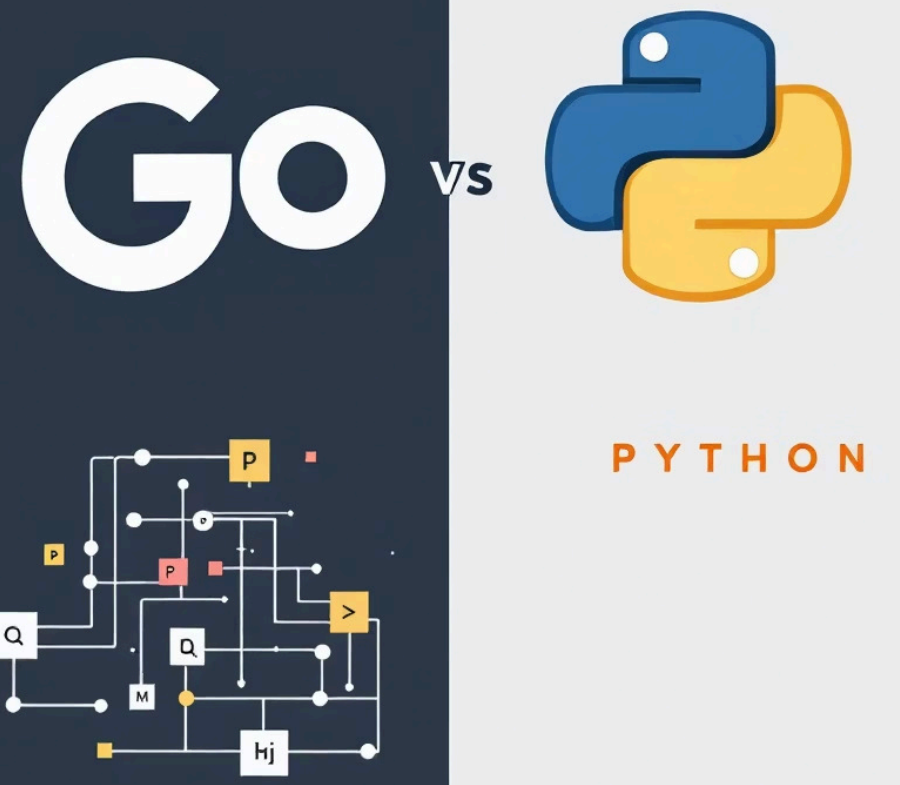


Go vs Python em Microsserviços

Demonstrativo de resultados de análise comparativa de desempenho e escalabilidade

Arquitetura do Sistema

1

API Go

Gin e GORM, imagem Docker Alpine (15MB), gestão manual de Connection Pool.

2

API Python

FastAPI, AsyncIO, asyncpg e SQLAlchemy Async para concorrência não bloqueante.

3

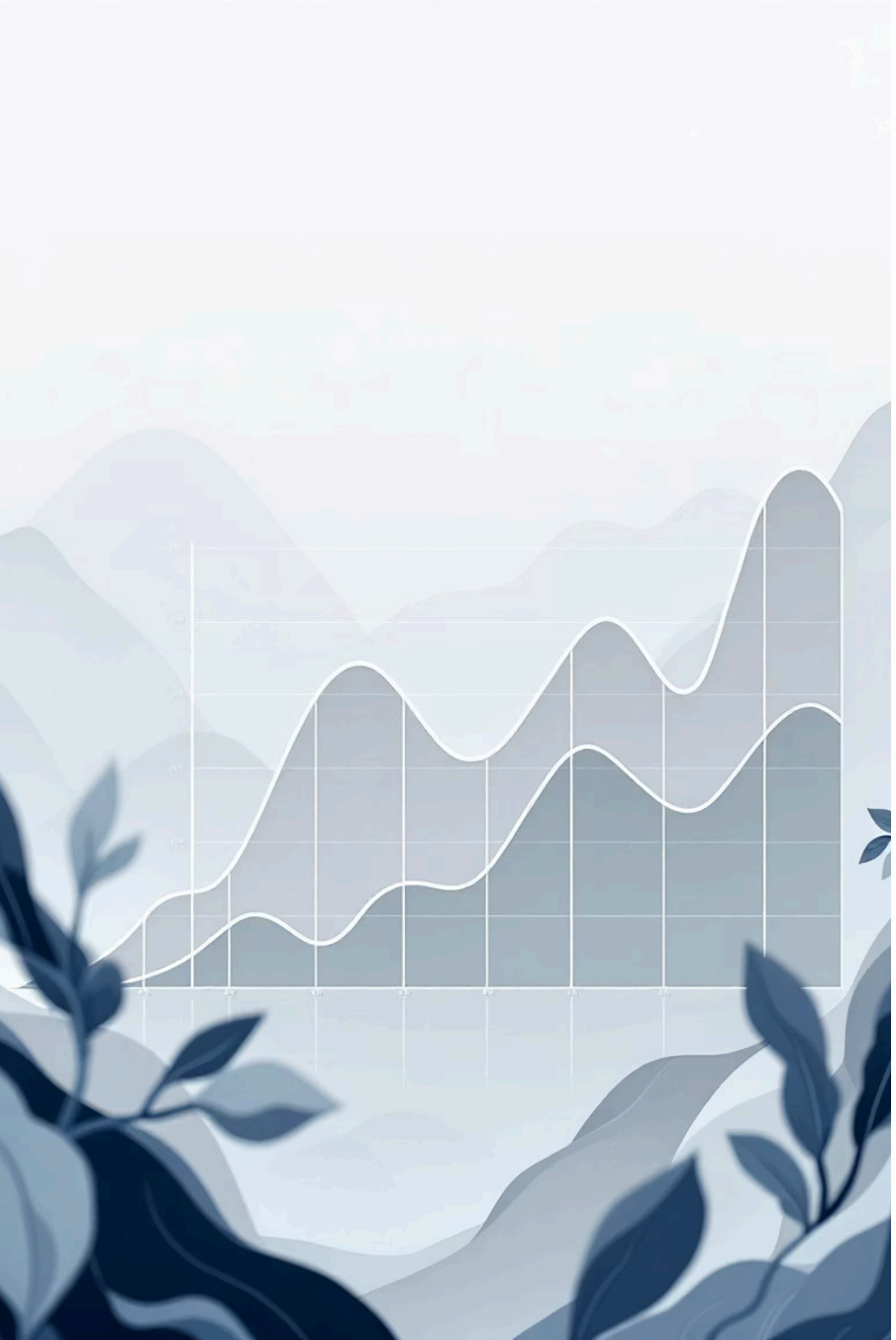
Gateway Nginx

Load Balancer, distribui tráfego via Round Robin entre réplicas das APIs.

4

Persistência

Banco de dados PostgreSQL



Metodologia de Teste

01

Ferramenta

Apache JMeter, simulando 1000 usuários virtuais simultâneos.

02

Fase 1: Identificação de Gargalos

Configuração inicial com 2 réplicas/API e PostgreSQL padrão. Colapso com 36,4% de erros (timeout de conexão).

03

Fase 2: Otimização

Aplicação de correções para mitigar problemas de desempenho.

Otimizações Aplicadas

Limite de conexões do PostgreSQL aumentado para 1000.

Número de réplicas das APIs dobrado (de 2 para 4 contêineres cada).

APIs reconfiguradas para manter mais conexões abertas (pool size aumentado).

Resultados: Antes e Depois da Otimização

Configuração	2 Réplicas / DB Padrão	4 Réplicas / DB Otimizado	-
Throughput Total	3.442 req/s	6.237 req/s	+81%
Taxa de Erro	36,4%	0,91%	-97%

O sistema eliminou erros e dobrou a capacidade.

Latência: Go vs Python

Go

Latência Média: 110 ms

Cerca de 50% menor que Python sob carga máxima, validando sua eficiência em memória e threads.

Python

Latência Média: 219 ms

Com asyncpg, sustentou a mesma vazão que Go, graças à arquitetura assíncrona.

Conclusão: Infraestrutura é Chave

Go: Vantagem em Latência

Ideal para cenários de altíssima performance.

Python: Alternativa Robusta

Viável com AsyncIO, desde que a infraestrutura seja bem dimensionada.