Backend Language Comparison

DataMigrate AI: Go vs Python vs TypeScript vs Others

Author: Alexander Garcia Angus
OKO Investments
November 27, 2025

Executive Summary

This document provides a comprehensive cost and technical analysis of 6 backend language options for DataMigrate AI. The analysis includes 2-year Total Cost of Ownership (TCO), development time estimates, performance comparisons, and architectural recommendations.

Key Finding: Go (Golang) + Python hybrid architecture offers the best balance of performance (5-10x faster than Python), cost ($31,624 TCO), and development speed while preserving all existing LangGraph code.

Quick Recommendation

Scenario	Recommended Option	Why
Best overall choice	Go + Python	5-10x faster, only $1,954 more than pure Python
Fastest development	Python (FastAPI)	Keep what you have, lowest cost
TypeScript team	TypeScript + Python	Familiar syntax, full-stack JavaScript
Maximum performance	Rust + Python	10-20x faster, but 42% more expensive

2-Year Total Cost of Ownership (TCO)

All calculations assume a $100/hour development rate and AWS infrastructure costs for Production and Staging environments running 24/7.

Language	Initial Dev	AWS (24mo)	Maintenance	Total TCO	Performance
Python (FastAPI)	$2,550	$10,320	$16,800	$29,670	Baseline
Go + Python	$4,600	$10,224	$16,800	$31,624	5-10x faster
TypeScript + Python	$4,775	$11,472	$16,800	$33,047	~Python
C# (.NET) + Python	$6,475	$12,480	$18,000	$36,955	4-6x faster
Java (Spring) + Python	$7,125	$13,560	$19,200	$39,885	3-5x faster
Rust + Python	$8,800	$9,456	$24,000	$42,256	10-20x faster

Key Insight: Go + Python adds only $1,954 (6.6%) to total cost while delivering 5-10x better performance than pure Python.

Detailed Cost Breakdown

1. Python (FastAPI) - Baseline: $29,670

• Initial Development: $2,550 (25.5 hours)

• AWS Infrastructure: $430/month × 24 = $10,320

• Maintenance: $700/month × 24 = $16,800

• Performance: API latency 150-200ms, 1,000 req/s

2. Go + Python - RECOMMENDED: $31,624

• Initial Development: $4,600 (46 hours - includes Go API rewrite)

• AWS Infrastructure: $426/month × 24 = $10,224 (slightly cheaper!)

• Maintenance: $700/month × 24 = $16,800

• Performance: API latency 30-50ms (5x faster), 10,000 req/s

• LangGraph agents stay in Python (no rewrite needed)

3. TypeScript (NestJS) + Python: $33,047

• Initial Development: $4,775 (47.75 hours)

• AWS Infrastructure: $478/month × 24 = $11,472

• Maintenance: $700/month × 24 = $16,800

• Performance: Similar to Python (~150ms latency)

4. C# (.NET Core) + Python: $36,955

• Initial Development: $6,475 (64.75 hours - learning curve)

• AWS Infrastructure: $520/month × 24 = $12,480

• Maintenance: $750/month × 24 = $18,000 (higher due to complexity)

• Performance: 40-70ms latency, 5,000 req/s

5. Java (Spring Boot) + Python: $39,885

• Initial Development: $7,125 (71.25 hours - verbose code)

• AWS Infrastructure: $565/month × 24 = $13,560 (high memory usage)

• Maintenance: $800/month × 24 = $19,200

• Performance: 50-80ms latency, 4,000 req/s

6. Rust (Actix-web) + Python: $42,256

• Initial Development: $8,800 (88 hours - steep learning curve)

• AWS Infrastructure: $394/month × 24 = $9,456 (lowest due to efficiency)

• Maintenance: $1,000/month × 24 = $24,000 (expensive due to complexity)

• Performance: 20-30ms latency (fastest), 15,000+ req/s

Development Time Comparison

Time estimates for implementing the DataMigrate AI backend API with typical CRUD operations, authentication, and database integration. Does NOT include LangGraph agent development (which stays in Python for all options).

Task	Python	Go	TypeScript	C#	Java	Rust
Project setup	1h	2h	2h	3h	4h	4h
User auth + JWT	3h	5h	4h	8h	10h	12h
PostgreSQL setup	2h	3h	3h	4h	5h	6h
CRUD endpoints (5)	5h	8h	7h	12h	15h	18h
API key management	3h	5h	4h	6h	8h	10h
Error handling	2h	4h	3h	5h	6h	8h
Testing	4h	6h	5h	8h	10h	12h
Documentation	2h	3h	3h	4h	5h	6h
Deployment config	3h	4h	4h	6h	8h	10h
Learning curve	0h	6h	8h	10h	15h	20h
<b>TOTAL HOURS</b>	<b>25.5h</b>	<b>46h</b>	<b>47.75h</b>	<b>64.75h</b>	<b>71.25h</b>	<b>88h</b>

Key Insight: Go requires 80% more development time than Python (46h vs 25.5h), but this is a one-time cost. Rust requires 245% more time (88h).

Performance Comparison

Benchmarks for typical API operations (user auth, CRUD, JSON serialization). These metrics are based on industry standards and real-world production systems.

Metric	Python	Go	TypeScript	C#	Java	Rust
API Latency (p50)	150ms	30ms	140ms	50ms	60ms	25ms
API Latency (p95)	300ms	60ms	280ms	100ms	120ms	45ms
Memory per Pod	200MB	50MB	180MB	150MB	250MB	40MB
Concurrent Req/s	1,000	10,000	1,200	5,000	4,000	15,000
Startup Time	3s	100ms	2.5s	1.5s	5s	50ms
Container Size	400MB	20MB	350MB	200MB	500MB	15MB

Performance Winner: Rust offers the best raw performance (25ms latency, 15k req/s), but Go provides excellent performance (30ms, 10k req/s) at much lower development cost.

Go + Python Hybrid Architecture

CRITICAL CLARIFICATION: Your existing LangGraph agents stay in Python. Only the REST API layer moves to Go.

Traffic Distribution

Service	Handles	Traffic %	Language
Go API	POST /login, GET /migrations, POST /api-keys, All CRUD	95%	Go
Python Service	POST /run-agents, LangGraph orchestration	5%	Python

Communication Flow

1. User sends POST /api/v1/migrations to Go API
2. Go validates input and saves to PostgreSQL (15ms)
3. Go returns migration object to user immediately (30ms total)
4. Go spawns goroutine to call Python service asynchronously
5. Python service runs LangGraph workflow (5-30 minutes)
6. Python updates migration status to 'completed' in database

Why LangGraph Must Stay in Python

Reason	Explanation
No Go equivalent	LangGraph is Python-only. No native Go port exists.
Building from scratch	Reimplementing LangGraph in Go = 6-12 months of work
Your competitive advantage	Your existing Python agent code is battle-tested and working
Claude API integration	Anthropic SDK is Python-first, Go support is experimental
Checkpoint system	LangGraph state management is complex to replicate

Code Examples: Go Calling Python

Go API Service (Port 8000)

```
package main import ( "bytes" "encoding/json" "github.com/gin-gonic/gin" ) func
createMigration(c *gin.Context) { var req MigrationRequest c.BindJSON(&req;) // 1. Save
to PostgreSQL (Go is fast at this) migration := Migration{ Name: req.Name, Status:
"pending", Metadata: req.MetadataJSON, } db.Create(&migration;) // 2. Call Python
service asynchronously go callPythonAgents(migration.ID) // 3. Return immediately
(don't wait for agents) c.JSON(200, migration) } func callPythonAgents(migrationID
uint) { payload := map[string]interface{}{ "migration_id": migrationID, } jsonData, _
:= json.Marshal(payload) http.Post( "http://python-service:8001/run-agents",
"application/json", bytes.NewBuffer(jsonData), ) }
```

Python Agent Service (Port 8001)

```
from fastapi import FastAPI from agents.orchestrator import run_langgraph_workflow app
= FastAPI() @app.post("/run-agents") async def run_agents(request: AgentRequest): #
This is your EXISTING code - no changes needed! result = await run_langgraph_workflow(
migration_id=request.migration_id, metadata_json=request.metadata_json ) return
{"status": "success", "result": result}
```

Decision Framework

When to Choose Each Option

Choose This	If You...	Trade-offs
Python (FastAPI)	Want fastest time-to-market, lowest cost, no performance issues yet	Slower API responses, lower throughput
Go + Python	Want 5-10x better performance, planning to scale, only $2k more	80% more dev time, two services to maintain
TypeScript + Python	Have JavaScript/TypeScript expertise, want full-stack consistency	Performance similar to Python, higher AWS costs
C# + Python	Are in Microsoft ecosystem, have .NET expertise, need Windows support	25% more expensive, vendor lock-in
Java + Python	Are in enterprise Java shop, need JVM compatibility	34% more expensive, heavy memory usage
Rust + Python	Have Rust expertise, need absolute maximum performance (1M+ users)	42% more expensive, steep learning curve

Final Recommendation

For DataMigrate AI: Go + Python Hybrid Architecture

Why Go + Python wins:
1. Performance: 5-10x faster API responses (30ms vs 150ms)
2. Scalability: 10x higher concurrent request capacity
3. Cost: Only $1,954 more than pure Python over 2 years (6.6% increase)
4. Zero risk: Keep all existing LangGraph code in Python
5. Learning curve: Go is easier to learn than Rust (1-2 weeks vs 3-6 months)
6. AWS costs: Actually slightly cheaper due to Go's efficiency ($426/mo vs $430/mo)

What stays in Python:
• All 6 LangGraph agents (Assessment, Planner, Executor, Tester, Rebuilder, Evaluator)
• State management and checkpointing
• Claude API integration
• Agent orchestration logic

What moves to Go:
• REST API endpoints (login, CRUD operations)
• JWT authentication
• PostgreSQL queries for users, migrations, API keys
• API key validation
• Request routing

Implementation Timeline

Phase 1: Learning Go (Week 1-2)

• Complete "A Tour of Go" (8 hours)
• Build simple REST API with Gin framework (8 hours)
• Learn GORM (PostgreSQL ORM for Go) (6 hours)
• Practice goroutines and channels (4 hours)

Phase 2: Implement Go API (Week 3-4)

• Set up Go project structure (4 hours)
• Implement user authentication + JWT (8 hours)
• Build CRUD endpoints for migrations (8 hours)
• Add API key management (5 hours)
• Implement Go → Python service communication (5 hours)
• Write tests and documentation (8 hours)

Phase 3: Integration Testing (Week 5)

• Test Go API with existing Python service (6 hours)
• Load testing and performance benchmarks (4 hours)
• Fix bugs and optimize (6 hours)

Phase 4: Deployment (Week 6)

• Create Docker images for both services (4 hours)
• Set up Kubernetes deployment configs (4 hours)
• Deploy to staging environment (4 hours)
• Production deployment and monitoring (4 hours)

Total Timeline: 6 weeks (46 hours of work + learning time)

AWS Infrastructure Costs

Monthly costs for Production + Staging environments running 24/7. All options use the same database (PostgreSQL RDS) and cache (Redis ElastiCache).

Component	Python	Go+Python	TypeScript+Python	C#+Python	Java+Python	Rust+Python
ECS Fargate (API)	$180	$160	$200	$220	$280	$140
ECS Fargate (Agents)	$80	$80	$80	$80	$80	$80
RDS PostgreSQL	$70	$70	$70	$70	$70	$70
ElastiCache Redis	$30	$30	$30	$30	$30	$30
Application Load Balancer	$25	$25	$25	$25	$25	$25
NAT Gateway	$45	$45	$45	$45	$45	$45
Data Transfer	$20	$16	$28	$30	$30	$14
<b>TOTAL/MONTH</b>	<b>$430</b>	<b>$426</b>	<b>$478</b>	<b>$520</b>	<b>$565</b>	<b>$394</b>

Key Insight: Go + Python actually costs $4/month LESS than pure Python on AWS due to Go's smaller memory footprint. Rust is cheapest at $394/month, but the high maintenance costs ($24k over 2 years) make it the most expensive overall option.

Conclusion

After comprehensive analysis of 6 backend language options, **Go + Python hybrid architecture** emerges as the optimal choice for DataMigrate AI.

The Numbers:

• Total Cost: $31,624 over 2 years (only 6.6% more than pure Python)
• Performance: 5-10x faster API responses (30ms vs 150ms)
• Scalability: 10x higher throughput (10,000 vs 1,000 req/s)
• Development Time: 46 hours (vs 25.5h for Python)
• AWS Costs: $426/month (actually $4/month cheaper than Python)

The Benefits:

• Keep all existing LangGraph agent code (zero rewrite risk)
• Easier to learn than Rust (1-2 weeks vs 3-6 months)
• Independent scaling (scale API and agents separately)
• Future-proof for growth (handles 100k+ users easily)
• Production-ready (used by Docker, Kubernetes, Uber, Dropbox)

When to Reconsider:

If you reach 1M+ daily active users and need absolute maximum performance at scale, then revisit Rust. But for 99% of SaaS applications (including DataMigrate AI at current and projected scale), Go provides the best balance of performance, cost, and developer productivity.

Author: Alexander Garcia Angus
Company: OKO Investments
Date: November 27, 2025
Document: Backend Language Comparison for DataMigrate AI