Arquitetura de Software (2025.2 - T02)

Sistema de Logística Aeroportuária em Tempo Real

13/01/2026

Equipe: Bruno Lima Ferreira, Guilherme Damasceno Nobre, Monalisa Silva Bezerra.

Professor: Dr. Enyo José Tavares Goncalves.

1. Objetivo

Este documento tem como objetivo aplicar os conceitos de arquitetura de software por meio do projeto arquitetural de um sistema que pertence ao domínio de Logística Aeroportuária em Tempo Real. Esse sistema é caracterizado por uma complexidade adequada de acordo com o ponto de vista operacional, além de possuir alta necessidade de processamento assíncrono e tomada de decisão rápida e segura.

2. Modelo de Documentação

Para a documentação arquitetural foi utilizado o Modelo C4, por se tratar de um modelo bem definido para descrever arquiteturas de forma compreensível, estruturada e progressiva. Esse modelo é organizado em quatro níveis de abstração, o que garante que a arquitetura seja apresentada desde uma visão mais geral até um nível mais detalhado. Essa é uma característica importante para sistemas distribuídos e baseados em eventos, como é o sistema proposto neste trabalho.

A Visão de Contexto (C1) apresenta o objetivo como um todo, destacando os limites, as principais responsabilidades e as interações com atores externos e outros sistemas. Nessa visão, são identificadas as integrações com sistemas de companhias aéreas, meteorologia, alfândega, logística operacional, segurança e sistemas de comunicação, destacando o papel central do Sistema de Logística Aeroportuária em Tempo Real.

A Visão de Contêineres (C2) apresenta os principais blocos que compõem o sistema, tais como API de ingestão de eventos, serviços serverless, broker de mensagens, mecanismos de persistência, sistemas de visualização e serviços de processamento de domínio (voo, cargas, notificações, clima). Essa visão compreende como o sistema está internamente organizado, quais são as tecnologias e a infraestrutura utilizadas e como ocorre a comunicação entre os diferentes contêineres.

A Visão de Componentes (C3) aprofunda o nível de detalhamento ao mostrar a estrutura interna de contêineres específicos, ressaltando os componentes responsáveis pelo processamento dos eventos, tomada de decisão, manutenção de estado e publicação de novos eventos. Essa visão facilita o entendimento das dependências entre componentes de um mesmo serviço, do fluxo de dados e das responsabilidades internas.

A Visão de Código (C4) descreve um nível ainda mais detalhado, por meio de diagramas de sequência e payloads de eventos. Essa visão tem o objetivo de ilustrar como as decisões arquiteturais se materializam em fluxos concretos de execução, e não apresentar a implementação completa do sistema.

A escolha do Modelo C4 neste trabalho se justifica por sua clareza, adequação e flexibilidade, permitindo se alinhar com os estilos Event-Driven e Serverless. Além disso, o modelo contribui para a comunicação entre os stakeholders, favorecendo uma melhor compreensão da arquitetura proposta e de suas decisões.

3. Domínio da Aplicação

O domínio do trabalho é caracterizado por alta complexidade operacional, múltiplos fatores envolvidos, grande volume de dados e forte dependência de informações atualizadas continuamente. Esse sistema engloba a coordenação e o monitoramento de atividades relacionadas a voos, passageiros, cargas, bagagens, infraestrutura aeroportuária e condições externas, como fatores climáticos e regulatórios.

No contexto aeroportuário, diversos sistemas operam simultaneamente, mesmo sendo independentes entre si, incluindo sistemas de companhias aéreas, serviços de meteorologia, operadores logísticos, autoridades aeroportuárias, alfândega e segurança. Cada um desses sistemas gera eventos que impactam diretamente na operação de um aeroporto, como atrasos de voo, restrições climáticas, retenções de carga, entre outros. O principal desafio desse domínio é integrar essas informações de forma eficiente, permitindo uma visão bem consolidada.

O Sistema de Logística Aeroportuária em Tempo Real tem como finalidade centralizar a ingestão, a correlação e o processamento desses eventos, transformando dados brutos originários de múltiplas fontes em informações úteis para a tomada de decisão. A partir dessa correlação, o sistema deverá ser capaz de identificar situações de risco, como atrasos iminentes.
Além do suporte à decisão operacional, o sistema atua facilitando a comunicação entre os diversos stakeholders. Dessa forma, o sistema gera alertas, notificações e relatórios adequados a cada público, como companhias aéreas, centros de controle operacional, autoridades regulatórias e passageiros.

Outro ponto relevante do domínio é a necessidade de escalabilidade, resiliência e disponibilidade contínua, uma vez que falhas ou atraso no processamento das informações podem causar impactos severos na operação, como prejuízos financeiros, falha de segurança e insatisfação dos usuários. Por esse motivo, o domínio escolhido se beneficia de arquiteturas distribuídas e orientadas a eventos, pois são capazes de lidar com picos de carga e mudanças operacionais frequentes.

Portanto, a escolha da arquitetura baseada em eventos e a serverless são adequadas ao domínio, pois tais abordagens permitem lidar de forma eficiente com as características dinâmica, distribuída e crítica das operações aeroportuárias, garantindo a integração de novos serviços ao longo do tempo e suporte à evolução contínua do sistema.

4. Estilo Arquitetural

4.1 Visão geral

A arquitetura associa os estilos Event-Driven e Serverless para atender aos requisitos do domínio de Logística Aeroportuária em Tempo Real. Nessa combinação, os eventos são as unidades de integração e as funções gerenciadas (serviços serverless) são as unidades de execução. Os produtores publicam eventos padronizados em um barramento, e as funções reativas consomem esses eventos para executar a lógica de negócio, persistir o estado e publicar eventos derivados. Isso reduz o esforço de operação e facilita a escala independentemente de cada função.

4.2 Event-Driven

No padrão event-driven, os produtores externos (sistemas meteorológicos, companhias aéreas, alfândega, etc) publicam eventos em um Event Broker organizado por tópicos de domínio. Os consumidores independentes processam esses eventos de forma assíncrona e republicam eventos derivados para fechar os ciclos operacionais. Os principais benefícios buscados com esse estilo são baixo acoplamento entre sistemas, capacidade para reprocessamento e escalabilidade granular de consumidores.

4.3 Serverless

O estilo serverless é aplicado alocando a execução das unidades de processamento em Functions (FaaS) e utilizando serviços gerenciados para mensageria e persistência. A API de ingestão é exposta por um API Gateway que valida e publica eventos. As funções são acionadas por mensagens do broker e atualizam repositórios gerenciados (por exemplo, S3 para arquivamento de eventos). O processamento de streams e a materialização de read models podem ser realizados por serviços serverless de data processing. As vantagens dessa abordagem são menor custo operacional (pois paga-se por execução), escalabilidade automática e maior agilidade no desenvolvimento e deploy de novas regras e componentes.

4.4 Correlação entre os estilos

Nesse sistema os eventos fornecem os contratos e o fluxo lógico, enquanto as funções serverless materializam a lógica relativa a esses contratos. Essa sinergia garante modularidade e elasticidade, o broker desacopla produtores e

consumidores, enquanto as functions escalam conforme a demanda de eventos. Os estilos, simultaneamente, oferecem um trade-off favorável para lidar com picos meteorológicos, alto volume de telemetria e possível necessidade de histórico reprocessável.

4.5 Padrões operacionais e mecanismos essenciais

Algumas práticas são adotadas para garantir robustez operacional, como idempotência (cada evento carrega um id e consumidores mantêm a tabela de duplicação), políticas de retry com backoff exponencial e uso de Dead Letter Queue (DLO) para eventos que falham persistentemente, particionamento por chaves relevantes quando ordering for crítico e retenção de eventos em um Event Store para replay e auditoria. As projeções materializadas (read models) atendem dashboard e consultas operacionais, enquanto o tracing distribuído, as métricas e os logs estruturados fornecem a observabilidade necessária para a operação.

4.6 Trade-offs, riscos e mitigação

As escolhas implicam em alguns trade-offs, a consistência eventual é uma consequência natural do event-driven, mitigada por projeções frequentes e, quando necessário, por chamadas síncronas pontuais em caminhos críticos. Outros trade-offs são: complexidade de debugging em sistemas distribuídos é tratada com tracing distribuído e correlação de id em eventos, risco de vendor lock-in no uso extensivo de serviços serverless e o potencial custo elevável durante picos de execução é controlado por limites, batching e análises de custo/benefício.

4.7 Requisitos não-funcionais e impacto arquitetural

As decisões arquiteturais toleram requisitos não-funcionais, como disponibilidade por replicação e multi-region do broker, particionamento e escalabilidade via serverless, segurança por autenticação, latências definidas por SLAs para caminhos críticos e auditabilidade garantida pelo Event Store que possibilita investigação de incidentes e reconstrução de estado.

5. Visões Arquiteturais (Modelo C4)

Os diagramas arquiteturais abaixo foram desenvolvidos para apresentar o domínio da aplicação escolhida. Com base neles que as diferentes visões do sistema são apresentadas.

5.1 Component Diagram - Notification Orchestrator

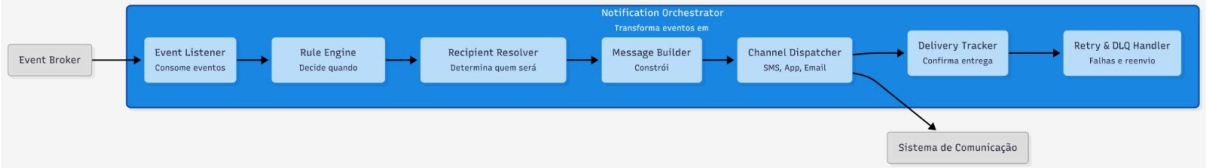

diagrama 1

O diagrama 1 mostra um Notificator Orchestrator (orquestrador de notificações) que recebe eventos de um Event Broker e transforma esses eventos em mensagens que chegam até o usuário para vários canais (SMS, app, e-mail). O fluxo principal é linear, com componentes responsáveis para decidir quando, quem, o que e como notificar, além de rastrear entregas e encaminhar falhas.

- **Event Broker**: Sistema que publica/encaminha eventos do domínio, por exemplo, “pedido enviado”, “pagamento aprovado”, “senha redefinida”. Fornece desacoplamento entre produtores e o orquestrador.

- **Event Listener - Consome eventos:** Componente que se conecta ao broker e consome os eventos. Normalmente faz parsing do payload e passa o evento para o pipeline interno. Aqui também costuma haver controle de offset/ack (para garantir que o evento não se perca).

- **Rule Engine - Decide quando:** Avalia regras (negócio e preferências) que definem se o evento deve gerar notificação e quais condições (por exemplo, só notificar clientes VIP, ou não enviar promoções à noite). Pode ser baseado em regras estáticas, DSL ou motor de regras (Drools, rules-as-code).

- **Recipient Resolver - Determina quem será:** Resolve os destinatários a partir do evento: usuário principal, contatos relacionados, grupos, etc. Também aplica preferências de contato (por exemplo, prefere push, não quer

SMS) e pode consultar um perfil de usuário (micro serviço de perfil).

- **Message Builder - Constrói:** Monta o conteúdo da notificação: escolhe template, substitui variáveis (nome, número do pedido), aplica internacionalização, curtos/longos conforme canal. Pode também gerar payloads diferentes por canal (texto para SMS, HTML para e-mail).

- **Channel Dispatcher - SMS, App, Email:** Encaminha o payload para o canal apropriado. Implementa adaptadores/integrations com provedores (Twilio, Firebase, SendGrid) ou com um *Sistema de Comunicação* interno (mostrado no diagrama 1). Pode paralelizar envios quando houver múltiplos canais.

- **Delivery Tracker - Confirma entrega:** Recebe confirmações/recebimentos (delivery receipts) dos canais ou dos provedores e atualiza o status da notificação (entregue, lido, falhou). Importante para relatórios e para métricas de SLA.

- **Retry & DLO Handler - Falhas e reenvio:** Trata falhas temporárias: enfileira para retry com backoff; quando se trata de uma falha definitiva, envia para uma Dead Letter Queue (DLO) para análise manual ou reprocessamento posterior.

- **Sistema de Comunicação** (externo): É o serviço/provedor real que envia as mensagens (provedores de SMS, servidores de e-mail, FCM/APNs para push). O dispatcher conversa com esse sistema.

5.2 Component Diagram - Flight Service

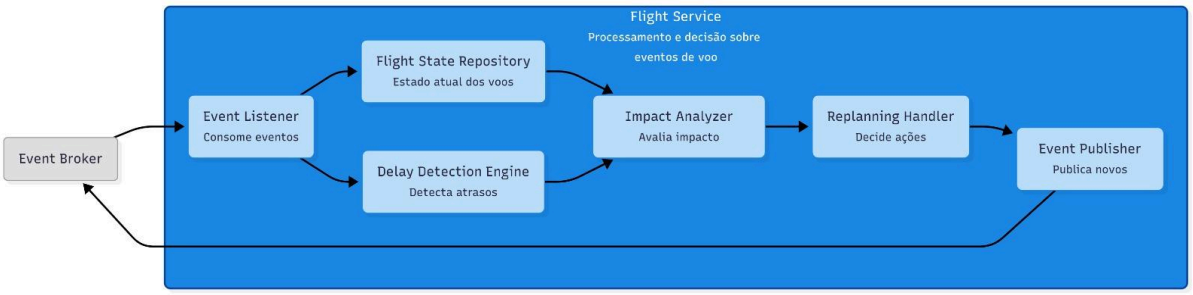

diagrama 2

O diagrama 2 mostra o Flight Service, responsável por processar eventos de voo, manter o estado atual, detectar atrasos, avaliar impactos e decidir ações de replanejamento. Ele se comunica com outros sistemas por meio de um Event Broker, seguindo um modelo event-driven. Seus componentes e responsabilidades são:

- **Event Broker:** Infraestrutura de mensageria. Distribui eventos como: voo programado, atraso detectado, portão alterado E voo cancelado. Permite desacoplamento entre sistemas.
- **Event Listener - Consome eventos:** Porta de entrada do Flight Service, escuta eventos do Event Broker, normaliza/valida os eventos recebidos e encaminha os dados para os componentes internos.

- **Flight State Repository - Estado atual dos voos:** Armazena o estado corrente de cada voo: horário planejado × horário real, status (no horário, atrasado, cancelado), aeronave, tripulação, portão. Serve como fonte de verdade para decisões.

- **Delay Detection Engine - Detecta atrasos:** Compara eventos reais com o planejamento. Identifica: atrasos acima de um limite e riscos de atraso futuro. Pode usar regras simples ou modelos preditivos. Emite sinais para análise de impacto.

- **Impact Analyzer - Avalia impacto:** Analisa consequências do atraso: se os passageiros perderão conexões, se a tripulação estoura limite de jornada, a aeronave afeta voos seguintes. Usa dados do Flight State Repository + Delay Detection Engine. Gera um “cenário de impacto”.

- **Replanning Handler - Decide ações:** Toma decisões com base no impacto: reacomodar passageiros, trocar aeronave, atrasar voos subsequentes, cancelar voo. Pode envolver regras de negócio complexas. Não executa tudo diretamente, faz a orquestração de decisões.

- **Event Publisher - Publica novos eventos**: Publica eventos derivados no Event Broker. Esses eventos serão consumidos por: Notification Service, sistemas de atendimento e aplicativos de passageiro. Fecha o ciclo de feedback do sistema.

5.3 Diagrama de Componentes e Serviços das Companhias Aéreas

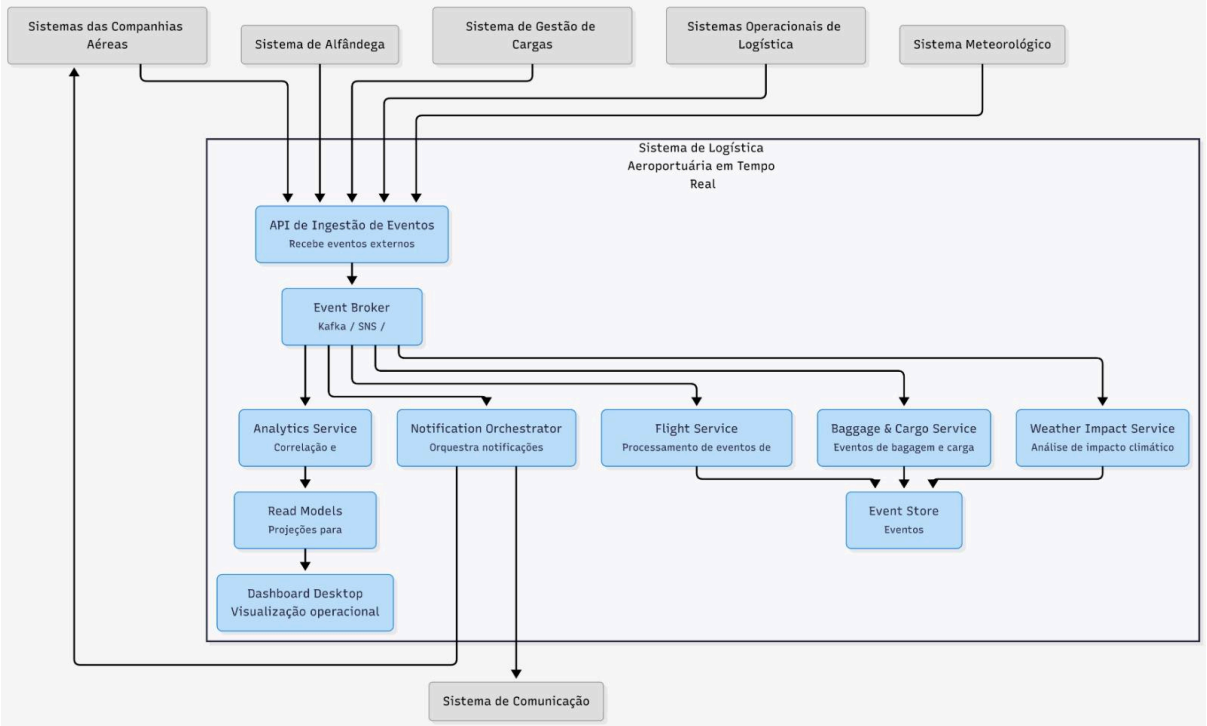

diagrama 3

O diagrama 3 recebe informações de sistemas externos (companhias aéreas, alfândega, gestão de cargas, logística operacional, meteorologia), ingere esses eventos, processa-os por vários serviços (voos, bagagem, clima, analytics), mantém um Event Store, produz projeções para dashboards e aciona um Notification **Orchestrator** que fala com um sistema de comunicação (SMS/push/e-mail). O Event Broker é a parte central da mensageria que conecta tudo. O fluxo principal e os papéis dos blocos do diagrama 3 podem ser definidos como:

- **Sistemas externos:** Sistemas das companhias aéreas, Alfândega, Gestão de Cargas, Operações de Logística, Meteorologia enviam eventos e atualizações.

- **API de Ingestão de Eventos:** Ponto de entrada público/seguro. Valida, autêntica e normaliza eventos externos e os publica no Event Broker. Também pode aplicar versionamento e um schema registry antes de publicar.

- **Event Broker:** Sistema de streaming/pub-sub. Desacopla produtores e consumidores. Serve topics por domínio (voos, bagagem, meteorologia, cargas). Suporta replays, particionamento e alta taxa de transferência.
- **Serviços consumidores:**
    - **Flight Service**: Processa eventos de voo, mantém estado dos voos, toma decisões de planejamento e publica eventos derivados.
    - **Baggage & Cargo Service**: Trata eventos de movimentação de bagagem e cargas, detecta perdas/offloads, atualiza rastreamento.
    - **Weather Impact Service**: Analisa alertas meteorológicos e calcula impacto em operações (visibilidade, capacidade de pouso).
    - **Notification Orchestrator**: Orquestra mensagens aos passageiros/operadores (usa preferências, templates e encaminha ao Sistema de Comunicação).
    - **Analytics Service**: Consome streams para correlação, agregações e extração de insights (ex.: previsão de atrasos por causa do clima).

- **Event Store:** Armazena eventos de forma durável (event sourcing) para auditoria, replays e reconstrução de estado. Serviços escrevem seus eventos importantes no Event Store.

- **Read Models - Dashboard Desktop:** O Analytics Service ou processos separados geram projeções/read models (ex.: tabela de voos atuais, filas de bagagem, KPIs) para consumo pela interface operacional (dashboard). O padrão CQRS: escrita por eventos, leitura por modelos otimizados.

- **Sistema de Comunicação:** Provedores reais (SendGrid, FCM, SMS gateway interno). O Notification Orchestrator envia para esse sistema, que faz a entrega aos usuários.

5.4 Diagrama de Arquitetura de Sistemas Aeroportuários e Logísticos

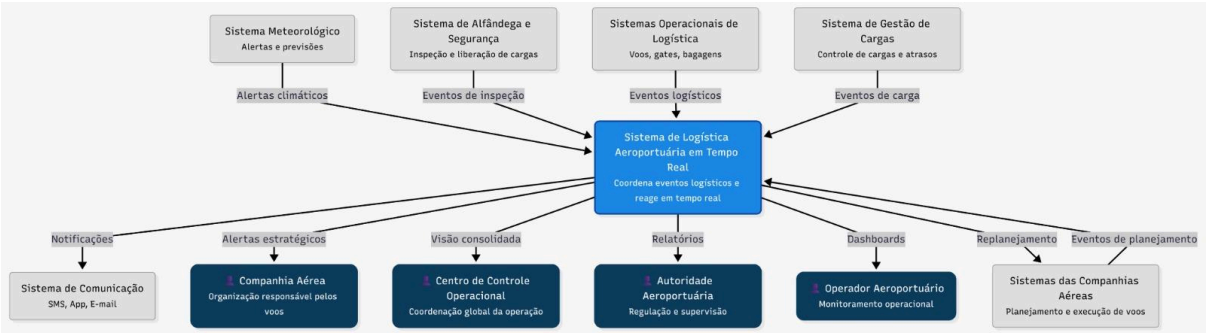

diagrama 4

O diagrama 4 apresenta o Sistema de Logística Aeroportuária em Tempo Real no centro. Esse diagrama recebe eventos de várias fontes externas (meteorologia, alfândega, operações, gestão de cargas), processa e correlaciona essas informações e então entrega diferentes produtos/saídas para interessados operacionais e de negócio (companhia aérea, centro de controle, autoridade, operador aeroportuário) além de enviar notificações aos passageiros via um Sistema de Comunicação. Cada um dos blocos abaixo envia tipos de eventos específicos:

- **Sistema Meteorológico:** Alertas climáticos e previsões (vento forte, neblina, tempestade, previsão de baixa visibilidade).

- **Sistema de Alfândega e Segurança:** Eventos de inspeção/retenção (cargas retidas, inspeção de segurança, liberações).

- **Sistemas Operacionais de Logística:** Eventos logísticos (status de gates, movimentação de bagagens, chegada/partida de veículos de rampa).

- **Sistema de Gestão de Cargas:** Eventos de carga (aviso de carga atrasada, mismatch de pallet, cargas perecíveis com problemas).

O sistema gera vários artefatos para diferentes públicos, sendo eles:

- **Sistema de Comunicação:** Recebe notificações para passageiros e stakeholders (SMS, app, e-mail). Exemplos: aviso de atraso, instrução de reacomodação.

- **Companhia Aérea:** Recebe alertas estratégicos (impacto em malha, necessidade de troca de aeronave, rebooking em massa).

- **Centro de Controle Operacional:** Recebe visão consolidada, painel com o estado atual e recomendações para coordenação global da operação.

- **Autoridade Aeroportuária:** Recebe relatórios e indicadores (capacidade do aeroporto, incidentes, conformidade regulatória).

- **Operador Aeroportuário:** Dashboards operacionais para tomada de decisão em tempo real.

- **Sistemas das Companhias Aéreas:** Recebem eventos de planejamento.

5.5 Diagrama de Classes do Módulo de Impacto Climático e Detecção de Atrasos

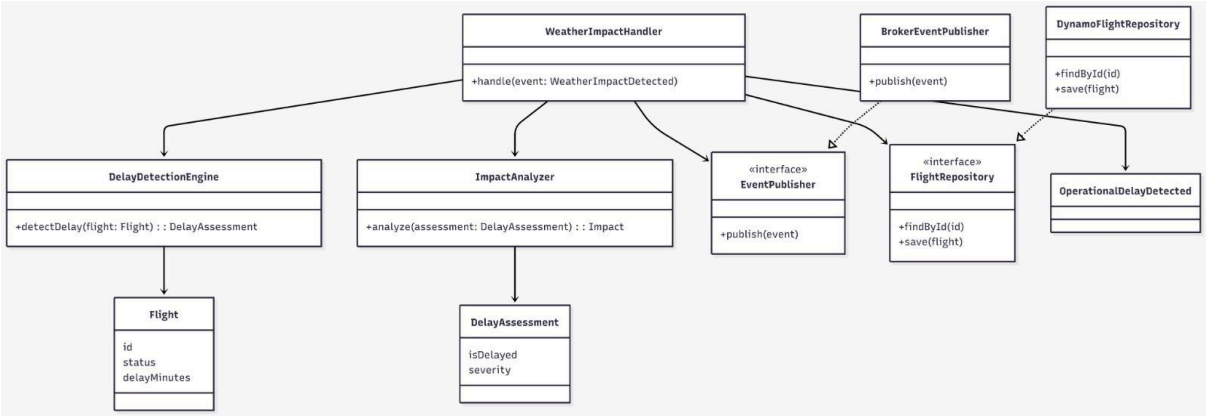

diagrama 5

O diagrama 5 é um diagrama de classes/componentes que descreve o fluxo de tratamento de um evento de impacto meteorológico e como isso resulta em detecção de atraso operacional e publicação de um novo evento. Seus componentes e responsabilidades são:

- **WeatherImpactHandler:** Componente orquestrador. Recebe o evento de impacto do tempo e coordena as próximas ações (detecção, análise, persistência e publicação).
- **DelayDetectionEngine:** Serviço que recebe um objeto e avalia se há atraso. Retorna um “DelayAssessment”.

- **ImpactAnalyzer:** Serviço que, a partir do “DelayAssessment”, produz um artefato de impacto (por exemplo qual o alcance do problema: conexão perdida, necessidade de rebooking, etc.).

- **EventPublisher (interface):** Interface abstrata para publicar eventos. Permite a injeção de diferentes implementações (broker, fila, outbox).

- **BrokerEventPublisher:** Implementação concreta que publica no broker (Kafka, EventBridge, etc.).

- **FlightRepository (interface):** Interface de persistência com métodos, abstrai o armazenamento do estado dos voos.

- **DynamoFlightRepository:** Implementação concreta da interface, usando DynamoDB (ou outro storage gerenciado).

- **OperationalDelayDetected:** Representa o evento (ou mensagem) gerado quando um atraso operacional é confirmado. É o tipo de evento que será publicado no broker.

- **Domain objects**
    - **Flight**: entidade com atributos (id, status, delayMinutes).
    - **DelayAssessment**: resultado da detecção (isDelayed, severity).

O fluxo do diagrama 5 pode ser apresentado como:

- **Recebe evento**: O sistema (por ex., uma função serverless acionada pelo broker) entrega um evento WeatherImpactDetected ao WeatherImpactHandler.handle(...).

- **Carrega o estado do voo:** WeatherImpactHandler usa FlightRepository.findById(flightId) para obter o Flight atual.

- **Detecta atraso:** Chama DelayDetectionEngine.detectDelay(flight) que avalia se o voo está ou ficará em atraso e retorna um DelayAssessment.

- **Analisa impacto: C**hama ImpactAnalyzer.analyze(DelayAssessment) para transformar a avaliação em um Impact (por ex., minor_delay, critical_delay, afetação de conexões).

- **Atualiza estado / decisões:** Com base no impacto, o handler pode:
    - Atualizar campos do Flight (status, delayMinutes).
    - Salvar alterações: FlightRepository.save(flight).

- **Publica evento derivado:** Se o impacto indicar atraso operacional, o handler constrói um OperationalDelayDetected e publica via EventPublisher.publish(event) (normalmente BrokerEventPublisher).

- **Consumidores downstream:** Outros serviços (Notification Orchestrator, Replanning Handler, Analytics) consomem OperationalDelayDetected e reagem.

6. Conclusão

Este trabalho apresentou o projeto arquitetural de um Sistema de Logística Aeroportuária em Tempo Real, construído nos estilos Event-Driven e Serverless. A arquitetura proposta permite integrar múltiplas fontes de eventos, processar decisões operacionais de forma assíncrona e reagir rapidamente a situações críticas, como impactos climáticos e atrasos de voo.

O uso de eventos como mecanismo principal de comunicação garante baixo acoplamento, escalabilidade e rastreabilidade, enquanto a abordagem serverless reduz a complexidade operacional e facilita a evolução do sistema. O fluxo exemplificado pelo tratamento de impacto meteorológico demonstra, de forma prática, como eventos externos são transformados em decisões e novos eventos operacionais.

Por fim, podemos concluir que a solução atende aos requisitos do domínio aeroportuário, oferecendo uma base sólida, flexível e escalável, adequada para sistemas que demandam alta disponibilidade, resposta em tempo real e integração contínua entre múltiplos atores.