DOCUMENTACIÓN DEL PROCESO DE DESPLIEGUE DE LA PÁGINA WEB DE GRID Y ERRORES

Proyecto: Microservicios en GCP (Cloud Run + MySQL)

1. Contexto general

- Proyecto desarrollado por un equipo del semestre anterior.
- La arquitectura incluye: Microservicios con FastAPI, Base de datos MySQL en una VM, Autenticación con JWT, variables de entorno (.env).
- Se intenta desplegar con GPC por recomendación del equipo anterior.

2. Configuración inicial

Base de datos MySQL

- Se levantó una VM en GCP con MySQL.
- Se creó un usuario con constraseña.
- Se creó la base de datos GRID.
- En .env se configuró:

```
MYSQL_USER=dev
MYSQL_PASSWORD=grid2025
MYSQL_HOST=107.178.217.239
MYSQL_PORT=3306
MYSQL_DB=GRID
```

3. Pasos para el despliegue

Para desplegar cada microservicio se deben seguir los siguientes pasos:

1. **Iniciar sesión en GCP**

```
gcloud auth login
```

2. **Configurar el proyecto**

```
gcloud config set project PROJECT_ID
```

3. **Habilitar servicios necesarios**

```
gcloud services enable
[run.googleapis.com](http://run.googleapis.com/)
[containerregistry.googleapis.com](http://containerregistry.
googleapis.com/)
```

4. **Construir la imagen de Docker**

```
docker build -t
[gcr.io/PROJECT_ID/YOUR_IMAGE_NAME](http://gcr.io/PROJECT_ID
/YOUR_IMAGE_NAME).
```

5. **Autenticarse en Docker con GCP**

```
gcloud auth configure-docker
```

6. **Subir la imagen a Google Container Registry**

```
docker push
[gcr.io/PROJECT_ID/YOUR_IMAGE_NAME](http://gcr.io/PROJECT_ID
/YOUR_IMAGE_NAME)
```

7. **Desplegar en Google Cloud Run**

```
gcloud run deploy YOUR_SERVICE_NAME --image
gcr.io/PROJECT_ID/YOUR_IMAGE_NAME --platform managed --
region REGION --allow-unauthenticated
```

4. Microservicio: Autenticación

Este microservicio se encarga de la gestión de autenticación y autorización para el sistema.

Problema inicial

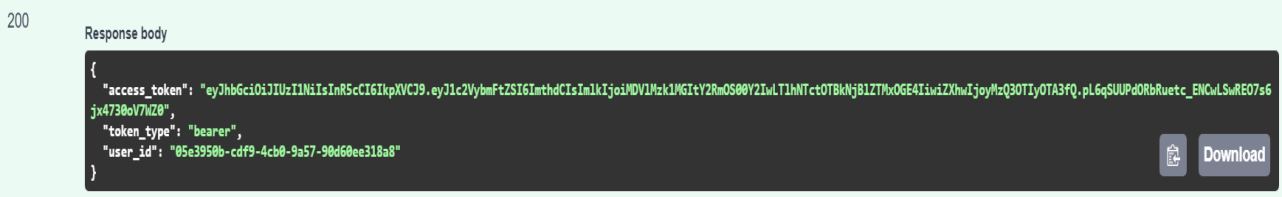
- El microservicio desplegaba correctamente, pero las tablas del modelo User no se creaban en la base de datos de la VM, lo cual generaba errores cuando se intentaban realizar consultas o logins.

Solución aplicada

- Se modificó el archivo database.py añadiendo manualmente:
- from app.models import User
- SQLAlchemy reconoció el modelo y lo incluyó en la creación de las tablas.
- A partir de esa corrección, el microservicio de autenticación funcionó correctamente.

Resultado:

- Endpoint /login genera tokens JWT válidos. Ejemplo de token:



Estado final

- El servicio fue desplegado exitosamente.

5. **Microservicio: Battery Service**

Descripción general

- Este microservicio fue diseñado para:
- Leer archivos de un .csv, procesar la información y generar un gráfico.
- No realiza ninguna conexión con la base de datos.
- No contaba con archivos esenciales para su despliegue en contenedores:
    - **Dockerfile**: Se tuvo que crear manualmente.
    - **requirements.txt**: También fue generado posteriormente, estimando las dependencias del código.

Configuración crítica para Cloud Run:

Variables de entorno:

--host 0.0.0.0: Permite que la aplicación escuche solicitudes desde cualquier IP externa (requerido en entornos cloud).

--port ${PORT:-8080}: Usa la variable PORT proporcionada por Cloud Run, con un valor por defecto de 8080 si no está definida.

Referencia al módulo correcto:

battery_api:app: Asegura que uvicorn ejecute la instancia app definida en battery_api.py. Este fue el ajuste clave para resolver un error donde el contenedor no estaba escuchando en el puerto definido por la variable PORT.

Estado final

- El servicio fue desplegado exitosamente.

Recomendaciones

No se justifica que esta funcionalidad sea un microservicio independiente:

- Su lógica es muy simple.
- No involucra persistencia de datos ni procesamiento complejo.
- Se podría integrar como un endpoint en otro microservicio más robusto (por ejemplo, el de consumo energético o el predictivo).

- El mantenimiento y despliegue de este microservicio implica más trabajo del que realmente aporta en funcionalidad.

6. Microservicio: Market Blockchain Microservices

Descripción general

Este microservicio está incluido en el repositorio del proyecto bajo el nombre market-blockchain-microservices. Sin embargo, no se tuvo en cuenta durante el proceso de despliegue, debido a su complejidad y falta de documentación clara por parte del equipo de desarrollo. Tras consultar con el cliente, se acordó no incluirlo y que esta parte no sería crítica para los objetivos inmediatos del proyecto.

Recomendación

Si en el futuro se desea incorporar esta funcionalidad, será necesario:

- Solicitar al equipo desarrollador la documentación completa.
- Validar su utilidad dentro del sistema.
- Integrarlo con una arquitectura basada en microservicios correctamente configurada.

7. Microservicio: Energy Consumption Service

Descripción general

Este microservicio es responsable de gestionar el registro y consulta de datos de consumo y generación energética asociados a dispositivos de usuario. Está construido con FastAPI, y depende de:

- Una base de datos MySQL donde se almacenan entradas de consumo, generación, dispositivos, y relaciones con usuarios.
- Un microservicio externo de autenticación, al cual consulta para validar tokens JWT recibidos en las solicitudes.

Errores encontrados

Error de autenticación con JWT – /devices/get-user-devices

Tipo de error: HTTP 500 (Internal Server Error)

Mensaje del log: jose.exceptions.JWTError: Not enough segments

Descripción: Este error indica que el microservicio intentó decodificar un JWT mal formado. Un token JWT debe tener tres partes separadas por puntos (.): **header.payload.signature**

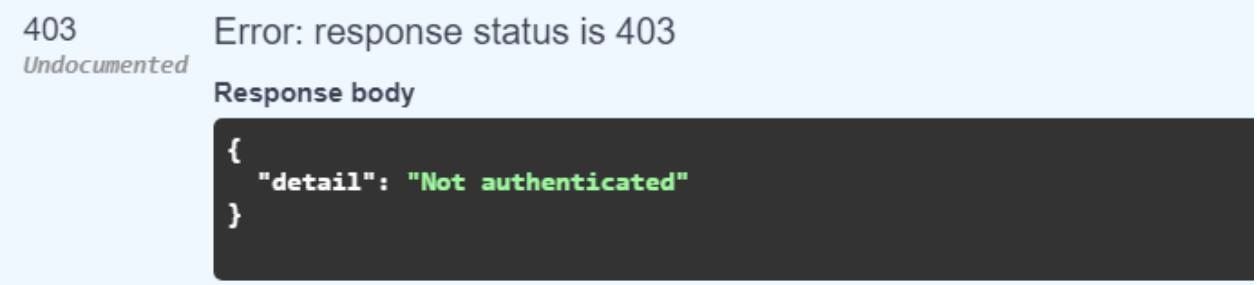
Solución aplicada: Se reemplazó el token inválido por uno válido y correctamente formado. Ejemplo del token funcional utilizado:

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

Resultado: El error 500 desapareció y el microservicio dejó de lanzar excepciones relacionadas con JWT malformados.

Error 403 al consumir endpoints de consumo

Al hacer un **GET** a **/api/v1/devices/get-user-devices/** se obtiene:



Verificaciones realizadas:

- Se confirmó que el token enviado es válido.
- El encabezado Authorization: Bearer <JWT> fue incluido correctamente.
- .env tiene la URL correcta del microservicio de autenticación.
- Los microservicios pueden comunicarse entre sí (sin error de red ni firewall).

A pesar de estas confirmaciones, el endpoint sigue respondiendo con 403 y mensaje “Not authenticated”.

Estado Final

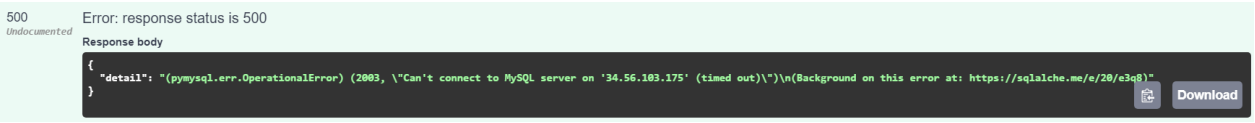
Microservicio parcialmente funcional:

- Se crearon tablas.
- Al menos un POST a /register-entry/ funcionó (200 OK).
- Endpoints como /get-user-devices/ devuelven 403 a pesar de un token válido.
- Problema persiste sin explicación clara en lógica actual.

8. Microservicio: Predictive Service

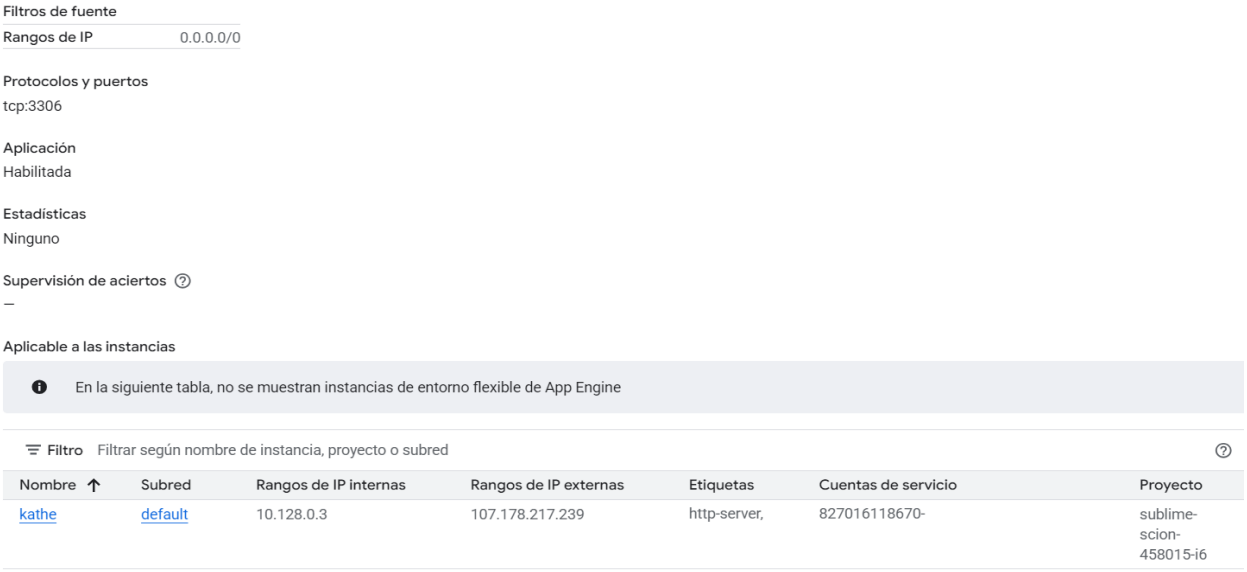
El Predictive Service es un microservicio diseñado para realizar predicciones de consumo energético basadas en datos almacenados en una base de datos. Durante el despliegue en Google Cloud Run, se encontró un error que impedía la conexión con la base de datos alojada en una VM de GCP. A continuación, se detallan las causas investigadas y las acciones realizadas.

Código de Error: 500 (Internal Server Error)



1. Reglas de Firewall bloqueadas

- Se verificaron las reglas de firewall en GCP: **gcloud compute firewall-rules list --filter="NETWORK=default".**
- Se confirmó que existía una regla permitiendo tráfico en el puerto 3306 desde 0.0.0.0/0.

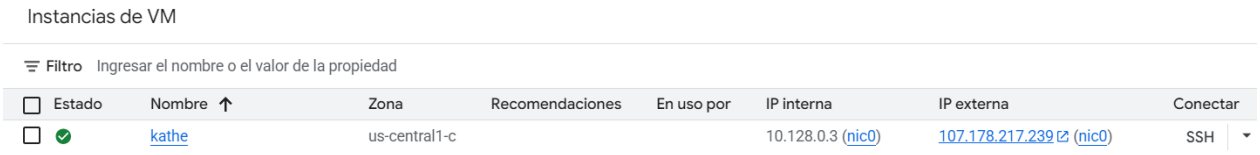


Las reglas de firewall no eran la causa del error.

2. Configuración de red en GCP

La VM podría carecer de una IP pública estática o tener políticas de red restrictivas.

- Se verificó la IP externa de la VM.

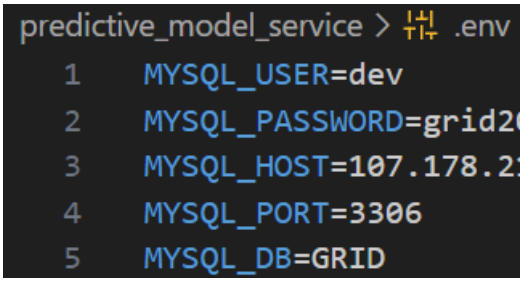


Resultado: La configuración de red era correcta.

3. Credenciales o variables de entorno

Posibles credenciales incorrectas en el archivo .env del servicio.

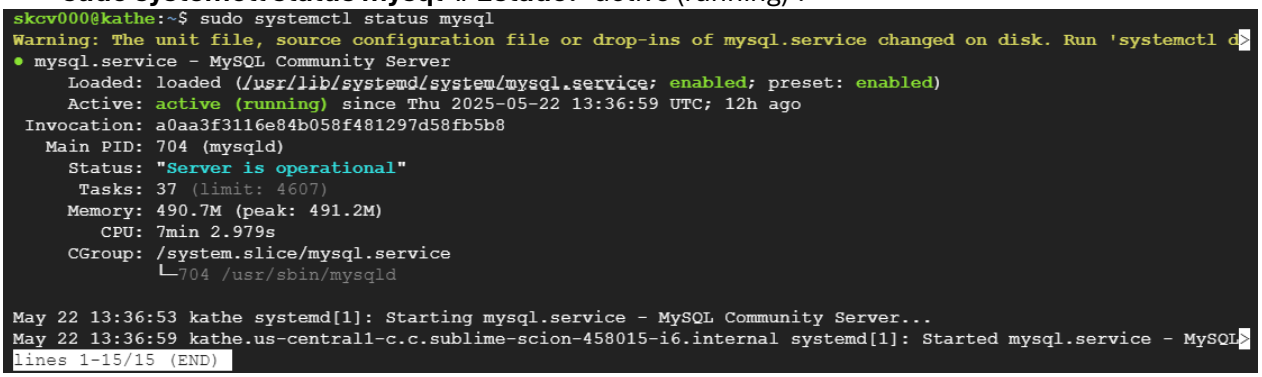
- Se verificaron las variables.



4. Estado del servicio MySQL

El servicio MySQL podría estar detenido en la VM.

- **sudo systemctl status mysql # Estado:** "active (running)".



- **sudo tail -f /var/log/mysql/error.log #** Sin errores críticos. MySQL estaba en ejecución y sin errores en los logs.

Estado Actual del Predictive Service

No desplegado: A pesar de descartar todas las causas comunes, el error de conexión a MySQL persistió y no pudo resolverse dentro del plazo del proyecto.

9. Frontend - Comunidades Energéticas

Descripción general

- Proyecto de frontend basado en Vite + React + TailwindCSS.
- No incluye requirements.txt ni Dockerfile porque no es un microservicio en Python, sino un frontend construido con Node.js.

Situación Actual:

- **Despliegue pospuesto**: El frontend no fue desplegado durante este ciclo. Las principales razones fueron:
    1. **Priorización del backend**: Los esfuerzos se enfocaron en resolver errores críticos en los microservicios esenciales, los cuales eran prerrequisitos para garantizar la funcionalidad integral del sistema.
    2. **Dependencias no resueltas**: La falta de estabilidad en los microservicios complejos hacía inviable validar una integración frontend-backend funcional.

Recomendaciones para futuros despliegues

Si se desea mantener una arquitectura homogénea usando contenedores se debe crear el dockerfile que permita empaquetar este frontend como imagen y desplegarlo en Cloud Run, al igual que el resto de los microservicios.
Para la integración con el backend, configurar variables de entorno en el frontend para apuntar a las URLs de los microservicios desplegados.

- Documentar el proceso de frontend: Incluir un DEPLOY_FRONTEND.md con los pasos específicos.

- Automatizar la integración: Usar GitHub Actions para construir y desplegar el frontend automáticamente al actualizar el repositorio.
- Validar end-to-end: Una vez estabilizado el backend, realizar pruebas de integración completa para garantizar que el frontend consume correctamente las APIs.

10. Conclusiones y recomendaciones finales

El despliegue de la página web de GRID en Google Cloud Platform (GCP) ha sido una experiencia que permitió, tanto poner en funcionamiento varios componentes del sistema, como comprender profundamente los desafíos reales de trabajar con una arquitectura basada en microservicios. Resultando en seis conclusiones principales, las cuales son:

1. **El valor de una arquitectura modular:** La separación de responsabilidades por microservicio (autenticación, consumo energético, procesamiento de datos, etc.) favorece la escalabilidad y el mantenimiento, siempre y cuando cada componente esté debidamente documentado y configurado.
2. **Importancia de la documentación técnica:** La ausencia de archivos clave (Dockerfile, requirements.txt) en el proyecto heredado retrasó el despliegue. Esto demuestra la necesidad de estandarizar la documentación técnica desde el inicio.
3. **Gestión de errores como motor de aprendizaje:** Se enfrentaron errores como el fallo en la creación de tablas por SQLAlchemy, tokens JWT malformados y errores 403 sin causa inmediata evidente. Aunque estos problemas no siempre pudieron resolverse por completo ni permitieron avanzar con el despliegue, su identificación, análisis y documentación fueron muy significativos en la comprensión del entorno de ejecución.
4. **Comunicación entre microservicios:** Uno de los principales desafíos fue lograr la interoperabilidad entre los microservicios desplegados en Cloud Run y la base de datos MySQL alojada en una instancia de VM. A pesar de que se verificaron y descartaron configuraciones comunes como reglas de firewall, IP pública, credenciales y estado del servicio, persistieron errores como el **403 Forbidden**. Esto sugiere que podría haber restricciones adicionales a nivel de red, políticas de acceso, o configuraciones específicas de IAM o VPC Serverless Connector que estén bloqueando la comunicación.
5. **Costo de complejidad innecesaria:** Se detectó que ciertos microservicios (como el Battery Service) no justifican su existencia como servicios independientes. Esto genera sobrecarga operativa sin un valor agregado claro.
6. **Despliegue del frontend no contemplado:** Aunque el frontend fue desarrollado con herramientas modernas (Vite, React, TailwindCSS), no se logró desplegar debido a los retos enfrentados en el backend durante este ciclo. Esto deja una parte crítica del sistema sin validar.

Finalmente, esta experiencia evidencia que un despliegue exitoso no depende únicamente del código, sino de un ecosistema bien documentado, colaborativo y con prácticas estandarizadas. Con estas mejoras, el sistema GRID alcanzará mayor confiabilidad, escalabilidad y mantenibilidad en el futuro. Con base en lo aprendido, se proponen las siguientes acciones para el equipo que continúe con el desarrollo y mantenimiento del proyecto:

A. Revisión de la arquitectura

- Evaluar qué microservicios pueden ser fusionados o eliminados. Por ejemplo, el Battery Service puede integrarse como un endpoint en un servicio más robusto.

B. Estándares de documentación

- Exigir que todo microservicio incluya:
    - Archivo Dockerfile.
    - Archivo requirements.txt o equivalente (package.json para Node.js).
    - Descripción de endpoints y lógica básica en un README.md.
    - Variables de entorno requeridas y su propósito.
    - Generar un esquema general del sistema (diagrama de arquitectura y flujos de comunicación).

C. Autenticación y seguridad

- Establecer pruebas unitarias y de integración para verificar la validación de JWT.
- Asegurar que los tokens se generen, transmitan y validen siguiendo las buenas prácticas (longitud, expiración, firma, etc.).
- Usar herramientas de monitoreo en Cloud Run para detectar errores de red o pérdida de conectividad con la base de datos.

D. Gestión del conocimiento

- Establecer una wiki interna o repositorio de documentación viva que detalle:
    - Cómo levantar el entorno local.
    - Cómo desplegar en GCP.
    - Soluciones conocidas a errores frecuentes.
    - Checklist para nuevos desarrolladores del proyecto.