Successful Go for microservices architecture

Hervé Ah-Leung

smartnumbers

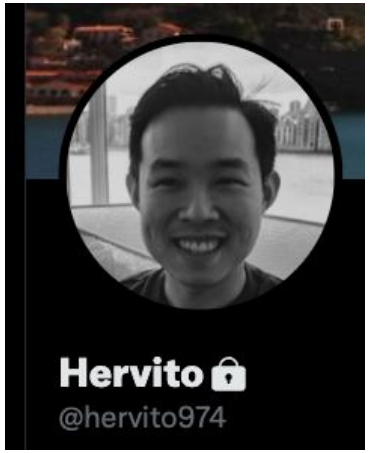
Who?

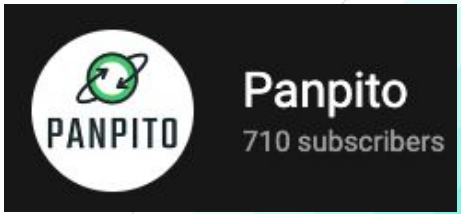
Who am I?

Senior Software engineer in London

Smartnumbers since 2019

(https://smartnumbers.com)





Microservices?

Monolith vs Microservices

Monolith

- Gigantic mono-service
- Single database
- Unique repo

Microservices

- Multiple autonomous meaningful services
- With their own databases
- In different repos

> difficult to extend/maintain

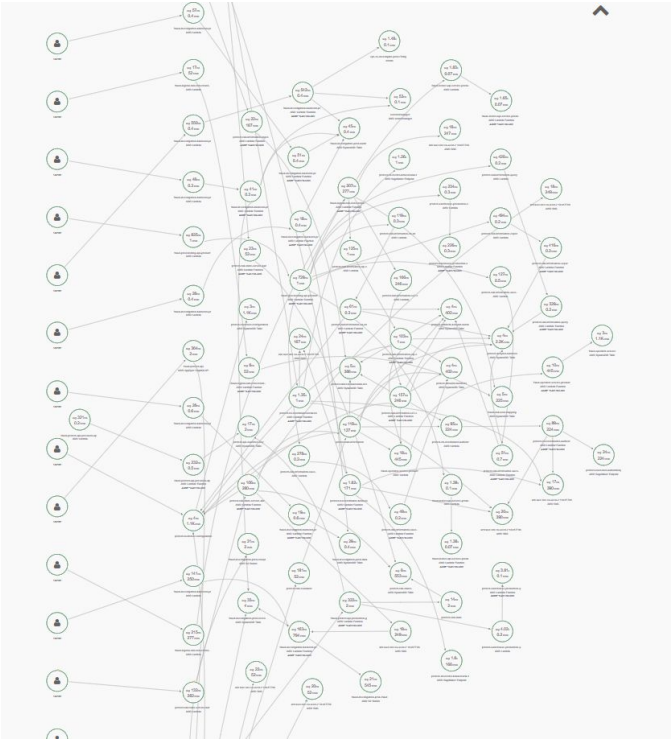
> often poor dev experience

> scalable and easy to extend

> better dev experience

Microservices disadvantages/challenges

Service map AWS X-Ray (March 2022)



Microservices disadvantages/challenges

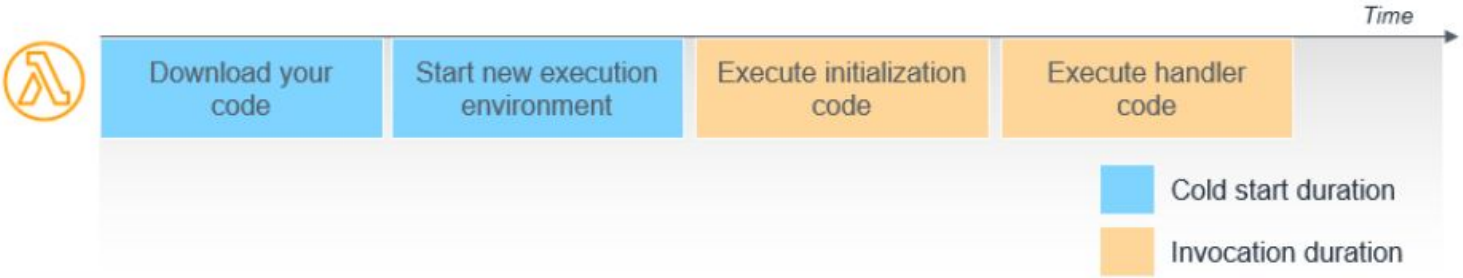
- Complexity of communication between services
- Increase of latency if not careful
- Easily extendable/maintainable? yes and no

> Any programming language could help solving/mitigating those problems, but how do we make the most of Go to tackle them?

Let’s take full advantage of Go!

Amazing serverless experience

Cold start problem 3 years ago with Scala with serverless functions



Amazing serverless experience

Cold start depends on:

- the language
- the package size
- If inside a private network or not

Thorough article:

https://mikhail.io/serverless/coldstarts/aws/

Amazing serverless experience

Observations:

- awful cold starts with Scala (Java): 1 to 2s
- much better with Go: ms

Learning

- Go binaries are lightweight, therefore the provisioning time is small
- First success for our microservices for latency

A well-supported and popular language

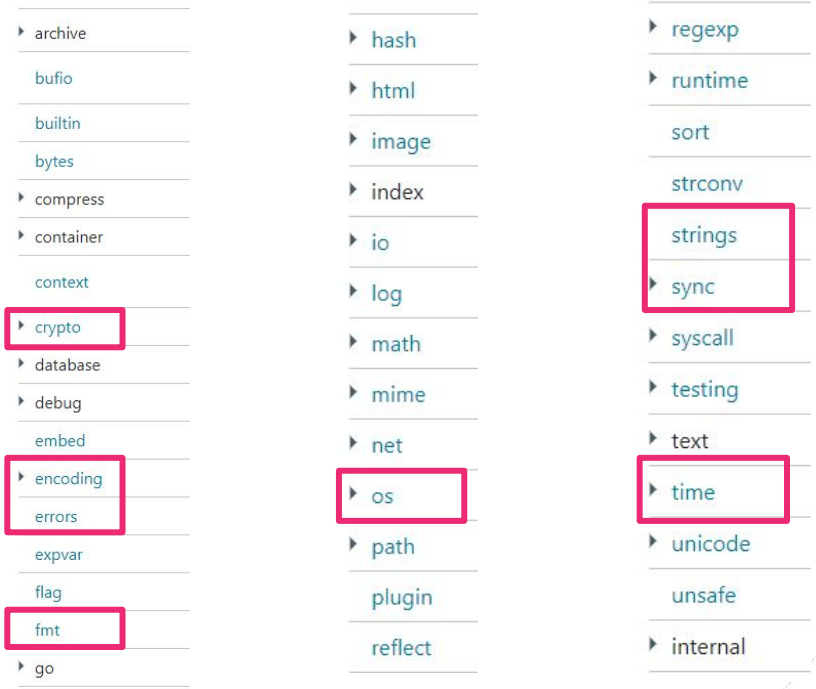
- Often Go SDKs available (AWS, GCP, Azure...)
- Plenty of tutorials available
- Living ecosystem and supportive community
- Living language (releases/improvements)

Learning

- Popularity of language massively helped to build smoothly the microservices

Excellent tooling

- Standard libraries!



Excellent tooling

- Dependencies management with inbuilt **Go modules** (we started with **dep**)

dep

Dependency management for Go

VS

Go Modules Reference

Table of Contents

Excellent tooling

Testing

- **gomega**: matcher/assertion lib
- **ginkgo**: BDD test framework

Libraries

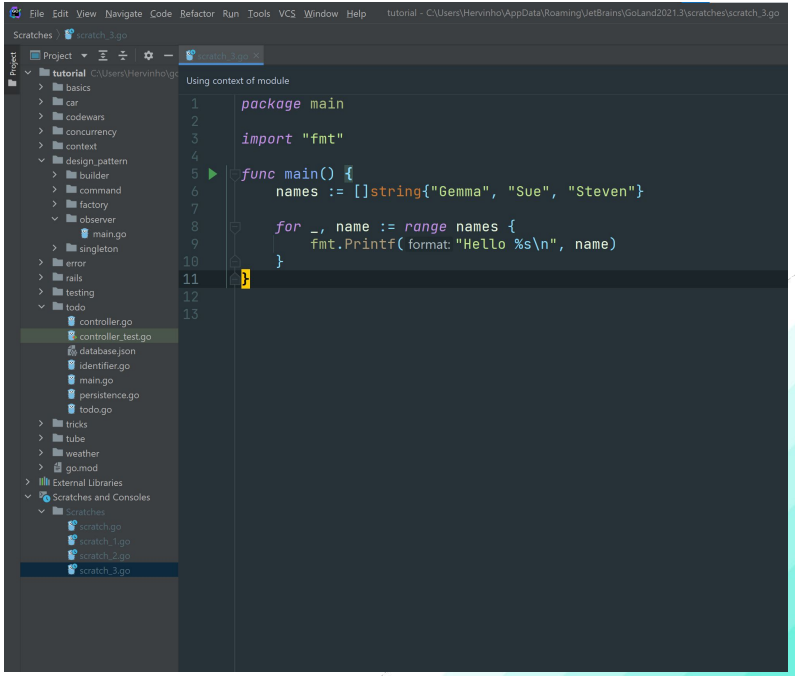
- **goReleaser**: binaries builder

GraphQL

- **gqlgen**: graphQL server generator

Excellent tooling

Quick special mention to Goland



Excellent tooling

Learning

- Key libraries/tooling make the dev experience solid
  - Faster to build microservices
  - Reliable quality
  - Writing Go is enjoyable

New joiner experience

- Relatively easy to learn
  - A few keywords
  - Explicit friendly syntax
- Room to improve if challenges are needed
  - Garbage collection
  - Concurrency model
- A junior member even did a presentation to the whole engineering department about Go

A tour of Go
go.dev/tour

Go by example
gobyexample.com/

New joiner experience

Learning

- Nice dev experience for a junior software engineer
- Rewarding learning curve
- Nice experience as a mentor
- Can focus on the microservices instead of language details

Any limitations?

Folders and packages organisation

Idea of contexts (not Go context...)

```
- controller
    - customer_controller.go
    - order_controller.go
    - sales_controller.go
- services
    - payment.go
    - shipment.go
    - authentication.go
    - refund.go
    - ...
- views
    - ...
- models
    - ...
```

```
- order
    - service.go
    - controller.go
    - request.go
- customer
    - authentication.go
    - model.go
- refund
    - service.go
    - form.go
```

Folders and packages organisation

Learning

Organising in context helps to identify common packages and extract them as common libraries.

Tooling then do the rest.

Specificities of the language

Not necessarily impacting a microservices architecture, but worth mentioning

- Pointers
  - Null pointer exception leading to actual panic

- Interfaces
  - Implicit interfaces concept sometimes difficult to understand for a beginner

Next steps?

Wider Go usage?

As a Go software engineer

I want to write my **infrastructure** in Go

I want to write my **deployment pipeline** in Go

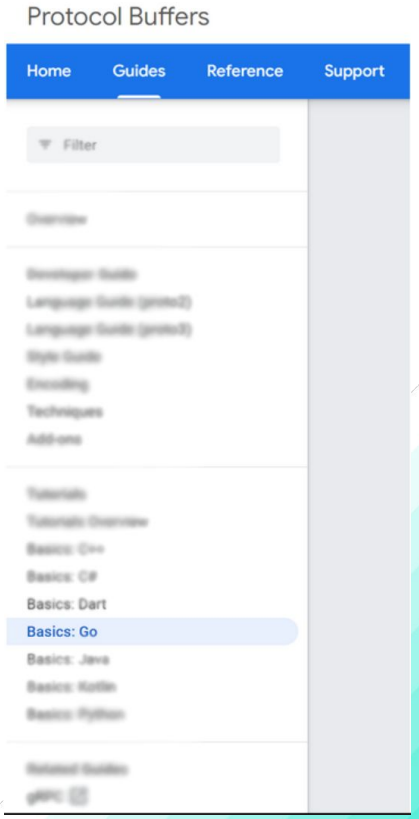
I want to write my **tasks runner** in Go

I want to write Go to generate my **documentation**

So then I don’t write awful YAML

Protobuf?

- Structure data serialisation



Generics?

Go 1.18 is released!

The Go Team
15 March 2022

Today the Go team is thrilled to release Go 1.18, which you can get by visiting the download page.

Go 1.18 is a massive release that includes new features, performance improvements, and our biggest change ever to the language. It isn't a stretch to say that the design for parts of Go 1.18 started over a decade ago when we first released Go.

Generics

In Go 1.18, we're introducing new support for generic code using parameterized types. Supporting generics has been Go's most often requested feature, and we're proud to deliver the generic support that the majority of users need today. Subsequent releases will provide additional support for some of the more complicated generic use cases. We encourage you to get to know this new feature using our generics tutorial, and to explore the best ways to use generics to optimize and simplify your code today. The release notes have more details about using generics in Go 1.18.

In conclusion?

Wrap up

- Main issues of a microservices architecture
  - Complexity of communication between services
  - Latency between services
- Go helps to ease the pain points
  - Excellent performances
  - Excellent tooling
  - Good popularity
  - Excellent dev experience
- Probably more Go features to come to build amazing software!

Thank you!