Rust Microservices Strategy

DataMigrate AI - Complete Guide

Author: Alexander Garcia Angus
Property of: OKO Investments

Executive Summary

This guide explains the hybrid FastAPI + Rust microservices architecture for DataMigrate AI. You do NOT replace FastAPI - instead, you ADD Rust microservices for performance-critical bottlenecks (20% of code) while keeping FastAPI for business logic (80% of code).

Key Benefits of Adding Rust

Benefit	Impact	When to Add
10x Performance	SQL parsing: 5s -> 500ms	User complaints about speed
50-70% Cost Reduction	$300/mo -> $120/mo at scale	AWS bills > $2,000/month
Competitive Advantage	10x faster than competitors	Need market differentiator
Better User Experience	Instant vs slow migrations	High user churn

Bottlenecks Rust Can Solve

1. SQL Parsing (BIGGEST)

Python: 5-10 seconds | Rust: 500ms (10-20x faster)

2. dbt Model Compilation

Python: 10-15 seconds | Rust: 1-2 seconds (5-10x faster)

3. Schema Validation

Python: 2-3 seconds | Rust: 200-300ms (10x faster)

4. Large JSON Parsing

Python: 1-2 seconds | Rust: 100-200ms (5-10x faster)

5. Bulk Data Transformation

Python: High memory | Rust: 10x less memory, 5x faster

Hybrid Architecture: FastAPI (80%) + Rust (20%)

KEEP in FastAPI: User auth, CRUD operations, API key management, LangGraph orchestration, business logic (changes frequently, I/O-bound)

MOVE to Rust: SQL parsing, dbt compilation, schema validation, bulk transformations (CPU-bound, performance-critical)

Cost-Benefit Analysis

Metric	FastAPI Only	FastAPI + Rust	Savings
Development	$0 (already built)	$6,000 (6 days)	N/A
Monthly AWS (10k migrations)	$300/month	$120/month	$180/month
Annual AWS Savings	$0	$2,160/year	$2,160/year
Payback Period	N/A	2 months	Break-even fast

Decision Framework: When to Add Rust

DON'T Add Rust If:

[X] Still in MVP (under 1,000 users)

[X] API response times are fine (<500ms p95)

[X] AWS costs are low (<$500/month)

[X] CPU usage is reasonable (<60%)

[X] Still adding features rapidly

DO Add Rust When:

[OK] Specific endpoints are slow (>1 second)

[OK] CPU usage is consistently high (>80%)

[OK] AWS costs are growing (>$2,000/month)

[OK] You've profiled and identified bottlenecks

[OK] Customers complain about performance

Implementation Roadmap

Phase	Timeline	Action	Expected Outcome
1. Profile	Week 1	Add profiling to FastAPI, identify slowest functions	Find bottlenecks
2. Build	Week 2-3	Build Rust SQL parser microservice	Working Rust service
3. Integrate	Week 4	Call Rust from FastAPI, add fallback	Hybrid working
4. Deploy	Week 5	Deploy to Kubernetes, monitor metrics	Production ready
5. Optimize	Week 6+	Fine-tune, expand to other services	Full optimization

Real-World Success: Discord

Before (2017): All Python, 100M users, $500k/year servers, slow message parsing

After (2020): Python (80%) + Rust (20%), 10x faster messages, 50% cost reduction ($250k/year savings)

Lesson: You don't replace Python, you augment it with Rust for bottlenecks!

Final Recommendation for OKO Investments

Phase 1 (Now - Month 12): Keep 100% FastAPI. Focus on features, get customers.

Phase 2 (Month 12-18): Add Rust for bottlenecks (SQL parsing, dbt compilation). Keep FastAPI for everything else.

Phase 3 (Month 18+): Expand Rust services as needed. Monitor cost savings.

Expected ROI: $6,000 investment, $2,160/year savings, 2-month payback period. Plus competitive advantage from 10x faster migrations.