

Universidade do Minho  
Escola de Engenharia  
Mestrado em Engenharia Informática

## Unidade Curricular de Aplicações e Serviços de Computação em Nuvem

Ano Letivo de 2024/2025

### Checkpoint Final - Airtrail Deployment



Miguel Gramoso  
A100835



Luís Figueiredo  
PG61532



Gustavo Barros  
PG61527



Vasco Mota  
PG58811



Enzo Vieira  
PG61518

December 30, 2025

ASCN

# Índice

<b>1.</b>	<b>Introdução .....</b>	<b>2</b>
<b>2.</b>	<b>Arquitetura Geral .....</b>	<b>2</b>
2.1.	<i>Frontend .....</i>	2
2.2.	<i>Backend .....</i>	3
2.3.	Base de Dados .....	3
<b>3.</b>	<b>Deployment .....</b>	<b>3</b>
<b>4.</b>	<b>Funcionalidades .....</b>	<b>4</b>
<b>5.</b>	<b>APIs fornecidas pela aplicação .....</b>	<b>5</b>
<b>6.</b>	<b>Discussão crítica sobre as questões colocadas na Tarefa 2.....</b>	<b>5</b>
6.1.	Questão 1: Análise da Instalação Base .....	5
6.2.	Questão 2: Otimizações Propostas .....	6
<b>7.</b>	<b>Instalação e configuração automática da aplicação .....</b>	<b>7</b>
7.1.	Ferramentas Utilizadas .....	7
7.2.	Abordagem e Estratégia de Automação .....	8
7.2.1.	Aprovisionamento da Infraestrutura (IaaS) .....	8
7.2.2.	Gestão de Configuração e Segredos .....	8
7.2.3.	Orquestração e Persistência .....	8
<b>8.</b>	<b>Ferramentas de monitorização, métricas e visualizações escolhidas .....</b>	<b>8</b>
<b>9.</b>	<b>Ferramentas de avaliação e testes desenvolvidos .....</b>	<b>9</b>
9.1.	Testes Funcionais .....	9
<b>10.</b>	<b>Análise de Resultados .....</b>	<b>10</b>
<b>11.</b>	<b>Reflexão final .....</b>	<b>10</b>

# 1. Introdução

Este relatório serve de apoio ao desenvolvimento do projeto da *UC* de Aplicações e Serviços de Computação em Nuvem, mais concretamente, ao *Checkpoint Final* do projeto.

Neste documento pretendemos fazer uma análise profunda da aplicação *Airtrail*, uma solução open-source e self-hosted que permite aos seus utilizadores registar, visualizar e analisar o seu histórico de viagens aéreas através de uma interface moderna e intuitiva.

A análise desta aplicação pretende não só identificar limitações na sua estrutura base, mas sobretudo servir de suporte à implementação de uma solução robusta que tire partido dos serviços geridos da Google Cloud, automatizando o seu ciclo de vida e monitorização.

# 2. Arquitetura Geral

O *Airtrail* é uma aplicação desenvolvida focada em uma arquitetura monolítica full-stack em *SveltKit* com o deployment feito com o auxilio de *Docker*. A aplicação foi desenhada de modo a utilizar um modelo cliente-servidor comum, onde o *Sveltekit* funciona tanto como *frontend* e *backend*.

## 2.1. Frontend

O *frontend* da aplicação foi construído com *Svelte*, resultando em uma *UI* apelativa aos utilizadores e de fácil compreensão. Os utilizadores conseguem observar um mapa com os seus voos

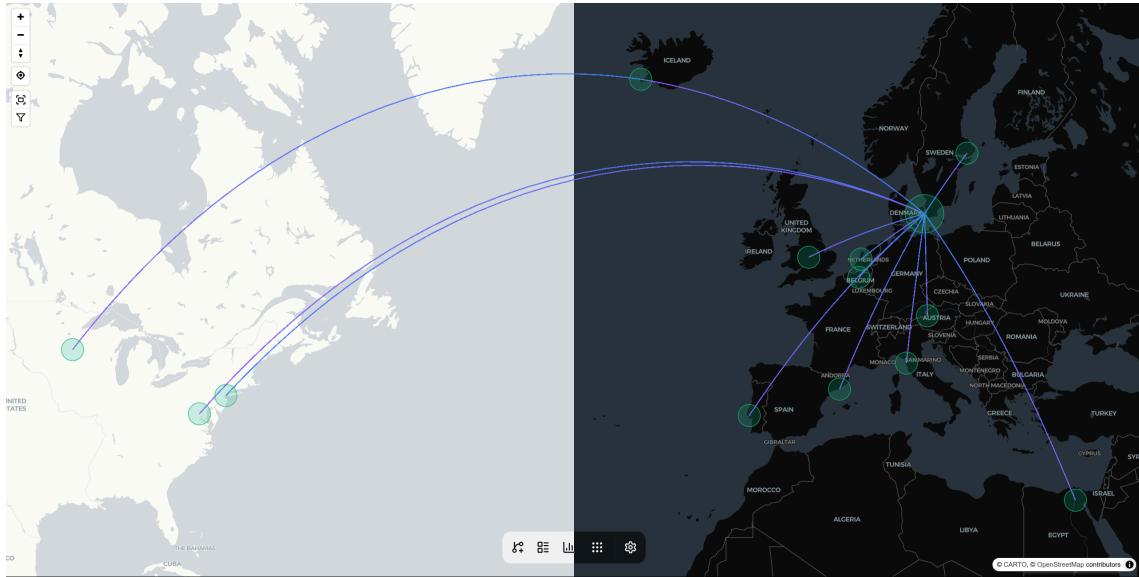


Figura 1: Exemplo de um mapa de voos

## 2.2. Backend

O *Backend* utiliza uma API RESTful através das *server routes* do *Sveltekit*. É responsável por todas as operações relacionadas com a lógica da aplicação, desde o processamento de voos até à validação de dados.

A autenticação é implementada através de OIDC (OpenID Connect), que permite a integração com múltiplos *providers*.

Um dos componentes a destacar são os múltiplos *parsers* para sites comuns de *track* de voos, como o MyFlightRadar24, App in the Air e JetLog.

## 2.3. Base de Dados

A base de dados utilizada pelo *AirTrail* é construída com *PostgreSQL*, um sistema de gestão de base de dados relacional. O *schema* da base de dados está organizado em tabelas principais que incluem registos de voos com as informações de origem, destino, datas, duração e companhia aérea, dados de utilizadores e as suas credenciais de autenticação, informações detalhadas sobre veículos utilizados, incluindo modelos e registos, catálogo de companhias aéreas, e base de dados de aeroportos com códigos identificadores e coordenadas geográficas.

## 3. Deployment

A aplicação já foi desenhada para a utilização de *Docker* no seu *deployment*, uma ferramenta que os alunos aprenderam a utilizar durante o decorrer da UC. É recomendado

a utilização de *Docker Compose* para organizar tanto o *container* da aplicação quanto o *container* de *PostgreSQL*.

## 4. Funcionalidades

O *AirTrail* disponibiliza um conjunto abrangente de funcionalidades que permitem ao utilizador gerir e visualizar os seus voos de forma intuitiva. Entre as principais funcionalidades destacam-se:

- **Autenticação:** sistema de *login* com suporte a múltiplos utilizadores e configuração de OAuth.
- **Atualizações:** notificação automática quando existe uma nova versão da aplicação, com instruções de atualização.
- **Gestão de voos (CRUD):** criação, edição, remoção e consulta de voos, com detalhes como origem, destino, horários, companhia aérea e passageiros.
- **Mapa-mundo interativo:** visualização geográfica dos voos, distinguindo entre voos passados e futuros; inclui filtros, localização atual do utilizador (caso haja essa permissão), navegação livre, redefinição do norte e controlo de *zoom*.
- **Listagem de voos:** apresentação em formato de lista, com filtros por data e outros critérios.
- **Estatísticas:** cálculo e visualização de estatísticas relacionadas com os voos já realizados, com a possibilidade de expandir cada gráfico para ecrã completo e consultar detalhes adicionais sobre os dados apresentados.
- **Ferramentas adicionais:**
  - Consola SQL integrada para interação direta com a base de dados.
  - Função para deteção e remoção de voos duplicados.
- **Globo de países visitados:** representação 3D dos países visitados, vividos, visitados em escala ou adicionados à lista de desejos, com possibilidade de sincronizar automaticamente a informação com os voos registados.
- **Definições e gestão de dados:** opções gerais da aplicação, troca entre modo claro e modo escuro, partilha pública de voos (*shares*), importação de dados de outras plataformas (como *FlightRadar24*, *App in the Air*, etc.), exportação de dados em *JSON* ou *CSV*, atualização automática e adição manual de informações sobre aeroportos, companhias aéreas e aviões.
- **Integração externa:** ligação opcional ao serviço *AeroDataBox*, que fornece uma pesquisa de voos melhorada como forma de substituição do serviço *adsbdb*.
- **Gestão de utilizadores (CRUD):** criação, edição, remoção e consulta de utilizadores.
- **Configuração de permissões e autenticação:** ativação e gestão da autenticação via OAuth, que controla o acesso dos utilizadores à instância *AirTrail* e permite automatizar o processo de início de sessão ou de registo de utilizadores.

## 5. APIs fornecidas pela aplicação

O *AirTrail* disponibiliza uma API REST que permite a gestão completa dos voos registrados pelo utilizador. As principais rotas incluem:

- GET `/flight/list` – devolve a lista de todos os voos do utilizador.
- GET `/flight/get/{id}` – obtém os detalhes de um voo específico.
- POST `/flight/save` – cria um novo voo ou atualiza um voo existente, consoante a presença do campo `id`.
- POST `/flight/delete` – remove um voo com base no seu identificador.

Estas rotas são utilizadas internamente pela aplicação para sincronizar os dados entre o *frontend* e o servidor.

Estas são as rotas documentadas oficialmente, responsáveis pela gestão de voos. Outras funcionalidades da aplicação são tratadas internamente e não possuem *endpoints* públicos descritos.

## 6. Discussão crítica sobre as questões colocadas na Tarefa 2.

### 6.1. Questão 1: Análise da Instalação Base

a. Para um número crescente de clientes, que componentes da aplicação poderão constituir um gargalo de desempenho?

O servidor caso o sistema não suporte escalamento horizontal, pois um único servidor pode ser saturado com pedidos, atrasando assim a resposta a todos eles. A base de dados única, sem replicação, devido a bloqueios e excesso de conexões. A rede é outro ponto crítico: largura de banda insuficiente ou latência elevada degradam o *throughput* e a experiência do utilizador.

b. Qual o desempenho da aplicação perante diferentes números de clientes e cargas de trabalho?

O grupo optou por adicionar um teste que permita fazer *benchmark* ao sistema implementado. Utilizando ferramentas como o `apache2-utils`, testamos a performance de realizar 100 pedidos com origem em 10 clientes concorrentes.

```

TASK [test_airtrail/benchmark : Executar Benchmark (100 pedidos, 10
utilizadores concorrentes)] **changed: [localhost]
TASK [test_airtrail/benchmark : Mostrar métricas principais no
terminal]
***ok: [localhost] => {
"msg": [
    "Pedidos por segundo: Requests per second: 27.26",
    "Tempo médio por pedido: Time per request:366.881"
]
}

```

Em cenários com poucos utilizadores, a aplicação deverá manter um desempenho aceitável, visto que os processamento das consultas à base de dados não terão grande carga. Contudo, com o aumento do número de clientes e requisições, é esperado que o tempo médio de espera cresça, por conta da falta de escalabilidade horizontal da instalação.

**c. Que componentes da aplicação poderão constituir um ponto único de falha?**

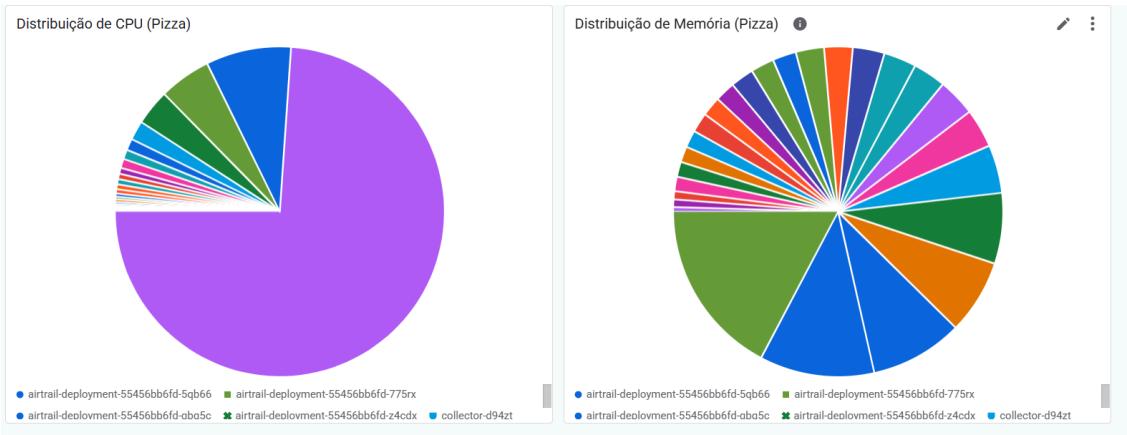
O servidor único e a base de dados única, pois com a falha de um destes componentes, toda a aplicação fica inoperável.

## 6.2. Questão 2: Otimizações Propostas

**a. Que otimizações de distribuição/replicação de carga podem ser aplicadas à instalação base?**

Algumas técnicas podem ser exploradas para otimizar a distribuição/replicação de carga.

A primeira delas pode ser a escalabilidade horizontal do servidor web, com a criação de múltiplas instâncias do *container web*. Outra abordagem é a replicação e persistência da base de dados, já que isso aumenta a disponibilidade do serviço e protege a aplicação contra perda de dados em caso de falha de um nó. Atualmente, a infraestrutura já integra um *load balancer* para a distribuição equitativa do tráfego entre os nós e mecanismos de monitorização para observar o estado dos serviços. Foi também implementado replicação e escalonamento automático, permitindo ajustar dinamicamente o número de instâncias do servidor conforme a carga de trabalho detetada, podendo variar de 2 a 5 réplicas. Uma nova réplica é aberta sempre que a cpu ou a memória chega a 80% de utilização.



Nesta imagem podemos ver as distribuições de CPU e memória após a implementação das funcionalidades.

#### b. Qual o impacto das otimizações propostas no desempenho e/ou resiliência da aplicação?

O *load balancer* aumenta o *throughput* e melhora o escalonamento horizontal, permitindo que mais réplicas possam ser criadas e mais utilizadores possam utilizar a aplicação. A replicação da base de dados melhora a resiliência e a disponibilidade e a monitorização e escalonamento automático permitirão uma resposta rápida a diferentes cargas de forma transparente.

## 7. Instalação e configuração automática da aplicação

Esta secção do relatório detalha as ferramentas e a estratégia adotada para garantir que a instalação e configuração da aplicação AirTrail sejam totalmente automatizadas, reprodutíveis e eficientes.

### 7.1. Ferramentas Utilizadas

A solução baseia-se num ecossistema de ferramentas **Cloud-Native** e de automação de infraestrutura:

- **Ansible:** Utilizado como o motor principal de automação e orquestração. Através de Playbooks e Roles, o Ansible gere desde a criação do cluster até ao deploy final da aplicação.
- **Google Kubernetes Engine (GKE):** O serviço gerido de Kubernetes da Google Cloud onde a aplicação é executada. O GKE oferece escalabilidade automática e gestão simplificada de nós.
- **Kubernetes (K8s):** Plataforma de orquestração de contentores que gere o ciclo de vida dos Pods, Services e Volumes.

- **Docker**: Tecnologia de contentorização que empacota a aplicação AirTrail e a base de dados PostgreSQL em imagens isoladas e prontas a correr.
- **Ansible Vault**: Ferramenta de segurança integrada no Ansible para cifrar dados sensíveis, como palavras-passe de base de dados e chaves de API.

## 7.2. Abordagem e Estratégia de Automação

A abordagem segue o princípio de **Infrastructure as Code** (IaC), dividida em fases lógicas para minimizar a intervenção manual:

### 7.2.1. Aprovisionamento da Infraestrutura (IaaS)

A criação do cluster de Kubernetes é feita através do playbook `gke-cluster-create.yml`. Este utiliza módulos do GCP para definir dinamicamente o número de nós, o tipo de máquina e a zona de deploy conforme configurado no inventário `inventory/gcp.yml`.

### 7.2.2. Gestão de Configuração e Segredos

Para proteger a integridade do sistema, a configuração de segredos é automatizada pela role `create-secrets`.

- **Kubernetes Secrets**: Utilizados para injetar credenciais de base de dados diretamente nos pods sem as expor no código-fonte.
- **Variáveis Dinâmicas**: O Ansible extrai automaticamente o IP externo do LoadBalancer após o deploy para atualizar a variável `ORIGIN`, corrigindo problemas de **Cross-Origin** comuns na instalação manual.

### 7.2.3. Orquestração e Persistência

A aplicação é decomposta em componentes distintos para garantir resiliência:

- **Segregação de Pods**: A base de dados PostgreSQL e o servidor web AirTrail correm em pods separados.
- **Persistência com PVC**: A role `deploy-postgres` configura um **Persistent Volume Claim** (PVC). Isto garante que, se o pod da base de dados for reiniciado ou a aplicação for parada para manutenção, os dados dos utilizadores e voos não sejam perdidos.

## 8. Ferramentas de monitorização, métricas e visualizações escolhidas

Para monitorizar a saúde e o desempenho do cluster *GKE*, utilizámos as ferramentas nativas da *Google Cloud*, especificamente o `Google Cloud Monitoring`.

As métricas e visualizações escolhidas foram:

- **Utilização de CPU e Memória RAM:** Monitorizámos estas métricas nos nós do *cluster* individualmente para o servidor *web* e a base de dados. Isto permitiu-nos identificar se a aplicação estava próxima dos limites dos recursos definidos.
- **Latência de Rede:** Analisámos o tempo de resposta dos pedidos *HTTP* que chegavam através do *LoadBalancer* para garantir que a experiência do utilizador não se degrada.
- **Dashboards:** Utilizámos o painel de controlo do *GKE* da Google Cloud. Esta escolha deve-se à sua integração imediata, não exigindo a instalação de componentes externos, e por facilitar a visualização em tempo real do estado dos *containers*.

## 9. Ferramentas de avaliação e testes desenvolvidos

### 9.1. Testes Funcionais

O ficheiro `test-all.yml` orquestra o ciclo de vida completo da aplicação para validar a sua funcionalidade e resiliência. Os testes estão divididos em cinco fases críticas:

- **Fase 1: Deploy e Acesso:** Garante que o cluster e a aplicação são instalados corretamente, verificando se o serviço responde com um código *HTTP* 200 e realizando a criação automática da conta de administrador.
- **Fase 2: Interação com a API:** Valida a lógica de negócio através da criação de um novo voo via API, confirmando que a comunicação entre o frontend, backend e base de dados está operacional.
- **Fase 3: Disponibilidade e Undeploy:** Verifica se, após um comando de **undeploy** (sem remoção de dados), a aplicação fica efetivamente inacessível, garantindo que a paragem do serviço funciona como esperado.
- **Fase 4: Persistência de Dados:** É a fase mais crítica. Após reinstalar a aplicação, o sistema tenta recuperar a informação do voo criado na Fase 2. O sucesso deste teste prova que o **Persistent Volume Claim (PVC)** está a funcionar e que os dados sobrevivem a reinícios do sistema.
- **Fase 5: Limpeza Total:** Testa o cenário de “clean start”, onde os dados são apagados propositalmente. O teste valida que, após esta limpeza, a aplicação não reconhece os dados antigos (retornando erro 401), garantindo a integridade dos scripts de remoção.

## 10. Análise de Resultados

Os resultados obtidos através da execução do ficheiro `test-all.yaml` e da monitorização na *Google Cloud* permitiram verificar o seguinte:

- **Persistência:** O teste de Fase 4 permitiu verificar que o volume persistente guarda os dados corretamente mesmo após o *undeploy* da aplicação. Isto cumpre um dos requisitos fundamentais da manutenção programada sem a perda de dados.
- **Automação:** O tempo total para provisionar o *cluster* e fazer o *deploy* da aplicação foi reduzido para poucos minutos, com zero intervenção manual após o lançamento do *Ansible*.
- **Estabilidade:** Durante os testes, o Google Cloud Monitoring não registou falhas críticas ou picos anormais de consumo de *RAM* no *PostgreSQL*, o que permite-nos verificar que as configurações das máquinas são adequadas para a carga de trabalho atual.

## 11. Reflexão final

Este trabalho permitiu-nos aplicar na prática conceitos aprendidos durante a *UC*, tanto a nível de infraestrutura, código e orquestração de *containers*.

O trabalho prático realizado permitiu-nos consolidar conhecimentos adquiridos ao longo do semestre da Unidade Curricular, sobretudo os conceitos relacionados com as infraestrutura, o desenvolvimento de código e gestão de sistemas de *containers*. Um dos objetivos alcançados na implementação foi a automatização integral do ciclo de vida da aplicação, o que permitiu a resolução dinâmica da variável **ORIGIN** – um obstáculo que se revelou durante o processo de instalação manual.

Optámos também por reforçar a robustez e a segurança da implementação através da utilização do *Ansible Vault* para a gestão de segredos e da configuração de *Persistent Volume Claims* para assegurar a integridade e persistência dos dados, tornando a infraestrutura apta para um cenário de utilização real.

Em suma, o projeto cumpriu com sucesso os objetivos pretendidos, resultando numa solução funcional, segura e de fácil reprodução no serviço *GKE*. Apesar disso, foram identificadas oportunidades de otimização futura, nomeadamente a implementação de um *Autoscaler* para garantir a escalabilidade perante variações de tráfego, bem como a replicação da base de dados. Esta última medida revela-se fundamental para eliminar o atual ponto único de falha, aumentando assim a resiliência e a disponibilidade de todo o sistema.