
G
O
D
O
O
O
O
D

Choosing a language →

Requirements →

Comparing use cases -

- Restful API's - Preferable with options to generate server code from swagger
- Consume messages from one of the queue/ streams like Kafka/SQS

Working with Following resources →

- MySQL
- Consuming & producing messages from/to Kafka at high volume
- Consuming and Producing messages from/to SQS
- Redis
- DynamoDB
- SDK for AWS services

Performance →

- Latency for RESTFUL API services
- Throughput For Stream Consumer
- Memory Footprint

Devops - Monitoring, Dockerization

Developer productivity -

- Language features
- available talentpool & trainability
  - IDE, Readibility & maintainability of existing code
- Code-test cycle

Comparing go & Java

Comparing use cases →

1) Restful API Server →
   - well supported platform → Restful API
   - Swagger codegen project go & Java

- Java -
  - Use annotations for configure - Aspect of API - endpoint, method series - (Request/Response) Format
- Go - explicit code for routing
- annotations - validate requer in Java for Go needs hand-coded

2) Consume messages from kafka/SQS
- both have numerous libraries

Java
- A lot options available - Concurrent Programming
- Availability of threads at core → A lot of utilities available for managing threads, distribute work across them, flexible synchronization option
→ Amount of complexities these libraries/utilities brings in, → very common for many applications to suffer greatly from race condition

Golang -
- Has simple model for concurrency based on goroutines (lightweight threads) & channels (a blocking queue)
- Result are more predictable & less buggy

→

3) Working with data-store -

Java -

- libraries tries to hide details of underlying implementation with a custom interface eg - JPA - has defined own query language

Go libraries - Simple wrappers on top of underlying system.

Dev's need to learn details (SQL)

- learning curve - deeper with libraries which hide details, debugging is hard
- Java libraries are well documented.
- Go libraries are pretty simple - going through the code & understanding is fairly simple.

4) Performance →

JVM - has been around 20 years

- lot of tuning available → which turn amazing performance

→

We can choose -

- Garbage collector - tuned for real time
- low-latency workload
- background high throughput workload

Basic setting gives decent performance

GC →

A single Garbage collector algorithm → highly optimized for very low latency & GC pauses in the order of microseconds

Memory footprint →

eg - Real time Notification Sever →

Some process running around 70 MB per process in production.

With Java pretty much Nothing runs for less than 512 MB of memory

This is beneficial → Docker allowing multiple services running around same machine.

⊛ Throughput side - both wins

Devops →

Java has various tools for monitoring

NewRelic → Supports aspects of applications like HTTP requests & DB queries

Go sends runtime metrics → StatsD, InfluxDB.

Developer productivity →

Java - Complexity has been increase with added libraries.

Go - Simple

Biggest Missing feature in Go → "Generics"

→ Utility functions have to be repeated many times

- Go - Error handling →

Errors are returned as part of methods return value.

if err != nil - boilerplate everywhere

- Go has "panic's" - which will bring whole process down if unhandled.

- Recommedation → is to Bring Process down that is scary (o)

Biggest advantage - Simplicity of Concurrent Programming

Lambda implementation of Java is a hacky at best.

Go supports - "functions as first class citizan"

- Passing function as parameter
- Defining anonymous functions

feels natural.

* Implicit interface implementation → allows elegant solutions.

→ IDE's & Tools →

Java - IntelliJ

Go - GoLand - tool chain hard to match

Java - Static code analyzer - enforce coding guidelines & flag some potential bugs

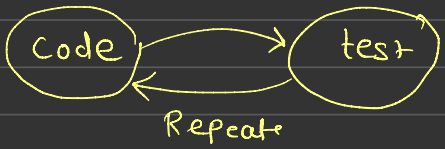
→ could cause exception like - NullPointerException

Go does this much better → Compilation will fail if variable is defined but not used

Many such a checks done by go compiler - go toolchain have option - to automatically format code on save.

Readability & maintainability of existing code → Go wins.

Code - test - cycle



Go code - unbelievably fast.

"No bootstrapping delay" → as go compiles into machine code.

Conclusion →

Go

- If majority of MS to be simple & single-purposed → use Go
- Simple to learn

Java

- If service is complex with lot of DB tables & numerous API endpoints, we feel - Productivity boost comes from libraries like JPA