Universidad de los Andes	**Universidad de los Andes** **Maestría en Seguridad de la Información - MESI**	Seguridad en cloud Marzo 2025

Documentación del Proyecto

ESTUDIANTES:

Diana María Andica Bueno - 202413738

Johanna Alexandra Villamil Salinas - 202513858

Diego Felipe Sánchez Medina - 202322998

Profesor: Julián Jiménez / Jesse Padilla

Bogotá D.C

2025

TABLA DE CONTENIDO

Objetivos

- Diseñar, desplegar y configurar una aplicación web en Google Cloud Platform
- Aplicar buenas prácticas de arquitectura en la nube
- Comprender y configurar mecanismos de escalabilidad horizontal y escalado automático.

Resumen del proyecto

Nombre del proyecto: Blogueando ando

Descripción: La aplicación permite a los usuarios registrarse, crear y gestionar publicaciones. Podrán añadir y eliminar posts, etiquetarlos, explorar todas las publicaciones y etiquetas disponibles, filtrarlas por etiqueta, publicar contenido y calificar los posts de otros usuarios.

Arquitectura general: La aplicación sigue una arquitectura desplegada en Google Cloud Platform, la cual contiene los siguientes componentes:

- Frontend
- Backend
- Base de datos
- Autenticación con IAM
- Almacenamiento de archivos
- Monitorización y logs
- Uso de VPC
- Balanceadores
- Docker

Configuración de la infraestructura

A continuación, se presenta el diagrama que representa la arquitectura de despliegue de la aplicación BlogueandoAndo en Google Cloud Platform (GCP), detallando cómo los distintos servicios interactúan dentro de una red privada virtual (VPC) para garantizar seguridad, escalabilidad y rendimiento. Se utilizaron los servicios de GCP, como: Compute Engine, Cloud SQL, Cloud Storage, Identity and Access Management – IAM, Logging, Reglas de firewall, red de VPC, docker.

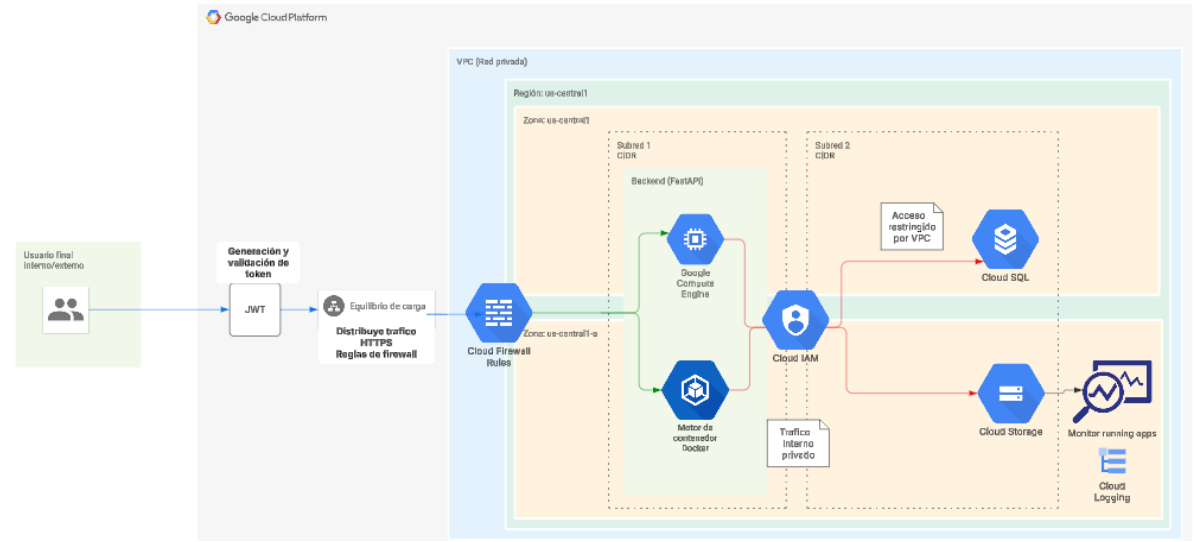


Figura 1. Diagrama de arquitectura

Usuarios y seguridad

- Los usuarios finales (internos o externos) acceden a la aplicación a través de un mecanismo de autenticación basado en JWT (JSON Web Token).
- Los tokens son generados y validados para gestionar el acceso seguro a los recursos protegidos.

Balanceo de Carga y Seguridad de Red

- Un balanceador de carga HTTPS distribuye el tráfico entrante entre las instancias del backend, asegurando disponibilidad y rendimiento.
- Las reglas de firewall controlan el acceso a la red, restringiendo conexiones no autorizadas y protegiendo los recursos internos.

Backend en Google Compute Engine

- El backend de la aplicación (FastAPI) se ejecuta en una instancia de Google Compute Engine (GCE).
- Se usa un motor de contenedores Docker para desplegar y gestionar la aplicación de manera eficiente.

Red Privada y Control de Acceso

- La infraestructura está dentro de una VPC (Red Privada Virtual) con subredes diferenciadas para el backend y los servicios de almacenamiento.
- Cloud IAM gestiona el acceso a los recursos de manera segura, asegurando que solo los servicios y usuarios autorizados puedan interactuar con la infraestructura.
- El tráfico interno dentro de la VPC es privado y protegido.

Servicios de Almacenamiento y Bases de Datos

- Cloud SQL almacena los datos de la aplicación y solo es accesible desde la VPC, garantizando seguridad.
- Cloud Storage se utiliza para almacenar archivos y contenido multimedia.
- El acceso a estos servicios está restringido mediante reglas de seguridad.

Monitoreo y Logs

- Cloud Logging y herramientas de monitoreo permiten supervisar el estado de las aplicaciones y detectar posibles fallos en tiempo real.

Configuración del frontend

El frontend de BlogueandoAndo está estructurado de manera organizada y modular, permitiendo un desarrollo y mantenimiento eficiente. Su arquitectura se divide en varias carpetas clave, como components, context, pages, services y styles, cada una con una función específica dentro del proyecto.

La carpeta components agrupa los elementos reutilizables de la interfaz. Dentro de esta, se encuentran los modales (PostModal.js, TagsModal.js, ConfirmDeleteModal.js, etc.), un componente de paginación (PaginationComponent.js) y el editor de texto enriquecido QuillEditor.js. Esta organización permite separar la lógica de la interfaz en componentes independientes, facilitando su reutilización en diferentes partes de la aplicación.

Por otro lado, en context se gestionan estados globales con AuthContext.js y ToastContext.js. AuthContext.js maneja la autenticación de usuarios, mientras que ToastContext.js se utiliza para mostrar una notificación al usuario del resultado de algunas acciones dentro de la aplicación.

La carpeta pages agrupa las vistas principales de la aplicación, como Home.js, Post.js, Login.js y Register.js. Estos archivos representan distintas secciones del frontend y están diseñados para interactuar con los servicios y contextos, asegurando que la aplicación recupere y administre datos de manera dinámica.

En la carpeta services, se encuentran postService.js, tagService.js y userService.js, archivos encargados de realizar las solicitudes a la API relacionadas con publicaciones, etiquetas y usuarios. Esta separación permite que la lógica de comunicación con el backend esté desacoplada de los componentes visuales, siguiendo una arquitectura más limpia y modular.

Configuración de la API

El backend de la aplicación se desarrolló como una API REST utilizando FastAPI en Python, encargándose de la autenticación, gestión de usuarios, publicaciones (posts) y etiquetas (tags).

Se encuentra desplegado en una instancia de Google Cloud Platform (GCP), donde la base de datos está alojada en Cloud SQL para garantizar fiabilidad y escalabilidad.

Configuración técnica

- Proveedor de Identidad: OAuth2
- Conector de Base de datos: SQLALchemy
- Autenticación: JWT (JSON Web Token)
- Seguridad: Comunicación segura mediante HTTPS entre el backend y el frontend.
- Restricciones: No se permite una calificación por fuera del rango menor a 1 o mayor a 5, es decir, la calificación menor que se da a un post es de 1 y la mayor calificación es de 5

Endpoints

A continuación, se detallan los servicios expuestos por la API:

✓ /register

Endpoint para registrar un nuevo usuario

- Método HTTP: POST
- Entradas:

```
{
  "email": "string",
  "password": "string",
  "name": "string"
}
```

- Salidas:
  - 200 – Registro exitoso
  - 400 - Ya existe una cuenta con este correo electrónico
  - 500 - Error al enviar el correo de confirmación

✓ /login

Para que un usuario ya registrado ingrese a la aplicación

- Método HTTP: POST
- Entradas:

```
{
  "email": "string",
  "password": "string"
}
```

- Salidas:
  - 200 – Ingreso exitoso
  - 401 – Credenciales incorrectas
  - 403 - Cuenta inactiva. Primero confirma el correo registrado

✓ /confirm_email
- Método HTTP: GET
- Entradas: Token
- Salidas:
  - 200 – Tu correo ha sido confirmado correctamente
  - 400 - El token no es válido
  - 404 - Usuario no encontrado

✓ /resend_email
- Método HTTP: GET
- Entradas: user_email
- Salidas:
  - 200 – Correo reenviado correctamente
  - 404 - Usuario no encontrado
  - 500 - Error al enviar el correo de confirmación

✓ /resend_email
- Método HTTP: GET
- Entradas: user_email
- Salidas:
  - 200 – Correo reenviado correctamente
  - 404 - Usuario no encontrado
  - 500 - Error al enviar el correo de confirmación

✓ /user
- Método HTTP: GET
- Entradas: current_user
- Salidas:
  - 200 – Usuario actual

- /password-reset-request
  - Método HTTP: POST
  - Entradas:

```
{
  "email": "string"
}
```

  - Salidas:
    - 200 – Te hemos enviado un enlace para recuperar tu contraseña
    - 404 - Usuario no encontrado

- /reset-password
  - Método HTTP: POST
  - Entradas:

```
{
  "token": "string",
  "new_password": "string"
}
```

  - Salidas:
    - 200 – Contraseña restablecida exitosamente
    - 400 - El token no es válido o ha expirado
    - 404 - Usuario no encontrado

- /post

  Un usuario logueado puede crear un post
  - Método HTTP: POST
  - Entradas:

```
{
  "title": "string",
  "content": "string",
  "is_public": true,
  "tags": [
    "string"
  ]
}
```

  - Salidas:
    - 201 – Respuesta exitosa

```
{
  "title": "string",
  "content": "string",
  "is_public": true,
  "tags": [
    "string"
  ],
  "id": 0,
  "publication_date": "string",
  "rating": 0,
  "user_name": "string"
}
```


- 401 - No estás autorizado para realizar esta acción

- /posts

  Un usuario logueado puede listar todos los posts
  - Método HTTP: GET
  - Entradas: size, skip, tags
  - Salidas:
    - 201 – Respuesta exitosa
    - 401 - No estás autorizado para realizar esta acción
- /post
  - Método HTTP: GET
  - Entradas: id
  - Salidas:
    - 200 – Respuesta exitosa


```
{
  "title": "string",
  "content": "string",
  "is_public": true,
  "tags": [
    "string"
  ],
  "id": 0,
  "publication_date": "string",
  "rating": 0,
  "user_name": "string"
}
```


    - 404 - No se encontró la publicación

✓ /post/{post_id}

Un usuario logueado puede actualizar el post
- Método HTTP: PUT
- Entradas:

```
{
  "title": "string",
  "content": "string",
  "is_public": true,
  "tags": [
    "string"
  ]
}
```

- Salidas:
  - 200 – Respuesta exitosa

```
{
  "title": "string",
  "content": "string",
  "is_public": true,
  "tags": [
    "string"
  ],
  "id": 0,
  "publication_date": "string",
  "rating": 0,
  "user_name": "string"
}
```

  - 401 - No estás autorizado para realizar esta acción
  - 403 - No tienes permiso para editar esta publicación
  - 404 - No se encontró la publicación
  - 500 - Error al actualizar el contenido en almacenamiento

✓ /post/{post_id}

Un usuario logueado puede eliminar un post

- Método HTTP: DELETE
- Entradas: id
- Salidas:
  - 200 – Publicación eliminada con éxito
  - 401 - No estás autorizado para realizar esta acción
  - 403 - No tienes permiso para eliminar esta publicación
  - 404 - No se encontró la publicación

✓ /set_rating

Asignarle calificación a un post

- Método HTTP: POST
- Entradas:

```
{
  "post_id": 0,
  "rating": 0
}
```

- Salidas:
  - 200 – Respuesta exitosa

```
{
    "title": "string",
    "content": "string",
    "is_public": true,
    "tags": [
      "string"
    ],
    "id": 0,
    "publication_date": "string",
    "rating": 0,
    "user_name": "string"
  }
```

  - 401 - No estás autorizado para realizar esta acción
  - 403 - No puedes calificar tu propia publicación
  - 404 - No se encontró la publicación
  - 406 - El valor de la calificación está fuera del rango permitido
  - 409 - Ya has calificado esta publicación

✓ /set_tags

Asignarle tag a un post

- Método HTTP: POST
- Entradas:

```
{
  "post_id": 0,
  "tags": [
    "string"
  ]
}
```

- Salidas:
  - 200 – Respuesta exitosa

```
{
      "title": "string",
      "content": "string",
      "is_public": true,
      "tags": [
        "string"
      ],
      "id": 0,
      "publication_date": "string",
      "rating": 0,
      "user_name": "string"
    }
```


- 401 - Debe estar autenticado para agregar tags

✓ /get_posts_tag

Obtener lista de los posts que contienen el tag

- Método HTTP: GET
- Entradas:


```
{
  "tag": "string"
}
```


- Salidas:
  - 200 – Respuesta exitosa


```
[
      {
        "title": "string",
        "content": "string",
        "is_public": true,
        "tags": [
          "string"
        ],
        "id": 0,
        "publication_date": "string",
        "rating": 0,
        "user_name": "string"
      }
    ]
```


✓ /upload

Para subir el contenido del post en formato .html a Cloud storage

- Método HTTP: POST
- Entradas:


```
{
  "filename": "string",
  "content": "string"
}
```


- Salidas:

- 200 – Respuesta exitosa
- 500 - No se pudo subir el post. Inténtalo más tarde

✓ /upload

Para descargar el contenido del post desde Cloud storage

- Método HTTP: GET
- Entradas: filename
- Salidas:
  - 200 – Respuesta exitosa

✓ /tags

Obtener todos los tags

- Método HTTP: GET
- Entradas: size, skip
- Salidas:
  - 200 – Respuesta exitosa
  - 401 - No estás autorizado para realizar esta acción

✓ /filter_tag

Obtener el tag que deseo encontrar filtrando con una parte de la palabra o con la palabra completa

- Método HTTP: GET
- Entradas:

```
{
  "tag": "string"
}
```

- Salidas:
  - 200 – Respuesta exitosa

```
[
    {
      "tag": "string",
      "id": 0
    }
]
```

Esquema de base de datos

Se utilizó un esquema de base de datos relacional para efectos del proyecto, sin embargo, en un entorno real creemos que se debe cambiar a una base de datos No relacional porque en la base de datos se presentaría una relación de muchos a muchos como es el caso de un post puede tener asociado más de una etiqueta y si se filtra por una etiqueta esta puede estar contenida dentro de muchos posts.

Para manejar este caso en nuestro proyecto, se creó la tabla “posts_tags”, donde se relaciona el post y sus etiquetas.

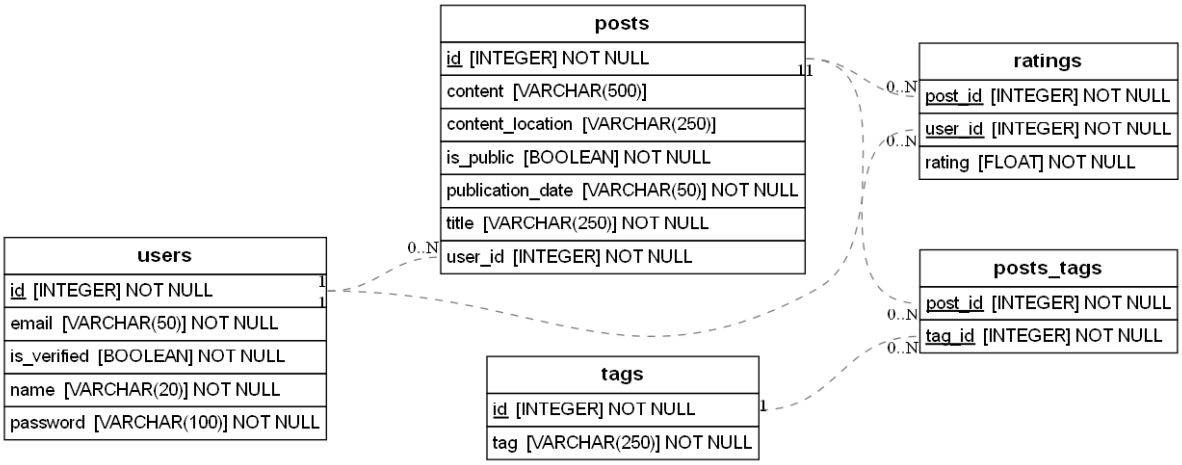


Figura 2. Diagrama Entidad Relación de la base de datos

La Base de Datos utilizada fue MySQL, la cual se llamó “posts” y contiene las siguientes tablas:

- ✓ Tabla “users”
  - id: Integer, primary_key, unique
  - name: VARCHAR(20), Not null
  - Email: VARCHAR(50), unique, Not null
  - Password: VARCHAR(100), Not null
  - is_verified:Boolean, Not null

- ✓ Tabla “posts”
  - Id: Integer, primary_key
  - user_id: ForeignKey("users.id"), Not null
  - Title: VARCHAR(250), Not null
  - Content: VARCHAR(250)
  - publication_date: VARCHAR(20), Not null
  - is_public: Boolean, Not null

- ✓ Tabla “ratings”
  - user_id: ForeignKey("users.id"), primary_key, Not null
  - post_id: ForeignKey("posts.id"), primary_key, Not null
  - rating = Float, Not null

- ✓ Tabla “posts_tags”
  - tag_id: ForeignKey("tags.id"), primary_key, Not null
  - post_id: ForeignKey("posts.id"), primary_key, Not null

- ✓ Tabla “tags”
  - Id: Integer, primary_key
  - Tag: VARCHAR(250), Not null

Decisiones de diseño

Frontend

Stack Tecnológico

- React: Permite una interfaz dinámica y rápida con actualización eficiente de los componentes.
- React Router: Manejo de rutas para navegación sin recargar la página.
- React Bootstrap: Uso de componentes preconstruidos para mejorar la interfaz y experiencia del usuario.

Estructura

La aplicación está organizada en carpetas que agrupan funcionalidades similares:

- components/: Contiene los componentes reutilizables, como PostModal.js, QuillEditor.js y PaginationComponent.js.
- pages/: Cada página principal (ej. Home.js, Post.js) está en esta carpeta, facilitando la navegación y modularidad.
- services/: Contiene servicios como postService.js y userService.js que manejan las llamadas a la API, separando la lógica de negocio del UI.
- styles/: Archivos CSS para el diseño y personalización de la interfaz.

Seguridad

- Sanitización de contenido: Uso de DOMPurify.sanitize() para evitar XSS cuando se renderizan posts creados por los usuarios.
- Manejo de autenticación: Uso de tokens JWT para proteger rutas y controlar el acceso a funcionalidades sensibles.
- HTTPS: La aplicación está configurada para operar bajo HTTPS, usando certificados SSL (inicialmente auto-firmados).

Backend

Stack Tecnológico

- FastAPI: Framework rápido y eficiente para construir la API, con validación de datos automática usando Pydantic.
- SQLAlchemy: ORM para gestionar la base de datos relacional de manera segura.
- JWT para autenticación: Implementación de autenticación basada en tokens para proteger los endpoints.

Estructura

El backend está organizado en módulos claros:

- models/: Define las estructuras de datos (post.py, user.py, tag.py).
- routers/: Define los endpoints (post.py, user.py, tag.py).
- security.py: Maneja la autenticación y validación de usuarios.
- database.py: Configura la conexión a la base de datos.

Seguridad

- Validación de datos con Pydantic: Se verifica que los datos recibidos cumplan con las estructuras definidas.
- Consultas parametrizadas con SQLAlchemy: Previene inyecciones SQL.
- Control de acceso: Solo los autores pueden editar o eliminar sus posts.
- CORS configurado: Para evitar ataques de origen cruzado (Cross-Origin Resource Sharing).

Decisiones de seguridad

En cuanto a seguridad, se tomaron las siguientes decisiones:

- Implementación de JSON Web Token – JWT, para realizar la autenticación y autorización de usuarios basados en tokens, donde se definió que el token expire en 30 minutos.

- Implementación de reglas de firewall entre la Aplicación web y las instancias

- Balanceadores: mitigan ataques de denegación de servicio distribuido (DDoS), bloquear tráfico malicioso, los usuarios finales no podrán acceder directamente a los servidores de backend, solo los usuarios autenticados pueden acceder a las aplicaciones detrás del balanceador

- Monitoreo: Es esencial para detectar, responder y prevenir amenazas en la infraestructura como anomalías en CPU y RAM por un DDoS, picos inusuales por un intento de extracción de datos protegiendo datos y respondiendo de forma rápida a los incidentes.

- Redes virtuales VPC: Controla tráfico de red y aísla recursos

- Contenedores: aislamiento del sistema operativo y de otros contenedores, reduciendo vulnerabilidades potenciales.

- Como medida de seguridad contra ataque de Cross-Site Scripting (XSS), se utilizó la función DOMPurify.sanitize(post.content) que procesa el contenido del post eliminando cualquier código malicioso antes de inyectarlo en la interfaz, asegurando que solo se rendericen elementos seguros.

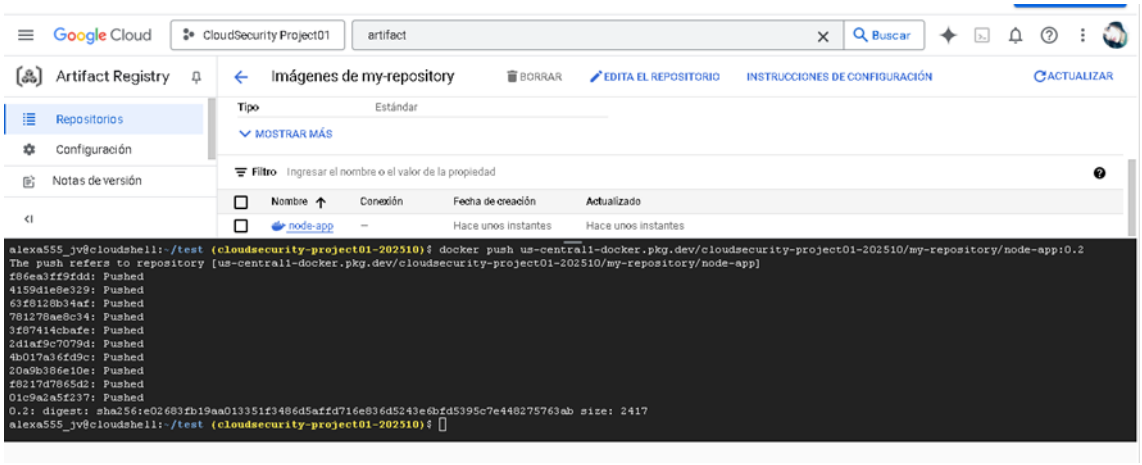
- Implementación de Autenticación Autorización utilizando IAM

Instrucciones de despliegue

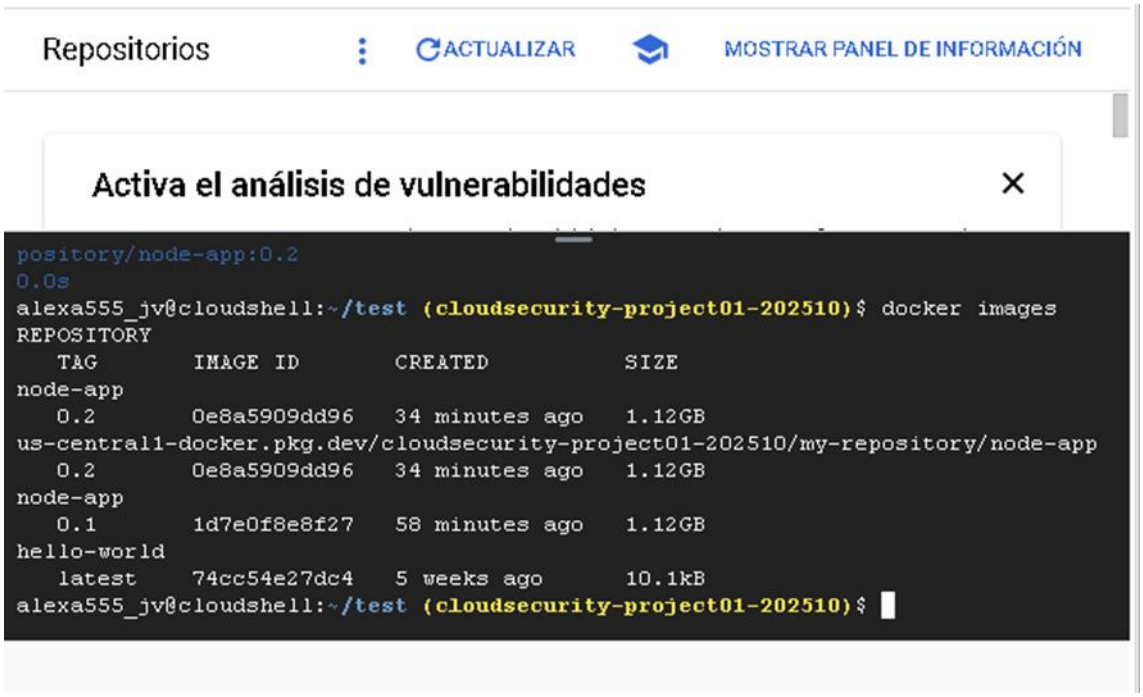
1. Configuración inicial en GCP
2. Configuración del backend
3. Configuración de bases de datos.
4. Contenerización del backend con Docker

Se deberá conectar a la VM por medio de SSH, luego se deberá buscar la carpeta del proyecto donde se encuentra la raíz del backend, posteriormente se crea el archivo Docker file en la raíz el backend

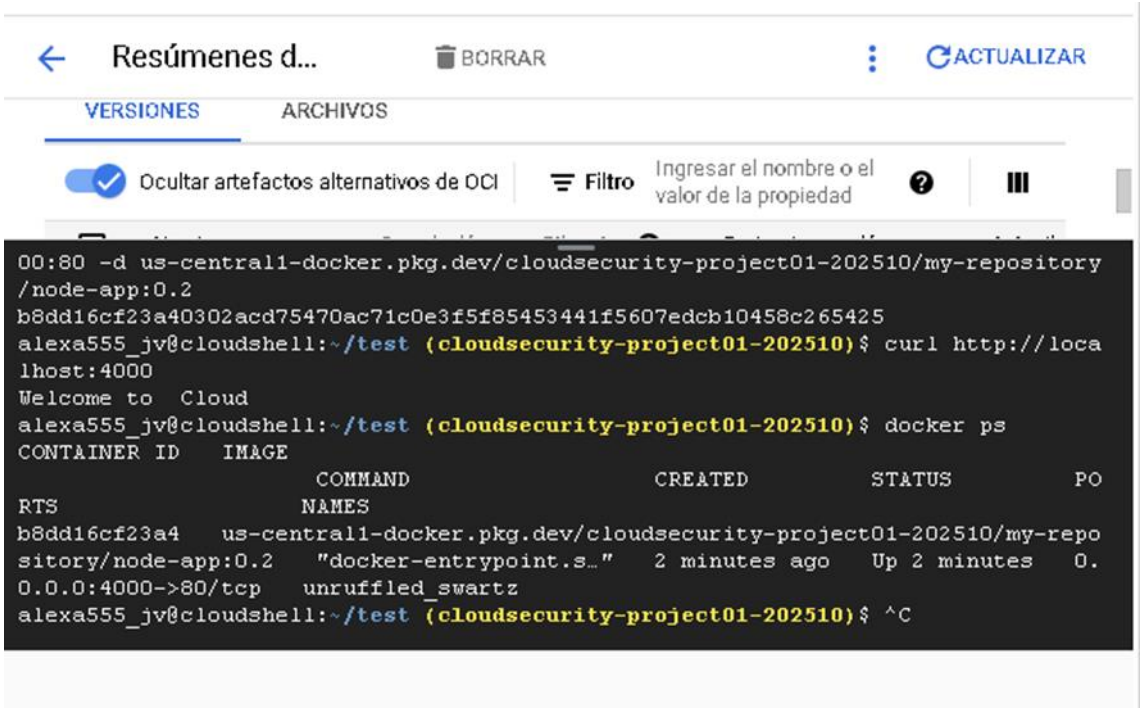
Luego de esto, se crea el contenedor docker



Se crea la imagen Docker dentro del proyecto, obteniendo el siguiente resultado:



Se ejecuta el comando Docker ps para observar los contenedores que están en ejecución:



Luego, se realiza una compilación, se crea un Docker file para indicar al Daemon de Docker como compilar la imagen, se sube la aimagen a GCR y se despliega la aplicación en GCP.

5. Subida y despliegue en cloud run
6. Configuración del almacenamiento en cloud storage
7. Despliegue del frontend
8. Monitoreo y escalabilidad
9. Pruebas y uso de la aplicación