

# Modernização da Plataforma "ProntoBurger"

## 1. O Contexto

A "ProntoBurger" é uma rede de fast-food (QSR) em plena expansão. Atualmente, opera com 150 lojas próprias e está iniciando um agressivo modelo de franquias. O ecossistema tecnológico atual é composto por sistemas legados que não suportam mais a complexidade e o volume das operações, gerando gargalos, inconsistência de dados e dificuldade para inovar.

Ecossistema Atual:

- POS (Ponto de Venda): Um sistema monolítico, on-premise, instalado em cada loja, com sincronização de dados via batch no final do dia.
- E-commerce: Uma plataforma web e um aplicativo móvel básicos, desenvolvidos por um fornecedor externo, com integração direta e frágil com o sistema de ERP.
- ERP: Sistema responsável por finanças, estoque e etvc.
- Agregadores de Delivery (iFood, Rappi): As integrações são feitas manualmente em cada loja por meio de um tablet, sem conexão com o POS.

Visão de Futuro: A liderança da "ProntoBurger" deseja criar uma plataforma unificada, escalável e resiliente que se torne a espinha dorsal da operação, suportando a expansão para 2.000 lojas (próprias e franqueadas) nos próximos 3 anos.

## 2. O Desafio

Você é o responsável pela arquitetura de soluções da "ProntoBurger". Sua primeira grande missão é desenhar a arquitetura de solução de ponta a ponta para a nova plataforma digital da empresa.

Esta plataforma deve centralizar a lógica de negócios, unificar a experiência do cliente em todos os canais e fornecer dados em tempo real para a tomada de decisões.

## 3. Objetivos de Negócio

1. Experiência Omnichannel: Unificar a jornada do cliente, permitindo que um pedido iniciado no app possa ser concluído ou retirado na loja, por exemplo.
2. Operação Centralizada: Criar um motor de promoções e cardápio centralizado, que possa ser distribuído para todos os canais (POS, App, Web, Agregadores) em tempo real.
3. Inteligência de Negócio: Fornecer dashboards em tempo real para a gestão (vendas por loja, itens mais vendidos, tempo de preparo).
4. Escalabilidade para Franquias: A arquitetura deve suportar o modelo de franquias, permitindo a fácil integração de novas lojas e a gestão isolada de seus dados operacionais.

## 4. Requisitos Não-Funcionais Chave

- Alta Disponibilidade: A plataforma deve ter um SLA de 99.95%.
- Escalabilidade: Suportar picos de 1000 pedidos por minuto durante o horário de almoço.
- Segurança: Conformidade com as boas práticas de segurança
- Baixa Latência: O tempo de resposta das APIs críticas para o cliente final não deve exceder 200ms.

## 5. O Que Deve Ser Entregue

O(a) candidato(a) deverá preparar e apresentar os seguintes artefatos:

1. Documento de Arquitetura (Architecture Overview): Um documento conciso explicando a visão geral da solução. Deve conter:

- Princípios arquiteturais que nortearam o desenho.
- Justificativa das principais escolhas tecnológicas (Cloud Provider, mensageria, banco de dados, etc.).
- Visão de como a Engenharia de Plataforma (Platform Engineering) habilitará as equipes de desenvolvimento.

2. Diagramas de Arquitetura:

- Diagrama de Contexto (C4 Model): Visão macro do sistema, seus usuários e suas interações com sistemas externos.
- Diagrama de Contêineres (C4 Model): Detalhamento dos principais blocos da solução (ex: Microsserviços, Bancos de Dados, API Gateway, Message Broker), como eles se comunicam e as tecnologias envolvidas.
- Diagrama de Sequência: Ilustrar o fluxo de um evento crítico, como a "Criação de um Pedido" vindo do aplicativo móvel, passando pela plataforma e chegando ao sistema da cozinha na loja (KDS - Kitchen Display System)

plataforma e integrar ao sistema da cozinha na loja (KDS - Kitchen Display System).

3. ADRs (Architecture Decision Records): Elaborar **2 a 3 ADRs** para as decisões arquiteturais mais críticas. Exemplos:

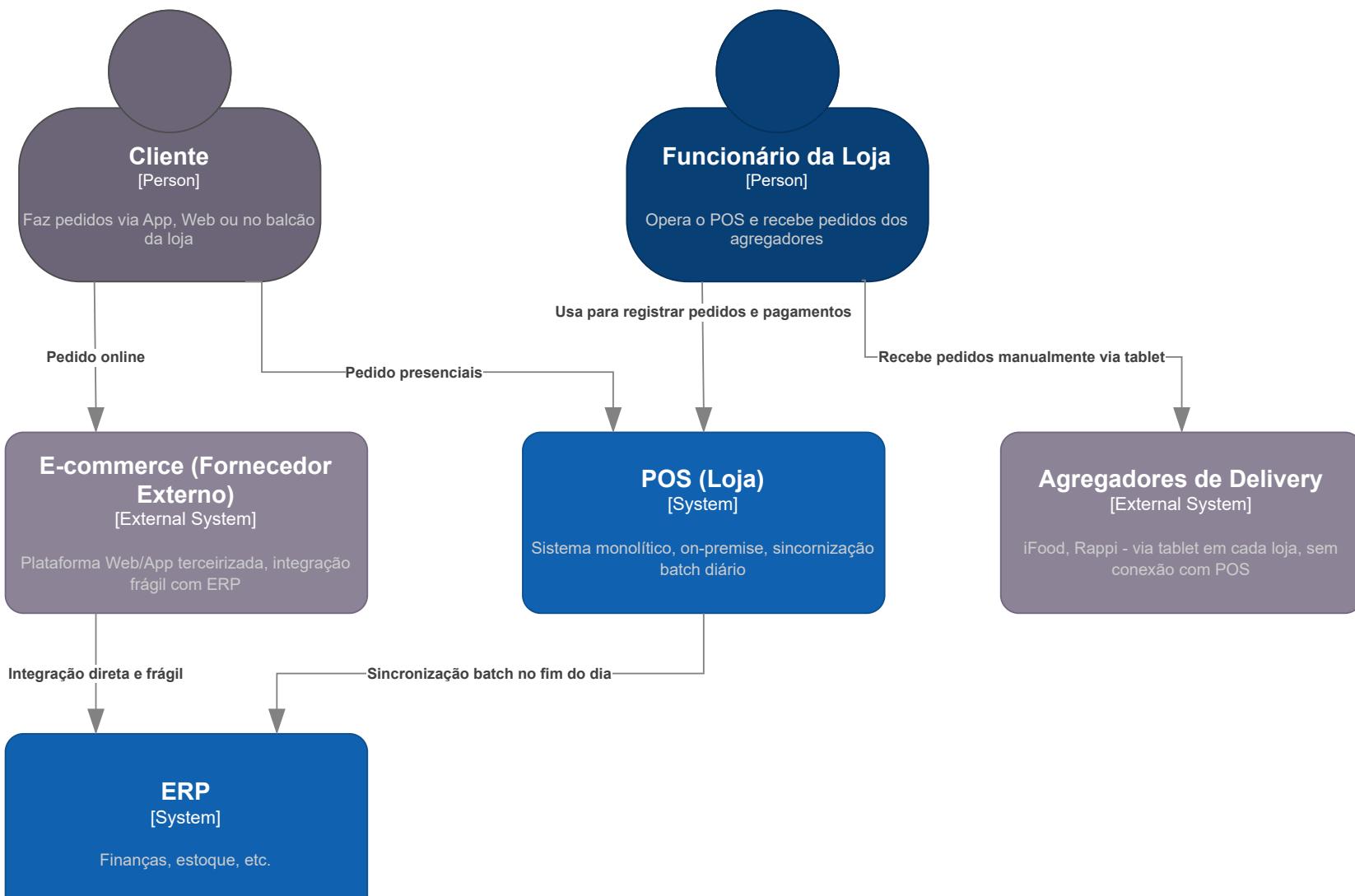
- Escolha do mecanismo de mensageria (ex: Kafka vs. RabbitMQ vs. SQS/SNS).
- Estratégia de persistência de dados para o serviço de pedidos (ex: SQL vs. NoSQL).

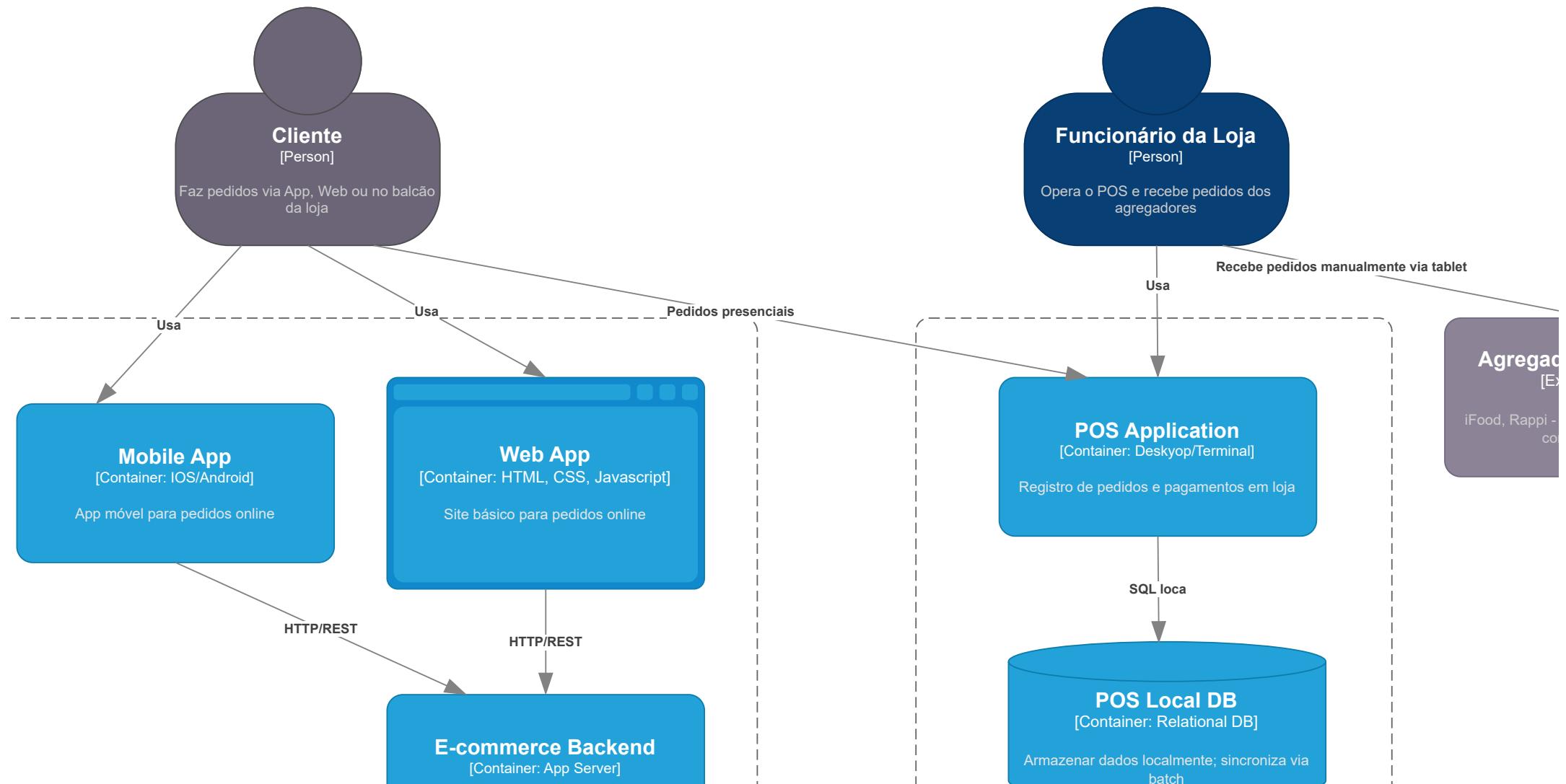
4. Modelagem de Domínio (DDD):

- Identificar e delinear os principais **Bounded Contexts** (Contextos Delimitados) da solução (ex: Pedidos, Pagamentos, Cardápio, Promoções, Estoque).
- Propor um **Context Map simplificado**, mostrando como esses contextos se relacionam.

## 6. Apresentação

O(a) candidato(a) terá 60 minutos para apresentar sua solução para um painel de arquitetos e líderes técnicos





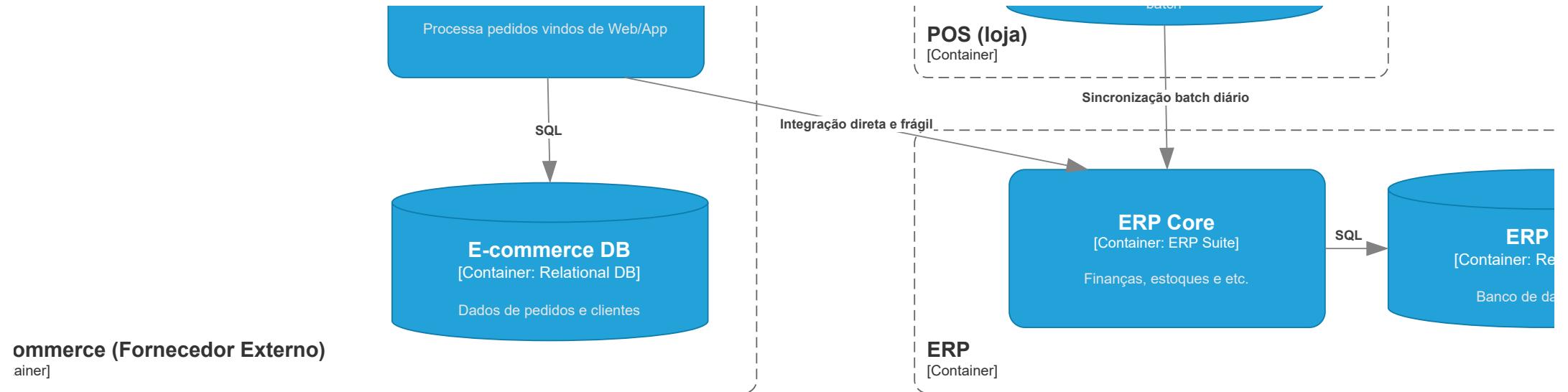


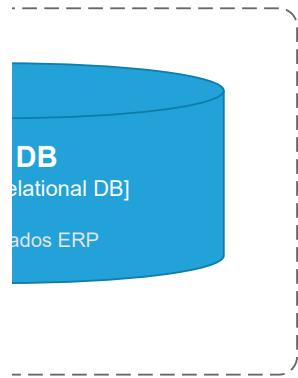
## **dores de Delivery**

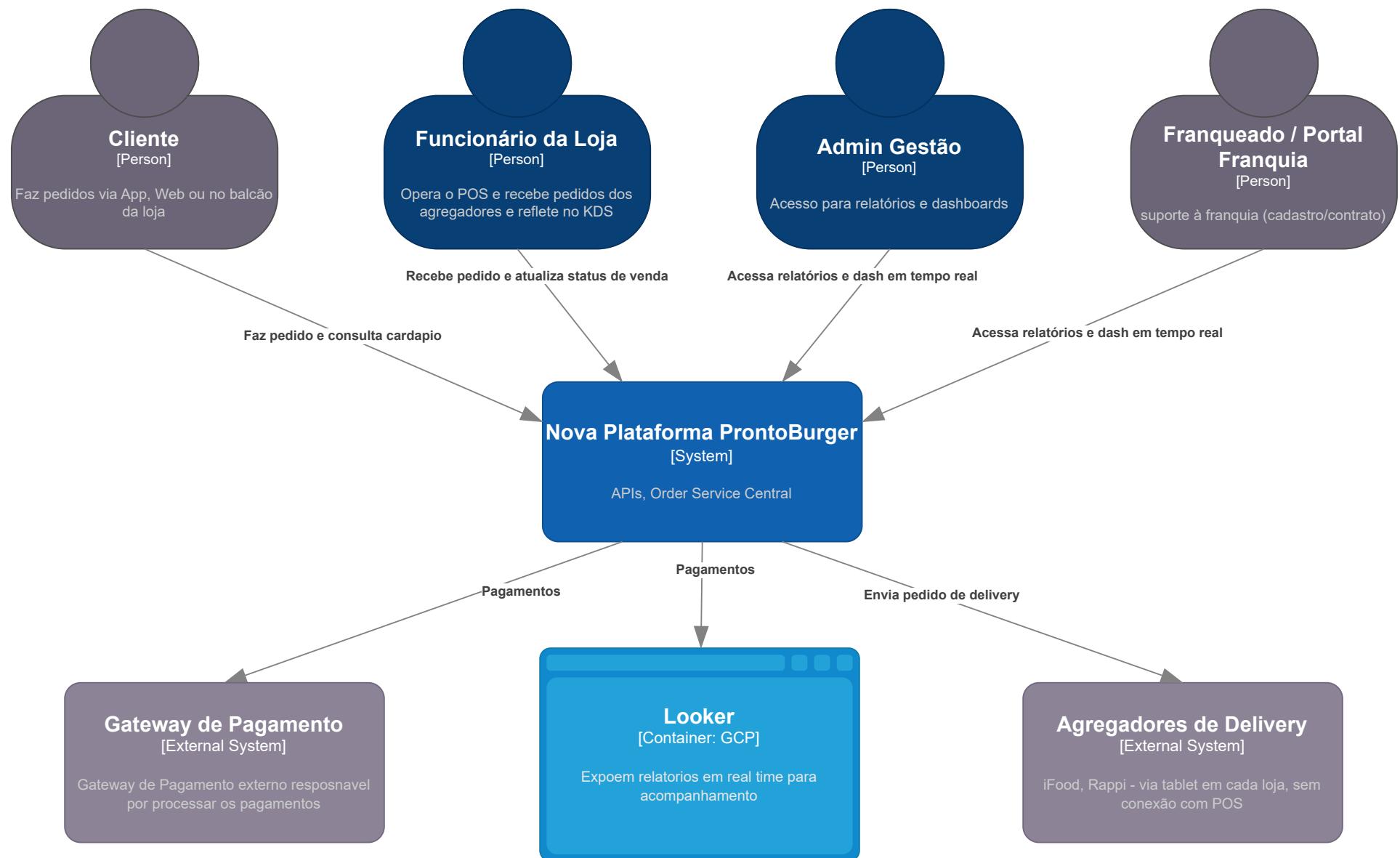
[External System]

via tablet em cada loja, sem  
nexão com POS

**E-ct**  
[Cont:  
— — —

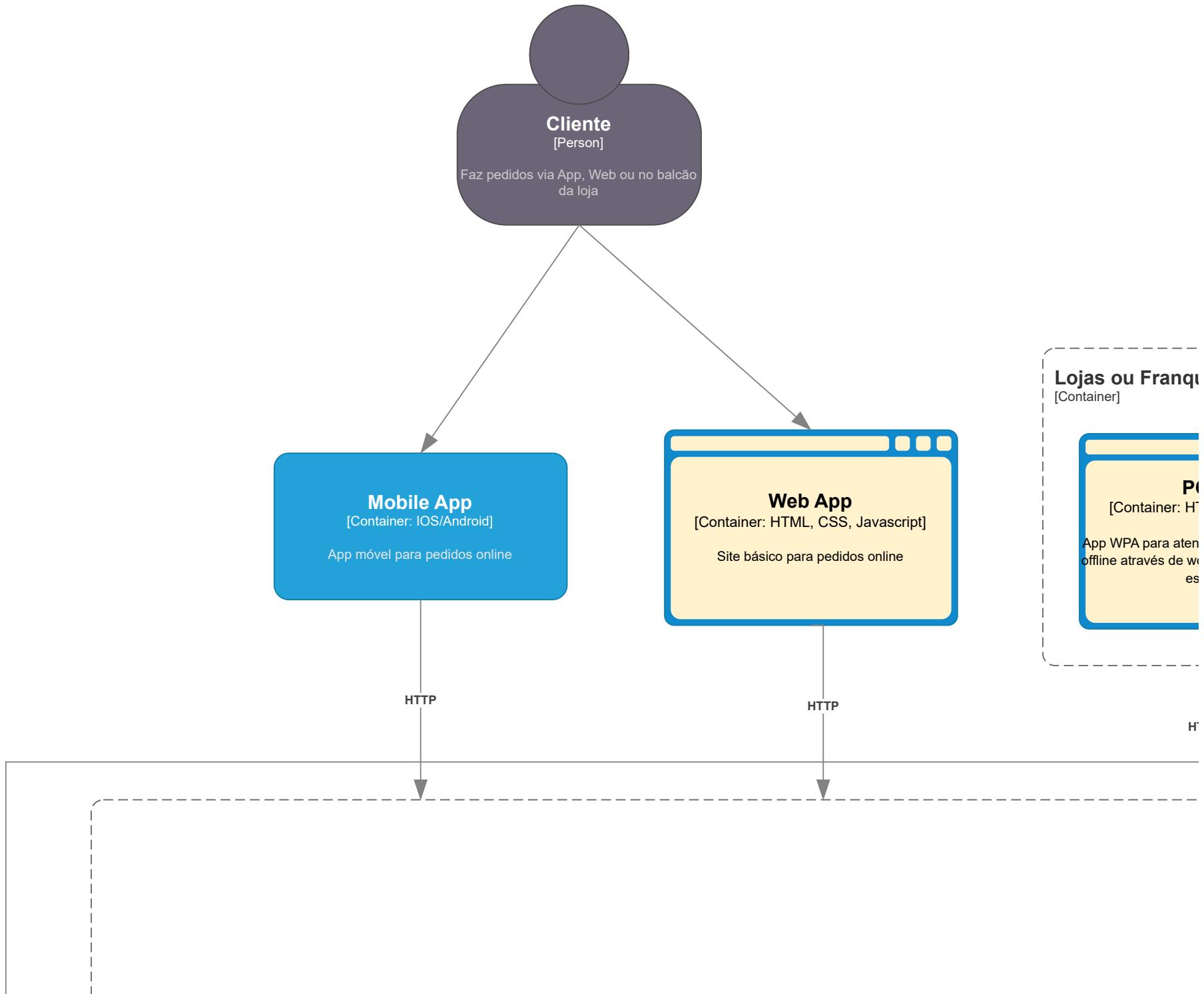


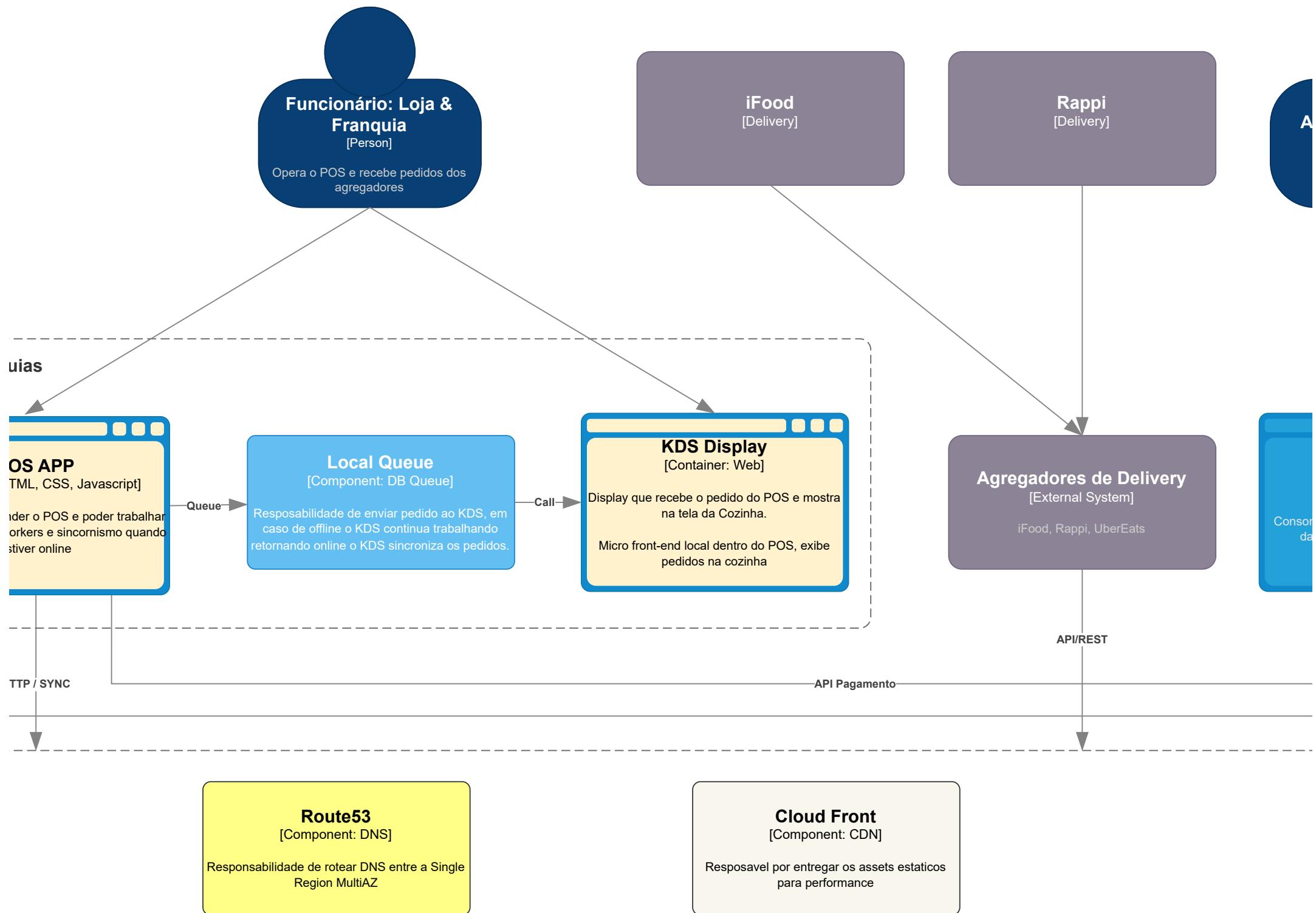




## Legenda

External Systems





Acesso Dashboard  
[Analytic Person]

Login Looker Studio

Looker Studio  
[Container: GCP]

me os dados do BigQuery para gerar dashboards tempo real e histórico

OAUTH2 / SAML

Pagamentos  
[Container]

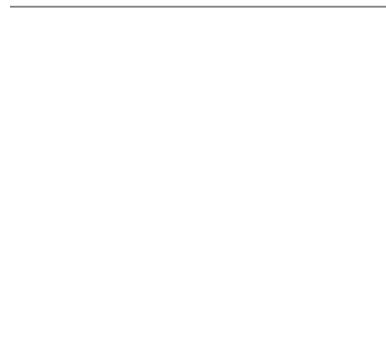
Gateway de Pagamento  
[System]

Entidade externa responsável por processar os pagamentos

REST

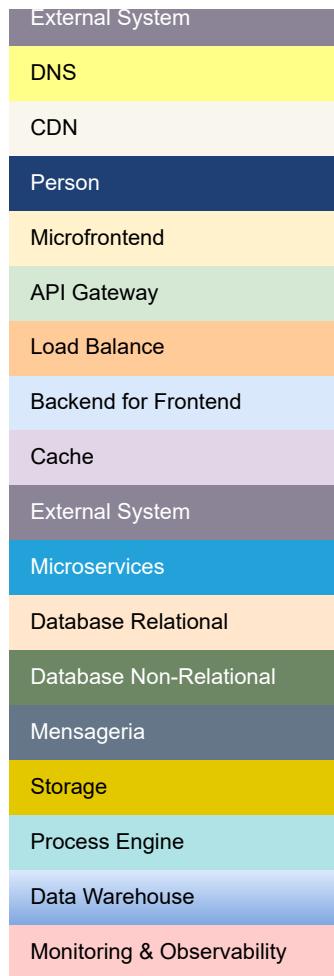
DNS & CDN

---





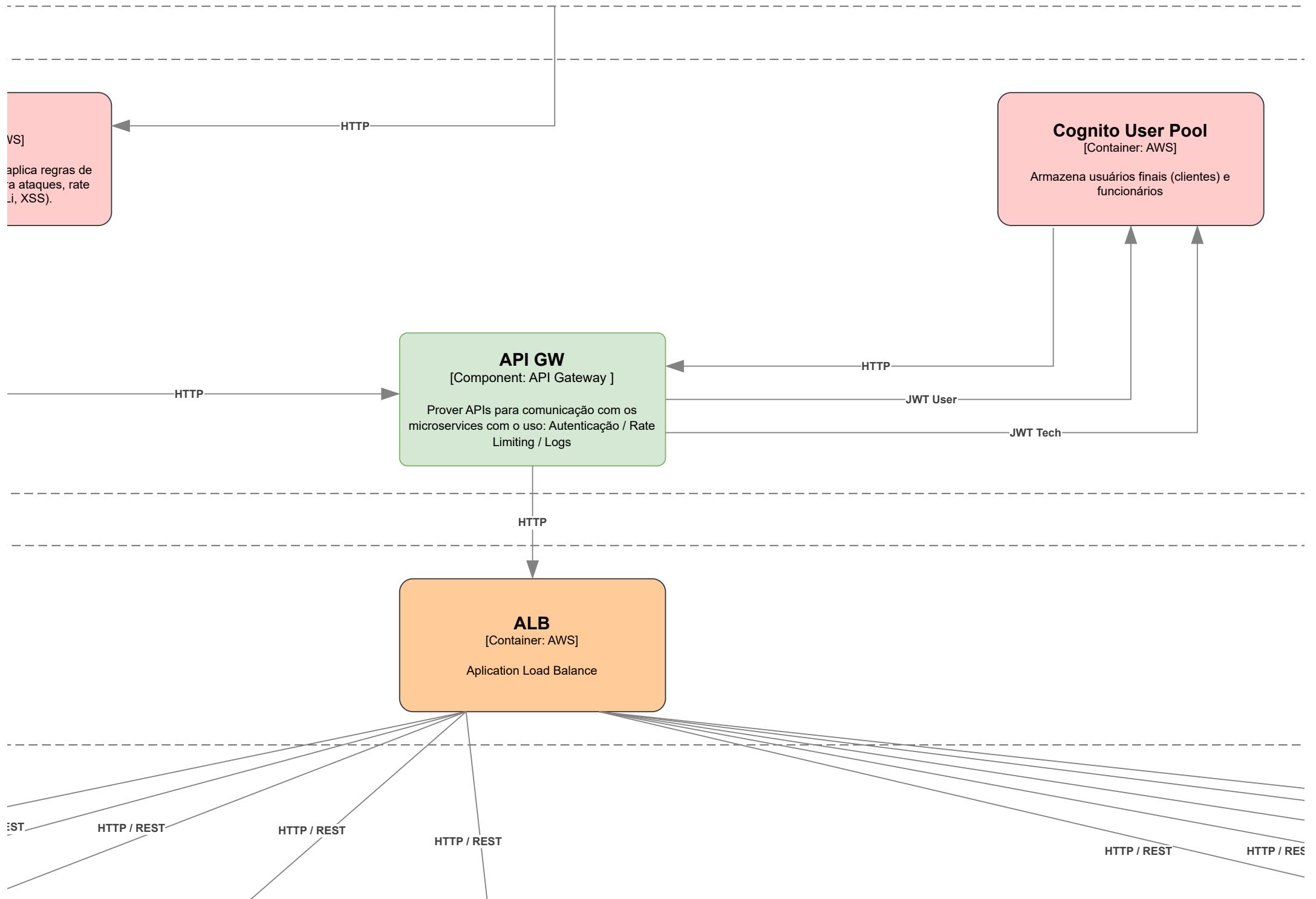




**WAF**

[Container: AW]

WAF intercepta o tráfego, aplicando regras de segurança (proteção contra DDoS, rate limiting básico, SQL injection, etc.).



[Bounded Context]

**Security**  
[Bounded Context]

**Load Balance**  
[Bounded Context]

ST      HTTP / REST

HTTP / REST

HTTP / REST



## Itens que necessitam atenção:

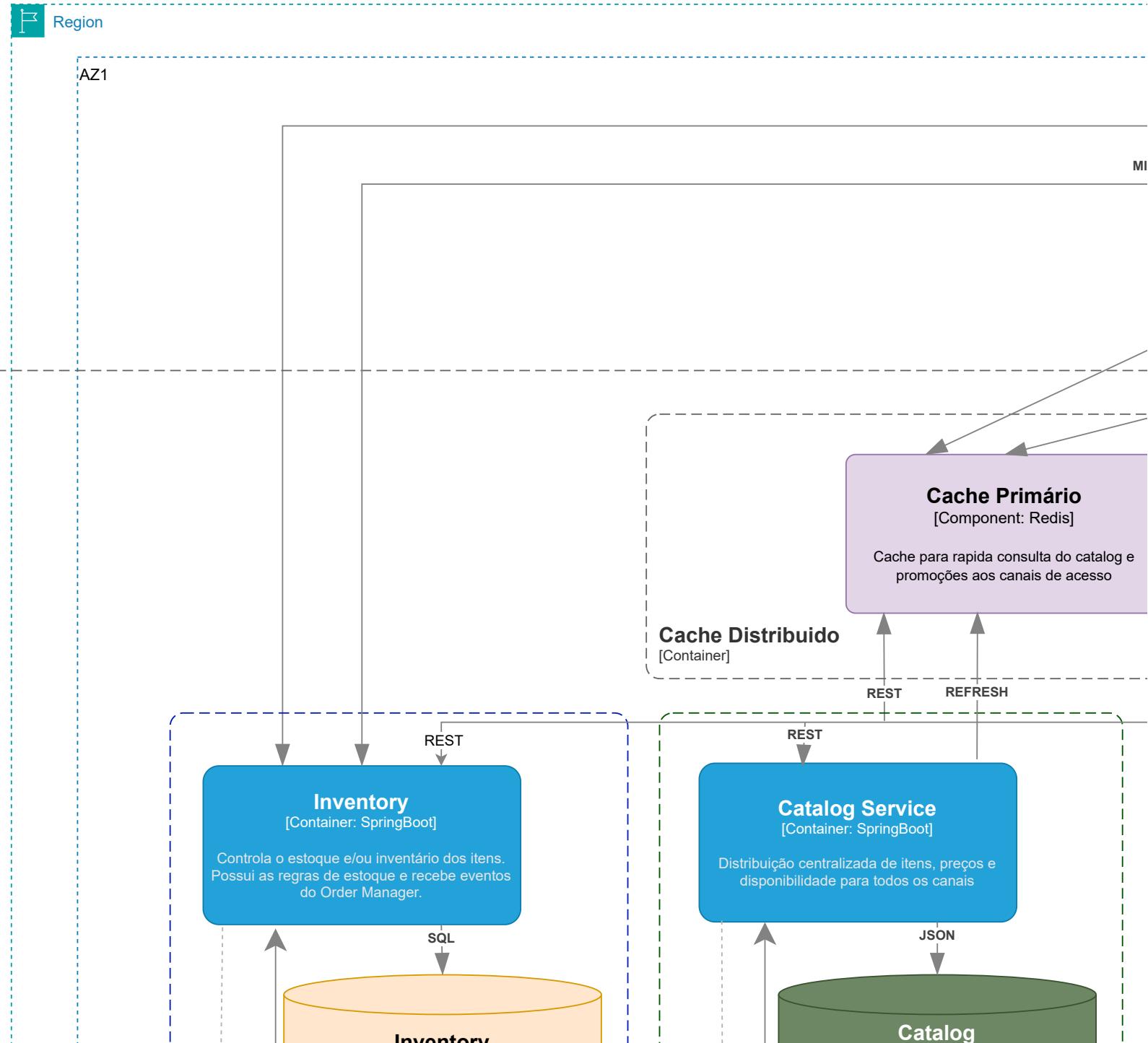
- Arquitetura dos containers do EKS, 1 cluster multi-AZ com nodes distribuidos, em caso de falha na para rodar em multiplas AZs.
- Arquitetura de Microfrontends em containers dentro do cluster EKS Multi-AZ.
- Arquitetura do MSK - Kafka, multi AZ com brokers redundantes. Kafka replica entre as AZs para tolerar falhas.
- Arquitetura do auroraDB Cluster com distribuição atomatica em multi AZs.
- Arquitetura do MongoDB demonstrando Replica Set que é composto por Primary e Secondaries.
- Arquitetura das camadas de Observabilidade: Logs, métricas, eventos, dashboard e alarmes através de integradores.
- Arquitetura de conexão multi nuvem AWS e GCP camadas de segurança dos dados em transito e em repouso.
- Arquitetura separada para Carrinho e o Checkout, documentar bem as responsabilidades de ambos.
- Representar o padrão Saga para o Checkout, com suporte a compensação e rollback para falhas.
- Pensar em um cache para manter o estado do carrinho enquanto o cliente navega.
- Representar o Financeiro/Contabilidade/Reconciliação.
- Pensar no Fiscal (Tinha um ERP).
- Representar Compras e Fornecedores e conexão com o Estoque.
- Representar a convivencia com o ERP e outro sistema legado.
- Falta console de Administração para catalogo, promoções e estoque.

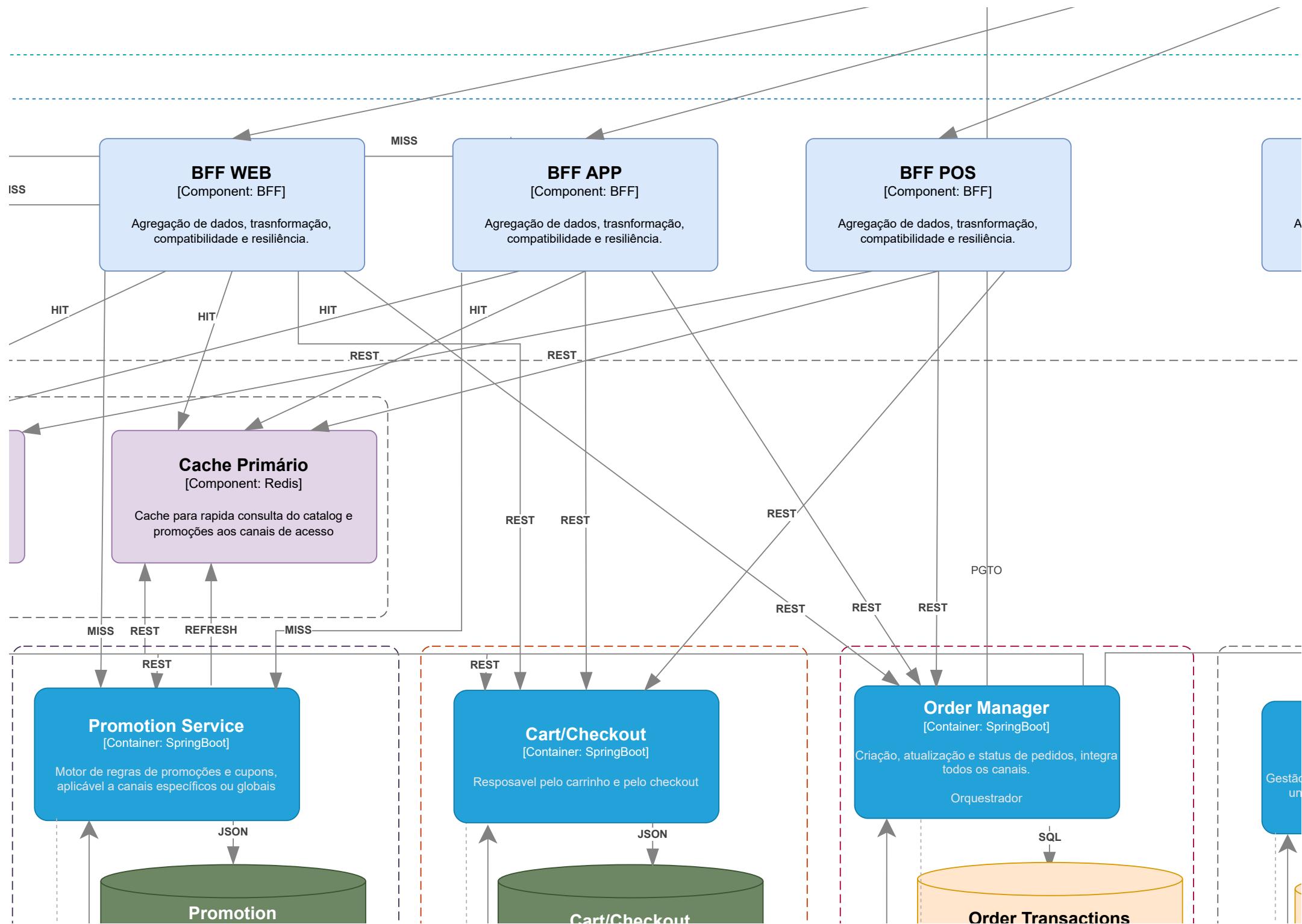
AZ-a EKS recia na AZ-b, com Auto Scaling Group

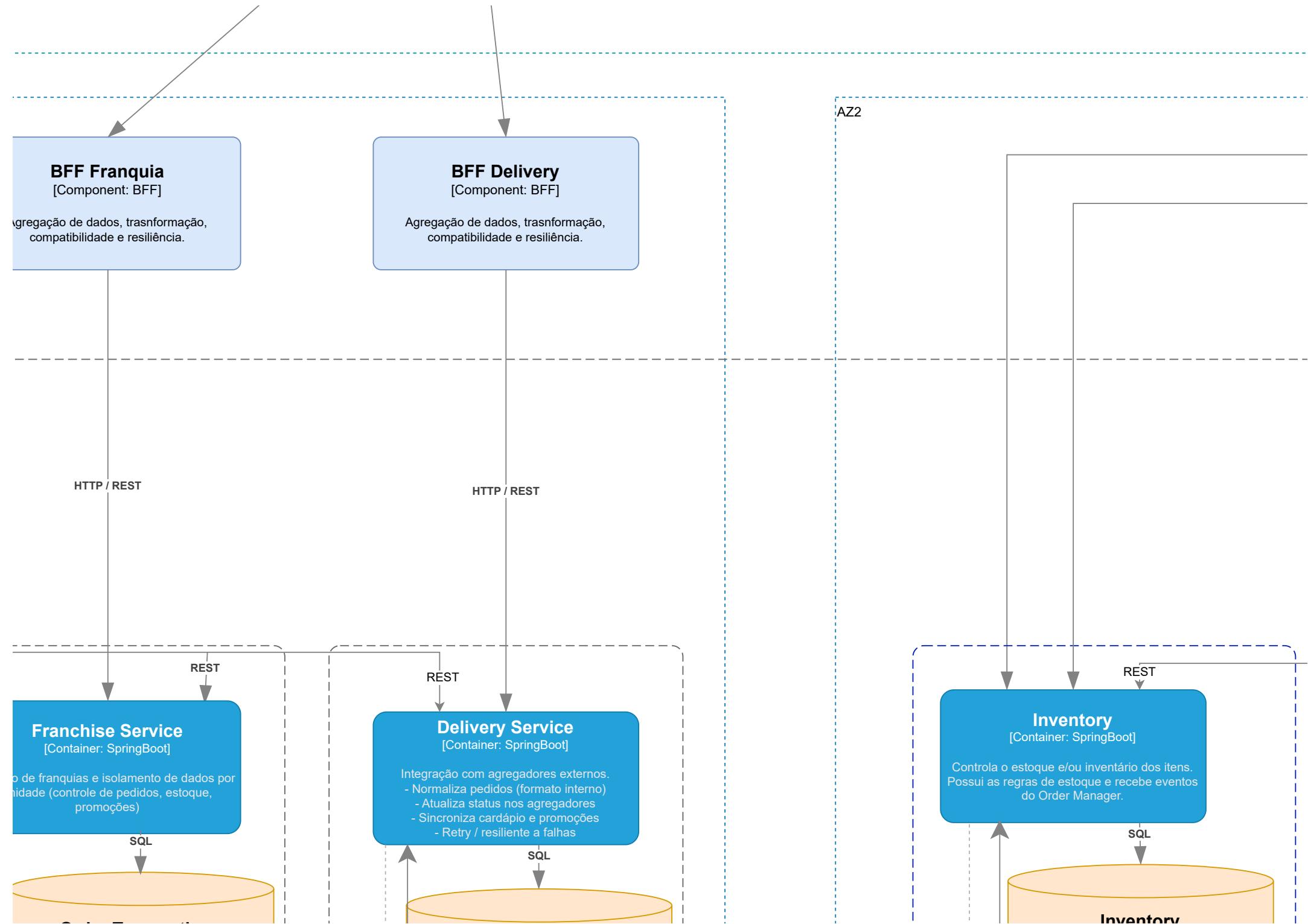
erancia a falha. Idempotência e Particionamento.

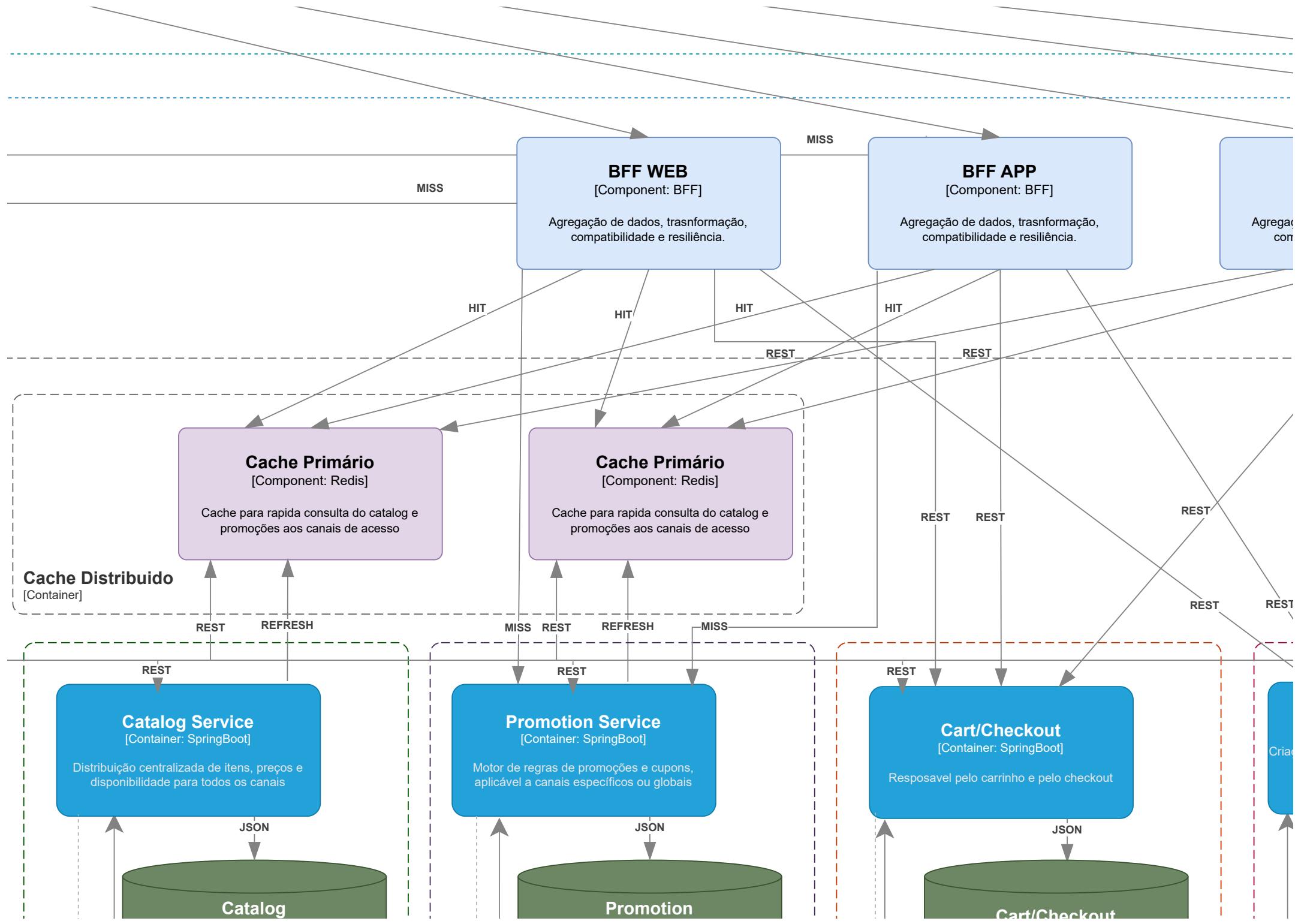
és do CloudWatch.  
descanso.

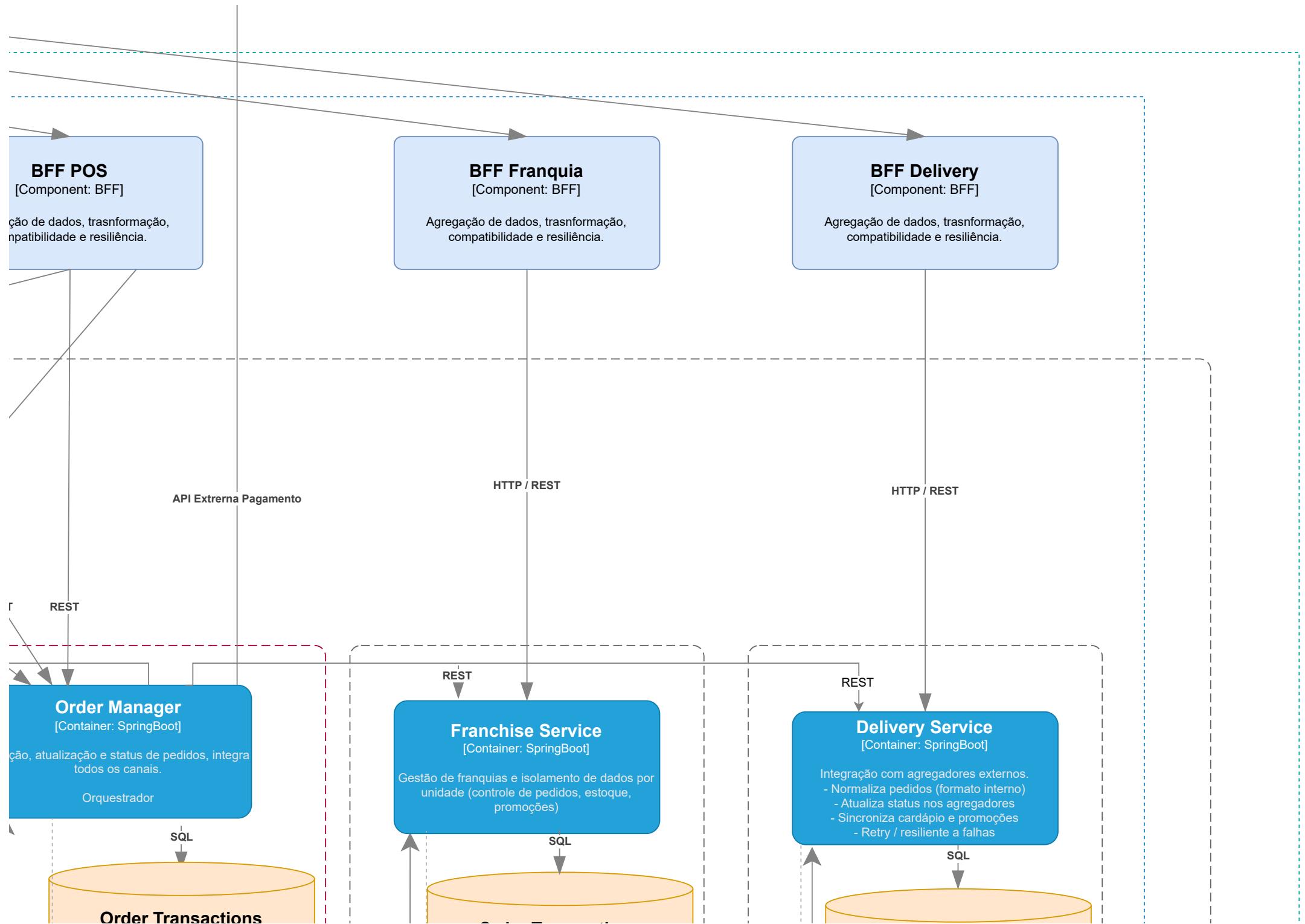
.











- Arquitetura da aplicação POS e KDS, em WPA utilizando service workers para caching e filas offline.
- Fluxo de cache, invalidação, atualização.
- Arquitetura de Prevenção a Fraude, conectado ao OrderManager.
- Detalhar a separação dos Tenants ex: lojald, franqueadold. AuroraDB e MongoDB com tenantId tabela.
- Portal de APIs, para documentações Swagger, exemplos e facilitar o onboarding dos parceiros ex.:...

e (React).

elas e coleções.  
: Delivery.

**Inventory**  
[Container: Aurora DB - Primary]

DB de inventario/estoque

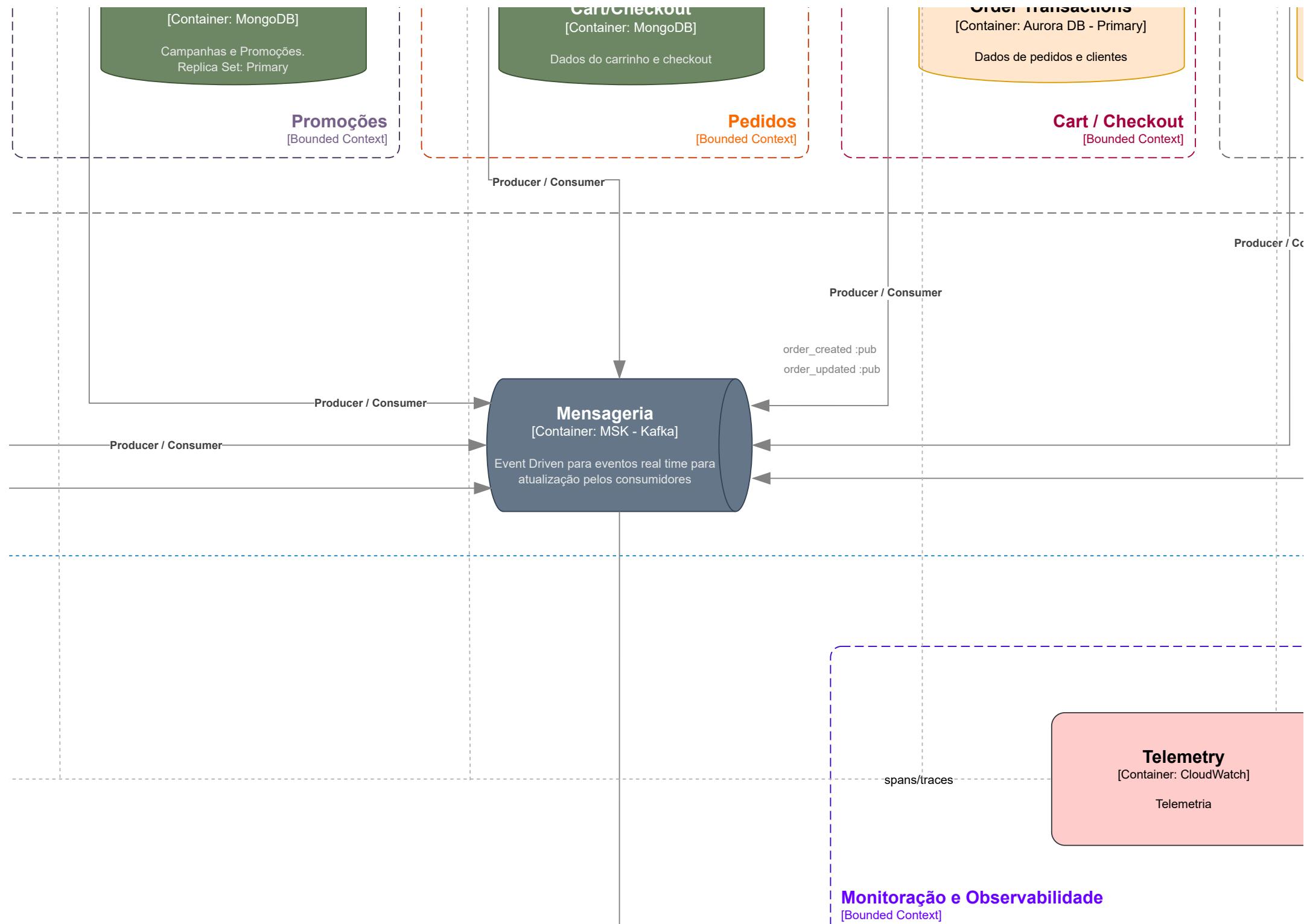
**Estoque**  
[Bounded Context]

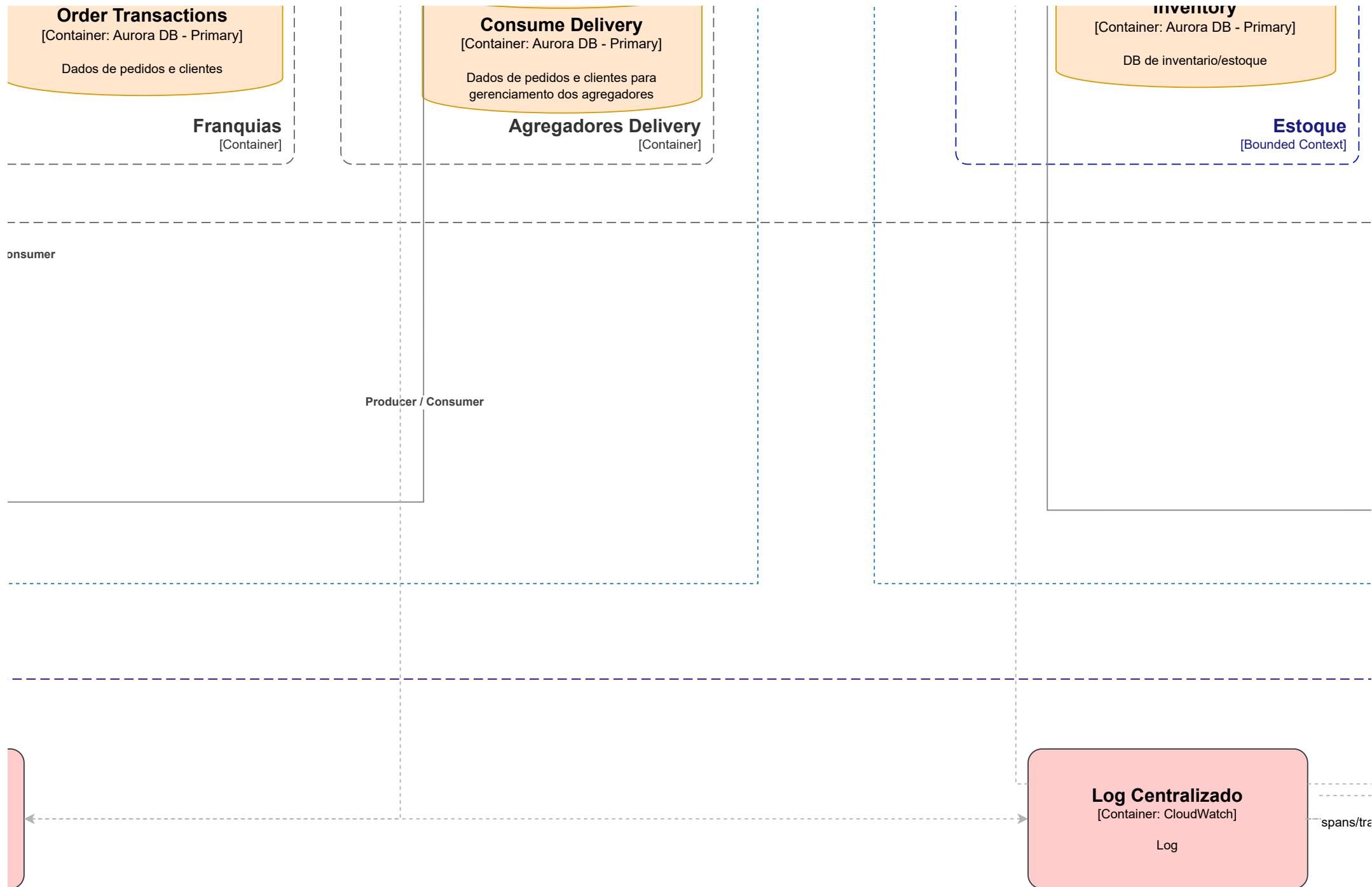
[Container: MongoDB]  
Itens do Menu podendo ser SKUs.  
Replica Set: Primary

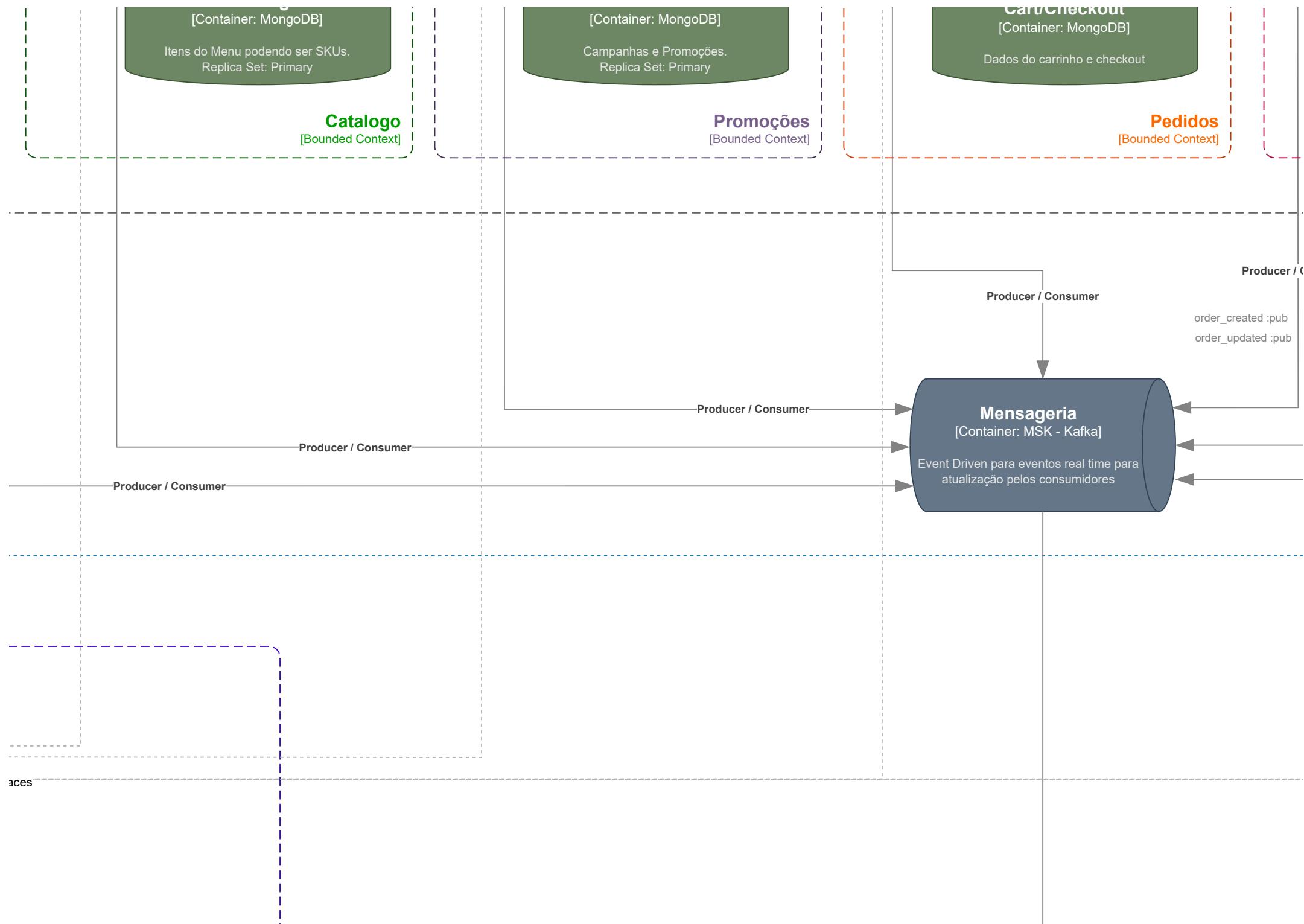
**Catalogo**  
[Bounded Context]

sub: order\_created

Producer / Consumer







**Order Transactions**  
[Container: Aurora DB - Primary]

Dados de pedidos e clientes

**Cart / Checkout**  
[Bounded Context]

**Order Transactions**  
[Container: Aurora DB - Primary]

Dados de pedidos e clientes

**Franquias**  
[Container]

**Consume Delivery**  
[Container: Aurora DB - Primary]

Dados de pedidos e clientes para gerenciamento dos agregadores

**Agregadores Delivery**  
[Container]

Consumer

Producer / Consumer

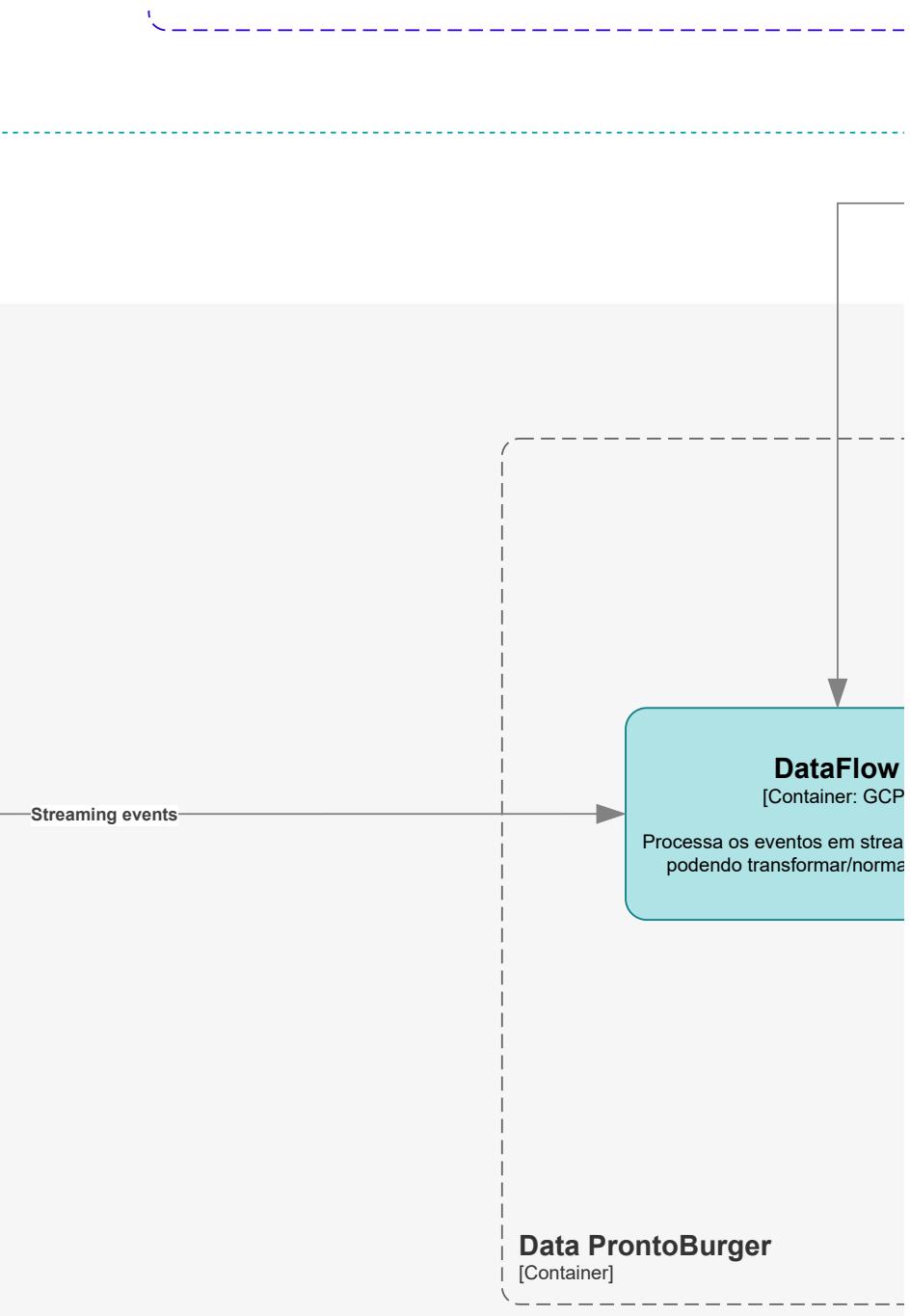
Producer / Consumer

Producer

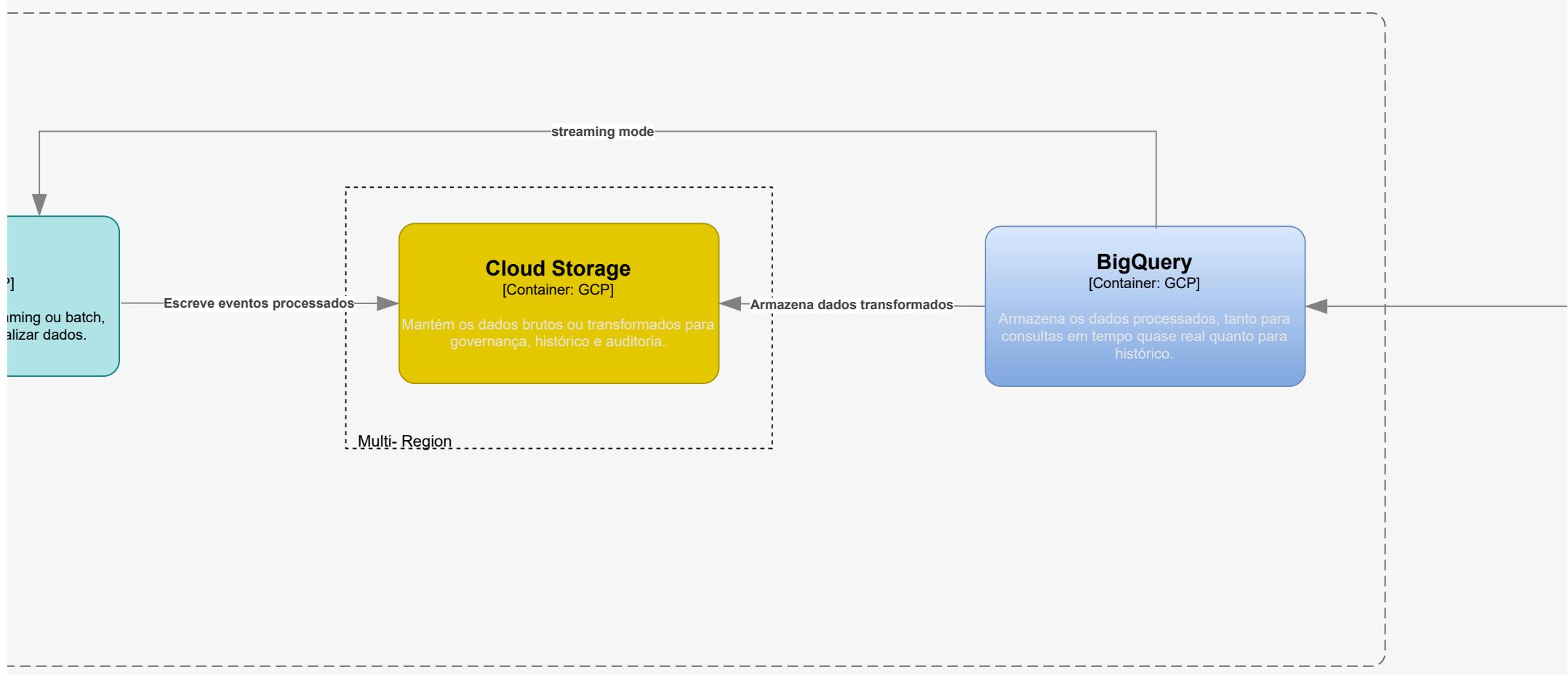




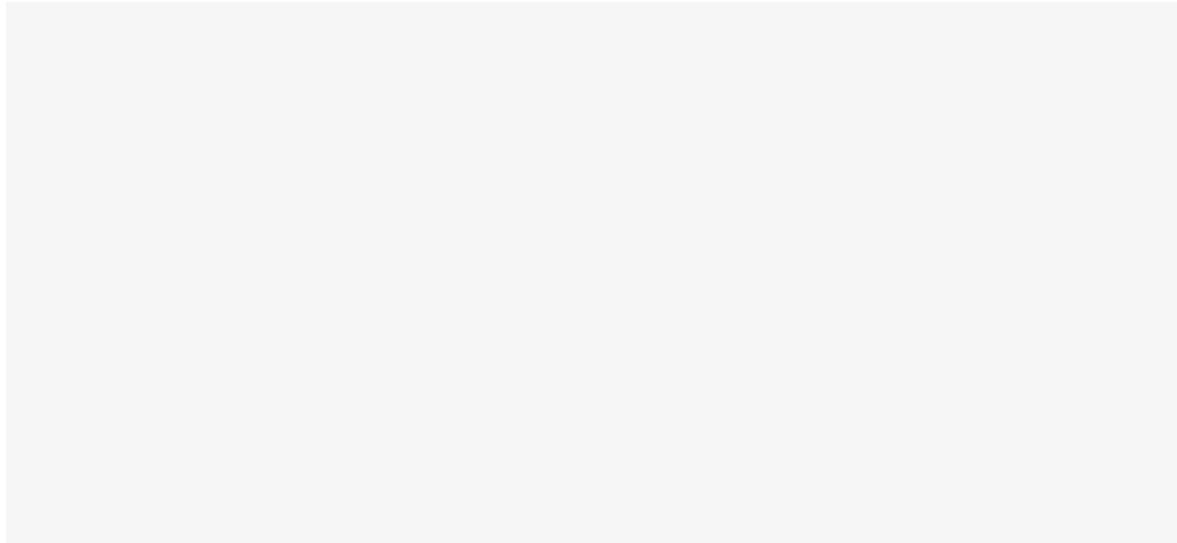




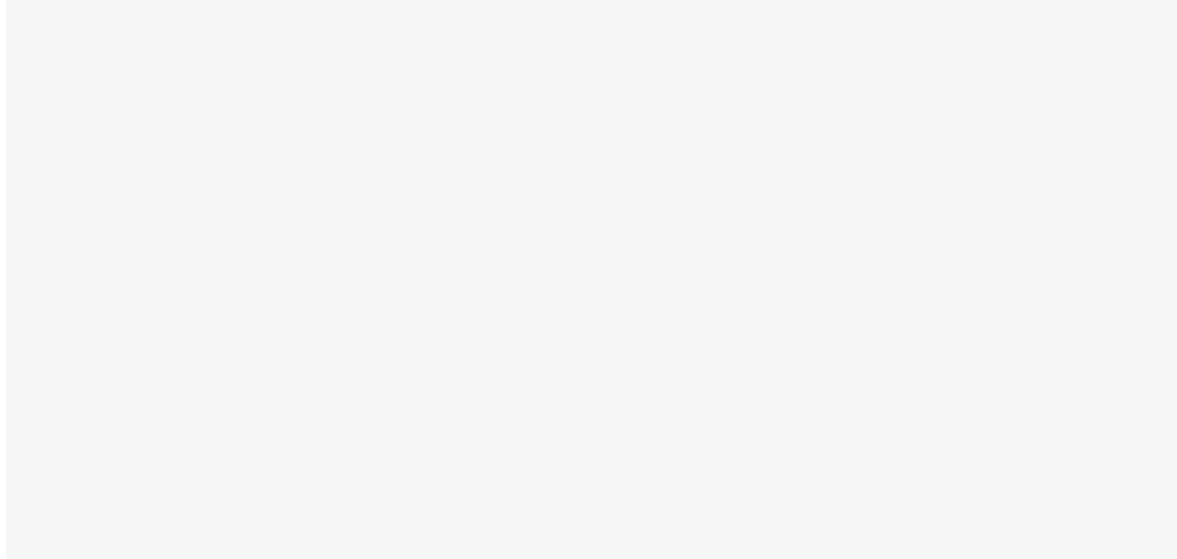
Streaming events

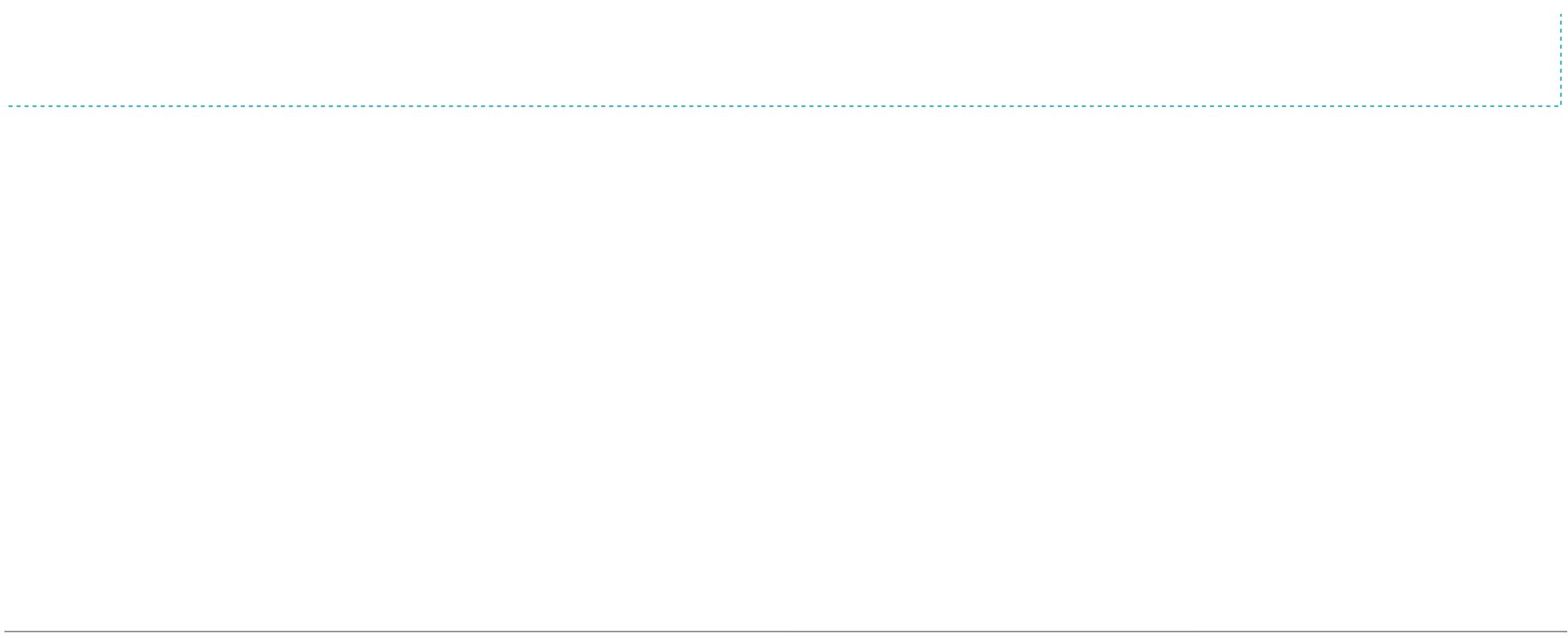


-----)



-----

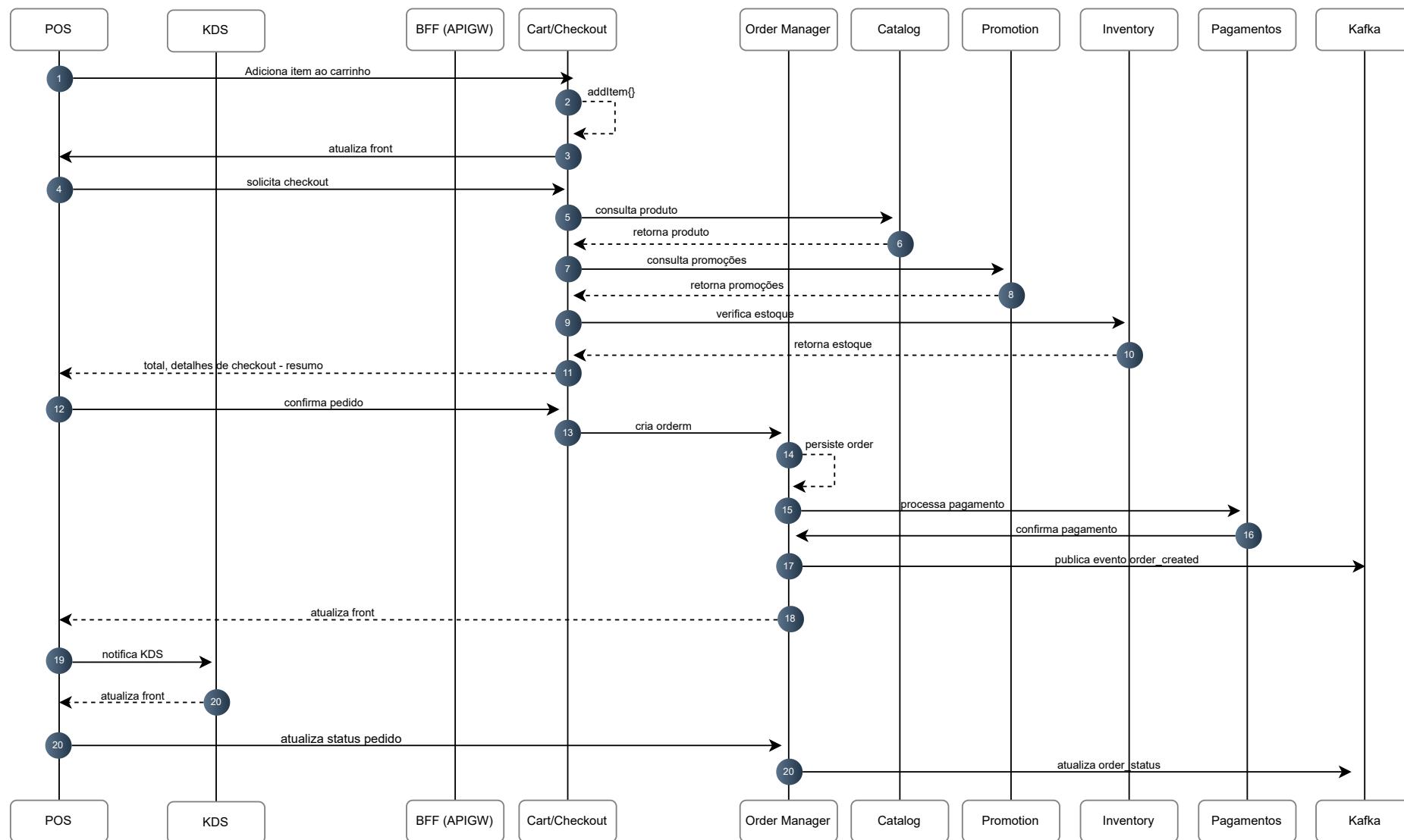








# Fluxo realizar Pedido no POS online



# ADR 001 — Escolha de Mensageria: MSK (Kafka) vs SNS/SQS

**Autor(es):** Arquiteto ProntoBurger

**Revisor(es):** Arquitetos e Engenheiros

**Status:** Avaliação

**Data:** 21/09/2025

## Contexto

A plataforma da ProntoBurger precisa propagar eventos de pedidos e outros domínios em tempo real. Esses eventos são consumidos por múltiplos serviços.

A decisão deve garantir baixa latência, alto throughput e durabilidade, além de permitir reprocessamento de eventos.

## Objetivo

Escolher a tecnologia de mensageria que atenda à escalabilidade de até 1000 pedidos/min, suporte múltiplos consumidores independentes, retenção de eventos e integração com ecossistema de dados ex.: streaming analytics.

## Decisão

Optamos por Amazon MSK - Managed Kafka como broker de eventos.

· Justificativa:

- o **Escalabilidade horizontal** via partitions e brokers, suportando picos de tráfego maiores que SNS/SQS.
- o **Modelo pub/sub com retenção configurável**, permitindo reprocessamento não suportado de forma nativa pelo SNS/SQS.
- o **Baixa latência** e throughput elevado, essencial para experiência omnichannel em tempo real. [Link](#)
- o **Integração ampla** com ferramentas de stream processing (Spark, BigQuery, Dataflow, BigTable).

· SNS/SQS foram descartados porque:

- o SNS não armazena eventos para reprocessamento. [Link](#)
- o SQS é adequado a filas ponto-a-ponto, mas não ao padrão fan-out com múltiplos consumidores independentes.

# ADR 002 — Estratégia de Persistência: SQL vs NoSQL

**Autor(es):** Arquiteto ProntoBurger  
**Revisor(es):** Arquitetos e Engenheiros

**Status: Avaliação**

**Data:** 21/09/2025

## Contexto

A plataforma ProntoBurger possui múltiplos microsserviços com diferentes necessidades de persistência:

- Cart/Checkout, Catalog e Promotion: mantém carrinhos/estado temporário, sessões de usuários e itens selecionados. E realiza cálculo de total, consulta estoque, catálogo e promoções, sem persistir pedidos.
- Order Manager: cria, atualiza e consulta pedidos confirmados, integrando pagamento, estoque, promoções e franquias.

Enquanto Cart/Checkout, Promotion e Catalog lidam com dados de curta duração e alta leitura/escrita rápida, o Order Manager precisa garantir **consistência forte e transações confiáveis**.

## Objetivo

Garantir persistência adequada de dados para cada serviço:

- Cart/Checkout, Promotion e Catalog: rápido acesso, flexibilidade de schema, escalabilidade horizontal.
- Order Manager: consistência, integridade transacional, consultas analíticas e relatórios gerenciais.

## Decisão

Adotamos persistência híbrida:

1. **MongoDB** para microsserviços de Cart /Checkout, Promotion e Catalog
  - o Justificativa:
    - . Altaperformance para leitura/escrita e dados temporários.
    - . Flexibilidade de schema para promoções, variações de itens e cálculos dinâmicos.
    - . Possui Replica Set primaria e secundária para alta disponibilidade.
2. **AuroraDB** para Order Manager
  - o Justificativa:
    - . Garantia de consistência transacional (ACID) para pedidos e pagamentos.
    - . Modelagem relacional adequada ao domínio de pedidos: clientes, itens, status, franquias.
    - . Suporte a escalabilidade e alta disponibilidade via read réplicas, multi-AZ e failover automático.

# Architecture Overview – Plataforma ProntoBurger

---

## 1. Visão Geral

A nova plataforma ProntoBurger será orientada a eventos, escalável e resiliente, com arquitetura de microsserviços, BFFs - Backend for Frontend e MFEs - Microfrontend.

Ela centraliza promoções e cardápio, unifica a jornada do cliente em todos os canais e vem atender a expansão para 2.000 lojas nos próximos 3 anos.

## 2. Princípios Arquiteturais

- Event-Driven Architecture – eventos como fonte de verdade para integração.
- Separação de responsabilidades por domínios - Bounded Contexts.
- BFFs para cada canal - App, Web, POS, Delivery.
- API Gateway, unificar acessos, segurança governança e roteamento.
- Escalabilidade via containers e serviços gerenciados.
- Resiliência e operação offline-first nas lojas.
- Omnichannel identidade unica Cognito, backend unificado, eventos e API GW.
- Observabilidade: métricas, logs e tracing.

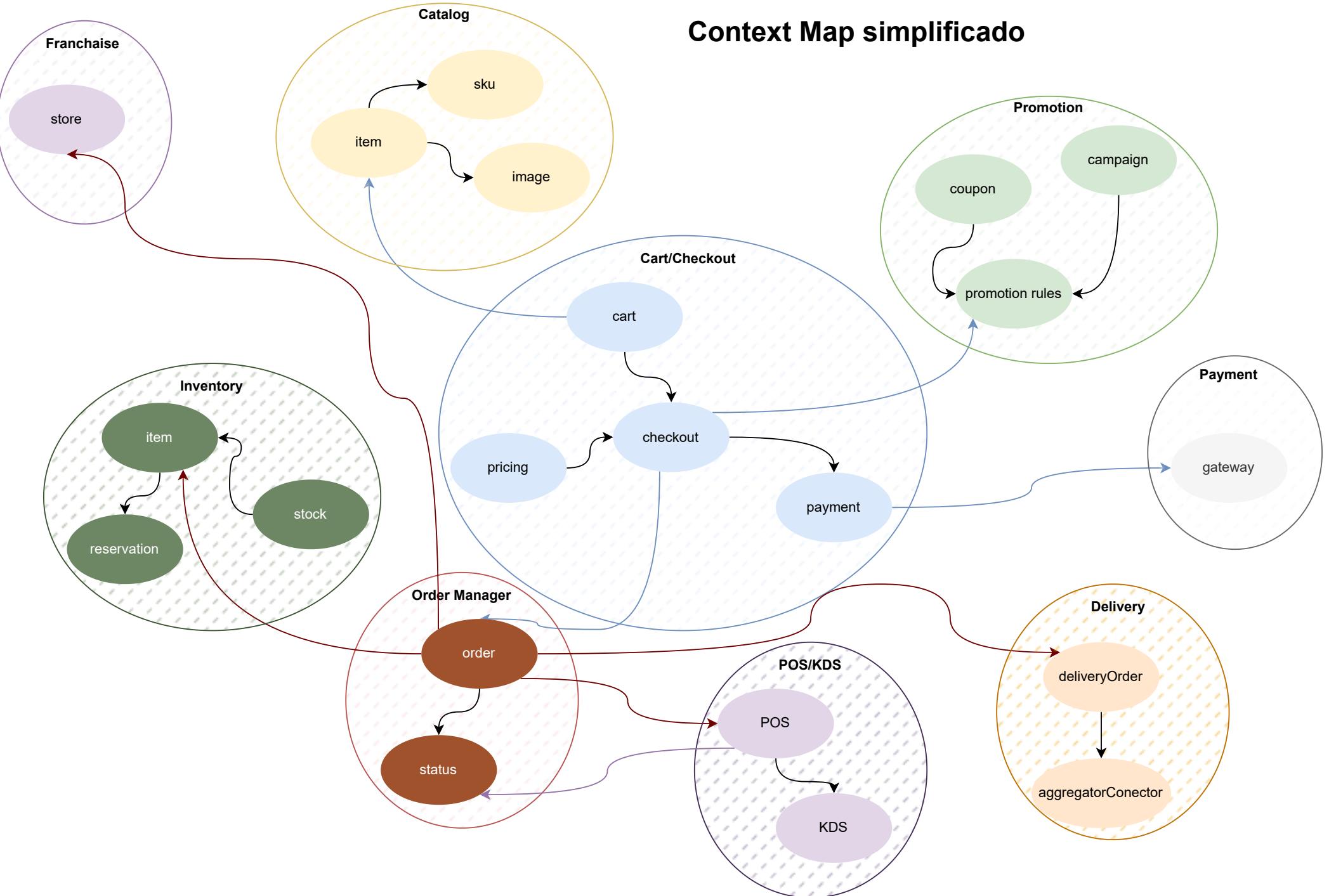
## 3. Justificativa de Principais Tecnologias

- Cloud Provider: AWS em Single Region em Multi-AZ e GCP para dados.
- Arquitetura híbrida: coreografia via enventos MSK e Cart/Checkout orquestra fluxos críticos.
- Mensageria: Amazon MSK, com alto throughput, fan-out, reprocessamento.
- Persistência: híbrida – AuroraDB e MongoDB com cache.
- API Gateway e BFFs – exposição por canal com autenticação e agregação.
- Observabilidade: Telemetria e CloudWatch.

#### 4. Requisitos Não-Funcionais

- Alta Disponibilidade: SLA de 99.95% com multi-AZ e autoscaling. [Link](#)
- Escalabilidade: suportar até 1000 pedidos/minuto em horários de pico.
- Baixa latência: APIs críticas abaixo de 200ms com cache e BFFs.
- Segurança: autenticação de usuários com Cognito e consumo de APIs.

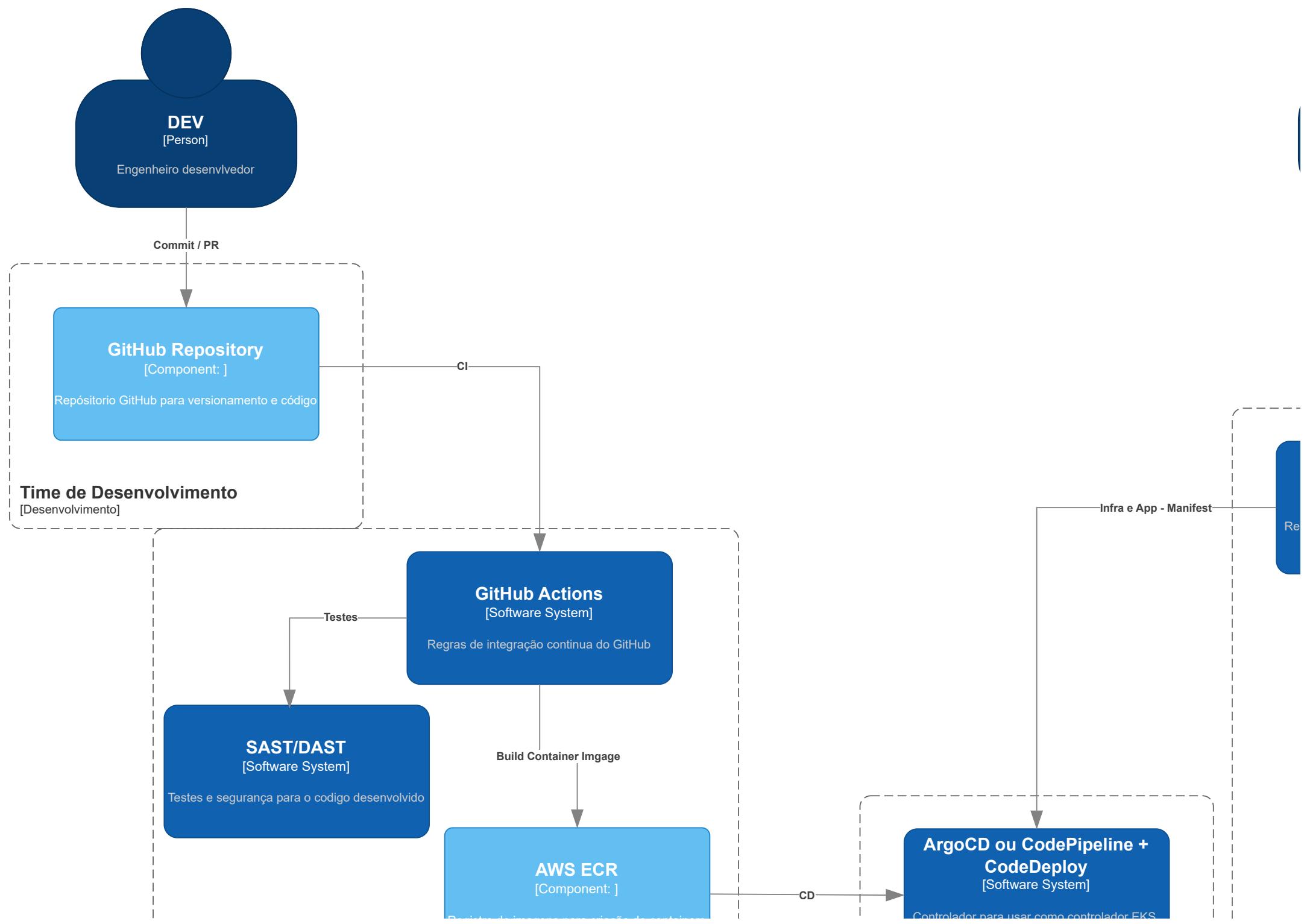
# Context Map simplificado



Este **Context Map** mostra os principais domínios da nova plataforma ProntoBurger e como eles se relacionam.

- Cada círculo representa um **Bounded Context**, com suas entidades centrais.
- As **setas** indicam dependências e trocas de informação.

O mapa reforça o princípio de **separação por domínios**, permitindo escalabilidade, baixo acoplamento e clareza na evolução do sistema.





Ops



responsible for provisioning infrastructure resources in AWS (\*.tf)

Infra

Registro de imagens para criação de containers

## Integração continua

[Integração continua]

Controlador para usar como controlador EKS.

Com Rollouts e técnicas de deploy Canary e

BlueGreen

Build Microservice

## EKS Cluster

[Software System]

Cluster EKS para provisão de  
pods/countainers

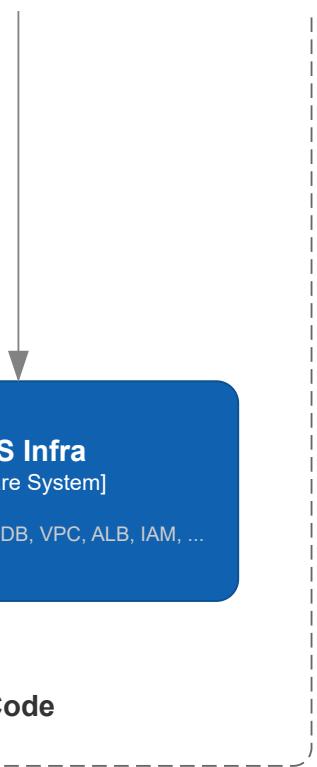
## GitOps

[GitOps para entrega continua]

## Infraes

[Container]

E



Strutura as Code

[

-----]