Informe de contexto

Integrantes: Catalina Herrera, Camilo Fuentes, Demian Maturana.

1. Definición del Contexto: Plataforma de Eventos "TicketFlow"

El proyecto **"TicketFlow"** simula una plataforma de venta de tickets para eventos masivos de alta concurrencia. En sistemas tradicionales, el proceso de generar la boleta fiscal y el ticket PDF suele ralentizar la venta de asientos. Para solucionar esto, se propone una arquitectura distribuida que desacopla la **reserva del asiento** (proceso crítico y sincrónico) de la **emisión del documento** (proceso administrativo y asincrónico). El objetivo es garantizar que el usuario asegure su entrada incluso si el servicio de facturación experimenta latencia o caídas, por lo tanto, el sistema debe mantener la venta operativa incluso si el servicio de facturación colapsa y tener en cuenta el "¿Cómo evito vender el mismo asiento dos veces si se cae el sistema?".

2. Arquitectura Técnica y Stack Tecnológico

Para cumplir con el requisito de heterogeneidad y aislamiento de fallos, se define la siguiente arquitectura:

- **Aplicación 1 (Gestor de Recintos y Reservas):**
    - **Rol:** Servicio crítico encargado de la integridad del mapa de asientos y bloqueo de reservas para evitar sobreventa (*double-booking*).
    - **Tecnología: Go (Golang).** Se ha seleccionado este lenguaje por su eficiencia en el manejo de alta concurrencia y bajo consumo de recursos, ideal para evitar condiciones de carrera (race conditions) en la reserva de entradas sin la sobrecarga de frameworks más pesados.
    - **Datos: PostgreSQL**. Configurada con replicación Maestro-Esclavo. Si el maestro cae, el orquestador promueve al esclavo para mantener la venta activa.
- **Aplicación 2 (Motor de Emisión y Facturación):**
    - **Rol:** Servicio de procesamiento asíncrono que genera los tickets en PDF y emite la boleta fiscal electrónica.
    - **Tecnología: Python (Flask)**. Se elige por su agilidad de desarrollo y la disponibilidad de librerías robustas para la generación de documentos (ej. ReportLab o WeasyPrint), facilitando la tarea de emitir boletas complejas.
    - **Datos: MariaDB**. Seleccionada para manejar estructuras JSON complejas de facturación electrónica.

- **Middleware (Orquestador de Eventos):**
    - **Tecnología: Go,** integrado con un Message Broker (**RabbitMQ**). Elegido porque Go es nativamente excelente para hacer de "Orquestador" o Proxy, que es lo que hace el Middleware.
    - **Rol:** Recibe la orden de compra desde la App 3. Publica el evento OrdenCreada. Actúa como buffer; si la App 2 cae, guarda los mensajes hasta que el servicio se recupere.
- **Aplicación 3 (Portal de Venta):**
    - **Rol:** Interfaz web para el usuario final. Solo mantiene réplica de lectura del catálogo de eventos.
    - **Tecnología: Python (Flask)**. Actúa como servidor web ligero (Backend-for-Frontend) que expone las vistas al usuario y se comunica con el Middleware.

3. Estrategia de Tolerancia a Fallos y Pruebas

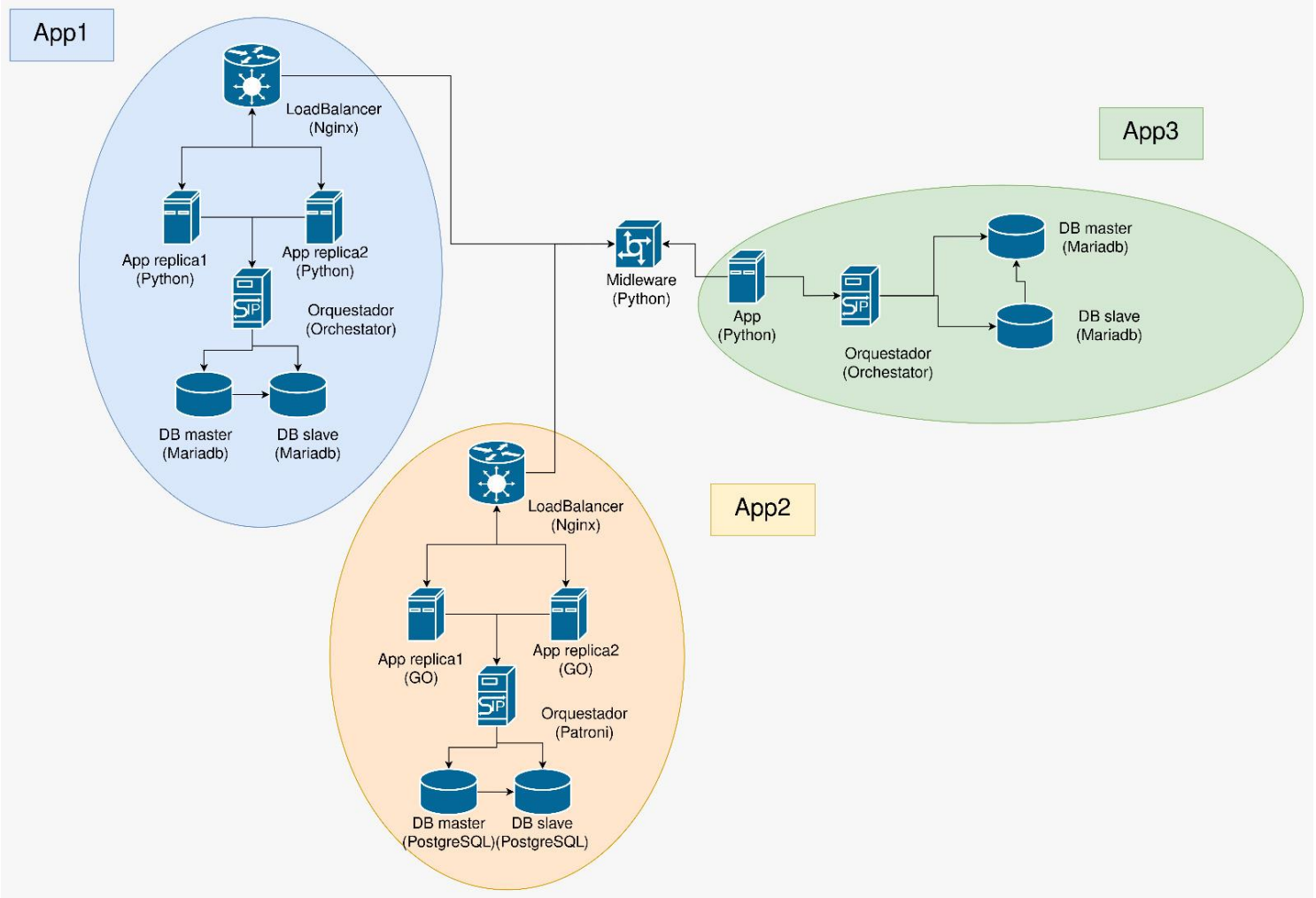
Se implementará un modelo de **Consistencia Eventual**. Las pruebas a documentar serán:

1) **Prueba de Integridad (Caída de App 2):** Se simulará la caída del contenedor de Python (Facturación) durante una compra.
    - *Resultado Esperado:* El usuario recibe confirmación "Compra Exitosa" (procesada por Java). El ticket PDF queda "En Proceso". Al levantar la App 2, el sistema consume la cola de RabbitMQ y envía el correo automáticamente.
2) **Prueba de Disponibilidad de Datos (Caída de MariaDB):** Se detendrá la instancia principal de MariaDB.
    - *Resultado Esperado:* El Orquestador detecta el fallo y redirige las consultas de lectura a la réplica. La venta se detiene momentáneamente (o pasa a modo solo lectura), pero el usuario puede seguir consultando sus tickets previos sin error 500.

4. Definición de SLA (Acuerdo de Nivel de Servicio) y SLO (Objetivos de Nivel de Servicio).

Componente	SLA (Disponibilidad)	Tiempo de Recuperación (RTO)	Estrategia
Reserva de Asientos (App 1)	99.9% (Crítico)	< 1 minuto	Replicación automática de BD.
Emisión de Boletas (App 2)	99.0% (No Crítico)	< 1 hora	Cola de mensajes persistente.
Portal Usuarios (App 3)	99.50%	< 5 minutos	Balanceo de carga.

Componente	Tipo de Métrica	SLO (Objetivo Técnico)	Justificación
Reserva de Asientos (App 1)	Latencia de Escritura	El 99% de las reservas se confirman en < 300 ms.	Al ser un servicio crítico, la respuesta debe ser inmediata para evitar conflictos de concurrencia.
Reserva de Asientos (App 1)	Tasa de Errores	Menos del 0.01% de peticiones fallidas (HTTP 500).	La integridad del mapa de asientos es prioritaria sobre cualquier otra función.
Emisión de Boletas (App 2)	Tiempo de Procesamiento	El 95% de los tickets PDF se generan en < 5 minutos.	Dado que es un proceso asíncrono y administrativo, se tolera un retraso, pero no indefinido.
Middleware (RabbitMQ)	Persistencia	100% de mensajes de compra guardados en disco.	Actúa como **buffer** ante caídas de la App 2, por lo que no puede perder ningún mensaje "OrdenCreada".
Portal Usuarios (App 3)	Latencia de Lectura	El 95% de las cargas del catálogo ocurren en < 10 segundos.	Una carga lenta en la interfaz provoca abandono de compra por parte del usuario.


App1
LoadBalancer
(Nginx)
App replica1
(Python)
App replica2
(Python)
Orquestador
(Orchestator)
DB master
(Mariadb)
DB slave
(Mariadb)
Midleware
(Python)
App3
App
(Python)
Orquestador
(Orchestator)
DB master
(Mariadb)
DB slave
(Mariadb)
App2
LoadBalancer
(Nginx)
App replica1
(GO)
App replica2
(GO)
Orquestador
(Patroni)
DB master
(PostgreSQL)
DB slave
(PostgreSQL)