Lufthansa Industry Solutions

Coding Smarter for a Greener Future
Comparison Java vs. Go

Berlin | 13.11.2025

Think digital. Act digital. Together.

[LHIND.de]

1
Intro

2
Project

3
Environmental Impact

4
Business Case

01 Intro

Person

Moritz Bölter

Senior Consultant Cloud Native Development

Training

- Dual studies in Applied Computer Science at Nordakademie Elmshorn
- Master studies in Practical Computer Science at Fernuniversität Hagen

Experiences

- TechLead EPACTO
- Greening of IT projects
- Release Integration & DevOps
- Backend developer
- Technical consultant for an ERP system

Emphases

- Greening of IT topics and projects
- Sustainability Ambassador
- Backend development with Java and Go

Why does sustainability in software matter?

Software impacts **energy usage** through CPU, memory, and network resources.

Lower energy usage → **lower costs** and carbon footprint.

Companies and customers increasingly **care about sustainability**.

Sustainable software development is becoming a **key business priority**.

Quick Overview: Go vs Java

Go

Designed for simplicity, fast startup, efficient memory use, compiled to native code

- Developed by Google (2009)
- Lightweight, compiled to native code
- Smaller developer community
- Minimalist Standard Library
- Low memory and CPU overhead
- Designed for minimal footprint
- Built-in concurrency model (goroutines, channels)
- Use cases:
  - Simple, fast, scalable APIs or data processors
  - Cloud-native apps, microservices
  - Real-time systems, streaming data, telemetry agents

Java

Mature, very powerful, garbage collected, JVM-based, usually heavier

- Developed by SUN Microsystems (1995) – now owned by Oracle
- Robust, platform-independent
- Mature, widely used
- Full-featured libraries for almost anything
- Runs on JVM - overhead, higher resource consumption (often)
- Write once, run anywhere via the JVM
- Strong OOP and design pattern culture
- Use cases:
  - Large, enterprise-grade applications
  - Cross-platform desktop or mobile apps
  - Heavy ecosystems and integrations

02 Project

Overview

Invoice tool

Approving and processing of invoices by defined rules and workflows

Customer A	Customer B
Decided to implement the solution using **Golang**.	Decided to implement the solution using **Java** (Spring Boot)

Platform Overview

Frontend

Invoice Tool

Service: Data in

- Focus on automated management of incoming data
- Direct integration with external systems (e.g., SAP)
- **No manual uploads or user interaction** required
- Automatic identification of relevant users
- **Notifications** triggered for required user actions
- **Streamlined end-to-end process** for data intake and user engagement

Service: Data out

- Manages outgoing data
- Prepares and sends data to external systems (currently SAP)
- Ensures **proper formatting** and protocol compliance
- Enables **seamless integration** and data sharing

Service: Rules in

- Reads **cost center and bank account data** from external systems (currently SAP)
- Acts as a mapping table to **assign data to users and invoices**
- **Validates and organizes** incoming **data**
- Ensures accuracy, consistency, and seamless system integration

Middleware

- Core of the system's business logic
- Applies user actions, data processing, and rules correctly
- Coordinates **data exchange** between components
- Interfaces (via HTTPS) for financial thresholds
- Uses **SMTP** to notify users about actions and updates

Libraries/Frameworks

Golang

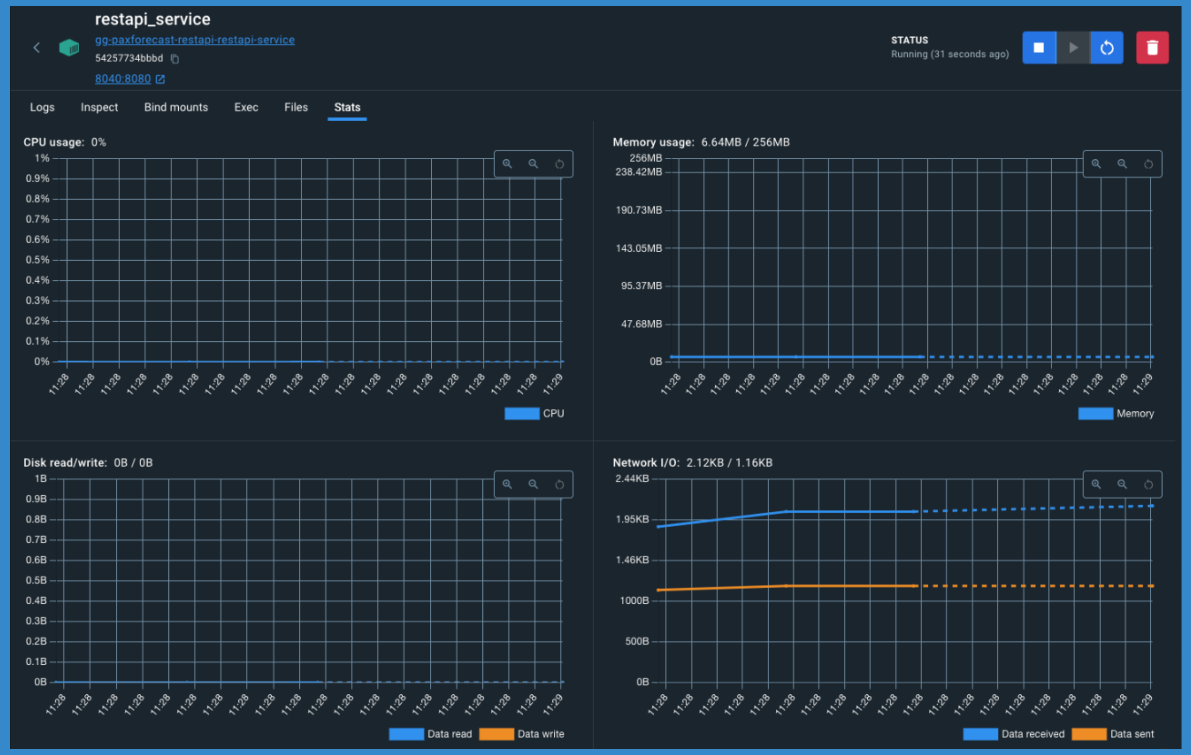
- go-sql-driver/mysql
- gookit/config/v2
- coreos/go-oidc
- gorilla/handlers
- gorilla/mux
- jmoiron/sqlx
- samber/slog-multi
- guregu/null/v6

Java

- Spring boot
  - Web
  - JPA
  - Actuator
  - Mail
  - Oauth2
  - Security
  - Webflux
- MySQL Connector
- Jsch
- Jakarta Mail

Application start

Go



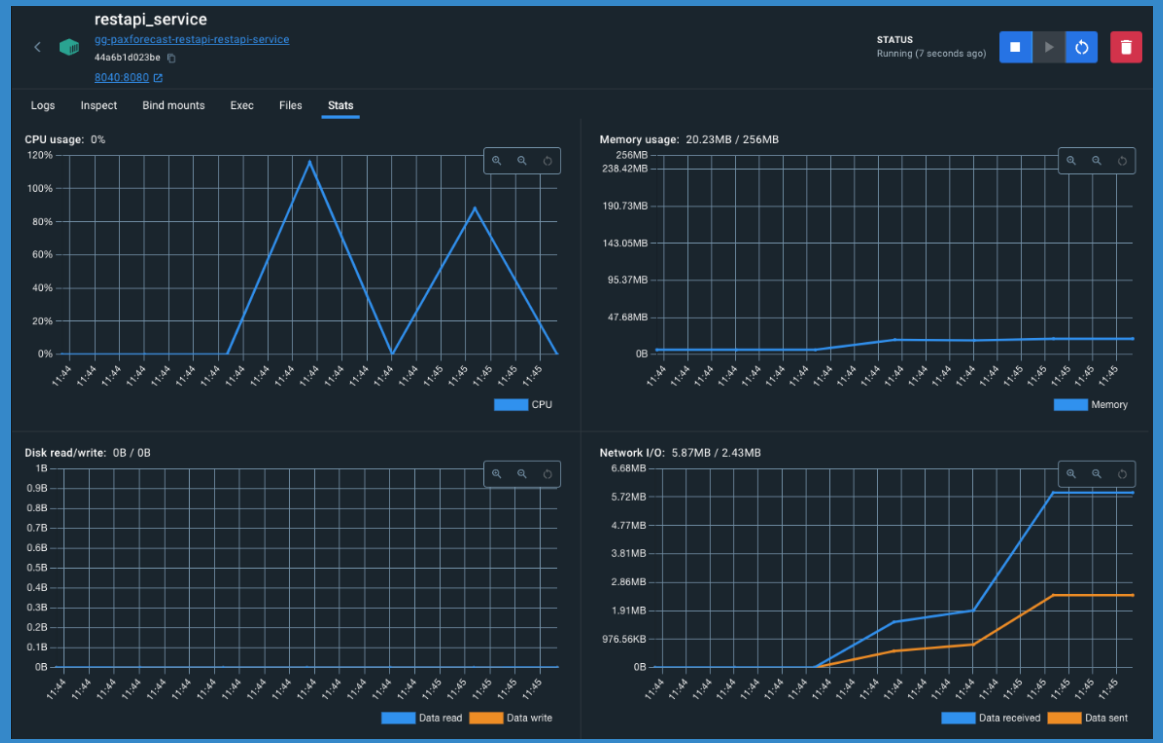
CPU: 0% | **Memory**: 6Mb

Java

CPU: 450% | **Memory**: 215Mb

Stress Test

Go



CPU: 82% | **Memory**: 20Mb

Java

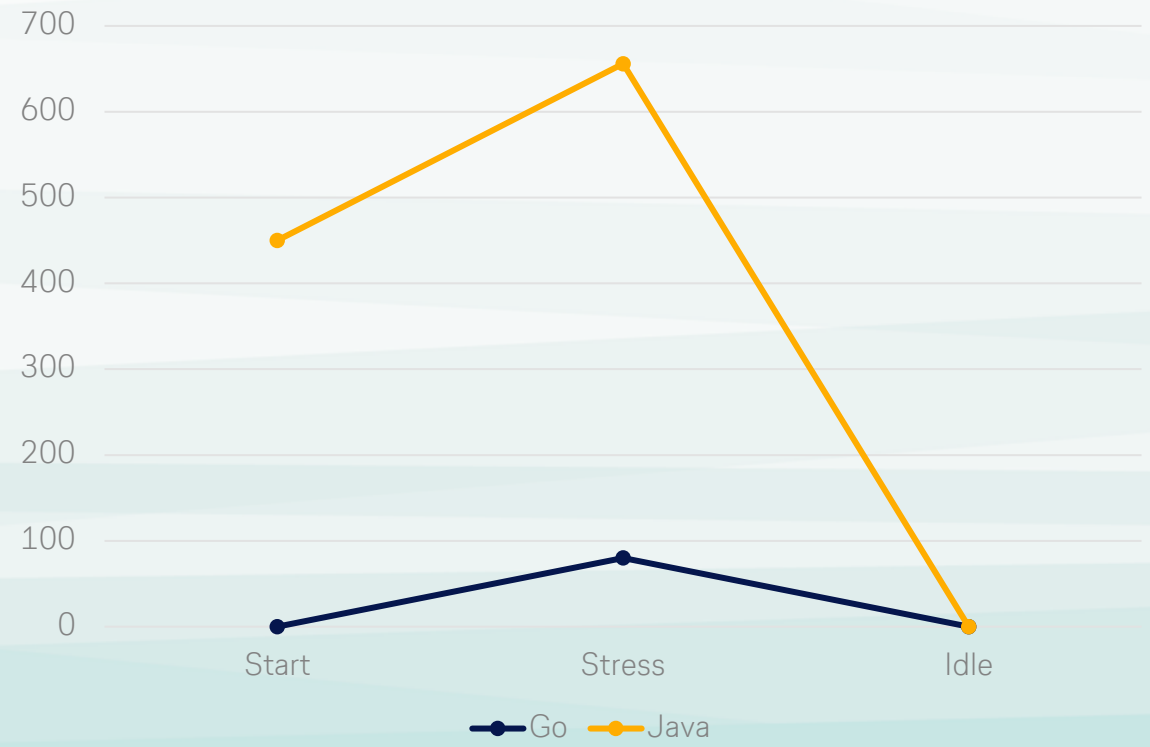
CPU: 576% | **Memory**: 570Mb

** 200 concurrent requests | Same functionality*

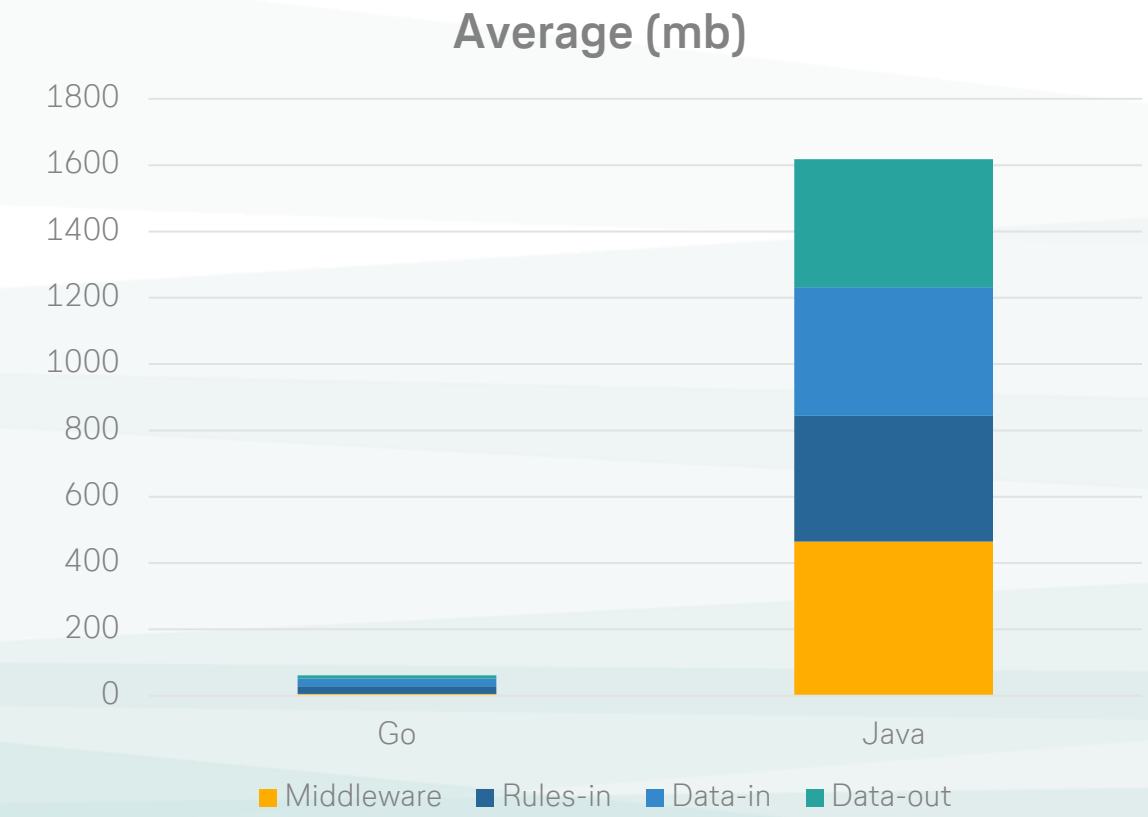
Comparison

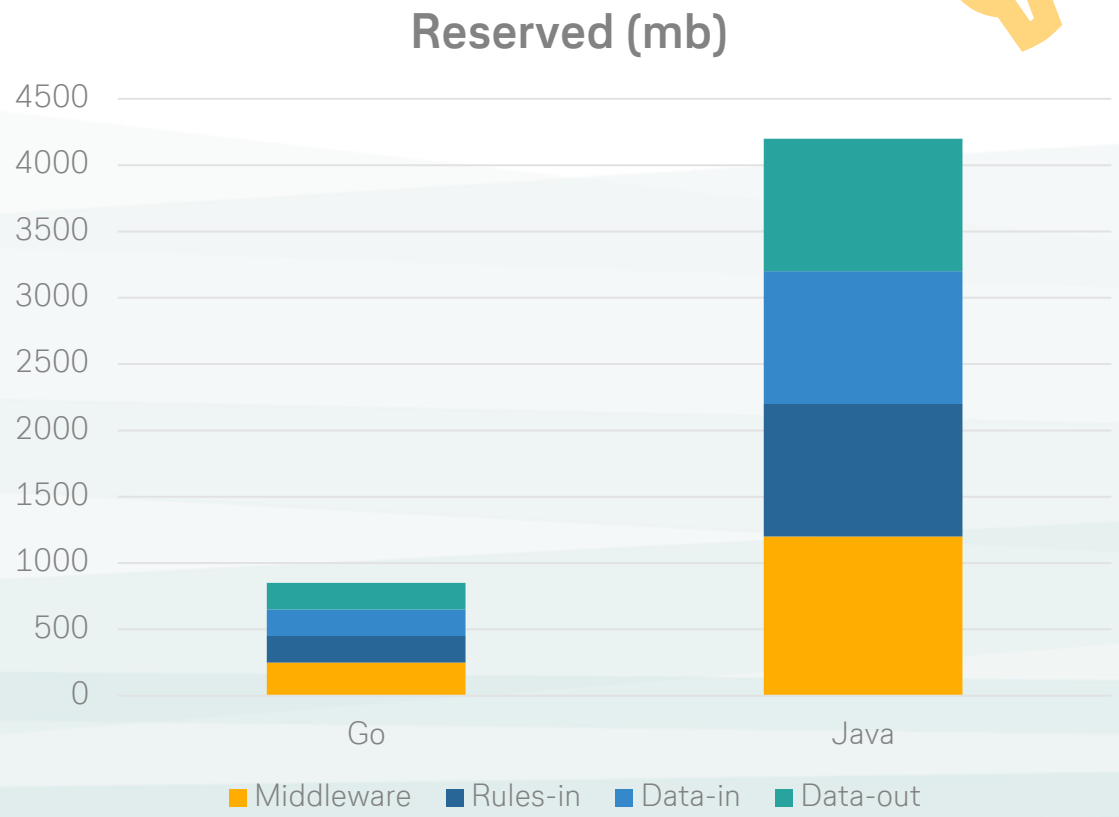
Memory

CPU



Memory usage

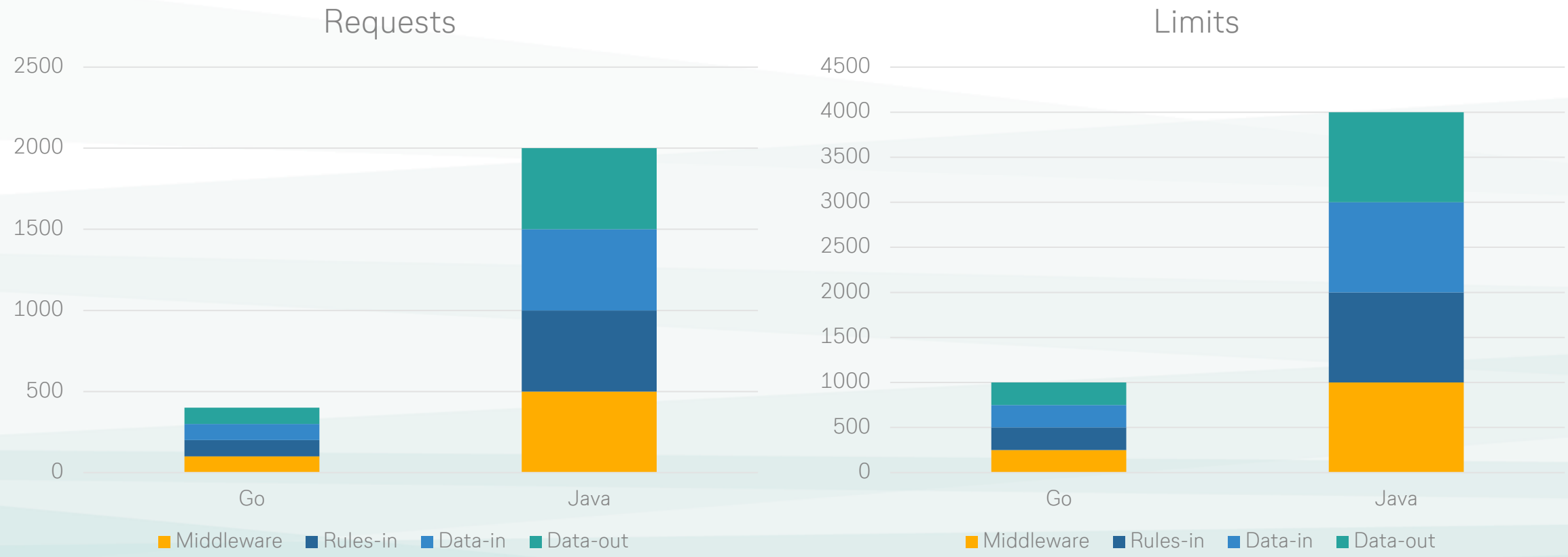




What is usually used in real life.

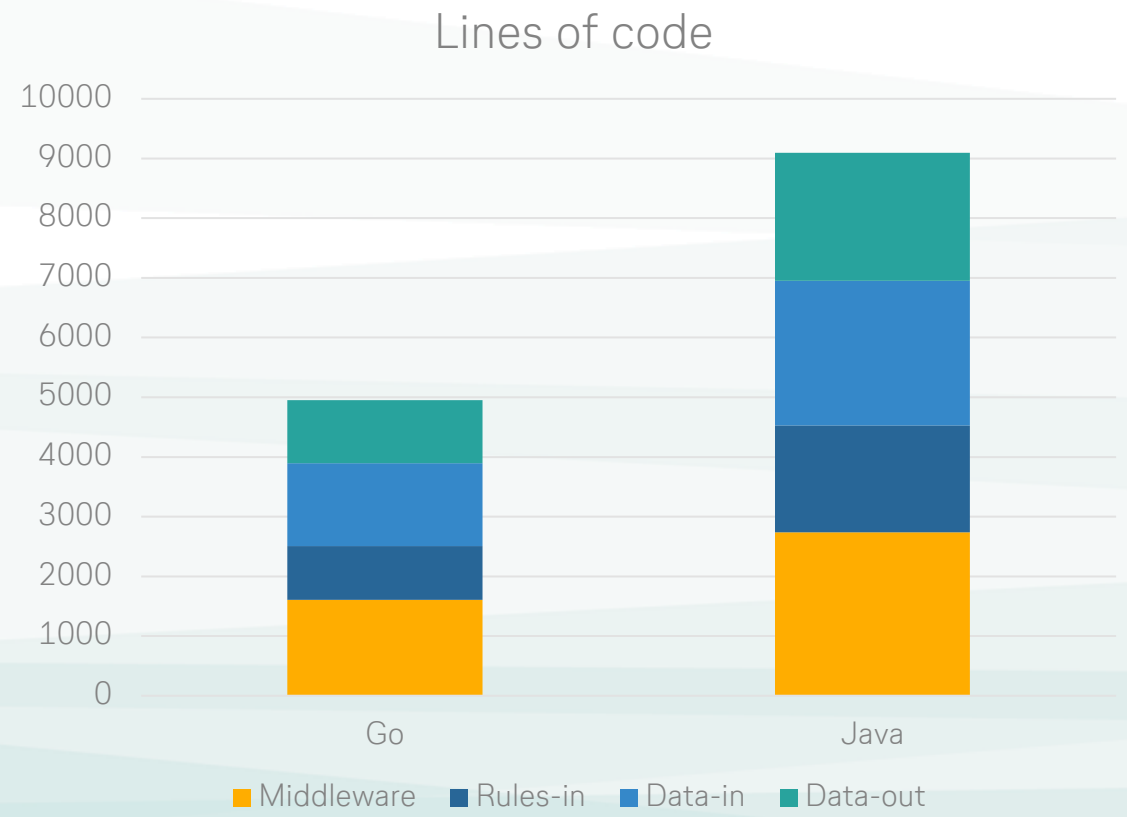
Maximum possible memory usage. Reserved previously by the system.

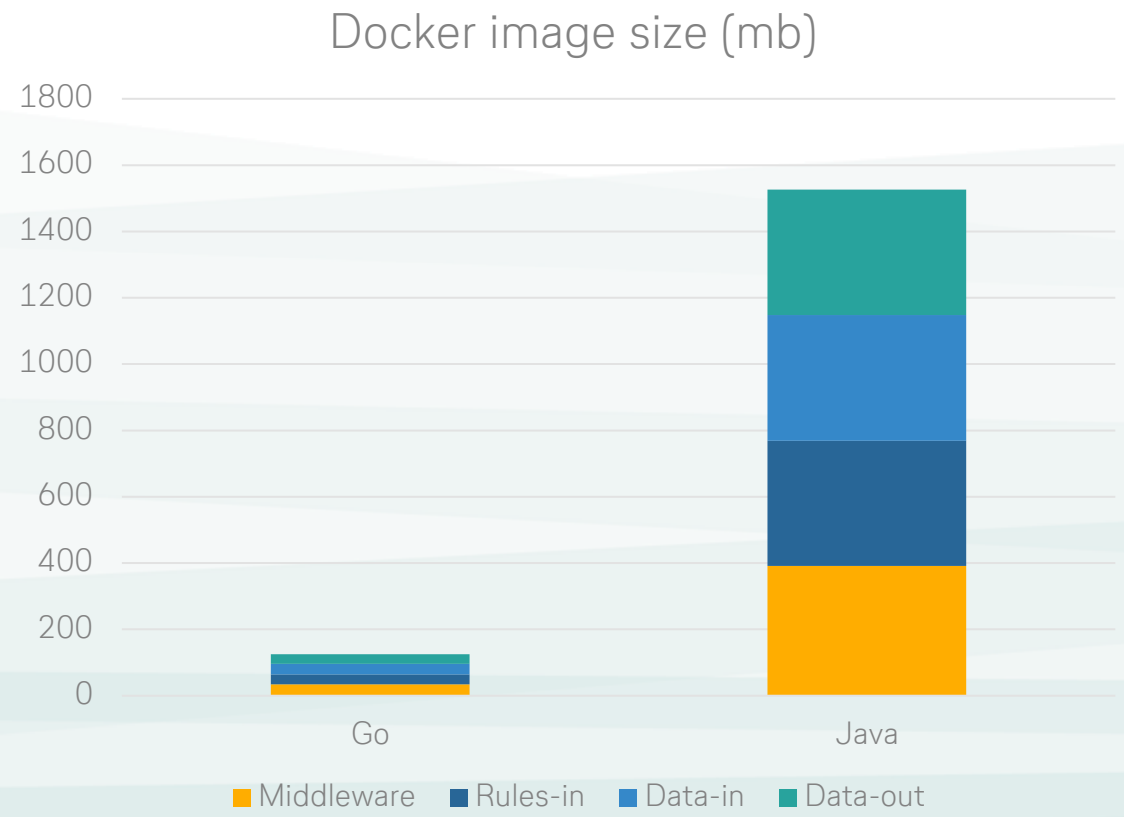
CPU usage



As the load increases, Java's resource usage grows significantly.

Development

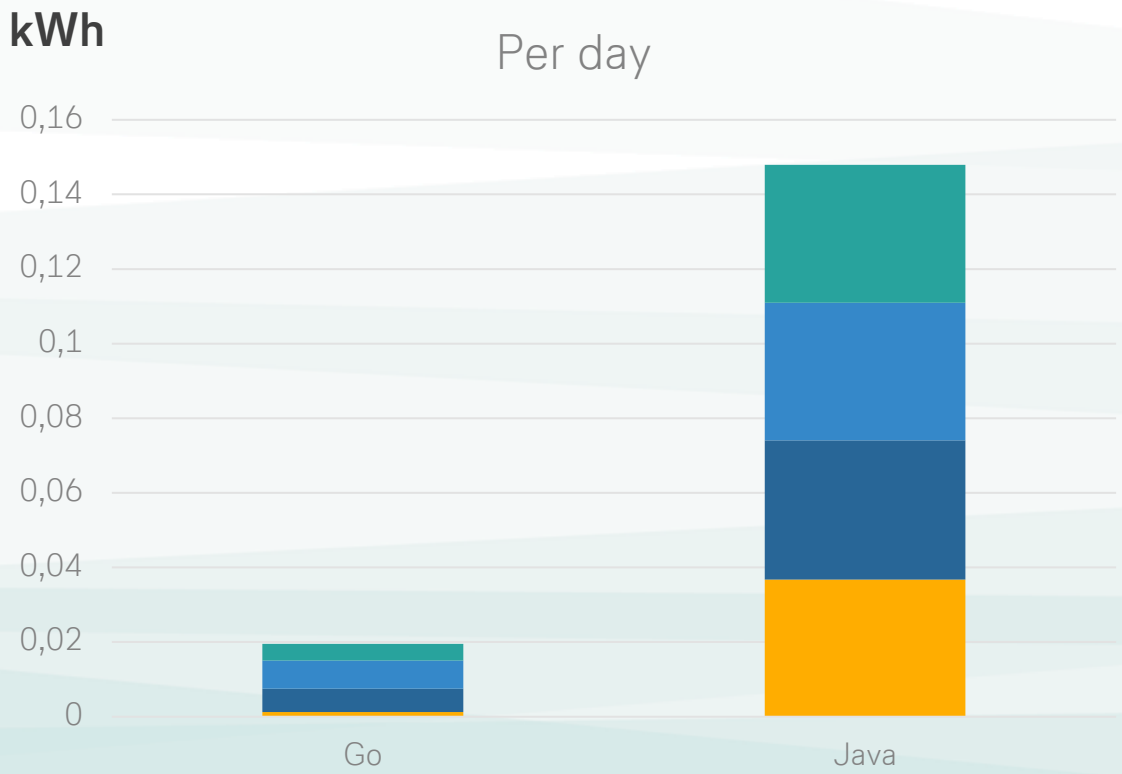


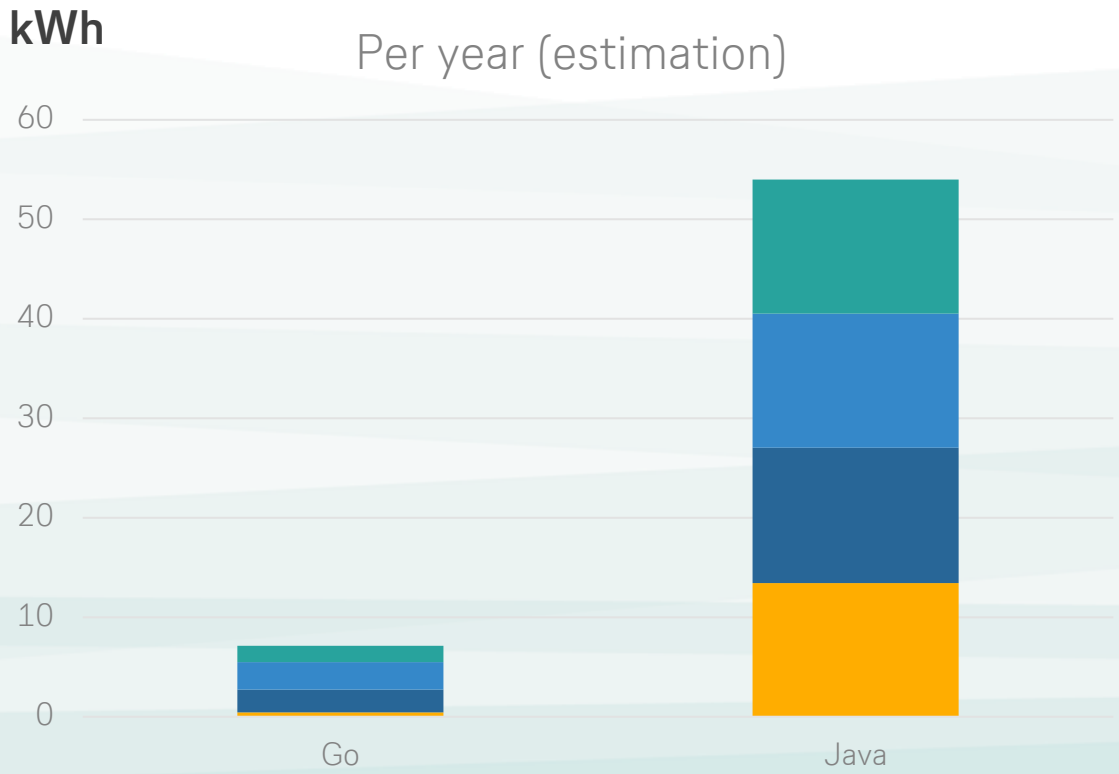


Go achieves the same functionality with far fewer lines of code and a Docker image size nearly 90% smaller than Java.

03 Environmental Impact

Power Consumption



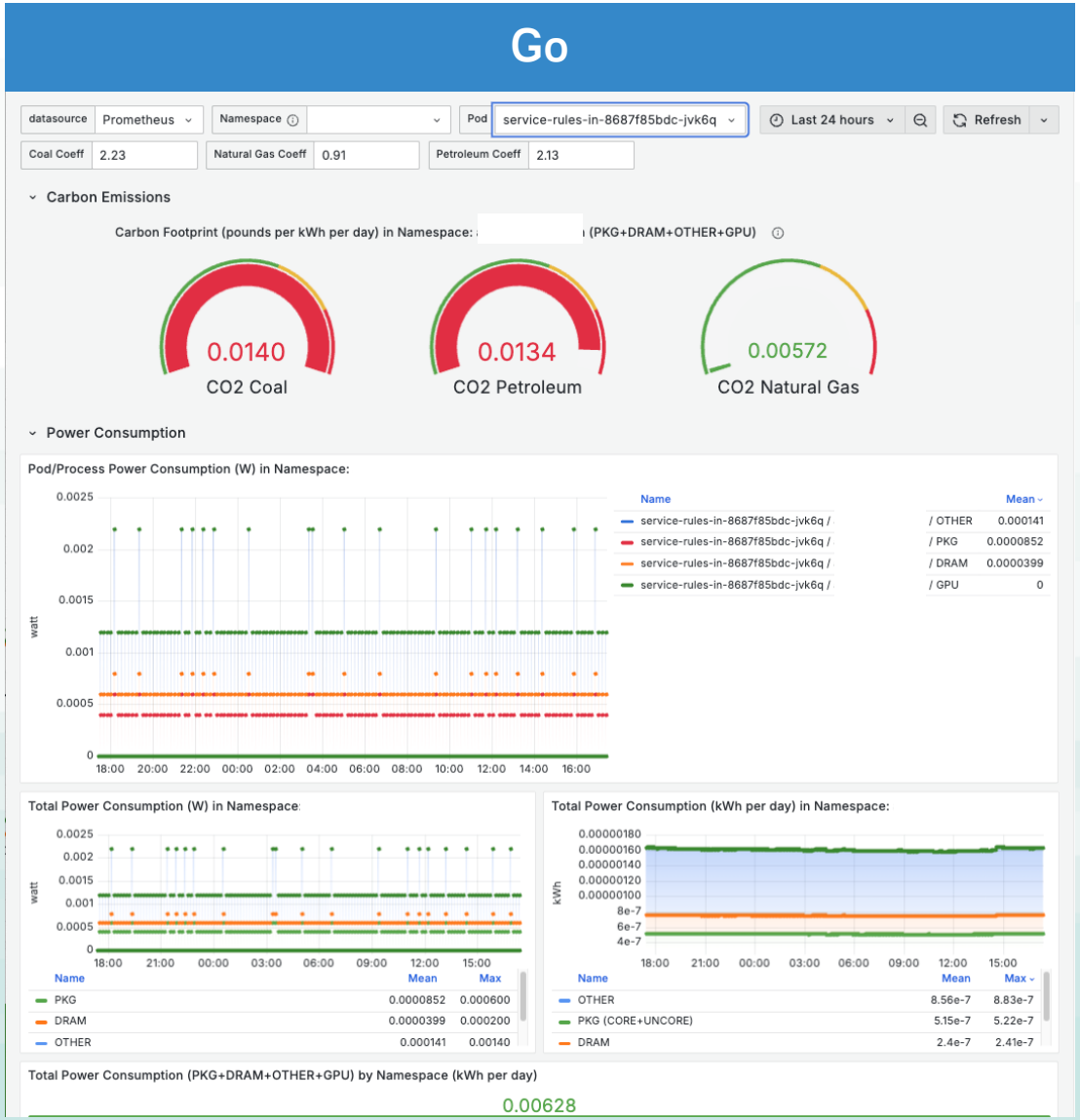


54 kWh is the average energy consumption for a refrigerator and freezer together over 27 days.

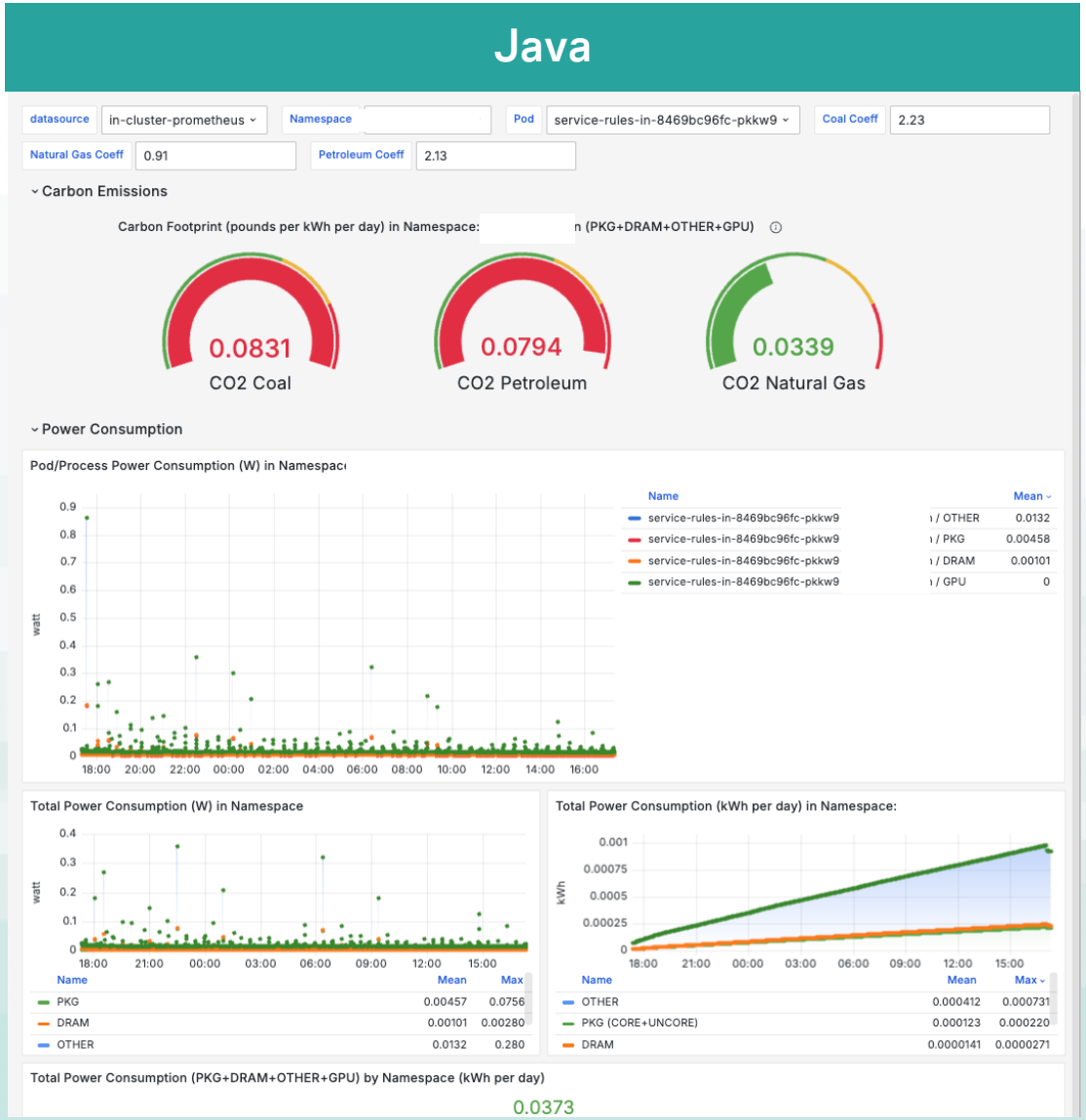
Metrics by kepler

CO_2 emissions per service & day (estimation)

Go



Java



Metrics by kepler

04 Business case

Business Case

	GO	Java
Power Consumption (per year)	~7kWh	~54kWh
CO2 emissions (per year)	~3kg	~22kg
Costs	~$0.0624/hour	~$0.2496/hour
CPU requirements (per environment)	1 Core	4 Cores
Memory usage	100MB~950MB	1.6GB~4.1GB
Processing time (per request)	15ms	78ms

Go is 3x **faster**.

Go is 6x more **sustainable**.

Go **codebase** is smaller and simpler.

Go saves 75% in infra **costs.**

Go uses 4x less **cpu.**

Go uses 4x less **memory.**

Lufthansa
Industry Solutions

THANK you

Questions?

Think digital. Act digital. Together.

[LHIND.de]