Documentação Case Canais

Ana Eliza Perobelli

Contexto

Este projeto foi desenvolvido como um **case técnico para Engenheira de Software Pleno**, com o objetivo de simular um sistema real de recebimento, processamento e persistência de reclamações provenientes de múltiplos canais.

O cenário proposto envolve dois tipos de entrada:

- **Canal físico**, onde documentos são digitalizados.
- **Canal externo**, onde reclamações são obtidas a partir de sistemas de terceiros.

O principal desafio é garantir que o sistema seja:

- desacoplado
- escalável
- resiliente
- observável

Por esse motivo, foi adotada uma **arquitetura hexagonal na aplicação**, combinada com uma **arquitetura orientada a eventos na integração entre os serviços**, utilizando serviços da AWS.

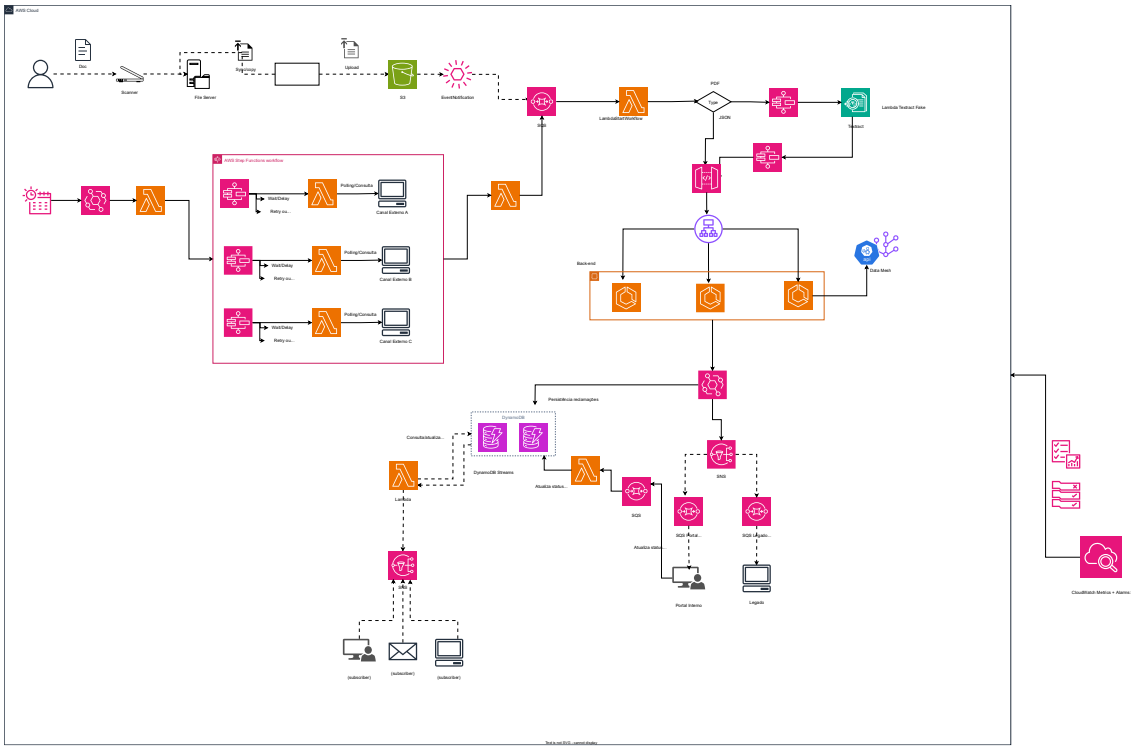
Ferramentas e tecnologias utilizadas

A arquitetura foi construída utilizando os seguintes serviços:

- **Amazon S3**: armazenamento dos documentos digitalizados.
- **Amazon EventBridge**: orquestração de eventos e agendamentos.
- **AWS Lambda**: execução de funções para coleta de dados externos.
- **Amazon SQS**: filas para processamento assíncrono.
- **.NET Worker**: consumidor das filas e responsável pelo processamento.

- **Amazon DynamoDB**: persistência das reclamações normalizadas.
- **Amazon SNS**: notificação de eventos (conceitual).
- **DLQ (Dead Letter Queue)**: tratamento de falhas.
- **Datamesh fake / Textract fake**: APIs mockadas para enriquecimento.

1. Fluxo projeto



No cenário atual do projeto, o monitoramento é realizado por meio de logs, métricas com Amazon CloudWatch e Dead Letter Queue (DLQ).

Como evolução da solução, o monitoramento poderia ser expandido utilizando:

- Alarmes automáticos configurados para disparar em caso de erros acima de um determinado percentual ou tempo de processamento superior ao esperado;
- SNS (Simple Notification Service) para envio de alertas dos alarmes do CloudWatch, integrando notificações via e-mail ou ferramentas de gestão de incidentes.

Quanto às estratégias já aplicadas para evitar gargalos, destacam-se:

- Processamento assíncrono com uso de SQS e EventBridge;
- Desacoplamento entre produtores e consumidores, permitindo filas separadas por canal;
- Isolamento de carga entre canais físicos e externos, evitando que um canal impacte o outro.

Como evolução da arquitetura, o processamento poderia ser executado em múltiplas instâncias ECS (containers) ou via Fargate Launch Type, eliminando a necessidade de gerenciar infraestrutura subjacente. Isso permite escalabilidade horizontal automática, acompanhando o aumento do volume de mensagens de forma eficiente.

2. Arquitetura utilizada

Para o projeto atual, recomendaria a utilização da **Arquitetura Hexagonal** (ou Ports & Adapters), que se alinha bem com princípios de **Clean Architecture**.

Motivos da recomendação:

- **Desacoplamento**: A arquitetura hexagonal separa o núcleo da aplicação (domínio e regras de negócio) das dependências externas (bancos de dados, filas, serviços externos), facilitando a **manutenção e testes**.
- **Flexibilidade**: Permite substituir adaptadores externos sem impactar o domínio, por exemplo, trocar SQS por outro sistema de mensageria ou EC2 por Fargate, sem mudanças significativas nas regras de negócio.
- **Escalabilidade**: Facilita a criação de múltiplos workers, microsserviços ou integrações com novos canais, mantendo isolamento entre produtores e consumidores.
- **Testabilidade**: Com as regras de negócio isoladas do lado externo, é possível escrever testes unitários e de integração de forma independente da infraestrutura.

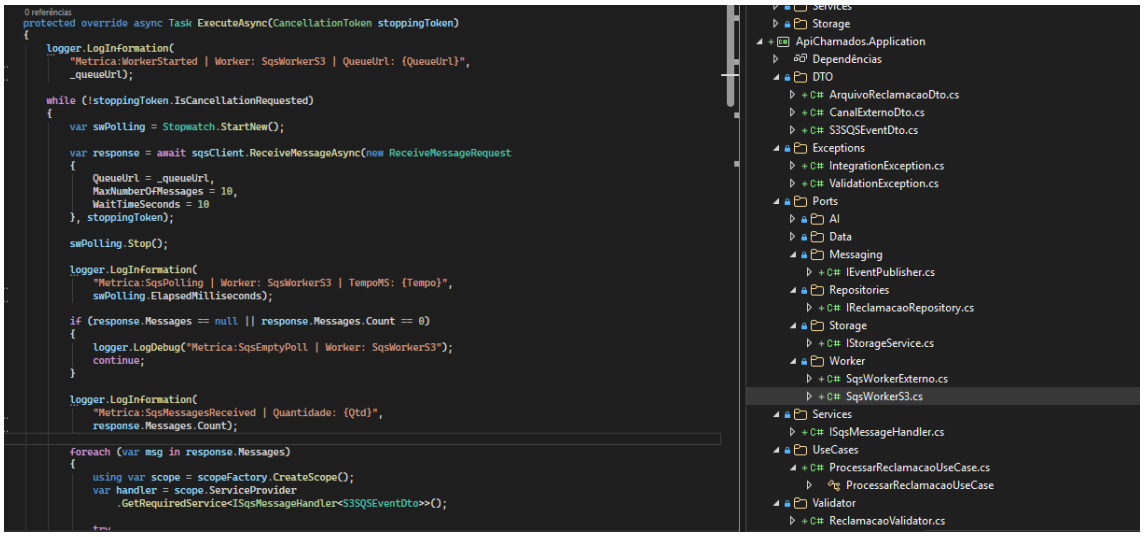
Como se aplicaria no projeto:

- **Camadas internas (Domínio / Application)**: contêm os **Use Cases** e entidades, com toda a lógica de negócio centralizada.
- **Ports (Interfaces)**: definem contratos para comunicação com o mundo externo (SQS, EventBridge, S3, APIs externas).
- **Adapters / Infraestrutura**: implementam essas interfaces, lidando com detalhes de mensageria, persistência e serviços externos.

3. **Estrutura de código e divisão de camadas**

ApiChamados.Application

- **DTO** (Data Transfer Objects): Objetos que padronizam a entrada de dados (ex: CanalExternoDto.cs), isolando as entidades internas do mundo externo.
- **Ports** (Portas): Interfaces (ex: IEventPublisher.cs) que definem o que o sistema precisa fazer, mantendo a aplicação independente de tecnologias como AWS ou SQL.
- **UseCases** (Casos de Uso): O "cérebro" do fluxo (ex: ProcessarReclamacaoUseCase.cs). Ele recebe as interfaces e dita a ordem das operações: extração via IA, validação e publicação de eventos.
- **Worker**: Componentes que monitoram gatilhos externos, como filas SQS (SqsWorkerS3.cs), para iniciar o processamento da aplicação.
- **Validator**: Camada de integridade que utiliza o ReclamacaoValidator.cs para garantir que apenas dados válidos avancem no fluxo.
- **Exceptions**: Centraliza erros customizados (ex: ValidationException.cs), permitindo um tratamento padronizado de falhas de negócio.



4. **Linguagens e bancos de dados recomendados**

Linguagens

.NET (C#) ou **Java (Spring Boot)** para o backend principal, por serem linguagens maduras, amplamente utilizadas em ambientes corporativos e com excelente suporte a processamento assíncrono, mensageria e observabilidade.

Python ou **Node.js** para funções serverless (Lambdas), devido à simplicidade, rapidez de desenvolvimento e boa integração com serviços externos.

Bancos de dados

- **DynamoDB (NoSQL)** para a camada operacional, indicado para alto volume de escrita, baixa latência e escalabilidade automática.
- **PostgreSQL / Aurora (SQL)** para cenários que exigem consultas complexas, relatórios e consistência relacional.

5. **Recomendações IA para o projeto**

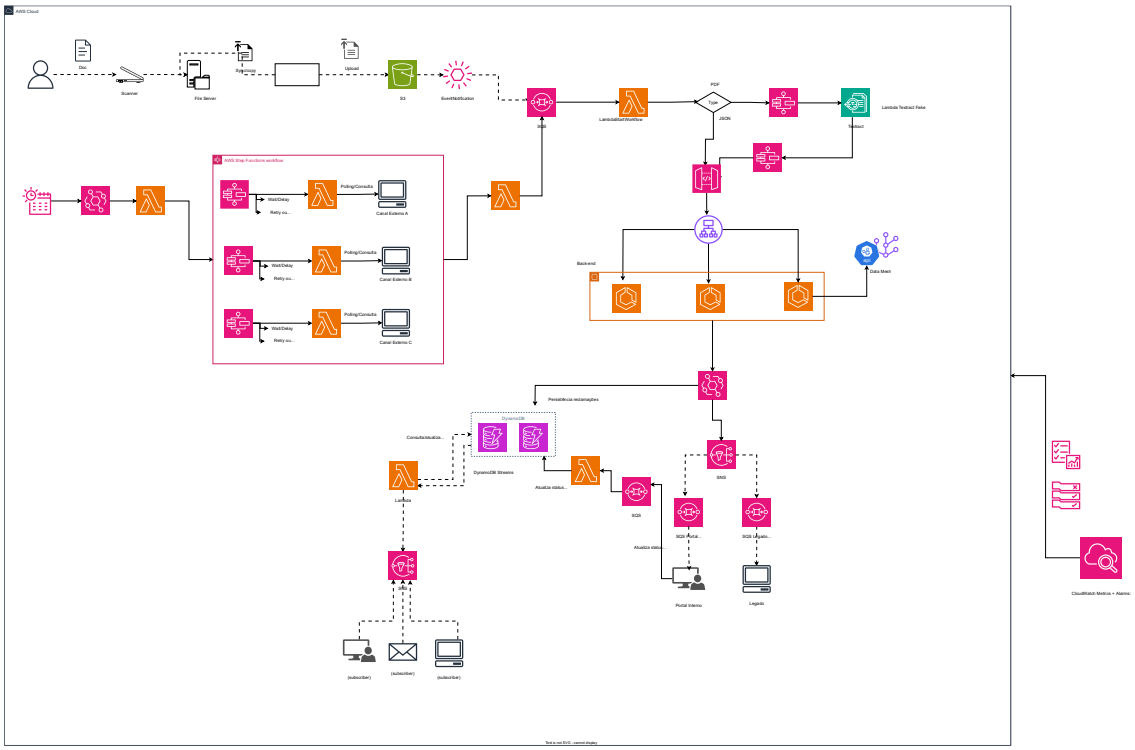
Amazon Comprehend -> modelos de NLP (Natural Language Processing) para analisar o conteúdo das mensagens e classificá-las automaticamente por prioridade, tipo ou canal com o Amazon Comprehend

Alertas inteligentes: IA poderia perceber quando o sistema está com problemas ou atrasos e acionar alertas automaticamente, sem precisar esperar uma pessoa perceber.

Sugestões automáticas: para mensagens que precisam de respostas padrões, a IA poderia gerar sugestões ou respostas automáticas, economizando tempo. (LLM)

Bônus: Roadmap da aplicação

Esta seção apresenta uma visão aprimorada do fluxo como um todo, simulando estar em produção, destacando processamento, monitoramento e mecanismos para garantir eficiência, escalabilidade e confiabilidade do sistema:


AWS Cloud
Doc
Scanner
File Server
Rename
Upload
S3
EventNotification
SNS
LambdaStartWorkflow
PDF
File
JSON
Textract
Lambda Textract Fake
AWS Step Functions workflow
Wait/Delay
Retry ou...
Polling/Consulta
Canal Externo A
Canal Externo B
Canal Externo C
Back-end
Data Mesh
Persistência reclamações
DynamoDB
ConsultaSituaCla...
DynamoDB Streams
Lambda
AtualizaStatus...
SQS
SQS Portal...
SQS Legado...
SNS
Portal Interno
Legado
(subscriber)
(subscriber)
(subscriber)
CloudWatch Metrics + Alarms
Text is not SVG - cannot display