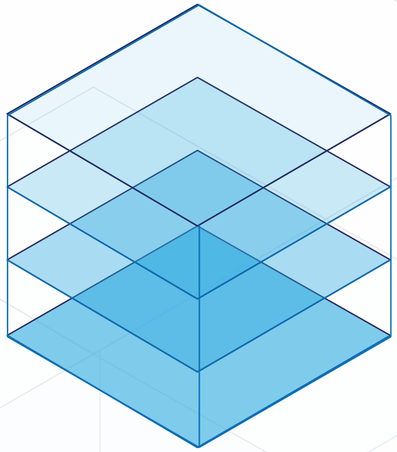


MISO

Maestría en Ingeniería de Software

Resultado Experimentación

Semana 7 – Análisis experimentación

Recursos

El código y los recursos de esta experimentación están disponibles en este repositorio

Los recursos desplegados en cloud run fueron los siguientes:

		Name	Deployment type	Req/sec	Region	Authentication	Ingress	Recommendation	Last deployed	Deployed by
☐	✓	cliente-web	Source	1.84	us-central1	Allow unauthenticated	All	–	2 hours ago	juanprodriguezg29@gmail.com
☐	✓	cr-bff-webapp	Container	0.8	us-central1	Allow unauthenticated	All	–	4 hours ago	juanprodriguezg29@gmail.com
☐	✓	cr-ms-productos	Container	2.78	us-central1	Allow unauthenticated	All	–	4 hours ago	juanprodriguezg29@gmail.com
☐	✓	proyecto-pedidos	Container	0.69	us-central1	Allow unauthenticated	All	–	23 hours ago	juanprodriguezg29@gmail.com

Los resultados de las curvas de distribución se pueden encontrar en este archivo de Excel

La colección de postman para probar los servicios de Cloud Run está en este enlace

Experimento #1

El objetivo principal de este experimento es evaluar cómo las tácticas de seguridad implementadas durante la creación de fabricantes pueden afectar la latencia del sistema, especialmente en escenarios de alta demanda. Se busca determinar la diferencia en el tiempo de respuesta entre un sistema con y sin medidas de seguridad

Metodología

Este experimento se lleva a cabo en un entorno controlado donde se simulan diferentes escenarios de carga y seguridad para analizar el impacto de la latencia. La metodología incluye los siguientes pasos:

Despliegue de los Servicios:

- Plataforma de despliegue: Los servicios se despliegan en Google Cloud Platform (GCP) para manejar las aplicaciones y servicios necesarios.
- Base de datos: Se utiliza PostgreSQL como sistema de gestión de bases de datos para almacenar los datos de los fabricantes.
- Aplicación Backend: Se implementa una API utilizando Flask, para gestionar la creación de fabricantes.
- Servicios adicionales: Se integra Firebase, proporcionando servicios de autenticación.

Tácticas de Seguridad Implementadas:

Durante el experimento, se implementan diferentes tácticas de seguridad, como la autenticación y autorización. Estas tácticas de seguridad afectan directamente la latencia de las operaciones de creación de fabricantes, ya que agregan un tiempo adicional de procesamiento por cada solicitud.

Pruebas de Latencia:

Para medir el impacto de las tácticas de seguridad, se diseñan tres escenarios:

- Escenario sin tácticas de seguridad: Aquí, el sistema no cuenta con ninguna medida adicional de seguridad, y la creación de fabricantes debe ocurrir en un tiempo menor a 2 segundos.
- Escenario con tácticas de seguridad: En este escenario, las medidas de seguridad se implementan sin optimizaciones adicionales. Se espera que el tiempo de creación de un fabricante supere los 3 segundos debido al tiempo adicional que requieren las medidas de seguridad.
- Escenario con tácticas de seguridad y manejo de latencia: En este caso, se combinan las tácticas de seguridad con estrategias para reducir la latencia. El objetivo es lograr una creación de fabricante en un tiempo menor a 3 segundos.

Pruebas de Carga con Locust:

- Locust se utiliza como herramienta para simular diferentes niveles de carga y medir el rendimiento del sistema bajo diferentes condiciones.
- Se configuran pruebas de estrés para generar una carga elevada en el sistema, simulando múltiples usuarios concurrentes que intentan crear fabricantes al mismo tiempo. Esto permite observar cómo las tácticas de seguridad afectan la latencia bajo condiciones de alta demanda.
- Durante estas pruebas, Locust mide métricas como el tiempo de respuesta promedio, el tiempo de respuesta máximo, el porcentaje de fallos y la capacidad de manejar múltiples solicitudes simultáneas.

Análisis de Resultados:

Después de realizar las pruebas en los tres escenarios, se analizan los datos recolectados para evaluar el impacto de las tácticas de seguridad en la latencia.

Experimento #1

Preparación: Determinar tiempo de autenticación de Firebase

Debido a la política de Firebase de bloquear direcciones IP que realicen muchas peticiones en corto tiempo; fue necesario realizar un estimado del periodo de latencia que se realiza en una autenticación completa para estimar este tiempo en un escenario de alta demanda.

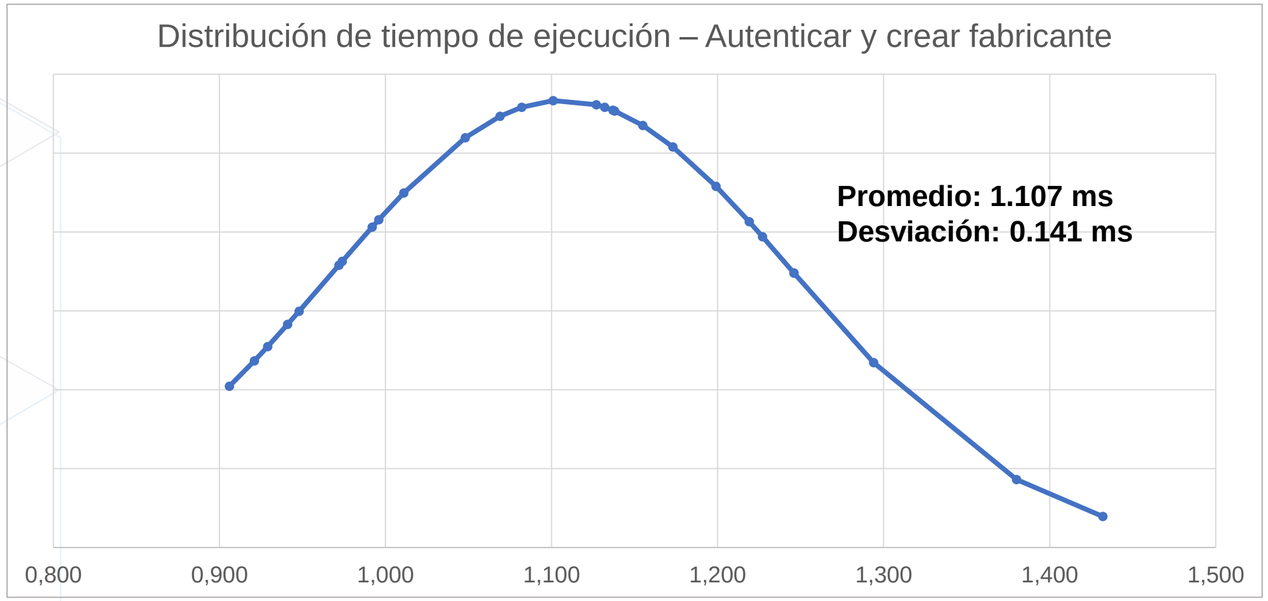
Se plantearon los siguientes supuestos:

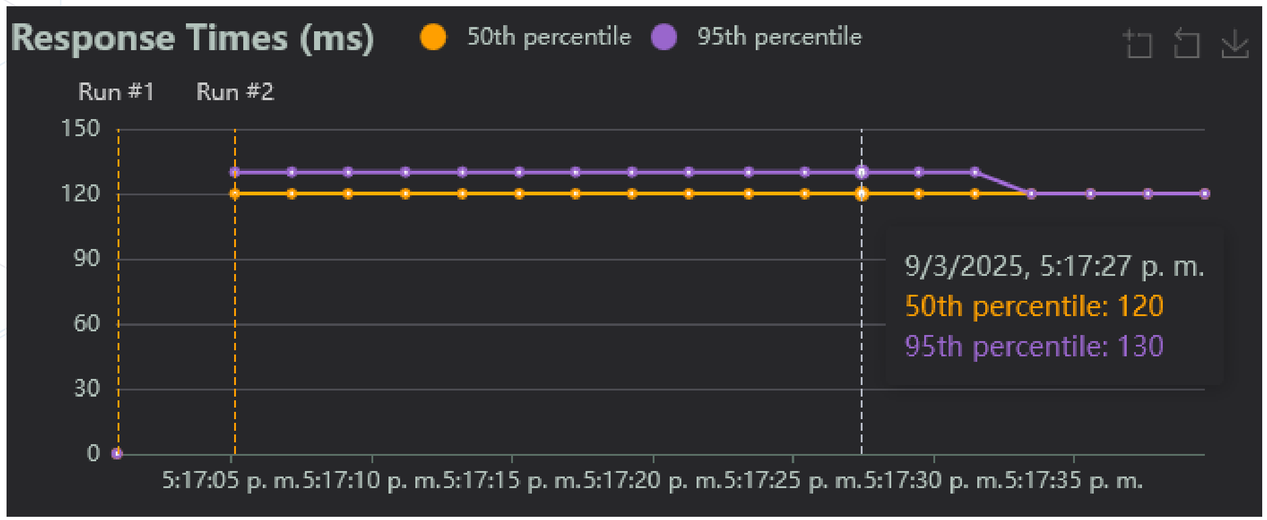
- La infraestructura de Firebase tiene una variación de latencia muy baja
- El tiempo de autenticación se mide como la diferencia entre el tiempo total de ejecución y el tiempo de respuesta del backend

Para ello se realizaron múltiples peticiones para obtener una curva de distribución, dónde se encontró que el tiempo de ejecución total es, en promedio, 1.11 segundos

Finalmente, se ejecutó el mismo proceso sin autenticar y se determinó que el tiempo de ejecución del backend es, con una confiabilidad del 95%, de 130ms.

Es decir, el tiempo de autenticación se establecerá para las siguientes pruebas que es, en promedio, de 980ms





Experimento #1

Pruebas sin mecanismos de seguridad

Con respecto a las pruebas sin seguridad se realizaron 3 escenarios con peticiones directas al API Gateway:

- Escenario con 400 usuarios concurrentes y 3 instancias en el BFF
- Escenario con 800 usuarios concurrentes y 3 instancias en el BFF
- Escenario con 800/600 usuarios concurrentes y 6 instancias en el BFF

- Los resultados muestran que el sistema se encuentra limitado a 220 peticiones en concurrencia en todos los escenarios, con un empeoramiento general del tiempo de respuesta.
- El mejor escenario es el primero, **ya que se consigue en promedio un tiempo de respuesta de 1800ms con el mismo nivel de procesamiento**, en comparación al resto donde el tiempo de respuesta escala a un promedio de 3500 ms
- No se observaron fallos en la respuesta durante la creación

Experimento #1

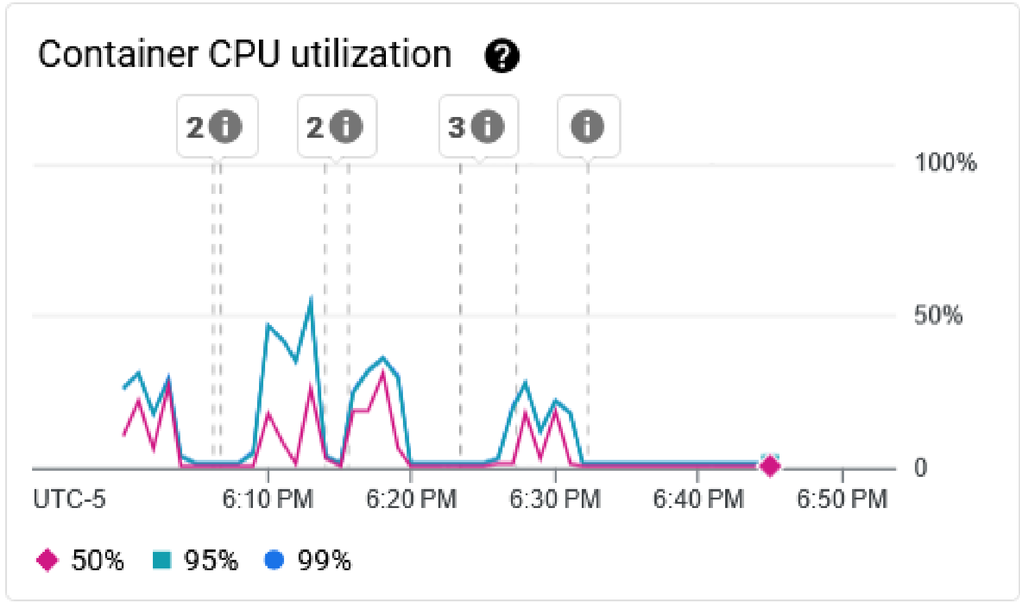
Pruebas sin mecanismos de seguridad

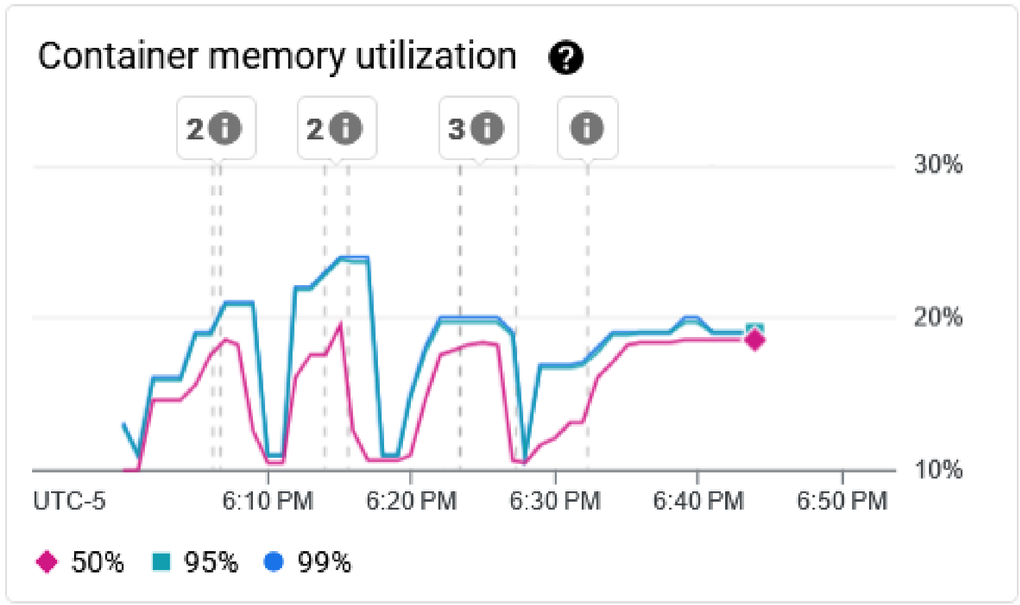
- Adicionalmente, no se observó que, bajo la configuración planteada, hubiese un elevado uso de recursos de CPU y memoria. Por tal razón, es posible descartar el límite de recursos en el Gateway.

- **Las anteriores conclusiones sugieren que se requiere de un sistema adicional para gestionar las peticiones y evitar la pérdida de información**

- **Por otra parte, el uso de recursos muestra que existe alguna configuración limitante dentro de la estructura de los servicios. Es necesario explorar las configuraciones de despliegue de Flask y el servidor WSGI**

Para las pruebas con autenticación y tácticas de caché, se utilizará únicamente el primer escenario





Experimento #1

Pruebas con mecanismos de seguridad y reducción de latencia

Para la prueba con mecanismos de seguridad se implementó un frontend construido en Next.JS y desplegado en cloud para simular el ambiente real en el que se ejecutará el servicio. En este caso se utilizó una estrategia para almacenar en caché el token de autenticación y así evitar los llamados a Firebase.

Las pruebas se ejecutaron usando un script construido en Playwright para poder ejecutar un script en la página web que se encarga de hacer el llamado al API Gateway

- Los resultados muestran que el sistema tiene un tiempo de carga promedio de 5597ms. Teniendo en cuenta el experimento con Firebase, **podemos estimar el tiempo con autenticación en 6577ms**
- No se observaron fallos durante la creación, y la base de datos contiene todos los objetos, sin embargo, el tiempo de creación es, en promedio, 2597ms más a lo especificado en los requisitos de calidad

Estas prueba demuestra que la hipótesis de diseño no se cumple, y que es necesario realizar un reajuste en este proceso, y todos aquellos que tengan una alta concurrencia.

Total Users: 400
Successful Requests: 400
Failed Requests: 0
Avg Load Time: 5597.34ms
Min Load Time: 803ms
Max Load Time: 13019ms

Experimento #2

El propósito de este experimento es identificar si es posible alcanzar un tiempo de escalamiento inferior a los 3 minutos cuando el sistema supera un límite de solicitudes concurrentes bajo los patrones de seguridad implementados. Específicamente, el experimento busca evaluar si se puede replicar correctamente el componente de pedidos dentro del tiempo estipulado

Metodología

Este experimento se lleva a cabo en un entorno controlado donde se simulan diferentes escenarios de carga y seguridad para analizar el impacto de la latencia. La metodología incluye los siguientes pasos:

Despliegue de los Servicios:

- Plataforma de despliegue: Los servicios se despliegan en Google Cloud Platform (GCP) para manejar las aplicaciones y servicios necesarios.
- Base de datos: Se utiliza PostgreSQL como sistema de gestión de bases de datos para almacenar los datos de los fabricantes.
- Aplicación Backend: Se implementa una API utilizando Flask, para gestionar la creación de fabricantes.
- Servicios adicionales: Se integra Firebase, proporcionando servicios de autenticación.

Manejo de Replicación y Escalamiento:

Durante el experimento, el sistema debe ser capaz de escalar correctamente en función de la carga y de replicar el componente de pedidos dentro de un tiempo de menos de 3 minutos. El objetivo es evitar la pérdida de solicitudes durante el escalamiento para lo que se implementan políticas de manejo de concurrencia que garanticen que el sistema pueda recibir y procesar todas las solicitudes de creación de pedidos durante el proceso de escalado.

Manejo de Concurrencia y Pérdida de Solicitudes:

Para reducir el número de autenticaciones y mejorar el tiempo de respuesta, se almacenarán datos en caché del cliente. Esto disminuye la carga sobre los sistemas de autenticación y acelera las respuestas, lo que puede ayudar a evitar la pérdida de solicitudes en situaciones de alta carga.

Pruebas de Carga con Locust:

- Locust se utiliza como herramienta para simular diferentes niveles de carga y medir el rendimiento del sistema bajo diferentes condiciones.
- Se simula un número elevado de usuarios concurrentes que realizan solicitudes de creación de pedidos.
- La herramienta permitirá evaluar el rendimiento del sistema bajo carga y comprobar si el escalamiento es capaz de manejar adecuadamente el número de solicitudes sin perder ninguna.

Análisis de Resultados:

Se analiza el tiempo completo necesario para replicar el componente de pedidos y se verifica si este se mantiene por debajo de los 3 minutos durante el escalamiento.

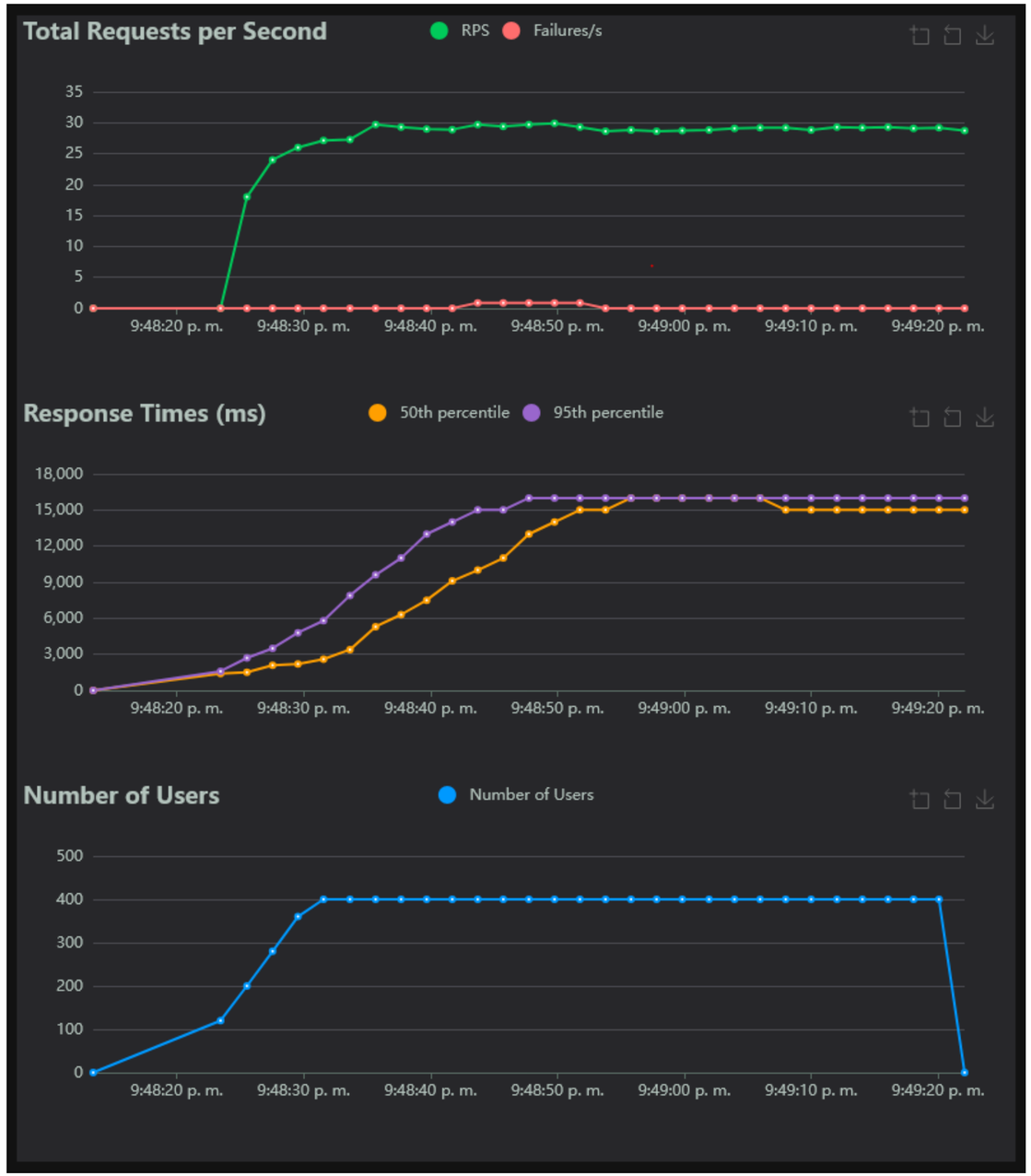
Experimento #2

Pruebas de tiempo de escalamiento

El experimento realizado involucraba la interacción entre todos los componentes de gestión de pedidos y productos. Esta interacción compleja permite el escenario perfecto de escalado para nuestro sistema.

- Para demostrar esta hipótesis, realizamos las pruebas de esfuerzo sobre el BFF para mostrar la capacidad de nuestro sistema.

- En este caso el sistema solo puede procesar hasta 30 pedidos en concurrente, con un tiempo de respuesta de más de 15 segundos para un escenario con 400 usuarios concurrentes



Experimento #2

Pruebas de tiempo de escalamiento

- Para el experimento 2 se realizó nuevamente la prueba realizada para el experimento 1 en el frontend con autenticación

- La prueba muestra una distribución normal hasta los 8000ms, donde luego se tienen varias peticiones que llegan hasta los 34214ms.

- **No se encontraron fallas en las peticiones, y los 400 pedidos fueron creados**

Total Users: 400
Successful Requests: 400
Failed Requests: 0
Avg Load Time: 7236.83ms
Min Load Time: 1262ms
Max Load Time: 34214ms

Experimento #2

Pruebas de tiempo de escalamiento

- Observando el escalamiento del componente web, se observa un valle entre las 9:44 PM y las 9:46 PM. Este espacio muestra el tiempo de inactividad de los contenedores de Cloud Run.

- Por otra parte, entre las 9:46 PM y las 9:47 PM, el sistema escala hasta 2 instancias del contenedor, demostrando que el sistema puede escalar en un tiempo de alrededor de 1 minuto

- Entre 9:39 PM y 9:43 PM se observa la última ejecución del experimento 1, demostrando la capacidad de escalado del sistema

En el caso del experimento, se demostró que el tiempo de escalamiento, incluso con un backend complejo, se pudo hacer en un periodo de tiempo corto, en especial con herramientas como Cloud Run de GCP. Esto permite demostrar que se cumplió la hipótesis de diseño.

Sin embargo, **el tiempo elevado de respuesta demuestra que es necesario implementar un sistema para la gestión de las peticiones.** El tiempo actual de respuesta en un ambiente síncrono es completamente inaceptable para un usuario

Diagrama refinado - Componentes

A partir de los resultados obtenidos se plantea el siguiente diagrama de componentes como una respuesta a la baja latencia del sistema, se agrega manejo de eventos para las peticiones que llegan a los componentes de Productos y GestorPedidos, considerados como cruciales en la arquitectura

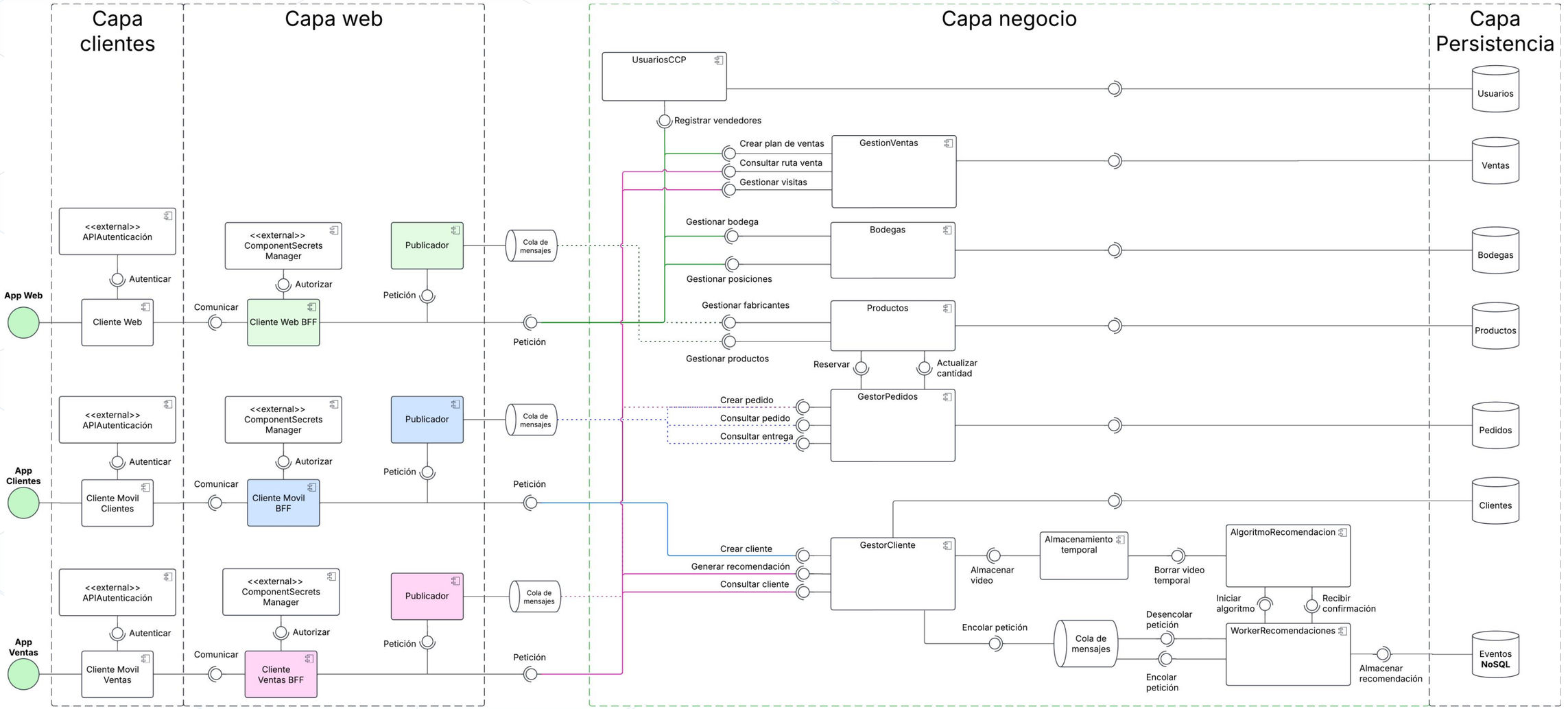
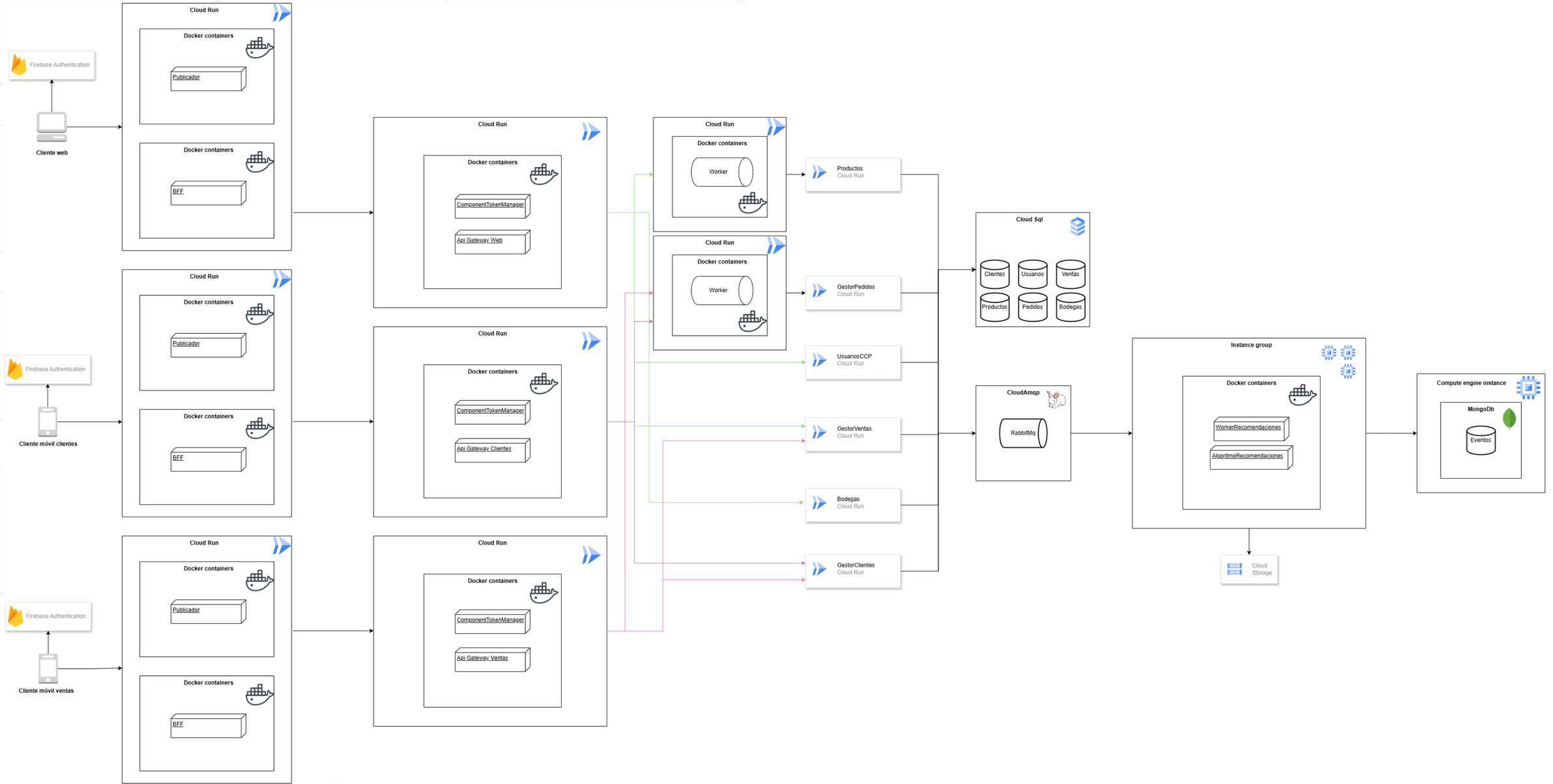


Diagrama refinado - Despliegue

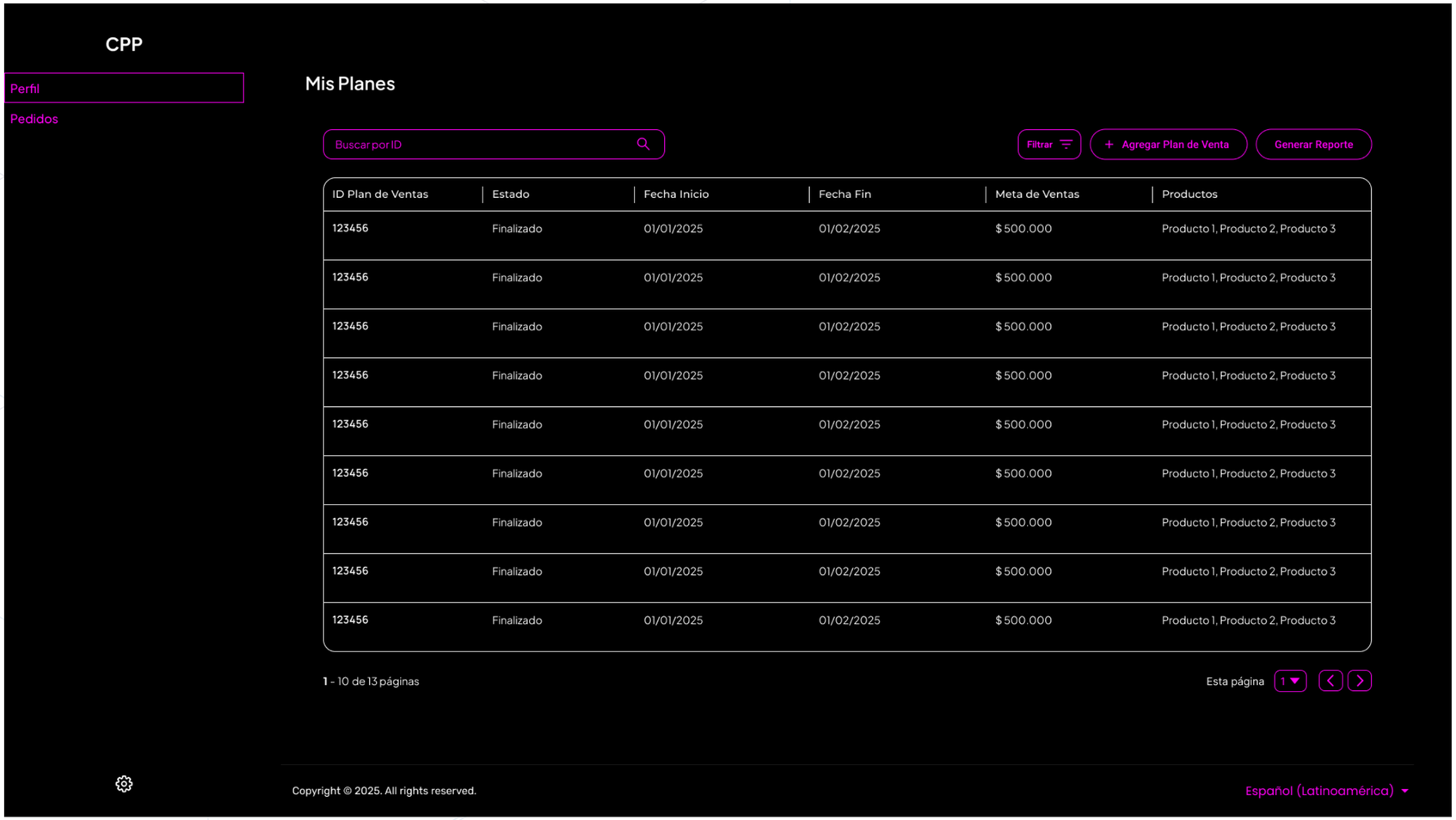
A partir de los resultados obtenidos se plantea el siguiente diagrama despliegue como una respuesta a la baja latencia del sistema. Diagrama con mejor resolución en este link. El refinamiento de este diagrama incluye el cambio en el uso de máquinas virtuales a instancias de Cloud Run, un publicador para cada cliente que se ejecuta en un docker container junto al BBF y dos brokers de eventos de los cuales consumirán productos y gestor pedidos.



Diseño y Prototipo UI

En este enlace se puede consultar el diseño de la interfaz de usuario (UI), el cual incluye características de accesibilidad, como el uso de la tipografía Lexend, diseñada para facilitar la lectura en personas con dislexia. Además, el diseño incorpora un tema de alto contraste y la opción de ajustar el tamaño de la fuente, mejorando la experiencia para usuarios con diferentes necesidades visuales.

Ejemplos mencionados



MISO
Maestría en Ingeniería de Software

CPP

Productos
Pedidos

Productos

Busca por SKU, lote, nombre, etc.

Filtrar
+ Ingresar Lotes

Nombre Producto	SKU	Lote	Volumen	Bodega	Posición	Cantidad Disponible	Cantidad Reservada	Fecha Ingreso
ID 12345 Producto 1	ABC-12345	1	1000	550e8400-e29b-41d4	a716-446655440000	500	500	01/01/01
ID 12345 Producto 1	ABC-12345	1	1000	550e8400-e29b-41d4	a716-446655440000	500	500	01/01/01
ID 12345 Producto 1	ABC-12345	1	1000	550e8400-e29b-41d4	a716-446655440000	500	500	01/01/01
ID 12345 Producto 1	ABC-12345	1	1000	550e8400-e29b-41d4	a716-446655440000	500	500	01/01/01
ID 12345 Producto 1	ABC-12345	1	1000	550e8400-e29b-41d4	a716-446655440000	500	500	01/01/01
ID 12345 Producto 1	ABC-12345	1	1000	550e8400-e29b-41d4	a716-446655440000	500	500	01/01/01
ID 12345 Producto 1	ABC-12345	1	1000	550e8400-e29b-41d4	a716-446655440000	500	500	01/01/01
ID 12345 Producto 1	ABC-12345	1	1000	550e8400-e29b-41d4	a716-446655440000	500	500	01/01/01
ID 12345 Producto 1	ABC-12345	1	1000	550e8400-e29b-41d4	a716-446655440000	500	500	01/01/01

1 - 10 de 13 páginas

Esta página 1

Español (Latinoamérica)

CPP

Fabaricantes

Productos

Productos

Busca por SKU, lote, nombre, etc.

Filtrar

+ Registrar Producto

Nombre Producto	SKU	Volumen	Fabricante	Valor Unitario	Fecha de Creación
ID 12345 **Producto 1**	10000	1000	Ortiz Group	$102750 COP	06/03/2025
ID 12345 **Producto 1**	10001	1000	Ortiz Group	$102750 COP	06/03/2025
ID 12345 **Producto 1**	10002	1000	Ortiz Group	$102750 COP	06/03/2025
ID 12345 **Producto 1**	10003	1000	Ortiz Group	$102750 COP	06/03/2025
ID 12345 **Producto 1**	10004	1000	Ortiz Group	$102750 COP	06/03/2025
ID 12345 **Producto 1**	10005	1000	Ortiz Group	$102750 COP	06/03/2025
ID 12345 **Producto 1**	10006	1000	Ortiz Group	$102750 COP	06/03/2025
ID 12345 **Producto 1**	10007	1000	Ortiz Group	$102750 COP	06/03/2025

1 - 10 de 13 páginas

Esta página 1

Español (Latinoamérica)