones anteriores si es necesario.

- Pruebas y validaciones: el código de la infraestructura se puede probar y validar antes de ser desplegado, lo que hace que el desarrollador pueda detectar posibles errores o problemas. Por

[1] un orquestador es una herramienta o software cuya función principal es automatizar y controlar el flujo de trabajo, asegurando que las diferentes partes de un sistema interactúen de manera adecuada y eficiente.

lo tanto, estos problemas se pueden resolver antes de que se implemente la infraestructura y garantizar así una mayor calidad en el despliegue.

- Reproducibilidad: la IaC permite la reproducción exacta de la infraestructura en distintos entornos como desarrollo, pruebas, producción... de manera consistente.

Otra práctica muy importante que se complementa con la infraestructura como código es la integración continua (CI) y la entrega continua (CD)[20]. La integración continua verifica y combina el código que se encuentra almacenado en un repositorio compartido de forma frecuente. De esta forma cualquier cambio en el código se somete a pruebas automatizadas para verificar su validez y calidad. La entrega continua permite la entrega rápida de software, de esta forma, los cambios que se hayan producido en el código se pueden implementar y desplegar automáticamente en el entorno deseado reduciendo así el tiempo de implementación. Por lo tanto, ambas ayudan a maximizar la eficiencia, la consistencia y la escalabilidad de la infraestructura, al tiempo que reducen el riesgo de errores y mejoran la colaboración de equipos de desarrollo de operaciones.

2.2 Terraform

Terraform[21] es una herramienta de código abierto que ha sido desarrollada por HashiCorp[22]. Esta herramienta ofrece la posibilidad a los desarrolladores de crear, cambiar y mejorar la infraestructura de manera más segura y eficiente. Para ello, utiliza un enfoque declarativo que define la infraestructura como código, lo que significa que el usuario puede describir como va a ser la infraestructura utilizando un lenguaje de alto nivel. Esto permite a los equipos tratar la infraestructura como cualquier otro software, con control de versiones, pruebas y ciclos de desarrollo ágiles.

Además, Terraform es una herramienta que ofrece compatibilidad con una variedad extensa de proveedores, los cuales son servicios o plataformas en la nube, permitiendo así la creación y gestión de la infraestructura de manera automatizada. Cada proveedor tiene su propio conjunto de recursos y funcionalidades que pueden ser gestionados de manera consistente mediante Terraform. Entre estos proveedores, los más comunes son: Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), Kubernetes[23]... Esto permite a los usuarios utilizar Terraform para desarrollar infraestructuras en cualquier proveedor de su elección, lo que facilita la creación de entornos híbridos o multi-cloud.

La infraestructura se define en una serie de archivos de configuración. Estos archivos han de tener un lenguaje específico que puede ser *HashiCorp Configuration Language* (HCL)[24] o formato JSON[25]. En estos ficheros se describen los recursos que el usuario quiere crear, configurar y administrar, así como las dependencias y relaciones que tienen entre sí. Los recursos son componentes individuales de la infraestructura como instancias de servidores, redes virtuales, bases de datos... Estos recursos especifican qué se va a crear y cómo han de ser configurados. También, Terraform mantiene un archivo de estado que permite rastrear el estado actual de la infraestructura, donde se guarda la información sobre los recursos creados, su configuración y las dependencias entre ellos.

En Terraform también se pueden crear módulos, que permiten la reutilización de código y la organización modular de la infraestructura. Los módulos son pequeños trozos de código en los que se engloban recursos y configuraciones específicas para facilitar la organización y el mantenimiento de infraestructuras que son más grandes y complejas. Terraform Registry[26] es el repositorio de módulos comunitarios que permite al usuario utilizar módulos desarrollados por otros beneficiarios o subir los suyos propios.

Los principales comandos de Terraform para interactuar con la infraestructura son:

- **'terraform init':** inicializa un nuevo proyecto de Terraform y descarga los proveedores necesarios. Este comando se debe ejecutar cada vez que se introduzca un nuevo proveedor o se cree un módulo nuevo.
- **'terraform plan':** muestra un plan detallado de los cambios que se van a realizar en la infraestructura sin llegar a ser aplicados.
- **'terraform apply':** aplica las cambios previstos en la infraestructura.
- **'terraform destroy':** destruye y elimina todos los recursos que han sido creados por Terraform.
- **'terraform state':** permite conocer el estado de los recursos desplegados.
- **'terraform refresh':** compara el estado local del proyecto con el estado actual de la infraestructura en la nube y actualiza el estado local para que coincida con la realidad de la nube.

Después de que la infraestructura se haya definido utilizando scripts de Terraform, el siguiente paso es desplegar esta infraestructura en la nube de Azure. Este proceso incluye también el control de versiones con Git para gestionar los cambios en los scripts de Terraform y el uso de Azure DevOps[27] para automatizar el despliegue mediante pipelines.

2.3 Azure

Azure es una plataforma de computación en la nube desarrollada por Microsoft[28] que permite a las organizaciones cumplir sus objetivos de transformación digital. Para ello Azure cuenta con una infraestructura global masiva compuesta por centros de datos ubicados en diferentes regiones del mundo e interconectados para proporcionar servicios de nube confiables y de alta disponibilidad. Esta plataforma se anunció en octubre de 2008 con el nombre de *"Project Red Dog"*[29], pero no se lanzó hasta el 1 de febrero de 2010 con el nombre de "Windows Azure" convirtiéndose en una de las principales plataformas de computación a nivel mundial. Posteriormente, se cambió el nombre a "Microsoft Azure" el 25 de marzo de 2014.

Azure destaca por su enfoque integral, ofreciendo una extensa gama de servicios que permiten a las empresas construir, implementar y administrar aplicaciones, servicios y recursos de Tecnologías de la Información (TI)[30] de manera eficiente y escalable. Estos servicios abarcan estas áreas:

- Computación: Azure da la posibilidad de crear máquinas virtuales (VM) para ejecutar aplicaciones y servicios en la nube. También proporciona servicios de contenedores como Azure Kubernetes Service (AKS) para orquestar y administrar aplicaciones de contenedores.
- Almacenamiento: hay diversos servicios de almacenamiento en Azure. Entre ellos están Azure Blob Storage para almacenar objetos, Azure Files para almacenar archivos y Azure Disk Storage para almacenar discos virtuales.
- Bases de datos: Azure ofrece Azure SQL Database para bases de datos relacionales, Azure Cosmos DB para bases de datos NoSQL, Azure Database for MySQL para bases de datos relacionales impulsado por MySQL y Azure Database for PostgreSQL para bases de datos relacionales con PostgreSQL.
- Redes: se pueden crear redes virtuales para conectar los recursos en la nube y las instalaciones locales. También proporciona balanceadores de carga, puertas de enlace VPN, enrutamiento de tráfico, Firewall...

- Análisis e inteligencia artificial: cuenta con servicios de análisis de datos entre los que están Azure Synapse Analytics y Azure Data Lake Analytics. Además, ofrece herramientas de inteligencia artificial y aprendizaje automático como Azure Machine Learning y Azure Cognitive Services.

- Internet de las cosas (IoT)[31]: ofrece una plataforma completa para conectar, supervisar y controlar dispositivos de IoT a gran escala.

- Desarrollo de aplicaciones: admite también una amplia gama de lenguajes de programación y proporciona servicios para desarrollar, implementar y administrar aplicaciones web gracias a servicios como Azure App Service, Azure Functions y Azure Logic Apps.

Otro aspecto clave de Azure es su enfoque en la seguridad y la conformidad. Microsoft invierte significativamente en la protección de los datos y la privacidad de los usuarios de Azure. La plataforma cuenta con una sólida infraestructura de seguridad, herramientas de cumplimiento y certificaciones de terceros para ayudar a las organizaciones a cumplir con los estándares y regulaciones específicos de la industria.

Azure también ofrece una amplia gama de herramientas y servicios adicionales para facilitar la gestión y el monitoreo de las soluciones en la nube. Estas herramientas incluyen Azure Monitor, que proporciona visibilidad y control sobre el rendimiento y la disponibilidad de los recursos, y Azure Advisor, que ofrece recomendaciones para optimizar y mejorar la eficiencia de las cargas de trabajo en Azure.

Todos estos servicios que ofrece Azure se distribuyen entre servicios de: Infraestructura como Servicio (IaaS)[32], Plataforma como Servicio (PaaS)[33] o Software como Servicio (SaaS)[34]. Los servicios IaaS permiten a los usuarios acceder a recursos informáticos virtuales. De esta forma los usuarios tienen un control total sobre los sistemas operativos, aplicaciones y las configuraciones que se ejecutan, por ejemplo, en una máquina virtual. Los servicios PaaS dan la posibilidad a los desarrolladores de crear, implementar y administrar aplicaciones sin preocuparse por la infraestructura. Estos servicios proporcionan el sistema operativo, entorno de ejecución... Y por último los servicios SaaS permiten a los usuarios acceder y utilizar aplicaciones basadas en la nube sin la necesidad de instalar o mantener ningún software o infraestructura adicional.

2.4 Git

Git es un sistema de control de versiones que los desarrolladores lo emplean para el seguimiento de los cambios generados en un proyecto de software. Este sistema de control de versiones fue creado por Linus Torvalds[35] en 2005 con la finalidad de facilitar la colaboración de los usuarios en el desarrollo de Linux, pero desde entonces se ha convertido en una herramienta fundamental en la industria del desarrollo de software.

El control de versiones es importante a la hora de desarrollar softwares, ya que ofrece a los equipos de desarrollo rastrear y gestionar los cambios que se han realizado en el código fuente a lo largo del tiempo. Por ello, Git destaca por su enfoque distribuido, lo que significa que cada desarrollador tiene una copia completa del repositorio en su propia máquina local.

Una de las principales características de Git es la capacidad para gestionar ramas (*branches*). Las ramas permiten crear líneas de desarrollo independientes dentro del mismo repositorio, lo que facilita la colaboración en paralelo y la implementación de nuevas funcionalidades sin afectar la rama principal (conocida como rama "master" o "main"). Esto promueve la flexibilidad y agilidad en el desarrollo de software, ya que varios desarrolladores pueden trabajar en diferentes características o correcciones de

errores de manera simultánea.

El flujo de trabajo habitual en Git consiste en crear una rama para una nueva tarea o funcionalidad, donde los cambios se realizan de forma aislada. Una vez que la tarea se completa y los cambios son aprobados, la rama se fusiona con la rama principal del proyecto. Este enfoque facilita la visualización de los cambios nuevos y, en caso de conflictos, Git ofrece herramientas integradas para resolverlos de manera eficiente, garantizando así una gestión eficaz del desarrollo del software.

Otras características importantes de Git son:

- Repositorios locales y remotos: cada desarrollador tiene una copia del repositorio en su máquina local permitiéndolo trabajar de forma independiente y sin tener que estar conectado a internet. Pero también da la posibilidad de sincronizar los cambios con repositorios remotos como GitHub[36], GitLab[37] o Azure DevOps, lo que facilita la colaboración entre desarrolladores y el intercambio de código.

- Historial y seguimiento de cambios: Git guarda un historial completo de todos los cambios que se han realizado en el repositorio. Esto permite revisar versiones anteriores, revertir cambios o investigar el origen de algún problema específico. Todo esto resulta útil en situaciones en las que se depura o rastrea la evolución del código.

- Ramificaciones y fusiones: al poder crear, gestionar y fusionar ramas, Git promueve la implementación de prácticas de desarrollo como la integración continua y la entrega continua.

- Etiquetas y versiones: ofrece la posibilidad de marcar puntos específicos en la historia del repositorio con etiquetas, facilitando la creación de versiones estables y gestión de vida del software.

Además de las características anteriores, Git ofrece una variedad de comandos. Estos comandos permiten a los desarrolladores realizar tareas como clonar repositorios, enviar cambios a un repositorio remoto, revisar el historial de cambios, revertir cambios no deseados... A continuación se muestran los más importantes:

- **'git init'**: se utiliza para inicializar un nuevo repositorio en un directorio vacío. Para ello crea una nueva carpeta .git con todos los archivos y configuraciones necesarias para poder llevar a cabo el control de versiones.

- **'git clone [url]'**: permite clonar un repositorio existente desde un repositorio remoto creando una copia del repositorio en tu máquina local.

- **'git add [archivo]'**: agrega un archivo específico al área de preparación que se incluirá en el próximo commit.

- **'git commit -m "mensaje" '**: crea un nuevo commit con los archivos que se han modificado. El mensaje permite al usuario describir cuales son los cambios que ha realizado.

- **'git push'**: envía los cambios desde la rama local al repositorio remoto.

- **'git pull'**: recupera y fusiona los cambios del repositorio remoto en tu rama local.

- **'git branch [nombre_rama]'**: crea una nueva rama con el nombre especificado.

- **'git checkout [nombre_rama]'**: permite cambiar a la rama especificada.

- **'git merge [rama]'**: combina los cambios de la rama especificada con la rama actual.

Gracias a todo esto, Git se ha convertido en una herramienta imprescindible en la industria del desarrollo de software, aportando una base sólida para la gestión eficiente de proyectos y el seguimiento de versiones, contribuyendo al éxito y eficacia de los proyectos de software.

2.5 Azure DevOps

Azure DevOps es una plataforma integral de desarrollo y entrega continua basada en la nube que es proporcionada por Microsoft Azure. Esta plataforma ha sido diseñada para satisfacer todas aquellas necesidades de los equipos de desarrollo de software. Para ello, combina herramientas y servicios que permiten a los equipos planificar, desarrollar, probar y entregar aplicaciones de manera más eficiente y confiable.

Las herramientas que ofrece Azure DevOps son:

- Azure Boards: esta herramienta permite hacer un seguimiento ágil de los proyectos, ya que ofrece a los equipos la posibilidad de planificar, realizar un seguimiento y discutir el progreso del trabajo de manera centralizada. En el tablero se puede ver de forma completa el flujo de trabajo, comenzando por la creación y asignación de tareas hasta el momento de la entrega final del producto.

- Azure Repos: es un sistema de control de versiones basado en Git. Ofrece repositorios privados y seguros en el que los usuarios pueden almacenar el código fuente. Además da la posibilidad a los equipos de gestionar ramas, realizar fusiones de código y realizar un seguimiento de los cambios originados en el código. Gracias a todo esto, los equipos de desarrolladores pueden trabajar en paralelo en distintas características o tareas y garantizar así la integridad del código en todo momento.

- Azure Pipelines: esta herramienta de compilación y entrega continua (CI/CD) permite automatizar los procesos de compilación, prueba y despliegue de aplicaciones en las distintas plataformas y en los entornos deseados. Para ello los usuarios establecen un flujo de trabajo CI/CD para facilitar la implementación continua y detección de errores de forma rápida.

- Azure Test Plans: es una solución integral de pruebas de software que ofrece a los equipos de desarrollo planificar, rastrear y realizar pruebas de testeo en las aplicaciones desarrolladas. De esta forma, los equipos pueden detectar errores y generar los informes con los resultados de los test para asegurar que el código final es de calidad.

- Azure Artifact: es un repositorio de paquetes donde se almacenan y distribuyen los paquetes de software de forma centralizada. Los distintos equipos pueden controlar versiones, gestionar dependencias y asegurar que el desarrollo de software se lleva a cabo de forma segura.

Además de todas estas herramientas clave, Azure DevOps permite la integración con otras herramientas como Azure Cloud, Azure Functions, Azure Kubernetes Service (AKS)... También se puede integrar con otras soluciones como JIRA[38].

Gracias a todas estas herramientas que ofrece Azure DevOps los equipos pueden trabajar de forma colectiva, ya que ofrece un entorno centralizado donde pueden seguir el proceso de desarrollo del software, realizar revisiones al código y comunicarse entre los distintos equipos. Además los equipos pueden automatizar la compilación, las pruebas y la implementación con los pipelines. También pueden ver el estado completo del proyecto y tener un control sobre las tareas asignadas a cada usuario para seguir con el desarrollo de forma eficiente. Por lo tanto, esta plataforma abarca todo el ciclo de vida de un proyecto, desde que se desarrolla el software, la entrega continua, planificación y seguimiento hasta

el momento de la compilación, las pruebas y la implementación. Como consecuencia de todo esto, Azure DevOps se ha convertido en una de las opciones más populares para los equipos de desarrollo de software que buscan una solución completa y confiable para sus proyectos.

2.6 Kali Linux

Como se comentó anteriormente, la parte atacante cuenta con el sistema operativo de código abierto Kali Linux. Este sistema operativo es una distribución de Linux basada en Debian[39]. Su origen se remonta a la distribución BackTrack Linux[40], la cual surgió en el año 2006 por los desarrolladores Mati Aharoni y Devon Kearns[41]. BackTrack ganó reconocimiento en la comunidad de la seguridad informática debido a su amplia gama de herramientas de seguridad y su facilidad de uso. La visión detrás de esta distribución era proporcionar una plataforma sólida y completa que permitiera llevar a cabo pruebas de penetración y otras actividades relacionadas con la seguridad. Sin embargo, en 2013, Mati y Devon optaron por avanzar hacia una nueva distribución capaz de ofrecer aún más características y funcionalidades. De este modo, surgió Kali Linux, que desde entonces ha experimentado múltiples versiones y actualizaciones.

Una de las características distintivas de Kali Linux consiste en su capacidad para la detección de vulnerabilidades en sistemas informáticos y redes, la recuperación de datos perdidos y el análisis de contraseñas. Como sistema operativo de código abierto, bajo la licencia GNU GPL (*GNU General Public License*)[42], Kali Linux se beneficia de una comunidad activa de desarrolladores. Se prevé que futuras actualizaciones de Kali Linux, programadas anualmente, no solo amplíen su contenido, sino que también introduzcan nuevos entornos y características.

Además de su amplia colección de herramientas, Kali Linux también destaca por su flexibilidad y capacidad de personalización. Al estar basado en Debian, Kali da la posibilidad a los usuarios de adaptar el sistema operativo a sus necesidades específicas. Esto incluye la capacidad de instalar y configurar nuevas herramientas de seguridad, así como la personalización del entorno de escritorio y la apariencia del sistema.

Otra característica importante de Kali Linux es su enfoque en la privacidad y seguridad. Este sistema operativo está diseñado con medidas de seguridad integradas para proteger la privacidad y la integridad de los datos del usuario. Esto incluye características como el cifrado del disco, la eliminación segura de archivos y la navegación anónima a través de múltiples herramientas.

A pesar de sus numerosas ventajas, el uso de Kali Linux también presenta alguna desventaja. En primer lugar, la forma de uso puede ser algo complejo para los usuarios que son nuevos en el campo de la informática, debido a la diversidad de las herramientas. Además, existe el riesgo de que el usuario emplee Kali Linux para actividades ilegales, generando problemas legales y éticos para los usuarios. También, algunas herramientas incluidas pueden requerir de un hardware específico o recursos de sistema significativos, limitando su uso en los sistemas que son menos potentes. Aunque es una buena herramienta para pruebas de seguridad, su uso en entornos de producción puede ser limitado, debido a su naturaleza de distribución de pruebas y su enfoque en la funcionalidad sobre la estabilidad.

Capítulo 3

Planificación del proyecto

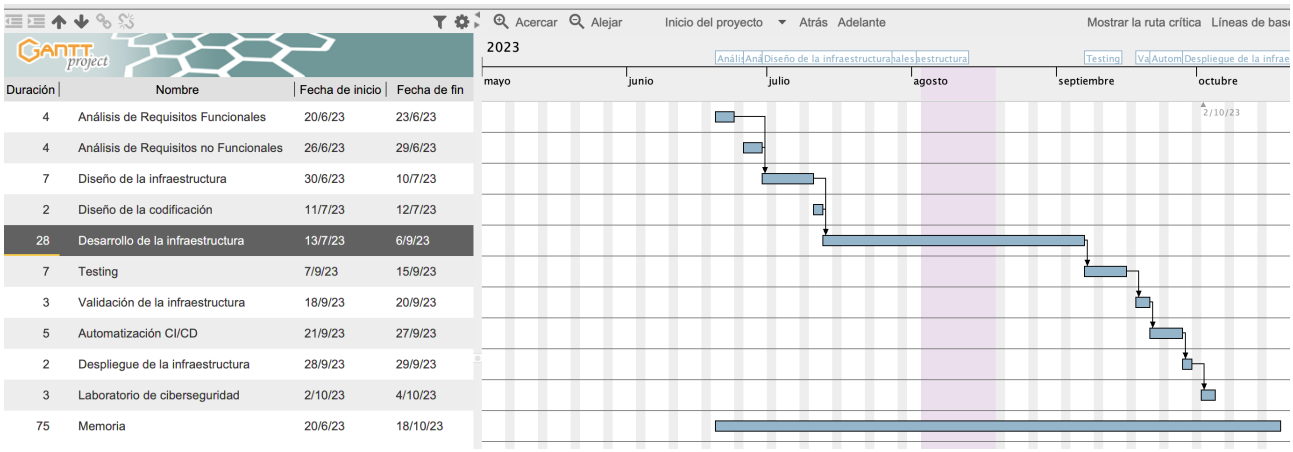
Para llevar a cabo correctamente un proyecto, es esencial crear una planificación detallada que incluya las tareas a realizar. En este caso, se han identificado once tareas específicas y se ha estimado el tiempo necesario para completar cada una de ellas. Esta información se ha plasmado en un diagrama de Gantt, elaborado con la herramienta GanttProject[43]. Al finalizar el proyecto, se ha elaborado un segundo diagrama de Gantt que refleja el tiempo real empleado en cada tarea y se ha calculado el coste global del proyecto.

3.1 Planificación inicial

La planificación inicial proporciona una hoja de ruta clara y estructurada para el desarrollo del proyecto. En esta sección se detallan las tareas identificadas, se estiman los tiempos necesarios para su ejecución y se indican aquellas tareas que tienen dependencias de otras. A continuación se detalla cada una de las tareas:

- **Análisis de requisitos funcionales:** identificar las funcionalidades que debe de cumplir la infraestructura para satisfacer las necesidades del cliente.
- **Análisis de requisitos no funcionales:** identificar las características de la infraestructura como seguridad y usabilidad.
- **Diseño de la infraestructura:** determinar las herramientas y recursos necesarios para construir la infraestructura, basándose en los requisitos funcionales y no funcionales identificados previamente.
- **Diseño de la codificación:** identificar los módulos específicos que serán desarrollados utilizando Terraform para configurar la infraestructura.
- **Desarrollo de la infraestructura:** implementar y configurar los recursos de la infraestructura según el diseño previamente establecido, asegurando que cumplan con los requisitos.
- **Testing:** una vez desarrollada la infraestructura en Terraform, se ejecutan pruebas unitarias y de integración para garantizar el correcto funcionamiento de la infraestructura.
- **Validación de la infraestructura:** verificar que todos los componentes de la infraestructura cumplen con los requisitos establecidos, tras haber completado las pruebas de testeo.

- **Automatización CI/CD:** validada la infraestructura, implementar las tareas de integración y despliegue continuo.
- **Despliegue de la infraestructura:** llevar a cabo el despliegue de la infraestructura una vez completada la automatización y validación, asegurando su correcta implementación.
- **Laboratorio de ciberseguridad:** identificar posibles vulnerabilidades en la infraestructura desplegada en Azure.
- **Memoria:** documentación detallada del proyecto, recopilando información y resultados de cada tarea realizada a lo largo del desarrollo del proyecto. Esta tarea se realiza de forma continua durante todo el proyecto.



Duración	Nombre	Fecha de inicio	Fecha de fin
4	Análisis de Requisitos Funcionales	20/6/23	23/6/23
4	Análisis de Requisitos no Funcionales	26/6/23	29/6/23
7	Diseño de la infraestructura	30/6/23	10/7/23
2	Diseño de la codificación	11/7/23	12/7/23
28	Desarrollo de la infraestructura	13/7/23	6/9/23
7	Testing	7/9/23	15/9/23
3	Validación de la infraestructura	18/9/23	20/9/23
5	Automatización CI/CD	21/9/23	27/9/23
2	Despliegue de la infraestructura	28/9/23	29/9/23
3	Laboratorio de ciberseguridad	2/10/23	4/10/23
75	Memoria	20/6/23	18/10/23

Figura 3.1: Planificación inicial del proyecto.

En la imagen 3.1 se presenta el diagrama de Gantt, el cual muestra la secuencia y duración de las tareas, facilitando así la gestión y el seguimiento del proyecto. Como se observa en la imagen y se comentó en la definición de las tareas, la redacción de la memoria es la actividad que requiere más tiempo. Además, el desarrollo de la infraestructura también ocupa una considerable cantidad de tiempo en el proyecto.

3.2 Planificación final

La planificación final del proyecto muestra una realidad muy común en la gestión de los proyectos, la diferencia entre los tiempos estimados y los tiempos reales de ejecución. En este caso, y como se muestra en el diagrama de Gantt de la figura 3.2, se ve de forma clara que la ejecución del proyecto se ha prolongado siete meses más de lo inicialmente previsto.

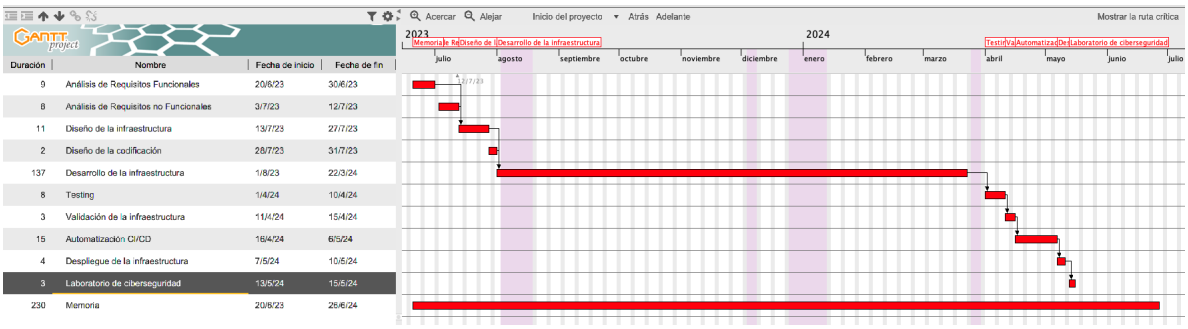


Figura 3.2: Planificación real del proyecto.

A pesar de haber elaborado una planificación inicial lo más realista posible, diversos factores externos han contribuido a su variación. La principal razón que está detrás de este desvío, ha sido la incorporación al mundo laboral. Aunque se estableció en la planificación inicial un horario de trabajo de 4 horas diarias, la jornada completa en la empresa ha obligado a reducir significativamente el tiempo disponible para el proyecto, limitándolo a solo una hora y media aproximada diaria. Este cambio ha generado retrasos significativos en el avance del proyecto. Como se puede observar, todas las tareas han experimentado un aumento en su duración planificada, siendo el desarrollo de la infraestructura la tarea más afectada por esta situación.

3.3 Presupuesto del proyecto

Otra parte importante en la gestión del proyecto es el presupuesto, que proporciona una estimación de los recursos necesarios para llevar a cabo todas las actividades planificadas. En este trabajo de fin de grado, el presupuesto no solo abarca los costes asociados a la ejecución de las tareas, los miembros del equipo y la mano de obra, sino también los gastos de los recursos que conforman la infraestructura. A continuación se explican de forma detallada cada uno de los gastos.

Personal

Para este proyecto, se ha contratado un analista de primer año cuyo coste por hora es de 13,5€. El analista trabaja a jornada reducida, es decir, 4 horas diarias. Según la planificación inicial, la duración del proyecto, incluyendo la fase de documentación, es de 75 días. Además, al finalizar el proyecto, el empleado proporcionará un mes de soporte adicional por si surgen problemas, aunque este costo se considera fuera del alcance del proyecto y se agregará solo si es necesario. Por lo tanto el gasto total en personal es:

Personal	Coste/Hora	Horas	Total
Alumno	13,5	300	4050
Total			4050

Tabla 3.1: Tabla de presupuesto del alumno.

Costes de los recursos

Los recursos que componen la infraestructura tienen un coste estimado por horas. Para realizar este cálculo, se elaborará un presupuesto basado en las horas durante las cuales la infraestructura estará desplegada, abarcando las fases de testeo, validación, despliegue y laboratorio, siendo un total de 15 horas. Para llevar a cabo este proceso, se ha utilizado Azure Calculator[44], una herramienta proporcionada por Azure. A continuación se muestra una tabla detallada con los costes:

Recurso	Coste/Hora	Total
Red Virtual	0,005	0,075
Máquinas virtuales	0,18	2,7
Direcciones IP públicas	0,015	0,225
Firewall	1,17	17,55
Recovery Service Vault	0,1	1,5
Servidor SQL y base de datos	0,5	7,5
Total	1,97	29,55

Tabla 3.2: Tabla de presupuesto de los recursos.

Presupuesto total

Si sumamos el coste del alumno y el coste de los recursos utilizados, obtenemos el presupuesto total estimado para este proyecto. Este cálculo incluye tanto el tiempo invertido por el alumno en el desarrollo y gestión del proyecto, considerando su tarifa por hora, como los costes asociados a los recursos tecnológicos y servicios utilizados en la infraestructura.

Actividad	Coste
Personal	4050
Recursos de Azure	29,55
Total	4079,55

Tabla 3.3: Presupuesto total del proyecto.

Capítulo 4

Escenario propuesto

Para este trabajo de fin de grado, se ha optado por desarrollar una infraestructura dedicada exclusivamente al entrenamiento en ciberseguridad, con la capacidad de enfrentar ataques específicos diseñados para este propósito. Mediante la Infraestructura como Código (IaC), se logrará desplegar y destruir fácilmente esta infraestructura con el uso de las canalizaciones, (*pipelines*). Es importante destacar que esta infraestructura es vulnerable, ya que carece de recursos que podrían fortalecer su seguridad, así como políticas que restrinjan el tráfico externo o a direcciones IP específicas. A continuación, se detallan los pasos necesarios para llevar a cabo el despliegue y los recursos que forman parte de la infraestructura diseñada.

4.1 Prerrequisitos

Para poder desplegar la infraestructura en la nube, el primer paso que se ha llevado a cabo ha sido crear una cuenta en Azure con el email de estudiante que proporciona la Universidad de Valladolid[45]. Esta cuenta gratuita dispone de un saldo de 200$ para ser gastados en un periodo de treinta días. Cumplido el plazo, Azure ofrece la opción de migrar a una cuenta de "pago por uso", donde el usuario únicamente abonará el coste de los servicios que emplee y durante el tiempo de su utilización. Además, Azure ofrece ciertos servicios gratuitos durante los primeros doce meses, aunque cuentan con algunas limitaciones.

Una vez creada la cuenta de Azure es necesario crear una suscripción. Esta suscripción alojará todos los recursos que han sido definidos con Terraform para la infraestructura diseñada. Posteriormente, es necesario crear un registro de aplicaciones, *App Registration*, dentro de Microsoft Entra ID, como se muestra en la figura 4.1. Este registro de aplicaciones constituye un identificador único para una aplicación que se integrará en los servicios de Azure. Mediante este registro de aplicación, el usuario podrá llevar a cabo el despliegue de la infraestructura que ha sido desarrollada con Terraform. Sin embargo, una vez creado el registro de aplicación, es importante asignarle el rol de contribuidor sobre la suscripción. Este rol otorga al registro de aplicación la capacidad de realizar cambios en la suscripción, siendo indispensable para poder llevar a cabo el despliegue de la infraestructura.

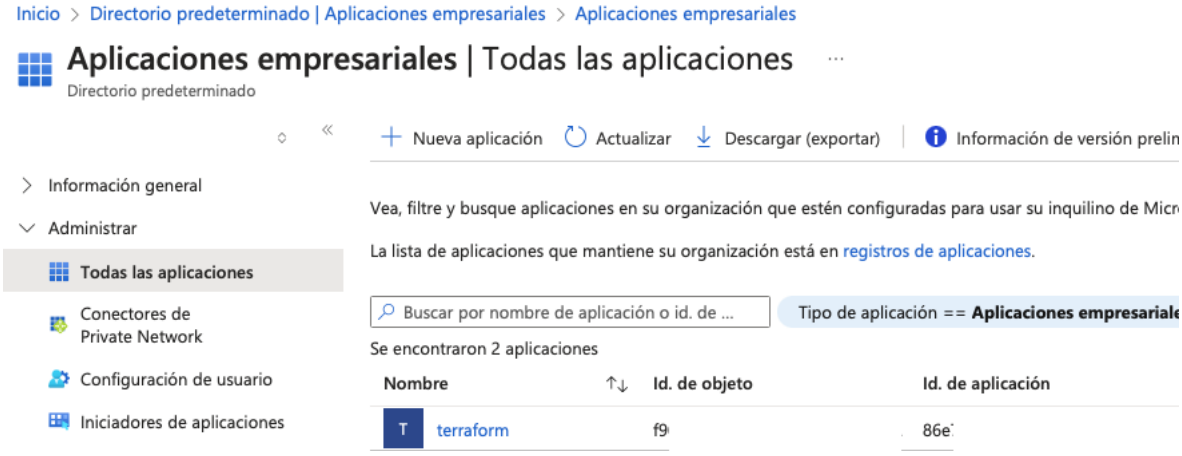


Figura 4.1: App Registration.

También es necesario conectar Azure con Azure DevOps con el objetivo de poder tener un control de versiones sobre el código que se va a desarrollar y también poder desplegar la infraestructura en Azure de forma automática a través de los *pipelines*. Para ello, el usuario debe de ir a *Project Settings* y a continuación ir al apartado *Service connections* como se muestra en la figura 4.2. Llegados a ese punto, para crear la nueva conexión hay que buscar *Azure Resource Manager* elegir la opción *Service Principal Automatic* y por último buscar la suscripción que ha sido creada anteriormente. De esta forma, se habrá establecido la conexión entre Azure y Azure DevOps.

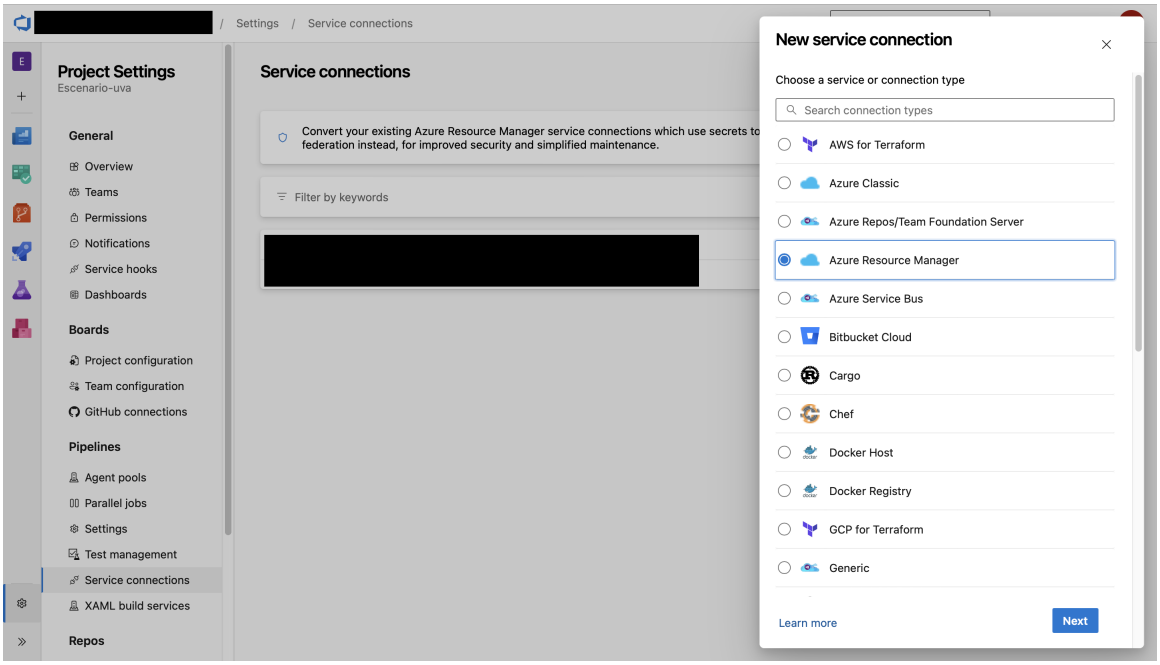


Figura 4.2: Service Connection.

Una vez completados los pasos previos, el siguiente paso es iniciar el trabajo con Terraform. La primera tarea consiste en la creación de un archivo denominado *"terraform.tfvars."*, cuya estructura se asemeja al ejemplo mostrado en la figura 4.3. Este archivo contiene la información necesaria para la configuración del entorno, es decir, los datos sensibles como *subscription_id, client_id, tenant_id* y *client_secret*. Estos últimos tres parámetros están vinculados al registro de aplicación que se ha creado previamente, el cual otorga la autorización para el despliegue de la infraestructura en Azure. Es imprescindible que estos datos se encuentren correctamente configurados en el archivo para que Terraform pueda autenticarse de manera apropiada y llevar a cabo las acciones de aprovisionamiento de recursos en la nube de Azure.

Cuando el proyecto se implemente localmente, sin ser agregado a ningún repositorio de acceso público para otros desarrolladores, se procederá a crear y configurar este archivo de acuerdo con las indicaciones que se detallan a continuación. Sin embargo, para este trabajo de fin de grado, dado que el despliegue se llevará a cabo de manera automatizada mediante pipelines y el código se almacenará en un repositorio de Azure DevOps, se prescinde de la inclusión de este archivo. En su lugar, se guardarán estos valores confidenciales, así como las claves de los recursos tales como las máquinas virtuales y las bases de datos, en un archivo privado que permanecerá oculto para todos los usuarios. Los detalles sobre cómo se almacenan estos valores se explicarán en la sección 4.5.1.

Figura 4.3: terraform.tfvars.

A la hora de comenzar a escribir el código, se ha elegido la estructura por módulos en Terraform, ya que es esencial para garantizar la organización, legibilidad y escalabilidad del código de la infraestructura. Por lo que la estructura del proyecto debe de tener un aspecto similar al que se muestra en la figura 4.4. La estructuración es la siguiente:

- Directorio *modules*: donde cada directorio interno es un recurso de Azure. Estos directorios cuentan con tres ficheros:
  - "nombre_del_modulo.tf": contienen los bloques de configuración de Terraform.

- "variables.tf": contienen las definiciones de las variables que se utilizan en los bloques de configuración.
- "outputs.tf": contienen la información sobre los recursos creados por Terraform.

Fuera del directorio *modules* se encuentran los siguientes ficheros:

- "main.tf": contiene la definición de la infraestructura que se va a crear.
- "provider.tf": contiene la configuración del proveedor de la nube. En este caso de Azure.
- "variables.tf": contiene las definiciones de las variables.
- ".gitignore": especifica que archivos y directorios debe ignorar Git y no incluir en el control de versiones.
- "azure-pipelines.yml": archivo de configuración que se va a emplear para la integración continua y entrega continua.

Figura 4.4: Estructuración del proyecto con Terraform.

Por último, es necesario habilitar el proveedor de recursos para el *Recovery Service Vault* y el SQL Server. De esta forma, Terraform puede acceder y gestionar los recursos en la cuenta de Azure, como crear, modificar y eliminar los recursos utilizando la sintaxis y las funciones proporcionadas por Terraform. Sin este paso, Terraform no puede interactuar con estos recursos. Los pasos que hay que seguir son:

1. En el portal de Azure, ir a la suscripción.
2. Moverse al apartado de Proveedor de recursos.
3. Buscar los proveedores. En este caso: Microsoft.RecoveryServices y Microsoft.Sql.
4. Registrarse.

Una vez completados todos estos pasos, que incluyen la creación de la cuenta en Azure, la configuración de la suscripción y el registro de aplicaciones, el enlace entre Azure y Azure DevOps y la creación de la estructura que va a seguir el proyecto; el siguiente paso es iniciar el proceso de codificación de la infraestructura con Terraform.

4.2 Infraestructura

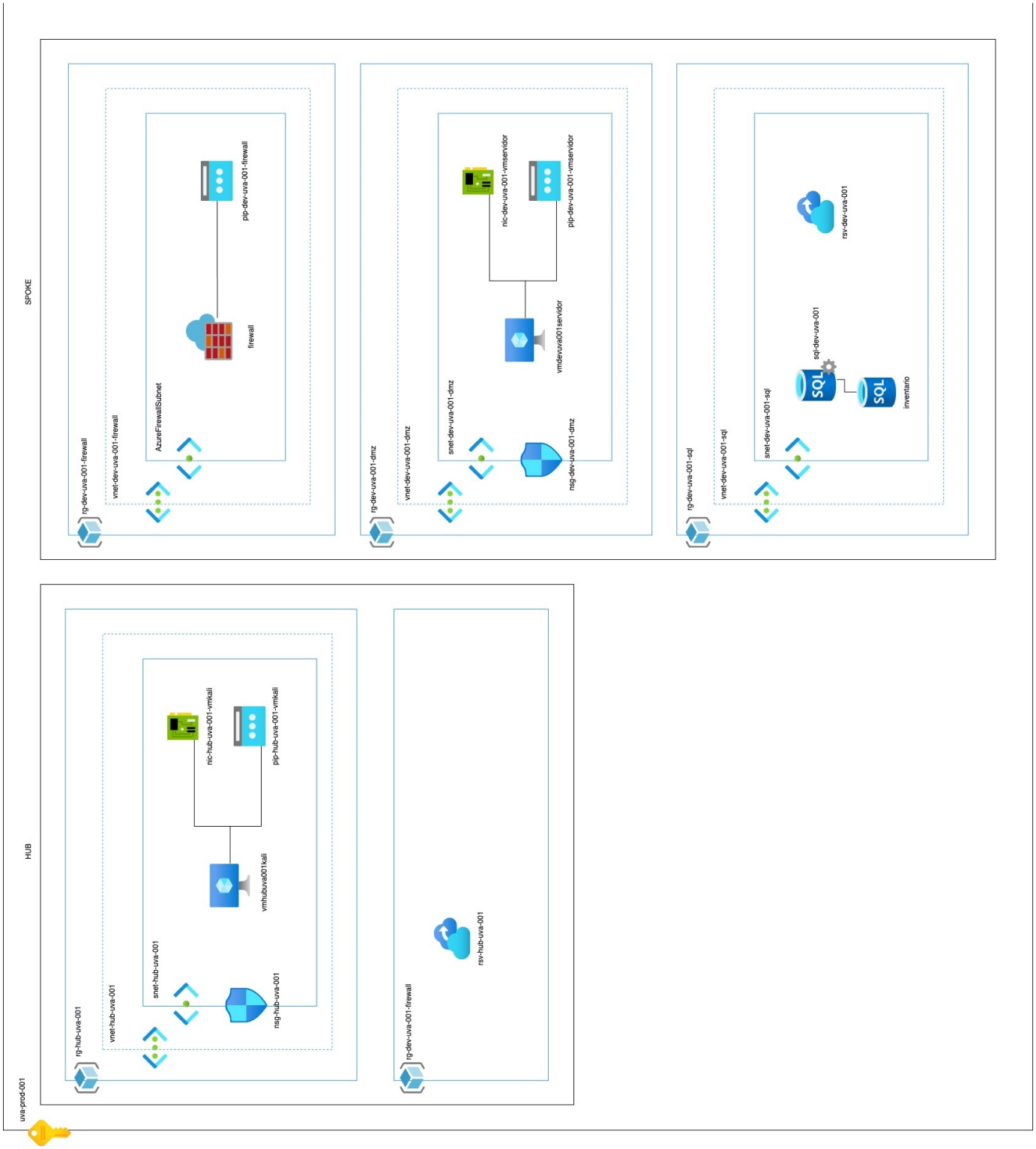


Figura 4.5: Infraestructura diseñada.

4.3 Descripción de la infraestructura

El escenario propuesto ha sido diseñado con el fin de facilitar la realización de pruebas de ciberseguridad, enfocadas en el ataque a una aplicación alojada en un servidor específico desde la máquina Kali.

Para garantizar la eficacia y reproducibilidad de este entorno, toda la infraestructura ha sido codificada utilizando la metodología de Infraestructura como Código (IaC) con Terraform. Se ha seleccionado Azure como la plataforma de despliegue, y todos los recursos serán desplegados en la región *West Europe*.

4.3.1 Disposición de los recursos en la infraestructura

Como se comentó anteriormente, el escenario ha sido pensado para poder realizar ataques de ciberseguridad, por ello, se ha optado por la implementación de una metodología HUB y SPOKE[46]. En este esquema, el HUB alberga la máquina virtual que desempeñará el rol de parte atacante, mientras que en el SPOKE se encuentran los recursos que serán objeto de los ataques desde el HUB.

Toda la infraestructura parte de una suscripción llamada uva-prod-001. A continuación se explica de forma breve la estructura de la infraestructura:

- **HUB:**
  - El grupo de recursos *rg-hub-uva-001* formado por los siguientes recursos: la red virtual *vnet-hub-uva-001*, la subred *snet-hub-uva-001*, el grupo de seguridad de red *nsg-hub-uva-001*, la máquina virtual *vmhubuva001kali*, la tarjeta de interfaz de red *nic-hub-uva-001-vmkali* y la IP pública *pip-hub-uva-001-vmkali*.
  - El grupo de recursos *rg-hub-uva-001-security* que cuenta con la bóveda de recuperación de servicios *rsv-hub-uva-001*.
- **SPOKE:**
  - El grupo de recursos *rg-dev-uva-001-firewall* donde se encuentran: la red virtual *vnet-dev-uva-001-firewall*, la subred *AzureFirewallSubnet*, el *Firewall* y la IP pública *pip-dev-uva-001-firewall*.
  - El grupo de recursos *rg-dev-uva-001-dmz* que aloja los siguientes recursos: la red virtual *vnet-dev-uva-001-dmz*, la subred *snet-dev-uva-001-dmz*, el grupo de seguridad de red *nsg-dev-uva-001-dmz*, la máquina virtual *vmdevuva001servidor*, la tarjeta de interfaz de red *nic-dev-uva-001-vmservidor* y la dirección IP pública *pip-dev-uva-001-vmservidor*.
  - El grupo de recursos *rg-dev-uva-001-sql* donde se encuentran: la red virtual *vnet-dev-uva-001-sql*, la subred *snet-dev-uva-001-sql*, el Servidor SQL *sql-dev-uva-001*, la base de datos Inventario y la bóveda de recuperación de servicios *rsv-dev-uva-001*.

4.3.2 Configuración de los recursos

La infraestructura se compone de diversos recursos cuya configuración se detallará a continuación. Todo el código de Terraform que abarca esta configuración está disponible en el anexo A y también se encuentra en el repositorio Terraform-infraestructura.

- Grupos de recursos:
  - rg-hub-uva-001, donde se encuentran los recursos necesarios para crear la máquina virtual que corresponde con la parte atacante de la infraestructura.

  - rg-hub-uva-001-security, aloja la bóveda de recuperación de servicios.
  - rg-dev-uva-001-firewall, donde se encuentra el Firewall, proporcionando un nivel de seguridad entre Internet y la red interna de la organización.
  - rg-dev-uva-001-dmz, aloja los recursos necesarios para crear una máquina virtual, que en este caso actúa como servidor, y corresponde a la parte que va a ser atacada dentro de la infraestructura. Para este escenario, el servidor necesita exponerse a Internet, por ese motivo se encuentra en la Zona Desmilitarizada.
  - rg-dev-uva-001-sql, donde se encuentran el resto de recursos que tiene la organización en su red interna.

- Redes Virtuales:
  - vnet-hub-uva-001.
  - vnet-dev-uva-001-firewall.
  - vnet-dev-uva-001-dmz.
  - vnet-dev-uva-001-sql.

- Subredes:
  - snet-hub-uva-001, asociada a la red virtual vnet-hub-uva-001.
  - AzureFirewallSubnet, asociada a la red virtual vnet-dev-uva-001-firewall.
  - snet-dev-uva-001-dmz, enlazada con la red virtual vnet-dev-uva-001-dmz.
  - snet-dev-uva-001-sql, vinculada con la red virtual vnet-dev-uva-001-sql.

  En la tabla 4.1, se encuentran los rangos de IPs que se han seleccionado para las redes virtuales y la subredes que forman parte de este escenario.

Recurso	Rango de IPs
vnet-hub-uva-001	10.0.0.0/16
vnet-dev-uva-001-firewall	10.1.0.0/16
vnet-dev-uva-001-dmz	10.2.0.0/16
vnet-dev-uva-001-sql	10.3.0.0/16
snet-hub-uva-001	10.0.0.0/24
AzureFirewallSubnet	10.1.0.0/24
snet-dev-uva-001-dmz	10.2.0.0/24
snet-dev-uva-001-sql	10.3.0.0/24

Tabla 4.1: Información de los rangos de IPs de las redes virtuales y las subredes.

- Grupos de seguridad de red:
  - nsg-hub-uva-001.
  - nsg-dev-uva-001-dmz.

  Ambos grupos de seguridad de red presentan la asociación de dos reglas distintivas que se recogen en la tabla 4.2. La primera regla, "allow_ssh" se encarga de habilitar la conexión mediante el protocolo SSH hacia la máquina virtual. La segunda regla, "allow_apps" permite el flujo de entrada hacia el puerto 8080, permitiendo así el acceso a la aplicación web.

Regla	Dirección	Acceso	Protocolo	Puerto origen	Puerto destino	Prioridad
allow_ssh	Inbound	Allow	TCP	*	22	100
allow_apps	Inbound	Allow	TCP	*	8080	110

Tabla 4.2: Configuración de las reglas de los grupo de seguridad de red.

- Máquinas virtuales:
  - vmhubuva001kali. Esta máquina virtual asume el rol de servidor en calidad de parte atacante para este trabajo de fin de grado y está enlazada a la tarjeta de interfaz de red nic-hub-uva-001-vmkali.
  - vmdevuva001servidor. Esta máquina virtual actúa como servidor y representa la parte atacada en este escenario, albergando una aplicación destinada a recibir los ataques. Está asociada a la tarjeta de interfaz de red nic-dev-uva-001-vm servidor.

  Ambas máquinas virtuales cuentan con la misma configuración que está detalla en la tabla 4.3.

Especificación	Valor	Descripción
Tamaño	Standard_F2	Ideal para entornos de servidores web con tráfico moderado
CPUs	2	No se requiere una potencia de procesamiento mayor para este escenario
Sistema de caché	ReadWrite	Permite almacenar temporalmente tanto datos leídos como escritos para mejorar el rendimiento
Cuenta de almacenamiento	Standard_LRS	Replica los datos dentro de una misma región del centro de datos, proporcionando redundancia y protección contra fallos de hardware
Sistema Operativo	Ubuntu	Última versión

Tabla 4.3: Configuración de las máquinas virtuales.

- Tarjetas de interfaz de red:
  - nic-hub-uva-001-vmkali, enlazada con la dirección IP pública pip-hub-uva-001-vmkali.
  - nic-dev-uva-001-vm servidor, asociada a la dirección IP pública pip-dev-uva-001-vmservidor.

  Estas tarjetas de interfaz de red permiten la conexión entre las máquinas virtuales y los dispositivos que se encuentren en la red global.
- IP públicas:
  - pip-hub-uva-001-vmkali.
  - pip-dev-uva-001-firewall.
  - pip-dev-uva-001-vmservidor.

  Estas direcciones IPs permiten a las máquinas virtuales ser accesibles de forma remota. También permiten al Firewall actuar como punto de control de seguridad entre la red interna e Internet. Su configuración se muestra en la tabla 4.4.

Especificación	Valor
SKU	Estándar
Tipo	Estática

Tabla 4.4: Configuración de las IPs públicas.

- Firewall:

  - Firewall, que desempeña un papel crucial al establecer una configuración adicional de seguridad con el fin de prevenir posibles ataques dirigidos hacia la infraestructura de red de la organización. El Firewall está asociado a la subred AzureFirewallSubnet y a la dirección IP pública pip-dev-uva-001-firewall.

  Las especificaciones del Firewall se encuentran en la tabla 4.5.

Especificación	Valor
SKU	AZFW_Hub
Tipo	Estándar

Tabla 4.5: Configuración del Firewall.

Las reglas establecidas en el Firewall se encuentran en la tabla 4.6.

Regla	Acceso	Protocolo	Puerto origen	Puerto destino	Prioridad
allow	Allow	TCP	*	[22, 8080]	100
deny_all	Deny	TCP	*	*	200

Tabla 4.6: Configuración de las reglas del Firewall.

- Bóveda de recuperación de servicios.

  - rsv-hub-uva-001.
  - rsv-dev-uva-001.

Especificación	Valor
SKU	Estándar
Habilitar eliminación suave	Si

Tabla 4.7: Configuración del Bóveda de recuperación de servicios.

Las bóvedas de recuperación de servicios están configuradas según se detalla en la tabla 4.7, cuyo uso está destinado a respaldar, recuperar y administrar datos, aplicaciones y servicios en situaciones de pérdida de datos o desastres dentro de la zona donde residen los recursos. Además, cuentan con políticas asociadas cuya configuración se presenta en la tabla 4.8. Estas políticas están enlazadas a las máquinas virtuales para llevar a cabo copias de seguridad de las mismas.

Especificación	Valor
Zona horaria	Estándar
Hora	23:00
Frecuencia	Diaria
Retención	10 días

Tabla 4.8: Configuración de las políticas de backup.

- Servidor SQL:
  - sql-dev-uva-001. Este Servidor SQL cuenta con la versión 12.

Para el Servidor SQL se ha añadido una regla al Firewall, con las especificaciones de la tabla 4.9. Esto se debe a que el servidor SQL cuenta con una medida de seguridad predeterminada y sin esta regla, el Servidor SQL rechaza cualquier intento de conexión externa.

Especificación	Valor
Nombre	ClientIPAddress
IP salida	86.127.226.2
IP final	86.127.226.2

Tabla 4.9: Configuración de la regla del Firewall para el Servidor SQL.

- Base de datos:
  - inventario.

Esta base de datos es del tipo Microsoft SQL Server, sistema de gestión de bases de datos relacional desarrollado por Microsoft. Su configuración se encuentra detallada en la tabla 4.10.

Especificación	Valor
Ordenación[1]	SQL_Latin1_General_CP1_CI_AS
Número máximo de GB	2
SKU	S0

Tabla 4.10: Configuración de la base de datos.

Una vez configurados todos los recursos, es esencial establecer la conexión entre las diferentes redes virtuales para garantizar una comunicación fluida y segura entre ellas. A continuación se detallan los pasos específicos involucrados en este proceso de conexión de redes virtuales.

4.3.3 Conexión entre las redes virtuales

La comunicación entre las distintas redes virtuales se hace a través de *peerings*[47]. Un *peering* es un mecanismo que permite la comunicación directa y privada entre dos o más redes virtuales en Azure. Es una forma eficiente de conectar las redes virtuales sin tener que utilizar conexiones VPN o ExpressRoute. Cuando se establece un *peering* entre dos VNets, se crea un enlace virtual privado y seguro entre ellas. Este enlace permite que los recursos de ambas redes se comuniquen directamente como si estuvieran en la misma red local.

Para hacer la comunicación entre el Hub y el Spoke se ha implementado un *peering* que conecta la red virtual del Hub con la red virtual del Firewall, de esta forma el Firewall como componente de seguridad de red, crea una barrera entre la red interna privada y la red externa. En este escenario, la red externa hace referencia al hub que es donde se encuentra la máquina virtual que actuará como parte atacante.

Dentro del Spoke, se van a implementar dos *peerings* más. El primero establece la conexión entre la red virtual del Firewall y la red virtual que contiene la máquina virtual que actúa como servidor. El segundo *peering* establece la conexión entre la red virtual del Firewall y la red virtual donde se encuentra el servidor SQL y la base de datos.

4.3.4 Instalación del sistema operativo Kali Linux en la máquina virtual vmhubuva001kali

Una vez se ha definido la configuración de la máquina virtual designada como parte atacante, *vmhubuva001kali*, previo a su despliegue es esencial proceder con la instalación de las herramientas de Kali en esta máquina virtual. Esta tarea se puede llevar a cabo durante la creación del recurso en Terraform mediante la inclusión del parámetro *custom_data()*. Este parámetro posibilita la transferencia de datos personalizados a la máquina virtual en el momento de su creación. En este caso, se emplea un script denominado *install_kali.sh*, el cual contiene los comandos necesarios para instalar todos los componentes pertinentes. Este enfoque garantiza que la máquina virtual esté configurada adecuadamente con las herramientas y utilidades requeridas antes de su implementación en el entorno de la infraestructura definida por código.

```
#!/bin/bash
sudo apt-get update -y
sudo apt install -y kali-linux-core
sudo apt-get install git -y
sudo git clone https://github.com/LionSec/katoolin.git
sudo cp katoolin/katoolin.py /usr/bin/katoolin
sudo chmod ugo+x /usr/bin/katoolin
cd /usr/bin
```

Finalizada la instalación de las herramientas de Kali en la máquina atacante, se completa la configuración necesaria para iniciar los ataques hacia la aplicación alojada en la máquina virtual designada como servidor. Este paso marca el comienzo de la fase de ataques de ciberseguridad, donde las herramientas recién instaladas pueden ser utilizadas para explorar y analizar la seguridad de la aplicación objetivo.

4.3.5 Ejecución de la aplicación en el servidor vmdevuva001servidor

Al definir el recurso de la máquina virtual destinada a alojar la aplicación en los módulos de Terraform, *vmdevuva001servidor*, es importante seguir una serie de pasos específicos. Una vez creada la máquina virtual, se procede a establecer la conexión utilizando las credenciales definidas durante su creación. Este proceso de inicio de sesión permite al usuario transferir tanto la aplicación como el archivo de configuración del servicio a la máquina virtual. A continuación, se presenta el código del archivo denominado *load_execute_server.service*, el cual, como se mencionó previamente, facilita la ejecución de la aplicación como un servicio dentro de la máquina virtual.

```
[Unit]
Description=Colapp Service

[Service]
User = adminUVAServidor

Environment="JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64"

WorkingDirectory=/home/adminUVAServidor/
ExecStart=/usr/bin/java -jar col-app-0.0.1-SNAPSHOT.war
SuccessExitStatus=143
TimeoutStopSec=10
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
```

Una vez se han copiado ambos archivos, se requiere ejecutar de forma remota una serie de comandos, que se muestran a continuación, en la máquina virtual para iniciar la ejecución de la aplicación. Los primeros dos comandos se encargan de otorgar permisos de ejecución tanto a la aplicación como al archivo de configuración que se copiaron previamente. Posteriormente, el tercer comando mueve el archivo de configuración a una nueva ubicación y lo renombra para que el sistema lo reconozca y pueda administrarlo como un servicio. Luego, el cuarto comando actualiza la lista de paquetes disponibles para la instalación y las versiones de cada paquete en el sistema. El quinto comando instala el paquete de desarrollo de Java (JDK)[48] necesario para ejecutar la aplicación. Acto seguido, el sexto comando recarga las unidades de servicio del sistema, garantizando que cualquier cambio realizado en las definiciones de servicio sea reconocido por el sistema. Finalmente, los últimos dos comandos se encargan de habilitar e iniciar el servicio correspondiente, asegurando que la aplicación se ejecute de

manera automática y se encuentre disponible para su uso.

```
    "chmod +x /home/adminUVAServidor/col-app-0.0.1-SNAPSHOT.war",
    "chmod +x /home/adminUVAServidor/load_execute_server.service",
    "sudo mv /home/adminUVAServidor/load_execute_server.service
    /etc/systemd/system/colapp.service",
    "sudo apt-get update -y",
    "sudo apt-get install -y openjdk-11-jdk",
    "sudo systemctl daemon-reload",
    "sudo systemctl enable colapp",
    "sudo systemctl start colapp",
```

Una vez completados estos pasos, la aplicación estaría lista para ser atacada desde la parte atacante, dado que se encuentra en ejecución. Para ofrecer una visión completa de este proceso, en la figura 4.6 se presenta la definición detallada de este recurso, donde figuran los pasos que se han comentado previamente. Esto proporciona una referencia visual y una comprensión más profunda de cómo se ha configurado y desplegado la aplicación en la infraestructura definida por código.

```
resource "azurerm_linux_virtual_machine" "virtual_machine_servidor" {
  count                           = length(var.virtual_machine_servidor)
  name                            = var.virtual_machine_servidor[count.index].name
  resource_group_name             = var.virtual_machine_servidor[count.index].resource_group_name
  location                        = var.location
  size                            = var.virtual_machine_servidor[count.index].size
  admin_username                  = var.virtual_machine_servidor[count.index].admin_username
  network_interface_ids           = var.virtual_machine_servidor[count.index].network_interface_ids
  disable_password_authentication = false
  admin_password                  = var.virtual_machine_servidor[count.index].admin_password

  os_disk {
    caching              = var.virtual_machine_servidor[count.index].caching
    storage_account_type = var.virtual_machine_servidor[count.index].storage_account_type
  }

  source_image_reference {
    publisher = "Canonical"
    offer     = "0001-com-ubuntu-server-jammy"
    sku       = "22_04-lts"
    version   = "latest"
  }
}

resource "null_resource" "app_deployment" {
  # No se requiere ningún atributo específico para el recurso null_resource.
  depends_on = [azurerm_linux_virtual_machine.virtual_machine_servidor]

  connection {
    type     = "ssh"
    user     =
    password =
    host     = azurerm_linux_virtual_machine.virtual_machine_servidor[0].public_ip_address
  }

  #Copiamos la aplicacion
  provisioner "file" {
    source      = "./col-app-0.0.1-SNAPSHOT.war"
    destination = "/home/adminUVAServidor/col-app-0.0.1-SNAPSHOT.war"

  }

  #Copiamos el archivo para ejecutar la aplicacion como servicio
  provisioner "file" {
    source      = "./load_execute_server.service"
    destination = "/home/adminUVAServidor/load_execute_server.service"
  }

  #Ejecutamos la aplicacion como servicio
  provisioner "remote-exec" {
    inline = [
      "chmod +x /home/adminUVAServidor/col-app-0.0.1-SNAPSHOT.war",
      "chmod +x /home/adminUVAServidor/load_execute_server.service", # Dar permisos de ejecución al script
      "sudo mv /home/adminUVAServidor/load_execute_server.service /etc/systemd/system/colapp.service",
```

Figura 4.6: Código de Terraform para la máquina virtual que contiene la aplicación vulnerable.

4.4 Testeo de la Infraestructura

Llegados a este punto el usuario dispone de todos los archivos de configuración de Terraform necesarios para llevar a cabo el despliegue de la infraestructura en la nube de Azure. Sin embargo, antes de proceder con la automatización del despliegue, es esencial realizar un proceso de verificación de cada recurso. Por lo tanto, se inicia la fase de implementación de pruebas unitarias y de integración.

4.4.1 Pruebas unitarias

Las pruebas unitarias, en el contexto de Terraform, se centran en verificar el funcionamiento individual de cada uno de los módulos de Terraform de forma aislada, sin depender de otros recursos externos. De esta forma se comprueba que cada módulo cumple con las especificaciones y requisitos acordados, produciendo los resultados esperados para las diferentes entradas y escenarios de uso.

Al realizar estas pruebas unitarias, los desarrolladores pueden detectar errores o defectos en el código facilitando así una corrección temprana de los mismos. Esta detección de errores contribuye a mantener la integridad y calidad del código, así como garantizar la confiabilidad de los módulos de Terraform antes de su implementación.

Para la ejecución de estas pruebas se emplea el uso de Terratest[49], una biblioteca de Go[50] que simplifica el proceso de pruebas de infraestructura como código utilizando Terraform. Terratest proporciona al usuario una serie de herramientas para crear y ejecutar las pruebas que validan el despliegue de la infraestructura. Completadas estas pruebas, Terratest se encarga de limpiar el entorno, asegurando que las configuraciones de Terraform funcionen correctamente antes del despliegue final de la infraestructura.

En el contexto de esta memoria se presenta un ejemplo de prueba unitaria realizada específicamente para la máquina virtual Kali. Sin embargo, en el anexo A.8 se incluyen las pruebas realizadas a cada uno de los módulos y recursos desplegados mediante Terraform. A continuación se presenta una descripción detallada de los pasos seguidos durante el proceso:

- Creación del archivo go.mod.

```
module test

go 1.21

require (
        github.com/gruntwork-io/terratest v0.28.5
        github.com/stretchr/testify v1.9.0
)

require (
        github.com/davecgh/go-spew v1.1.1 // indirect
        github.com/pmezard/go-difflib v1.0.0 // indirect
        golang.org/x/crypto v0.0.0-20200220183623-bac4c82f6975 // indirect
        golang.org/x/net v0.0.0-20191209160850-c0dbc17a3553 // indirect
        golang.org/x/sys v0.0.0-20200107162124-548cf772de50 // indirect
        gopkg.in/yaml.v3 v3.0.1 // indirect
)
```

- Creación del fichero virtualmachine_test.go

```
package test

import (
        "testing"

        "github.com/gruntwork-io/terratest/modules/terraform"
        "github.com/stretchr/testify/assert"
)

func VirtualMachineTest(t *testing.T) {
        //Elegimos el directorio donde se va ha ejecutar terraform
        fixtureFolder := "../"
        terraformOptions := &terraform.Options{
                TerraformDir: fixtureFolder,
        }

        defer terraform.Destroy(t, terraformOptions) //Destruimos los recursos una
            ↪ vez finalice
        terraform.InitAndApply(t, terraformOptions)  //Realizamos el apply en
            ↪ terraform

        //Almacenamos los outputs que obtenemos en el apply
        instanceID := terraform.Output(t, terraformOptions, "vmhubuva001kali_id")
        instanceName := terraform.Output(t, terraformOptions, "vmhubuva001kali_name")
        publicIPinstance := terraform.Output(t, terraformOptions, "
            ↪ vmdevuva001kali_public_ip_address")
        //Realizamos las comparaciones
        assert.NotEmpty(t, instanceID)
        assert.NotEmpty(t, instanceName)
        assert.NotEmpty(t, publicIPinstance)
}
```

- Ejecución del archivo .go

```
    go test
```

Al finalizar la ejecución del archivo .go, las salidas de los resultados de las pruebas se muestran en la terminal. Estas salidas proporcionan la información detallada sobre los valores de los atributos de la máquina virtual y el resultado de la prueba, como se muestra en la figura 4.7.

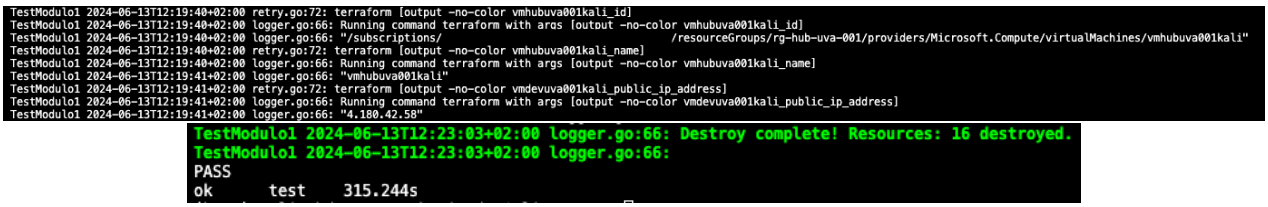


Figura 4.7: Valores del Test Unitario.

Comprobadas las características de cada módulo de forma individual, es importante realizar pruebas de integración para evaluar el comportamiento y funcionamiento de todos los módulos en su conjunto. A continuación, se detallan algunas de las pruebas realizadas para este propósito.

4.4.2 Pruebas de integración

Para validar que la infraestructura se despliega correctamente, se van a realizar una serie de pruebas de integración. Estas pruebas permiten al usuario verificar la interacción y correcta integración de los distintos servicios en la nube, asegurando que funcionan como se espera de ellos. El objetivo de estas pruebas es detectar problemas de comunicación, errores de configuración o dependencias que no han sido resueltas.

Desplegada la infraestructura de prueba, el primer paso es verificar que todos los recursos se han creado correctamente. Para ello, el usuario debe acceder al portal de Azure y, seleccionando la suscripción correspondiente, podrá visualizar todos los recursos desplegados, tal como se muestra en la figura 4.8.

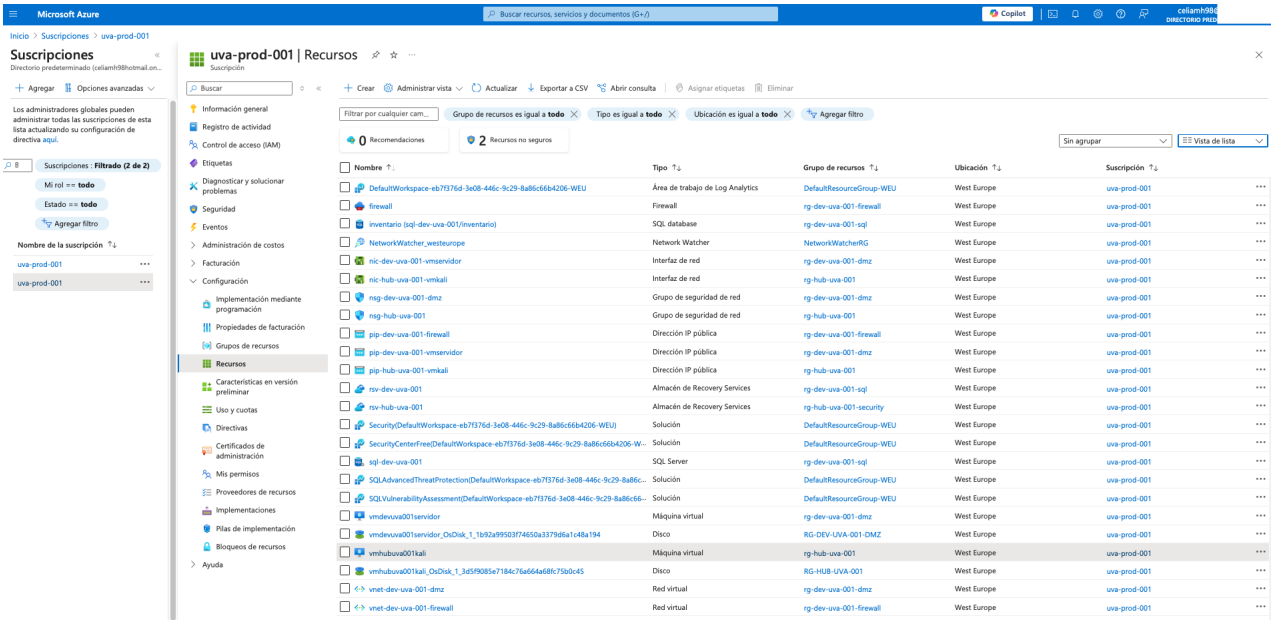


Figura 4.8: Recursos desplegados.

Una vez comprobados que todos los recursos están desplegados en el portal de Azure, el siguiente paso es verificar que tienen las configuraciones establecidas en los archivos de configuración de Terraform. En esta infraestructura se han definido dos reglas en los grupos de seguridad de red y en el Firewall. Como se mencionó en la sección de la configuración de los recursos (4.3.2), estas reglas permiten la conexión SSH por el puerto 22 y la conexión de la ejecución de la aplicación por el puerto 8080 en el servidor. Ambas reglas deben estar presentes en ambos recursos para evitar problemas de conexión. Como se muestra en la imagen 4.9, tanto los grupos de seguridad de red como el firewall tienen asignadas estas dos reglas.

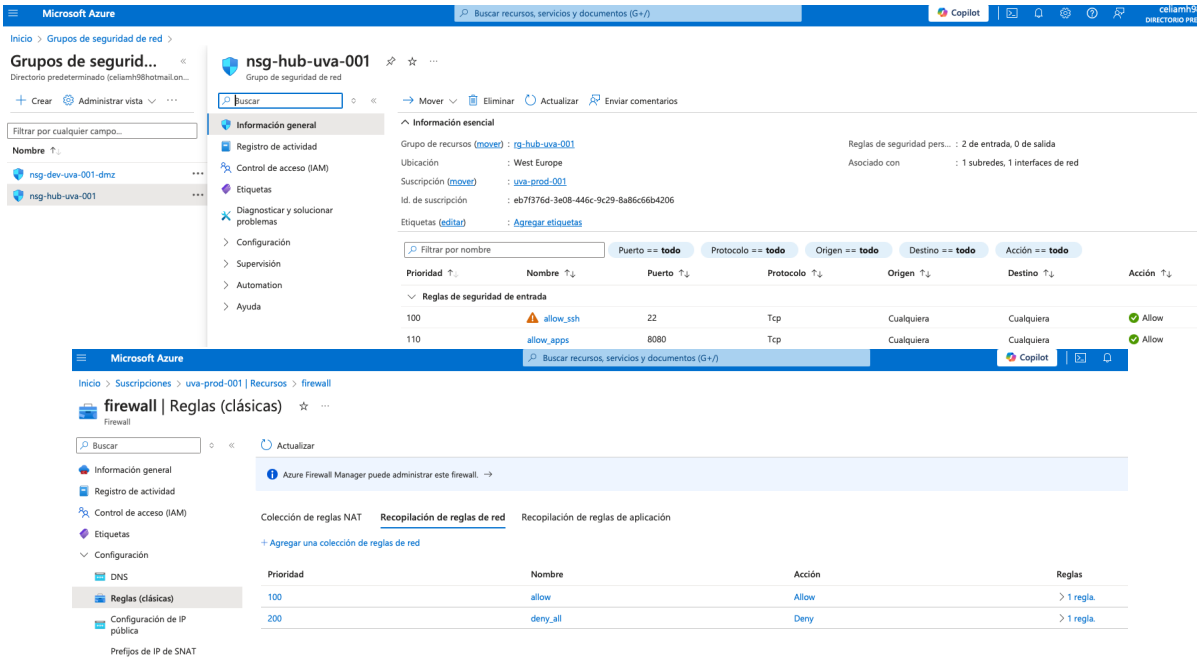


Figura 4.9: Reglas asignadas a los Grupos de seguridad de red y al Firewall.

Otra prueba realizada que permite verificar el correcto funcionamiento es la conexión entre la máquina virtual Kali Linux hacia el servidor que aloja la aplicación. Esta conexión es importante para llevar a cabo los ataques planificados. Para este propósito, el usuario accede a Network Watcher en el portal de Azure, una herramienta configurada automáticamente una vez que la infraestructura está desplegada. En la sección "Comprobación del flujo de IP", se completan los campos requeridos, incluida la dirección IP de la máquina virtual del servidor y el puerto 22 para el acceso mediante SSH. Tras esto, se presiona el botón "Comprobar flujo de IP", y el resultado indica si el acceso está permitido o no. La figura 4.10 muestra como se llevó a cabo esta prueba.

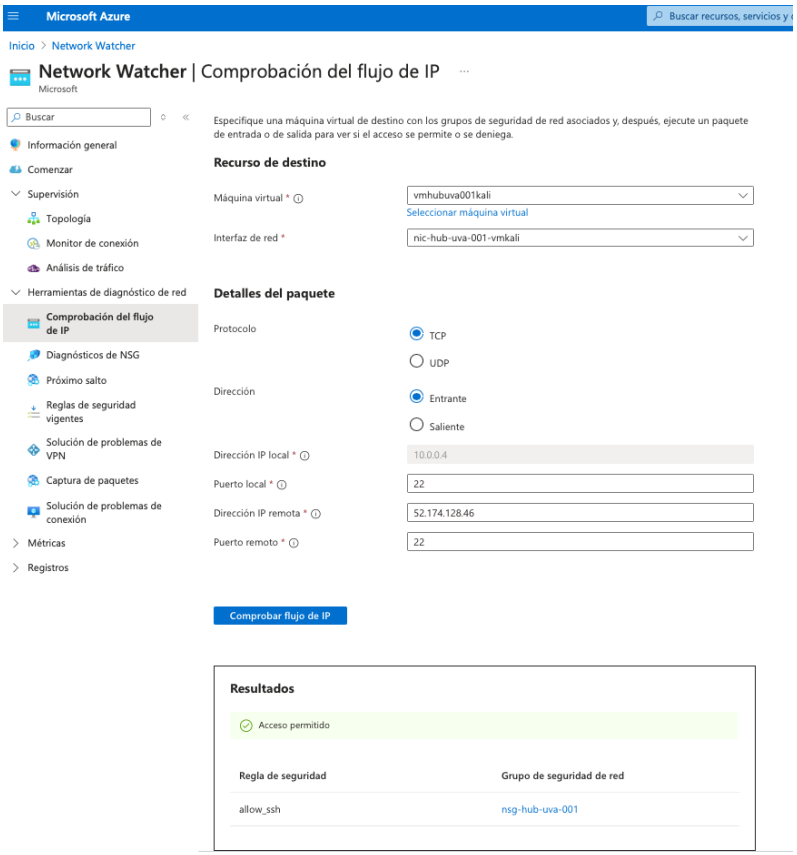


Figura 4.10: Conexión desde la máquina virtual Kali al servidor.

Estas son algunas pruebas que se han realizado para comprobar que la infraestructura se ha desplegado correctamente y funciona según lo esperado. A continuación, se proporciona una descripción detallada de los pasos necesarios para automatizar el proceso de despliegue de la infraestructura.

4.5 Automatización con Azure DevOps

Una vez que se ha creado la infraestructura, el siguiente paso es proceder con la automatización. Sin embargo, para lograrlo, es necesario configurar diversos elementos que habilitarán al pipeline para llevar a cabo el despliegue de dicha infraestructura.

4.5.1 Almacenamiento de Claves

Como se mencionó en el apartado 4.1 es necesario contar con un archivo *terraform.tfvars* para cada proyecto de Terraform. Este archivo contiene las claves que permiten la autenticación de Terraform en la cuenta de Azure. Sin embargo, estos valores son sensibles y no deben ser conocidos ni divulgados por los desarrolladores. Con el fin de evitar este problema, Azure DevOps ofrece la opción de almacenar archivos de forma segura, los cuales son encriptados para salvaguardar su contenido. Estos archivos seguros se utilizan en las tareas de los pipelines de Azure DevOps durante el proceso de compilación, pruebas o despliegue de aplicaciones. Aunque sean accesibles desde los pipelines, sólo los usuarios con los permisos adecuados pueden subir, descargar o acceder a su contenido.

Para este trabajo de fin de grado se ha creado un archivo llamado *azurerm.tfbackend* que contie-

ne todos los valores sensibles empleados en el código de Terraform. Dichos valores son las claves de acceso a Azure, los nombres y contraseñas de los servidores, el nombre y contraseña del servido SQL y el nombre de la base de datos. Una vez que este archivo ha sido creado y almacenado en Azure DevOps, tal como se muestra en la figura 4.11, es fundamental tener presente que se deben de otorgar los permisos para que el pipeline tenga acceso a este archivo y pueda realizar todas las tareas descritas.

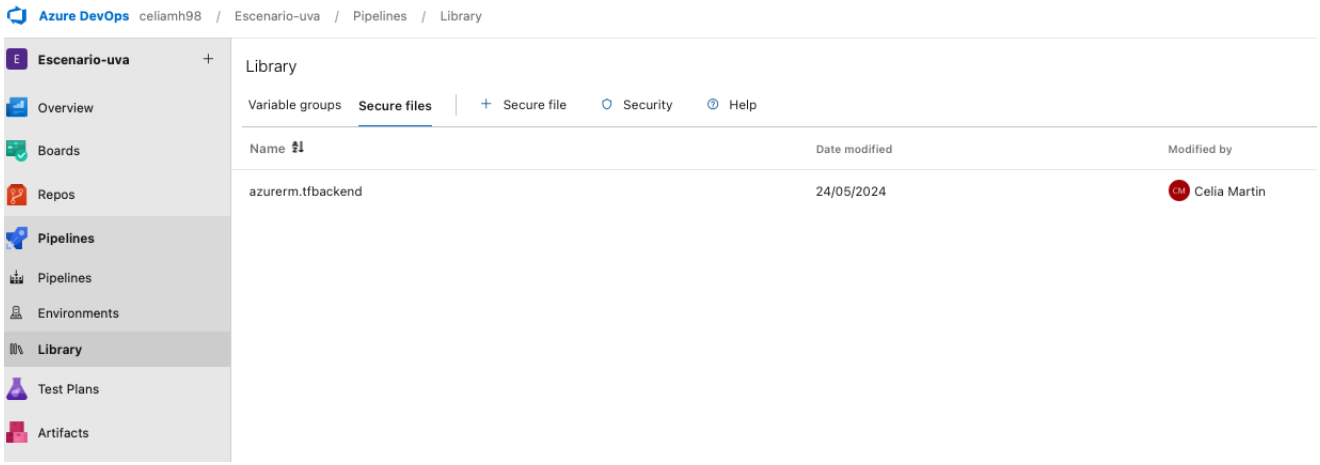


Figura 4.11: Archivo seguro

4.5.2 Creación de la cuenta de almacenamiento para almacenar el estado

Para llevar a cabo un despliegue eficiente y seguro de la infraestructura con Terraform, es importante la creación de una cuenta de almacenamiento dedicada para gestionar el estado de Terraform.

Cuando se utiliza Terraform para aprovisionar recursos en la nube, el estado de la infraestructura se guarda en local de forma predeterminada. Sin embargo, esta práctica puede ocasionar problemas en entornos donde múltiples desarrolladores están haciendo despliegues y trabajan con la misma infraestructura. Al optar por almacenar el estado de Terraform en una cuenta de almacenamiento en Azure, se asegura que la información sobre los recursos desplegados esté disponible para el resto de usuarios. Esto no solo previene conflictos durante los despliegues, sino que también evita la duplicidad de los recursos. Si un usuario crea un recurso y su estado no está almacenado en la nube, otro usuario que intente desplegar la infraestructura no detectará que este recurso ha sido creado previamente, ocasionando errores y redundancias innecesarias.

Aunque almacenar el estado proporciona mayor eficiencia y seguridad, es importante bloquear el estado de Terraform. Este bloqueo evita que varios usuarios modifiquen el estado de forma simultanea garantizando la integridad de la configuración y permitiendo a los usuarios realizar cambios de manera secuencial y organizada.

Los pasos que hay que seguir para la creación de la cuenta de almacenamiento son los siguientes:

1. En el portal de Azure, acceder a las cuentas de almacenamiento.

2. Crear una cuenta de almacenamiento con las siguientes características[2]:

[2]El nombre de la cuenta de almacenamiento debe de ser único, por lo que el nombre 'tfstate' puede ocasionar errores. Para evitar este problema, se recomienda añadir un valor aleatorio al nombre, lo que garantiza su singularidad.

Especificación	Valor
Nombre	tfstate247
Tipo	Estándar
Tipo de replicación	LRS
Tipo de acceso	Privado

Tabla 4.11: Características de la cuenta de almacenamiento de estado.

3. Crear un contenedor privado dentro de la cuenta de almacenamiento, cuyo nombre es *tfstate*.
4. Añadir el bloque del backend con los valores de los recursos que se han creado en los pasos anteriores dentro del archivo *provider.tf* como se muestra en la figura 4.12.

Figura 4.12: Bloque Backend.

Finalizados los pasos anteriores, el proceso avanza hacia la automatización del despliegue, un paso crucial para completar el flujo de trabajo desde el desarrollo hasta la entrega de software. Para ello el usuario debe de comenzar con la creación del *pipeline* como se muestra en la siguiente sección.

4.5.3 Creación del pipeline

Los *pipelines*[51] de integración continua/despliegue continuo (CI/CD) permiten la construcción, realización de pruebas y automatización de tareas repetitivas facilitando la entrega rápida y confiable de las aplicaciones. Un *pipeline* es un *script* en formato *.yaml*[52] que contiene una serie de pasos automatizados que conducen al código desde su desarrollo inicial hasta su implementación.

Azure DevOps ofrece la posibilidad de crear dentro de su plataforma los *pipelines*, pero antes de empezar a escribir el código hay que instalar Terraform. Para ello el usuario debe de entrar en *Organization Settings* y en el apartado de extensiones dar en el botón de *Browse marketplace* como se muestra en la figura 4.13. A continuación se abre una página en internet donde en el buscador el usuario debe escribir terraform y elegir la opción de microsoft.com que aparece verificada con un *tick* azul.

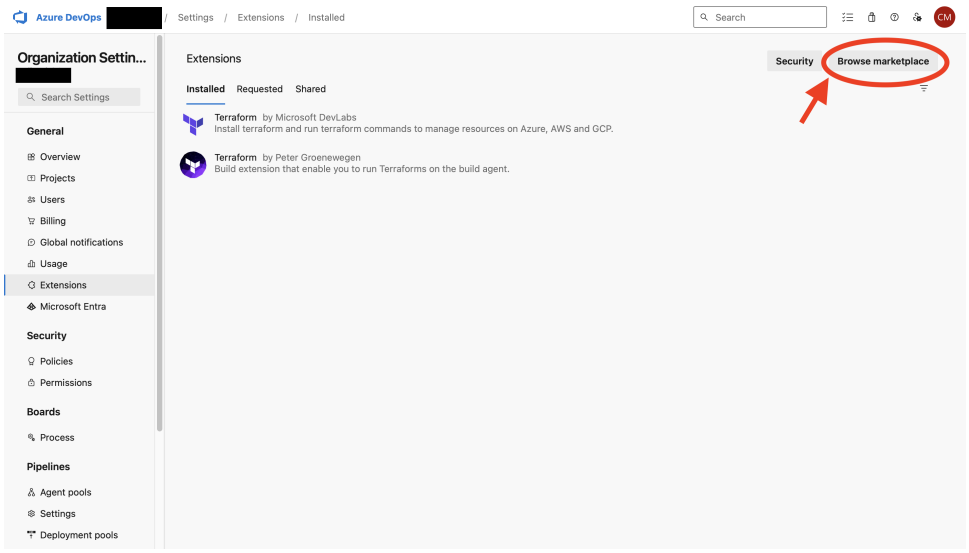


Figura 4.13: Instalar Terraform en Azure DevOps.

Una vez instalado el agente de Terraform en Azure DevOps se puede comenzar a escribir el *pipeline*. Para ello el usuario debe dirigirse al apartado de *Pipelines* dentro de su proyecto, dar al botón *New pipeline* y elegir la opción Azure Repos Git, donde hay que seleccionar el repositorio en el que se va a crear. Concluidos estos pasos, aparecerá un pipeline de ejemplo como el que se muestra en la figura 4.14, en el que hay que borrar tanto el apartado de *steps* como el de *script*.

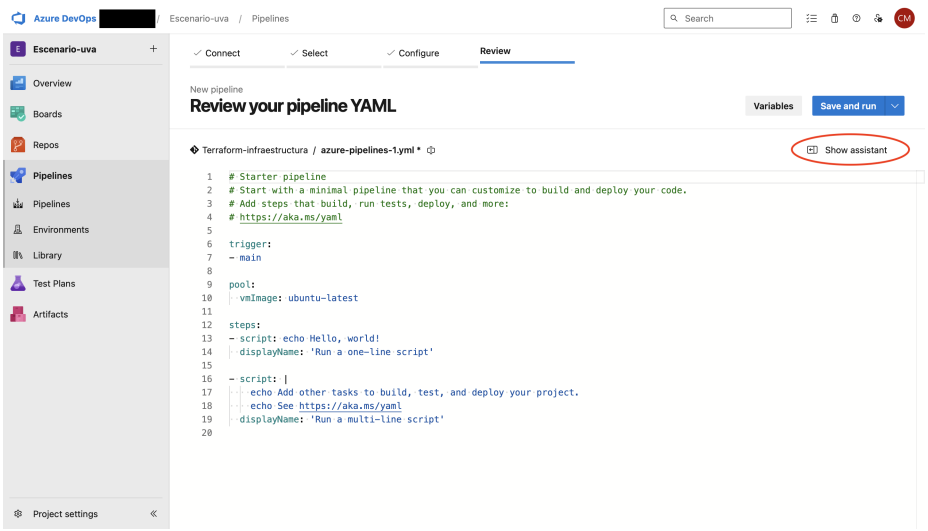


Figura 4.14: Pipeline vacío.

Llegados a este punto, el usuario puede empezar a crear la secuencia de comandos que desea que contenga su *pipeline*. Pulsando el botón de *Show assistant* como se muestra en la figura 4.14 se abre un buscador donde hay que elegir la extensión de Terraform que se instaló anteriormente. Gracias al agente instalado, el usuario puede crear las tareas de forma sencilla seleccionando en el apartado de comandos aquellos que desea ejecutar para automatizar el despliegue de la infraestructura, siendo el resultado algo similar al que se muestra en la figura 4.15.

azure-pipelines.yml

Contents Highlight changes

```yaml
# Starter pipeline

trigger:
- main #esto indica que ejecutara el pipeline cada vez que se incluya algo nuevo en la rama main. Nosotros trabajamos en otra rama y cuand

pool:
  vmImage: ubuntu-latest


variables:
- group: Escenario_ciberseguridad_TFG_Celia Martin

stages:
  - stage: tfvalidate #Con esto validamos que no haya errores, si falla se cerrará sin ejecutarse
    jobs:
      - job: validation
        continueOnError: false
        steps:

          - task: DownloadSecureFile@1 #Descargamos el archivo que contiene los secretos y variables confidenciales
            name: prod
            displayName: Download backend config file
            inputs:
              secureFile: "azurerm.tfbackend"

          - task: TerraformInstaller@1
            inputs:
              terraformVersion: 'latest'  #con esto instalamos la version de terraform que tenemos en nuestro provider

          - task: TerraformTaskV4@4
            displayName: terraforminit
            inputs:
              provider: 'azurerm'
              command: 'init'
              backendServiceArm: 'infraestructura-uva'
              backendAzureRmResourceGroupName: 'tfstate'
              backendAzureRmStorageAccountName: 'tfstate247'
              backendAzureRmContainerName: 'tfstate'
              backendAzureRmKey:

          - task: TerraformTaskV4@4
            displayName: terraformvalidate
            inputs:
              provider: 'azurerm'
              command: 'validate'

          - task: TerraformTaskV4@4
            displayName: terraformplan
            inputs:
              provider: 'azurerm'
              command: 'plan'
              workingDirectory: '$(System.DefaultWorkingDirectory)'
```

Figura 4.15: Pipeline de ejemplo.

Ahora llega el momento de ejecutar el *pipeline*, pero aparece un error de ejecución. Este error es debido a que se va a ejecutar un pipeline para un proyecto privado en una cuenta gratuita. Para solucionar el error, el usuario debe rellenar el formulario de solicitud de paralelismo que aparece en esta página web (Azure DevOps Parallelism Request). De dos a tres días, Microsoft procede a resolver la solicitud y una vez aprobada ya se puede ejecutar el pipeline.

Una vez completados estos pasos, el usuario puede ejecutar su pipeline en Azure DevOps. La plataforma proporciona al usuario una interfaz intuitiva que le permite monitorear el progreso y estado de cada paso y tareas definidos en el pipeline como se muestra en la figura 4.16. Además, ofrece la capacidad de revisar y abordar cualquier problema que pueda surgir durante la ejecución, proporcionando información detallada sobre los resultados obtenidos en cada paso del proceso. Esta visibilidad y control facilitan la gestión eficiente del pipeline y garantizan una ejecución exitosa del flujo de trabajo definido.

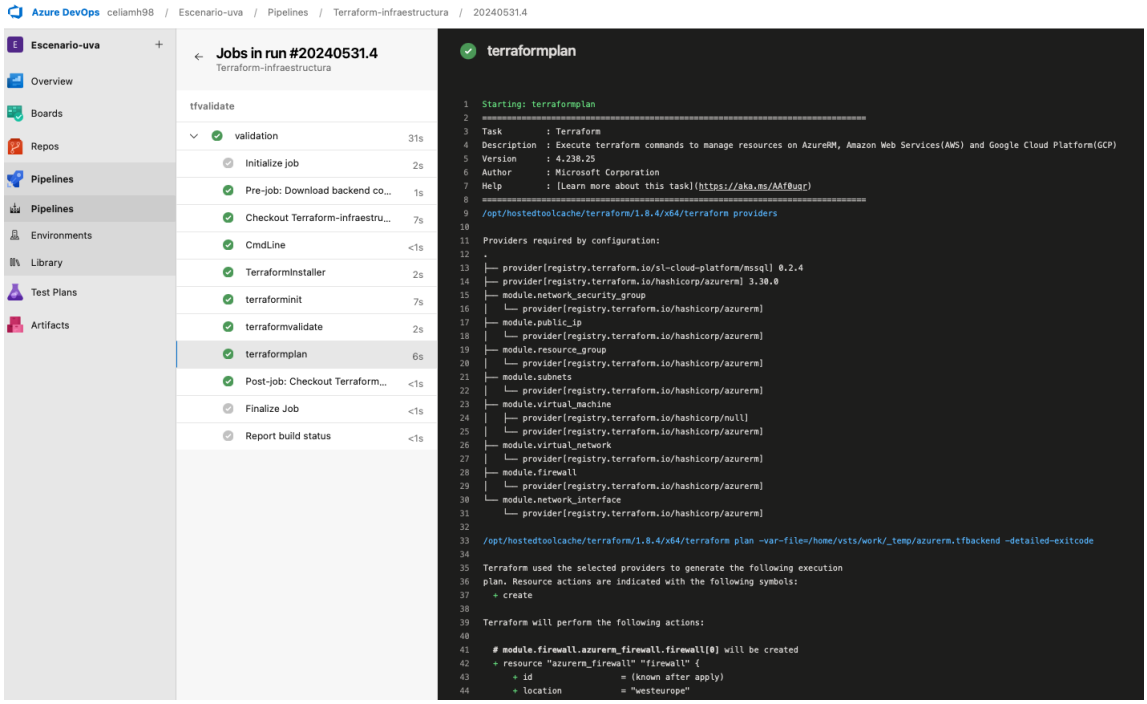


Figura 4.16: Ejecución del pipeline.

Finalizados todos estos pasos, incluyendo el desarrollo de la infraestructura, la verificación de su correcto funcionamiento y la automatización de los despliegues, la infraestructura está lista para ser desplegada. Con todo preparado, es posible iniciar las pruebas de ciberseguridad en el escenario diseñado, permitiendo evaluar los diversos ataques que se pueden realizar a la aplicación en ejecución y así identificar las vulnerabilidades existentes.

Capítulo 5

Laboratorio de Ciberseguridad

Con la infraestructura desplegada y la aplicación en funcionamiento, es el momento de llevar a cabo un conjunto de pruebas de ciberseguridad enfocadas específicamente en la aplicación. Para estudiar la seguridad se va ha realizar una investigación de los puertos abiertos en el servidor que la alberga y la detección de posibles vulnerabilidades de inyección SQL. Es importante destacar que, si bien la identificación de estas vulnerabilidades es fundamental, la resolución de las mismas no forma parte de la planificación de este trabajo de fin de grado. El objetivo principal es evaluar el estado actual de la seguridad de la aplicación y del servidor, proporcionando una base sólida para futuras acciones correctivas y mejoras.

5.1 Listado de puertos abiertos

El primer paso es identificar los puertos que tiene abiertos el servidor que aloja la aplicación. Este análisis va a permitir identificar cualquier puerto expuesto que pueda representar una vulnerabilidad potencial. Para llevar a cabo esta tarea, el usuario debe de acceder a la máquina virtual donde está instalado Kali Linux mediante una conexión SSH. Una vez dentro, se procede a investigar el listado de puertos abiertos en el servidor que alberga la aplicación.

Los pasos que debe de seguir el usuario son:

- Actualizar todos los paquetes y descargar el paquete nmap.

```
sudo apt update
sudo apt install nmap
```

- Listado de puertos abiertos.

```
nmap -Pn 20.4.16.77
```

En este caso el servidor que contiene la aplicación tiene la dirección IP pública 20.4.16.77. En la figura 5.1 se muestra el listado de puertos que tiene abiertos.

```
[adminUVAKali@vmhubuva001kali:~$ nmap -Pn 20.4.16.77
Starting Nmap 7.80 ( https://nmap.org ) at 2024-06-12 14:16 UTC
Nmap scan report for 20.4.16.77
Host is up (0.0035s latency).
Not shown: 998 filtered ports
PORT     STATE SERVICE
22/tcp   open  ssh
8080/tcp open  http-proxy

Nmap done: 1 IP address (1 host up) scanned in 4.86 seconds
```

Figura 5.1: Listado de puertos abiertos.

Como se muestra en la figura, se ve la presencia de dos puertos abiertos en el servidor. El primero de ellos es el puerto 22, que permite la conexión mediante el protocolo SSH hacia la máquina virtual. Esta conexión ofrece un acceso remoto seguro garantizando así la confidencialidad y la integridad de la comunicación entre el cliente y el servidor.

El segundo puerto que está abierto es el puerto 8080 y alberga la ejecución de la aplicación web. Este puerto se suele utilizar normalmente para servidores web alternativos o para ejecutar aplicaciones web que se encuentran adicionalmente ejecutándose en el puerto estándar HTTP(80). El puerto 8080 no es considerado un puerto estándar y su uso implica la implementación de medidas adicionales de seguridad, para proteger la aplicación web y los datos que maneja, de posibles vulnerabilidades y amenazas externas.

5.2 Vulnerabilidades de SQL Injection

Los ataques de inyección SQL (SQL Injection)[53] son un tipo de ciberataque que explota vulnerabilidades en una aplicación mediante la inserción de código SQL maliciosos en una base de datos subyacente. Este tipo de ataque es ocasionado cuando una aplicación no comprueba de forma correcta la entrada del usuario, permitiendo así que un atacante inserte código SQL arbitrario en una consulta SQL.

El objetivo de este estudio consiste en analizar las vulnerabilidades que tiene la aplicación alojada en el servidor mediante la ejecución de consultas SQL con comandos no previstos por el desarrollador de la aplicación, alterando así el comportamiento de la consulta SQL. Esta aplicación cuenta con una página donde el usuario puede iniciar sesión introduciendo su correo electrónico y la contraseña. Si el campo de correo electrónico es vulnerable a la inyección SQL, al introducir **aa'** (como se muestra en la figura 5.2), el apóstrofe que se añade al final hace que se rompa la cadena y se produzca un error de sintaxis.

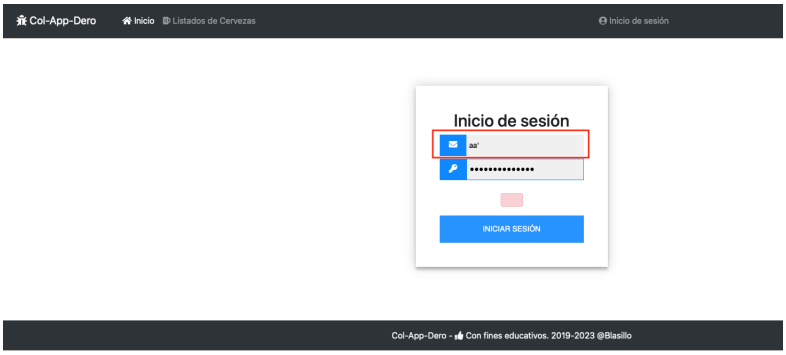


Figura 5.2: Ejemplo de ataque de inyección SQL.

Otro tipo de ataque común es el uso de la consulta **'OR 1=1'**, una cláusula lógica que siempre resulta ser verdadera, lo que convierte toda la condición **'WHERE'** en verdadera. Al emplear ambas consultas, se genera una página de error, como se muestra en la figura 5.3, permitiendo al usuario verificar que la aplicación es vulnerable.

Figura 5.3: Página de error.

Una vez el usuario ha podido comprobar que la aplicación es vulnerable a los ataques de inyección SQL, el atacante tiene como objetivo final obtener una tabla con todos los usuarios que tiene la base de datos almacenados. Se parte de la siguiente consulta que se ha comprobado que es vulnerable a SQL Injection:

```
SELECT * FROM USUARIOS
WHERE EMAIL=' + login + '
AND PASSWORD=' + password \+ '
```

A continuación se muestran los comandos que el usuario debe introducir en el buscador de la pantalla del Listado de Cervezas junto con las respuestas obtenidas:

- Anular la parte del WHERE.

```
' AND 1=0 -- -
```

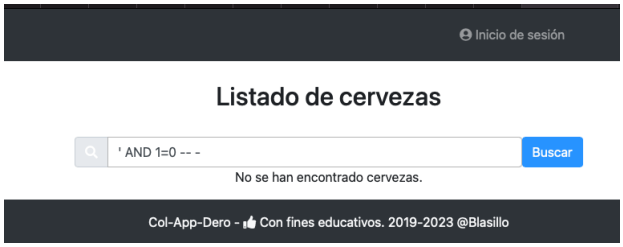


Figura 5.4: Anular password.

- Saber el número de columnas que tiene la tabla. Para ello deberá ir incrementando el número hasta que salte la excepción. En este caso, se ha comprobado que son 7.

```
' AND 1=0 ORDER BY 7-- -
```

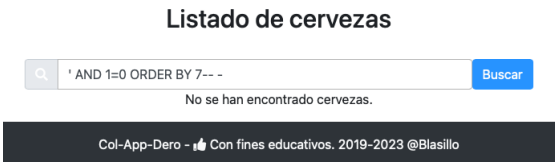


Figura 5.5: Obtener el número de filas.

- Emplear el uso de UNION que permite la concatenación de dos consultas.

```
' AND 1=0 UNION SELECT NULL,NULL,NULL,NULL,NULL,NULL, NULL-- -
```

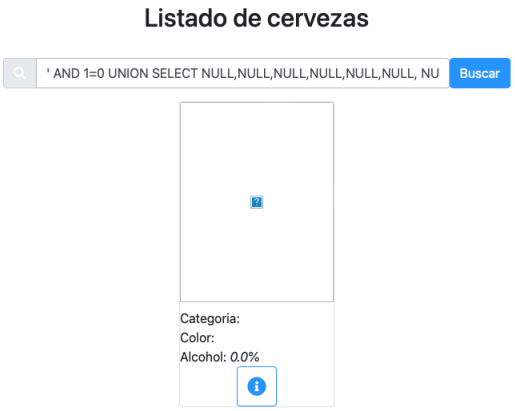


Figura 5.6: Concatenar dos consultas.

- Investigar en que posición de los campos mapea cada una de las columnas.

```
' AND 1=0 UNION SELECT 1,2,3,4,5,6,7 -- -
```

Figura 5.7: Campo que mapea las columnas.

- Descubrir cual es el nombre de las bases de datos que contiene.

```
' AND 1=0 UNION SELECT 1,SCHEMA_NAME,3,4,5,6,7 FROM INFORMATION_SCHEMA.
    ↪ SCHEMATA-- -
```

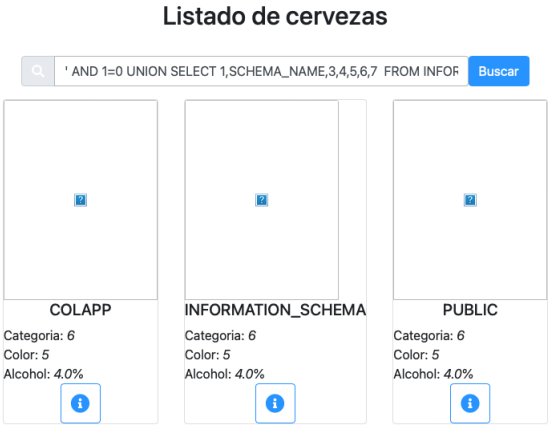


Figura 5.8: Nombre de las bases de datos.

- Obtener el nombre del resto de las tablas.

```
' AND 1=0 UNION SELECT NULL,TABLE_NAME,NULL,NULL,NULL,NULL,NULL FROM
    ↪ INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA='COLAPP'-- -
```

Figura 5.9: Nombre de cada una de las tablas.

- Investigar cuales son los nombres de las columnas de la tabla de datos que queremos obtener la información.

```
' AND 1=0 UNION SELECT NULL,COLUMN_NAME,NULL,NULL,NULL,NULL,NULL FROM
    ↪ INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME=USUARIOS-- -
```

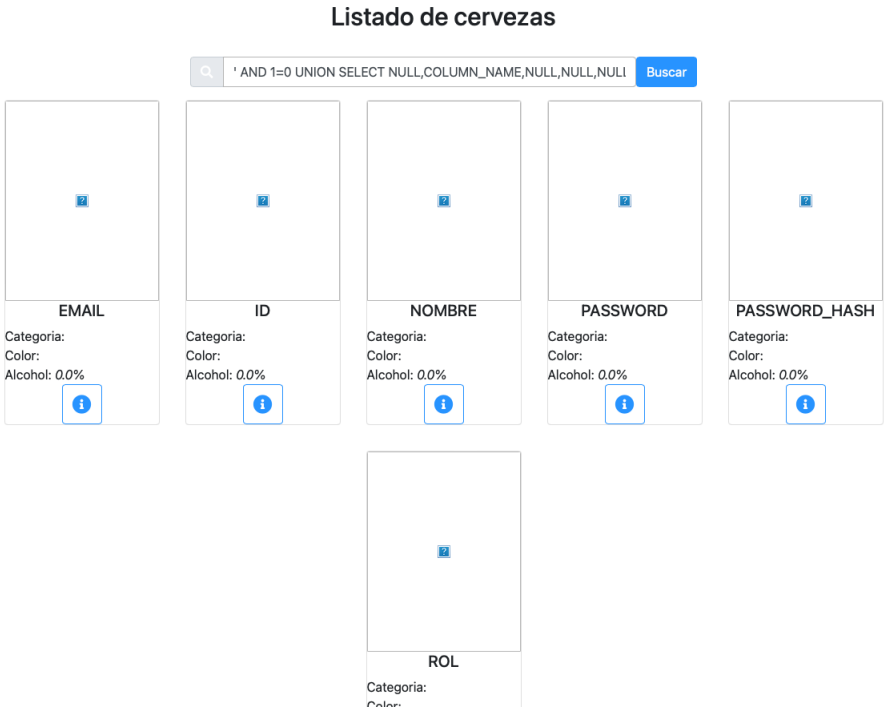


Figura 5.10: Nombre de las columnas de la tabla Usuarios.

- Obtener el listado con los nombres de los usuarios que se encuentran almacenados en tabla.

```
' AND 1=0 UNION SELECT NULL,NOMBRE,NULL,NULL,NULL,NULL,NULL FROM USUARIOS --
    ↪ -
```

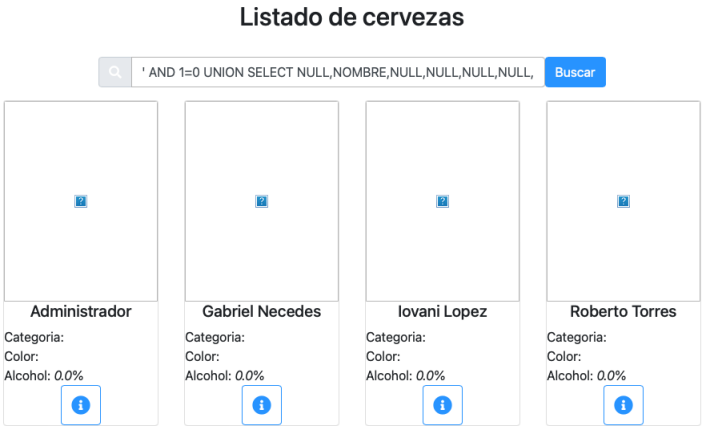


Figura 5.11: Nombre de los usuarios.

Como se puede ver en la figura 5.11 la base de datos contiene cuatro nombres de usuario almacenados. Esto demuestra que el usuario ha sido capaz de verificar la vulnerabilidad de la aplicación a ataques de inyección SQL. Aprovechando esta vulnerabilidad, el usuario ha logrado anular la consulta original y ejecutar una consulta adicional para extraer los datos que estaba buscando.

Capítulo 6

Conclusiones

El uso de la infraestructura como código ha sido fundamental en este trabajo de fin de grado para llevar a cabo el despliegue de la infraestructura en la nube de Azure. Utilizando la herramienta Terraform, se ha logrado automatizar y estandarizar el despliegue de los recursos necesarios, siguiendo estrictamente los requisitos previamente analizados. La elección de esta herramienta se basa en su capacidad para facilitar la replicación y escalabilidad de la infraestructura en entornos cloud, asegurar la consistencia y minimizar los posibles errores humanos. Además, permite al desarrollador desplegar la infraestructura para realizar pruebas y luego destruirla fácilmente, reduciendo así los costes operativos.

Adicionalmente, se ha implementado un proceso de despliegue automatizado mediante el uso de pipelines en Azure DevOps. Este enfoque garantiza que los recursos se desplieguen de manera consistente y eficiente en el entorno de Azure con solo ejecutar el pipeline, eliminando la necesidad de que el desarrollador o el cliente utilicen directamente Terraform. Además, el uso de Azure DevOps es fundamental para almacenar todo el código desarrollado en repositorios, permitiendo así una gestión centralizada y un historial de cambios bien documentado.

La infraestructura ha sido diseñada específicamente para llevar a cabo ataques de ciberseguridad. Cada componente ha sido configurado para interactuar de manera óptima con los demás, garantizando un entorno adecuado para ejecutar y evaluar diversas técnicas de ataque, como la inyección SQL. Los recursos que conforman esta infraestructura son los siguientes:

- Cinco grupos de recursos.
- Cuatro redes virtuales.
- Cuatro subredes.
- Dos grupos de seguridad de red.
- Dos máquinas virtuales Linux.
- Tres direcciones IP públicas.
- Dos tarjetas de interfaz de red.
- Un Firewall.
- Un Servidor SQL.

- Una base de datos SQL.

- Dos bóvedas de recuperación de servicios.

- Peerings entre las redes virtuales.

Esta infraestructura está diseñada para poder realizar pruebas de ciberseguridad, constituyendo un escenario ideal para laboratorios donde los alumnos pueden ejecutar ataques en un entorno controlado y obtener los resultados esperados. La máquina virtual con el sistema operativo Kali Linux actúa como la parte atacante, mientras que la máquina virtual que aloja la aplicación es la parte atacada, simulando la infraestructura de una empresa. De esta forma, la parte atacante puede realizar los ataques a la aplicación y extraer datos almacenados en la aplicación, como los nombres de los usuarios registrados.

Como trabajo futuro, se plantea la ampliación de la infraestructura mediante la integración de Microsoft EntraID, anteriormente conocido como Azure Active Directory. Esta adición permitiría una gestión más centralizada y segura de los usuarios, creando un entorno similar al de una empresa. Todos estos usuarios podrían ser los alumnos de la clase, cada uno con el rol de contribuidor sobre la suscripción de Azure. Esto les permitiría desplegar la infraestructura, realizar los ataques necesarios para su estudio y posteriormente destruirla.

Bibliografía

[1] On-premise infraestructura, [en línea]. https://preteco.com/que-es-la-infraestructura-on-premise/. [Último acceso: 10 de Junio de 2024].

[2] John mccarthy, [en línea]. https://es.wikipedia.org/wiki/John_McCarthy_(cientfico). [Último acceso: 19 de Junio de 2024].

[3] Salesforce, [en línea]. https://www.salesforce.com/news/stories/the-history-of-salesforce/. [Último acceso: 12 de Junio de 2024].

[4] Marcbenioff, [en línea]. https://es.wikipedia.org/wiki/Marc_Benioff. [Último acceso: 18 de Junio de 2024].

[5] Joe Peppard. Customer relationship management (crm) in financial services. *European Management Journal*, 18(3):312–327, 2000.

[6] Amazon y su evolución de aws, [en línea]. https://es.wikipedia.org/wiki/Amazon_Web_Services. [Último acceso: 10 de Junio de 2024].

[7] Amazon web services portal, [en línea]. https://aws.amazon.com/es/?nc2=h_lg. [Último acceso: 20 de Junio de 2024].

[8] Azure portal, [en línea]. https://azure.microsoft.com/es-es/. [Último acceso: 20 de Junio de 2024].

[9] Google cloud, [en línea]. https://cloud.google.com/?hl=es. [Último acceso: 20 de Junio de 2024].

[10] Anthony T Velte Toby J Velte and Ph D Robert Elsenpeter. *Cloud computing*. 2010.

[11] Nelson R Rodríguez, María Antonia Murazzo, Susana Beatriz Chávez, and Miguel José Guevara. Arquitectura de cloud computing híbrida basada en tecnología open source. In *XX Congreso Argentino de Ciencias de la Computación (Buenos Aires, 2014)*, 2014.

[12] Cameron Fisher. Cloud versus on-premise computing. *American Journal of Industrial and Business Management*, 8(9):1991–2006, 2018.

[13] Hauke Hansen, Wolfgang Huhn, Olivier Legrand, Daniel Steiners, and Thomas Vahlenkamp. *CAPEX Excellence: optimizing fixed asset investments*. John Wiley and Sons, 2009.

[14] Kief Morris. *Infrastructure as code*. O'Reilly Media, 2020.

[15] Terraform tool, [en línea]. https://www.terraform.io. [Último acceso: 19 de Junio de 2024].

[16] Linux so, [en línea]. https://www.redhat.com/es/topics/linux. [Último acceso: 10 de Junio de 2024].

[17] Kali linux, [en línea]. `https://www.kali.org`. [Último acceso: 10 de Junio de 2024].

[18] Charles G Cobb. Agile project management. *Making sense of agile project management: Balancing control and agility. New Jersey: John Wiley & Sons*, pages 101–130, 2011.

[19] Git tool, [en línea]. `https://git-scm.com`. [Último acceso: 21 de Junio de 2024].

[20] Mohammad Rizky Pratama and Dana Sulistiyo Kusumo. Implementation of continuous integration and continuous delivery (ci/cd) on automatic performance testing. In *2021 9th International Conference on Information and Communication Technology (ICoICT)*, pages 230–235. IEEE, 2021.

[21] Kirill Shirinkin. *Getting Started with Terraform*. Packt Publishing Ltd, 2017.

[22] Hashicorp, [en línea]. `https://www.hashicorp.com`. [Último acceso: 19 de Junio de 2024].

[23] Kubernetes, [en línea]. `https://kubernetes.io/es/`. [Último acceso: 21 de Junio de 2024].

[24] Hcl, [en línea]. `https://developer.hashicorp.com/terraform/language/syntax/configuration`. [Último acceso: 21 de Junio de 2024].

[25] Json, [en línea]. `https://developer.hashicorp.com/terraform/language/syntax/json`. [Último acceso: 21 de Junio de 2024].

[26] Terraform registry, [en línea]. `https://registry.terraform.io`. [Último acceso: 20 de Junio de 2024].

[27] Azure devops, [en línea]. `https://azure.microsoft.com/es-es/products/devops`. [Último acceso: 20 de Junio de 2024].

[28] Microsoft, [en línea]. `https://www.microsoft.com/es-es/`. [Último acceso: 21 de Junio de 2024].

[29] Azure historia, [en línea]. `https://techcommunity.microsoft.com/t5/educator-developer-blog/the-history-of-microsoft-azure/ba-p/3574204`. [Último acceso: 21 de Junio de 2024].

[30] Beatriz Helena Díaz Pinzón, José Santiago Gómez Medina, Juan David García González, Hjalmar Arturo Melo Román, and Fabián Enrique Sanabria Villamizar. Contribución de las iniciativas de tecnologías de la información en las organizaciones: una revisión de la literatura. *Innovar*, 27(66):41–55, 2017.

[31] Jordi Salazar and Santiago Silvestre. Internet de las cosas. *Techpedia. České vysoké učení technické v Praze Fakulta elektrotechnická*, 2016.

[32] Juan Rolando Ramos Ali. Infraestructura como servicio (iaas). *Revista de Información, Tecnología y Sociedad*, page 106, 2012.

[33] Leonardo De Seta. ¿ qué es una plataforma como servicio (paas)? *Internet: https://dosideas.com/noticias/actualidad/504-ique-es-una-plataforma-comoservicio-paas*, 2009.

[34] Fernando Maldonado. Saas: Un mercado en plena expansión. IDC, 2009.

[35] Linus torvalds, [en línea]. `https://es.wikipedia.org/wiki/Linus_Torvalds#`. [Último acceso: 10 de Junio de 2024].

[36] Github, [en línea]. `https://github.com`. [Último acceso: 25 de Junio de 2024].

[37] Gitlab, [en línea]. `https://about.gitlab.com`. [Último acceso: 25 de Junio de 2024].

[38] Jira tool, [en línea]. `https://www.atlassian.com/es/software/jira`. [Último acceso: 20 de Junio de 2024].

[39] Martin F Krafft. *The Debian system: concepts and techniques*. No Starch Press, 2005.

[40] Raphael Hertzog, Jim O'Gorman, and Mati Aharoni. Kali linux revealed. *Mastering the Penetration Testing Distribution*, 2017.

[41] Historia kali linux, [en línea]. `https://kali-linux.neocities.org/linux`. [Último acceso: 20 de Junio de 2024].

[42] GNU General Public License. Gnu general public license. *Retrieved December*, 25:2014, 1989.

[43] Ganttproject, [en línea]. `https://www.ganttproject.biz`. [Último acceso: 21 de Junio de 2024].

[44] Azure calculator, [en línea]. `https://azure.microsoft.com/es-es/pricing/calculator/`. [Último acceso: 21 de Junio de 2024].

[45] Universidad de valladolid, [en línea]. `https://www.uva.es/export/sites/uva/`. [Último acceso: 12 de Junio de 2024].

[46] Pau Viaplana Fons. Hub & spoke vs punto a punto: análisis práctico de las tipologías de redes aéreas. 2010.

[47] Brandon Schlinker, Todd Arnold, Italo Cunha, and Ethan Katz-Bassett. Peering: Virtualizing bgp at the edge for research. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 51–67, 2019.

[48] Jdk java, [en línea]. `https://www.oracle.com/es/java/technologies/downloads/`. [Último acceso: 19 de Junio de 2024].

[49] Terratest, [en línea]. `https://terratest.gruntwork.io`. [Último acceso: 21 de Junio de 2024].

[50] Go, [en línea]. `https://go.dev/dl/`. [Último acceso: 21 de Junio de 2024].

[51] Pipeline microsoft azure, [en línea]. `https://learn.microsoft.com/es-es/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops`. [Último acceso: 19 de Junio de 2024].

[52] Yaml sintaxis, [en línea]. `https://www.redhat.com/es/topics/automation/what-is-yaml`. [Último acceso: 19 de Junio de 2024].

[53] Justin Clarke-Salt and Justin Clarke. *SQL injection attacks and defense*. Elsevier, 2012.

Appendices

Apéndice A

Código de Terraform

A.1 Módulos de Terraform

A.1.1 Firewall

firewall.tf

```
resource "azurerm_firewall" "firewall" {
    count               = length(var.firewall)
    name                = var.firewall[count.index].name
    location            = var.location
    resource_group_name = var.firewall[count.index].resource_group_name
    sku_name            = var.firewall[count.index].sku_name
    sku_tier            = var.firewall[count.index].sku_tier

    ip_configuration {
        name                 = var.firewall[count.index].ip_configuration_name
        subnet_id            = var.firewall[count.index].ip_configuration_subnet_id
        public_ip_address_id = var.firewall[count.index].
            ↪ ip_configuration_public_ip_address_id
    }
}
```

variables.tf

```
variable "firewall" {
  description = "Manages an Azure Firewall"
  type = list(object({
    name                                = string #Specifies the name of the
        ↪ Firewall.
    resource_group_name                 = string #The name of the resource group in
        ↪  which to create the resource.
    sku_name                            = string #SKU name of the Firewall.
    sku_tier                            = string #SKU tier of the Firewall.
    ip_configuration_name               = string #Specifies the name of the IP
        ↪ Configuration.
    ip_configuration_subnet_id          = string #Reference to the subnet
        ↪ associated with the IP Configuration.
    ip_configuration_public_ip_address_id = string #The ID of the Public IP Address
        ↪ associated with the firewall.
  }))
}

variable "location" {
  description = "Azure region for resources"
  type        = string
}
```

outputs.tf

```
output "firewall_id" {
  value = azurerm_firewall.firewall[0].id
}
```

A.1.2 Network Interface Card

network_interface.tf

```
resource "azurerm_network_interface" "nic" {
  count               = length(var.nic)
  name                = var.nic[count.index].name
  location            = var.location
  resource_group_name = var.nic[count.index].resource_group_name
  ip_configuration {
    name                          = "internal"
    subnet_id                     = var.nic[count.index].subnet_id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          = var.nic[count.index].public_ip_address_id
  }
}

resource "azurerm_network_interface_security_group_association" "nic_nsg" {
  count                     = length(var.nic_nsg)
  network_interface_id      = var.nic_nsg[count.index].network_interface_id
  network_security_group_id = var.nic_nsg[count.index].network_security_group_id
}
```

variables.tf

```
variable "nic" {
  description = "Manages a Network Interface"
  type = list(object({
    name                 = string #The name of the Network Interface.
    resource_group_name  = string #The name of the Resource Group in which to create
        ↪ the Network Interface.
    subnet_id            = string #The ID of the Subnet where this Network Interface
        ↪ should be located in.
    public_ip_address_id = string #Reference to a Public IP Address to associate with
        ↪  this NIC
  }))
}

variable "nic_nsg" {
  description = "Manages the association between a Network Interface and a Network
      ↪ Security Group"
  type = list(object({
    network_interface_id      = string #The ID of the Network Interface.
    network_security_group_id = string #The ID of the Network Security Group which
        ↪ should be attached to the Network Interface.
  }))
}

variable "location" {
  description = "Azure region for resources"
  type        = string
}
```

outputs.tf

```
output "nic_hub_uva_001_vmkali_name" {
  value = azurerm_network_interface.nic[0].name
}

output "nic_hub_uva_001_vmkali_id" {
  value = azurerm_network_interface.nic[0].id
}

output "nic_dev_uva_001_vmservidor_name" {
  value = azurerm_network_interface.nic[1].name
}

output "nic_dev_uva_001_vmservidor_id" {
  value = azurerm_network_interface.nic[1].id
}
```

A.1.3 Network Security Group

network_security_group.tf

```
resource "azurerm_network_security_group" "nsg" {
  count               = length(var.nsg)
  name                = var.nsg[count.index].name
  location            = var.location
  resource_group_name = var.nsg[count.index].resource_group_name

  dynamic "security_rule" {
    for_each = var.nsg[count.index].security_rule
    content {
      name                       = security_rule.value.name
      priority                   = security_rule.value.priority
      direction                  = security_rule.value.direction
      access                     = security_rule.value.access
      protocol                   = security_rule.value.protocol
      source_port_range          = security_rule.value.source_port_range
      destination_port_range     = security_rule.value.destination_port_range
      source_address_prefix      = security_rule.value.source_address_prefix
      destination_address_prefix = security_rule.value.destination_address_prefix
    }
  }
}
```

variables.tf

```
variable "nsg" {
  description = "Manages a network security group that contains a list of network
    ↪ security rules"
  type = list(object({
    name                = string #Specifies the name of the network security group.
    resource_group_name = string #he name of the resource group in which to create
       ↪ the network security group
    security_rule = list(object({
      name                       = string #The name of the security rule.
      priority                   = number #Specifies the priority of the rule.
      direction                  = string #The direction specifies if rule will be
         ↪ evaluated on incoming or outgoing traffic.
      access                     = string #Specifies whether network traffic is
         ↪ allowed or denied.
      protocol                   = string #Network protocol this rule applies to.
      source_port_range          = string #Source Port or Range.
      destination_port_range     = string #Destination Port or Range.
      source_address_prefix      = string #CIDR or source IP range or * to match any
         ↪ IP.
      destination_address_prefix = string #CIDR or destination IP range or * to match
         ↪  any IP.
    }))
  }))
}

variable "location" {
  description = "Azure region for resources"
  type        = string
}
```

outputs.tf

```
output "nsg_hub_uva_001_id" {
  value = azurerm_network_security_group.nsg[0].id
}

output "nsg_dev_uva_001_dmz_id" {
  value = azurerm_network_security_group.nsg[1].id
}
```

A.1.4 Public IP

public_ip.tf

```
resource "azurerm_public_ip" "public_ip" {
  count               = length(var.public_ip)
  name                = var.public_ip[count.index].name
  sku                 = var.public_ip[count.index].sku
  location            = var.location
  resource_group_name = var.public_ip[count.index].resource_group_name
  allocation_method   = var.public_ip[count.index].allocation_method
}
```

variables.tf

```
variable "public_ip" {
  description = "Manages a Public IP Address"
  type = list(object({
    name                = string #Specifies the name of the Public IP
    resource_group_name = string #The name of the Resource Group where this Public IP
        ↪  should exist.
    sku                 = string #The SKU of the Public IP.
    allocation_method   = string #Defines the allocation method for this IP address.
  }))
}

variable "location" {
  description = "Azure region for resources"
  type        = string
}
```

outputs.tf

```
output "pip_hub_uva_001_vmkali_id" {
  value = azurerm_public_ip.public_ip[0].id
}

output "pip_dev_uva_001_vmservidor_id" {
  value = azurerm_public_ip.public_ip[1].id
}

output "pip_dev_uva_001_firewall_id" {
  value = azurerm_public_ip.public_ip[2].id
}
```

A.1.5 Recovery Service Vault

recovery_service_vault.tf

```
resource "azurerm_recovery_services_vault" "recovery_service_vault" {
  count               = length(var.recovery_service_vault)
  name                = var.recovery_service_vault[count.index].name
  location            = var.location
  resource_group_name = var.recovery_service_vault[count.index].resource_group_name
  sku                 = var.recovery_service_vault[count.index].sku

  soft_delete_enabled = var.recovery_service_vault[count.index].soft_delete_enabled
}

resource "azurerm_backup_policy_vm" "bcvm" {
  count               = length(var.bcvm)
  name                = var.bcvm[count.index].name
  resource_group_name = var.bcvm[count.index].resource_group_name
  recovery_vault_name = var.bcvm[count.index].recovery_vault_name

  timezone = "Romance Standard Time"

  backup {
    frequency = "Daily"
    time      = "23:00"
  }

  retention_daily {
    count = 10
  }
}

resource "azurerm_backup_protected_vm" "vm_backup" {
  count               = length(var.vm_backup)
  resource_group_name = var.vm_backup[count.index].resource_group_name
  recovery_vault_name = var.vm_backup[count.index].recovery_vault_name
  source_vm_id        = var.vm_backup[count.index].source_vm_id
  backup_policy_id    = var.vm_backup[count.index].backup_policy_id
}
```

variables.tf

```
variable "recovery_service_vault" {
  description = "Manages a Recovery Services Vault"
  type = list(object({
    name                = string #Specifies the name of the Recovery Services Vault.
    resource_group_name = string #The name of the resource group in which to create
        ↪ the Recovery Services Vault.
    sku                 = string #Sets the vault's SKU.
    soft_delete_enabled = bool   #Is soft delete enable for this Vault?
  }))
}

variable "location" {
  description = "Azure region for resources"
  type        = string
}

variable "bcvm" {
  description = "Manages an Azure Backup VM Backup Policy"
  type = list(object({
    name                = string #Specifies the name of the Backup Policy
    resource_group_name = string #The name of the resource group in which to create
        ↪ the policy.
    recovery_vault_name = string #Specifies the name of the Recovery Services Vault
        ↪ to use.
  }))
}

variable "vm_backup" {
  description = "Manages Azure Backup for an Azure VM"
  type = list(object({
    resource_group_name = string #The name of the resource group in which to create
        ↪ the Recovery Services Vault.
    recovery_vault_name = string #Specifies the name of the Recovery Services Vault
        ↪ to use.
    source_vm_id        = string #Specifies the ID of the VM to backup.
    backup_policy_id    = string #Specifies the id of the backup policy to use.
  }))
}
```

outputs.tf

```
output "rsv_hub_uva_001_name" {
  value = azurerm_recovery_services_vault.recovery_service_vault[0].name
}

output "rsv_dev_uva_001_name" {
  value = azurerm_recovery_services_vault.recovery_service_vault[1].name
}

output "bc_hub_uva_001_vmkali_id" {
  value = azurerm_backup_policy_vm.bcvm[0].id
}
output "bc_dev_uva_001_vmservidor_id" {
  value = azurerm_backup_policy_vm.bcvm[1].id
}
```

A.1.6 Resource Group

resource_group.tf

```
resource "azurerm_resource_group" "rsg" {
  count    = length(var.resource_groups)
  name     = var.resource_groups[count.index].name
  location = var.location
}
```

variables.tf

```
variable "resource_groups" {
  description = "Manages a Resource Group"
  type = list(object({
    name = string #The Name which should be used for this Resource Group.
  }))
}

variable "location" {
  description = "Azure region for resources"
  type        = string
}
```

outputs.tf

```
output "rg_hub_uva_001_name" {
  value = azurerm_resource_group.rsg[0].name
}

output "rg_hub_uva_001_security_name" {
  value = azurerm_resource_group.rsg[1].name
}

output "rg_dev_uva_001_firewall_name" {
  value = azurerm_resource_group.rsg[2].name
}

output "rg_dev_uva_001_dmz_name" {
  value = azurerm_resource_group.rsg[3].name
}

output "rg_dev_uva_001_sql_name" {
  value = azurerm_resource_group.rsg[4].name
}
```

A.1.7 SQL Server

sql_db.tf

```
resource "azurerm_mssql_server" "sql_server" {
  count                        = length(var.sql_server)
  name                         = var.sql_server[count.index].name
  resource_group_name          = var.sql_server[count.index].resource_group_name
  location                     = var.location
  version                      = var.sql_server[count.index].version
  administrator_login          = var.sql_server[count.index].administrator_login
  administrator_login_password = var.sql_server[count.index].
      ↪ administrator_login_password
}

resource "azurerm_mssql_firewall_rule" "firewall_rule_sql_server" {
  count            = length(var.firewall_rule_sql_server)
  name             = var.firewall_rule_sql_server[count.index].name
  server_id        = var.firewall_rule_sql_server[count.index].server_id
  start_ip_address = var.firewall_rule_sql_server[count.index].start_ip_address
  end_ip_address   = var.firewall_rule_sql_server[count.index].end_ip_address
}

resource "azurerm_mssql_database" "database" {
  count       = length(var.database)
  name        = var.database[count.index].name
  server_id   = var.database[count.index].server_id
  collation   = var.database[count.index].collation
  max_size_gb = var.database[count.index].max_size_gb
  sku_name    = var.database[count.index].sku_name
}
```

variables.tf

```
variable "sql_server" {
  description = "Manages␣a␣Microsoft␣SQL␣Azure␣Database␣Server"
  type = list(object({
    name                         = string #The name of the Microsoft SQL Server.
    resource_group_name          = string #The name of the resource group in which to
        ↪  create the Microsoft SQL Server.
    version                      = string #The version for the new server.
    administrator_login          = string #The administrator login name for the new
        ↪ server.
    administrator_login_password = string #The password associated with the
        ↪ administrator_login user.
  }))
}

variable "firewall_rule_sql_server" {
  description = "Allows␣you␣to␣manage␣an␣Azure␣SQL␣Firewall␣Rule"
  type = list(object({
    name             = string #The name of the firewall rule.
    server_id        = string #The resource ID of the SQL Server on which to create
        ↪ the Firewall Rule.
    start_ip_address = string #The starting IP address to allow through the firewall
        ↪ for this rule.
    end_ip_address   = string #The ending IP address to allow through the firewall
        ↪ for this rule.
  }))
}
```

```
variable "database" {
  description = "Manages a MS SQL Database"
  type = list(object({
    name        = string #The name of the MS SQL Database.
    server_id   = string #The id of the MS SQL Server on which to create the database
        ↪ .
    collation   = string #Specifies the collation of the database.
    max_size_gb = number #The max size of the database in gigabytes.
    sku_name    = string #Specifies the name of the SKU used by the database.
  }))
}

variable "location" {
  description = "Azure region for resources"
  type        = string
}
```

outputs.tf

```
output "sql_dev_uva_001_id" {
  value = azurerm_mssql_server.sql_server[0].id
}

output "sql_dev_uva_001_name" {
  value = azurerm_mssql_server.sql_server[0].name
}
```

A.1.8 Subnets

subnets.tf

```
resource "azurerm_subnet" "subnet" {
  count                = length(var.subnet)
  name                 = var.subnet[count.index].name
  resource_group_name  = var.subnet[count.index].resource_group_name
  virtual_network_name = var.subnet[count.index].virtual_network_name
  address_prefixes     = var.subnet[count.index].address_prefixes
}

resource "azurerm_subnet_network_security_group_association" "ssh" {
  count                     = length(var.ssh)
  subnet_id                 = var.ssh[count.index].subnet_id
  network_security_group_id = var.ssh[count.index].network_security_group_id
}
```

variables.tf

```
variable "subnet" {
  description = "Manages a subnet"
  type = list(object({
    name                 = string       #The name of the subnet.
    resource_group_name  = string       #The name of the resource group in which to
        ↪ create the subnet.
    virtual_network_name = string       #The name of the virtual network to which to
        ↪ attach the subnet.
    address_prefixes     = list(string) #The address prefixes to use for the subnet.
  }))
}

variable "ssh" {
  description = "Associates a Network Security Group with a Subnet within a Virtual
      ↪ Network"
  type = list(object({
    subnet_id                 = string #The ID of the Subnet.
    network_security_group_id = string #The ID of the Network Security Group which
        ↪ should be associated with the Subnet.
  }))
}

variable "location" {
  description = "Azure region for resources"
  type        = string
}
```

outputs.tf

```
output "snet_hub_uva_001_id" {
  value = azurerm_subnet.subnet[0].id
}

output "AzureFirewallSubnet_id" {
  value = azurerm_subnet.subnet[1].id
}

output "snet_dev_uva_001_dmz_id" {
  value = azurerm_subnet.subnet[2].id
}

output "snet_dev_uva_001_sql_id" {
  value = azurerm_subnet.subnet[3].id
}
```

A.1.9 Máquinas Virtuales

virtual_machine.tf

```
resource "azurerm_linux_virtual_machine" "virtual_machine" {
  count                           = length(var.virtual_machine)
  name                            = var.virtual_machine[count.index].name
  resource_group_name             = var.virtual_machine[count.index].
      ↪ resource_group_name
  location                        = var.location
  size                            = var.virtual_machine[count.index].size
  admin_username                  = var.virtual_machine[count.index].admin_username
  network_interface_ids           = var.virtual_machine[count.index].
      ↪ network_interface_ids
  disable_password_authentication = false
  admin_password                  = var.virtual_machine[count.index].admin_password
  custom_data                     = filebase64("install_kali.sh")


  os_disk {
    caching              = var.virtual_machine[count.index].caching
    storage_account_type = var.virtual_machine[count.index].storage_account_type
  }

  source_image_reference {
    publisher = "Canonical"
    offer     = "0001-com-ubuntu-server-jammy"
    sku       = "22_04-lts"
    version   = "latest"
  }
}

resource "azurerm_linux_virtual_machine" "virtual_machine_servidor" {
  count                           = length(var.virtual_machine_servidor)
  name                            = var.virtual_machine_servidor[count.index].name
  resource_group_name             = var.virtual_machine_servidor[count.index].
      ↪ resource_group_name
  location                        = var.location
  size                            = var.virtual_machine_servidor[count.index].size
  admin_username                  = var.virtual_machine_servidor[count.index].
      ↪ admin_username
  network_interface_ids           = var.virtual_machine_servidor[count.index].
      ↪ network_interface_ids
  disable_password_authentication = false
  admin_password                  = var.virtual_machine_servidor[count.index].
      ↪ admin_password

  os_disk {
    caching              = var.virtual_machine_servidor[count.index].caching
    storage_account_type = var.virtual_machine_servidor[count.index].
        ↪ storage_account_type
  }

  source_image_reference {
    publisher = "Canonical"
    offer     = "0001-com-ubuntu-server-jammy"
    sku       = "22_04-lts"
    version   = "latest"
  }
}
```

```
resource "null_resource" "app_deployment" {
  # No se requiere ningun atributo especifico para el recurso null_resource.
  depends_on = [azurerm_linux_virtual_machine.virtual_machine_servidor]

  connection {
    type     = "ssh"
    user     = "adminUVAServidor"
    password = "f/x!YI_t6U12"
    host     = azurerm_linux_virtual_machine.virtual_machine_servidor[0].
       ↪ public_ip_address
  }

  #Copiamos la aplicacion
  provisioner "file" {
    source      = "./col-app-0.0.1-SNAPSHOT.war"
    destination = "/home/adminUVAServidor/col-app-0.0.1-SNAPSHOT.war"

  }

  #Copiamos el archivo para ejecutar la aplicacion como servicio
  provisioner "file" {
    source      = "./load_execute_server.service"
    destination = "/home/adminUVAServidor/load_execute_server.service"
  }

  #Ejecutamos la aplicacion como servicio
  provisioner "remote-exec" {
    inline = [
      "chmod␣+x␣/home/adminUVAServidor/col-app-0.0.1-SNAPSHOT.war",
      "chmod␣+x␣/home/adminUVAServidor/load_execute_server.service", # Dar permisos
         ↪ de ejecucion al script
      "sudo␣mv␣/home/adminUVAServidor/load_execute_server.service␣/etc/systemd/system
         ↪ /colapp.service",
      "sudo␣apt-get␣update␣-y",
      "sudo␣apt-get␣install␣-y␣openjdk-11-jdk",
      "sudo␣systemctl␣daemon-reload",
      "sudo␣systemctl␣enable␣colapp",
      "sudo␣systemctl␣start␣colapp",
    ]
  }
}
```

variables.tf

```
variable "virtual_machine" {
  description = "Manages␣a␣Linux␣Virtual␣Machine"
  type = list(object({
    name                  = string       #The name of the Linux Virtual Machine.
    resource_group_name   = string       #The name of the Resource Group in which the
       ↪  Linux Virtual Machine should be exist.
    size                  = string       #The SKU which should be used for this
       ↪ Virtual Machine.
    admin_username        = string       #The username of the local administrator
       ↪ used for the Virtual Machine.
    network_interface_ids = list(string) #A list of Network Interface IDs which
       ↪ should be attached to this Virtual Machine.
    caching               = string       #The Type of Caching which should be used
       ↪ for the Internal OS Disk.
    storage_account_type  = string       #The Type of Storage Account which should
       ↪ back this the Internal OS Disk.
    admin_password        = string       #The Password which should be used for the
       ↪ local-administrator on this Virtual Machine.
  }))
}

variable "virtual_machine_servidor" {
  description = "Manages␣a␣Linux␣Virtual␣Machine"
  type = list(object({
    name                  = string       #The name of the Linux Virtual Machine.
    resource_group_name   = string       #The name of the Resource Group in which the
       ↪  Linux Virtual Machine should be exist.
    size                  = string       #The SKU which should be used for this
       ↪ Virtual Machine.
    admin_username        = string       #The username of the local administrator
       ↪ used for the Virtual Machine.
    network_interface_ids = list(string) #A list of Network Interface IDs which
       ↪ should be attached to this Virtual Machine.
    caching               = string       #The Type of Caching which should be used
       ↪ for the Internal OS Disk.
    storage_account_type  = string       #The Type of Storage Account which should
       ↪ back this the Internal OS Disk.
    admin_password        = string       #The Password which should be used for the
       ↪ local-administrator on this Virtual Machine.
  }))
}

variable "location" {
  description = "Azure␣region␣for␣resources"
  type        = string
}
```

outputs.tf

```
output "vmhubuva001kali_id" {
  value = azurerm_linux_virtual_machine.virtual_machine[0].id
}

output "vmdevuva001servidor_id" {
  value = azurerm_linux_virtual_machine.virtual_machine_servidor[0].id
}

output "vmhubuva001kali_name" {
  value = azurerm_linux_virtual_machine.virtual_machine[0].name
}

output "vmdevuva001servidor_name" {
  value = azurerm_linux_virtual_machine.virtual_machine_servidor[0].name
}

output "vmdevuva001servidor_public_ip_address" {
  value = azurerm_linux_virtual_machine.virtual_machine_servidor[0].public_ip_address
}

output "vmdevuva001kali_public_ip_address" {
  value = azurerm_linux_virtual_machine.virtual_machine[0].public_ip_address
}
```

A.1.10 Red Virtual

virtual_network.tf

```
resource "azurerm_virtual_network" "vnet" {
  count               = length(var.vnet)
  name                = var.vnet[count.index].name
  location            = var.location
  resource_group_name = var.vnet[count.index].resource_group_name
  address_space       = var.vnet[count.index].address_space
}

resource "azurerm_virtual_network_peering" "peering" {
  count                        = length(var.peering)
  name                         = var.peering[count.index].name
  resource_group_name          = var.peering[count.index].resource_group_name
  virtual_network_name         = var.peering[count.index].virtual_network_name
  remote_virtual_network_id    = var.peering[count.index].remote_virtual_network_id
  allow_virtual_network_access = var.peering[count.index].
      ↪ allow_virtual_network_access
}
```

variables.tf

```
variable "vnet" {
  description = "Manages a virtual network including any configured subnets."
  type = list(object({
    name                = string       #The name of the virtual network.
    resource_group_name = string       #The name of the resource group in which to
        ↪ create the virtual network.
    address_space       = list(string) #The address space that is used the virtual
        ↪ network.
  }))
}

variable "location" {
  description = "Azure region for resources"
  type        = string
}

variable "peering" {
  description = "Manages a virtual network peering which allows resources to access
      ↪ other resources in the linked virtual network"
  type = list(object({
    name                          = string #The name of the virtual network peering.
    resource_group_name           = string #The name of the resource group in which to
        ↪  create the virtual network peering.
    virtual_network_name          = string #The name of the virtual network.
    remote_virtual_network_id     = string #The full Azure resource ID of the remote
        ↪ virtual network.
    allow_virtual_network_access = bool   #Controls if the VMs in the remote virtual
        ↪ network can access VMs in the local virtual network.
  }))
}
```

outputs.tf

```
output "vnet_hub_uva_001_name" {
  value = azurerm_virtual_network.vnet[0].name
}

output "vnet_dev_uva_001_firewall_name" {
  value = azurerm_virtual_network.vnet[1].name
}

output "vnet_dev_uva_001_dmz_name" {
  value = azurerm_virtual_network.vnet[2].name
}

output "vnet_dev_uva_001_sql_name" {
  value = azurerm_virtual_network.vnet[3].name
}

output "vnet_hub_uva_001_id" {
  value = azurerm_virtual_network.vnet[0].id
}

output "vnet_dev_uva_001_firewall_id" {
  value = azurerm_virtual_network.vnet[1].id
}

output "vnet_dev_uva_001_dmz_id" {
```

```
  value = azurerm_virtual_network.vnet[2].id
}

output "vnet_dev_uva_001_sql_id" {
  value = azurerm_virtual_network.vnet[3].id
}
```

A.2 Pipeline

A.2.1 azure-pipelines.yml

```
# Starter pipeline

trigger:
- main #esto indica que ejecutara el pipeline cada vez que se incluya algo nuevo en
    ↪ la rama main. Nosotros trabajamos en otra rama y cuando estamos seguros de los
    ↪  cambios lo movemos a la rama main

pool:
  vmImage: ubuntu-latest



variables:
- group: Escenario_ciberseguridad_TFG_Celia Martin
- name: bkstrgrg
  value: 'azDevTFRGBack'
- name: bkstrg
  value: 'backend'
- name: bkcontainer
  value: 'tfstate'
- name: bkstrgkey
  value: 'devpipeline.terraform.tfstate'


stages:
  - stage: tfvalidate #Con esto validamos que no haya errores, si falla se cerrar
      ↪ sin ejecutarse
    jobs:
      - job: validation
        continueOnError: false
        steps:

          - task: DownloadSecureFile@1 #Descargamos el archivo que contiene los
              ↪ secretos y variables confidenciales
            name: prod
            displayName: Download backend config file
            inputs:
              secureFile: "azurerm.tfbackend"

          - task: TerraformInstaller@1
            inputs:
              terraformVersion: 'latest'  #con esto instalamos la version de
                  ↪ terraform que tenemos en nuestro provider


          - task: TerraformTaskV4@4
            displayName: terraforminit
            inputs:
```

```
        provider: 'azurerm'
        command: 'init'
        backendServiceArm: 'infraestructura-uva'
        backendAzureRmResourceGroupName: 'tfstate'
        backendAzureRmStorageAccountName: 'tfstate247'
        backendAzureRmContainerName: 'tfstate'
        backendAzureRmKey: '1
            ↪ Fwnr4C4BB2B25AiDOZmuSiiAmFCFrermNRps71Wn56dN3uo2XXLPhCUjp8FJ7i2W342Tq0ifVIg
            ↪ +AStoGx1Ow=='

    - task: TerraformTaskV4@4
      displayName: terraformvalidate
      inputs:
        provider: 'azurerm'
        command: 'validate'

    - task: TerraformTaskV4@4
      displayName: terraformplan
      inputs:
        provider: 'azurerm'
        command: 'plan'
        workingDirectory: '$(System.DefaultWorkingDirectory)'
        environmentServiceNameAzureRM: 'infraestructura-uva'
        commandOptions: '-var-file=$(prod.secureFilePath)'

    - task: TerraformTaskV4@4
      displayName: terraformapply
      inputs:
        provider: 'azurerm'
        command: 'apply'
        workingDirectory: '$(System.DefaultWorkingDirectory)'
        environmentServiceNameAzureRM: 'infraestructura-uva'
        commandOptions: '-auto-approve -var-file=$(prod.secureFilePath)'
```

A.3 main.tf

```
#RESOURCE GROUP
module "resource_group" {
  source   = "./modules/resource_group"
  location = var.location

  resource_groups = [
    {
      name = "rg-hub-uva-001"
    },
    {
      name = "rg-hub-uva-001-security"

    },
    {
      name = "rg-dev-uva-001-firewall"
    },
    {
      name = "rg-dev-uva-001-dmz"
    },
    {
      name = "rg-dev-uva-001-sql"
  }]
}

#VIRTUAL NETWORK
module "virtual_network" {
  source   = "./modules/virtual_network"
  location = var.location

  vnet = [
    {
      name                = "vnet-hub-uva-001"
      resource_group_name = module.resource_group.rg_hub_uva_001_name
      address_space       = ["10.0.0.0/16"]
    },
    {
      name                = "vnet-dev-uva-001-firewall"
      resource_group_name = module.resource_group.rg_dev_uva_001_firewall_name
      address_space       = ["10.1.0.0/16"]

    },
    {
      name                = "vnet-dev-uva-001-dmz"
      resource_group_name = module.resource_group.rg_dev_uva_001_dmz_name
      address_space       = ["10.2.0.0/16"]
    },
    {
      name                = "vnet-dev-uva-001-sql"
      resource_group_name = module.resource_group.rg_dev_uva_001_sql_name
      address_space       = ["10.3.0.0/16"]
  }]

  peering = [{
    #Peering between hub vnet and firewall vnet
    name                          = "peering_hub_firewall"
    resource_group_name           = module.resource_group.rg_hub_uva_001_name
    virtual_network_name          = module.virtual_network.vnet_hub_uva_001_name
    remote_virtual_network_id     = module.virtual_network.
        ↪ vnet_dev_uva_001_firewall_id
```

```
    allow_virtual_network_access = true
    },
    {
      #Peering between firewall vnet and hub vnet
      name                         = "peering_firewall_hub"
      resource_group_name          = module.resource_group.
          ↪ rg_dev_uva_001_firewall_name
      virtual_network_name         = module.virtual_network.
          ↪ vnet_dev_uva_001_firewall_name
      remote_virtual_network_id    = module.virtual_network.vnet_hub_uva_001_id
      allow_virtual_network_access = true
    },
    {
      #Peering between firewall vnet and dmz vnet
      name                         = "peering_firewall_dmz"
      resource_group_name          = module.resource_group.
          ↪ rg_dev_uva_001_firewall_name
      virtual_network_name         = module.virtual_network.
          ↪ vnet_dev_uva_001_firewall_name
      remote_virtual_network_id    = module.virtual_network.vnet_dev_uva_001_dmz_id
      allow_virtual_network_access = true
    },
    {
      #Peering between dmz vnet and firewall vnet
      name                         = "peering_dmz_firewall"
      resource_group_name          = module.resource_group.rg_dev_uva_001_dmz_name
      virtual_network_name         = module.virtual_network.vnet_dev_uva_001_dmz_name
      remote_virtual_network_id    = module.virtual_network.
          ↪ vnet_dev_uva_001_firewall_id
      allow_virtual_network_access = true
    },
    {
      #Peering between firewall vnet and dev vnet
      name                         = "peering_dev_firewall"
      resource_group_name          = module.resource_group.rg_dev_uva_001_sql_name
      virtual_network_name         = module.virtual_network.vnet_dev_uva_001_sql_name
      remote_virtual_network_id    = module.virtual_network.
          ↪ vnet_dev_uva_001_firewall_id
      allow_virtual_network_access = true
    },
    {
      #Peering between dev vnet and firewall vnet
      name                         = "peering_firewall_dev"
      resource_group_name          = module.resource_group.
          ↪ rg_dev_uva_001_firewall_name
      virtual_network_name         = module.virtual_network.
          ↪ vnet_dev_uva_001_firewall_name
      remote_virtual_network_id    = module.virtual_network.vnet_dev_uva_001_sql_id
      allow_virtual_network_access = true
    }
  ]
}

#SUBNET
module "subnets" {
  source   = "./modules/subnets"
  location = var.location

  subnet = [
    {
      name                 = "snet-hub-uva-001"
      resource_group_name  = module.resource_group.rg_hub_uva_001_name
```

```
      virtual_network_name = module.virtual_network.vnet_hub_uva_001_name
      address_prefixes     = ["10.0.0.0/24"]
    },
    {
      name                 = "AzureFirewallSubnet"
      resource_group_name  = module.resource_group.rg_dev_uva_001_firewall_name
      virtual_network_name = module.virtual_network.vnet_dev_uva_001_firewall_name
      address_prefixes     = ["10.1.0.0/24"]

    },
    {
      name                 = "snet-dev-uva-001-dmz"
      resource_group_name  = module.resource_group.rg_dev_uva_001_dmz_name
      virtual_network_name = module.virtual_network.vnet_dev_uva_001_dmz_name
      address_prefixes     = ["10.2.0.0/24"]
    },
    {
      name                 = "snet-dev-uva-001-sql"
      resource_group_name  = module.resource_group.rg_dev_uva_001_sql_name
      virtual_network_name = module.virtual_network.vnet_dev_uva_001_sql_name
      address_prefixes     = ["10.3.0.0/24"]
  }]

  ssh = [{
    subnet_id                 = module.subnets.snet_hub_uva_001_id
    network_security_group_id = module.network_security_group.nsg_hub_uva_001_id
    },

    {
      subnet_id                 = module.subnets.snet_dev_uva_001_dmz_id
      network_security_group_id = module.network_security_group.
          ↪ nsg_dev_uva_001_dmz_id
    }
  ]
}

#NETOWRK INTERFACE CARD
module "network_interface" {
  source   = "./modules/network_interface"
  location = var.location

  nic = [
    {
      name                 = "nic-hub-uva-001-vmkali"
      resource_group_name  = module.resource_group.rg_hub_uva_001_name
      subnet_id            = module.subnets.snet_hub_uva_001_id
      public_ip_address_id = module.public_ip.pip_hub_uva_001_vmkali_id
    },
    {
      name                 = "nic-dev-uva-001-vmservidor"
      resource_group_name  = module.resource_group.rg_dev_uva_001_dmz_name
      subnet_id            = module.subnets.snet_dev_uva_001_dmz_id
      public_ip_address_id = module.public_ip.pip_dev_uva_001_vmservidor_id
    }
  ]

  nic_nsg = [
    {
      network_interface_id      = module.network_interface.nic_hub_uva_001_vmkali_id
      network_security_group_id = module.network_security_group.nsg_hub_uva_001_id
    },
    {
```

```
      network_interface_id      = module.network_interface.
          ↪ nic_dev_uva_001_vmservidor_id
      network_security_group_id = module.network_security_group.
          ↪ nsg_dev_uva_001_dmz_id
    }
  ]
}

#PUBLIC IP
module "public_ip" {
  source   = "./modules/public_ip"
  location = var.location

  public_ip = [
    {
      name                = "pip-hub-uva-001-vmkali"
      resource_group_name = module.resource_group.rg_hub_uva_001_name
      sku                 = "Standard"
      allocation_method   = "Static"
    },
    {
      name                = "pip-dev-uva-001-vmservidor"
      resource_group_name = module.resource_group.rg_dev_uva_001_dmz_name
      sku                 = "Standard"
      allocation_method   = "Static"
    },
    {
      name                = "pip-dev-uva-001-firewall"
      resource_group_name = module.resource_group.rg_dev_uva_001_firewall_name
      sku                 = "Standard"
      allocation_method   = "Static"
    }
  ]
}

#SQL
module "sql_db" {
  source   = "./modules/sql_db"
  location = var.location

  sql_server = [
    {
      name                         = "sql-dev-uva-001"
      resource_group_name          = module.resource_group.rg_dev_uva_001_sql_name
      version                      = "12.0"
      administrator_login          = var.sql_username
      administrator_login_password = var.sql_password
  }]

  firewall_rule_sql_server = [
    {
      name             = "ClientIPAddress"
      server_id        = module.sql_db.sql_dev_uva_001_id
      start_ip_address = "86.127.226.2"
      end_ip_address   = "86.127.226.2"
    }
  ]

  database = [
    {
      name        = "inventario"
      server_id   = module.sql_db.sql_dev_uva_001_id
```

```
      collation    = "SQL_Latin1_General_CP1_CI_AS"
      max_size_gb  = 2
      sku_name     = "S0"
  }]
}

#FIREWALL
module "firewall" {
  source   = "./modules/firewall"
  location = var.location

  firewall = [
    {
      name                                  = "firewall"
      resource_group_name                   = module.resource_group.
          ↪ rg_dev_uva_001_firewall_name
      sku_name                              = "AZFW_Hub" #"AZFW_VNet"
      sku_tier                              = "Standard"
      ip_configuration_name                 = "configuration"
      ip_configuration_subnet_id            = module.subnets.AzureFirewallSubnet_id
      ip_configuration_public_ip_address_id = module.public_ip.
          ↪ pip_dev_uva_001_firewall_id
  }]
}

#RECOVERY SERVICE VAULT
module "recovery_service_vault" {
  source   = "./modules/recovery_service_vault"
  location = var.location

  recovery_service_vault = [
    {
      name                = "rsv-hub-uva-001"
      resource_group_name = module.resource_group.rg_hub_uva_001_security_name
      sku                 = "Standard"
      soft_delete_enabled = true
    },
    {
      name                = "rsv-dev-uva-001"
      resource_group_name = module.resource_group.rg_dev_uva_001_sql_name
      sku                 = "Standard"
      soft_delete_enabled = true
  }]

  bcvm = [
    {
      name                = "bc-hub-uva-001-vmkali"
      resource_group_name = module.resource_group.rg_hub_uva_001_security_name
      recovery_vault_name = module.recovery_service_vault.rsv_hub_uva_001_name
    },
    {
      name                = "bc-dev-uva-001-vmservidor"
      resource_group_name = module.resource_group.rg_dev_uva_001_sql_name
      recovery_vault_name = module.recovery_service_vault.rsv_dev_uva_001_name
    }
  ]

  vm_backup = [
    {
      resource_group_name = module.resource_group.rg_hub_uva_001_security_name
      recovery_vault_name = module.recovery_service_vault.rsv_hub_uva_001_name
      source_vm_id        = module.virtual_machine.vmhubuva001kali_id
```

```
      backup_policy_id    = module.recovery_service_vault.bc_hub_uva_001_vmkali_id
    },
    {
      resource_group_name = module.resource_group.rg_dev_uva_001_sql_name
      recovery_vault_name = module.recovery_service_vault.rsv_dev_uva_001_name
      source_vm_id        = module.virtual_machine.vmdevuva001servidor_id
      backup_policy_id    = module.recovery_service_vault.
          ↪ bc_dev_uva_001_vmservidor_id
    }
  ]
}

#VIRTUAL MACHINE
module "virtual_machine" {
  source   = "./modules/virtual_machine"
  location = var.location

  virtual_machine = [
    {
      name                = "vmhubuva001kali"
      resource_group_name = module.resource_group.rg_hub_uva_001_name
      size                = "Standard_F2"
      admin_username      = var.user_kali
      network_interface_ids = [module.network_interface.nic_hub_uva_001_vmkali_id]
      caching             = "ReadWrite"
      storage_account_type = "Standard_LRS"
      admin_password      = var.vmkalipassword
  }]

  virtual_machine_servidor = [
    {
      name                = "vmdevuva001servidor"
      resource_group_name = module.resource_group.rg_dev_uva_001_dmz_name
      size                = "Standard_F2"
      admin_username      = var.user_servidor
      network_interface_ids = [module.network_interface.nic_dev_uva_001_vmservidor_id
          ↪ ]
      caching             = "ReadWrite"
      storage_account_type = "Standard_LRS"
      admin_password      = var.vmservidorpassword
  }]
}

#NETWORK SECURY GROUP
module "network_security_group" {
  source   = "./modules/network_security_group"
  location = var.location

  nsg = [
    {
      name                = "nsg-hub-uva-001"
      resource_group_name = module.resource_group.rg_hub_uva_001_name
      security_rule = [{
        name                       = "allow_ssh"
        priority                   = 100
        direction                  = "Inbound"
        access                     = "Allow"
        protocol                   = "Tcp"
        source_port_range          = "*"
        destination_port_range     = "22"
        source_address_prefix      = "*"
        destination_address_prefix = "*"
```

```
        },
        {
          name                       = "allow_apps"
          priority                   = 110
          direction                  = "Inbound"
          access                     = "Allow"
          protocol                   = "Tcp"
          source_port_range          = "*"
          destination_port_range     = "8080"
          source_address_prefix      = "*"
          destination_address_prefix = "*"
      }]

    },

    {
      name                = "nsg-dev-uva-001-dmz"
      resource_group_name = module.resource_group.rg_dev_uva_001_dmz_name
      security_rule = [{
        name                       = "allow_ssh"
        priority                   = 100
        direction                  = "Inbound"
        access                     = "Allow"
        protocol                   = "Tcp"
        source_port_range          = "*"
        destination_port_range     = "22"
        source_address_prefix      = "*"
        destination_address_prefix = "*"
        },
        {
          name                       = "allow_apps"
          priority                   = 110
          direction                  = "Inbound"
          access                     = "Allow"
          protocol                   = "Tcp"
          source_port_range          = "*"
          destination_port_range     = "8080"
          source_address_prefix      = "*"
          destination_address_prefix = "*"
      }]
    }
  ]
}
```

A.4 provider.tf

```
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "=3.30.0"
    }
    mssql = {
      source  = "sl-cloud-platform/mssql"
      version = "0.2.4" # Reemplaza "x.x.x" con la version deseada del proveedor
          ↪ MSSQL
    }

  }

  backend "azurerm" {
    resource_group_name  = "tfstate"
    storage_account_name = "tfstate247"
    container_name       = "tfstate"
    key                  = "1
        ↪ Fwnr4C4BB2B25AiDOZmuSiiAmFCFrermNRps71Wn56dN3uo2XXLPhCUjp8FJ7i2W342Tq0ifVIg
        ↪ +AStoGx1Ow=="
  }
}

provider "azurerm" {
  features {}
  skip_provider_registration = true #Necesario para que no sea necesario dar mas
      ↪ permisos a un app registration
  subscription_id            = var.SUBSCRIPTION_ID
  client_id                  = var.CLIENT_ID
  client_secret              = var.CLIENT_SECRET
  tenant_id                  = var.TENANT_ID
}

provider "mssql" {
  host     = var.SQL_DATABASE
  username = var.SQL_USERNAME
  password = var.SQL_PASSWORD
}
```

A.5 variables.tf

```
variable "location" {
  description = "Azure region for resources"
  type        = string
}

variable "subscription_id" {
  description = "Subscription id"
  type        = string
}

variable "client_id" {
  description = "Subscription id"
  type        = string
}

variable "client_secret" {
  description = "Subscription id"
  type        = string
}

variable "tenant_id" {
  description = "Subscription id"
  type        = string
}

variable "sql_username" {
  type = string
}

variable "sql_password" {
  type = string
}

variable "sql_database" {
  type = string
}

variable "vmkalipassword" {
  type = string
}

variable "vmservidorpassword" {
  type = string
}

variable "user_kali" {
  type = string
}

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

A.6 output.tf

```
#Maquinas Virtuales
output "vmhubuva001kali_id" {
  value = module.virtual_machine.vmhubuva001kali_id
}

output "vmdevuva001servidor_id" {
  value = module.virtual_machine.vmdevuva001servidor_id
}

output "vmhubuva001kali_name" {
  value = module.virtual_machine.vmhubuva001kali_name
}

output "vmdevuva001servidor_name" {
  value = module.virtual_machine.vmdevuva001servidor_name
}

output "vmdevuva001servidor_public_ip_address" {
  value = module.virtual_machine.vmdevuva001servidor_public_ip_address
}

output "vmdevuva001kali_public_ip_address" {
  value = module.virtual_machine.vmdevuva001kali_public_ip_address
}

#Firewall
output "firewall_id" {
  value = module.firewall.firewall_id
}

output "firewall_name" {
  value = module.firewall.firewall_name
}

#Network Interface
output "nic_hub_uva_001_vmkali_name" {
  value = module.network_interface.nic_hub_uva_001_vmkali_name
}

output "nic_hub_uva_001_vmkali_id" {
  value = module.network_interface.nic_hub_uva_001_vmkali_id
}

output "nic_dev_uva_001_vmservidor_name" {
  value = module.network_interface.nic_dev_uva_001_vmservidor_name
}

output "nic_dev_uva_001_vmservidor_id" {
  value = module.network_interface.nic_dev_uva_001_vmservidor_id
}

#Network Security Group
output "nsg_hub_uva_001_id" {
  value = module.network_security_group.nsg_hub_uva_001_id
}

output "nsg_dev_uva_001_dmz_id" {
  value = module.network_security_group.nsg_dev_uva_001_dmz_id
}
```

```
#Public IP
output "pip_hub_uva_001_vmkali_id" {
  value = module.public_ip.pip_hub_uva_001_vmkali_id
}

output "pip_dev_uva_001_vmservidor_id" {
  value = module.public_ip.pip_dev_uva_001_vmservidor_id
}

output "pip_dev_uva_001_firewall_id" {
  value = module.public_ip.pip_dev_uva_001_firewall_id
}

#Resource Group
output "rg_hub_uva_001_name" {
  value = module.resource_group.rg_hub_uva_001_name
}

output "rg_hub_uva_001_security_name" {
  value = module.resource_group.rg_hub_uva_001_security_name
}

output "rg_dev_uva_001_firewall_name" {
  value = module.resource_group.rg_dev_uva_001_firewall_name
}

output "rg_dev_uva_001_dmz_name" {
  value = module.resource_group.rg_dev_uva_001_dmz_name
}

output "rg_dev_uva_001_sql_name" {
  value = module.resource_group.rg_dev_uva_001_sql_name
}

#SQL
output "sql_dev_uva_001_id" {
  value = module.sql_db.sql_dev_uva_001_id
}

output "sql_dev_uva_001_name" {
  value = module.sql_db.sql_dev_uva_001_name
}

#Subnet
output "snet_hub_uva_001_id" {
  value = module.subnets.snet_hub_uva_001_id
}

output "AzureFirewallSubnet_id" {
  value = module.subnets.AzureFirewallSubnet_id
}

output "snet_dev_uva_001_dmz_id" {
  value = module.subnets.snet_dev_uva_001_dmz_id
}

output "snet_dev_uva_001_sql_id" {
  value = module.subnets.snet_dev_uva_001_sql_id
}

#Vnet
```

```
output "vnet_hub_uva_001_name" {
  value = module.virtual_network.vnet_hub_uva_001_name
}

output "vnet_dev_uva_001_firewall_name" {
  value = module.virtual_network.vnet_dev_uva_001_firewall_name
}

output "vnet_dev_uva_001_dmz_name" {
  value = module.virtual_network.vnet_dev_uva_001_dmz_name
}

output "vnet_dev_uva_001_sql_name" {
  value = module.virtual_network.vnet_dev_uva_001_sql_name
}

output "vnet_hub_uva_001_id" {
  value = module.virtual_network.vnet_hub_uva_001_id
}

output "vnet_dev_uva_001_firewall_id" {
  value = module.virtual_network.vnet_dev_uva_001_firewall_id
}

output "vnet_dev_uva_001_dmz_id" {
  value = module.virtual_network.vnet_dev_uva_001_dmz_id
}

output "vnet_dev_uva_001_sql_id" {
  value = module.virtual_network.vnet_dev_uva_001_sql_id
}
```

A.7 .gitignore

```
# Local .terraform directories
**/.terraform
*.tfstate
*.tfstate.*


# Crash log files
crash.log

# Exclude all .tfvars files, which are likely to contain sentitive data, such as
# password, private keys, and other secrets. These should not be part of version
# control as they are data points which are potentially sensitive and subject
# to change depending on the environment.
*.tfvars

# Ignore override files as they are usually used to override resources locally and so
# are not checked in
override.tf
override.tf.json
*_override.tf
*_override.tf.json

# Include override files you do wish to add to version control using negated pattern
#
# !example_override.tf

# Include tfplan files to ignore the plan output of command: terraform plan -out=
    ↪ tfplan
# example: *tfplan*

# Ignore CLI configuration files
.terraformrc
terraform.rc
```

A.8 Tests Unitarios

A.8.1 firewall_test.go

```
package test

import (
        "testing"

        "github.com/gruntwork-io/terratest/modules/terraform"
        "github.com/stretchr/testify/assert"
)

func FirewallTest(t *testing.T) {
        //Elegimos el directorio donde se va ha ejecutar terraform
        fixtureFolder := "../"
        terraformOptions := &terraform.Options{
                TerraformDir: fixtureFolder,
        }

        defer terraform.Destroy(t, terraformOptions) //Destruimos los recursos una
            ↪ vez finalice
        terraform.InitAndApply(t, terraformOptions)  //Realizamos el apply en
            ↪ terraform

        //Almacenamos los outputs que obtenemos en el apply
        instanceID := terraform.Output(t, terraformOptions, "firewall_id")
        instanceName := terraform.Output(t, terraformOptions, "firewall_name")

        //Realizamos las comparaciones
        assert.NotEmpty(t, instanceID)
        assert.NotEmpty(t, instanceName)
}
```

A.8.2 networkinterface_test.go

```
package test

import (
        "testing"

        "github.com/gruntwork-io/terratest/modules/terraform"
        "github.com/stretchr/testify/assert"
)

func NetworkInterfaceTest(t *testing.T) {
        //Elegimos el directorio donde se va ha ejecutar terraform
        fixtureFolder := "../"
        terraformOptions := &terraform.Options{
                TerraformDir: fixtureFolder,
        }

        defer terraform.Destroy(t, terraformOptions) //Destruimos los recursos una
            ↪ vez finalice
        terraform.InitAndApply(t, terraformOptions)  //Realizamos el apply en
            ↪ terraform

        //Almacenamos los outputs que obtenemos en el apply
        nicIDKali := terraform.Output(t, terraformOptions, "nic_hub_uva_001_vmkali_id
            ↪ ")
```

```
        nicNameKali := terraform.Output(t, terraformOptions, "
            ↪ nic_hub_uva_001_vmkali_name")
        nicIDServidor := terraform.Output(t, terraformOptions, "
            ↪ nic_dev_uva_001_vmservidor_id")
        nicNameServidor := terraform.Output(t, terraformOptions, "
            ↪ nic_dev_uva_001_vmservidor_name")

        //Realizamos las comparaciones
        assert.NotEmpty(t, nicIDKali)
        assert.NotEmpty(t, nicNameKali)
        assert.NotEmpty(t, nicIDServidor)
        assert.NotEmpty(t, nicNameServidor)
}
```

A.8.3 networksecuritygroup_test.go

```
package test

import (
        "testing"

        "github.com/gruntwork-io/terratest/modules/terraform"
        "github.com/stretchr/testify/assert"
)

func NetworkSecurityGroupTest(t *testing.T) {
        //Elegimos el directorio donde se va ha ejecutar terraform
        fixtureFolder := "../"
        terraformOptions := &terraform.Options{
                TerraformDir: fixtureFolder,
        }

        defer terraform.Destroy(t, terraformOptions) //Destruimos los recursos una
            ↪ vez finalice
        terraform.InitAndApply(t, terraformOptions)  //Realizamos el apply en
            ↪ terraform

        //Almacenamos los outputs que obtenemos en el apply
        nicIDHub := terraform.Output(t, terraformOptions, "nsg_hub_uva_001_id")
        nicIDDev := terraform.Output(t, terraformOptions, "nsg_dev_uva_001_dmz_id")

        //Realizamos las comparaciones
        assert.NotEmpty(t, nicIDHub)
        assert.NotEmpty(t, nicIDDev)
}
```

A.8.4 publicip_test.go

```
package test

import (
        "testing"

        "github.com/gruntwork-io/terratest/modules/terraform"
        "github.com/stretchr/testify/assert"
)

func PublicIPTest(t *testing.T) {
        //Elegimos el directorio donde se va ha ejecutar terraform
        fixtureFolder := "../"
        terraformOptions := &terraform.Options{
                TerraformDir: fixtureFolder,
        }

        defer terraform.Destroy(t, terraformOptions) //Destruimos los recursos una
            ↪ vez finalice
        terraform.InitAndApply(t, terraformOptions)  //Realizamos el apply en
            ↪ terraform

        //Almacenamos los outputs que obtenemos en el apply
        pipKaliID := terraform.Output(t, terraformOptions, "pip_hub_uva_001_vmkali_id
            ↪ ")
        pipServidorID := terraform.Output(t, terraformOptions, "
            ↪ pip_dev_uva_001_vmservidor_id")
        pipFirewallID := terraform.Output(t, terraformOptions, "
            ↪ pip_dev_uva_001_firewall_id")

        //Realizamos las comparaciones
        assert.NotEmpty(t, pipKaliID)
        assert.NotEmpty(t, pipServidorID)
        assert.NotEmpty(t, pipFirewallID)
}
```

A.8.5 resourcegroup_test.go

```
package test

import (
        "testing"

        "github.com/gruntwork-io/terratest/modules/terraform"
        "github.com/stretchr/testify/assert"
)

func ResourceGroupTest(t *testing.T) {
        //Elegimos el directorio donde se va ha ejecutar terraform
        fixtureFolder := "../"
        terraformOptions := &terraform.Options{
                TerraformDir: fixtureFolder,
        }

        defer terraform.Destroy(t, terraformOptions) //Destruimos los recursos una
            ↪ vez finalice
        terraform.InitAndApply(t, terraformOptions)  //Realizamos el apply en
            ↪ terraform
```

```
        //Almacenamos los outputs que obtenemos en el apply
        hubName := terraform.Output(t, terraformOptions, "rg_hub_uva_001_name")
        securityName := terraform.Output(t, terraformOptions, "
            ↪ rg_hub_uva_001_security_name")
        firewallName := terraform.Output(t, terraformOptions, "
            ↪ rg_dev_uva_001_firewall_name")
        dnzName := terraform.Output(t, terraformOptions, "rg_dev_uva_001_dmz_name")
        sqlName := terraform.Output(t, terraformOptions, "rg_dev_uva_001_sql_name")

        //Realizamos las comparaciones
        assert.NotEmpty(t, hubName)
        assert.NotEmpty(t, securityName)
        assert.NotEmpty(t, firewallName)
        assert.NotEmpty(t, dnzName)
        assert.NotEmpty(t, sqlName)
}
```

A.8.6 sqldb_test.go

```
package test

import (
        "testing"

        "github.com/gruntwork-io/terratest/modules/terraform"
        "github.com/stretchr/testify/assert"
)

func SQLDBTest(t *testing.T) {
        //Elegimos el directorio donde se va ha ejecutar terraform
        fixtureFolder := "../"
        terraformOptions := &terraform.Options{
                TerraformDir: fixtureFolder,
        }

        defer terraform.Destroy(t, terraformOptions) //Destruimos los recursos una
            ↪ vez finalice
        terraform.InitAndApply(t, terraformOptions)  //Realizamos el apply en
            ↪ terraform

        //Almacenamos los outputs que obtenemos en el apply
        instanceID := terraform.Output(t, terraformOptions, "sql_dev_uva_001_id")
        instanceName := terraform.Output(t, terraformOptions, "sql_dev_uva_001_name")

        //Realizamos las comparaciones
        assert.NotEmpty(t, instanceID)
        assert.NotEmpty(t, instanceName)
}
```

A.8.7 subnets_test.go

```
package test

import (
        "testing"

        "github.com/gruntwork-io/terratest/modules/terraform"
        "github.com/stretchr/testify/assert"
)

func SubnetsTest(t *testing.T) {
        //Elegimos el directorio donde se va ha ejecutar terraform
        fixtureFolder := "../"
        terraformOptions := &terraform.Options{
                TerraformDir: fixtureFolder,
        }

        defer terraform.Destroy(t, terraformOptions) //Destruimos los recursos una
            ↪ vez finalice
        terraform.InitAndApply(t, terraformOptions)  //Realizamos el apply en
            ↪ terraform

        //Almacenamos los outputs que obtenemos en el apply
        hubID := terraform.Output(t, terraformOptions, "snet_hub_uva_001_id")
        firewallID := terraform.Output(t, terraformOptions, "AzureFirewallSubnet_id")
        dmzID := terraform.Output(t, terraformOptions, "snet_dev_uva_001_dmz_id")
        sqlID := terraform.Output(t, terraformOptions, "snet_dev_uva_001_sql_id")

        //Realizamos las comparaciones
        assert.NotEmpty(t, hubID)
        assert.NotEmpty(t, firewallID)
        assert.NotEmpty(t, dmzID)
        assert.NotEmpty(t, sqlID)
}
```

A.8.8 virtualmachine_test.go

```
package test

import (
        "testing"

        "github.com/gruntwork-io/terratest/modules/terraform"
        "github.com/stretchr/testify/assert"
)

func VirtualMachineTest(t *testing.T) {
        //Elegimos el directorio donde se va ha ejecutar terraform
        fixtureFolder := "../"
        terraformOptions := &terraform.Options{
                TerraformDir: fixtureFolder,
        }

        defer terraform.Destroy(t, terraformOptions) //Destruimos los recursos una
            ↪ vez finalice
        terraform.InitAndApply(t, terraformOptions)  //Realizamos el apply en
            ↪ terraform

        //Almacenamos los outputs que obtenemos en el apply
```

```
        //Kali
        instanceID := terraform.Output(t, terraformOptions, "vmhubuva001kali_id")
        instanceName := terraform.Output(t, terraformOptions, "vmhubuva001kali_name")
        publicIPinstance := terraform.Output(t, terraformOptions, "
            ↪ vmdevuva001kali_public_ip_address")

        //Servidor
        instanceIDServidor := terraform.Output(t, terraformOptions, "
            ↪ vmdevuva001servidor_id")
        instanceNameServidor := terraform.Output(t, terraformOptions, "
            ↪ vmdevuva001servidor_name")
        publicIPinstanceServidor := terraform.Output(t, terraformOptions, "
            ↪ vmdevuva001servidor_public_ip_address")
        //Realizamos las comparaciones
        assert.NotEmpty(t, instanceID)
        assert.NotEmpty(t, instanceName)
        assert.NotEmpty(t, publicIPinstance)
        assert.NotEmpty(t, instanceIDServidor)
        assert.NotEmpty(t, instanceNameServidor)
        assert.NotEmpty(t, publicIPinstanceServidor)
}
```

A.8.9 vnet_test.go

```
package test

import (
        "testing"

        "github.com/gruntwork-io/terratest/modules/terraform"
        "github.com/stretchr/testify/assert"
)

func VnetTest(t *testing.T) {
        //Elegimos el directorio donde se va ha ejecutar terraform
        fixtureFolder := "../"
        terraformOptions := &terraform.Options{
                TerraformDir: fixtureFolder,
        }

        defer terraform.Destroy(t, terraformOptions) //Destruimos los recursos una
            ↪ vez finalice
        terraform.InitAndApply(t, terraformOptions)  //Realizamos el apply en
            ↪ terraform

        //Almacenamos los outputs que obtenemos en el apply
        hubID := terraform.Output(t, terraformOptions, "vnet_hub_uva_001_id")
        hubName := terraform.Output(t, terraformOptions, "vnet_hub_uva_001_name")
        firewallID := terraform.Output(t, terraformOptions, "
            ↪ vnet_dev_uva_001_firewall_id")
        firewallName := terraform.Output(t, terraformOptions, "
            ↪ vnet_dev_uva_001_firewall_name")
        dmzID := terraform.Output(t, terraformOptions, "vnet_dev_uva_001_dmz_id")
        dmzName := terraform.Output(t, terraformOptions, "vnet_dev_uva_001_dmz_name")
        sqlID := terraform.Output(t, terraformOptions, "vnet_dev_uva_001_sql_id")
        sqlName := terraform.Output(t, terraformOptions, "vnet_dev_uva_001_sql_name")

        //Realizamos las comparaciones
        assert.NotEmpty(t, hubID)
        assert.NotEmpty(t, hubName)
```

```
        assert.NotEmpty(t, firewallID)
        assert.NotEmpty(t, firewallName)
        assert.NotEmpty(t, dmzID)
        assert.NotEmpty(t, dmzName)
        assert.NotEmpty(t, sqlID)
        assert.NotEmpty(t, sqlName)
}
```

A.8.10 go.mod

```
module test

go 1.21

require (
        github.com/gruntwork-io/terratest v0.28.5
        github.com/stretchr/testify v1.9.0
)

require (
        github.com/davecgh/go-spew v1.1.1 // indirect
        github.com/pmezard/go-difflib v1.0.0 // indirect
        golang.org/x/crypto v0.0.0-20200220183623-bac4c82f6975 // indirect
        golang.org/x/net v0.0.0-20191209160850-c0dbc17a3553 // indirect
        golang.org/x/sys v0.0.0-20200107162124-548cf772de50 // indirect
        gopkg.in/yaml.v3 v3.0.1 // indirect
)
```

Apéndice B

Glosario de Términos

Redes: Las redes informáticas son sistemas que permiten la conexión entre dispositivos electrónicos y computadoras para compartir recursos y comunicarse entre sí. Los dispositivos que conforman una red pueden incluir computadoras personales, servidores, enrutadores, conmutadores, puntos de acceso inalámbrico, entre otros.

Virtual Network: Una virtual network (VNet) en azure es un servicio de red que permite a los usuarios crear y gestionar redes privadas en la nube de Azure. Se trata de una representación virtual de una red tradicional en las instalaciones, pero alojada en la infraestructura de Azure. Al utilizar una virtual network, los usuarios pueden conectar y aislar recursos en la nube, como máquinas virtuales, bases de datos y otros servicios de manera segura y controlada.

Subredes: Las subredes (subnets) son segmentos lógicos que se crean dentro de una red más grande, como una red local o una virtual network en la nube, para dividir y organizar los recursos de manera eficiente. Las subredes permiten una administración más granular de los dispositivos y ayudan a optimizar el tráfico en la red.

Subred Privada: Una subred privada es un segmento de red que utiliza direcciones IP privadas, por lo tanto, no son accesibles directamente desde internet. Estas direcciones IP privadas se encuentran dentro de los rangos definidos en las especificaciones de direcciones IP privadas, como 10.0.0.0/8, 172.16.0.0/12 y 192.168.0.0/16. Estas subredes se utilizan para aislar y proteger recursos internos en una red local o en una nube pública, haciendo que estos recursos solo sean accesibles desde dentro de la red o a través de conexiones VPN seguras. Al utilizar subredes privadas los recursos no están expuestos directamente a internet, esto proporciona una capa adicional de seguridad, ya que los dispositivos no son accesibles directamente desde el exterior.

Subred Pública: Una subred pública es un segmento de red que utiliza direcciones IP públicas lo que hace que sean accesibles directamente desde internet. Estas direcciones IP están registradas y asignadas globalmente y son enrutadas a través de internet. Los recursos en una subred pública están expuestos directamente a internet, lo que puede representar un mayor riesgo de seguridad si no se implementan medidas adecuadas de protección como firewalls y políticas de seguridad.

Zona Desmilitarizada: Una Zona Desmilitarizada (DMZ) es una subred intermedia y neutral que se sitúa entre la red interna privada y la red externa pública, generalmente Internet. La DMZ actúa como un área de separación entre la red interna, que contiene recursos críticos y sensibles, y

el mundo exterior, donde se encuentra el tráfico no confiable de Internet. El propósito principal de una DMZ es permitir el acceso controlado y seguro a ciertos servicios o recursos que necesitan estar disponibles para el público o para usuarios externos, como servidores web, correos electrónicos públicos, aplicaciones públicas... Al colocar estos servicios en la DMZ, se evita que el tráfico externo no confiable tenga acceso directo a la red interna que contiene datos sensibles o valiosos.

Network Security Group: Un Network Security Group (NSG) es un recurso de seguridad virtual que actúa como un firewall a nivel de red para controlar el tráfico de entrada y salida en una red virtual de Azure. Es una herramienta clave ya que permite asegurar y proteger los recursos que están alojados en la nube gracias a las reglas definidas que determinan que tipos de comunicación están permitidos y cuáles deben ser bloqueados.

Máquina Virtual: Una máquina virtual, también conocida como "instancia", es una representación virtual de un sistema operativo y recursos de hardware que se aloja en servidores remotos dentro de un entorno de computación en la nube. Estas máquinas virtuales permiten a los usuarios ejecutar aplicaciones y sistemas operativos sin la necesidad de poseer ni mantener un hardware físico en sus propias ubicaciones. El concepto de máquinas virtuales en la nube ha revolucionado la forma en que las organizaciones y los individuos despliegan y gestionan sus recursos informáticos, ya que ofrece ventajas significativas en términos de escalabilidad, flexibilidad, seguridad y eficiencia.

Servidor Web: Un servidor web es un software que se ejecuta en un ordenador o en un conjunto de ordenadores conectados a una red y tiene la función de almacenar, gestionar y entregar el contenido web a los usuarios que acceden a través de un navegador web. Esta diseñado para recibir solicitudes de los clientes (generalmente navegadores web) y responder a esas solicitudes enviando el contenido solicitado, como páginas web, imágenes, vídeos y otros archivos.

SQL Server: SQL Server es un sistema de gestión de bases de datos (DBMS) desarrollado por Microsoft. Es una plataforma confiable y de alto rendimiento que se utiliza para almacenar, gestionar y recuperar datos en aplicaciones y entornos empresariales. SQL Server se basa en el lenguaje de consulta estructurado (SQL), que es el estándar para interactuar con los sistemas de bases de datos relacionales. Esta tecnología permite a los usuarios realizar consultas y manipular datos de manera eficiente y segura.

Bases de Datos: Una base de datos es un conjunto estructurado de datos organizados y almacenados de manera que se pueden acceder, administrar y actualizar fácilmente. Los datos almacenados pueden contener cualquier tipo de información, como nombres, números, fechas, imágenes y más. Las bases de datos permiten a los usuarios o aplicaciones interactuar con los datos de manera eficiente y segura.

Firewall: Un firewall es una barrera de seguridad utilizada para proteger redes y sistemas informáticos contra amenazas externas no autorizadas. Funciona como un filtro entre una red privada o un sistema y las redes públicas, como internet. Su objetivo principal es controlar el tráfico de datos que entra y sale de la red, permitiendo solo el paso de datos autorizados y bloqueando o rechazando el tráfico no deseado o malicioso.

Azure Backup: Azure Backup es un servicio de copia de seguridad y recuperación ofrecido por Microsoft Azure que permite a los usuarios proteger y respaldar datos de sus máquinas virtuales, aplicaciones y archivos en la nube. Este servicio es una solución rentable y confiable para asegurar los datos almacenados en la nube de Azure, garantizando la disponibilidad y recuperación de la información en caso de desastres, errores humanos o fallos del sistema.

Recovery Service Vault: El Recovery Service Vault (RSV) de Microsoft Azure es un componente esencial para la gestión y organización de copias de seguridad y recuperación en la nube. Es un servicio que actúa como un contenedor lógico que almacena y administra los datos de copia de seguridad de máquinas virtuales, servidores locales y otros recursos protegidos por Azure Backup u otras soluciones de recuperación.