

Documentación Proyecto 1

Pablo Ortega Cadavid - 202021700

Santiago Andrés Gélvez Galvis - 202212387

1. Configuración de la Infraestructura

El sistema está compuesto por los siguientes componentes:

- **Frontend:**
Implementado con React y desplegado en un contenedor Docker que escucha en el puerto **5173**. Los usuarios acceden a la IP pública del frontend desde cualquier ubicación.
- **Backend:**
Una API desarrollada con FastAPI que se ejecuta en contenedores Docker en instancias de VM. Cada contenedor expone el puerto 8000.
- **Balanceador de Carga:**
Un balanceador se encarga de distribuir las solicitudes entre dos (o más) instancias del backend. Este componente garantiza que la carga se distribuya de forma eficiente y oculta las IPs privadas de las instancias. Además de permitir escalabilidad horizontal en caso de ser necesario.
- **Base de Datos:**
Una instancia de CloudSQL con PostgreSQL que se comunica únicamente a través de IPs privadas con el backend, para realizar las operaciones CRUD.

Todos los componentes se encuentran dentro de una VPC, lo que asegura que la comunicación interna se realice de forma privada y segura.

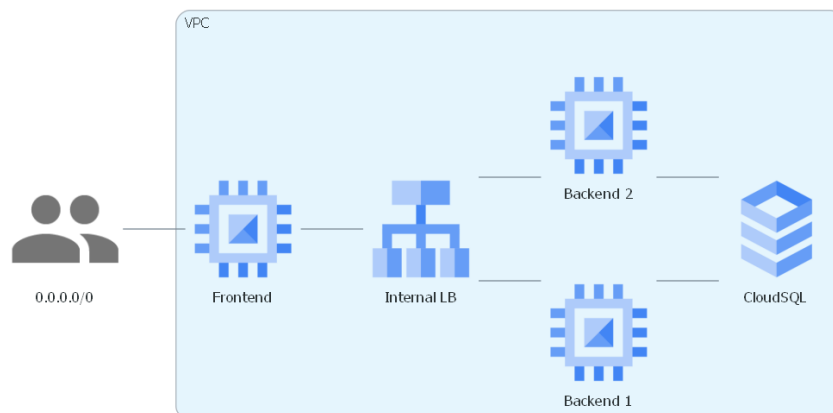


Diagrama integral de la infraestructura y flujo de comunicación entre los componentes.

2. Configuración de la API

La API, desarrollada con FastAPI, gestiona usuarios, posts, calificaciones y etiquetas. A continuación, se describen los endpoints principales:

a) Autenticación

POST /auth/login

Función: Permite a un usuario iniciar sesión y obtener un token JWT.

Solicitud (JSON): { "email": "usuario@example.com", "password": "password123" }

Respuesta exitosa (200): { "access_token": "eyJhbGciOiJI...", "token_type": "bearer" }

Errores: 401 Unauthorized si las credenciales son incorrectas.

POST /auth/register

Función: Registra un nuevo usuario.

Solicitud (JSON): { "name": "Usuario Ejemplo", "email": "usuario@example.com", "password": "password123" }

Respuesta exitosa (201): Usuario creado exitosamente.

b) Gestión de Posts

GET /posts

Función: Recupera la lista de posts existentes.

Autenticación: No requerida.

Respuesta (200):

[{ "id": 1, "title": "Primer Post", "content": "Contenido del post", "author": "usuario@example.com" }]

POST /posts

Función: Crea un nuevo post.

Autenticación: Requerida (el token JWT se debe enviar en el encabezado "Authorization" como "Bearer <token>").

Solicitud (JSON): { "title": "Nuevo Post", "content": "Contenido del post" }

Respuesta exitosa (201): Se confirma la creación del post.

Errores: 401 Unauthorized si el token es inválido o no se proporciona.

Otros endpoints (por ejemplo, para calificaciones y etiquetas) seguirán una lógica similar: se describe la acción, se indica si se requiere autenticación, y se muestran ejemplos de solicitud y respuesta para casos de éxito y error.

3. Esquema de la Base de Datos

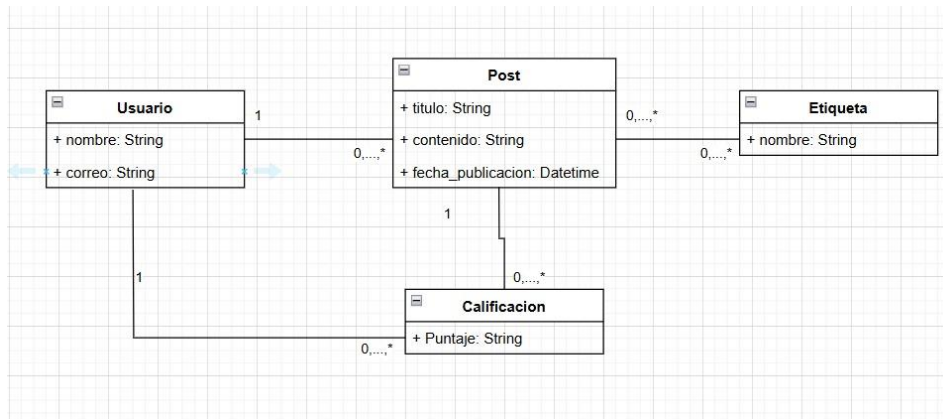
El modelo de datos se organiza de la siguiente forma:

- **Usuarios:**
Cada usuario tiene nombre, correo y contraseña. Un usuario puede crear posts y calificaciones.
- **Posts:**
Cada post tiene un autor (usuario) y puede tener uno o más tags asignados.
- **Calificaciones:**
Se crean por usuarios y se refieren a un post específico.
- **Tags (Etiquetas):**
Pueden asignarse a posts y no están asociados a un usuario específico.



```
ssh.cloud.google.com/v2/ssh/projects/blogapp-451901/zones/us-east1-d/instances/blogapp-backend1?authuser=0&hl=es_419&projectN...
ssh.cloud.google.com/v2/ssh/projects/blogapp-451901/zones/us-east1-d/instances/blogapp-backend1?authuser=0&hl=es_419&...
SSH en el navegador SUBIR ARCHIVO DESCARGAR ARCHIVO
GNU nano 5.4 Dockerfile
FROM python:3.10
WORKDIR /app
COPY requirements.txt .
RUN pip install --upgrade pip
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 8000
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Se incluye el Dockerfile utilizado para crear la imagen de uno de los contenedores del backend. En él se expone el puerto 8000 y se inicia la aplicación mediante el servidor *uvicorn*.



Esquema del modelo de datos (estructura y relaciones entre entidades).

4. Instrucciones de Despliegue y Uso

a) Despliegue del Backend (FastAPI)

1. Clonar el repositorio:
`git clone <repositorio>`
`cd blogApp-main`
2. Instalar las dependencias:
`pip install -r requirements.txt`
3. Ejecutar la API:
`uvicorn app.main:app --host 0.0.0.0 --port 8000`
Se recomienda ejecutar la API dentro de un contenedor Docker para facilitar el despliegue y la escalabilidad.

b) Despliegue del Frontend (React con Vite)

1. Clonar el repositorio:
`git clone <repositorio>`
`cd bloggAppFront-main/blog-app-front`
2. Instalar las dependencias:
`npm install`
3. Iniciar la aplicación:
`npm run dev`
La aplicación se ejecutará en el puerto 5173.

5. Decisiones de Diseño y Seguridad

- **Infraestructura Segura:**

El único nodo accesible desde Internet es el frontend, mientras que los demás componentes (balanceador, backend, base de datos) se comunican de forma privada dentro de la VPC.

- **Reglas de Firewall:**

- Acceso público restringido únicamente al puerto 3000 del frontend.
- Comunicación entre el balanceador y el backend únicamente por el puerto 8000.
- El backend se comunica con la base de datos exclusivamente a través del puerto 5432.

- **IAM en CloudSQL:**



Se configuraron roles y políticas de IAM para garantizar que solo las instancias autorizadas del backend puedan acceder a la base de datos.

- **Escalabilidad:**

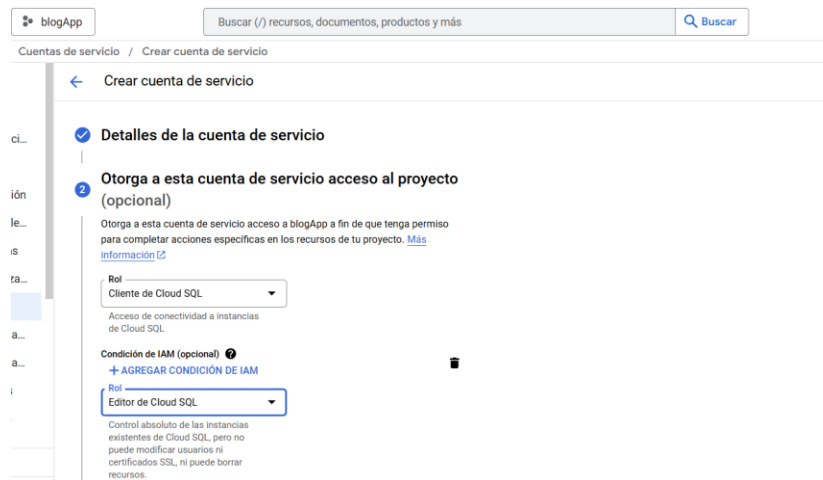
La implementación del balanceador de carga permite la escalabilidad horizontal. En caso de aumento de demanda, basta con agregar nuevas instancias al grupo de backend y desplegar el contenedor correspondiente en la nueva VM.

Adicionalmente con los balanceadores de carga es posible realizar health checks en las instancias para monitorear su funcionamiento.

- **Cuenta de servicio para el backend:** Se creó una cuenta de servicio específica (IAM) para que las instancias del backend accedan a CloudSQL con los permisos mínimos necesarios.

<input type="checkbox"/>	Correo electrónico	Estado	Nombre ↑	Descripción
<input type="checkbox"/>	 blogapp-backend-sa@blogapp-451901.iam.gserviceaccount.com	 Habilitado	blogapp-backend-sa	Acceder a cloudSQL a través de cuentas de servicio

Correo de la cuenta de servicio.



Permisos de la cuenta de servicio.

- **Principio de privilegio mínimo:** Se aplican roles con los permisos estrictamente necesarios para cada función dentro del sistema. Como es el caso de las reglas de firewall.
-

IAM en CloudSQL y Cuentas de Servicio

Para reforzar la seguridad del acceso a la base de datos, se implementaron cuentas de servicio con permisos restringidos:

- **Cuenta de servicio para el backend:** Se creó una cuenta de servicio específica para que las instancias del backend accedan a CloudSQL con los permisos mínimos necesarios.
- **Accesos reducidos para administradores:** Los administradores de infraestructura cuentan con permisos limitados solo para tareas de gestión y monitoreo, evitando cambios no autorizados.
- **Principio de privilegio mínimo:** Se aplican roles con los permisos estrictamente necesarios para cada función dentro del sistema.
- **Auditoría y rotación de credenciales:** Se estableció una revisión periódica de accesos y credenciales para garantizar la seguridad y minimizar riesgos.

6. Video

<https://youtu.be/gtwrYzI8PWU>