ADR 000 - Adoption of Go Programming Language

Status: Accepted
Date: June 30th, 2025

Context

As our system has evolved, maintaining applications written in multiple languages (primarily Python and Node.js) has introduced increasing complexity in deployment, performance tuning, and code maintenance. The development team identified several issues:

- **Performance bottlenecks** in high-throughput services written in interpreted languages.
- **Inconsistent tooling and build pipelines**, increasing setup time and friction across teams.
- **Scalability concerns** in handling concurrent operations efficiently.

To address these challenges, we explored languages and frameworks that provide simplicity, performance, and strong concurrency support. Go (Golang) emerged as a candidate due to its growing ecosystem, strong community support, and alignment with cloud-native development patterns.

Decision

We will **adopt the Go programming language** for building new backend and command-line (CLI) applications.

Rationale

1. **Performance**: Go produces compiled binaries that run close to native machine speed, offering substantial improvements over interpreted languages.

2. **Concurrency**: Built-in goroutines and channels simplify writing concurrent code without third-party libraries or complex threading models.
3. **Simplicity**: Go's minimalistic syntax and standard library reduce cognitive overhead and speed up onboarding.
4. **Ecosystem Fit**: Go integrates seamlessly with modern infrastructure tooling such as Docker, Kubernetes, and Terraform.
5. **Maintainability**: Static typing, strong tooling support (e.g., `go fmt`, `go vet`), and consistent idioms promote maintainable and reliable codebases.
6. **Community and Longevity**: Go has strong community adoption in well-established companies (Google, Cloudflare, Uber), ensuring long-term stability.

Benchmarks and internal proofs of concept demonstrated **20–30% faster response times** in comparable services written in Go versus Python.

Implications

People / Training

- Developers will need foundational Go training (expected ramp-up time: ~2 weeks).
- Pair programming and code review guidelines will be adjusted to promote Go best practices.

Process Adjustments

- CI/CD pipelines will be updated to build and test Go binaries.
- Introduce `golangci-lint` for linting and `go test` coverage metrics in CI workflows.

Tooling

- Adopt official Go toolchain, `gofmt`, and version management tools (e.g. `asdf` or `gvm`).
- Evaluate dependency management using Go modules.

Risks

- Initial learning curve for team members unfamiliar with Go.
- Limited third-party libraries compared to Python ecosystem.

- Possible short-term dip in delivery velocity during the transition phase.

Trade-offs

Benefits:

- Simplifies service development and builds consistent performance characteristics across backends.
- Reduces deployment complexity (static binaries, fewer runtime dependencies).
- Improves system scalability and observability through clearer concurrency patterns.

Drawbacks:

- Requires investment in training and documentation.
- Limited support for some high-level frameworks and specialized libraries currently available in Python or Node.js.
- Differences in error handling philosophy (no exceptions) may require mindset adjustment.

Key Evaluation Metrics

- 20% improvement in service latency and throughput.
- Measurable reduction in runtime memory footprint.
- Faster CI/CD pipelines due to static builds.
- Positive developer satisfaction reported after 3 months.
- Successful delivery of at least one production-grade service written in Go.

Conclusion

Adopting Go as a primary backend and CLI development language aligns with our strategic goals for scalability, simplicity, and maintainability. While onboarding and training represent an initial investment, the long-term benefits in performance, reliability, and developer productivity outweigh the short-term transition costs.

References

- Internal POC results comparing Go vs. Python microservice performance.
- Go Official Documentation
- Effective Go
- Cognitect: Documenting Architecture Decisions (Nygard, 2011)
- misterGF: Architecture Decisions Are Design in Motion