# Microservices for the Masses with Spring Boot, JHipster, and OAuth

**Matt Raible** | **@mraible**

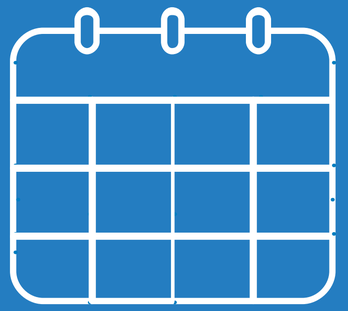
March 11, 2020

Photo by **Tambako The Jaguar** flickr.com/photos/tambako/4580951085

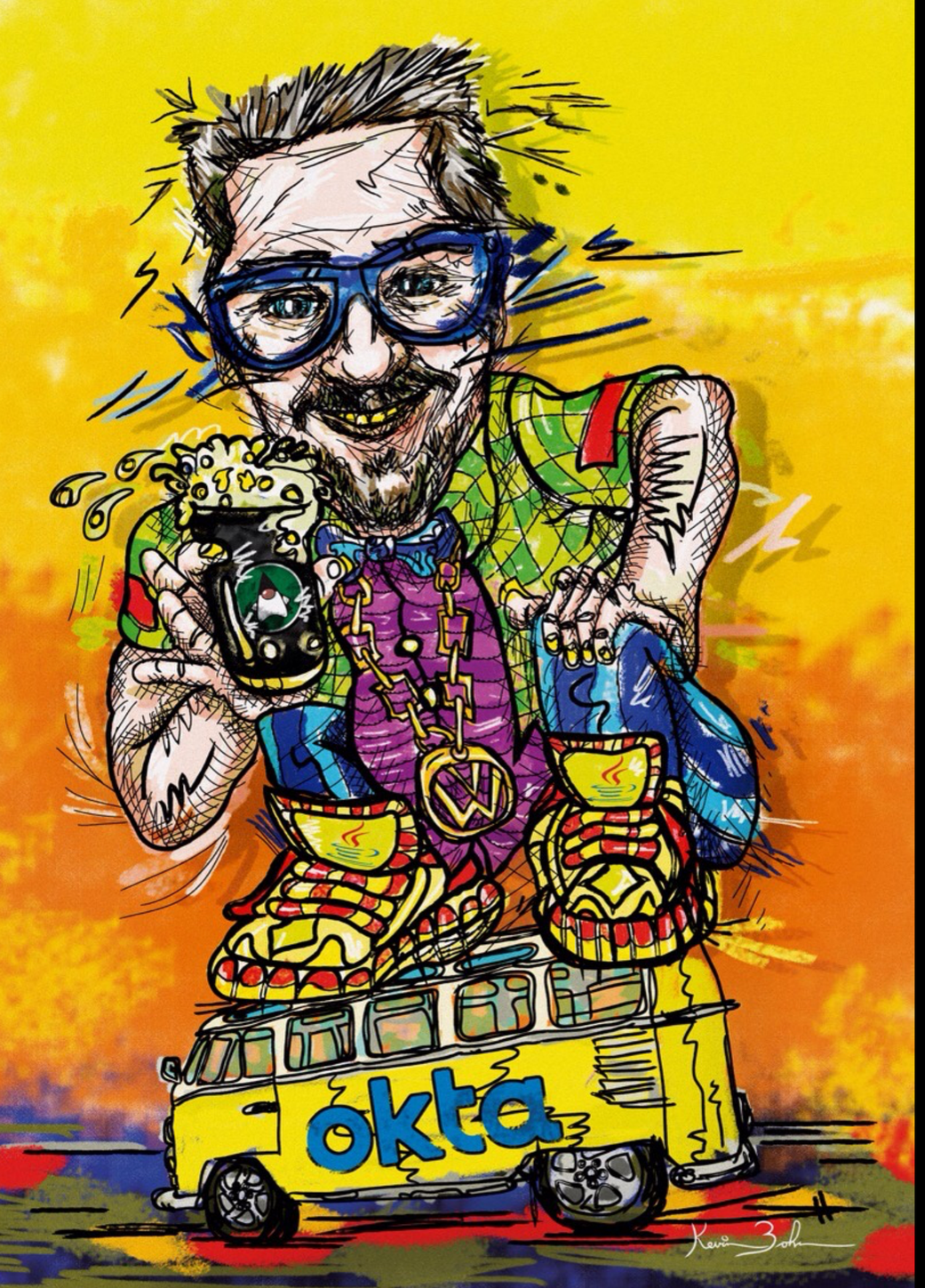
{okta}

Do you use microservices?

# Agenda



1. Introduction to Microservices
2. Microservices with JHipster
3. Deploying to the Cloud
4. JHipster Roadmap




# Hi, I'm Matt Raible!

Father, Husband, Skier, Mountain Biker, Whitewater Rafter

**Open Source Connoisseur**

Web Developer and Java Champion

Okta Developer Advocate

**Bus Lover**

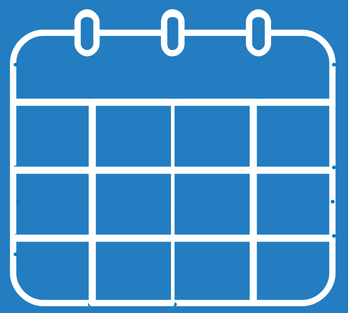
Blogger on raibledesigns.com and developer.okta.com/blog


The UPS Store
perry nails
CLEANERS
DUGOUT
PHILLIPS 66
DENTIST

developer.okta.com


What About You?

# Part 1



**Introduction to Microservices**

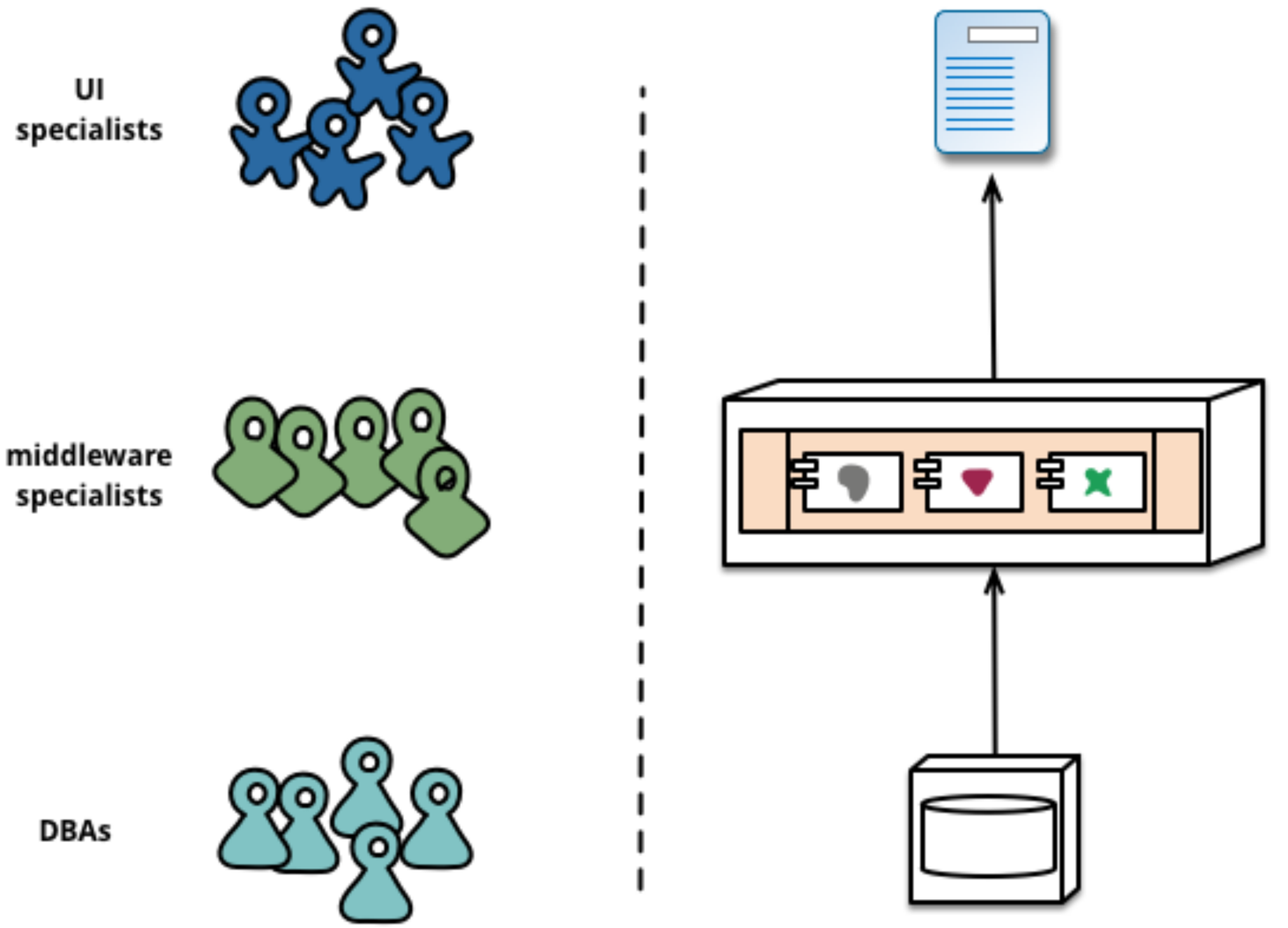
History of Microservices

Microservices Architecture Philosophy

Why Microservices?

Demo: A Microservices Architecture with Spring Boot and Spring Cloud

# Microservices Visionaries




Siloed functional teams...

... lead to silod application architectures.
Because Conway's Law

# Conway’s Law

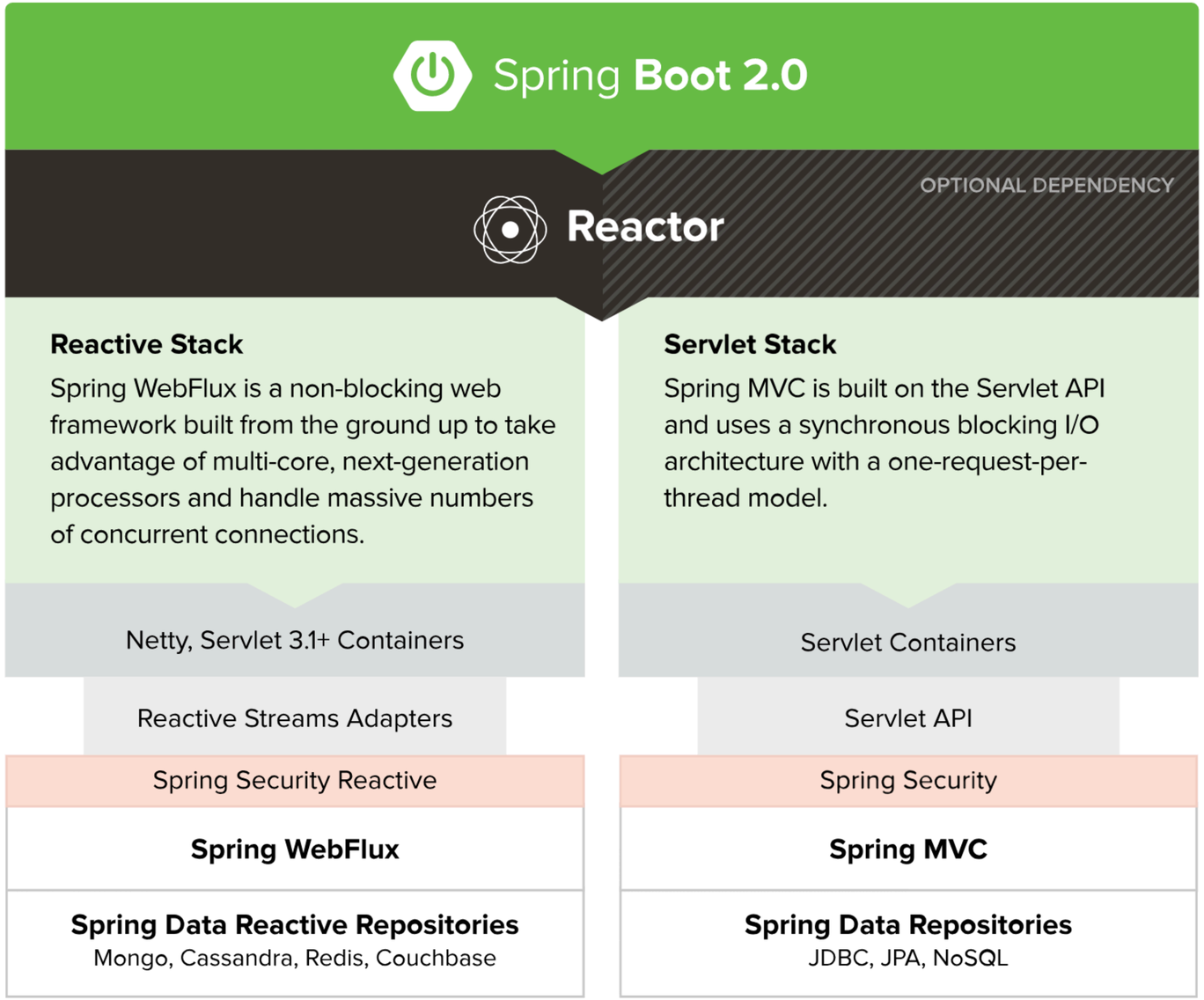
> “Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.”

**Melvin Conway** *1967*

“Do one thing and do it well.”

> “You shouldn't start with a microservices architecture. Instead begin with a monolith, keep it modular, and split it into microservices once the monolith becomes a problem.”

**Martin Fowler** *March 2014*


Spring Boot 2.0
Reactor
OPTIONAL DEPENDENCY
Reactive Stack
Spring WebFlux is a non-blocking web framework built from the ground up to take advantage of multi-core, next-generation processors and handle massive numbers of concurrent connections.
Servlet Stack
Spring MVC is built on the Servlet API and uses a synchronous blocking I/O architecture with a one-request-per-thread model.
Netty, Servlet 3.1+ Containers
Servlet Containers
Reactive Streams Adapters
Servlet API
Spring Security Reactive
Spring Security
Spring WebFlux
Spring MVC
Spring Data Reactive Repositories
Mongo, Cassandra, Redis, Couchbase
Spring Data Repositories
JDBC, JPA, NoSQL


start.spring.io

Spring Initializr

https://start.spring.io

# SPRING INITIALIZR bootstrap your application now

Generate a Maven Project with Java and Spring Boot 2.1.0

## Project Metadata

Artifact coordinates

Group

com.example

Artifact

demo

## Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Web, Security, JPA, Actuator, Devtools...

Selected Dependencies

Generate Project ⌘ + ⏎

Don't know what to look for? Want more options? Switch to the full version.

start.spring.io is powered by Spring Initializr and Pivotal Web Services

Spring Initializr

start.spring.io

Light UI | Github | Twitter | Help

# Spring Initializr

Bootstrap your application

**Project**

Maven Project | Gradle Project

**Language**

Java | Kotlin | Groovy

**Spring Boot**

2.3.0 M2 | 2.3.0 (SNAPSHOT) | 2.2.6 (SNAPSHOT) | 2.2.5 | 2.1.14 (SNAPSHOT) | 2.1.13

**Project Metadata**

Group

com.example

Artifact

demo

Options

**Dependencies**

Search dependencies to add

Web, Security, JPA, Actuator, Devtools...

Selected dependencies

No dependency selected

Generate - ⌘ + ↵ | Explore - Ctrl + Space | Share...



start.spring.io is powered by

Spring Initializr and Pivotal Web Services

# Demo

Using **start.spring.io**, create:

- A service registry
- A gateway
- A catalog service

Create an endpoint in the catalog service

Create a filtered endpoint in the gateway

Show failover capabilities

Show Spring Security OAuth

**https://github.com/oktadeveloper/java-microservices-examples**

# Create Java Microservices using start.spring.io

```
http https://start.spring.io/starter.zip javaVersion==11 \
  artifactId==discovery-service name==eureka-service \
  dependencies==cloud-eureka-server baseDir==discovery-service \
  | tar -xzvf -
```

# Enable Eureka Server & Configure application.properties

```
@EnableEurekaServer
```

```
server.port=8761
eureka.client.register-with-eureka=false
```

# Create Car Service

```
http https://start.spring.io/starter.zip \
  artifactId==car-service name==car-service baseDir==car-service \
  dependencies==actuator,cloud-eureka,data-jpa,h2,data-
rest,web,devtools,lombok | tar -xzvf -
```

# Enable Discovery & Configure application.properties

```
@EnableDiscoveryClient
```

```
server.port=8090
spring.application.name=car-service
```

# Create API Gateway

```
http https://start.spring.io/starter.zip \
  artifactId==api-gateway name==api-gateway baseDir==api-gateway \
  dependencies==cloud-eureka,cloud-feign,data-rest,web,cloud-
hystrix,lombok | tar -xzvf -
```

# Enable Discovery & Configure application.properties

```
@EnableDiscoveryClient
```

```
spring.application.name=api-gateway
```

# Build a REST API in Car Service

```
@Data
@NoArgsConstructor
@Entity
class Car {

    public Car(String name) {
        this.name = name;
    }

    @Id
    @GeneratedValue
    private Long id;

    @NonNull
    private String name;
}
```

# Build a REST API in Car Service

```
@RepositoryRestResource
interface CarRepository extends JpaRepository<Car, Long> {
}
```

# Build a REST API in Car Service

```
@Bean
ApplicationRunner init(CarRepository repository) {
    return args -> {
        Stream.of("Ferrari", "Jaguar", "Porsche", "Lamborghini",
                "Bugatti", "AMC Gremlin", "Triumph Stag",
                "Ford Pinto", "Yugo GV").forEach(name -> {
            repository.save(new Car(name));
        });
        repository.findAll().forEach(System.out::println);
    };
}
```

# Consume Cars API in Gateway

```
@EnableFeignClients
@EnableCircuitBreaker
@EnableDiscoveryClient
@SpringBootApplication
public class ApiGatewayApplication {

    public static void main(String[] args) {
        SpringApplication.run(ApiGatewayApplication.class, args);
    }
}
```

# Consume Cars API in Gateway

```
@Data
class Car {
    private String name;
}

@FeignClient("car-service")
interface CarClient {

    @GetMapping("/cars")
    @CrossOrigin
    CollectionModel<Car> readCars();
}
```

# Consume Cars API in Gateway

```
@RestController
class CoolCarController {

    private final CarClient carClient;

    public CoolCarController(CarClient carClient) {
        this.carClient = carClient;
    }

    // code on next slide
}
```

# Consume Cars API in Gateway

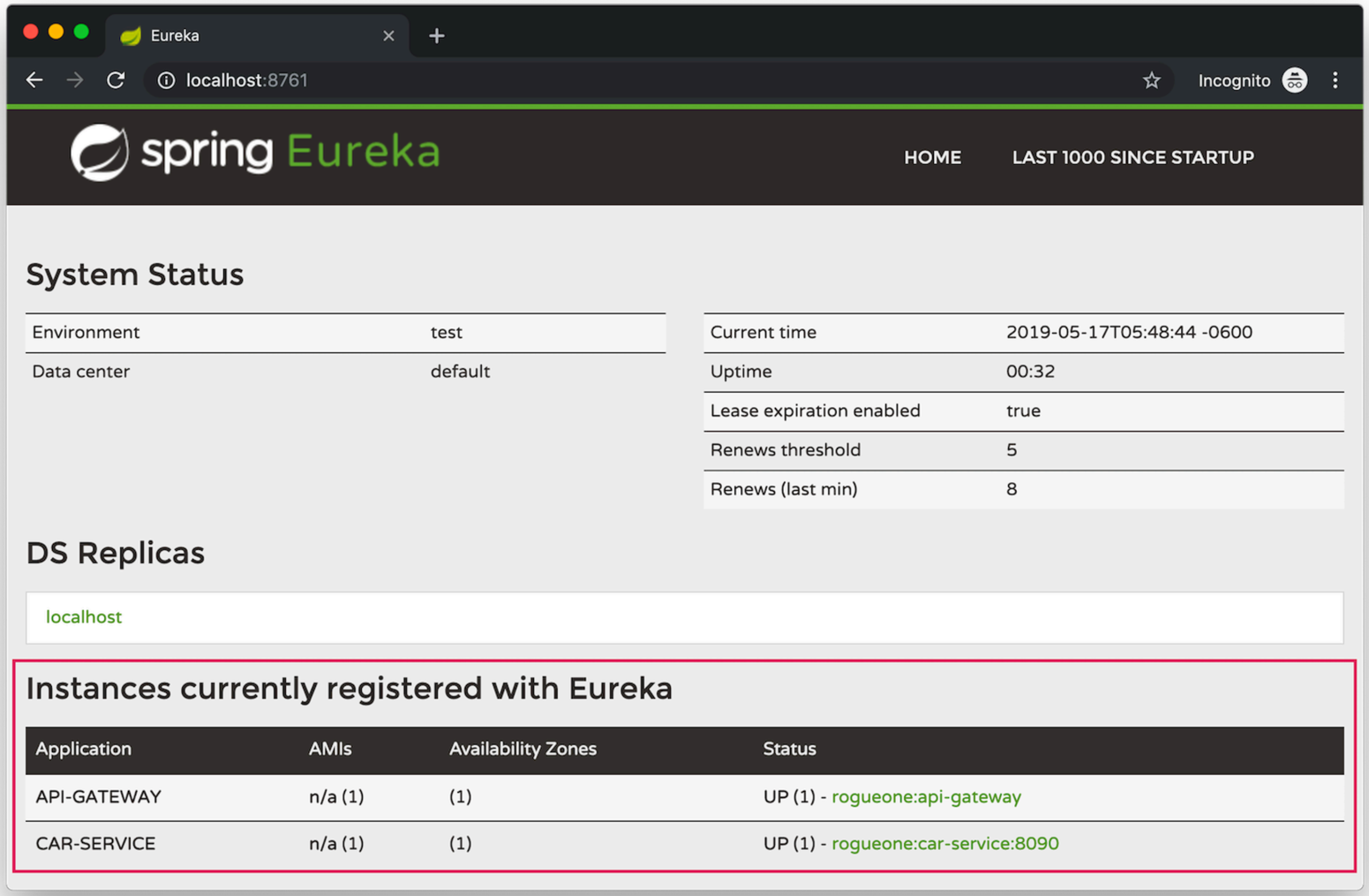
```
private Collection<Car> fallback() {
    return new ArrayList<>();
}

@GetMapping("/cool-cars")
@CrossOrigin
@HystrixCommand(fallbackMethod = "fallback")
public Collection<Car> goodCars() {
    return carClient.readCars()
            .getContent()
            .stream()
            .filter(this::isCool)
            .collect(Collectors.toList());
}
```

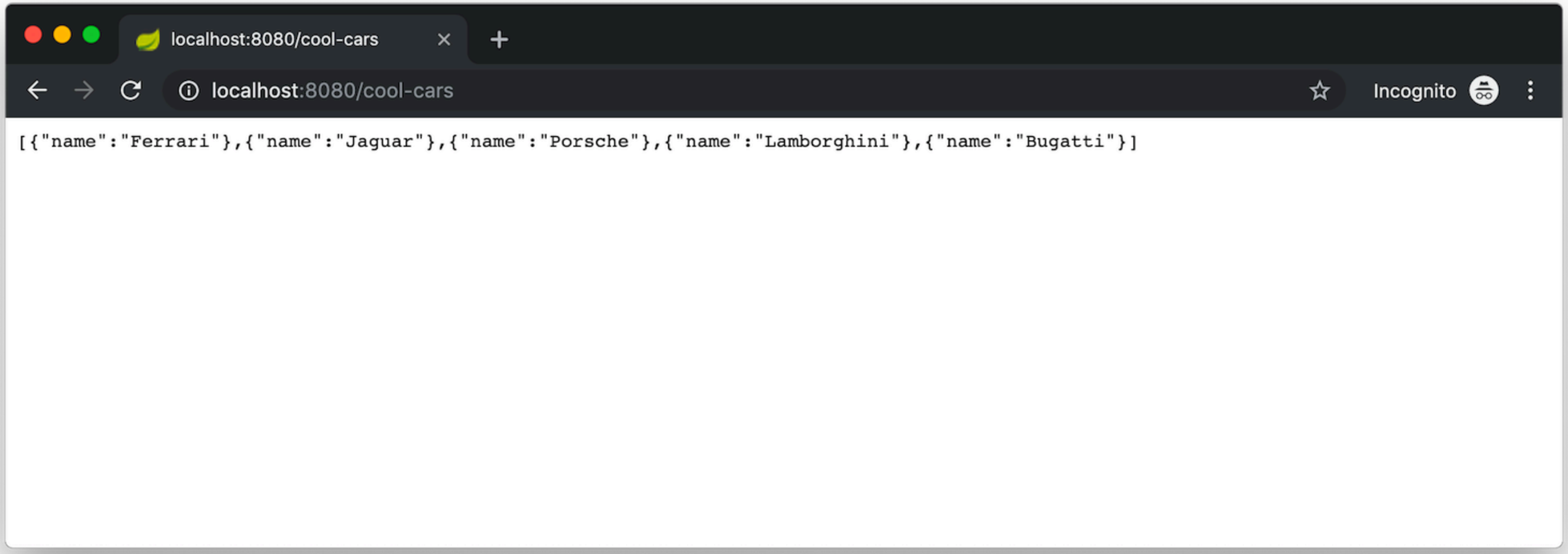
# Consume Cars API in Gateway

```
private boolean isCool(Car car) {
    return !car.getName().equals("AMC Gremlin") &&
            !car.getName().equals("Triumph Stag") &&
            !car.getName().equals("Ford Pinto") &&
            !car.getName().equals("Yugo GV");
}
```

# Start everything with ./mvnw



# Access https://localhost:8080/cool-cars



# Java Microservices with Spring Boot and Spring Cloud

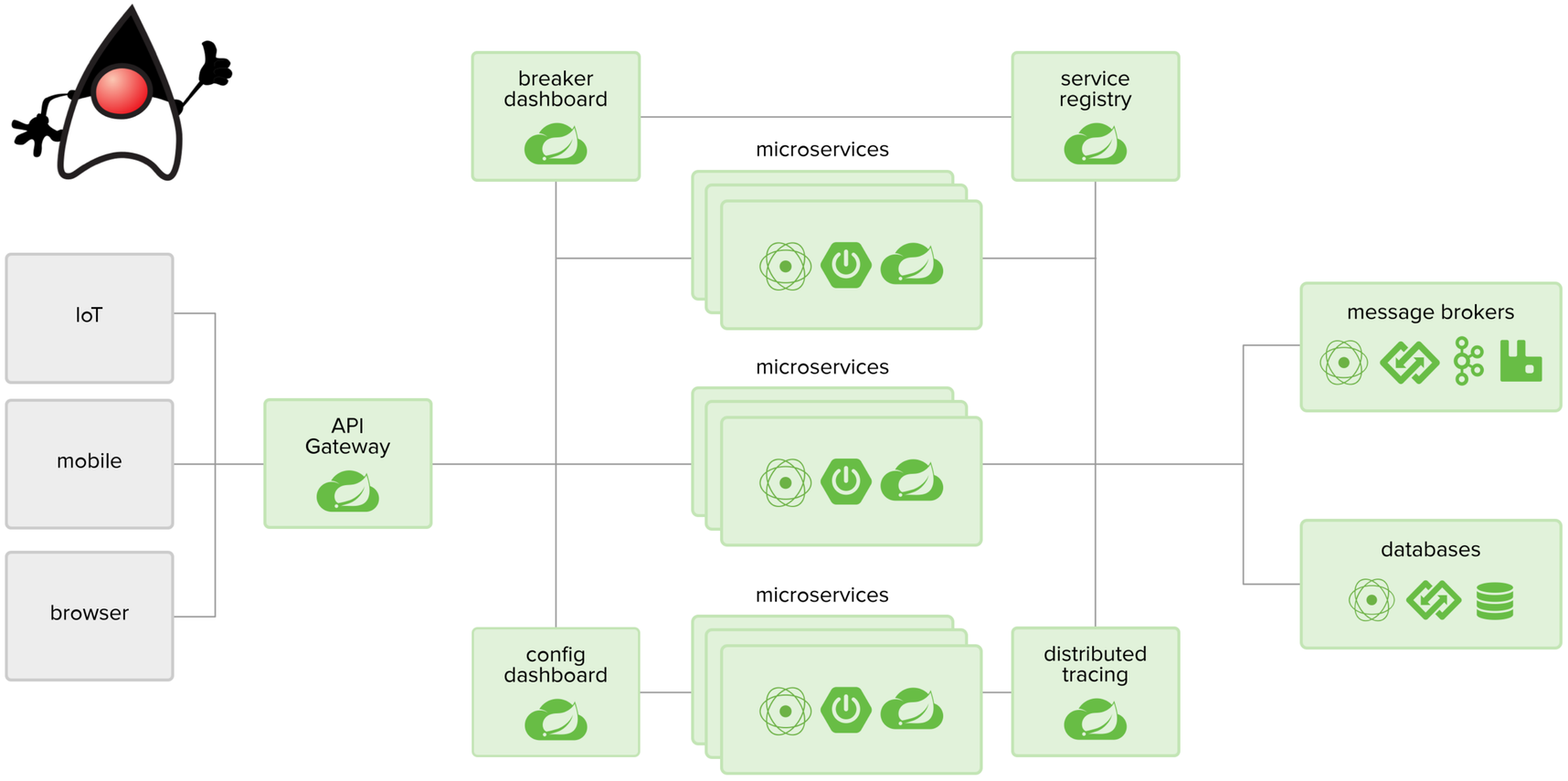
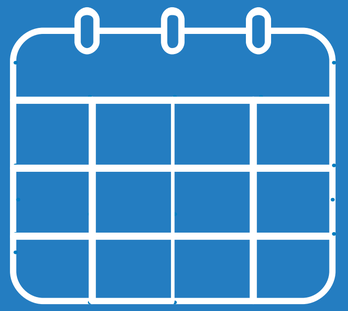



Diagram from https://spring.io

https://developer.okta.com/blog/2019/05/22/java-microservices-spring-boot-spring-cloud

# Part 2



**Microservices with JHipster**

What is JHipster?

Installing and Using JHipster

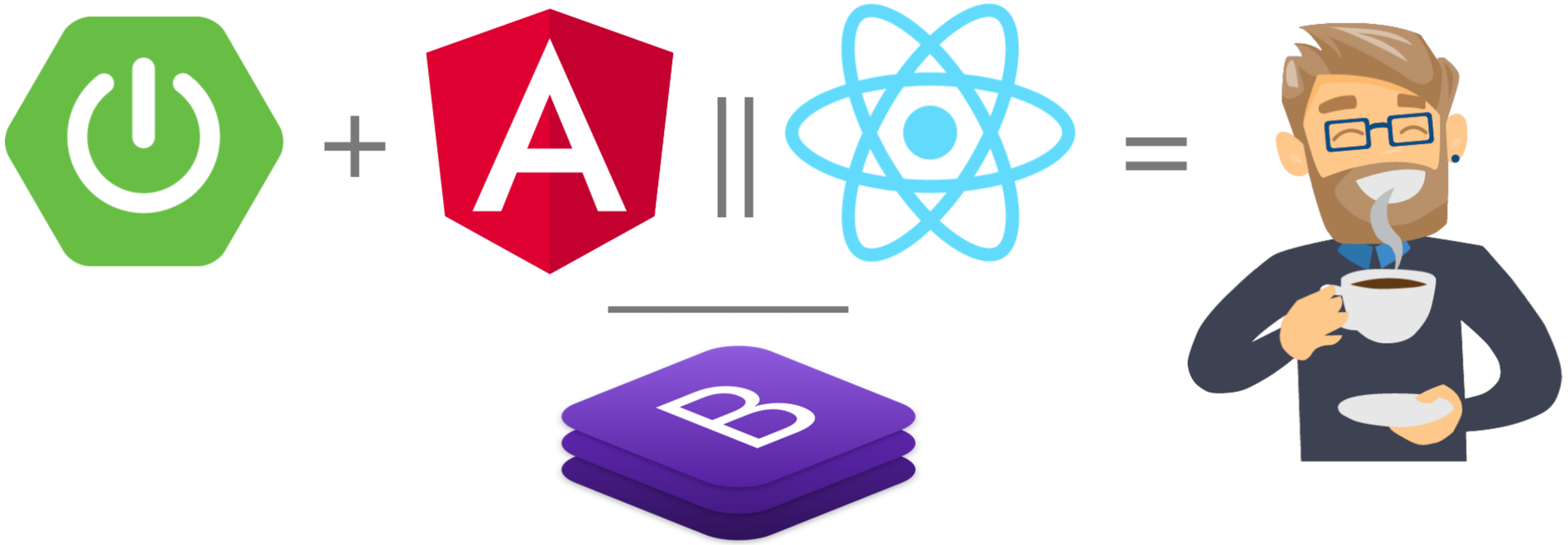
JHipster's Microservice Features

Progressive Web Applications Overview

# What is JHipster?

# JHipster

jhipster.tech



JHipster is a development platform to generate, develop and deploy
Spring Boot + Angular/React Web applications and Spring microservices.

and Vue! ✨

# JHipster is Inclusive

https://github.com/jhipster/jhipster-artwork

# JHipster Goals

A **high-performance and robust Java** stack on the server side with Spring Boot

A sleek, modern, **mobile-first front-end** with modern frameworks

A robust **microservice architecture** with JHipster Registry, Netflix OSS, Elastic Stack, and Docker

A **powerful workflow** to build your application with Webpack and Maven/Gradle

# How to Use JHipster

Install JHipster and Yeoman, using npm:

```
npm install -g generator-jhipster
```

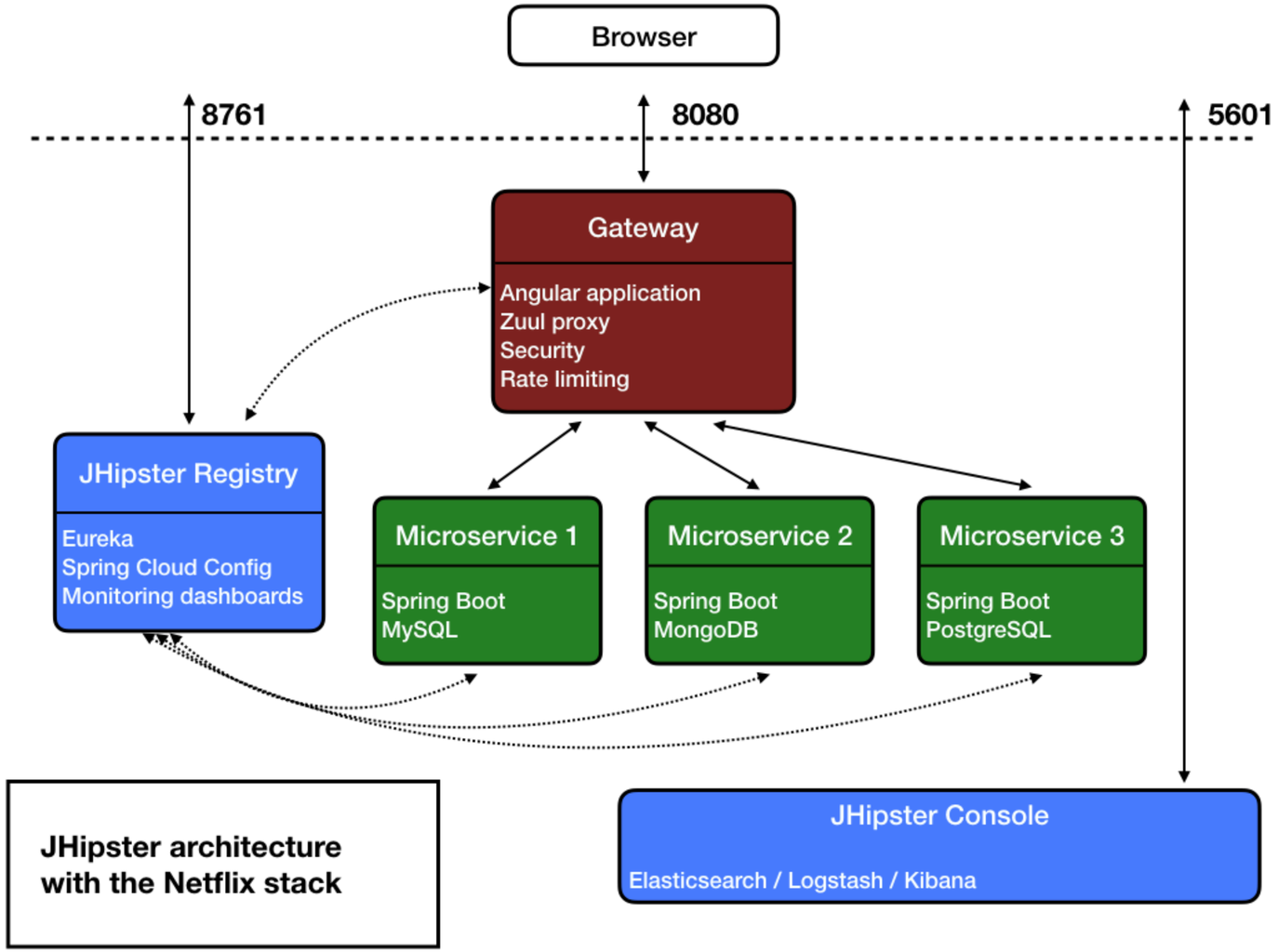
Create a directory and cd into it:

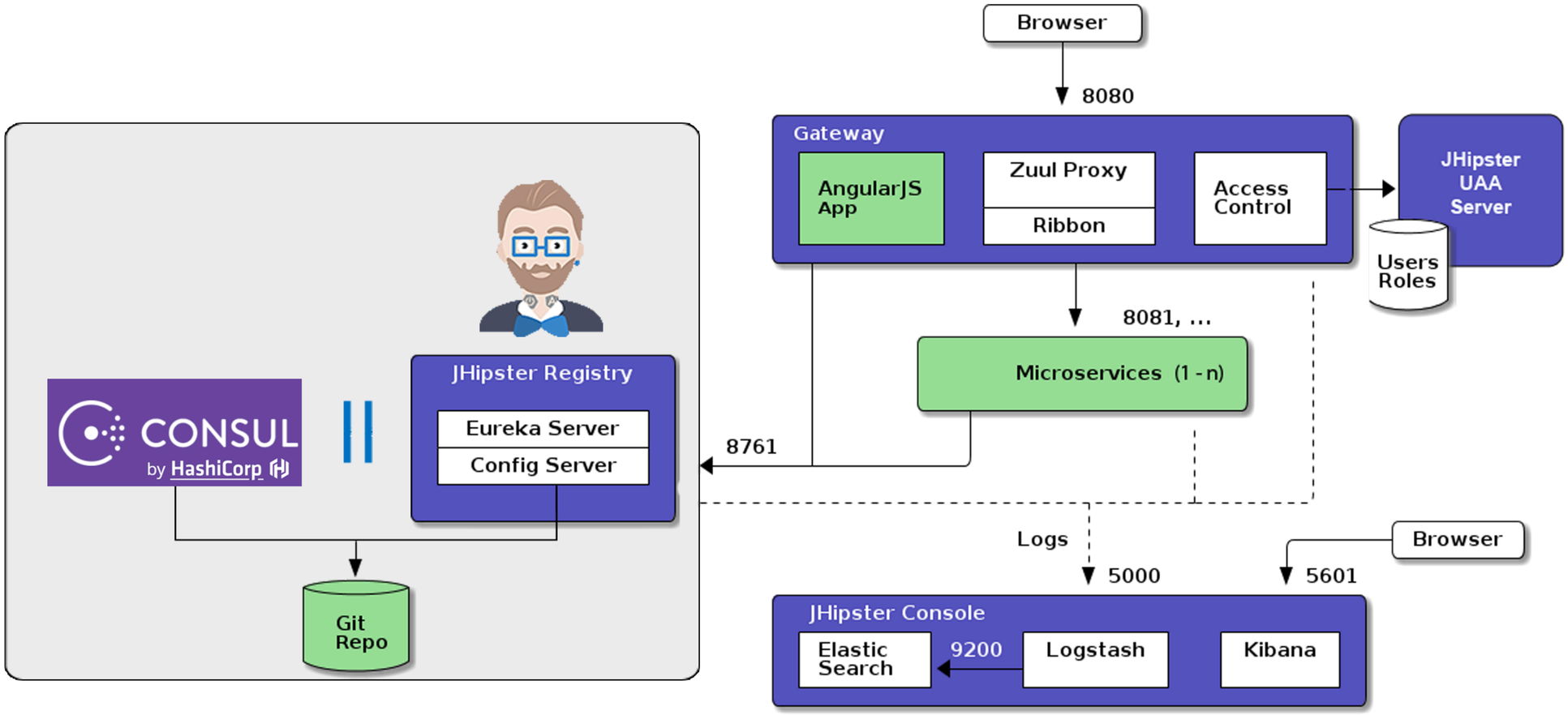
```
mkdir newapp && cd newapp
```

Run it!

```
jhipster
```

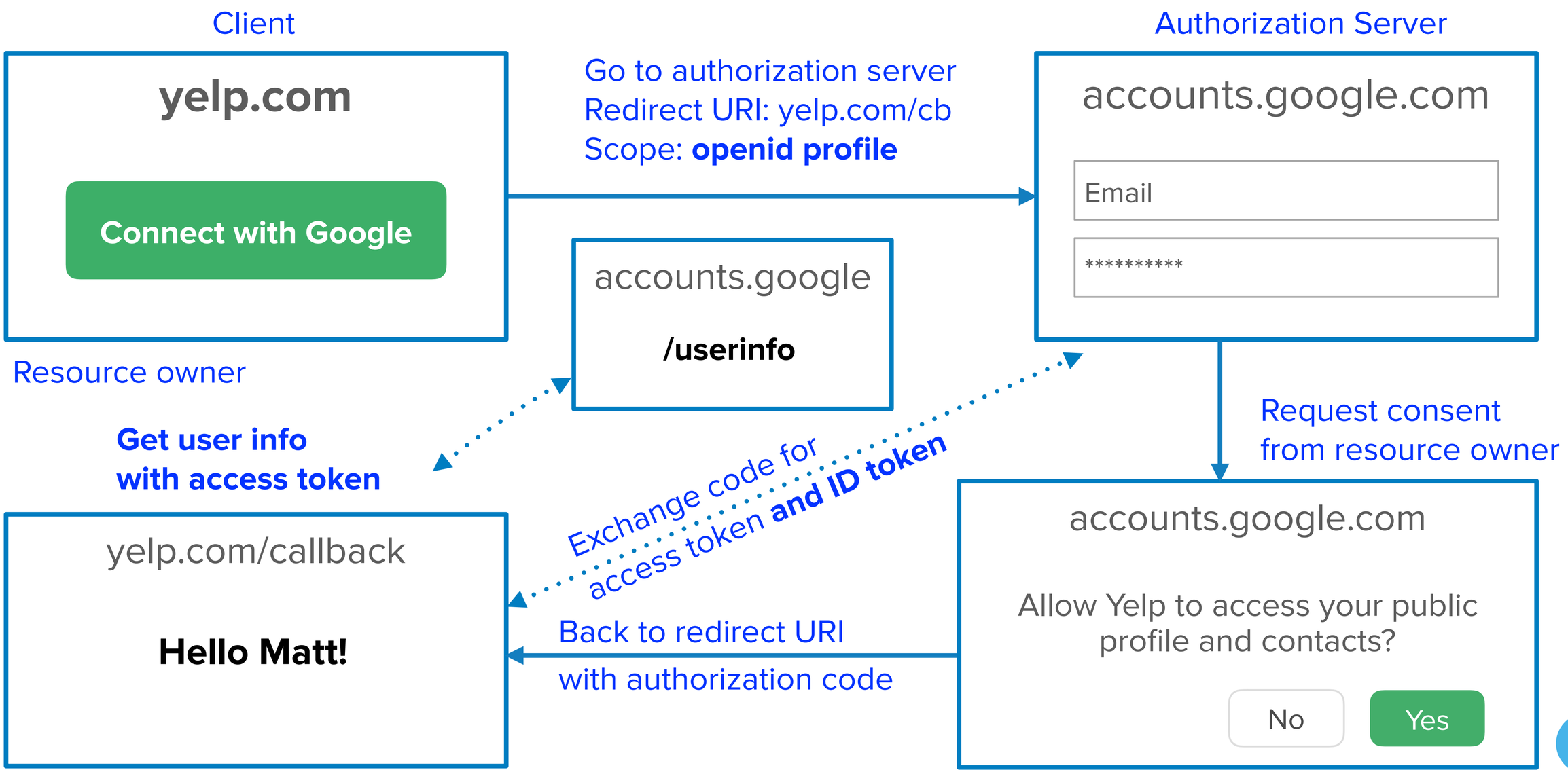
# Microservices with JHipster



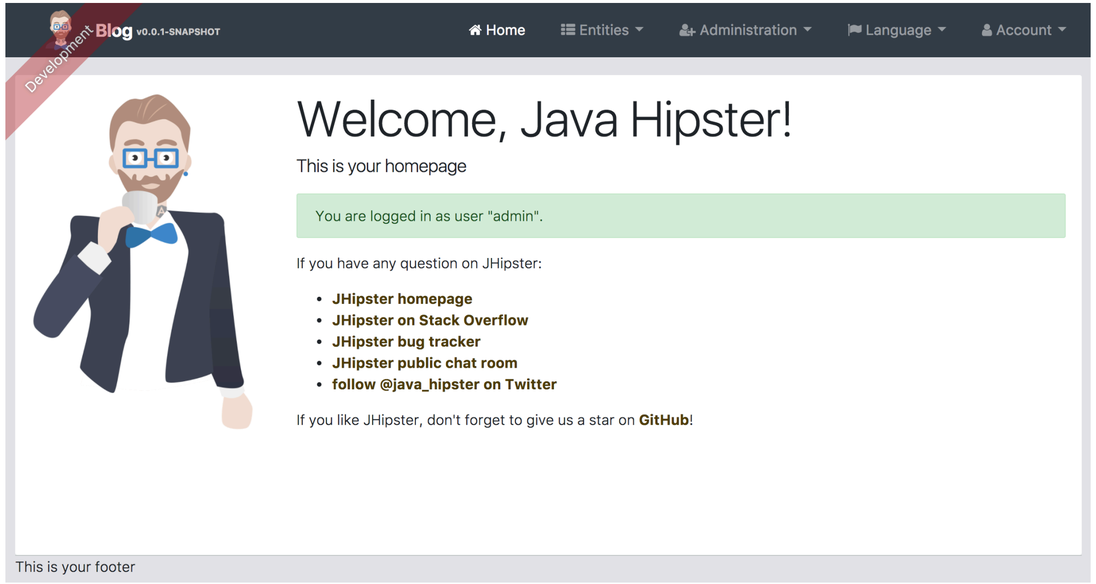


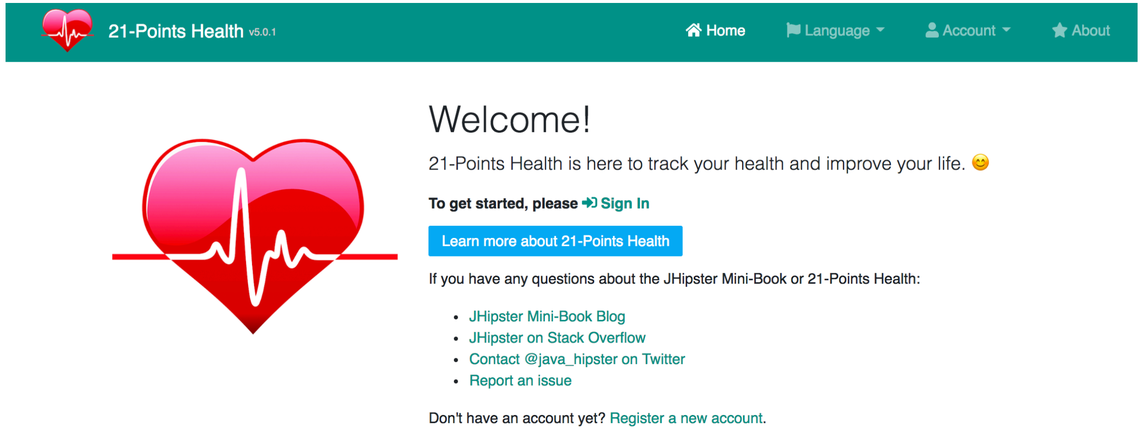

https://www.jhipster.tech/microservices-architecture

# OAuth 2.0 and OIDC



# Monolith Examples




**JHipster 6 Demo**

github.com/mraible/jhipster6-demo

youtu.be/uQqlO3IGpTU

**21-Points Health**

github.com/mraible/21-points

infoq.com/minibooks/jhipster-mini-book

# Progressive Web Apps

Originate from a **secure origin**, load while **offline**, and reference a **web app manifest**.

# Progressive Web Apps

Can be **installed** on your mobile device, look and act like a **native** application, but are distributed through the **web**.

# Progressive Web Apps

Are **fast!**

“We’ve failed on mobile.”

**Alex Russell**

https://youtu.be/K1SFnrf4jZo

# Enable PWA in JHipster

*gateway/src/main/webapp/index.html*

```
<script>
    if ('serviceWorker' in navigator) {
        window.addEventListener('load', function() {
            navigator.serviceWorker.register('/service-worker.js')
                .then(function () {
                    console.log('Service Worker Registered');
                });
        });
    }
</script>
```

# Force HTTPS in Spring Boot

*gateway/src/main/java/com/okta/developer/gateway/config/SecurityConfiguration.java*

```
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.requiresChannel()
            .requestMatchers(r -> r.getHeader("X-Forwarded-Proto") != null)
            .requiresSecure();
    }
}
```

https://developer.okta.com/blog/2018/07/30/10-ways-to-secure-spring-boot

# Demo

Using JHipster, create:

- A gateway
- A store microservices app
- A blog microservices app

Generate entities in apps and on gateway

Convert gateway to be a PWA

Run everything in Docker

**https://github.com/oktadeveloper/java-microservices-examples**

# JHipster 6.8.0 Lighthouse Report

| Performance | Accessibility | Best Practices | SEO | Progressive Web App |
|---|---|---|---|---|
| 91 | 95 | 86 | 93 | PWA |

0–49 50–89 90–100

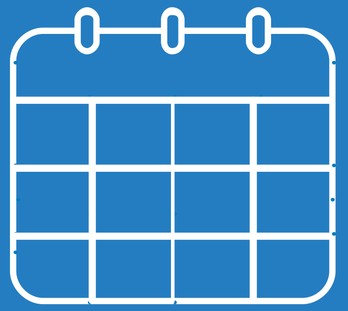
91

## Performance

**Metrics**

| | | | |
|---|---|---|---|
| First Contentful Paint | 2.8 s | First Meaningful Paint | 2.8 s |
| Speed Index | 2.8 s | First CPU Idle | 2.8 s |
| Time to Interactive | 3.3 s | Max Potential First Input Delay | 70 ms |

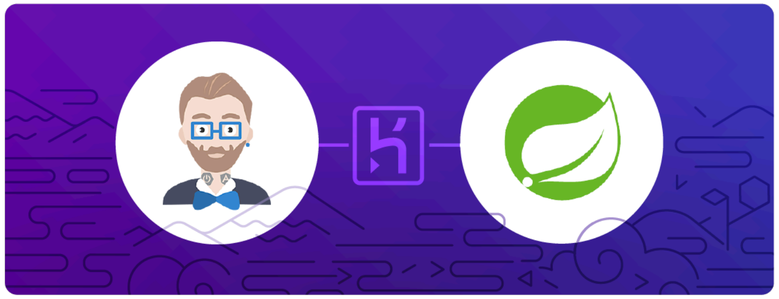
# Part 3



## Deploy to the Cloud

Options for Deploying JHipster

- Heroku
- Cloud Foundry
- AWS
- Google Cloud
- Microsoft Azure



HEROKU

## For monoliths:

```
jhipster heroku
```

## For microservices:

Deploy JHipster Registry

Build and deploy microservice

Build and deploy gateway

**http://bit.ly/heroku-jhipster-microservices**

CLOUDFOUNDRY

## For monoliths:

```
jhipster cloudfoundry
```

## For microservices:

Deploy JHipster Registry

Build and deploy microservice

Build and deploy gateway

**https://www.jhipster.tech/cloudfoundry/**


## Using Elastic Container Service

```
jhipster aws-containers
```

## Using Elastic Beanstalk

```
jhipster aws
```

## Boxfuse

```
boxfuse run -env=prod
```

http://www.jhipster.tech/aws

http://www.jhipster.tech/boxfuse

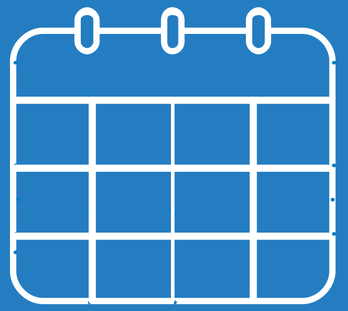
Google Cloud Platform

```
mvn package -Pprod jib:dockerBuild
```

```
jhipster kubernetes
./kubectl-apply.sh
kubectl get svc gateway
```

https://developer.okta.com/blog/2017/06/20/develop-microservices-with-jhipster

# Part 4

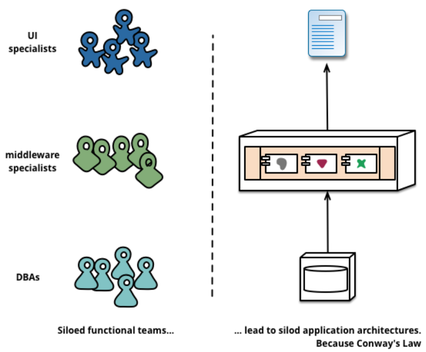


**JHipster Roadmap**

What You Learned

What's Next for JHipster

# What You Learned




Spring **Cloud**

okta

# Microservices with Spring Cloud Config and JHipster

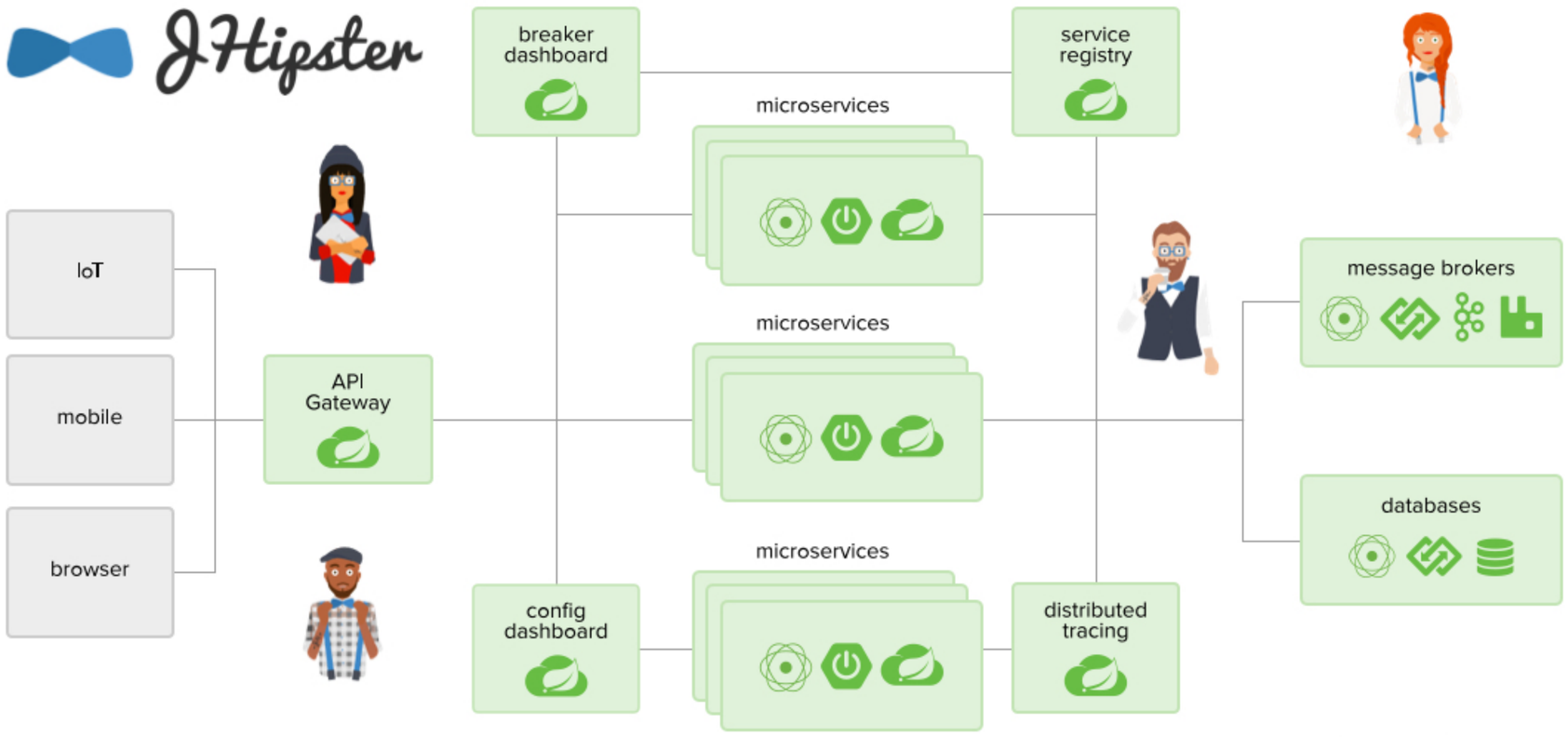



Diagram from https://spring.io

https://developer.okta.com/blog/2019/05/23/java-microservices-spring-cloud-config

# JHipster Mobile Apps and Microservices on Pluralsight

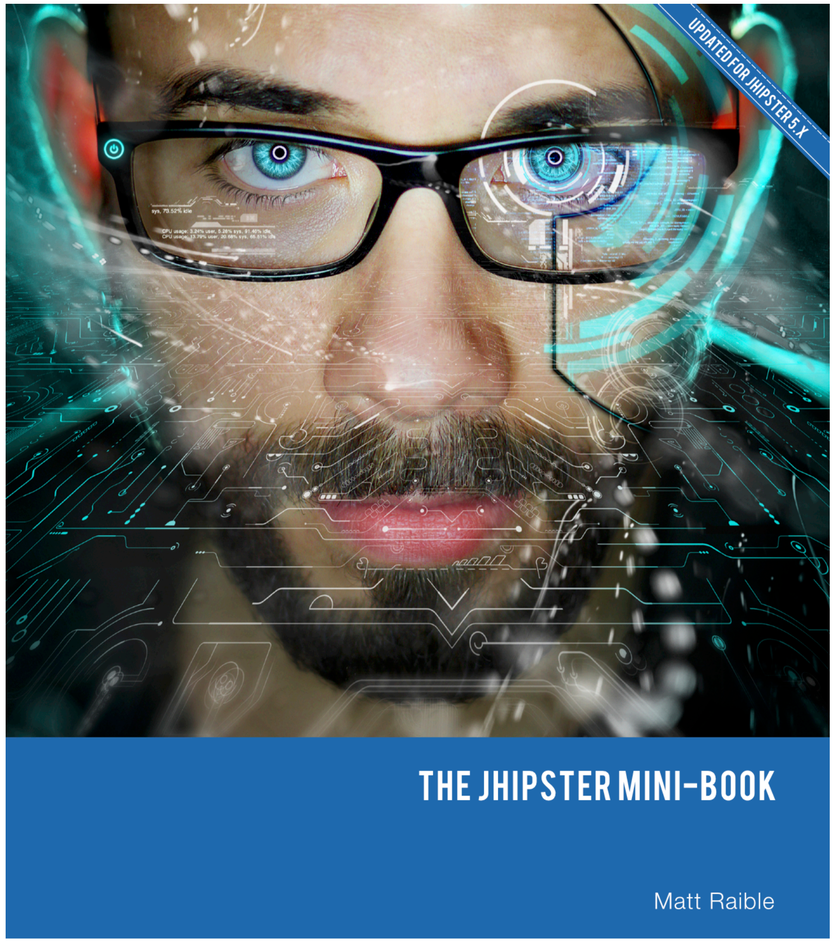

pluralsight.com/courses/play-by-play-developing-microservices-mobile-apps-jhipster

# What’s Next for JHipster?

Spring Boot 2.2

Full Reactive with WebFlux
and Spring Cloud Gateway

GraphQL and Micro Frontends




InfoQ
ENTERPRISE SOFTWARE DEVELOPMENT SERIES

# The JHipster Mini-Book

Written with **Asciidoctor**

FREE!

Free download from InfoQ:

**infoq.com/minibooks/jhipster-mini-book**

Quick and to the point, 164 pages

Developed a real world app:

**www.21-points.com**

Buy for $20 or download for **FREE**

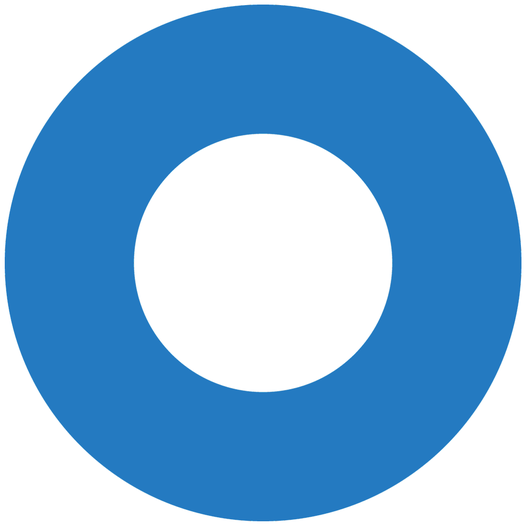
# Learn More

**Spring Boot**

spring.io/guides

**JHipster**

www.jhipster.tech



**Okta APIs**

developer.okta.com

stackoverflow

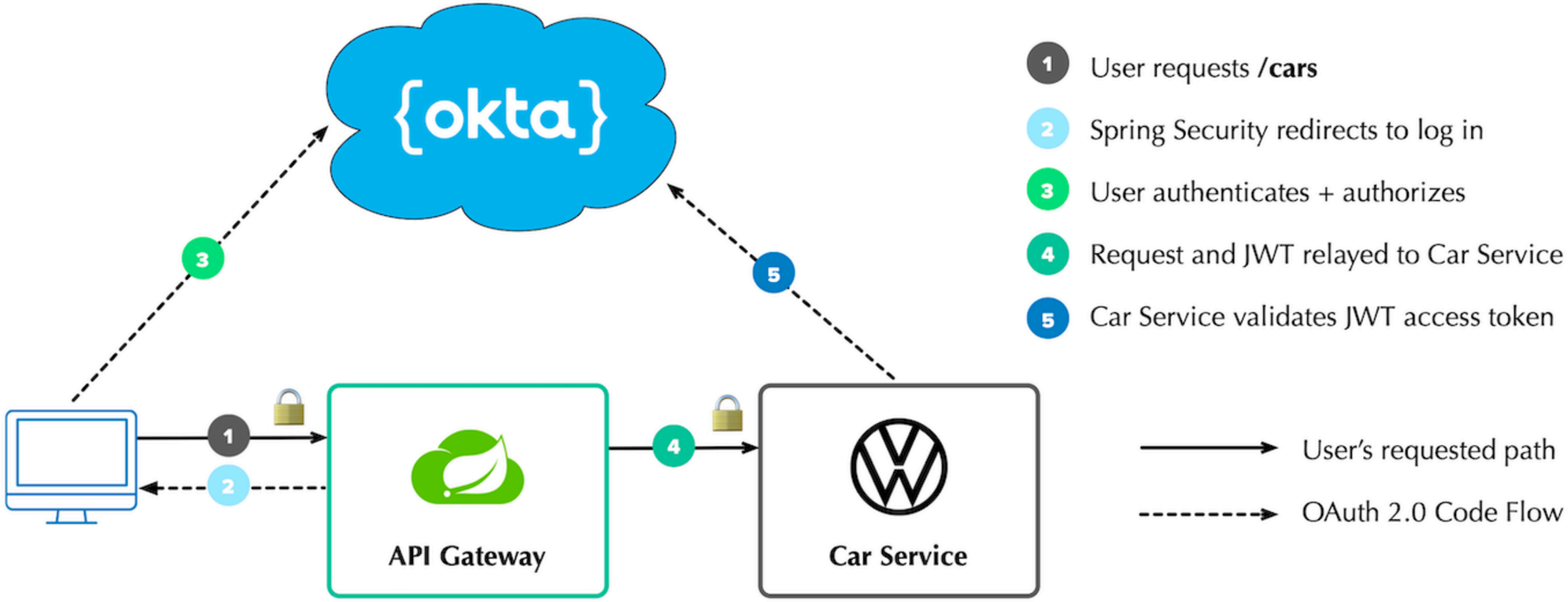
stackoverflow.com

developer.okta.com/blog

@oktadev

# Reactive Microservices with Spring Cloud Gateway

## Spring Cloud Gateway + OAuth 2.0




https://developer.okta.com/blog/2019/08/28/reactive-microservices-spring-cloud-gateway

# **Action:** Try JHipster!


https://developer.okta.com/blog/2019/04/04/java-11-java-12-jhipster-oidc

# Use the Source, Luke!

https://github.com/oktadeveloper/java-microservices-examples

developer.okta.com

# Thanks!

## Keep in Touch

raibledesigns.com

@mraible

## Presentations

speakerdeck.com/mraible

## Code

github.com/oktadeveloper


developer.okta.com