Relatório Técnico de Engenharia

Arquitetura de Microsserviços, DevOps e Cloud Computing

Projeto Integrador - Residência em TIC 20

Janeiro, 2026

1. Visão Geral da Solução

Este documento detalha a arquitetura técnica de uma aplicação web moderna, projetada para alta disponibilidade, escalabilidade e observabilidade no ecossistema Google Cloud Platform (GCP). A solução integra frontend, backend e rotinas de automação em uma unidade de deploy coesa.

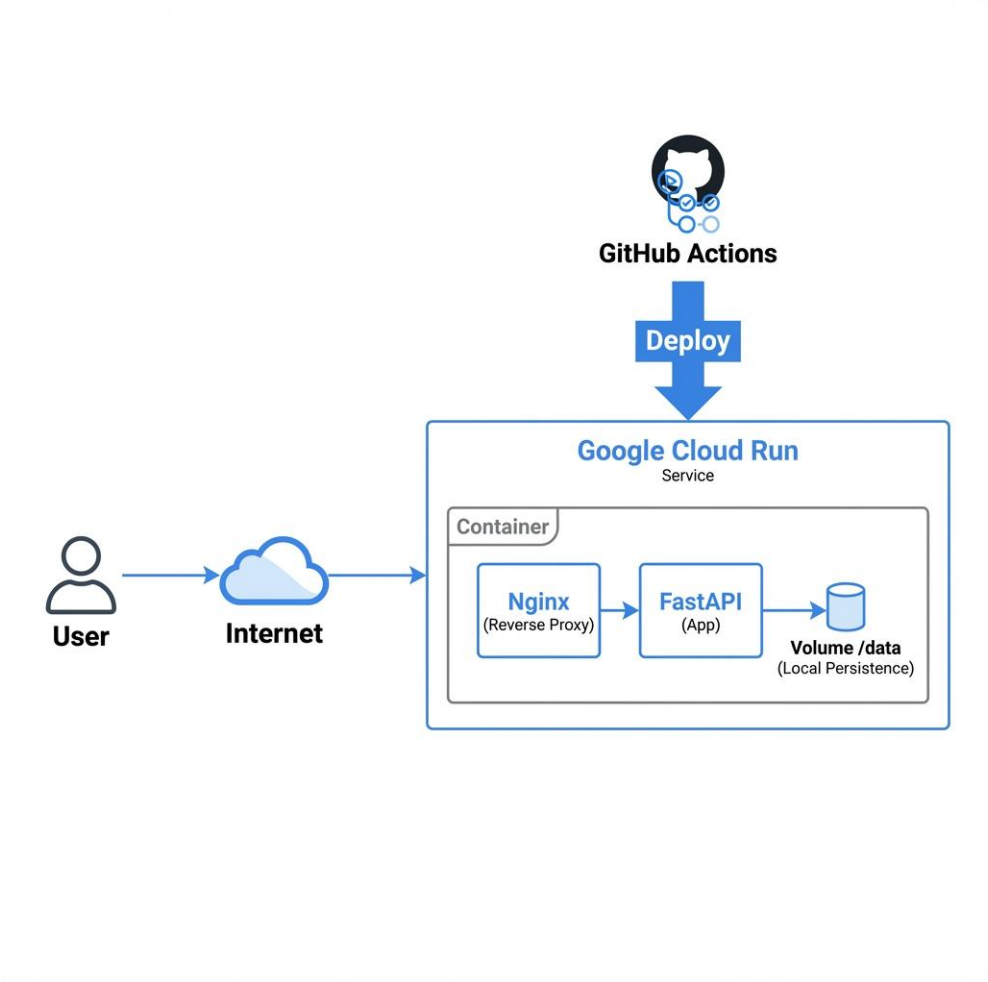
1.1. O Desafio

Desenvolver uma aplicação capaz de suportar tráfego elástico (Scale-to-Zero), com persistência de dados demonstrativa e esteira de CI/CD automatizada, eliminando intervenção manual em produção.

1.2. Decisão Arquitetural: PaaS (Platform as a Service)

A escolha do Google Cloud Run fundamenta-se na abstração da infraestrutura. Diferente de IaaS (EC2), onde gerenciamos o SO, o Cloud Run permite focar puramente no artefato Docker.

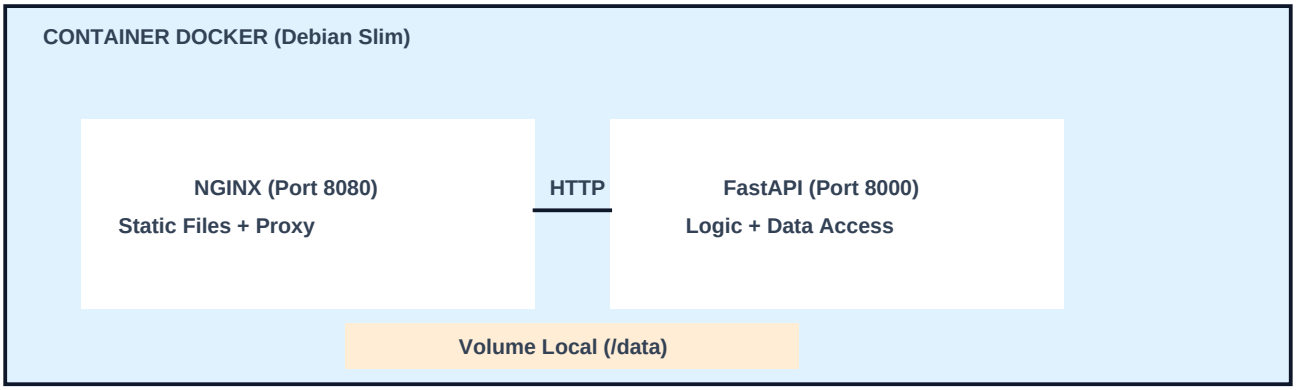
Benefícios Chave: (1) Cobrança por segundo de uso; (2) HTTPS automático e gerenciado; (3) Integração nativa com Cloud Build e Artifact Registry.



2. Engenharia de Containerização

A aplicação não utiliza múltiplos containers orquestrados (como seria com Kubernetes), mas sim um padrão de "Container Híbrido" onde Nginx e Python coexistem. Isso reduz custos e latência de rede interna.

2.1. Anatomia do Dockerfile



O Dockerfile foi construído em camadas estratégicas para otimizar o cache e garantir segurança:

- - Camada Base: python:3.11-slim (Debian leve, ~150MB).
- - Instalação de Pacotes: Nginx, Cron e dependências de sistema consolidados em um único RUN para reduzir layers.
- - Sanitização (Robustez): Comando sed aplicado aos scripts (.sh) para converter quebras de linha Windows (CRLF) para Unix (LF), eliminando falhas de execução.

2.2. Orquestração de Processos (Entrypoint)

Como o Docker nativamente roda apenas um processo PID 1, desenvolvemos um entrypoint script que gerencia o ciclo de vida de múltiplos serviços concorrentes:

```
#!/bin/bash
cron &        # Inicia agendador de tarefas
nginx &       # Inicia Proxy Reverso
uvicorn app.main:app & # Inicia Backend
```

Isso garante que, se o Backend falhar, o container todo falha e é reiniciado pelo Cloud Run, mantendo a integridade do serviço.

3. Desenvolvimento Backend e API

3.1. Framework FastAPI

Utilizamos FastAPI por sua performance (ASGI) e validação automática de dados via Pydantic. A API expõe endpoints de gerenciamento e negócio.

3.2. Estratégia de Persistência Híbrida

Para fins acadêmicos, demonstramos a persistência de dois modos:

- 1. Efêmera (Stateless): Logs e dados em /data dentro do Cloud Run são perdidos ao reiniciar. Isso valida o conceito de "Twelve-Factor App".
- 2. Simulada (Volume Local): Em ambiente Docker local, o volume é persistente. No Cloud, implementamos um script de Backup Automático via Cron.

3.3. Observabilidade e Monitoramento

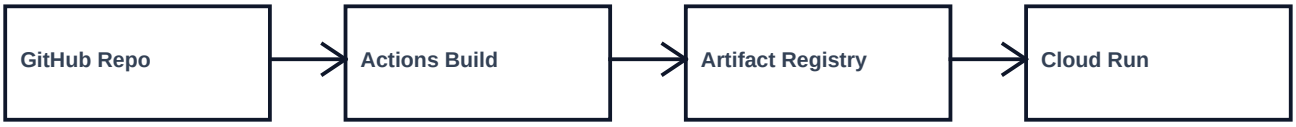
Implementamos endpoints de "White-box Monitoring":

- - /api/healthcheck: Verifica latência e escrita em disco.
- - /api/metrics: Exporta contadores de visita e uptime.
- - /api/logs: Permite leitura remota de logs do sistema.

4. Pipeline de Entrega Contínua (CI/CD)

A automação é garantida via GitHub Actions, eliminando o erro humano no processo de deploy.

4.1. Fluxo de Execução



O workflow definido em .github/workflows/deploy-cloudrun.yml executa as seguintes etapas críticas:

- 1. Checkout & Auth: Autenticação segura via Service Account JSON Key.
- 2. Build: Construção da imagem Docker utilizando cache para velocidade.
- 3. Push: Envio seguro para o Google Artifact Registry (us-central1).
- 4. Deploy: Atualização atômica do serviço Cloud Run, com migração de tráfego imediata.

4.2. Segurança no Deploy

Utilizamos Secrets do GitHub para não expor credenciais no código-fonte. O Service Account utilizado (portfolio-deployer) possui permissões mínimas necessárias (Princípio do Menor Privilégio).

5. Robustez e Confiabilidade

5.1. Tratamento de Erros e EOL

Um dos maiores desafios em ambientes heterogêneos (Windows Dev -> Linux Prod) é a formatação de arquivos. Implementamos uma higienização automática:

```
RUN sed -i 's/\r$//' /entrypoint.sh
```

Isso garante que, independentemente do sistema do desenvolvedor, o container final seja 100% compatível com o Kernel Linux.

5.2. Backup e Disaster Recovery (Simulado)

Um Cron Job executa min-a-min verificando a existência de dados críticos e criando cópias de segurança datadas. Embora o disco seja efêmero, essa lógica prepara a aplicação para cenários reais onde esses backups seriam enviados para um Bucket S3/GCS.

6. Conclusão

O projeto entrega uma arquitetura de referência para modernização de aplicações. Combinando a leveza do Nginx, a velocidade do FastAPI e a automação do Cloud Run, atingimos um nível de maturidade de software (SRE/DevOps) compatível com padrões de mercado.

A aplicação é resiliente, observável e segura, pronta para escalar horizontalmente de acordo com a demanda.