Plataforma web para creación y gestión de máquinas virtuales

Marco Pablo Orozco Saravia

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Plataforma web para creación y gestión de máquinas virtuales

Trabajo de graduación en modalidad de trabajo profesional presentado
por
Marco Pablo Orozco Saravia
para optar al grado académico de Licenciado en Ingeniería en Ciencias de la Computación y Tecnologías de la Información

Guatemala, noviembre del 2024

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Plataforma web para creación y gestión de máquinas virtuales

Trabajo de graduación en modalidad de trabajo profesional presentado
por
Marco Pablo Orozco Saravia
para optar al grado académico de Licenciado en Ingeniería en Ciencias de la Computación y Tecnologías de la Información

Guatemala, noviembre del 2024

Vo.Bo.:

(f) _____

Ing. Erick Marroquin

Tribunal Examinador:

(f) _____

Ing. Erick Marroquin

(f) _____

Ing. Alexander Bolaños

(f) _____

Ing. Douglas Barrios

Fecha de aprobación: Guatemala, 3 de Diciembre de 2024.

Índice

Lista de Figuras

Lista de Cuadros

Resumen

El Departamento de Computación reconoce la importancia de la inclusión de sistemas Linux y máquinas virtuales en el pénsum de la carrera, así como la necesidad de implementación de tecnologías en la nube. En respuesta a esto se desarrolló una plataforma para crear y gestionar máquinas virtuales.

El objetivo principal fue proporcionar acceso a instancias de máquinas virtuales para los programas de Ingeniería en Ciencias de la Computación y Tecnologías de la Información, funcionando como una prueba de concepto para una nube computacional universitaria. Se implementaron tecnologías clave como React, Vite, FastAPI, Docker y OpenStack para facilitar la interacción entre el *frontend* y el *backend*, además de gestionar los recursos de infraestructura de manera eficiente.

Finalmente, se recomienda validar la funcionalidad de la plataforma en diferentes entornos para asegurar su robustez y adaptabilidad a distintos contextos. Esto garantizaría que el sistema se mantenga flexible y confiable, cumpliendo con las expectativas de los usuarios y las necesidades futuras de la universidad.

CAPÍTULO 1

Introducción

La computación en la nube se ha convertido en un pilar fundamental en el mundo de la tecnología, ofreciendo una amplia gama de servicios y soluciones que transforman la manera en que interactuamos con la información y los recursos tecnológicos. Sus beneficios se reflejan en mejoras significativas en la productividad, la escalabilidad y la reducción de costos en los proyectos (Google, s.f.). Por otra parte, la comprensión y el uso de máquinas virtuales ofrecen múltiples ventajas en el aprendizaje del desarrollo de software. Estas incluyen la posibilidad de utilizar sistemas operativos como Linux de manera cómoda, ejecutar aplicaciones de diferentes sistemas operativos y disponer de una computadora adicional para realizar tareas o pruebas (Ramírez, 2020).

La Universidad del Valle de Guatemala reconoce la creciente necesidad de integrar tecnologías avanzadas en la educación. En respuesta, ha emprendido la implementación de una solución de computación en la nube con el objetivo de proporcionar recursos innovadores y accesibles a sus estudiantes. Esta iniciativa facilitará a los estudiantes el acceso a máquinas virtuales y mejorará sus habilidades prácticas en un entorno digital avanzado. Con esta estrategia, la universidad aspira a preparar a sus alumnos para enfrentar los desafíos del ámbito laboral y proporcionarles herramientas que complementen su aprendizaje en los cursos de la carrera de Ingeniería en Ciencias de la Computación y Tecnologías de la Información. Estas herramientas serán especialmente útiles en materias como sistemas operativos, diseño y desarrollo web, microprocesadores, y otras áreas clave del programa.

Ante la necesidad de este proyecto, se propone el desarrollo de una plataforma web para la creación y gestión de máquinas virtuales. Aunque su desarrollo puede ser complejo, se considera que la propuesta presentada en este protocolo podría marcar el inicio de una red de servicios en la nube destinada a la universidad y sus estudiantes.

El objetivo general es la construcción de una plataforma para crear y gestionar instancias de máquinas virtuales en los servidores disponibles de la Universidad del Valle de Guatemala, con el fin de proporcionar a los estudiantes acceso a recursos tecnológicos avanzados y mejorar su formación práctica en ingeniería de software.

La propuesta se centra en tres puntos clave:

- Uso de tecnologías de proveedores de computación en la nube: integración de servicios avanzados para asegurar la eficiencia, escalabilidad y fiabilidad de la infraestructura.

- Desarrollo de un *backend*: Creación de una arquitectura capaz de gestionar las solicitudes de los usuarios y ejecutar comandos en el sistema operativo del servidor de manera eficiente.
- Implementación de un *frontend* amigable: Diseño de una interfaz intuitiva y fácil de usar que facilite la creación y gestión de máquinas virtuales por parte de los usuarios.

En la plataforma, los usuarios podrán registrarse e iniciar sesión. Una vez registrados, tendrán acceso al panel de creación de máquinas virtuales, donde podrán seleccionar entre los parámetros disponibles proporcionados por el proveedor de infraestructura en la nube. Además, podrán gestionar las instancias creadas, encenderlas, apagarlas, eliminarlas y reiniciarlas según sea necesario. Todas las interacciones con el panel serán procesadas por el *backend*, el cual deberá interpretar las especificaciones proporcionadas por el usuario y traducirlas en comandos entendibles por la plataforma de gestión de la infraestructura en la nube, asegurando así la creación correcta de la máquina virtual deseada. Una vez creada, se proporcionará al usuario la clave y la dirección de la máquina para que pueda establecer la conexión necesaria.

CAPÍTULO 2

Objetivos

2.1. Objetivo general

Construir una plataforma para crear y gestionar instancias de máquinas virtuales en los servidores disponibles de la Universidad del Valle de Guatemala, con el fin de proporcionar a los estudiantes acceso a recursos tecnológicos avanzados y mejorar su formación práctica en ingeniería de software.

2.2. Objetivos específicos

- Desarrollar una API que capte las solicitudes del *frontend* y se comunique con la línea de comandos para permitir la creación y gestión de máquinas virtuales de manera automática, facilitando su creación desde el *frontend*.
- Programar *scripts* automatizados que ejecuten conjuntos de comandos de terminal para facilitar la configuración y despliegue de máquinas virtuales, utilizando herramientas de *scripting*.
- Programar una interfaz web intuitiva que permita a los usuarios crear y gestionar instancias de máquinas virtuales para proporcionar una experiencia de usuario amigable y accesible, empleando tecnologías web y técnicas de diseño de interfaces de usuario.
- Desplegar la plataforma en los servidores de la universidad para optimizar el uso de recursos tecnológicos existentes y proporcionar un entorno práctico para las necesidades de los estudiantes.

CAPÍTULO 3

Justificación

La creación de esta plataforma ofrece una solución a la necesidad de la Universidad del Valle de Guatemala de tener un proveedor de infraestructura en la nube, particularmente, para los estudiantes de la carrera de ciencias de la computación y tecnologías de la información.

La interacción con tecnologías web y el manejo de máquinas virtuales son habilidades fundamentales para el desarrollo de software, ya que estas permiten el acceso a sistemas operativos, ahorro de costos, desarrollo y pruebas de software (Zúñiga, s.f.). Bajo esta premisa, es crucial proporcionar a los estudiantes las herramientas y la experiencia necesarias para desarrollar competencias sólidas en diversas aplicaciones de estas tecnologías, como diseño web, desarrollo de aplicaciones móviles, desarrollo de API, entre otros. Estas competencias no solo mejoran su formación académica, sino que también los preparan de manera integral para enfrentar los desafíos del mundo laboral.

Este último aspecto es muy importante ya que podemos destacar la importancia de conocer Linux, esto les permite laborar en mantenimiento y administración de sistemas Linux como servidores demás sistemas que lo integran (Sánchez, 2023). Este es uno de los numerosos beneficios de contar con máquinas virtuales. Esta plataforma facilitará el acceso de estos beneficios a los estudiantes, teniendo una plataforma oficial y optimizada para crear estos equipos.

Además, esta plataforma tiene un gran potencial para enriquecer y facilitar la labor docente, ofreciendo a los profesores herramientas que complementan y potencian la enseñanza de cursos que requieren el uso de distintos sistemas operativos, como Linux. Asignaturas como Programación de Microprocesadores, Tecnologías Web, Organización de Computadoras, y Assembler, entre otras, se beneficiarán de esta infraestructura. Con la plataforma, los docentes pueden estar seguros de que sus estudiantes tendrán acceso constante a entornos de práctica adecuados, permitiéndoles aplicar lo aprendido de manera práctica y directa. Además, los profesores también tendrán acceso sencillo y directo a estos sistemas operativos a través de la plataforma desarrollada.

Ante esta necesidad, puede surgir la idea de utilizar servicios de terceros, sin embargo, estas poseen limitaciones que no están alineadas con los objetivos de este proyecto.

En primer lugar, pueden surgir problemas relacionados con el costo de los servicios en la nube. Servicios como Amazon AWS suelen ofrecer tarifas basadas en el uso por hora o mediante suscripciones mensuales (Amazon, s.f.). Esta es la norma para manejar los costos asociados a estos servicios. Suponiendo que se utilizará la infraestructura en la nube para muchos cursos, podemos estimar la necesidad de 100 máquinas virtuales para los estudiantes. Existen herramientas *Opensource* como

Openstack que nos aseguran un presupuesto amigbale para la implementación de la plataforma.

Además, para el lanzamiento del proyecto se utilizarán servidores desocupados de la Universidad, que cuentan con 30 GB de memoria RAM, una capacidad adecuada para manejar procesos de multitarea intensa y aplicaciones de software corporativo (Kingston, 2024). Una ventaja adicional es que podemos seleccionar el sistema operativo que utilizaremos en estos servidores, lo que proporciona flexibilidad y personalización según las necesidades del proyecto.

El uso de los servidores de la universidad y el desarrollo interno de la plataforma permite una libertad total para elegir la configuración específica necesaria, diseñada específicamente para satisfacer los requisitos y necesidades de la institución. Al personalizar estos servidores según las especificaciones y estándares establecidos por la universidad, se puede ofrecer una experiencia de usuario perfectamente ajustada a las demandas y objetivos de los cursos que integren estos servicios en su programa académico. Esta personalización asegura que la plataforma esté alineada con las metas educativas y proporcione los recursos adecuados para maximizar el aprendizaje y la eficiencia.

CAPÍTULO 4

Marco teórico

4.1. Computación en la nube

La computación en la nube, conocida también como *cloud computing*, ha revolucionado la manera en que las organizaciones acceden y gestionan recursos tecnológicos. Para el contexto de este proyecto, centrado en la creación y gestión de instancias de máquinas virtuales, es fundamental explorar varios aspectos clave de esta tecnología.

La computación en la nube permite acceder a servicios de software, almacenamiento y procesamiento de datos a través de Internet, eliminando la necesidad de infraestructuras locales específicas. Esto facilita el acceso remoto y la escalabilidad de recursos según la demanda (Rockcontent, 2018).

4.1.1. Servicios de infraestructura en la nube

Infraestructura como servicio (IaaS)

La infraestructura como servicio ofrece los elementos fundamentales de la tecnología en la nube, proporcionando acceso a funciones de red, servidores (ya sean virtuales o físicos) y almacenamiento de datos.

Software como servicio (SaaS)

Los proveedores de software como Servicio (SaaS) ofrecen aplicaciones de software que ellos mismos ejecutan y gestionan. Normalmente, cuando se habla de SaaS, se está refiriendo a aplicaciones que los usuarios finales utilizan y que son proporcionadas por terceros (AWS, s.f.)

4.1.2. Máquina virtual

Es un entorno informático que funciona como un sistema aislado con su propia CPU, memoria, interfaz de red y almacenamiento, el cual se crea a partir de un conjunto de recursos de hardware. (Red Hat, 2024)

4.1.3. Instancia

Es un sinónimo de máquina virtual (Google, s.f.)

4.1.4. Proveedor de infraestructura en la nube

Un proveedor de servicios en la nube es una compañía tecnológica que ofrece recursos de computación de manera escalable y bajo demanda, como capacidad de procesamiento, almacenamiento de datos y aplicaciones a través de Internet. Los modelos de servicio en la nube comúnmente se categorizan en IaaS (infraestructura como servicio), PaaS (plataforma como servicio) y SaaS (software como servicio) (Google Cloud, s.f.).

4.2. Plataformas de software de código abierto

El software de código abierto es aquel cuyo código fuente está disponible para que cualquier persona pueda revisarlo, modificarlo y mejorarlo. El código fuente es el componente del software que los programadores usan para cambiar cómo funciona una aplicación o añadir nuevas funcionalidades. Al tener acceso a este código, cualquier usuario puede personalizar el software, agregar nuevas funciones o corregir errores (AWS, s.f.).

4.3. Tecnologías web

Las tecnologías web son el conjunto de herramientas y técnicas utilizadas para crear aplicaciones y servicios que funcionan a través de internet.

4.3.1. *Frontend*

El *frontend* es la parte de la aplicación web con la que los usuarios interactúan directamente. Se enfoca en mejorar la experiencia del usuario mediante una interfaz visualmente atractiva y funcionalmente intuitiva. Las tecnologías clave en el desarrollo *frontend* incluyen HTML, CSS y JavaScript, cada una de las cuales aporta elementos esenciales para la creación de páginas web (conquerblocks, s.f.).

Tecnologías indispensables para el desarrollo *frontend*

HTML: Lenguaje de Marcado de Hipertexto en inglés, es el lenguaje utilizado para definir el contenido de las páginas web. Consiste en un conjunto de etiquetas interpretadas por los navegadores, las cuales permiten estructurar y organizar elementos como texto, imágenes, listas, tablas y vídeos que conforman una página web (Vadavo, 2023).

CSS: Es un lenguaje esencial utilizado para definir y personalizar la presentación visual de documentos estructurados en HTML. Su principal función radica en crear el diseño estético y la apariencia de las páginas web y las interfaces de usuario. Mediante la aplicación de reglas y estilos, CSS permite controlar aspectos como colores, fuentes, márgenes y disposición de elementos en una página, asegurando una experiencia visual coherente y atractiva para los usuarios.

En el desarrollo web y la programación *frontend*, CSS desempeña un papel crucial al proporcionar herramientas para la personalización y mejora de la interacción del usuario con la web. Por ejemplo, las transformaciones CSS permiten modificar la posición, escala y rotación de elementos en la página, añadiendo dinamismo y fluidez al diseño (Souto, 2023).

JavaScript: Es un lenguaje de programación que permite añadir funciones complejas a las páginas web. Siempre que una página web realiza acciones más allá de mostrar información estática, como actualizaciones en tiempo real, mapas interactivos, animaciones de gráficos 2D/3D, o control de reproductores de video, es muy probable que JavaScript esté involucrado. Este lenguaje es esencial para crear experiencias web dinámicas y enriquecidas, mejorando significativamente la interactividad y funcionalidad de los sitios web (MDN Web Docs, s.f.).

Framework

Es un conjunto de herramientas y librerías diseñado para facilitar el desarrollo de aplicaciones de manera más rápida y eficiente. Actúa como un marco de trabajo que establece un conjunto de reglas y convenciones, proporcionando una estructura básica sobre la cual se puede construir software. Al utilizar un *framework*, los desarrolladores pueden ahorrar tiempo y esfuerzo, ya que ofrece soluciones predefinidas para problemas comunes en el desarrollo de software. Esto permite que los desarrolladores se concentren en las funcionalidades específicas de su aplicación en lugar de tener que resolver problemas técnicos desde cero. En resumen, un *framework* es esencial para estandarizar y simplificar el proceso de desarrollo, mejorando la eficiencia y la calidad del software (Lucena, 2023).

Ejemplos de *framework*s utilizados en el desarrollo web: *React*, Vue, Angular.

4.3.2. *Backend*

El *backend* administra la lógica de la aplicación. El *backend* capta las solicitudes del usuario para poder generar una respuesta utilizando el protocolo HTTP. El *backend* interactúa con otros servicios como bases de datos, API, microservicios, etc. (Amazon, s.f.).

4.4. Interfaces de programación

Una API, o interfaz de programación de aplicaciones, es un conjunto de normas y protocolos que facilitan la comunicación y el intercambio de datos, características y funcionalidades entre diferentes aplicaciones de software (Goodwin, 2024).

Para este proyecto, la función de la API es establecer un protocolo de comunicación robusto y eficiente entre la plataforma, la base de datos y el servidor. Esto permitirá ejecutar las solicitudes de los usuarios de manera fluida y segura. La API actuará como intermediario, gestionando las interacciones y asegurando que las solicitudes de los usuarios se traduzcan correctamente en operaciones en la base de datos y el servidor.

4.5. *Scripts* de automatización

un script de automatización es un conjunto de instrucciones diseñado para cumplir un objetivo específico. Este objetivo puede variar ampliamente, como iniciar un conjunto de servicios, desbloquear el firewall, apagar el sistema bajo ciertas condiciones, entre otros. Están escritos para realizar tareas repetitivas de forma automática, sin intervención humana. Estos *scripts* son esenciales para la eficiencia operativa en el desarrollo y administración de sistemas (Aranz, 2024).

4.6. Diseño de interfaz de usuario

Una interfaz de usuario es la representación visual que facilita la interacción entre un usuario y un dispositivo, software, producto o servicio. También conocida como UI (user interface), se encarga de convertir señales, imágenes, símbolos o acciones del sistema en elementos comprensibles y accesibles para el usuario. (Segui, 2024)

4.6.1. *Wireframe*

Un *wireframe* es un esquema visual que traza la estructura básica de un proyecto o producto tecnológico. También conocido como el plano de la página o de la interfaz, ilustra cómo se organizan y conectan los distintos elementos entre sí, proporcionando una visión clara de la disposición y jerarquía de la información. (Miro, n.d.).

4.6.2. Diseño responsivo

Es un enfoque de desarrollo web que adapta automáticamente el contenido y la estructura de un sitio a las dimensiones y características del dispositivo utilizado por el usuario. Esto asegura que los sitios se ajusten de manera óptima para proporcionar una experiencia fluida, ya sea que se acceda desde teléfonos móviles, tabletas o computadoras de escritorio (Coppola, 2023).

4.6.3. Pruebas de usabilidad

Técnica de evaluación en la que los usuarios interactúan con un producto, sistema o sitio web y se observa su comportamiento. Se utilizan para identificar problemas de usabilidad y mejorar la experiencia del usuario.

4.7. Tecnologías utilizadas

4.7.1. *OpenStack*

OpenStack es una plataforma de tecnología open source que utiliza recursos virtuales agrupados para diseñar y gestionar nubes privadas y públicas (Red Hat, n.d.).

4.7.2. *Docker*

Es la tecnología de organización en contenedores que posibilita la creación y el uso de los contenedores de Linux (Red Hat, n.d.).

4.7.3. Contenedores

Los contenedores son paquetes livianos que encapsulan el código de una aplicación junto con todas sus dependencias, como versiones específicas de entornos de ejecución y bibliotecas necesarias para ejecutar correctamente los servicios de software. Esto asegura que las aplicaciones se ejecuten de manera consistente en cualquier entorno (Google Cloud, n.d.).

4.7.4. PostgreSQL

Es una base de datos relacional de código abierto (Microsoft, n.d.).

4.7.5. Base de datos relacional

Una base de datos relacional (RDB) es una forma de estructurar información en tablas, filas y columnas (Microsoft, n.d.).

4.7.6. *React*

React es una librería para crear interfaces de usuarios con código abierto (Coppola, 2023).

CAPÍTULO 5

Alcance

El alcance de este proyecto se centra en desarrollar una prueba de concepto de computación en la nube específicamente orientada a ofrecer acceso a máquinas virtuales (VM) para los estudiantes de la Universidad del Valle de Guatemala. Esta prueba de concepto tiene un objetivo concreto y limitado: permitir la creación, gestión y uso de máquinas virtuales, sin extenderse a otros servicios o funciones avanzadas de computación en la nube. La plataforma se diseñará para ser funcional y eficiente, cubriendo las necesidades básicas de los usuarios en cuanto a acceso y administración de máquinas virtuales, pero sin incluir servicios adicionales como almacenamiento en la nube, análisis de datos, o servicios de inteligencia artificial.

Además, se utilizarán servidores existentes en la universidad para alojar la plataforma, con la intención de maximizar los recursos ya disponibles y evitar costos adicionales de infraestructura en la nube. El sistema será capaz de soportar múltiples instancias y usuarios, pero se implementará en un entorno de prueba controlado, de manera que se mantenga la viabilidad operativa y se puedan realizar mejoras iterativas en caso de ser necesario. Esta decisión también permite que la universidad personalice el entorno de nube para sus necesidades específicas, estableciendo una base que podría ampliarse en el futuro si el concepto resulta exitoso.

En resumen, el alcance de este proyecto se limita a demostrar la viabilidad de un entorno de máquinas virtuales gestionado internamente por la universidad, proveyendo un sistema funcional que cumple con los requisitos mínimos de operación y enseñanza, pero sin ampliar la plataforma hacia soluciones empresariales o infraestructuras complejas.

CAPÍTULO 6

Metodología

6.1. Descripción inicial del sistema

El sistema propuesto consiste en una plataforma web para la creación y gestión de máquinas virtuales diseñada específicamente para la Universidad del Valle de Guatemala. Su principal objetivo es proporcionar a los estudiantes de las carreras de Ingeniería en Ciencias de la Computación y Tecnologías de la Información un acceso fácil a instancias de máquinas virtuales y funcionar como una prueba de concepto de una nube computacional para la universidad, esto aprovechando los servidores disponibles en la Universidad.

Para cumplir su cometido se proponen 4 entidades clave:

- *Frontend*.
- *Backend*.
- Base de datos.
- Proveedor de herramientas para gestionar infraestructura en la nube.

6.2. Planificacion del desarrollo

Se estableció la siguiente planificación para el desarrollo de la plataforma:

- Procedimientos de instalación de software y definición de tecnologías a utilizar
    1. Elección de tecnologías a utilizar para *frontend*, *backend* y base de datos. Se elegirán las tecnologías para desarrollar 3 entidades clave. Esto en base a la investigación de las tecnologías más tentativas: React, Python, SQL y Openstack. Justificando su uso en el proyecto

Cabe destacar que los modulos de *backend*, *frontend* e interacción con proveedor de infraestructura en la nube se desarrollarán simultáneamente, esto con el objeitvo de que ambos se actualicen y retroaliemtnen entre ellas constantemente. Esto fue necesario ya que es necesario evaluar el comportamiento entre ambos módulos.

- Desarrollo del *frontend* del proyecto

  1. Elaboración de prototipo en Figma de las secciones críticas del proyecto, este diseño será puesto a prueba con una encuesta
  2. Desarrollo de un diseño funcional prototípico con el objetivo de probar funcionalidades en el *backend*
  3. En base a los resultados de la encuesta, realizar una nueva propuesta de diseño. Se pondrá a prueba con una encuesta
  4. En base a los resultados de la segunda encuesta, realizar una nueva propuesta de diseño y aplicarla al diseño funcional

- Desarrollo del *backend* del proyecto

  1. Integración de base de datos
  2. Integración del API
  3. Implementación de ejecución de comandos
  4. Añadir funciones para interactuar con la base de datos
     a) Crear usuarios
     b) Crear máquinas virtuales
     c) Editar máquinas virtuales
  5. Establecer *endpoints* y funciones de base de datos correspondientes
     a) *Endpoint* para usuarios
     b) *Endpoint* para máquinas virtuales

- Desarrollo de *scripts* de automatización

  Su desarrollo comenzará cuando se requiera de su implementación en el *backend*

  1. Reconocer funcionalidad que requiera de un *script* de automatización
  2. Reunir comandos necesarios
  3. Escribir el *script*

- Despliegue de la plataforma en servidores de la Universidad

  El despliegue se realizará al tener un proyecto con las funcionalidades principales realizadas

  1. Instalar dependencias.
  2. Clonar el repositorio o isntalar el proyecto.
  3. Realizar portforwarding a las instancias creadas.
  4. Actualizar el proyecto continuamente.

- Pruebas con usuarios. Decidí utilizar una prueba de usabilidad siguiendo el formato de Hubspot para pruebas de usabilidad.

  1. Establecer las tareas del estudio: las tareas que cada usuario deberá realizar son las siguientes:
     - Crear un usuario
     - Iniciar sesión

- Crear una instancia con sistema operativo cirros
- Conectarse mediante ssh a la instsancia creada
- Editar el nombre de la instancia
- Hacer un reboot a la instancia creada
- Eliminar la instancia creada

2. Establecer un estándar para el éxito de las tareas. La tarea se considerará exitosa si el usuario fue capaz de realizarla. Si el usuario requirió de más de 2 preguntas para realizarla se considerará exitosa pero con posibles mejoras. Si el usuario no pudo realizarla o requirió más de 2 preguntas se consideraá fallida.
3. Redactar un plan de estudio. La prueba consiste en los siguientes pasos:
   a) Introducir el proyecto
   b) Realizar las tareas de forma consecutiva
   c) Recibir opiniones generales del usuario
4. Encontrar a los participantes. Para ello primero debemos definir el perfil de los potenciales usuarios. Los participantes deben ser estudiantes y maestros de la Universidad del Valle de Guatemala y deben conocer el concepto de máquinas virtuales. Para elegir el tamaño de la muestra se utilizó la técnica de muestreo del Nielsen Norman Group: "Why You Only Need to Test with 5 Users"de forma que se elegirán a 2 maestros y 3 alumnos para la prueba.
5. Realizar la prueba.
6. Analizar los datos obtenidos.
   (Hubspot, 2023)

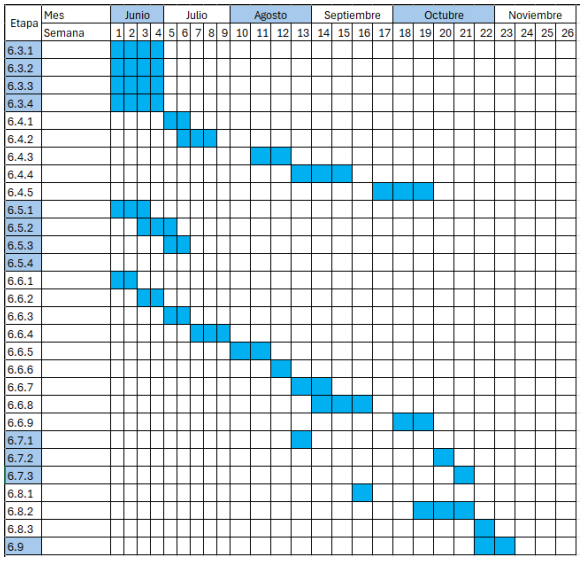

Figura 6.1: Cronograma de actividades realizadas

6.3. Procedimientos de instalación de software y definición de tecnologías a utilizar

6.3.1. *Framework*/librería para el desarrollo *frontend*

Se realizó una investigación para seleccionar las tecnologías más adecuadas, tomando en cuenta las necesidades del proyecto y las mejores prácticas actuales en desarrollo web. Se analizaron diferentes fuentes, incluyendo "State of JS" y "npm trends", para respaldar la toma de decisiones con datos concretos. Para el desarrollo se tomaron las siguientes decisiones:

- *Framework* para *frontend*: **React**.
- *Build Tools*: **Vite**.

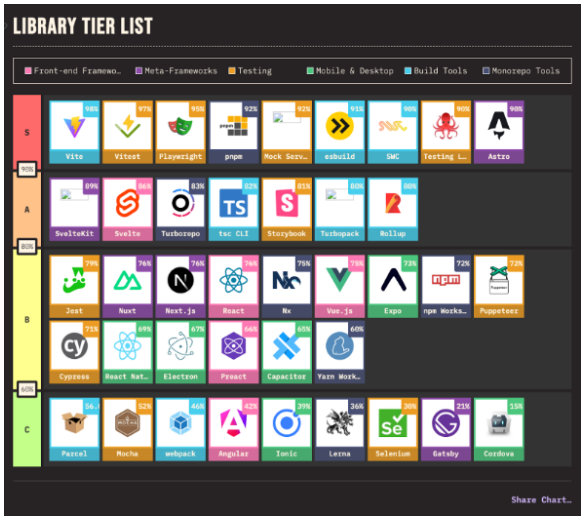

Figura 6.2: Tier list de librerías de Javascript

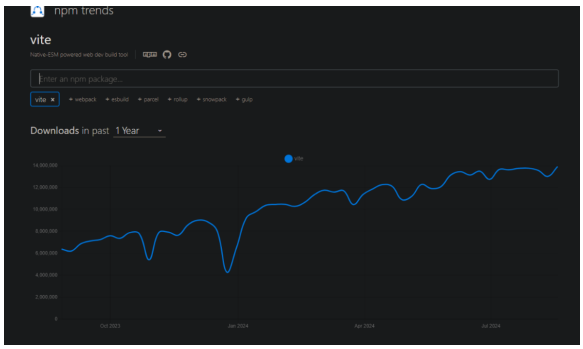

Figura 6.3: Gráfico de descargas de Vite utilizando npm

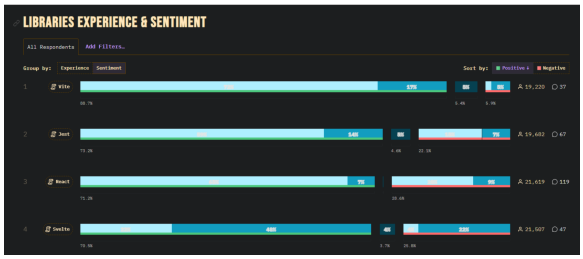

Figura 6.4: Gráfico de experiencias de desarrolladores respecto a librerías de Javascript

En relación al *feedback* de la comunidad sobre las tecnologías *frontend*, escalabilidad y rendimiento, los gráficos anteriormente mostrados a continuación proporcionan información valiosa para la elección de las herramientas a utilizar en este proyecto. Las conclusiones derivadas de este análisis son las siguientes:

- **Vite** ha emergido como la herramienta de construcción (build tool) más popular y preferida por los desarrolladores. Esto se refleja claramente en su destacada posición en el gráfico de npm trends, donde se observa un crecimiento sostenido en la cantidad de descargas de esta herramienta. Su popularidad creciente respalda su selección para el proyecto, especialmente considerando la necesidad de escalabilidad y rendimiento óptimo.
- En cuanto a las librerías para el desarrollo *frontend*, la mayoría se sitúa en una posición similar en términos de popularidad y satisfacción, con la excepción de ***Svelte***, que sobresale ligeramente en comparación con otras alternativas. No obstante, la mayoría de estas librerías son competentes en términos de escalabilidad y rendimiento, según los datos de *State of JS*.
- **React** y ***Svelte*** destacan como herramientas con un *feedback* extremadamente positivo, liderando sus respectivas categorías: *frontend frameworks* y *build tools*. Este *feedback*, combinado con la popularidad y el soporte continuo de la comunidad, justifica la elección de React como el *framework* principal para el desarrollo *frontend* en este proyecto.
- En términos de escalabilidad, la creciente popularidad y la retroalimentación positiva de **Vite** se consolidan como argumentos clave para su elección. Además, dado que este proyecto también sirve como una prueba de concepto para el despliegue de una nube computacional en la universidad, es fundamental que su desarrollo sea adaptable a futuros desarrolladores. Esto se logra mediante la utilización de tecnologías ampliamente reconocidas y enseñadas en el plan de estudios de la carrera de Ciencias de la Computación y Tecnologías de la Información, donde **React** ocupa un lugar destacado. Su popularidad en la industria está bien documentada en gráficos como los de *State of JS*, y los resultados de la encuesta de *Stack Overflow* reafirman que **Node.js** y **React.js** son las tecnologías web más comunes en desarrollo.

En conclusión, para el desarrollo del sistema de gestión de máquinas virtuales de la Universidad del Valle de Guatemala, se decidió utilizar **React** como *framework* principal para el *frontend*, junto con **Vite** como herramienta de construcción (*build tool*). Esta decisión se tomó con base en un análisis detallado de las tendencias actuales en la industria, que destacan tanto la popularidad como el rendimiento y la escalabilidad de estas tecnologías. **React** y **Vite** no solo cuentan con un fuerte respaldo de la comunidad de desarrolladores, sino que también se alinean con las necesidades de escalabilidad del proyecto y son tecnologías reconocidas en el plan de estudios de la universidad, lo que asegura su sostenibilidad y adaptabilidad a largo plazo.

6.3.2. Tecnologías para el desarrollo del API

Uno de los aspectos clave del proyecto es la facilidad para ejecutar comandos de consola a través de la aplicación, especialmente en lo que respecta a la automatización de tareas y la gestión de infraestructura. En este contexto, la selección de tecnologías se centró en dos componentes principales: el lenguaje de desarrollo para el API y el sistema de gestión de bases de datos.

6.3.3. Tecnologías para el desarrollo del API

Uno de los aspectos clave del proyecto es la facilidad para ejecutar comandos de consola a través de la aplicación, especialmente en lo que respecta a la automatización de tareas y la gestión de infraestructura. En este contexto, la selección de tecnologías se centró en dos componentes principales: el lenguaje de desarrollo para el API y el sistema de gestión de bases de datos.

Lenguaje para el desarrollo del API: Python

Se optó por Python como lenguaje de desarrollo para el API debido a varias razones fundamentadas. Python es reconocido no solo por su popularidad y facilidad de uso, sino también por su extensa biblioteca estándar, que incluye herramientas como la librería `subprocess`, la cual permite la ejecución eficiente de comandos de consola directamente desde el código (Subprocess Management, n.d.). Esto es crucial para la funcionalidad de nuestro sistema, que requiere una interacción directa y fluida con la línea de comandos del servidor.

Además, Python ha demostrado ser extremadamente versátil y adaptable en el desarrollo de API, respaldado por una vasta colección de *framework*s y librerías especializadas en este tipo de aplicaciones (LinkedIn, n.d.). Esta adaptabilidad, combinada con su simplicidad, fue determinante en la elección de Python para el desarrollo de nuestro API.

Sistema de Gestión de Base de Datos: PostgreSQL en Docker

En cuanto a la gestión de datos, se seleccionó PostgreSQL como el sistema de base de datos debido a su robustez y fiabilidad en entornos de producción. Sin embargo, dado que la base de datos tiene un rol secundario en este proyecto, no se buscaron características altamente especializadas. En su lugar, se priorizó la facilidad de despliegue y mantenimiento, para lo cual se decidió ejecutar PostgreSQL en un contenedor Docker.

Docker ha emergido como una de las tecnologías de virtualización más utilizadas y reconocidas en los últimos años (Stack Overflow Developer Survey 2023, n.d.). Su capacidad para encapsular aplicaciones en contenedores replicables ofrece ventajas significativas para nuestro proyecto, como la facilidad de reconstrucción en caso de fallos, y la capacidad de establecer un servicio continuo y gratuito para la plataforma. Mediante el uso de volúmenes en Docker, se garantiza que los datos almacenados en PostgreSQL sean fácilmente replicables y respaldados, asegurando la integridad y disponibilidad de la información sin complicaciones (NordVPN, 2024)

6.3.4. Sistema operativo del servidor

La decisión se tomó de manera rápida debido a las circunstancias iniciales. Para conocer y evaluar las especificaciones técnicas del servidor, fue fundamental instalar un sistema operativo. Se eligió **Ubuntu Server 22.04** por ser una distribución robusta y ampliamente utilizada en entornos de

servidores, lo que garantiza estabilidad y soporte a largo plazo. Durante la etapa de interacción con la infraestructura en la nube, se utilizó una máquina virtual corriendo Ubuntu Server 22.04 para evaluar su rendimiento, facilidad de uso y capacidad de adaptación a las tareas específicas del proyecto. Esta máquina virtual sirvió como base durante todo el desarrollo, lo que no solo permitió realizar pruebas de manera eficiente, sino que también facilitó el proceso de instalación y configuración, dado que ya se había adquirido experiencia previa trabajando con este sistema operativo.

De esta forma, la elección de Ubuntu Server 22.04 fue clave para agilizar el proceso y asegurar una instalación segura de la aplicación.

6.4. Desarrollo del *frontend*

Para el desarrollo del primer prototipo, se establecieron las siguientes consideraciones fundamentales:

- **Visualización de Instancias Desplegadas:** Es crucial que el sistema permita a los usuarios visualizar de manera clara y detallada todas las instancias de máquinas virtuales que han sido desplegadas. Esta funcionalidad asegura que los usuarios puedan gestionar eficientemente sus recursos y realizar un seguimiento adecuado del estado y la configuración de cada instancia.
- **Creación de Instancias:** La capacidad de crear nuevas instancias de máquinas virtuales es una función esencial del prototipo. Debe proporcionar una interfaz intuitiva y eficiente que facilite el proceso de configuración y despliegue de nuevas instancias, ajustándose a las especificaciones requeridas por los usuarios.

Estas funcionalidades son consideradas las más críticas para el éxito del proyecto en su etapa inicial. El enfoque principal del primer prototipo se centró en implementar y optimizar estas capacidades para asegurar su funcionamiento efectivo y satisfacer las necesidades básicas de los usuarios.

Cabe destacar que, aunque la implementación de un sistema de autenticación (*login/register*) es una característica fundamental en la mayoría de las aplicaciones, se ha decidido no incluirla en este primer prototipo. Esta decisión se basa en que la funcionalidad de autenticación es un aspecto estándar y ampliamente conocido, que puede ser abordado en fases posteriores del desarrollo sin comprometer el enfoque principal en las funcionalidades clave del prototipo.

6.4.1. Primer prototipo *wireframe* en Figma

Se comenzó con un *wireframe*, con el objetivo de consolidar la estructura básica de la página.

6.4.2. Primera versión de diseño funcional

Figura 6.11: Diseño de visualización de instancias (con instancias creadas)

Figura 6.12: Diseño de Crear una instancia

La primera versión del diseño se enfocó en proporcionar una interfaz funcional para facilitar el desarrollo conjunto del *backend* y las funcionalidades esenciales de la plataforma. Este diseño inicial, aunque rudimentario y estéticamente básico, tuvo el propósito fundamental de permitir la integración y prueba de las características críticas que interactúan entre el *frontend* y el *backend*. Al centrarse en la funcionalidad más que en el aspecto visual, se garantizó que las funcionalidades esenciales estuvieran operativas y se pudiera avanzar en la construcción de la plataforma.

Durante esta fase inicial, se establecieron las bases para el desarrollo del sistema, permitiendo implementar y ajustar las funcionalidades clave, como la visualización y creación de instancias de máquinas virtuales. El diseño provisional permitió identificar y resolver problemas técnicos tempranamente, facilitando una comprensión clara de los requisitos funcionales y operativos de la aplicación.

Gracias a esto, fue sencillo realizar el siguiente paso en el diseño de la aplicación. En respuesta a los comentarios en la encuesta, se realizaron mejoras significativas en el diseño. La segunda versión del diseño de la plataforma incorpora las recomendaciones de los usuarios, con un enfoque particular en la página de visualización de máquinas. Este diseño revisado no sólo aborda las inquietudes estéticas y funcionales mencionadas, sino que también busca proporcionar una experiencia de usuario más intuitiva y visualmente atractiva.

6.4.3. Segunda versión de diseño funcional

Figura 6.13: Diseño de visualización de instancias en tabla (con instancias creadas)

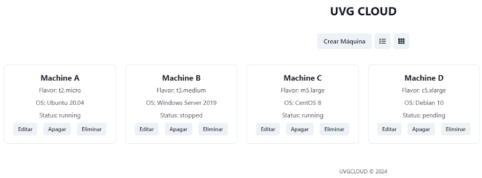

Figura 6.14: Diseño de visualización de instancias en cards (con instancias creadas)

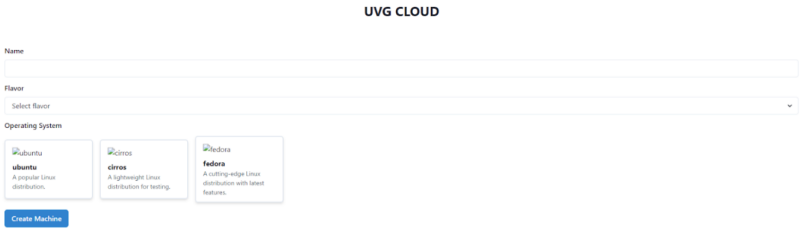

Figura 6.15: Diseño de Crear una instancia

En comparación con la versión anterior del diseño, se observó una transformación significativa en la interfaz, destacando principalmente la incorporación de dos opciones distintas para la visualización de las máquinas virtuales. Basándonos en el *feedback* de la encuesta, donde un usuario sugirió una visualización tipo tarjeta para las instancias creadas, implementamos esta propuesta como una segunda opción junto al botón de Çrear máquina". Esta nueva opción permite a los usuarios seleccionar entre una vista de lista tradicional y una vista en tarjetas, proporcionando una mayor flexibilidad y adaptabilidad según las preferencias individuales.

Además de la incorporación de esta nueva funcionalidad, el diseño se ajustó para lograr una apariencia más homogénea y coherente. Se realizó una revisión exhaustiva de los elementos visuales para asegurar que todos los componentes de la interfaz, incluyendo el botón de Çrear máquinaz el footer, se integrarán de manera armónica dentro de un estilo uniforme y profesional. Este enfoque

no solo mejora la estética general, sino que también contribuye a una experiencia de usuario más fluida y cohesiva.

Estos cambios reflejan un compromiso con la mejora continua del diseño, respondiendo de manera efectiva a las sugerencias de los usuarios y optimizando la interfaz para satisfacer mejor sus necesidades y expectativas.

6.4.4. Tercera versión de diseño funcional

Esta versión se enfocó en la aplicación de colores y logotipos a la plataforma.

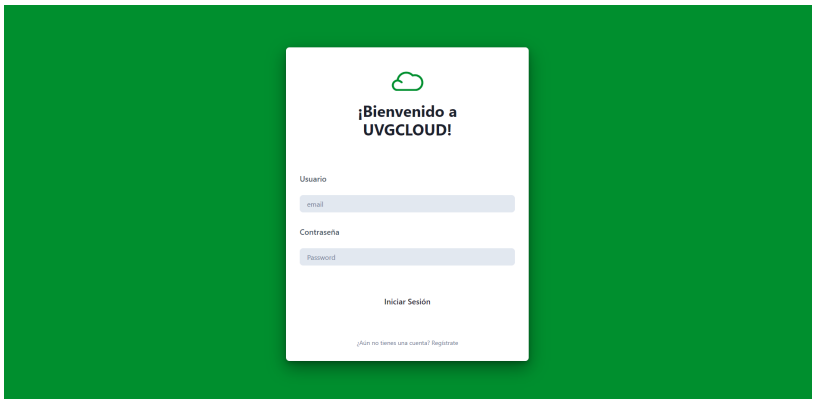

Figura 6.16: Diseño de Login

Figura 6.17: Diseño de visualización de instancias en cards (vacio)

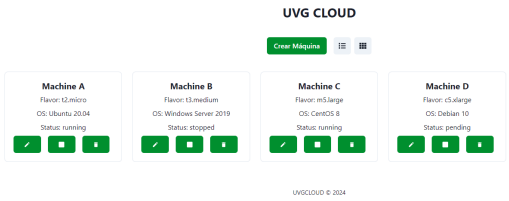

Figura 6.18: Diseño de visualización de instancias en cards (con instancias creadas)

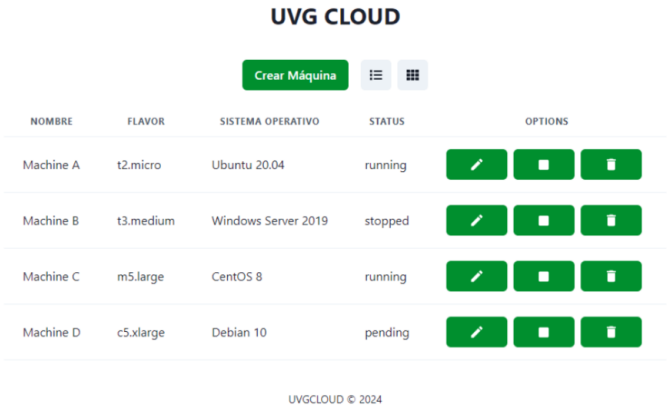

Figura 6.19: Diseño de visualización de instancias en tabla (con instancias creadas)

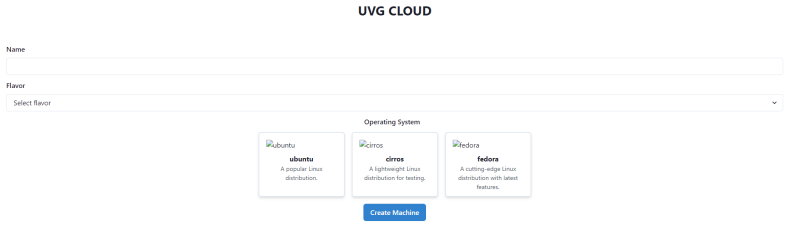

Figura 6.20: Diseño de crear una instancia

Se puede apreciar que son cambios mínimos. Esta versión fue desafiada con una encuesta con el objetivo de obtener *feedback* del público objetivo.

- **P**antalla de Inicio de Sesión (Login) La mayoría de los usuarios valoraron el enfoque minimalista e intuitivo del diseño de la pantalla de inicio de sesión, destacando su simplicidad y organización clara. Sin embargo, algunos consideraron que ciertos detalles, como el espaciado entre los campos y la visibilidad del botón de Ïniciar sesión", necesitaban mejoras para optimizar la legibilidad y la usabilidad. También sugirieron añadir más elementos visuales al fondo, como un patrón discreto, para hacer el diseño más atractivo sin comprometer su sencillez.
- **P**antalla para Visualizar Instancias El diseño de la pantalla de visualización de instancias fue percibido como moderno, funcional y fácil de interpretar. Los usuarios elogiaron la claridad de la información y la disposición de los elementos, aunque algunos señalaron que el tamaño del ícono de la nube, cuando no había instancias, era demasiado grande. También sugirieron ajustar el tamaño y consistencia de los íconos, mejorar la paleta de colores y aprovechar mejor el espacio vacío en la pantalla para hacer el diseño más equilibrado y estéticamente agradable.
- **P**antalla para Crear Instancia Aunque los usuarios encontraron el diseño intuitivo y en línea con el estilo general del sistema, varios comentaron que la disposición y el tamaño de los campos podrían haberse mejorado. Algunos señalaron que la pantalla se sentía desconectada del resto del diseño, con un uso excesivo de espacio vacío y una falta de jerarquía visual en los elementos. Las sugerencias incluyeron ajustar el espaciado y utilizar colores más consistentes para alinearse mejor con el resto de la interfaz.

Como se puede apreciar tanto en las gráficas como en las opiniones generales recogidas, la percepción sobre el diseño es bastante positiva. Los usuarios han mostrado un alto grado de satisfacción, y aunque han hecho algunos comentarios, la mayoría de ellos se han centrado en aspectos menores que requieren ajustes o detalles por mejorar. Es importante destacar que no se han reportado problemas graves que puedan comprometer significativamente la experiencia del usuario. Este *feedback* refleja que el diseño actual cumple con las expectativas y que las sugerencias se enfocan más en perfeccionamientos que en correcciones estructurales.

6.4.5. Cuarta versión de diseño funcional

Considerando las opiniones obtenidas e la última encuesta, se desarrollaron actualizaciones en base a las sugerencias de los usuarios. Los cambios más significativos se hicieron en la sección de creación y edición de instancias, debido a que fue la peor recibida y la que más comentarios de mejora tuvo.

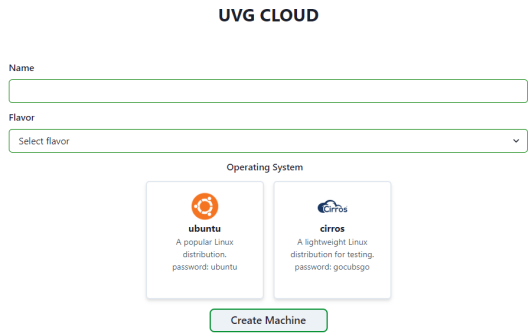

Figura 6.24: Diseño de sección de creación de máquinas virtuales

6.5. Interacción con proveedor de infraestructura en la nube

6.5.1. Despliegue con Kolla-Ansible en un contenedor de Docker

En las primeras etapas del proyecto, se tomó la decisión de utilizar Kolla-Ansible como la herramienta principal para desplegar Openstack. Esta elección se basó en su capacidad para automatizar el despliegue y la gestión de servicios de Openstack, lo que lo hace ideal para entornos complejos de nube. Con el objetivo de facilitar la migrabilidad de la aplicación a otros sistemas, se planteó la idea de contenerizar todo el entorno dentro de un único contenedor de Docker. Este enfoque permitiría encapsular todas las dependencias y configuraciones necesarias, garantizando que la aplicación pudiera ejecutarse de manera consistente en diferentes entornos, independientemente del sistema operativo o la infraestructura subyacente. Para lograrlo, se decidió construir un Dockerfile basado en Ubuntu, donde se instalarían todas las herramientas y componentes requeridos por Kolla-Ansible. El uso de un único contenedor Docker no solo simplifica la gestión y el despliegue del sistema, sino que también facilita su replicación en otros servidores. Esta estrategia de contenerización asegura que el sistema pueda ser migrado o escalado sin problemas, manteniendo al mismo tiempo la integridad de las configuraciones y el funcionamiento óptimo de la plataforma.

```
# Usa la imagen base de Ubuntu 22.04
FROM ubuntu:22.04

ENV DEBIAN_FRONTEND noninteractive

# Actualiza el índice de paquetes e instala los paquetes necesarios
RUN apt-get update


# Actualiza el índice de paquetes nuevamente
RUN apt-get update

# Instala los paquetes necesarios (incluyendo OpenStack y FastAPI)
RUN apt-get install -y \
    python3-pip \
    && pip3 install fastapi uvicorn

# RUN apt-get install -y docker.io


#CMD ["bash", "-c", "tail -f /dev/null"]

# Copiar los archivos de tu aplicación (si los tienes) al contenedor
# COPY . /app

# Comando para iniciar tu aplicación FastAPI (si tienes uno)
WORKDIR /project


### OPENSTACK
# Instala las dependencias necesarias
ENV USER=root

RUN apt-get update && \
    apt-get install -y \
        git \
        python3-dev \
        libffi-dev \
        gcc \
        libssl-dev \
        python3-venv \
        sudo && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# Crea un directorio de trabajo
WORKDIR /project
```

Figura 6.25: Dockerfile del despliegue con kolla-Ansible

El Dockerfile creado para esta versión fue diseñado para contener todos los comandos y configuraciones necesarias para levantar el servicio de Kolla-Ansible, considerado como el núcleo crítico del funcionamiento de la aplicación. Dada la importancia de este servicio, se optó por centrar los esfuerzos iniciales únicamente en su implementación, con la intención de asegurar su correcto fun-

cionamiento antes de proceder con la integración de otros componentes.

Sin embargo, durante el proceso de despliegue, se encontraron varios desafíos técnicos. Específicamente, los requerimientos de Kolla-Ansible presentaron conflictos con ciertas configuraciones inherentes a los contenedores de Docker. Estos conflictos resultaron en una serie de errores que impidieron el correcto inicio del servicio. Estos estaban relacionados con la imposibilidad de la descarga de una imagen específica de Docker necesaria para la configuración del servicio y problemas con un recurso específico que estaba siendo utilizado por Docker:

Figura 6.26: DoConjunto de errores al implementar Kolla-Ansible

Estos errores son muy específicos. A pesar de realizar múltiples ajustes y consultar diversas respuestas en foros de la comunidad, los problemas persistieron, impidiendo el correcto funcionamiento del servicio. Después de varios intentos fallidos y una exhaustiva revisión de posibles soluciones, decidí reevaluar la estrategia de contenerización. Como resultado, opté por contenerizar todos los servicios necesarios para la aplicación, excepto Openstack. Esta decisión se basó en la necesidad de garantizar la estabilidad y funcionalidad del sistema, priorizando aquellos componentes que podían integrarse de manera más efectiva dentro del entorno contenerizado.

6.5.2. Despliegue con Kolla-Ansible en Máquina virtual Ubuntu Server

El desarrollo en esta fase se complicó considerablemente debido a las exigentes configuraciones de Kolla-Ansible. La instalación en modo desarrollador presentó numerosos problemas, en particular errores relacionados con la imagen de Docker mencionados anteriormente. Aunque parecía que la única solución viable era un despliegue en modo producción, esta opción fue descartada debido a que los recursos requeridos superaban la capacidad asignable a una máquina virtual en mi sistema. La principal ventaja de Kolla-Ansible era precisamente la flexibilidad de elegir entre modos de

despliegue, una característica que consideré crucial para mantener un desarrollo homogéneo.

Ante estos desafíos, opté por investigar alternativas como Microstack, herramienta reconocida por su capacidad para realizar despliegues rápidos y eficientes de Openstack. Con base en estas investigaciones, decidí que estas alternativas serían más adecuadas para continuar con el desarrollo del proyecto. Además, planeé realizar una evaluación más detallada para seleccionar una tecnología idónea para el despliegue en producción.

6.5.3. Despliegue con Microstack en Ubuntu Server

Se optó por realizar el despliegue utilizando Microstack para desarrollo tanto como para producción, dado que proporciona una solución rápida y eficiente a los problemas que se presentaron relacionados con el despliegue de Openstack. Aunque puede parecer contradictorio usar Microstack tanto para el entorno de desarrollo como para producción debido a las demandas típicas de un sistema en producción, en este caso particular se considera una elección adecuada debido a que el proyecto es una prueba de concepto.

Microstack representa un punto de partida sólido para un sistema más complejo y ofrece comandos que automatizan tareas cruciales como la asignación de IP y la configuración de SSH, facilitando así un despliegue ágil y eficiente de instancias. Esta tecnología no solo permite validar la funcionalidad de la plataforma, sino que también asegura que el servicio de despliegue de instancias sea operativamente viable. Los pasos detallados para el despliegue de Microstack están explicados en el siguiente video: Link

Microstack fue desplegado y como prueba de ello adjunto foto del login de Horizon, conectandome desde el navegador:

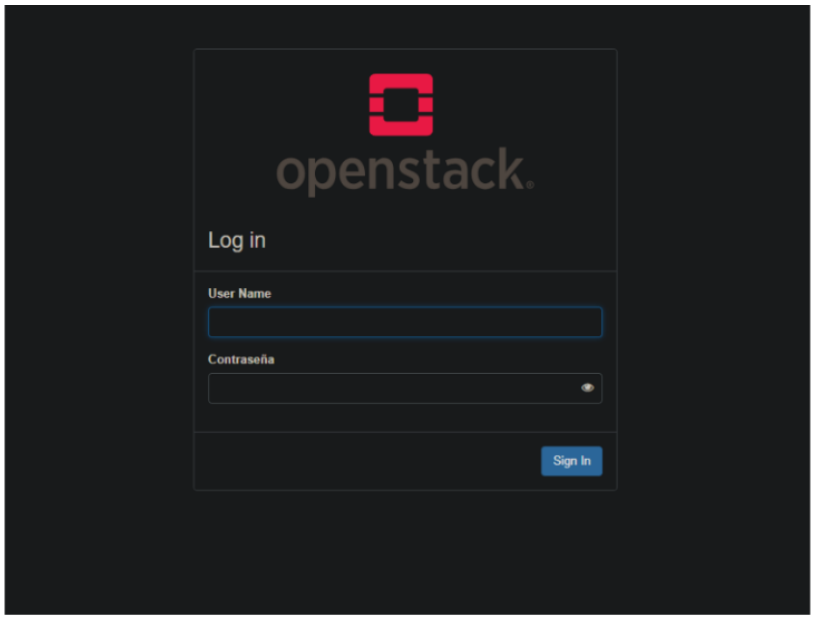

Figura 6.27: Dashboard de Horizon

6.5.4. Problemas con Microstack durante el desarrollo

Durante el desarrollo, despliegue en servidores y pruebas de usabilidad, surgieron varios problemas con Microstack, derivados de limitaciones en la propia herramienta de Openstack. Uno de los problemas principales fue la imposibilidad de utilizar la función *resize*. Esta funcionalidad, que debería cambiar el estado de la máquina a *resize* para permitir ajustes en los recursos asignados, no funcionó correctamente, ya que la máquina no lograba mantener el estado y regresaba automáticamente a su estado original, en contradicción con el proceso descrito en la documentación oficial de Openstack.

- Microstack Documentation. `https://microstack.run/docs`
- Openstack Nova Documentation. `https://docs.openstack.org/nova/train/user/resize.html`

Además, en las pruebas de usabilidad y despliegue de la plataforma en los servidores, se presentaron errores adicionales relacionados con la asignación de IP flotantes y problemas de autenticación en el servicio Keystone. Estos problemas resultaron ser complejos y específicos; al investigar soluciones, tanto la *documentación oficial de Microstack* [**?**] como la de Openstack [**?**] no ofrecieron información que abordara directamente estos inconvenientes. Tampoco se encontraron respuestas útiles en foros de discusión o comunidades de usuarios, lo que limitó las opciones de resolución.

Ante esta situación, se optó por realizar reinicios periódicos tanto de la herramienta como del servidor. Esta solución temporal permitió resolver algunos errores de despliegue de instancias, como los problemas de autenticación y asignación de IP, sin embargo, el error en la funcionalidad de *resize* persistió. La máquina pasaba al estado *Resize* pero retornaba de inmediato a su estado inicial, sin cumplir el proceso descrito en la documentación y sin que se encontrara una solución permanente.

6.6. Desarrollo del *backend* y despliegue de base de datos.

6.6.1. Creación de archivo docker-compose con despliegue de base de datos

Como se mencionó anteriormente, uno de los objetivos clave era facilitar la migrabilidad de la aplicación a otros sistemas. Inicialmente, se consideró la posibilidad de contenerizar todo el entorno dentro de un único contenedor de Docker. Sin embargo, debido a los problemas enfrentados en la fase inicial de "Pruebas con proveedor de infraestructura en la nube", esta idea fue finalmente descartada.

En lugar de ello, se decidió adoptar un enfoque basado en múltiples contenedores, donde cada dependencia necesaria se separa y se gestiona de manera individual. Esto se logró utilizando Docker Compose, lo que permitió levantar y gestionar cada servicio de forma más eficiente y organizada.

El proceso de configuración de la base de datos fue particularmente sencillo. Solo fue necesario descargar la imagen de PostgreSQL correspondiente y definir las configuraciones adecuadas en el archivo de Docker Compose. Una de las configuraciones más cruciales fue la integración de volúmenes de Docker.

Se montó un volumen entre la carpeta que contiene todos los datos de la base de datos y un directorio vacío, creado específicamente en la máquina *host*. Este volumen proporciona una forma fácil de acceder a los datos y realizar copias de seguridad, además de servir como una medida de seguridad adicional. En caso de que el contenedor se detenga inesperadamente, el volumen asegura que los datos de la base de datos se repliquen y estén protegidos.

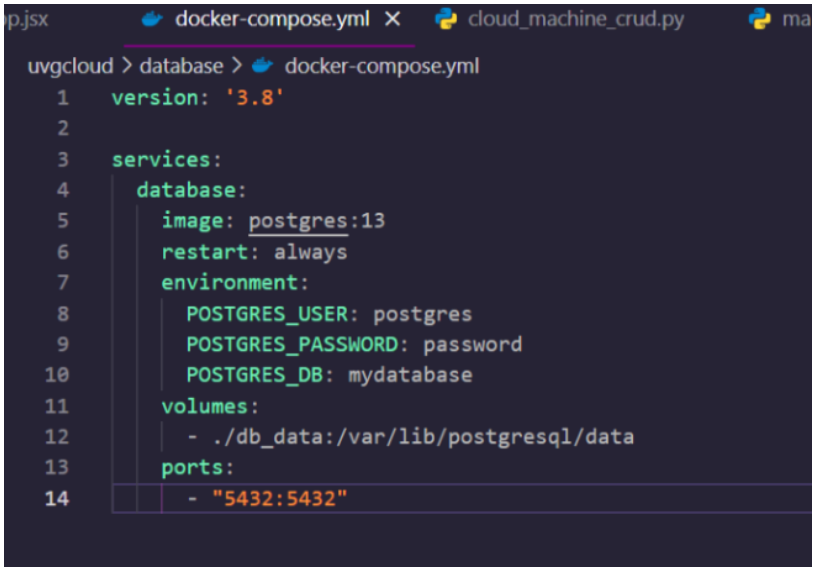

Figura 6.28: Docker-compose.yml para SQL

6.6.2. Integración de Python con FastAPI en contenedor a la plataforma

Esta integración requirió de la instalación de herramientas que nos permitan la interacción del *frontend* y con la base de datos en sql. Estas se especifican en el archivo requirements.txt y son las siguientes:

- fastapi
- uvicorn[standard]
- sqlalchemy
- psycopg2-binary
- alembic
- pydantic-settings
- pydantic
- bcrypt
- websockets

Para el manejo del API se requiere de FastAPI y Uvicorn. Para la interacción entre la base de datos y el API se requiere de sqlalchemy y alembic.

Antes de comenzar a construir el Dockerfile, se necesitaron pruebas desde la máquina *host* hacia la base de datos para verificar el correcto funcionamiento de cada tecnología integrada. Por lo que se comenzó con la verificación de la conexión y creación de tablas:

```python
import time
from fastapi import FastAPI
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

app = FastAPI()

# Database setup
DATABASE_URL = "postgresql://postgres:password@db:5432/mydatabase"
engine = None

# Function to check if PostgreSQL is ready
def check_postgres():
    try:
        engine = create_engine(DATABASE_URL)
        with engine.connect():
            print("Connected to PostgreSQL!")
            return engine
    except Exception as e:
        print(f"PostgreSQL is not ready yet: {e}")
        return None

# Wait until PostgreSQL is ready
while not (engine := check_postgres()):
    time.sleep(1)  # Wait for 1 second before retrying

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

# Example route
@app.get("/")
def read_root():
    return {"Hello": "World"}
```

Figura 6.29: Archivo de Python que conecta a la base de datos

Al aplicar un upgrade a la base de datos con Alembic obtenemos un directorio con estas versiones. Al cumplir estos requisitos y obtener respuesta exitosa, podemos proceder a construir el Dockerfile.

Code Blame 25 lines (23 loc) · 475 Bytes Code 55% faster with GitHub Copilot

```yaml
version: '3.8'

services:
  db:
    image: postgres:13
    restart: always
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: password
      POSTGRES_DB: mydatabase
    volumes:
      - ./db_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  backend:
    build: .
    ports:
      - "8000:8000"
    depends_on:
      - db
    environment:
      DATABASE_URL: postgresql://postgres:password@db:5432/mydatabase
    volumes:
      - ./app:/app
```

Figura 6.30: Archivo Docker-compose integrando el *backend* y la base de datos

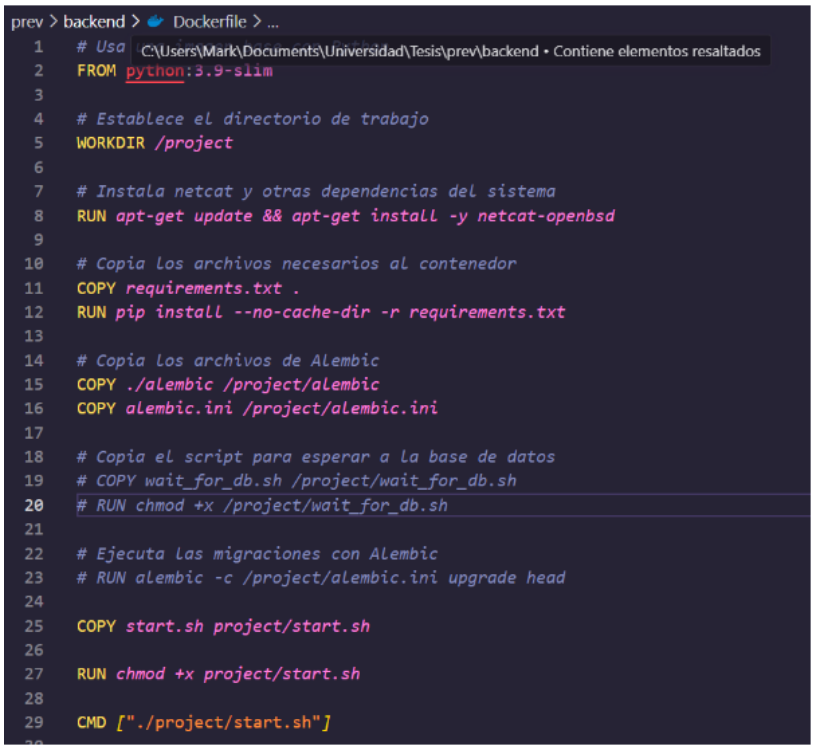

Figura 6.31: Dockerfile para configurar el proyecto de Python

Figura 6.32: Bash *script* para espera a que el servicio "db.^estté en ejecución

Debido a que necesitamos que la base de datos complete su despliegue antes de ejecutar el API, declaramos un bash *script* que es ejecutado al final del Dockerfile, este se encarga de esperar a que el proceso db (nombre otorgado a este servicio en el archivo Docker-compose.yml) para que comience el upgrade de la última versión de alembic registrada en el proyecto y finalmente se ejecute el API. Esto se hace con el objetivo de aplicar los cambios en tablas y estructura de la base de datos cada vez que la paltaforma se despliegue.

6.6.3. Estructura de API

Se decidió desarrollar un *endpoint* específico para cada tabla de la plataforma.

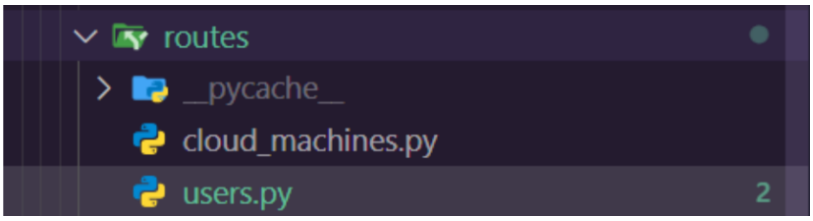

Figura 6.33: Estructura del API (proyecto de Python)

Los datos son enviados a través de query parameters y documentos desde el *frontend* y son verificados con ayuda de la librería pydantic.

Figura 6.34: Archivo de *endpoint*: usuarios

Los esquemas de Pydantic son utilizados en las rutas para tener un estándar de nombres y datos a la hora de recibir información, esto para facilitar la comunicación entre el *frontend* y *backend*.

6.6.4. Sistema de inicio se sesión y creación de tabla usuario

Se comenzó con esta tabla, permite un sistema de inicio de sesión sencillo. Una vez teniendo la estructura requerida, se comenzó con la construcción del enrutamiento.

```python
import uuid
from sqlalchemy import Column, String
from sqlalchemy.dialects.postgresql import UUID
from app.db.base import Base

class User(Base):
    __tablename__ = 'Users'
    id = Column(UUID(as_uuid=True), primary_key=True, default=uuid.uuid4, unique=True, index=True)
    email = Column(String, unique=True, index=True)
    password = Column(String)

    def __str__(self):
        return f'email: {self.email}'
```

Figura 6.35: Archivo de Python para crear una tabla de usuarios en SQL

Cada ruta de "/users" contiene una operación CRUD correspondiente.

```python
# app/db/crud.py

from sqlalchemy.orm import Session
from app.models.users import User
import uuid
from app.utils.hashing import hash_password, verify_password

# CREATE
def create_user(db: Session, email: str, password: str) -> User:
    user_id = uuid.uuid4()  # Generar un UUID para el nuevo usuario
    hashed_password = hash_password(password)
    db_user = User(id=user_id, email=email, password=hashed_password)
    db.add(db_user)
    db.commit()
    db.refresh(db_user)
    return db_user

# READ
def get_user(db: Session, user_id: uuid.UUID) -> User:
    return db.query(User).filter(User.id == user_id).first()

def get_user_by_email(db: Session, email: str) -> User:
    return db.query(User).filter(User.email == email).first()

def authenticate_user(db: Session, email: str, password: str) -> User:
    user = get_user_by_email(db, email)
    if user and verify_password(password, user.password):
        return user
    return None
```

Figura 6.36: Archivo de Python para operaciones CRUD

Cabe destacar que en los métodos "createuser" y "authenticateuser" encriptamos la contraseña con ayuda de la librería bcrypt, de forma que la contraseña es encriptada antes de almacenarse en la base de datos, y la forma de autenticarse es comparando la contraseña ingresada desde el *frontend* en su versión encriptada con la almacenada en la base de datos.

6.6.5. Ejecución de un comando solicitado por el *frontend* en la consola del servidor

Para ejecutar esta tarea requerimos de la librería subprocess y la creación de una nueva ruta enfocada en las instancias de uvgcloudmachines.

```python
# Montar las rutas de los comandos
app.include_router(cloud_machines.router, prefix="/cloud_machines", tags=["cloud_machines"])
```

```python
# Dependencia para obtener la sesión de la base de datos
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()

@router.get("/")
def user_root():
    return {"Hello": "Commands"}

@router.get("/do_command/", response_model=None)
def do_command(db: Session = Depends(get_db)):
    result = subprocess.run(['ls'], stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
    # Almacenar la salida en una variable
    output = result.stdout
    error = result.stderr
    print(output)
    print(error)
    return [output, error]
```

Figura 6.37: Código de Python para crear y agregar la ruta: commands

Al acceder al *endpoint* correspondiente a "docommand"se realiza un ls con la audia de subprocess. Esta fue una prueba para verificar el funcionamiento de la aplicación.

En el *frontend* se desarrolló un botón para hacer una solicitud a esta dirección, de forma que al clickar en él se ejecute el comando correspondiente en la consola del servidor que ejecuta el *backend*.

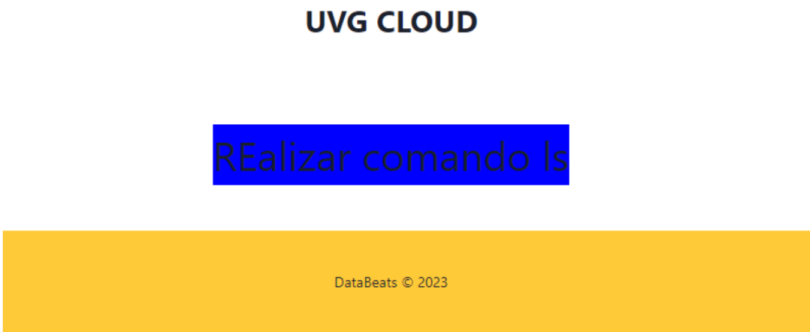

Figura 6.38: Proyecto web para solicitar un comando "ls.ªl *backend*

Figura 6.39: Despliegue del resultado de "ls.en los logs del servidor

El resultado del comando fue expuesto en los logs del servidor de FastAPI. El correcto funcionamiento es evidenciado al enumerar los archivos que se encuentran en el directorio del archivo principal.

6.6.6. Migración de API al sistema *host*

A pesar de haber obtenido un resultado exitoso se pasó por alto una consideración crítica, este comando es ejecutado en el contenedor de Python, no en la máquina *host*. A primera vista, la solución puede parecer sencilla, solo necesitamos encontrar la forma de redirigir este comando a la máquina *host*.

Uno de los principios clave de un contenedor es el aislamiento. El aislamiento de contenedores es la práctica de mantener los contenedores, que son unidades ligeras y aisladas que encapsulan una aplicación junto con sus dependencias, de manera separada e independiente entre sí y del sistema anfitrión (NordVPN, 2024). Por lo que realizar esta operación pasa por alto las buenas prácticas de contenedorización.

Aún así, se procedió con la búsqueda de una solución a este problema, sin embargo, no existen muchos enfoques viables. La mayoría de información está dirigida a la ejecución de comandos desde *host* a un contenedor, el mejor artículo que se pudo encontrar fue el siguiente: How to run shell *script* on *host* from docker container?

En esta dirección se habla acerca de una posible solución a través de la construcción de un pipe que es escuchado de forma infinita por el sistema, lamentablemente no tengo capturas del proceso. Enfrenté 2 problemas: una carpeta compartida de virtualbox contiene algunas restricciones, una de estas no permitió crear el archivo pipe en mi directorio de trabajo, al realizar pruebas con él, se logró la ejecución de un comando pero no la escucha infinita para que cualquier comando sea ejecutado.

Ante esta clara limitación, se decidió migrar el API al *host*, de forma que el único servicio contenedorizado sería el de la base de datos.

6.6.7. Pruebas con instancias en el API (uvgcloudmachines)

Una vez ya establecida una estructura sólida para el proyecto, se procedió a integrar los comandos de Microstack en el API.

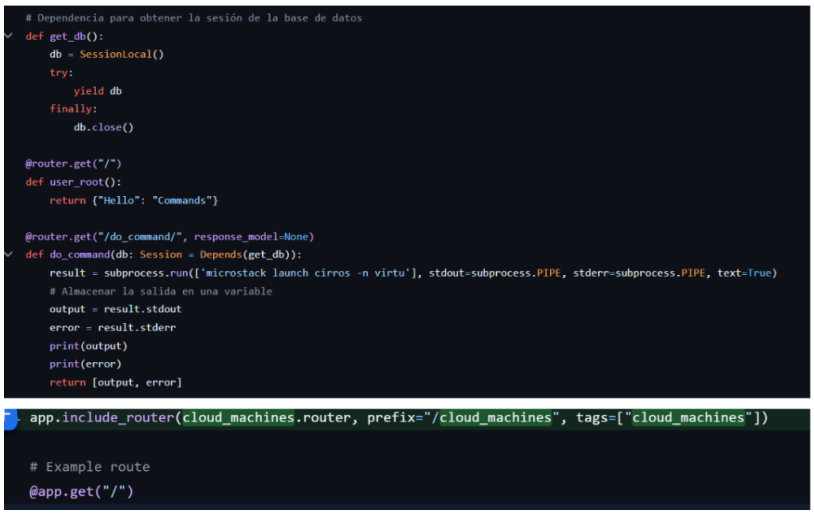

Figura 6.40: Código de Python para integrar el comando "launch"de Microstack

Se cambió el nombre de la dirección correspondiente y se añadió un comando de prueba de Microstack, este tiene el objetivo de crear una máquina de tamaño default, con sistema operativo

cirros y de nombre “virtu”. Al ejecutar esta función obtuvimos la máquina desplegada en Horizon:

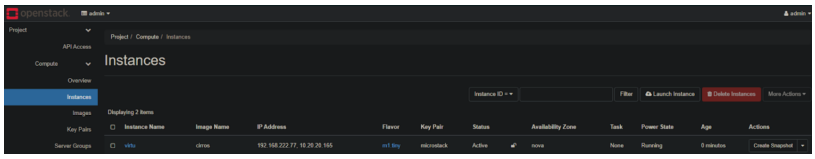

Figura 6.41: Tabla de instancias creadas en Horizon

Una vez verificado el despliegue de una máquina de forma exitosa, se procedió con el desarrollo de creación de máquinas según parámetros recibidos por el *frontend* y el almacenamiento de la información de las instancias en la base de datos.

6.6.8. Creación de instancias (uvglcloudmachines) y almacenamiento de su información en la base de datos

Para almacenar una instancia en la base de datos se optó por esta estructura:

```python
import uuid
from sqlalchemy import Column, String, Text
from sqlalchemy.dialects.postgresql import UUID
from app.db.base import Base

class Cloud_Machine(Base):
    __tablename__ = 'Cloud_machines'
    id = Column(String, primary_key=True, default=uuid.uuid4, unique=True, index=True)
    owner = Column(UUID(as_uuid=True), nullable=False, index=True)
    vm_name = Column(String, index=True)
    vm_size = Column(String, index=True)
    os = Column(String, nullable=True)
    status = Column(String, index=True)
```

Figura 6.42: Código de Python para crear una tabla para Cloud Machines

Debido al funcionamiento de Openstack, el id único creado en el *backend* se utilizó para nombrar la instancia en Openstack, esto con el objetivo de facilitar comandos de edición y eliminación de instancias. El usuario podrá ver el nombre que le asignó a su instancia en el *frontend*, pero en el *backend* y Openstack esta se maneja con el id asignado en *backend*:

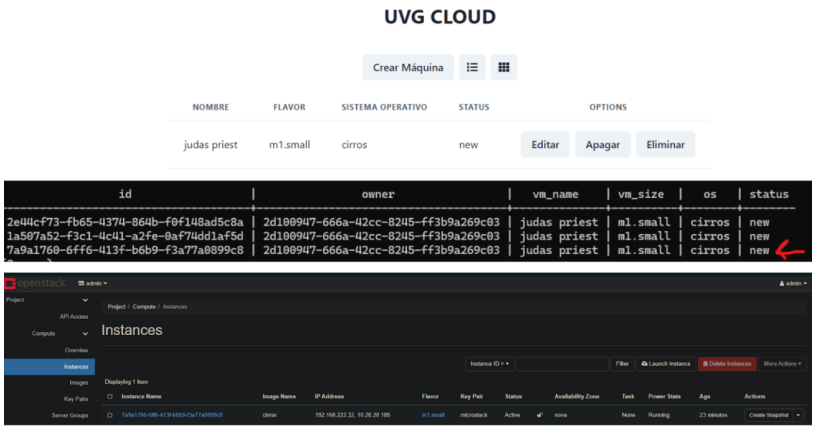

Figura 6.43: Conjunto de imágenes para evidenciar la creación de una máquina con un nombre único

Con esto mejoramos la experiencia de usuario al permitir cualquier nombre para una instancia y se asegura un manejo adecuado de esta en el *backend* y Openstack.

6.6.9. Edición y eliminación de instancias

Se añadieron las funciones de edición y eliminación de máquinas virtuales de nombre de máquinas virtuales. Estas utilizan la misma estructura que la función de crearlas, de forma que se hace la modificación en Openstack y luego se refleja en la base de datos.

```python
# UPDATE
def update_cloud_machine(db: Session, machine_id: str, vm_name: str = None, vm_size: str = None, os:
    # Obtener la máquina existente por su ID
    db_cloud_machine = db.query(Cloud_Machine).filter(Cloud_Machine.id == machine_id).first()

    if not db_cloud_machine:
        return None  # Si no se encuentra la máquina, retornar None o lanzar una excepción

    # Actualizar los campos solo si se pasan nuevos valores
    if vm_name != None:
        db_cloud_machine.vm_name = vm_name
    # if vm_size:
    #     db_cloud_machine.vm_size = vm_size
    # if os:
    #     db_cloud_machine.os = os

    # Commit para guardar los cambios
    db.commit()
    db.refresh(db_cloud_machine)  # Refrescar la instancia para obtener los valores actualizados
    return db_cloud_machine
```

```python
DELETE
ef delete_cloud_machine(db: Session, machine_id: str):
    # Buscar la máquina por su ID
    machine_to_delete = db.query(Cloud_Machine).filter(Cloud_Machine.id == machine_id).first()

    # Si la máquina existe, la elimina
    if machine_to_delete:
        db.delete(machine_to_delete)
        db.commit()  # Confirmar los cambios en la base de datos
        return True  # Retorna True si fue eliminada exitosamente
    else:
        return False  # Retorna False si la máquina no fue encontrada
```

Figura 6.44: Código de Python para eliminar y editar Cloud Machines en la base de datos

6.7. *scripts* de automatización

6.7.1. Utilización de comandos incluidos en Microstack

Para realizarlos se requirió de establecer la distribución que desplegará Openstack. Como se mencionó anteriormente en "Interacción con proveedor de infraestructura en la nube", se optó por utilizar Microstack. La ventaja de desplegarlo con esta herramienta es que provee *scripts* automatizados en su instalación, lo que facilita la realización de las siguientes tareas:

- **Despliegue de instancia personalizada.** El comando utilizado es el siguiente:

```
usage: launch [-h] [-n NAME] [-k KEY] [-f FLAVOR] [-t NET_ID] [-w] [-r]
[--availability-zone AVAILABILITY_ZONE] image
```

La plataforma utiliza `-n`, `-f` e `image` para desplegar una instancia.

asegurar que todas las instancias creadas por los usuarios se actualicen al estado "running", es decir, que puedan utilizarse. Se estableció como una rutina a ejecutarse al encender el servidor, lo que garantiza que las máquinas estén en funcionamiento.

```
# Obtener la lista de instancias en estado SHUTOFF y reiniciarlas
instance_ids=$(microstack.openstack server list --status SHUTOFF -f value -c ID)

for instance_id in $instance_ids; do
    echo "Iniciando la instancia $instance_id"
    microstack.openstack server start "$instance_id"
done
```

Figura 6.46: Bash *script* para poner en ejecución a todas las instancias

6.8. Despliegue de la plataforma en servidores de la Universidad del Valle de Guatemala

6.8.1. Git pull del repositorio junto con creación de ReadMe

Este objetivo se desarrolló en una etapa tardía del proyecto, para asegurarse de tener un prototipo funcional y valga la pena una instalación en el servidor. Esta decisión cobra más importancia al conocer las circunstancias del servidor objetivo, debido a que el acceso a este es restringido y requirió de procesos ajenos a mi responsabilidad.

Una vez completadas las funciones de registro, creación y edición de instancias se procedió con la creación de ReadMe en el repositorio, esta primera revisión se hizo enfocada en indicar las instrucciones para instalar y desplegar la plataforma. Una vez en el servidor, se clonó el repositorio y se instalaron todas las dependencias: requerimientos de Python, Microstack y Docker. En base a las instrucciones fue posible desplegar la aplicación en el servidor y realizar los procesos correspondientes para desplegar una máquina virtual

Esta instalación fue previa a la segunda encuesta de diseño del *frontend*, por lo que el desarrollo continuó en la máquina virtual. A partir de este punto, basta con actualizar la rama main en el servidor para asegurarse de contar con la última versión del proyecto.

6.8.2. Conexión de usuarios a máquinas virtuales

Como se mencionó en el proceso de creación de scipts de automatización, es necesario aplciar portfowarding para permitir la conexión de los usuarios a las máquinas creadas. La forma de establecer esta conexión es asignar un puerto del servidor a la dirección IP creada por Microstack para una máquina virtual, para ello debemos conocer nuestra IP pública, esta se consiguió con un ifconfig en el servidor.

CAPÍTULO 7

Resultados

En base a la planificación indicada en la metodología podemos presentar el resultado obtenido de cada sección de la misma.

7.1. Definición de tecnologías

- Se utilizó React con el module bundler Vite.
- Se utilizó Python para desarrollar el proyecto de API
- Se utilizó un contnedor Docker de SQL para la base de datos
- Se utilizó Openstack desplegado con Microstack para la infraestrcutura en la nube

7.2. Desarrollo del *frontend* del proyecto

El *frontend* obtenido consistió en un proyecto de React con 4 secciones diferentes:

Cada sección incluyó los cambios solicitados en la última encuesta. Siendo este el prototipo funcional final del *frontend* para el proyecto.

Este proyecto permite realizar solicitudes para:

- Crear un usuario
- Iniciar sesión
- Visualizar instancias creadas junto con información de la misma
- Visualizar si no se tiene una instancia creada
- Creación de una instancia con 3 diferentes parámetros: nombre, flavor y sistema operativo

- Reboot de una instancia
- Eliminación de una instancia

7.3. Desarrollo del *backend* del proyecto

El resultado consiste en un proyecto que integra Python y Docker. Se utiliza un contenedor de Docker para desplegar una base de datos SQL. El proyecto de Python funciona como el API del proyecto, este es capaz de comunicarse con la base de datos integrando Alembic para:

- Ingresar un usuario
- Verificar la información de un usuario
- Ingresar una instancia
- Asociar una isntancia a un usuario
- Borrar instancias
- Modificar información de una instancia

El proyecto de Python integra subprocess para podere comunicarse con la línea de comandos del servidor y ejecutar *scripts* y comandos que permiten:

- Crear una instancia en Openstack
- Reiniciar una instancia
- Eliminar una instancia
- Asignar un puerto a la ip flotante de una instancia

El archivo docker-compose despliega el contenedor. este monta un volumen para permitir la persistencia de los datos frente a cada arranque del contenedor, además de presentar la ruta donde se encuentra toda la información de la base de datos.

7.4. Desarrollo de *scripts* de automatización

Se desarrollaron 2 *scripts* de automatización.

- *Script* de arranque automático de instancias: consiste en un bash *script* que realiza una iteración entre todas las isntancias desplegadas para iniciarlas.
- *Script* de creación de máquinas virtuales: consiste en un bash *script* que despliega una instancia de Openstack, le asigna un puerto del equipo a la ip flotante de la misma, permite el tráfico al puerto y dirige el tráfico al puerto 22 de la ip flotante

7.5. Despliegue de la plataforma en servidores de la Universidad

El proyecto se encuentra en ejecución en los servidores, es posible realizar cambios y actualizar el servidor clonando el repositorio del proyecto. El servidor fue configurado para permitir el alojamiento de ips flotantes en sus puertos. El servidor mantiene el *frontend*, *backend*, base de datos y Openstack en ejecución.

7.6. Pruebas con usuarios de primera versión funcional del proyecto

La prueba fue realizada con 5 participantes que cumplían las características del perfil de usuario de la prueba. Cada tarea fue calificada con un "sísi, con preguntasz "no". Cada una referenciando a los 3 resultados establecidos en la metodología de la prueba.

Nomenclatura para tareas:

1. Crear un usuario
2. Iniciar sesión
3. Crear una instancia con sistema operativo cirros
4. Conectarse mediante SSH a la instsancia creada
5. Editar el nombre de la instancia
6. Hacer un reboot a la instancia creada
7. Eliminar la instancia creada

Usuarios	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	Si	Si	Si	no	Si	Si, con preguntas	Si
2	Si	Si	Si	no	Si, con preguntas	Si	Si
3	Si	Si	Si, con preguntas	Si	Si, con preguntas	Si	Si
4	Si	Si con preguntas	Si, con preguntas	no	Si	Si, con preguntas	Si
5	Si	Si	Si	no	Si	Si, con preguntas	Si

Tabla 7.1: Cuadro de resultados de pruebas con usuarios sobre las tareas a evaluar

Categorización de preguntas por cada sección:

- Iniciar sesión
  1. ¿Mi usuario se pudo crear?
- Crear una instancia con sistema operativo cirros
  1. ¿Cuáles son las specs de cada flavor?
  2. ¿Qué significa flavor y cada uno de los que puedo seleccionar?
- Conectarse mediante ssh a la instancia creada

1. ¿Cómo me conecto?
2. ¿Cuál es el usuario que debo utilizar?
3. ¿Hay un botón para conectarse?
4. ¿Cuál es la contraseña para ingresar a cirros?
5. ¿Tengo que usar puTTy o la pueod usar en la página?

- Editar el nombre de la instancia
  1. ¿Hay restricción de caracteres?
- Hacer un reboot a la instancia creada
  1. ¿Cuál es el botón para hacer el reboot?

Comentarios generales sobre cada sección

- Crear un usuario
  1. Podrías añadir un mensaje para confirmar la creación del usuario
  2. Me agrada la estética de esta sección
- Iniciar sesión
  1. Me parece un login básico
- Crear una instancia con sistema operativo cirros
  1. Deberias añadir una descripción para cada flavor
  2. Estaría genial que pudieramos agregar fedora
  3. Quizás añadir una descripción a las instancias seria genial
  4. Deberías de añadir una consola o logs para indicar el estado del proceso de creación.
- Conectarse mediante ssh a la instsancia creada
  1. Deberías de añadir un tutorial para conectarse desde ssh
  2. Sería útil añadir una opción para copiar el comando para conectarse a SSH
  3. Deberáis de considerar a la gente que no sabe como conectarse desde ssh a algún equipo.
- Editar el nombre de la instancia
  1. Deberías de añadir verificaciones para nombres inapropiados
  2. Deberías de añadir verificaciones para no poner un nombre nulo
- Hacer un reboot a la instancia creada
  1. Deberías de añadir elementos para resaltar más los botones de esta sección.
  2. Podrías agregar un texto al poner el mouse sobre el botón para indicar que hace cada botón.
- Eliminar la instancia creada
- Me agradan los íconos de esta sección

CAPÍTULO 8

Discusión de resultados

La discusión será desarrollada a través de los objetivos del proyecto.

8.1. Desarrollar una API que capte las solicitudes del *frontend* y se comunique con la línea de comandos para permitir la creación y gestión de máquinas virtuales de manera automática, facilitando su creación desde el *frontend*.

El API capta las solicitudes del *frontend* y es capaz de producir cambios y consultas ne la base de datos. Además, utiliza subprocess para ejecutar comandos y bash *scripts* en la línea de comandos del equipo *host*. Estas características permiten realizar las funcionalidades más importantes de la plataforma. Es posible concluir que el proyecto cumple con el objetivo propuesto.

8.2. Desarrollar una API que capte las solicitudes del *frontend* y se comunique con la línea de comandos para permitir la creación y gestión de máquinas virtuales de manera automática, facilitando su creación desde el *frontend*.

El API capta las solicitudes del *frontend* y es capaz de producir cambios y consultas en la base de datos. Además, utiliza subprocess para ejecutar comandos y bash *scripts* en la línea de comandos del equipo *host*. Estas características permiten realizar las funcionalidades más importantes de la plataforma:

- Ingresar un usuario
- Verificar la información de un usuario

- Ingresar una instancia
- Asociar una isntancia a un usuario
- Borrar instancias
- Modificar información de una instancia

Estas funcionalidades permiten la creación y gestión de máquinas virtuales de forma automática integrando *scripts*. Siendo estas implementadas según las solicitudes del *frontend*

Es posible concluir que el proyecto cumple con el objetivo propuesto.

8.3. Programar *scripts* automatizados que ejecuten conjuntos de comandos de terminal para facilitar la configuración y despliegue de máquinas virtuales, utilizando herramientas de *scripting*.

El entregable fue un conjunto de *scripts* incluidos de Microstack, junto con 2 *scripts* de automatización. Uno con el objetivo de iniciar todas las máquinas en caso de estar inactivas al iniciar el servidor y el segundo con el objetivo de crear una instancia, asociar y permitir el tráfico hacia el puerto donde la instancia es alojada. Estos se integran en el API para configurar y desplegar máquinas virtuales, siendo indispensables para ejecutar funcioens críticas del proyecto. Se concluyó que este conjunto de *scripts* y comandos cumple con el objetivo

8.4. Programar una interfaz web intuitiva que permita a los usuarios crear y gestionar instancias de máquinas virtuales para proporcionar una experiencia de usuario amigable y accesible, empleando tecnologías web y técnicas de diseño de interfaces de usuario.

El entregable que consistió en un proyecto de React fue puesto a prueba con las pruebas de usabilidad realizadas al terminar el porototipo final del proyecto. Esta prueba otorgó resultados variados para cada sección.

Para las siguientes secciones:

- Crear un usuario
- Iniciar sesión
- Eliminar la instancia creada

Los usuarios comprendieron las secciones que contenían estas tareas, esto se puede ver reflejado al solo tener una pregunta respecto a estas.

Para las siguientes secciones:

- Crear una instancia con sistema operativo cirros
- Editar el nombre de la instancia
- Hacer un reboot a la instancia creada

Fue posible realizar la tarea pero requirió de preguntas relacionadas a parámetros de creación y funcionalidades de botones. Las preguntas de los usuarios se enfocaron en ser más detallado con aspectos como añadir descripciones a cada flavor disponible para crear una máquina y añadir textos o algún otro elemento más aclaratorio además de íconos para especificar las funciones de reboot, edición y eliminación. Nomenclatura para tareas:

- conectarse mediante ssh a la instsancia creada

Las pruebas de usabilidad realizadas al prototipo de interfaz para la gestión de máquinas virtuales evidenciaron que la plataforma es intuitiva para las tareas básicas, como la creación de usuario, el inicio de sesión y la eliminación de instancias. Los usuarios lograron completar estas tareas con facilidad, reflejando una interfaz accesible que apoya la navegación en operaciones simples. Esta efectividad inicial es fundamental para los estudiantes, pues les facilita una entrada amigable en el manejo de entornos de infraestructura en la nube. Sin embargo, en las tareas más específicas, como el entendimiento de los parámetros de creación de máquinas virtuales, la edición de nombres, y el reinicio de instancias, los usuarios encontraron algunas dificultades mínimas, ya que con pocas intervenciones del organizador de la prueba la tarea pudo realizarse. Estas tareas requerían una comprensión más profunda de los parámetros de configuración, lo que motivó a los usuarios a plantear preguntas sobre el uso de ciertos términos técnicos y la funcionalidad de algunos botones. Los comentarios de los usuarios sugirieron que la inclusión de descripciones para los "flavors" o tipos de máquinas virtuales sería muy útil para ayudarles a elegir los recursos más adecuados según sus necesidades y conocimientos. El proceso de conexión SSH, en particular, presentó un reto adicional para los usuarios con menos experiencia técnica. Sin guías claras dentro de la interfaz, algunos usuarios no pudieron completar esta tarea de manera autónoma, lo que resalta la importancia de incluir guías paso a paso o tutoriales que expliquen cómo conectarse por SSH. Estas guías podrían presentarse en forma de ayuda emergente o como una sección de tutoriales dentro de la interfaz, proporcionando a los usuarios los conocimientos necesarios para progresar de manera autónoma en el uso de la plataforma. Además, se observó que algunos íconos y elementos visuales de las funciones de edición, reinicio, eliminación y consulta de la contraseña default de un sistema operativo no eran lo suficientemente claros para ciertos usuarios. Esto sugiere que una iconografía más intuitiva, junto con descripciones emergentes, mejoraría la claridad de las funcionalidades avanzadas y reduciría la necesidad de una capacitación adicional. Incluir iconos más representativos y etiquetas descriptivas en los botones promovería una navegación más intuitiva y fluida. En términos de funcionalidad, el *frontend* mostró un desempeño robusto, sin errores en la estructura del código o en la implementación. Esto confirma que la arquitectura del proyecto es sólida y cumple con los requisitos técnicos, proporcionando una base confiable para el proyecto. En conclusión, el *frontend* desarrollado permite a los usuarios gestionar instancias de máquinas virtuales de forma accesible y amigable, cumpliendo satisfactoriamente con los objetivos de funcionalidad y estabilidad técnica del proyecto. No obstante, para lograr una experiencia de usuario más completa, se recomienda mejorar la claridad en la configuración de instancias y añadir orientación para tareas técnicas como la conexión SSH, mediante descripciones, iconografía adicional y guías de uso. Estos ajustes proporcionarían una navegación más fluida, incluso para usuarios con menos experiencia técnica, garantizando así una experiencia de gestión integral y autónoma.

8.5. Desplegar la plataforma en los servidores de la universidad para optimizar el uso de recursos tecnológicos existentes y proporcionar un entorno práctico para las necesidades de los estudiantes.

No existe un entregable tangible para este proceso, más allá de la confirmación de la instalación exitosa en el servidor universitario. El servidor fue configurado específicamente para aceptar tráfico en los puertos requeridos, permitiendo a los usuarios alojar y gestionar las máquinas virtuales

creadas. Este despliegue aprovechó de manera óptima los recursos tecnológicos existentes en la universidad, ofreciendo un entorno práctico y accesible para que los estudiantes pudieran realizar sus tareas de prueba y experimentación. Además, el flujo de trabajo fue diseñado para que el proyecto en el servidor pudiera ser actualizado fácilmente; bastaba con sincronizar los cambios realizados en la rama principal del repositorio para mantener el sistema al día, facilitando así la gestión y mantenimiento del entorno en tiempo real. Las pruebas de usuario se llevaron a cabo con éxito en esta plataforma en el entorno del servidor, lo cual confirma la correcta implementación y despliegue de la plataforma. Con esta configuración, los usuarios pudieron no solo crear máquinas virtuales sino también conectarse a ellas, validando la funcionalidad y accesibilidad del sistema en un entorno práctico y controlado. Estos procedimientos pueden confirmar que la plataforma fue desplegada correctamente y poroprcionó un entorno óptimo para su funcionamiento, podemos concluir que se cumplió el objetivo.

8.6. Proveedor de infraestructura en la nube

El despliegue de Openstack mediante Microstack fue en general exitoso, aunque se observaron problemas específicos en el proceso de despliegue de máquinas virtuales y en la funcionalidad de *resize*. Microstack ofrece un entorno optimizado para realizar pruebas con Openstack, lo cual se ajusta bien al concepto de este proyecto como una prueba de concepto de computación en la nube. Sin embargo, los problemas encontrados indican que, aunque Microstack fue una opción adecuada para la fase experimental, no es la solución ideal para un entorno estable de desarrollo y producción. Con base en esta experiencia, se recomienda implementar un despliegue completo de Openstack en futuras iteraciones, tanto para desarrollo como para producción. Esta alternativa proporcionaría una mayor robustez y flexibilidad, mejorando la capacidad de manejo y ajuste de las instancias virtuales sin los inconvenientes experimentados. Un despliegue completo de Openstack también permitiría aprovechar un rango más amplio de funcionalidades avanzadas de gestión de infraestructura en la nube, adaptándose mejor a las necesidades de un entorno educativo en crecimiento y asegurando una mayor estabilidad y rendimiento para los usuarios finales.

8.7. Construir una plataforma para crear y gestionar instancias de máquinas virtuales en los servidores disponibles de la Universidad del Valle de Guatemala, con el fin de proporcionar a los estudiantes acceso a recursos tecnológicos avanzados y mejorar su formación práctica en ingeniería de software

Tras cumplir con los objetivos específicos planteados, es posible confirmar que el objetivo general del proyecto ha sido alcanzado exitosamente. Los entregables generados para cada objetivo contribuyen a la creación de una plataforma robusta que permite a los usuarios registrar cuentas, así como crear, editar y gestionar máquinas virtuales dentro de los servidores de la Universidad del Valle de Guatemala. La plataforma proporciona a los estudiantes acceso a recursos tecnológicos avanzados, consolidando así un entorno de aprendizaje práctico que apoya su formación en ingeniería de software. Esta plataforma se encuentra desplegada en el servidor designado al equipo de desarrollo, propiedad de la Universidad, y ha sido configurada para permitir tráfico hacia los puertos correspondientes, facilitando el acceso y uso de las máquinas virtuales creadas. Los usuarios pueden ejecutar sistemas, gestionar configuraciones y operar las instancias según sus necesidades de manera directa y eficiente. Además, la actualización de la plataforma en el servidor ha sido simplificada mediante la integración con un sistema de control de versiones, lo cual permite implementar mejoras o solucionar problemas mediante actualizaciones en la rama del repositorio del proyecto, garantizando un mantenimiento ágil y continuo. A través de pruebas de usuario, se ha validado que la plataforma

es funcional y permite a los estudiantes crear y modificar máquinas virtuales según sus necesidades, aunque los usuarios expresaron la necesidad de contar con descripciones adicionales en cuanto a los tipos de máquinas (flavors) y una guía de conexión mediante SSH. Estos aspectos serán esenciales para futuras mejoras orientadas a optimizar la experiencia de usuario. En conclusión, la plataforma cumple con el objetivo de brindar a los estudiantes un entorno práctico, accesible y funcional para gestionar máquinas virtuales, proporcionando una herramienta útil para su formación en un entorno controlado y avanzado.

CAPÍTULO 9

Conclusiones

1. La plataforma desarrollada cumple plenamente con el objetivo general de proporcionar a los estudiantes un recurso valioso para la creación y gestión de máquinas virtuales. Este sistema les otorga acceso a una infraestructura tecnológica avanzada en un entorno controlado y permite la realización de actividades prácticas que refuerzan su aprendizaje en el ámbito de la ingeniería de software. Además, la implementación en los servidores de la universidad optimiza el uso de recursos existentes, contribuyendo al desarrollo de competencias técnicas en los estudiantes mediante el uso de herramientas prácticas y de relevancia en la industria.

2. La API desarrollada permite la integración eficiente entre el frontend y el sistema de gestión de máquinas virtuales, captando correctamente las solicitudes de los usuarios y facilitando la automatización de procesos como la creación y modificación de instancias. Mediante el uso de comandos y scripts ejecutados en la línea de comandos, la API logra gestionar las instancias de forma autónoma y eficaz.

3. El proyecto incluye un conjunto de scripts de automatización que facilitan tanto la configuración inicial como el despliegue de máquinas virtuales. Estos scripts permiten realizar tareas críticas de manera automática, como el inicio de máquinas virtuales inactivas y la configuración de parámetros de red, garantizando así un funcionamiento continuo y eficiente del sistema. La integración de estos scripts en el API valida la eficiencia y utilidad de la automatización en la gestión de recursos, cumpliendo el objetivo específico y optimizando el flujo de trabajo para los usuarios.

4. La interfaz web desarrollada en React proporciona una experiencia de usuario intuitiva para tareas básicas como la creación de usuario, inicio de sesión y gestión de instancias, cumpliendo así con la mayoría de los requisitos de accesibilidad y usabilidad.

5. En la interfaz web Los flavors, botones de acciones y la conexión SSH podrían beneficiarse de mejoras adicionales, como iconografía y guías de usuario. Estos ajustes permitirán mejorar la experiencia global y facilitar el uso del sistema para usuarios con menos experiencia técnica.

6. La implementación de OpenStack mediante Microstack fue adecuada para la fase experimental de este proyecto como prueba de concepto.

7. La plataforma fue desplegada exitosamente en los servidores de la Universidad del Valle de Guatemala, optimizando el uso de los recursos tecnológicos existentes.

CAPÍTULO 10

Recomendaciones

1. Para mejorar la experiencia de usuario y reducir las barreras en tareas avanzadas, se recomienda desarrollar guías interactivas o un sistema de ayuda en línea que oriente a los usuarios, especialmente en la conexión SSH y la comprensión de los diferentes tipos de instancias disponibles.

2. Añadir descripciones más claras y elementos visuales específicos, como iconos para cada función principal, facilitaría la navegación y el acceso intuitivo a opciones clave, como creación, edición y gestión de instancias.

3. Desarrollar un sistema de monitoreo que permita a los usuarios recibir retroalimentación en tiempo real sobre el estado de sus instancias y conexión sería un aporte de valor. Esto podría incluir notificaciones sobre el estado de conexión, acceso SSH y alertas en caso de inactividad de las máquinas virtuales.

4. Para maximizar el potencial de la plataforma, se sugiere explorar su despliegue en entornos externos o en la nube, permitiendo que los estudiantes accedan a la plataforma de forma remota y adaptando el proyecto a una experiencia de gestión de infraestructura virtual escalable.

5. Para garantizar estabilidad, escalabilidad y acceso a funcionalidades avanzadas en futuros desarrollos y entornos de producción se recomienda considerar el reemplazo de Microstack por una herramienta más adecuada para entornos de servidor que permita un despliegue optimizado y escalable de Openstack.

6. Implementar otras features para máquinas virtuales y permitidas por Openstack como creación de volúmenes y guardado de snapshots

7. Implementar otras features de Openstack a la plataforma como orquestación de aplicaciones, automatización, redes como servicio, seguridad, etc.

Bibliografía

[1] Amazon Web Services. (n.d.-a). Modelos de servicio en la nube | Tipos de cloud computing. Recuperado de https://aws.amazon.com/es/types-of-cloud-computing/.

[2] Amazon Web Services. (n.d.-b). ¿Qué es el código abierto? – Explicación del código abierto. Recuperado de https://aws.amazon.com/es/what-is/open-source/.

[3] Amazon Web Services. (2024). Amazon Web Services Documentation. Recuperado de https://docs.aws.amazon.com/.

[4] Arranz, V. (2024). Los 8 lenguajes de programación que todo experto en ciberseguridad debe conocer. Recuperado de https://www.campusciberseguridad.com/blog/item/181-8-lenguajes-progrmacion-experto-ciberseguridad-debe-conocer.

[5] Calero, V. (2023). Qué es un script: Definición, significado y ejemplos. Recuperado de https://www.arimetrics.com/glosario-digital/script.

[6] Comms, M. (n.d.). ¿Cuánto tiempo debo dedicar a mis estudios en línea? Recuperado de https://blog.uvm.mx/cuanto-tiempo-debo-dedicar-a-mis-estudios-en-linea.

[7] De Souza, I. (2021). Entiende las diferencias entre Front-End y Back-end en el ambiente de los sitios web. Recuperado de https://rockcontent.com/es/blog/front-end-y-back-end/.

[8] De Zúñiga, F. G. (2024). ¿Qué es una máquina virtual y para qué sirve? Recuperado de https://www.arsys.es/blog/que-es-una-maquina-virtual-y-para-que-sirve.

[9] Google Cloud. (n.d.-a). Ventajas de la computación en la nube. Recuperado de https://cloud.google.com/learn/advantages-of-cloud-computing?hl=es-419.

[10] Google Cloud. (2023). ¿Qué es una instancia en Google Compute Engine? Recuperado de https://cloud.google.com/compute/docs/instances.

[11] Google Cloud. (2024-a). ¿Qué es un proveedor de servicios en la nube? Recuperado de https://cloud.google.com/learn/what-is-a-cloud-service-provider?hl=es.

[12] Google Cloud. (2024-b). ¿Qué son los contenedores? Recuperado de https://cloud.google.com/learn/what-are-containers?hl=es.

[13] Google Cloud. (2024-c). ¿Qué es una base de datos relacional (RDBMS)? Recuperado de https://cloud.google.com/learn/what-is-a-relational-database?hl=es-419.

[14] Hubspot. (2023-a). Qué es un diseño responsive: características y ejemplos. Recuperado de https://blog.hubspot.es/website/diseno-responsive.

[15] Hubspot. (2023-b). Pruebas de usabilidad: qué son, cómo hacerlas y ejemplos. Recuperado de https://blog.hubspot.es/website/pruebas-usabilidad.

[16] Hubspot. (2023-c). ¿Qué es React y para qué sirve? Recuperado de https://blog.hubspot.es/website/que-es-react.

[17] IBM. (n.d.-a). ¿Qué es una API (interfaz de programación de aplicaciones)? Recuperado de https://www.ibm.com/mx-es/topics/api.

[18] IBM. (n.d.-b). Version 8.4 and later, and SaaS. Recuperado de https://www.ibm.com/docs/es/mas-cd/maximo-manage/continuous-delivery?topic=ccsp-examples-using-automation-scripts-during-processing-by-channels-services.

[19] Intel. (n.d.). Descripción general de los modelos de implementación de la nube. Recuperado de https://www.intel.la/content/www/xl/es/cloud-computing/deployment-models.htm.

[20] Kingston Technology Company. (n.d.). ¿Cuánta memoria necesita para ejecutar aplicaciones de Windows, macOS o Linux? Recuperado de https://www.kingston.com/es/blog/pc-performance/memory-assessor.

[21] Singhania, S. (2024). Top 10 programming languages to build REST APIs in 2024. Recuperado de https://www.linkedin.com/pulse/top-10-programming-languages-build-rest-apis-2024-sanjay-singhania-igfcf/.

[22] Lucena, P. (n.d.). ¿Qué es el framework? Recuperado de https://www.cesuma.mx/blog/que-es-el-framework.html.

[23] MDN Web Docs. (n.d.). ¿Qué es JavaScript? — Aprende desarrollo web. Recuperado de https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/What_is_JavaScript.

[24] Microsoft Azure. (n.d.-a). Precios - Cloud Services. Recuperado de https://azure.microsoft.com/es-es/pricing/details/cloud-services/.

[25] Microsoft Azure. (n.d.-b). ¿Qué es una nube híbrida? Recuperado de https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-hybrid-cloud-computing.

[26] Miro. (n.d.). Wireframe: qué es, cómo hacerlo y ejemplos. Recuperado de https://miro.com/es/wireframe/que-es-wireframe/.

[27] NordVPN. (2024). Container isolation definition — Glossary. Recuperado de https://nordvpn.com/es-mx/cybersecurity/glossary/container-isolation/.

[28] Python Documentation. (n.d.). subprocess — Subprocess management. Recuperado de https://docs.python.org/3/library/subprocess.html.

[29] Red Hat. (n.d.-a). El concepto de OpenStack. Recuperado de https://www.redhat.com/es/topics/openstack.

[30] Red Hat. (n.d.-b). ¿Qué es Docker y cómo funciona? Ventajas de los contenedores Docker. Recuperado de https://www.redhat.com/es/topics/containers/what-is-docker.

[31] Singhania, S. (2024). Top 10 programming languages to build REST APIs in 2024. Recuperado de https://www.linkedin.com/pulse/top-10-programming-languages-build-rest-apis-2024-sanjay-singhania-igfcf/.

[32] State of JavaScript. (2023). Libraries. Recuperado de https://2023.stateofjs.com/en-US/libraries/.

[33] Stack Overflow. (2023-a). Stack Overflow Developer Survey 2023. Recuperado de `https://survey.stackoverflow.co/2023/#overview`.

[34] Stack Overflow. (2023-b). Stack Overflow Developer Survey 2023: Most popular technologies other tools. Recuperado de `https://survey.stackoverflow.co/2023/#section-most-popular-technologies-other-tools`.

[35] Souto, V. R. (n.d.). ¿Qué es CSS3? Recuperado de `https://www.hackaboss.com/blog/que-es-css`.

[36] Vadavo. (2023). HTML: qué es y para qué sirve. Recuperado de `https://www.vadavo.com/blog/html-que-es-y-para-que-sirve/`.