



M I C R O N A U T™

A new way to build microservices

Luram Archanjo

# Who am I?



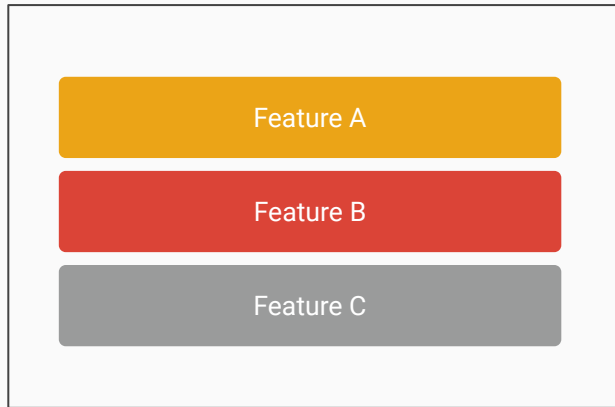
- Software Engineer at Sensedia
- MBA in java projects
- Java and microservice enthusiastic

# Agenda

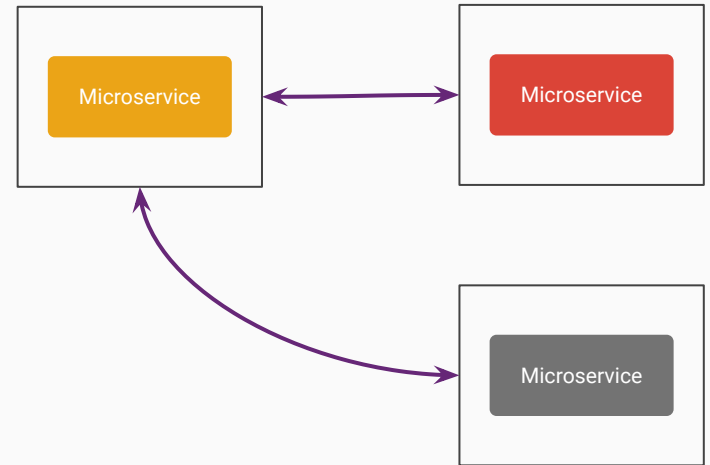
- Microservices
- Java & Frameworks
- Ahead of Time (AOT) Compilation
- GraalVM
- Micronaut
- Questions

# Moving to Microservices

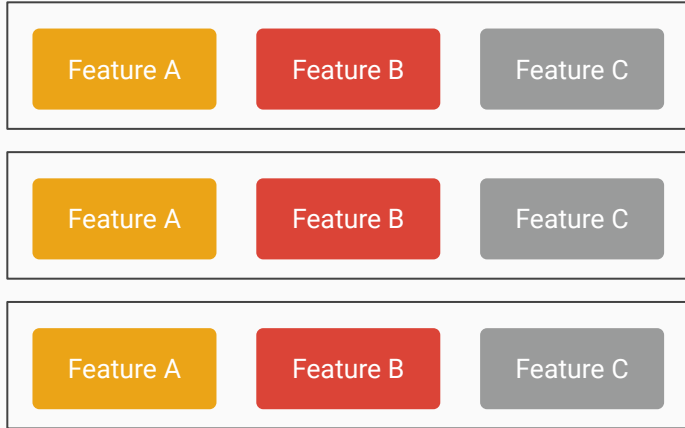
## Monolith



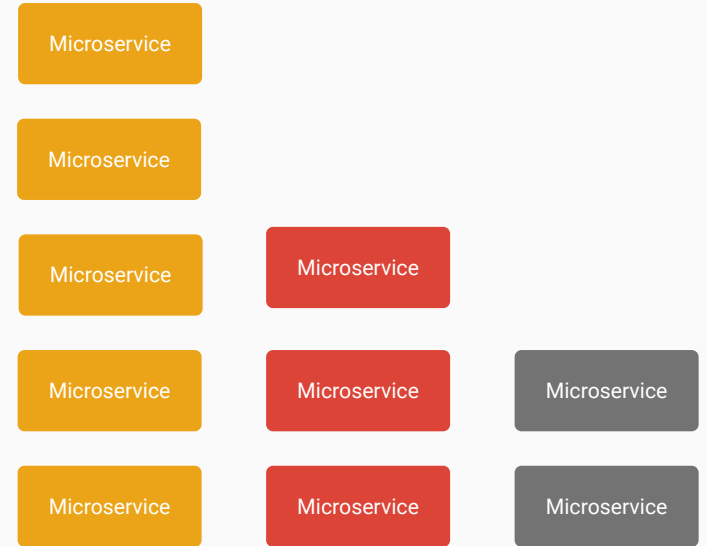
## Microservices



## Monolith Scalability



## Microservices Scalability



Our resources are finite!

How to use less resources  
using Java language?

Our frameworks are design to  
low memory footprint?



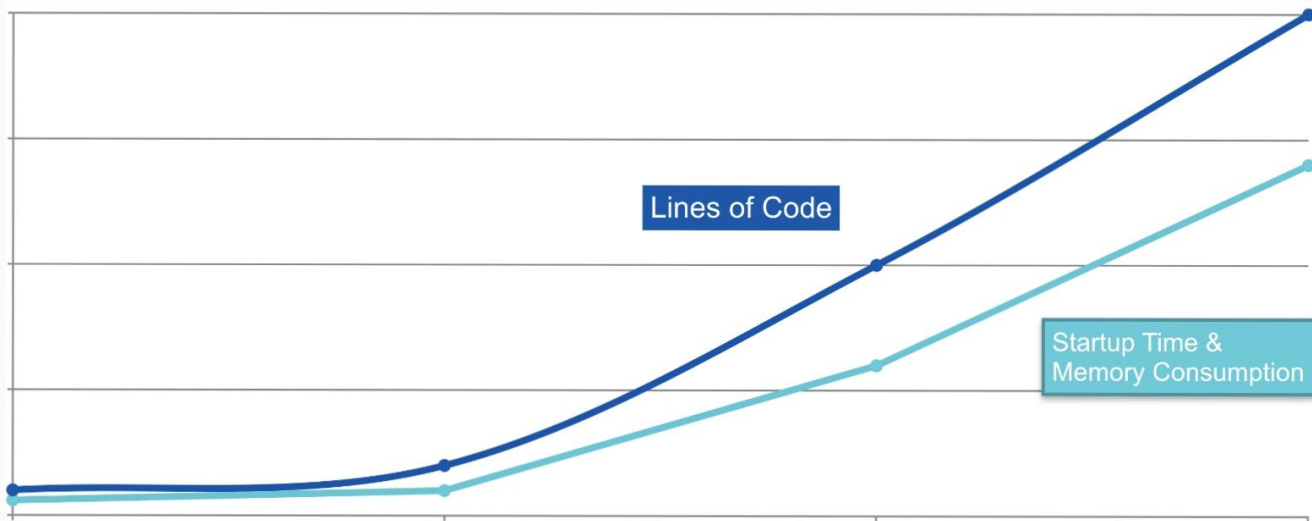


No, because we've tried to  
adapt existing legacy  
technologies for  
Microservices

# What do Spring and Jakarta EE undertaking? What are the results about it?

Spring is an amazing technical achievement and does so many things, but does them at **Runtime**.

- Reads the byte code of every bean it finds.
- Synthesizes new annotations for each annotation on each bean method, constructor, field etc. to support Annotation metadata.
- Builds Reflective Metadata for each bean for every method, constructor, field etc.



# The rise of Java Microframeworks

# Microframeworks

A microframework is a term used to refer to minimalistic web application frameworks:

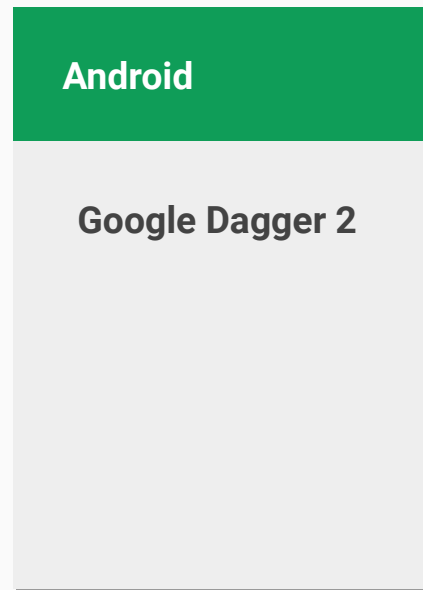
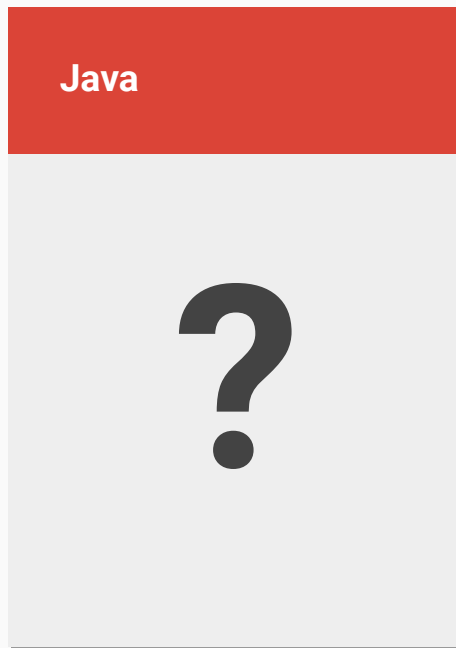
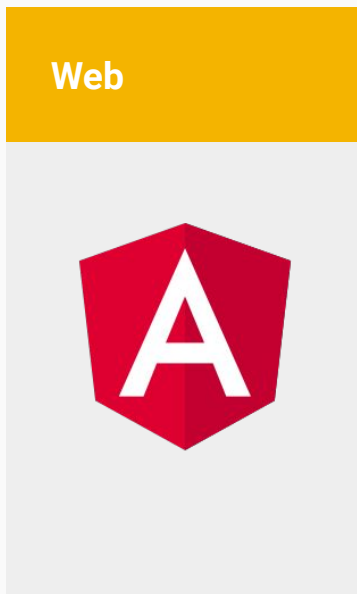
- Without authentication and authorization
- Without database abstraction via an object-relational mapping.
- Without input validation and input sanitation.



Less modules, functions and dependencies are not enough!

# Ahead of Time (AOT) Compilation

Ahead-of-time compilation (AOT compilation) is the act of compiling a higher-level programming language, or an intermediate representation such as Java bytecode, into a native machine code so that the resulting binary file can execute natively.





M I C R O N A U T <sup>TM</sup>

uses Ahead of Time (AOT)  
Compilation

What are the results of using  
Ahead of Time (AOT)  
Compilation?



# The results of using Ahead of Time (AOT) Compilation

Data from Micronaut website:

- Startup time around a second.
- All Dependency Injection, AOP and Proxy generation happens at compile time.
- Can be run with as little as 10mb Max Heap.



I don't believe, show me!

Is it possible to improve  
more?

Yes, with GraalVM™

GraalVM is an universal virtual machine:

- Runs Java, Scala, Kotlin etc.
- Native image compiled with ahead-of-time improves the startup time and reduce the memory footprint of JVM-based applications.

GraalVM works well when:

- Little or no runtime reflection is used.
  - Use third party libraries selectively.
- Limited or no dynamic classloading.

# Demo



+ GraalVM™

MICRONAUT™

# AOT Summary

## Spring + JVM

Startup ≈ 6s

Memory ≈ 15mb

## Micronaut + JVM

Startup ≈ 0.8s

Memory ≈ 10mb

## Micronaut + GraalVM

Startup ≈ 0.4s

Memory ≈ 5mb



What else does Micronaut  
do?



# Blocking or Non-Blocking HTTP server built on Netty

With a smooth learning curve, Micronaut HTTP server makes it as easy as possible to expose APIs that can be consumed by HTTP clients.

## Non-Blocking (RxJava + Netty)

```
@Controller("/hello")
public class HelloController {

    @Get
    public Single<String> hello() {
        return Single.just("Hello Micronaut");
    }

}
```

## Blocking

```
@Controller("/hello")
public class HelloController {

    @Get
    public String hello() {
        return "Hello Micronaut";
    }

}
```

# Dependency Injection and Inversion of Control (IoC)

This is a similar approach taken by Spring and Google Dagger, but without reflection and proxies. All the injections are done in compile time.

```
@Singleton
public class HelloService {

    public String hello() {
        return "Hello Micronaut";
    }

}
```



```
@Controller("/hello")
public class HelloController {

    @Inject
    private HelloService helloService;

    @Get
    public String hello() {
        return helloService.hello();
    }

}
```

# Cloud Native Features

# Distributed Tracing

When operating Microservices in production it can be challenging to troubleshoot interactions between Microservices in a distributed architecture. Micronaut features integration with both Zipkin and Jaeger (via the Open Tracing API).

```
@Controller("/hello")
public class HelloController {

    @Inject
    private HelloService helloService;

    @Get("/{name}")
    @NewSpan("hello")
    public String hello(@SpanTag String name) {
        return helloService.hello();
    }
}
```



# Serverless Functions

Serverless architectures where as a developer you deploy functions that are fully managed by the Cloud environment and are executed in ephemeral processes require a unique approach.

```
@FunctionBean("hello-function")
public class HelloFunction implements Supplier<String> {

    @Override
    public String hello() {
        return "Hello world";
    }
}

@FunctionClient
public interface HelloFunctionClient {

    String hello();
}
```



# Summary

## 2° Place

- Productivity with annotations
- Blocking or Non-Blocking HTTP server built on Netty

## 1° Place

### Ahead of Time (AOT) Compilation

- Low memory footprint
- Fast Startup
- IoC

## 3° Place

### Cloud Native Features

- Service Discovery
- Distributed Tracing
- Serverless
- Distributed Configuration

# Thanks a million!

## Questions?



[/larchanjo](#)



[/luram-archanjo](#)