**6-5-2025**

Oscar Andres Macias Narvaez

📝**Propuesta de Implementación  
EVIDENCIA DE CUMPLIMIENTO DE ASRS – APP DJANGO EN GCP**

### CONTROL DE CAMBIOS⚖️

|  |  |  |  |
| --- | --- | --- | --- |
| 📄**Versión** | 🗓️**Fecha** | ✏️**Autor** | 🛠️**Cambios Relevantes** |
| **1.0** | **06-may-25** | **Oscar Macias** | **Análisis inicial de infraestructura** |

**ÍNDICE**

[**OBJETIVO**  3](#_Toc193881614)

[**METODOLOGÍA 📌** 4](#_Toc193881632)

[**ARQUITECTURA PROPUESTA 🗺** 5](#_Toc193881637)

[**EVIDENCIA Y ENTREGABLES 📂** 7](#_Toc193881642)

[**CONCLUSIÓN 🎯** 17](#_Toc193881643)

**REPOSITORIO**  18

**PRÓXIMOS PASOS**🔜 19

[**REFERENCIAS TÉCNICAS ✅** 20](#_Toc193881644)

🛠️ **OBJETIVO**

Se desarrollará una aplicación base utilizando el framework **Django**, la cual será desplegada en **Google Cloud Platform** con el propósito de demostrar el cumplimiento de dos **ASRs (Requisitos de Seguridad de la Arquitectura)** fundamentales:

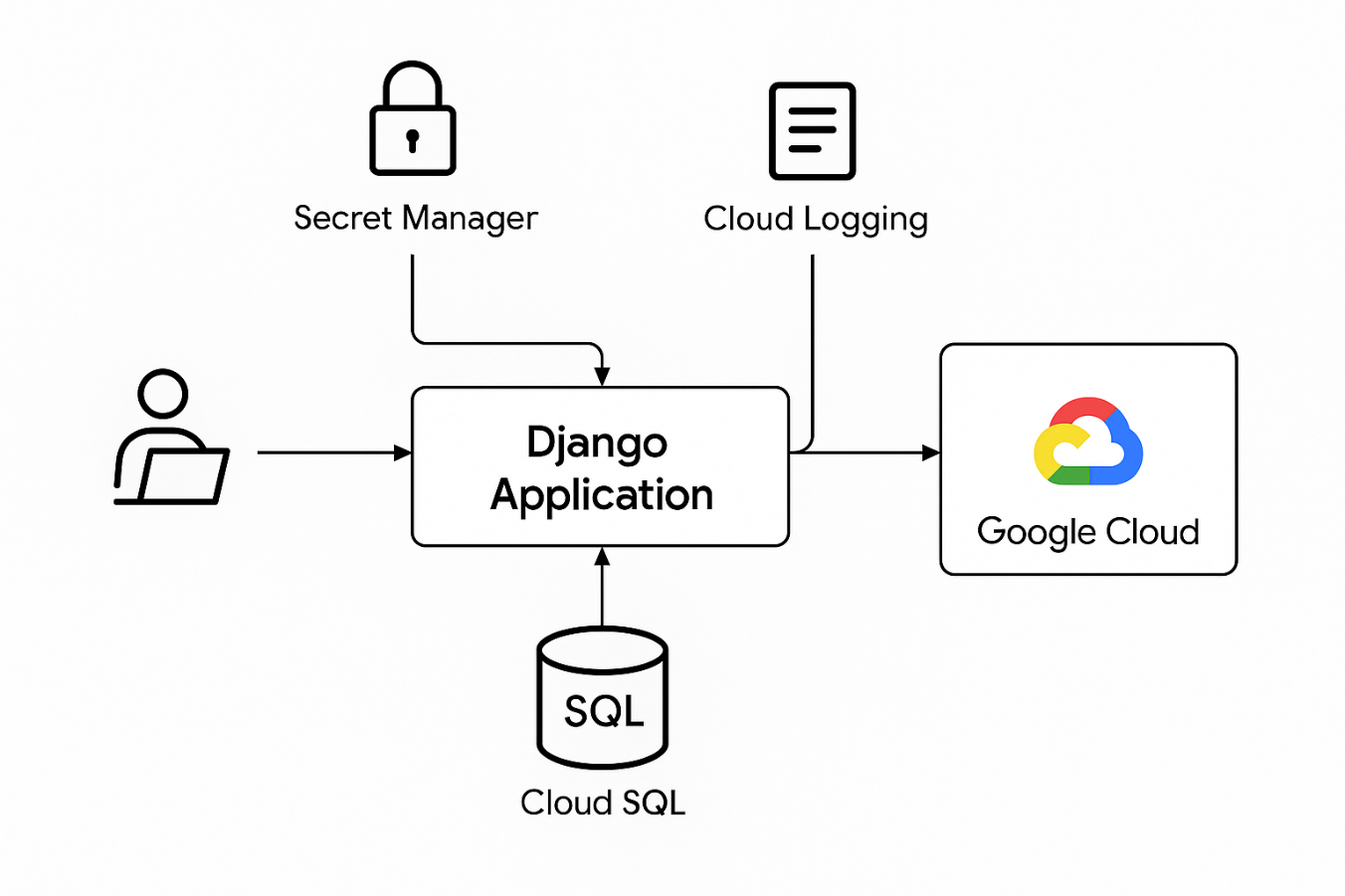
* **🔒Confidencialidad:**  
  Las credenciales de acceso y variables sensibles serán protegidas mediante el uso de archivos .env, la librería django-environ y una configuración segura del entorno. Esto evitará su exposición en el código fuente o en repositorios públicos.
* 🧾**Integridad:**  
  Se implementará un sistema de registro de eventos críticos —como la creación, modificación y eliminación de objetos— en logs estructurados, utilizando el sistema de logging de Django y su integración con **Cloud Logging** en GCP.

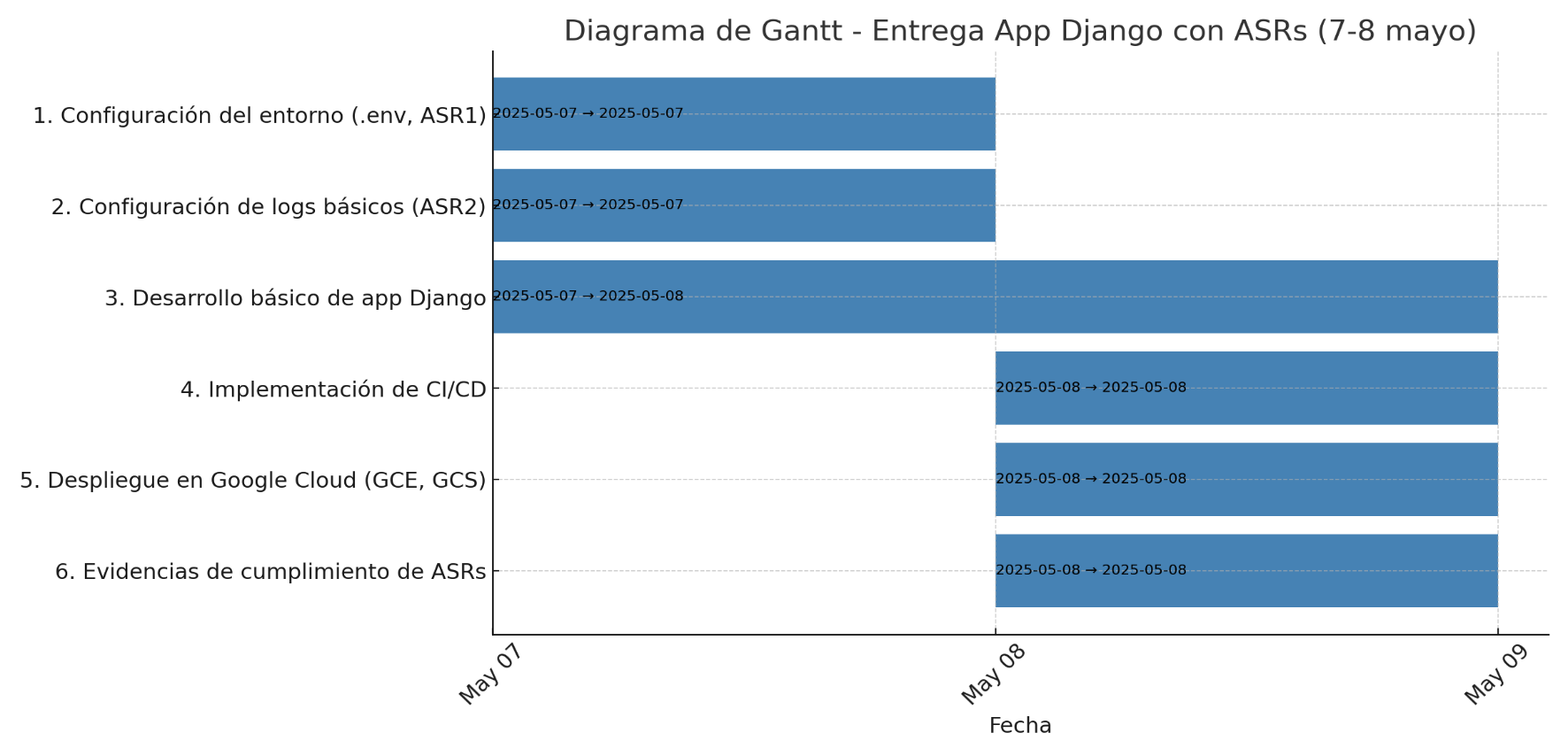
Adicionalmente, se establecerá un pipeline de **CI/CD** con servicios de Google Cloud como **Cloud Build**, **Artifact Registry**, y **Cloud Run**, lo que permitirá automatizar el despliegue continuo de la aplicación, garantizar la rastreabilidad de los cambios y reforzar la seguridad en el ciclo de vida del software.

**METODOLOGÍA 📌**

|  |  |
| --- | --- |
| **Fase** | **Actividad** |
| **🏗 Configuración** | **Crear app Django, repositorio Git, definir modelo básico con tracking.** |
| **🔐 Seguridad** | **Variables en .env, integración con django-environ, sin exposición.** |
| **📜 Logs** | **Configurar logging estructurado en JSON para rastreo de eventos.** |
| **☁ Despliegue en GCP** | **Crear servicios: Cloud Run, Storage, Logging, Secret Manager.**   |  | | --- | |  | |
| **🔁 CI/CD** | **Crear pipeline con Cloud Build que haga push y deploy automático.**   |  | | --- | |  | |
| **🧪 Evidencia**   |  | | --- | |  | | **Validar logs, mostrar variables en entorno seguro y despliegue correcto.** |

**ARQUITECTURA PROPUESTA** 🗺

****

****

**EVIDENCIA Y ENTREGABLES** 📂

* 🗂️ **Código fuente** con archivo .env. example y configuración segura para variables sensibles.
* 📄 **Capturas de logs** mostrando eventos críticos registrados (creación, modificación y eliminación).
* 📸 **Evidencia visual** del despliegue en Google Cloud Platform y ejecución exitosa del pipeline CI/CD.
* 🔗 **Acceso temporal** (opcional) a la aplicación desplegada para revisión funcional.
* 📘 **Documento técnico final** que detalla la implementación, cumplimiento de los ASRs y arquitectura utilizada.

**CONCLUSIÓN 🎯**

📢 El presente actualiza el punto de partida para la toma de decisiones estratégicas. Se recomienda priorizar la optimización para garantizar una plataforma estable, rápida y escalable para los usuarios 🚀

**REPOSITORIO**

Para replicar la prueba, el código fuente y la configuración de la infraestructura utilizada están disponibles en **GitHub**:

**🔗** [**OMaciasd/load-testing-locust**](https://github.com/OMaciasd/load-testing-locust)



**Contenido del Repositorio:**

* **docker-compose.yml:** Definición de los servicios (Locust, InfluxDB, Grafana).
* **locustfile.py:** Escenarios de carga utilizados.
* **grafana-dashboard.json:** Configuración de paneles en Grafana.
* **README.md:** Instrucciones para levantar el entorno y ejecutar pruebas.

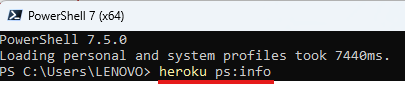
**PRÓXIMOS PASOS** 🔜

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Recurso** | **Pruebas pendientes en Heroku** | **Aplicado** | **Prueba de carga y estrés** | **Superado** |
| 🛢 **PostgreSQL** | 📌 **Configurar Heroku Enterprise con PgBouncer** |  |  |  |
| 💾 **AWS** | 📌 **Realizar la migración aceptada.** |  |  |  |

**📌 Con estas acciones, evaluaremos una segunda opinión de un servicio más eficiente, confiable y rentable.**

**REFERENCIAS TÉCNICAS ✅**

**🔧 Optimizar Dynos con PUMA**

* ****Reducir RAM y mejorar eficiencia, ajustando el número de **threads**.
* Evitar **Memory Quota Exceeded (R14)**.
  + Usa el dashboard de Heroku o Heroku CLI para ver el consumo 📊

- Si ves **R14 (Memory Quota Exceeded)** o **H12 (Request timeout)**, es momento de ajustar workers o escalar dynos.

* + **Heroku mata dynos** si superan la RAM permitida 🎯

max\_requests = 1000

max\_requests\_jitter = 100

* Habilitar **Keep-Alive** para reducir conexiones innecesarias 🛠

keepalive = 5

timeout = 30

* + **1 dyno bien configurado** puede manejar **300 usuarios**.
  + Para **1000 usuarios**, podrías necesitar **4 dynos**, reduciendo el costo en un **60% haciendo uso del auto escalado habilitado.**

**Habilitar logs y monitoreo**

* Utilizar herramientas como **Papertrail** o **LogDNA** para errores en tiempo real.
* Configurar un sistema de alertas para prevenir fallos críticos🔹

**🔥 Optimizar PostgreSQL**

* 🧩 Usar **menos claves y valores más compactos** **🚀**



* + Identificar consultas lentas con **EXPLAIN ANALYZE** 🔹
* Monitorear, e Indexar queries más utilizadas para evitar cuellos de botella.



* + **HSET** crea un hash en Redis con la clave **$user:$VALUE\_USER**.
  + Almacena los campos **$name** con sus respectivos valores.
* Reducir el **TTL** para cacheo ✅



* + **SET** almacena el valor en la clave **$home**.
  + **"<html>...</html>"** representa el contenido **HTML** que se guarda.
  + **EX 600** establece una expiración de **600 segundos (10 minutos)**.
  + Uso de **Read Replicas**.
  + Esto crea una réplica de solo lectura de la base de datos principal🔹
  + La aplicación puede distribuir lecturas entre réplicas para mejorar rendimiento🔹
* Implementar **pooling de conexiones** (PgBouncer), para manejar conexiones y evitar que muchas consultas consuman recursos ✅
  + Si ves algo como **PgBouncer** en la respuesta, significa que la conexión está pasando a través de él.
  + Si la URL comienza con **postgres://** en lugar de **rediss://**, significa que **PgBouncer** está activo.

**💲 Alternativas de Ahorro de Costos**

* Implementar API Gateway (**Kong** / **Envoy (Kubernetes)**) para reducir carga.
* Habilitar **Preboot** para evitar reinicios costosos.



📌 **Usar HST en esta fase (Pruebas de Carga en Staging).**

* 📢 Para garantizar un lanzamiento exitoso, se recomienda integrar un sistema de detección de anomalías que identifique problemas de rendimiento en los tiempos de respuesta, uso de recursos y comportamiento de endpoints durante las pruebas de carga con Locust. Esto permitirá identificar cuellos de botella y problemas de escalabilidad antes del despliegue en producción.