

Documentação Arquitetural - 12SOAT

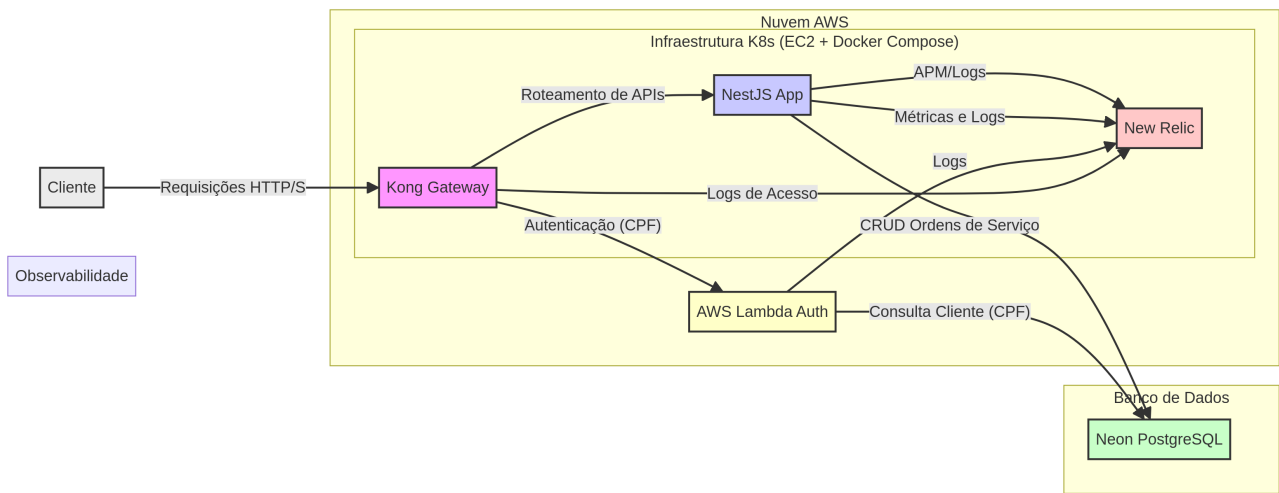
Fase 3 - Tech Challenge

Introdução

Este documento apresenta a documentação arquitetural completa do projeto 12SOAT - Fase 3, conforme os requisitos do Tech Challenge. A arquitetura foi concebida para garantir segurança, escalabilidade, alta disponibilidade e observabilidade, utilizando práticas de infraestrutura como código (IaC) e soluções serverless.

1. Diagrama de Componentes

O Diagrama de Componentes oferece uma visão de alto nível da arquitetura, abrangendo a nuvem (AWS), o API Gateway (Kong), o banco de dados (Neon PostgreSQL) e o monitoramento (New Relic).

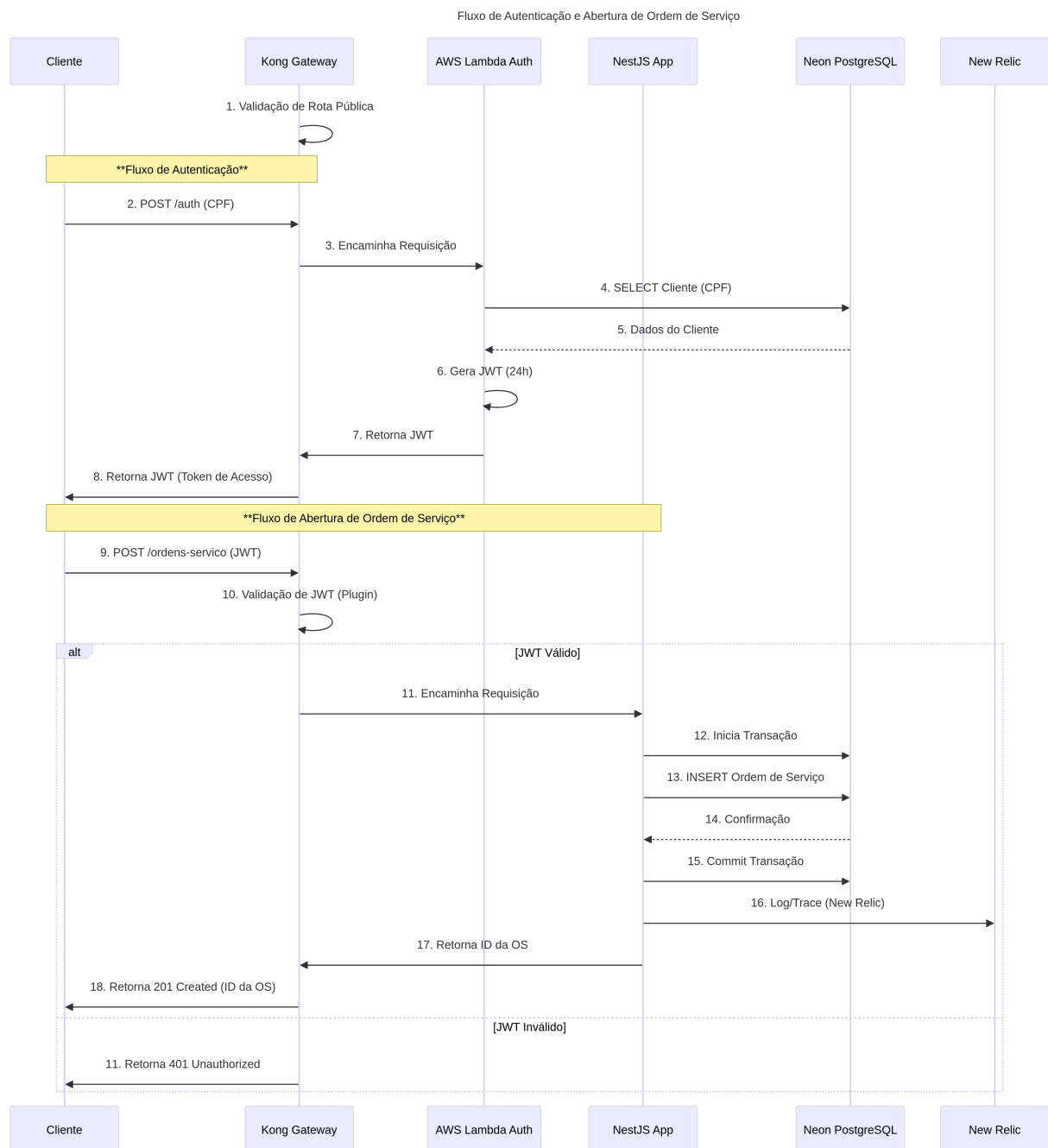


Componentes Chave:

Componente	Tecnologia	Função
Cliente	Browser/App	Inicia as requisições HTTP/S.
Kong Gateway	Kong	API Gateway. Roteia requisições e centraliza a validação de JWT.
AWS Lambda Auth	AWS Lambda (Node.js)	Função serverless para autenticação via CPF e geração de JWT.
NestJS App	NestJS (TypeScript)	Aplicação principal de gestão de Ordens de Serviço (OS).
Neon PostgreSQL	PostgreSQL Serverless	Banco de dados relacional para dados transacionais.
New Relic	APM/Observabilidade	Coleta métricas, logs e traces para monitoramento em tempo real.

2. Diagrama de Sequência

O Diagrama de Sequência detalha o fluxo de interação para as operações críticas de **Autenticação** e **Abertura de Ordens de Serviço**.



Fluxo de Autenticação (Passos 1-8)

1. O Cliente envia o CPF para o Kong Gateway (`POST /auth`).
2. O Kong roteia a requisição para a AWS Lambda Auth.
3. A Lambda consulta o `Neon PostgreSQL` para validar o cliente.
4. Em caso de sucesso, a Lambda gera um JWT e o retorna ao Kong.
5. O Kong retorna o JWT ao Cliente, que o utilizará nas requisições protegidas.

Fluxo de Abertura de OS (Passos 9-18)

1. O Cliente envia a requisição de criação de OS com o JWT no cabeçalho para o Kong (`POST /ordens-servico`).
 2. O Kong valida o JWT (Passo 10).
 3. Se o JWT for válido, o Kong roteia para a NestJS App.
 4. A NestJS App executa a lógica de negócio, interage com o Neon PostgreSQL (transação de INSERT) e envia logs/traces para o New Relic .
 5. A resposta de sucesso é retornada ao Cliente.
-

3. Request for Comments (RFCs)

RFC-001: Escolha da Plataforma Cloud (AWS)

Campo	Valor
Status	Aprovado
Decisão	Utilização da Amazon Web Services (AWS) como provedor de infraestrutura cloud.
Justificativa	Maior maturidade de mercado, ecossistema vasto, suporte robusto a Serverless (AWS Lambda) e integração completa com ferramentas de IaC (Terraform).

RFC-002: Escolha do Banco de Dados (Neon PostgreSQL)

Campo	Valor
Status	Aprovado
Decisão	Utilização do Neon PostgreSQL como banco de dados relacional.
Justificativa	Combina a robustez do PostgreSQL com a escalabilidade e otimização de custos do modelo serverless, atendendo ao requisito de Banco de Dados Gerenciado com escalabilidade .

RFC-003: Estratégia de Autenticação (Lambda + JWT + Kong Gateway)

Campo	Valor
Status	Aprovado
Decisão	Fluxo de autenticação desacoplado usando AWS Lambda, JWT e validação via Kong Gateway.
Justificativa	Atende ao requisito de Function Serverless para autenticação , isola a lógica de identidade da aplicação principal e centraliza a segurança no API Gateway (Kong).

4. Architecture Decision Records (ADRs)

ADR-001: Padrão de Comunicação (HTTP/REST Síncrono)

Campo	Valor
Status	Aceito
Decisão	Adotar o padrão de comunicação HTTP/REST Síncrono para todas as interações de front-end para back-end e entre o Kong Gateway e a aplicação NestJS.
Consequências	Positivas: Simplicidade, baixa latência para operações transacionais. Negativas: Acoplamento temporal, necessidade de padrão assíncrono futuro para operações de longa duração.

ADR-002: Estratégia de Escalabilidade (Horizontal Pod Autoscaler - HPA)

Campo	Valor
Status	Aceito
Decisão	Adotar o Horizontal Pod Autoscaler (HPA) do Kubernetes como a principal estratégia de escalabilidade horizontal para a aplicação NestJS e o Kong Gateway.
Consequências	Positivas: Escalabilidade automática, otimização de custos, alta disponibilidade. Negativas: Complexidade de infraestrutura (Kubernetes/EKS), dependência de métricas de monitoramento.

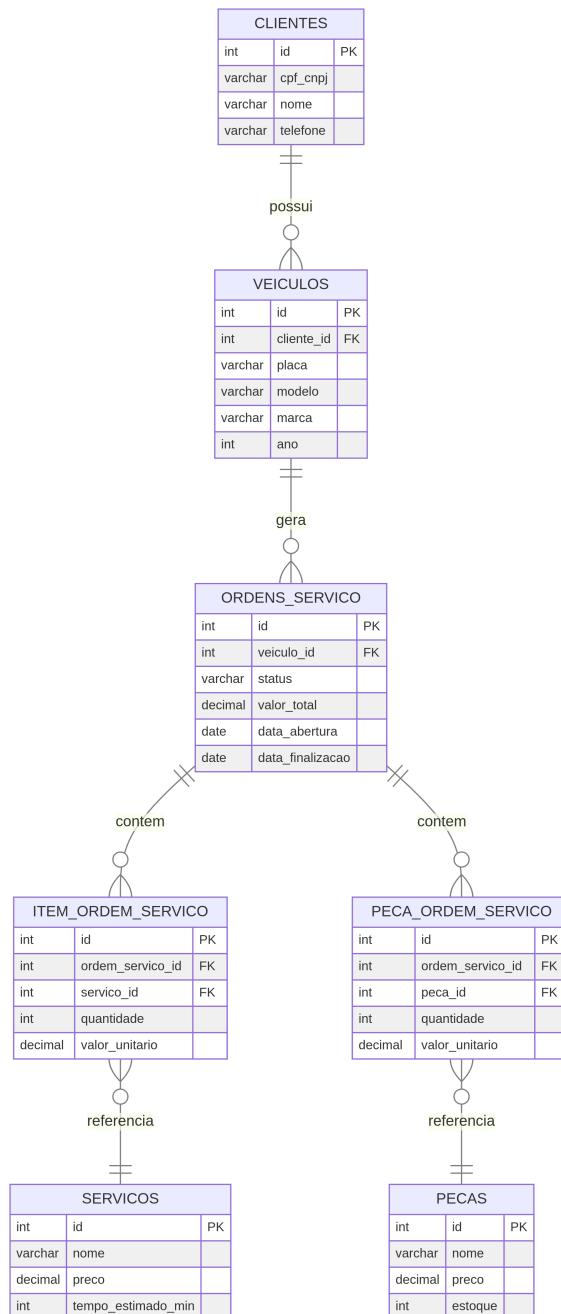
5. Justificativa Formal do Banco de Dados e Modelo Relacional

5.1. Escolha do Banco de Dados: Neon PostgreSQL

A escolha do Neon PostgreSQL é justificada pela sua capacidade de oferecer um SGBD relacional maduro (PostgreSQL) com a flexibilidade e o custo-benefício de uma arquitetura serverless. Isso garante a **integridade transacional** necessária para dados financeiros e de gestão, ao mesmo tempo que permite a **escalabilidade elástica** exigida pelo projeto.

5.2. Diagrama Entidade-Relacionamento (ER)

O modelo de dados é relacional e segue a Terceira Forma Normal (3NF), minimizando redundância e garantindo a integridade.



5.3. Ajustes e Relacionamentos Chave

O modelo utiliza tabelas de ligação (`ITEM_ORDEM_SERVICO` e `PECA_ORDEM_SERVICO`) para resolver o relacionamento **N:M** entre `ORDENS_SERVICO` e as entidades `SERVICOS` / `PECAS` .

Relacionamento	Tipo	Explicação
CLIENTES - VEICULOS	1:N	Um Cliente possui N Veículos.
VEICULOS - ORDENS_SERVICO	1:N	Um Veículo gera N Ordens de Serviço.
ORDENS_SERVICO - ITEM_ORDEM_SERVICO	1:N	Detalha os serviços realizados na OS.
ORDENS_SERVICO - PECA_ORDEM_SERVICO	1:N	Detalha as peças utilizadas na OS.
Ajuste Chave	Integridade Histórica	As tabelas de ligação armazenam <code>valor_unitario</code> e <code>quantidade</code> no momento da OS, preservando a integridade histórica do valor cobrado, mesmo que os preços mestres (<code>SERVICOS</code> e <code>PECAS</code>) sejam alterados posteriormente.
Integridade	Chaves Estrangeiras	Todas as tabelas de detalhe utilizam chaves estrangeiras para garantir a integridade referencial dos dados.