GraalVM™

QUARKUS

Kubernetes for Java Developers

Horacio Gonzalez
@LostInBrittany

OVHcloud

Who are we?

Introducing myself and introducing ~~OVH~~ OVHcloud

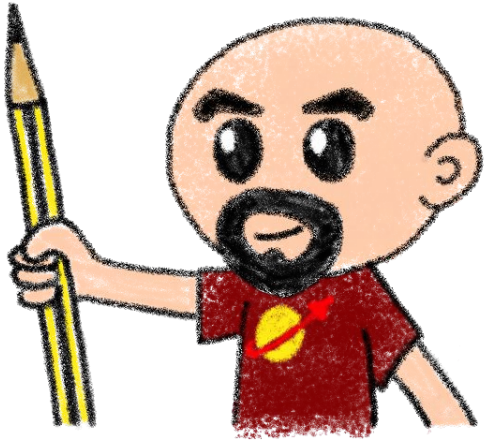
Horacio Gonzalez

@LostInBrittany

Spaniard lost in Brittany, developer, dreamer and all-around geek

OVHcloud

DevRel Leader



DevFest du Bout du Monde

Finist Devs

Google Developers Experts 2019

Web Technologies GDE

Flutter

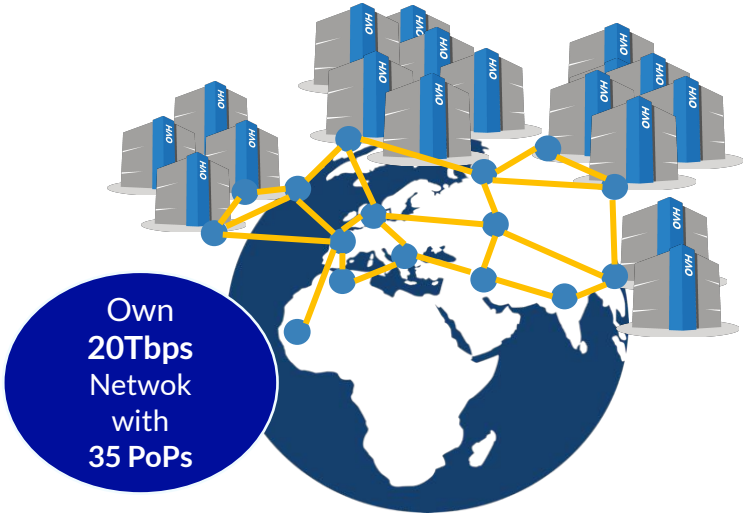
OVHcloud: A Global Leader

200k Private cloud VMs running

Dedicated IaaS Europe

Hosting capacity : **1.3M** Physical Servers

360k Servers already deployed



Own **20Tbps** Netwok with **35 PoPs**

30 Datacenters

> **1.3M** Customers in **138** Countries

OVHcloud: Our solutions

Cloud

- VPS
- Public Cloud
- Private Cloud
- Serveur dédié
- Cloud Desktop
- Hybrid Cloud

Mobile Hosting

- Containers
- Compute
- Database
- Object Storage
- Securities
- Messaging

Web Hosting

- Domain names
- Email
- CDN
- Web hosting
- MS Office
- MS solutions

Telecom

- VoIP
- SMS/Fax
- Virtual desktop
- Cloud Storage
- Over the Box

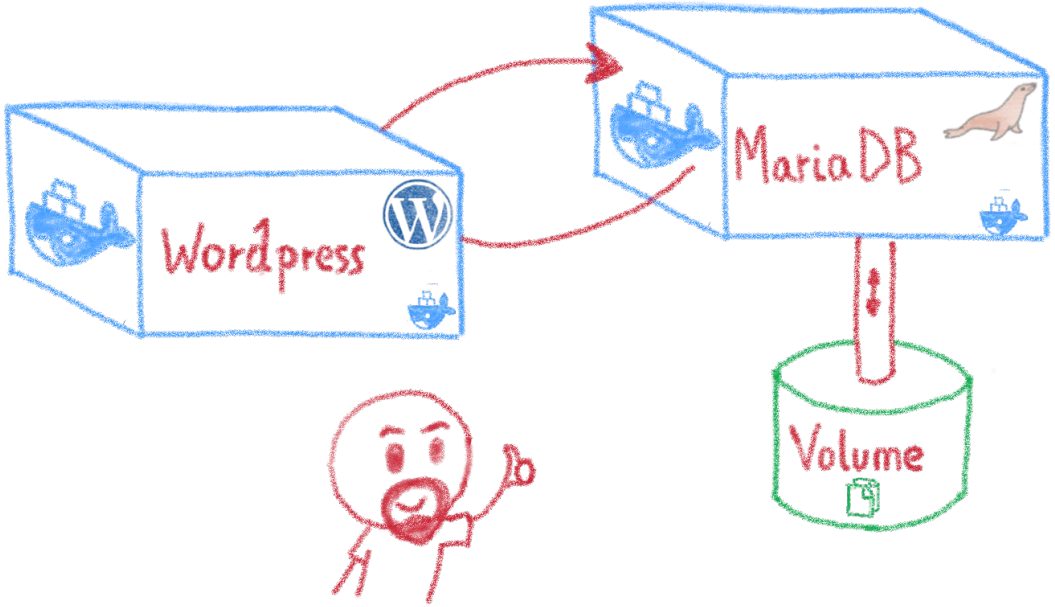
Orchestrating containers

Like herding cats… but in hard mode!

From bare metal to containers

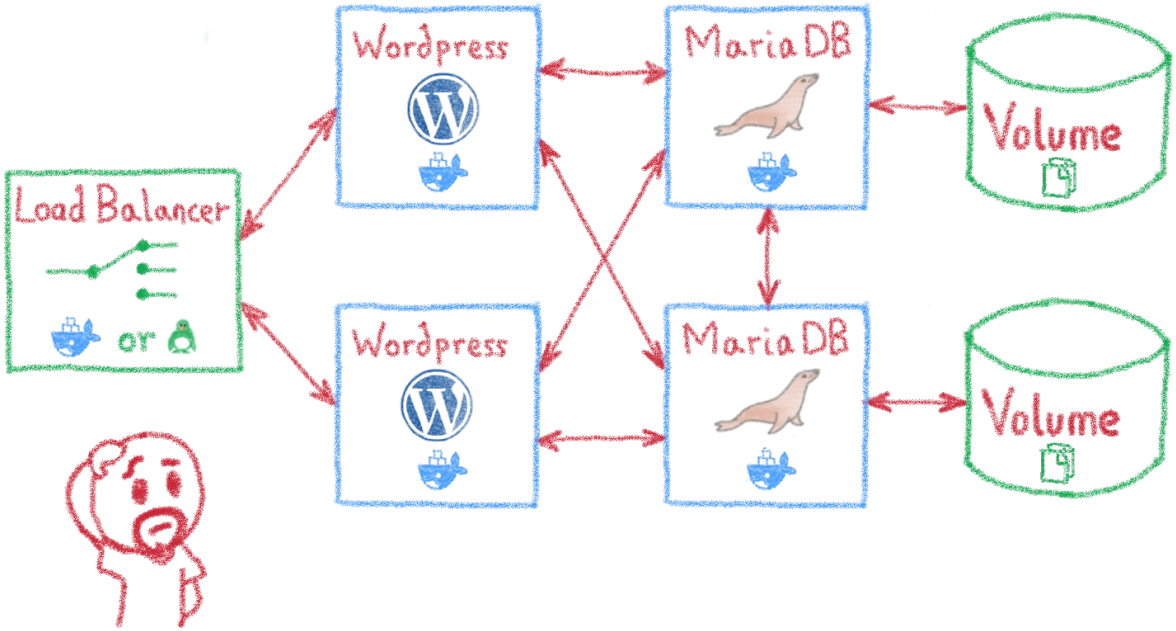
Another paradigm shift

Containers are easy...



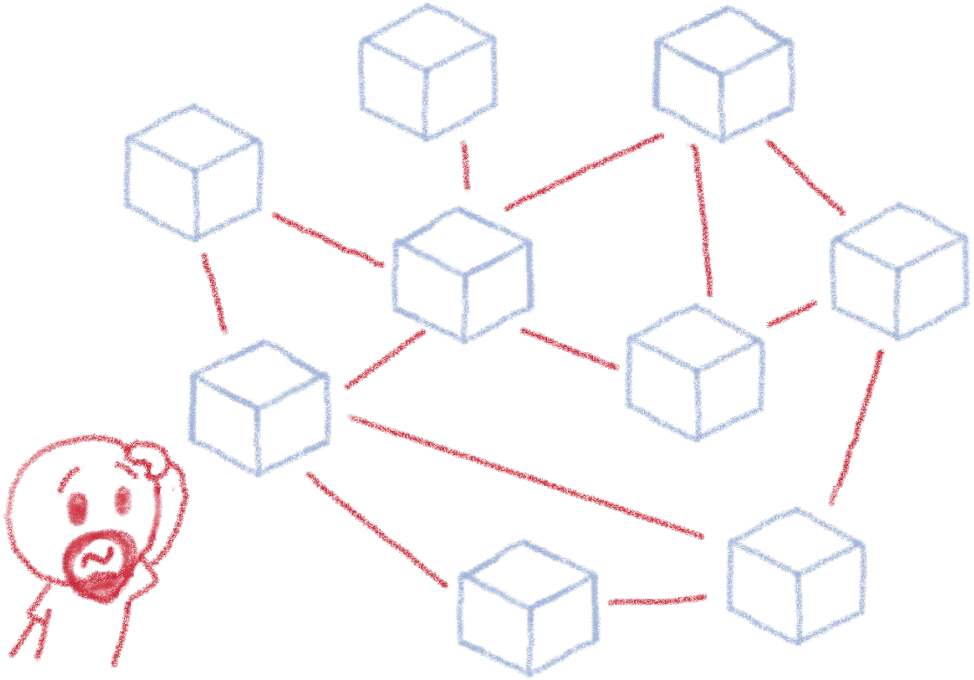
For developers

Less simple if you must operate them



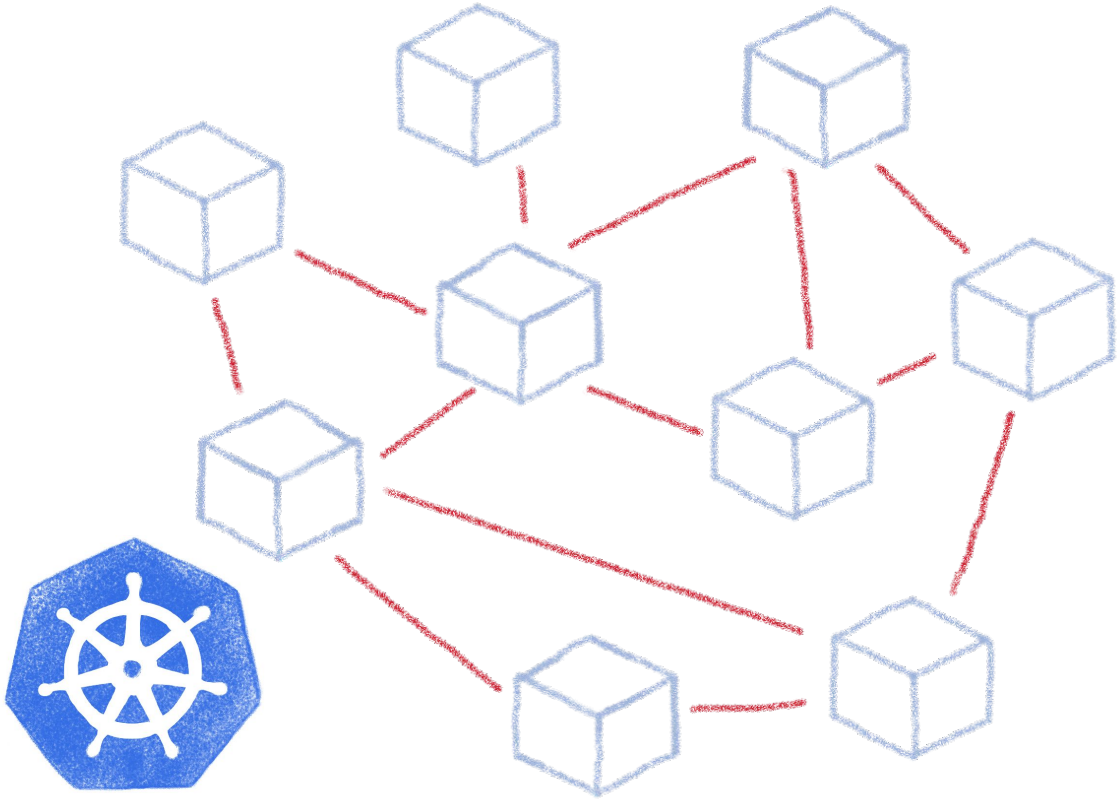
Like in a production context

And what about microservices?

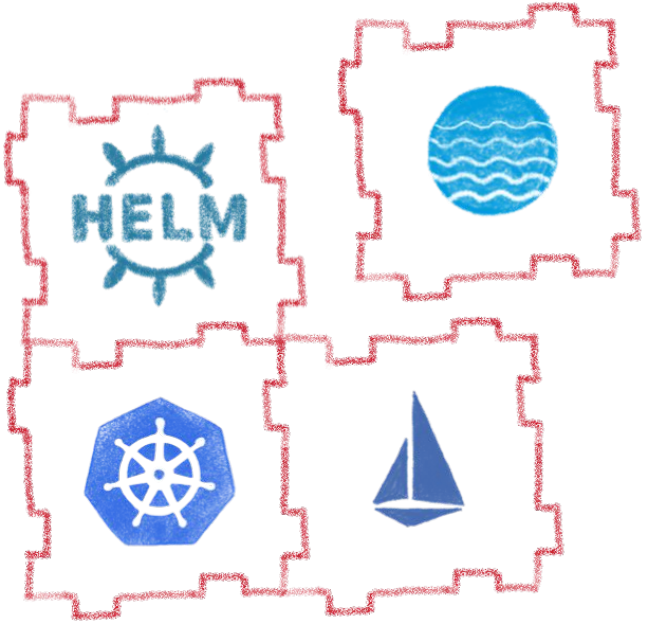


Are you sure you want to operate them by hand?

Taming microservices with Kubernetes



Kubernetes is modular



Fully extensible

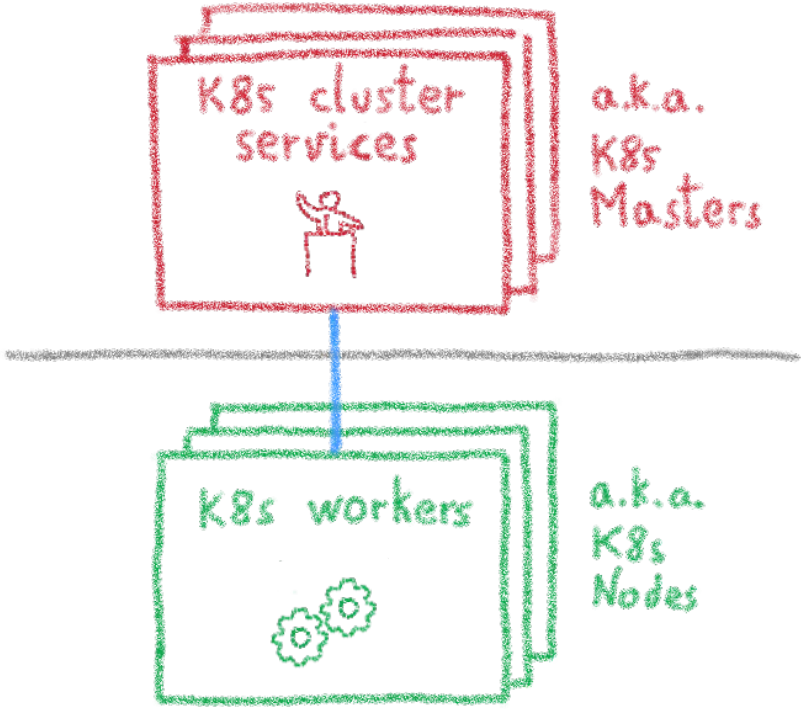
- Kubernetes API
- Cluster demons
- Controllers
- Custom resources
- ...

Operators

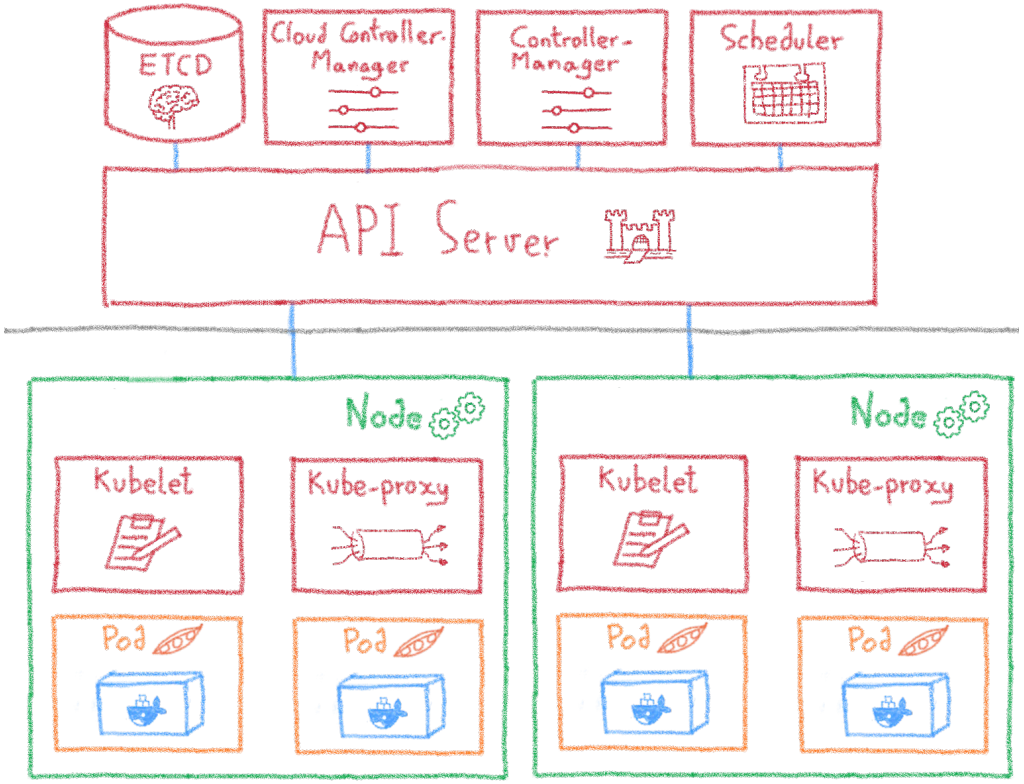
Kubernetes

Way more than a buzzword!

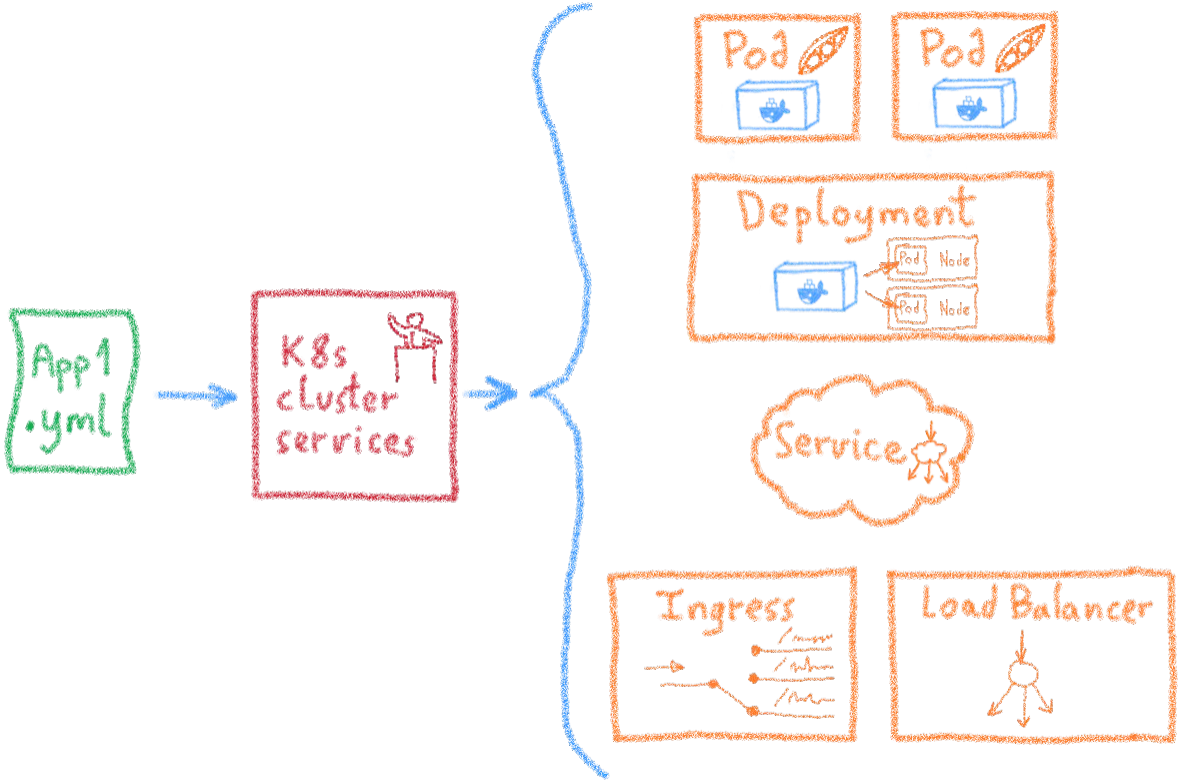
Masters and nodes



Some more details



Desired State Management



Ingress

Services

Deployments

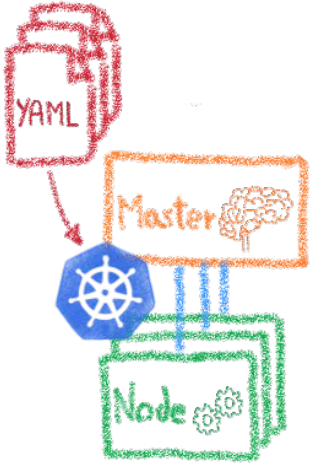
Pods

Sidecars

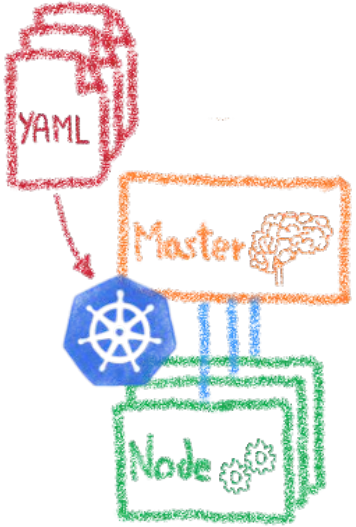
Replica Sets

Multi-environment made easy

Dev, staging, prod, multi-cloud...

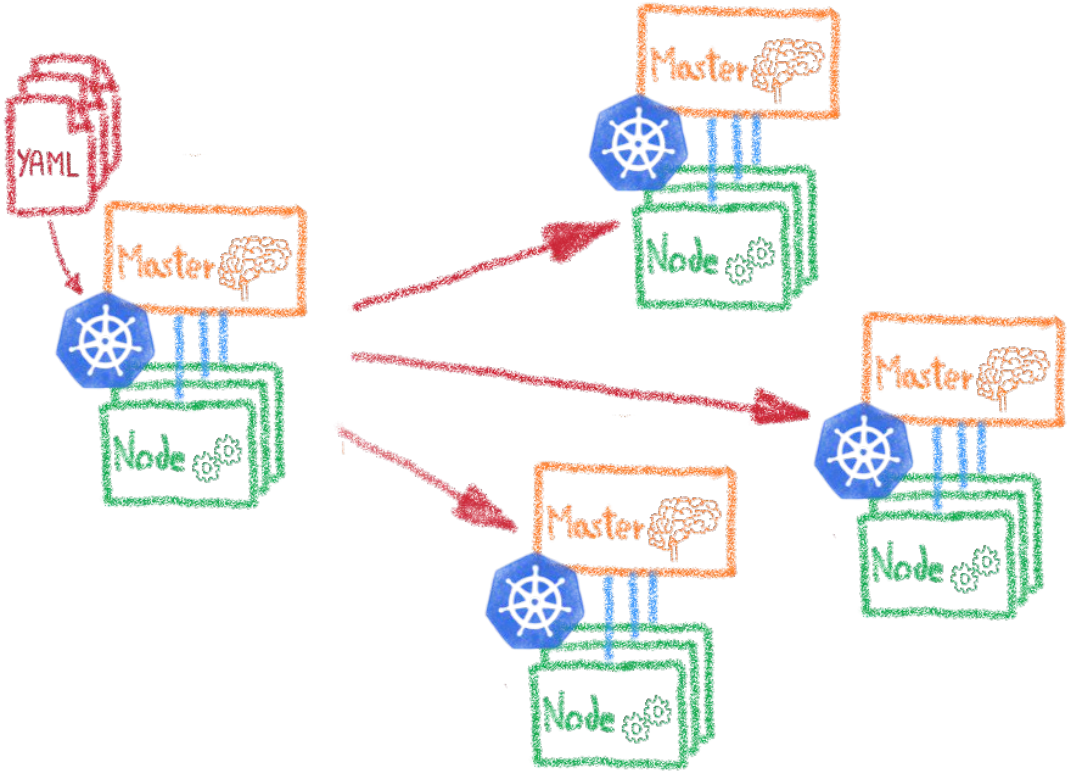


Declarative infrastructure



Multi-environment made easy

Having identical, software defined envs



Dev envs

Staging

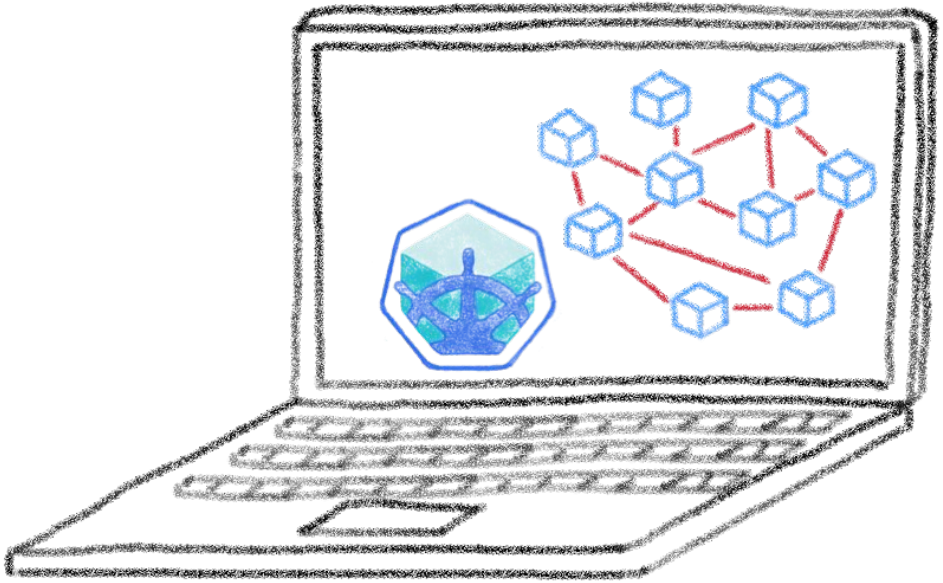
Multi-cluster

Multi-cloud

I have deployed on Minikube, woah!

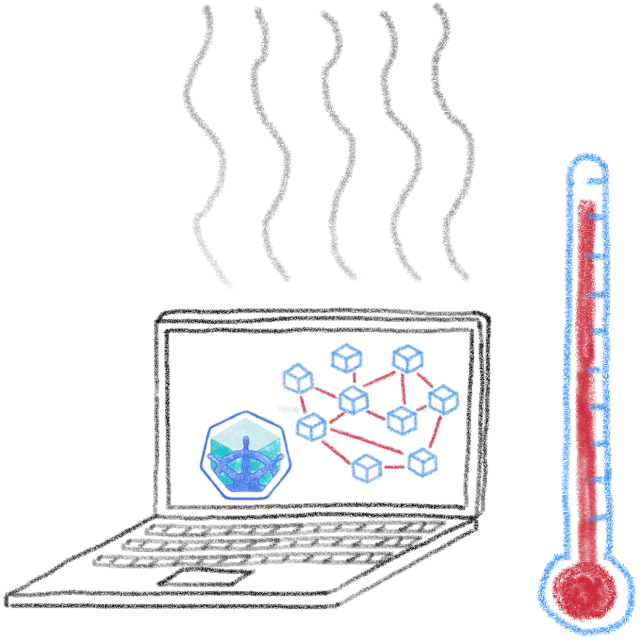
A great fastlane into Kubernetes

Running a full K8s in your laptop



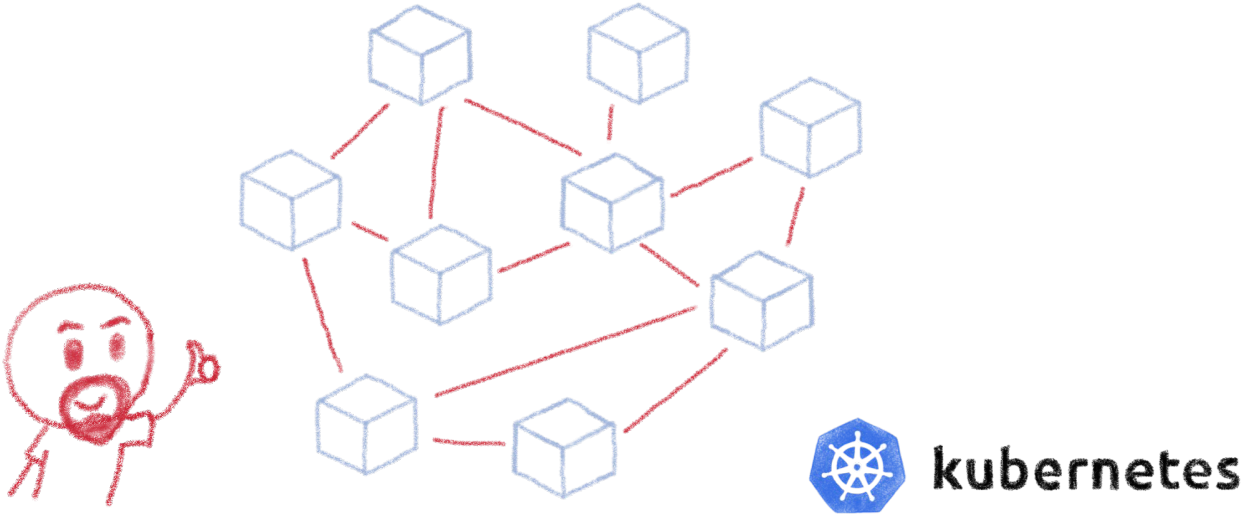
A great learning tool

Your laptop isn't a true cluster



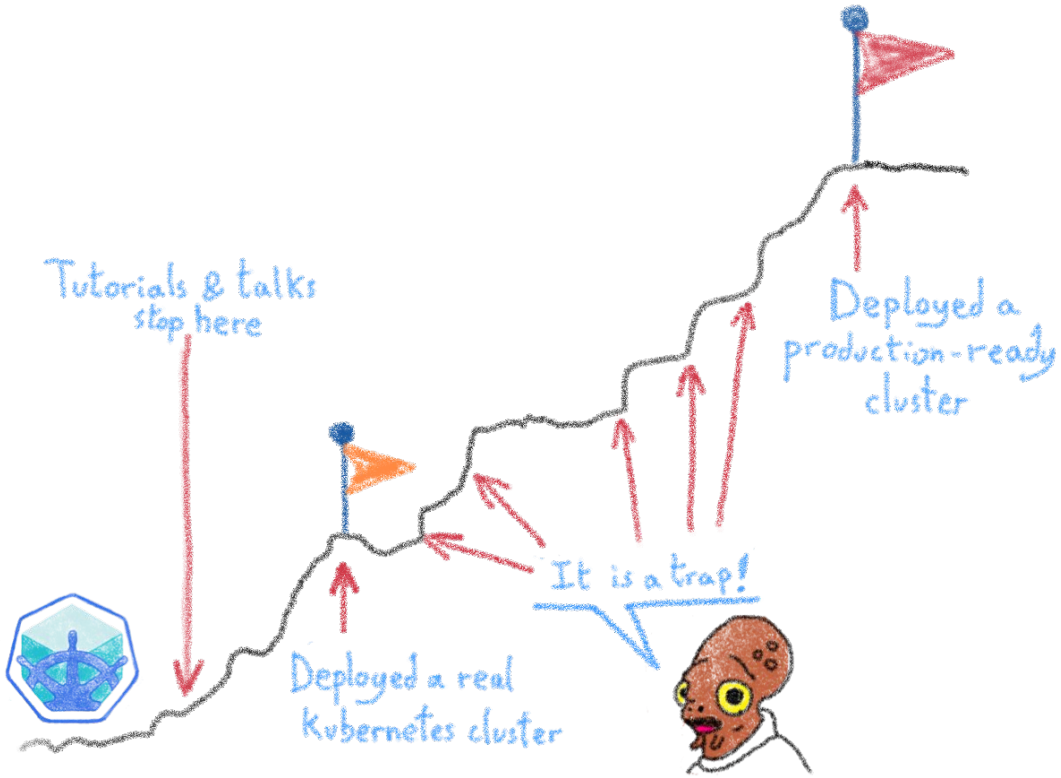
Don't expect real performances

Beyond the first deployment



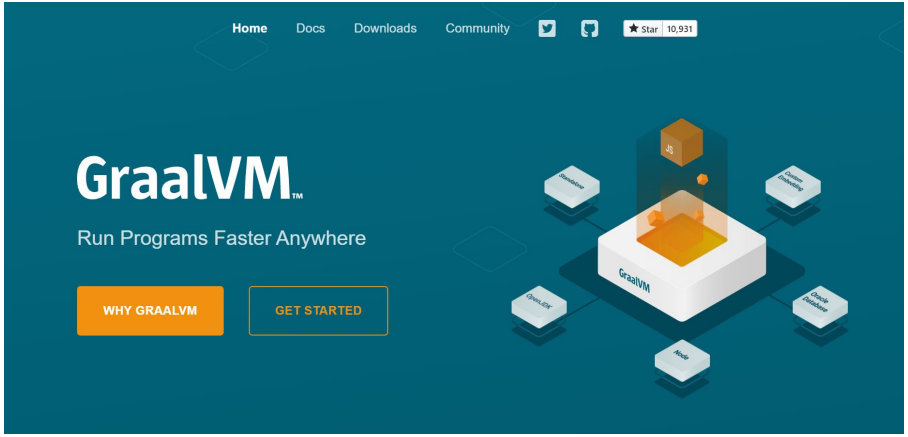
So I have deployed my distributed architecture on K8s, everything is good now, isn't it?

Minikube is only the beginning



GraalVM

An alternative JVM with a twist



A long time ago, when the JVM was young

HotSpot becomes the offical JVM in Java 1.3

HotSpot has tiered compilation

It starts in interpreter mode, then C1 JIT
and, if needed, C2 JIT

Really powerful, really complex

Last big addition: JVM Intrinsics

It worked really well, but its getting old...

C++ stack, old code base, difficult to maintain

The Java platform to the rescue

JVMCI

AoT Compilation

JVM Compiler Interface (JVMCI) - JEP 243

Ahead of Time (AoT) Compilation - JEP 295

Graal project

GraalVM™

An Oracle project to rethink the JVM

Graal compiler

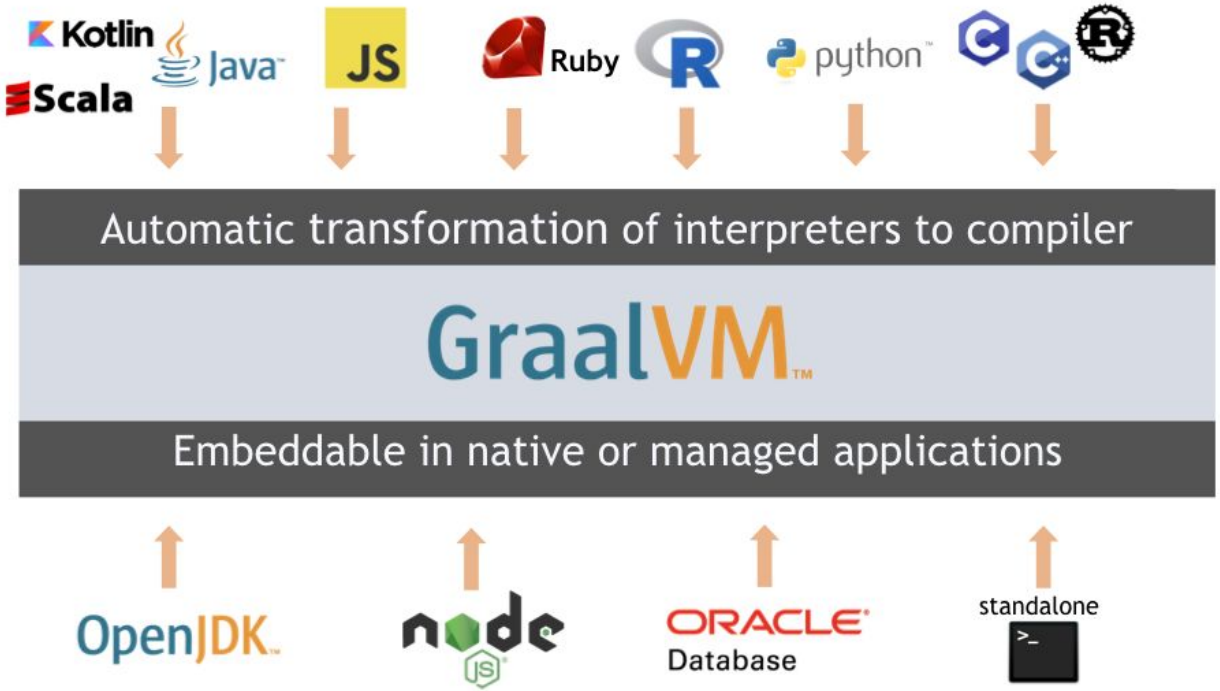
- A Java compiler written in Java
  - Capable of compiling itself!
- Independent of HotSpot
  - Can be used in HotSpot with JVMCI
- Can do either JIT or AOT compilations

What's GraalVM?

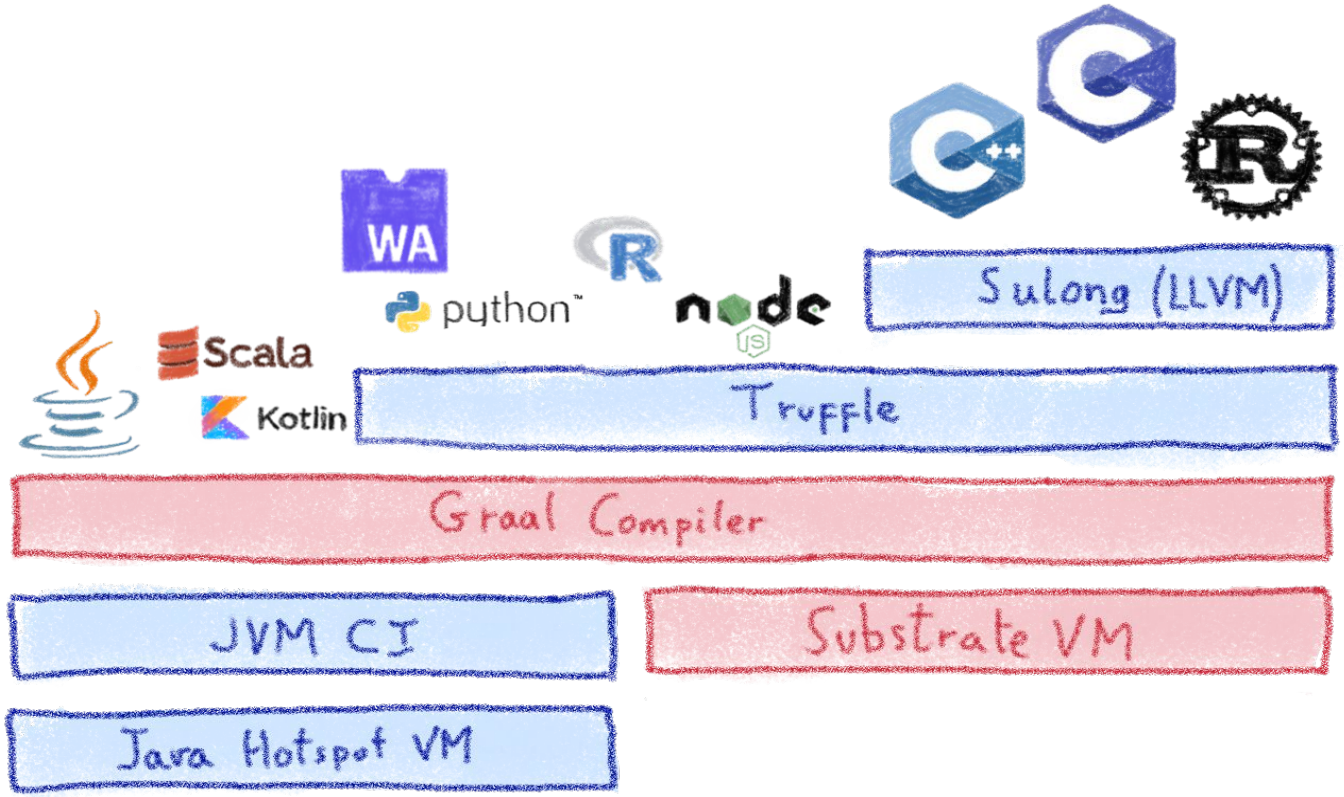
A standalone Java Development Kit to execute:

- JVM-based languages
- Dynamic languages
- LLVM-based languages

What's GraalVM?



What's GraalVM?



GraalVM Features

GraalVM lets you:

- Run your code faster and more efficiently
- Interoperate directly with most modern programming languages
- Embed languages with the GraalVM SDK
- Create compiled native images
- Use a single set of tools to monitor, debug, and profile all your code

GraalVM base package

The base installation includes:

- The JVM
- The Graal compiler
- The LLVM bitcode interpreter
- The JavaScript runtime

Why GraalVM?

For Java programs:

- Run Java faster
- Make Your Application Extensible
- Create a Native Image

Why GraalVM?

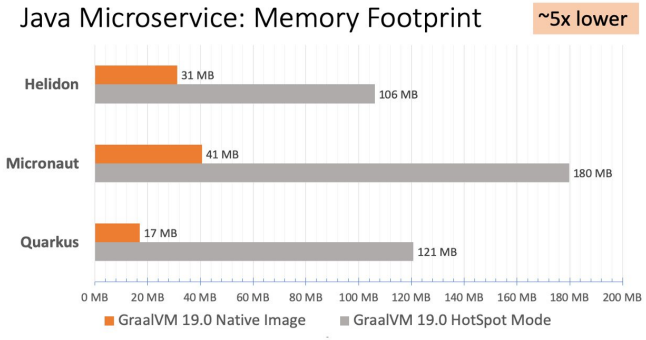
For JavaScript programs:

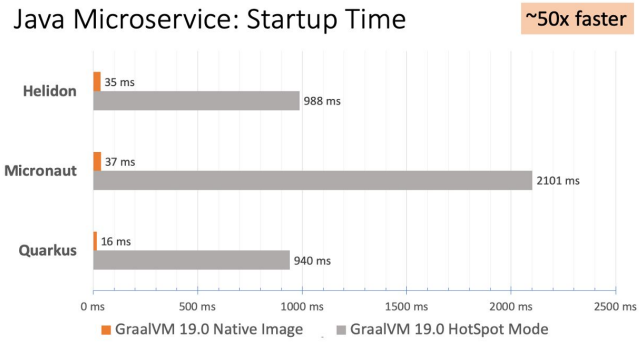
- Reuse Libraries from Java, R, or Python
- Run with Large Heaps
- Define Data Structures in C/C++

Why GraalVM?

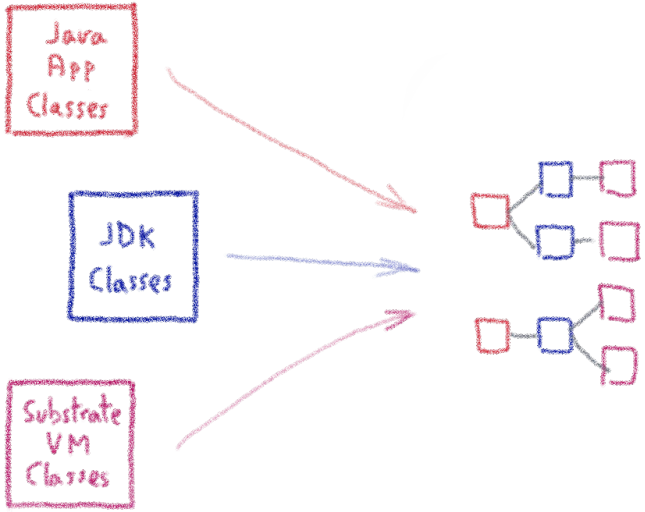
GraalVM native images reduce:

- Runtime memory footprint
- Startup time





And how does it compile ?



Dead code elimination

Closed world assumption

Adding some limitations...

No dynamic classloading

Manual listing for reflection

Manual listing for dynamic proxy

Two versions

GraalVM™

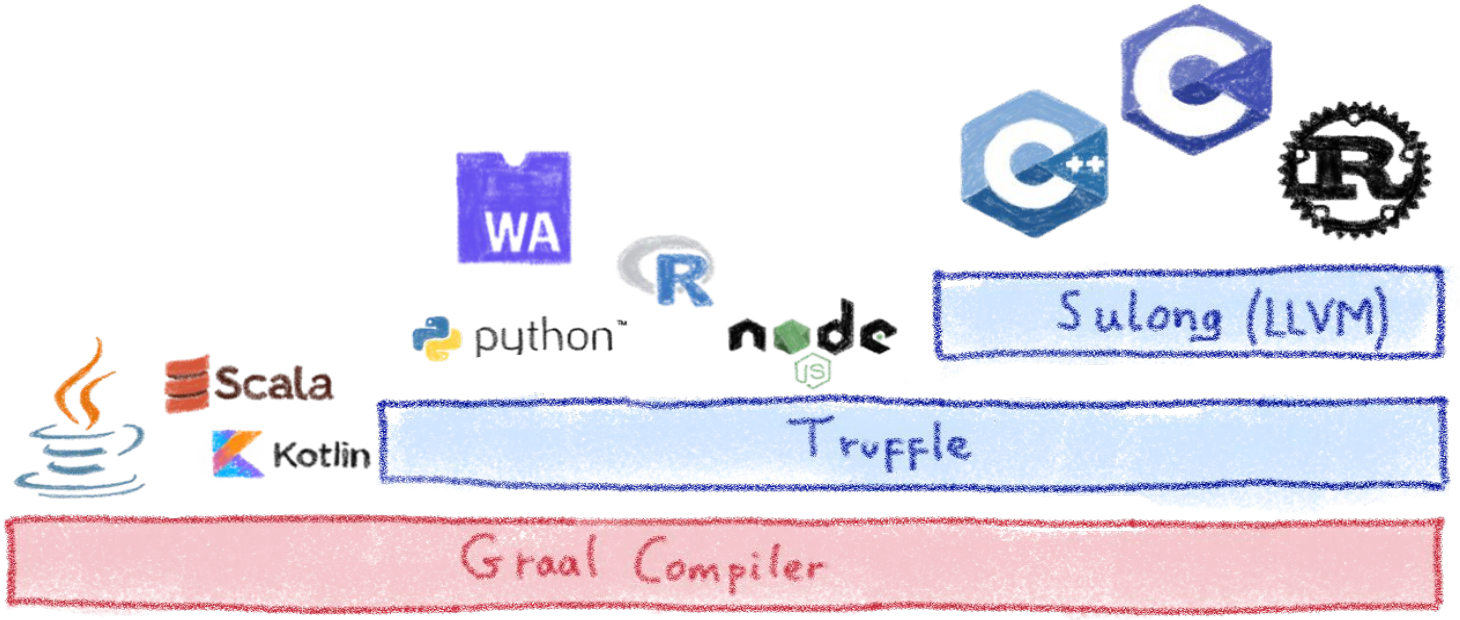
Community Edition & Enterprise Edition

Polyglot GraalVM

Wasm, JS, Ruby, Python, R, C, C++, Rust…

GraalVM™

Sulong and Truffle



Lots of languages... and growing!

Running JavaScript code

```
function sayHello() {
    console.log('Hello!');
}

sayHello();
```

Using GraalVM `js` command

```
$ ~/opt/graalvm/bin/js sayHello.js
Hello!
```

Running NodeJS code

```
const http = require("http");
const span = require("ansispan");
require("colors");

http.createServer(function (request, response) {
    response.writeHead(200, {"Content-Type": "text/html"});
    response.end(span("Hello Graal.js!".green));
}).listen(8000, function() {
  console.log("Graal.js server running at http://127.0.0.1:8000/".red);
});
```

Using GraalVM node and npm commands

```
$ ~/opt/graalvm/bin/npm install colors ansispan
[...]
+ colors@1.4.0
+ ansispan@0.0.4
added 2 packages from 3 contributors in 14.951s
$ ~/opt/graalvm/bin/node helloNode.js
Graal.js server running at http://127.0.0.1:8000/
```

Running WebAssembly Programs

```c
#include <stdio.h>

int main() {
  int number = 1;
  int rows = 10;
  for (int i = 1; i <= rows; i++) {
    for (int j = 1; j <= i; j++) {
      printf("%d ", number);
      ++number;
    }
    printf(".\n");
  }
  return 0;
}
```

Using wasm launcher

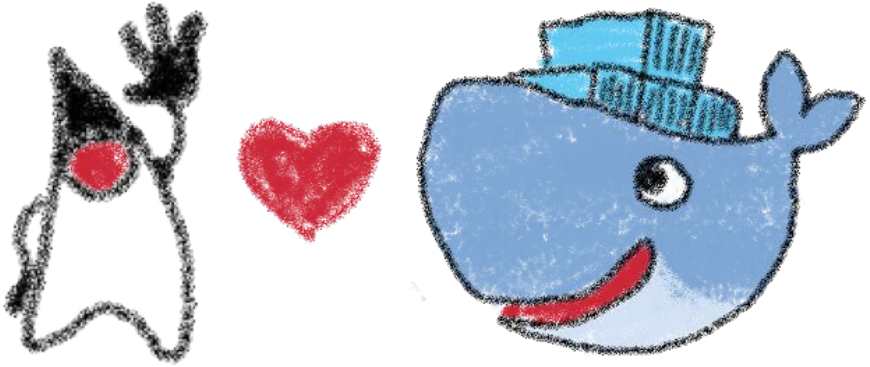
```
graalvm/bin/wasm --Builtins=memory,env:emscripten your_module.wasm
```

Embedding WebAssembly Programs

```
import org.graalvm.polyglot.*;
import org.graalvm.polyglot.io.ByteSequence;

// You need to load the .wasm contents into a byte array.
byte[] binary = readBytes("example.wasm");

Source.Builder sourceBuilder =
    Source.newBuilder("wasm", ByteSequence.create(binary), "example");
Source source = sourceBuilder.build();

Context.Builder contextBuilder = Context.newBuilder("wasm");
Context context = contextBuilder.build();
context.eval(source);

Value mainFunction = context.getBindings("wasm").getMember("_main");
mainFunction.execute();
```

GraalVM ❤️ Kubernetes

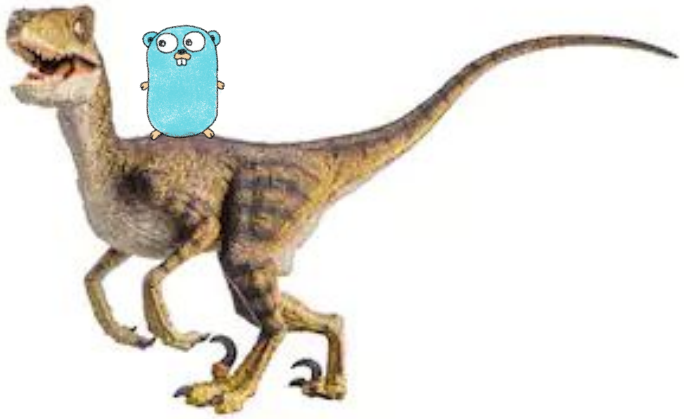
Giving Java a place in a Cloud Native world



Containers didn't love Java

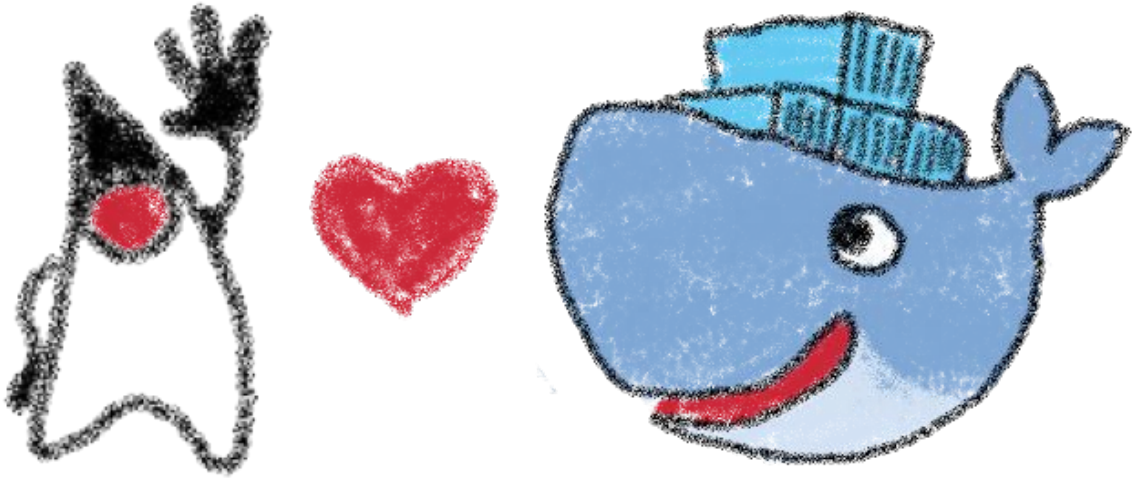
Big images, slow to start, memory hungry...

GraalVM change things



Small images, fast start, low memory footprint

Java is now a real alternative in Cloud Native

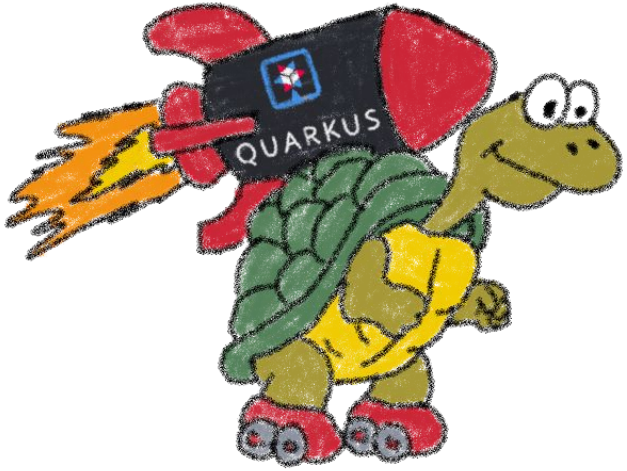


First class cloud player!

But what about old apps?

Most of them difficult to compile in GraalVM

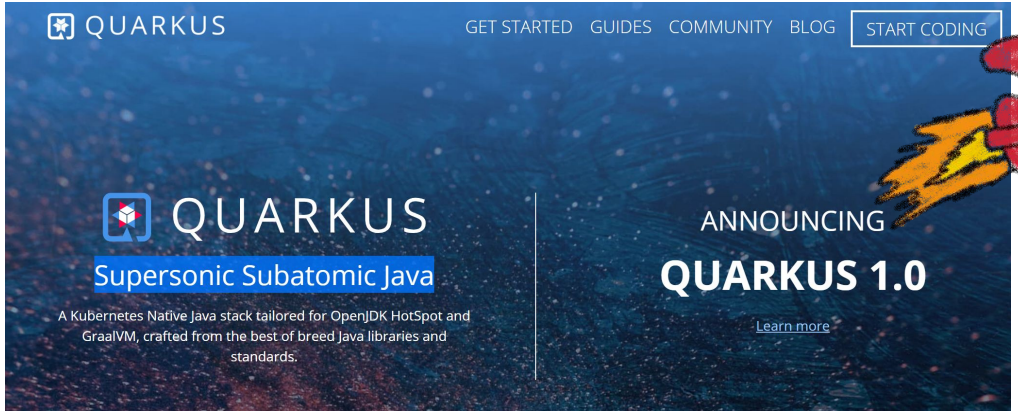
Enter Quarkus

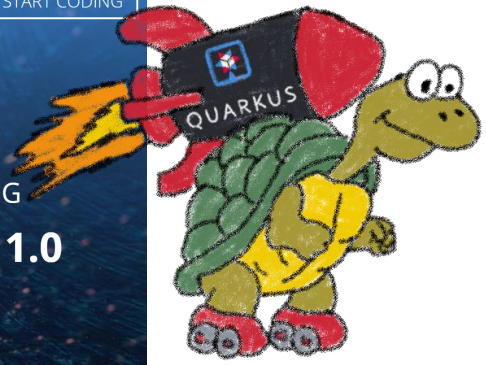


A new generation Java app stack

Quarkus

Supersonic Subatomic Java

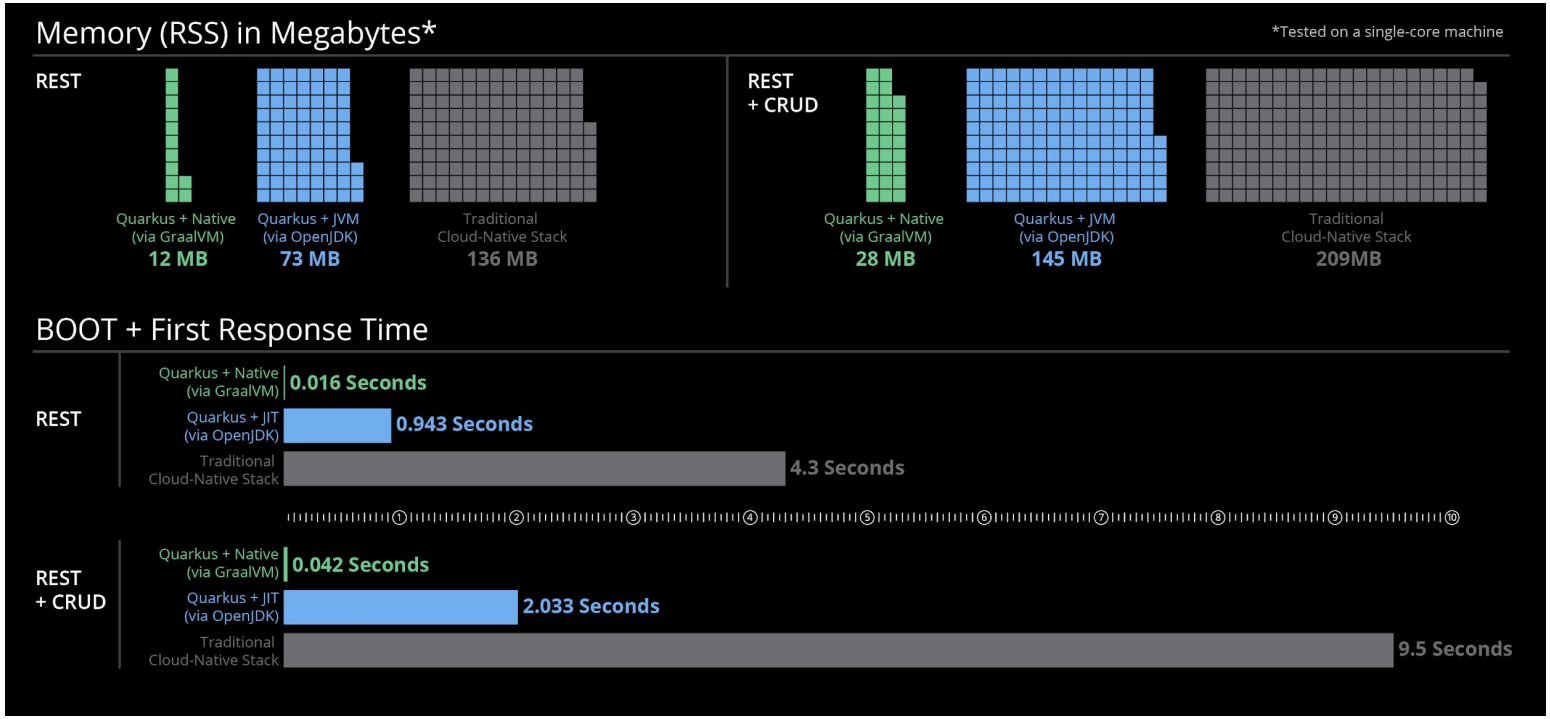




What's Quarkus?

- A Kubernetes Native Java stack
- Tailored for OpenJDK HotSpot and GraalVM
- Crafted from the best of breed Java libraries and standards

Container first



Unifies imperative and reactive

IMPERATIVE	REACTIVE

```
@Inject
SayService say;

@GET
@Produces(MediaType.TEXT_PLAIN)
public String hello() {
    return say.hello();
}
```

```
@Inject @Channel("kafka")
Publisher<String> reactiveSay;

@GET
@Produces(MediaType.SERVER_SENT_EVENTS)
public Publisher<String> stream() {
    return reactiveSay;
}
```

Combine imperative code and the non-blocking reactive style

By developers, for developers

- Unified configuration
- Zero config, live reload in the blink of an eye
- Streamlined code for the 80% common usages, flexible for the 20%
- No hassle native executable generation

Leveraging the ecosystem

Over fifty best-of-breed libraries
wired on a standard backbone

Conclusion

That's all, folks!