Design Patterns para Microsserviços com MicroProfile

Víctor Orozco

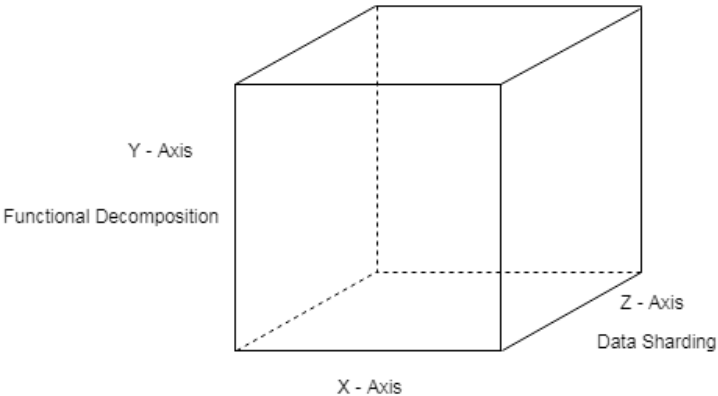
4 de Dezembro de 2020

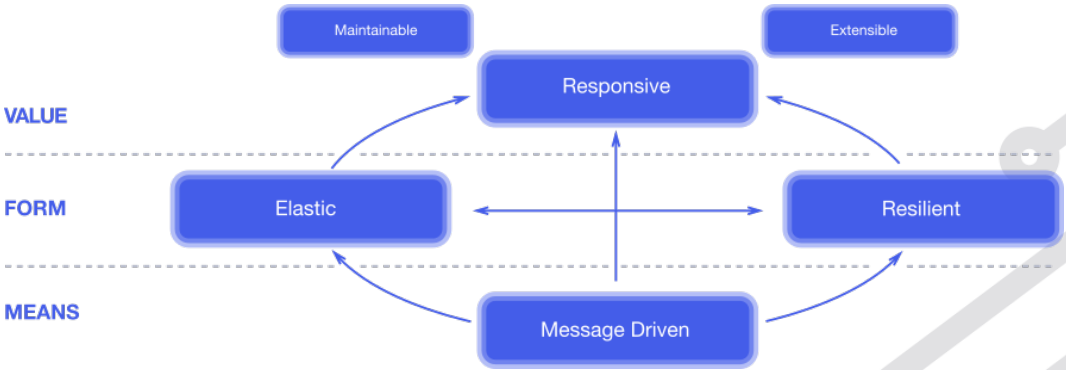
Nabenik

Design Patterns

Microserviços = Metapadrão arquitetural

Arquitetura que estrutura o aplicativo como um conjunto de **serviços colaborativos fracamente acoplados**. Esta abordagem corresponde ao eixo Y do *scale cube*. O objetivo final são **sistemas reativos**.





Application Server

- Transacionalidade distribuída (JTA/XA)
- Contratos (JNDI)
- Service discovery (JNDI)
- Deployment (EAR/Class Loaders/Dashboards)
- Métricas (JMX)
- Segurança (SoteriaRI/JACC)

Microserviços

Aplicativos Cloud Native

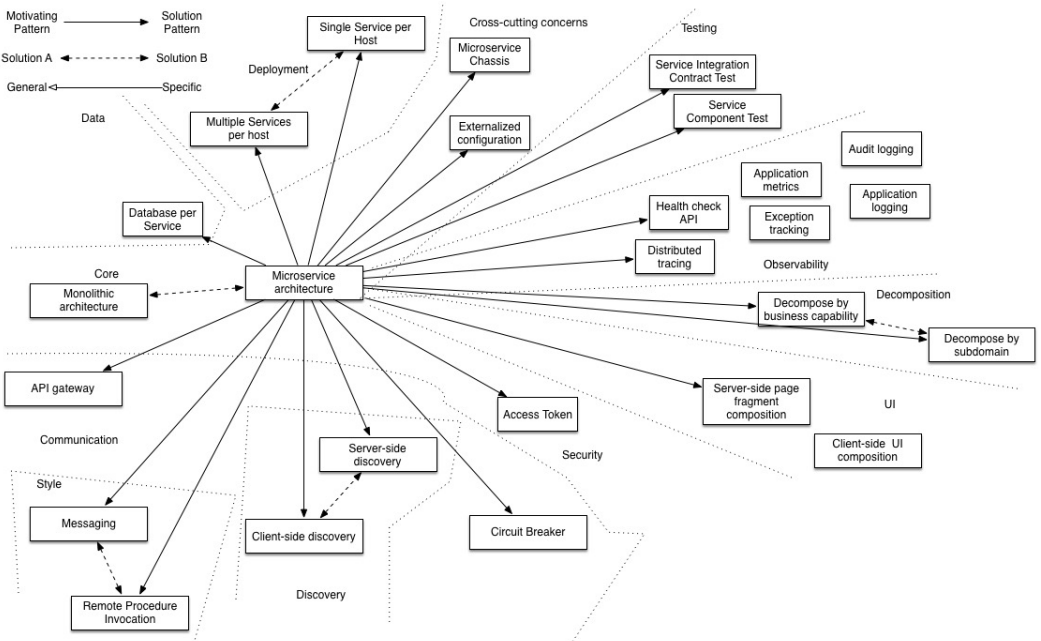
- Sistemas reativos
- 12 fatores Cloud Native
- Design patterns
- Domain Driven Design
- Microservice chassis e/ou service mesh
- Orquestração de contêineres

Microserviços = Metapadrão arquitetural

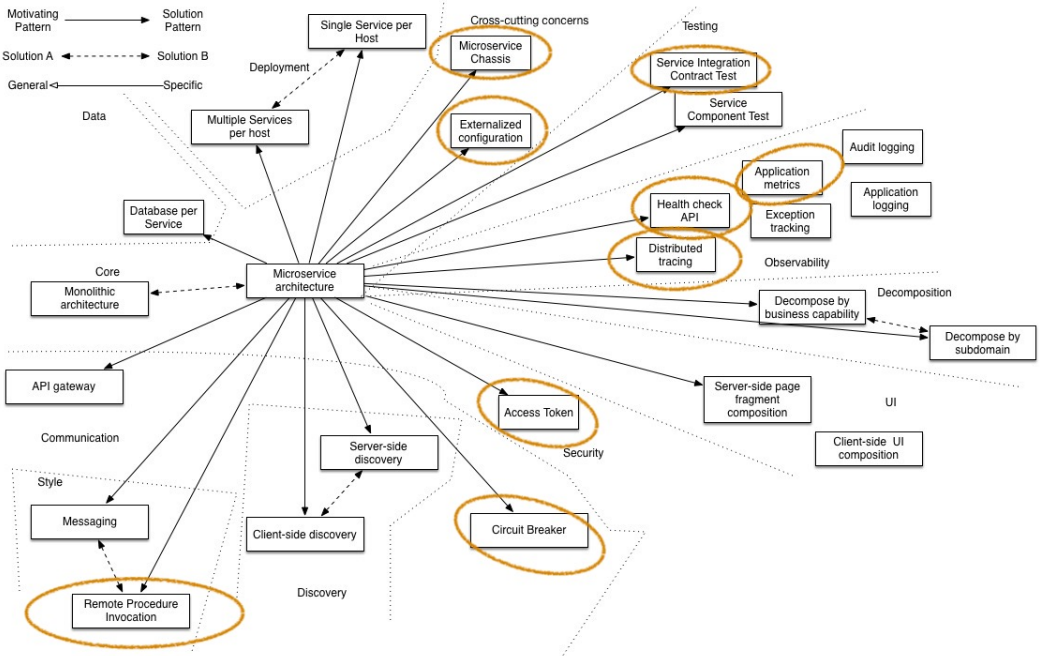
Cloud Native

- (Gostamos de ter) Sistemas reativos
- (É possível com a metodologia dos) 12 fatores Cloud Native
- (Usamos soluções testadas chamadas de) design patterns
- (Fragmentamos o sistema mediante) Domain Driven Design
- **(Implementamos os serviços com frameworks) microservice chassis e/ou service mesh**
- (E fazemos deployment) mediante orquestração de contêineres

Os Dessign Patterns são uma linguagem comum para *implementar e avaliar plataformas Cloud Native*.


Motivating Pattern
Solution Pattern
Solution A
Solution B
General
Specific
Data
Deployment
Single Service per Host
Multiple Services per host
Database per Service
Cross-cutting concerns
Microservice Chassis
Externalized configuration
Testing
Service Integration Contract Test
Service Component Test
Audit logging
Application metrics
Application logging
Health check API
Exception tracking
Distributed tracing
Observability
Core
Monolithic architecture
Microservice architecture
Decomposition
Decompose by business capability
Decompose by subdomain
API gateway
Server-side page fragment composition
UI
Client-side UI composition
Access Token
Security
Communication
Server-side discovery
Style
Messaging
Client-side discovery
Circuit Breaker
Discovery
Remote Procedure Invocation

T)

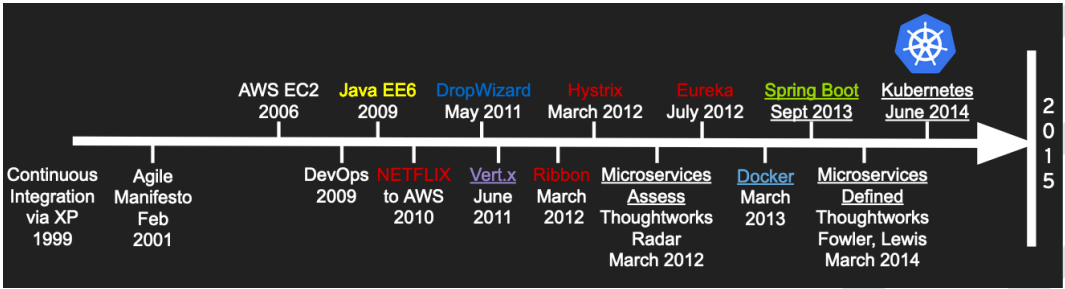

Motivating Pattern
Solution Pattern
Solution A
Solution B
General
Specific
Data
Deployment
Single Service per Host
Multiple Services per host
Database per Service
Cross-cutting concerns
Microservice Chassis
Externalized configuration
Testing
Service Integration Contract Test
Service Component Test
Audit logging
Application metrics
Application logging
Health check API
Exception tracking
Distributed tracing
Observability
Core
Monolithic architecture
Microservice architecture
Decomposition
Decompose by business capability
Decompose by subdomain
API gateway
Server-side page fragment composition
UI
Client-side UI composition
Access Token
Security
Communication
Server-side discovery
Style
Messaging
Client-side discovery
Circuit Breaker
Discovery
Remote Procedure Invocation

T)

MicroProfile

A historia

Os *patterns* foram criados antes/junto com os chassis e antes do service mesh/K8S



Créditos: Rafael Benevides

Chassis

No ponto de vista dos design patterns. Os frameworks ”cloud native”são soluções para problemas ”cross-cuting concerns”.

Chassis EE

O MicroProfile é uma especificação para chassis fundamentada no Java/Jakarta EE

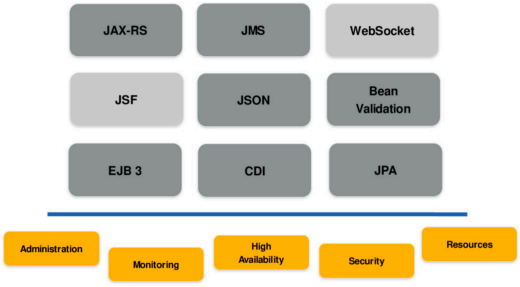
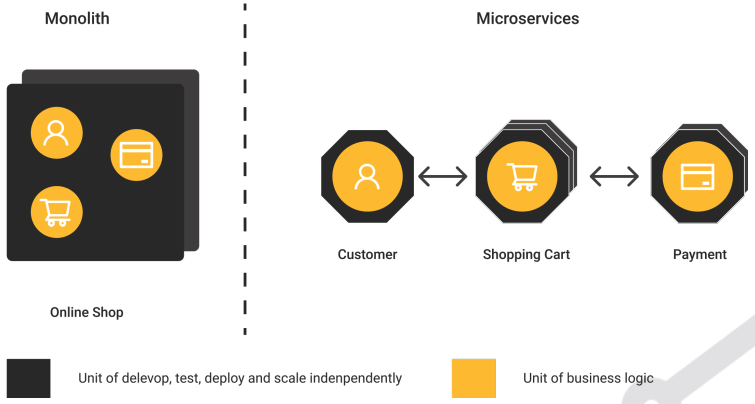
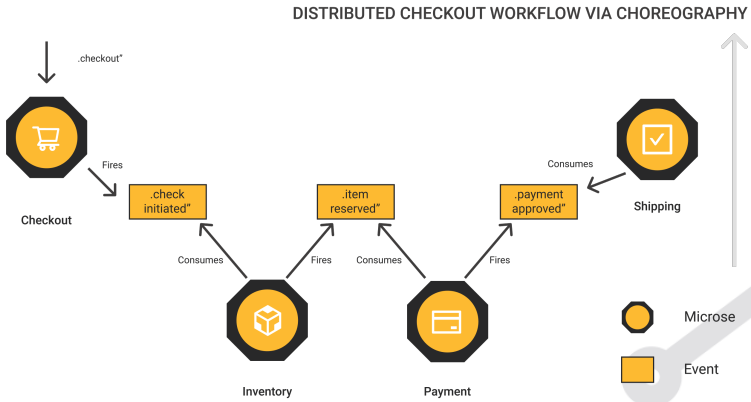


Figura 1: Créditos: Reza Rahman

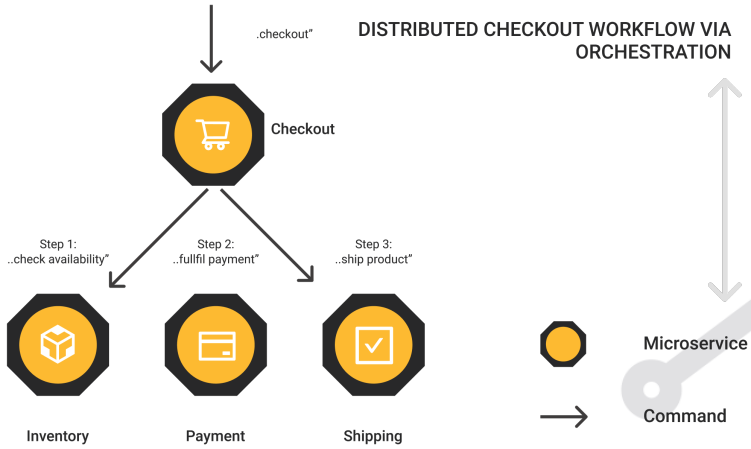
MicroProfile



Patterns complementarios - Event Sourcing, CQRS



Patterns complementarios - SAGA



Cross-cutting concerns no mundo real

- **Health checks & Metrics** - Coletar metricas (Prometheus/Grafana) e estabelecer regras no deployment
- **Resilence & Fault Tolerance** - Sobreposição entre service Mesh -e.g. Likerd, Istio- e MicroProfile Fault Tolerance
- **Configuration** - Injeção de configuração no ambiente
- **Authentication & Authorization** - API Gateway + MicroProfile JWT
- **Standarized documentation** - OpenAPI + Swagger Server
- **Tracing** - MicroProfile Tracing + Zipkin
- **Remote Procedure & Messaging** - JAX-RS + MicroProfile Rest Client + K8S service discovery

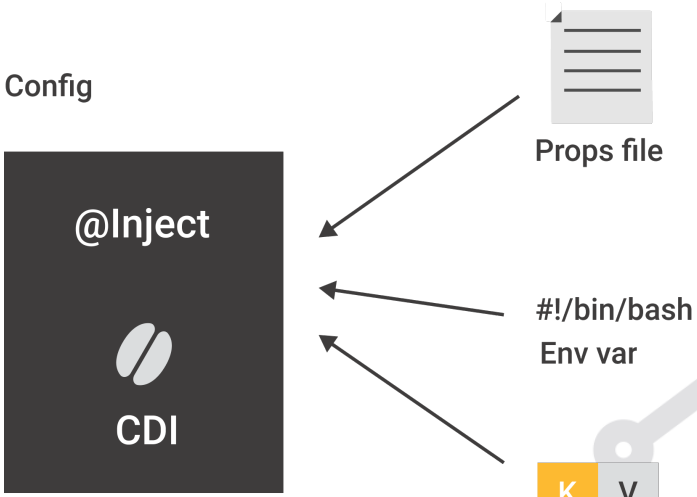
MicroProfile - APIs

EE + MicroProfile - Demo

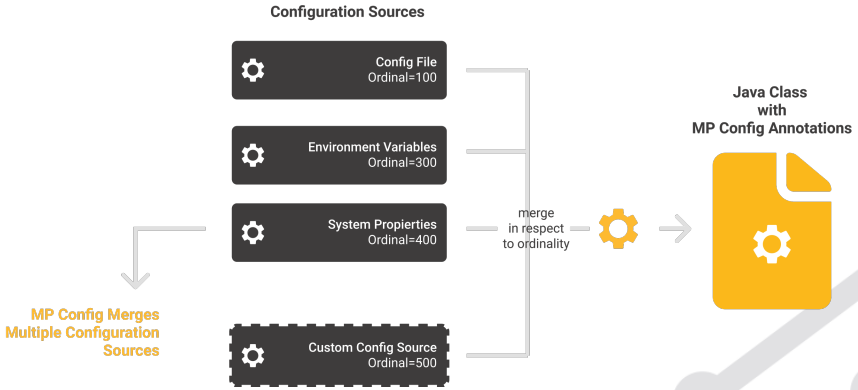
Oracle Helidon (Chassis) + Oracle Kubernetes Engine

- Configuração
- Contrato e cliente REST
- Resiliência
- Deployment
- Health Check
- Metricas

Config



Config



Config

```
@Inject
@ConfigProperty(name = "omdbservice.url")
String omdbDaemonServiceUrl;
```

Ext. da configuração(VM, Docker, Kubernetes)

Config

```
@Inject
@ConfigProperty(name = "application.currency")
private String currency;

@Inject
@ConfigProperty(name = "application.list.maxSize",
        defaultValue="10")
private Integer maxSize;
```

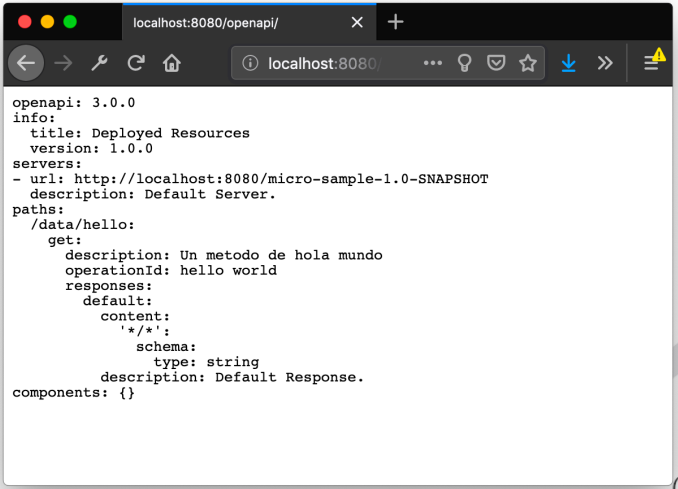
Documentação padronizada

```
@ApplicationPath("/api")
@OpenAPIDefinition(info = @Info(
        title = "Example␣application",
        version = "1.0.0",
        contact = @Contact(
        name = "Victor␣Orozoc",
        email = "vorozco@nabenik.com",
        url = "http://vorozco.com")
        ),
        servers = {
                @Server(url = "/example",
                description = "localhost")
        }
)
public class ApplicationConfig extends Application {
```

Documentação padronizada

```
@GET @Path("/{key}")
@Operation(description = "Get␣the␣value␣for␣this␣key")
@APIResponses({
        @APIResponse(responseCode = "200",
        description = "Successful,␣returning␣the␣value")
})
@Produces(MediaType.TEXT_PLAIN)
public Response getConfigValue(@PathParam("key") String key)
```

OpenAPI

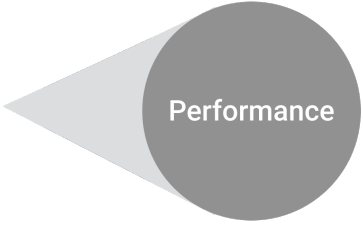


Fault Tolerance

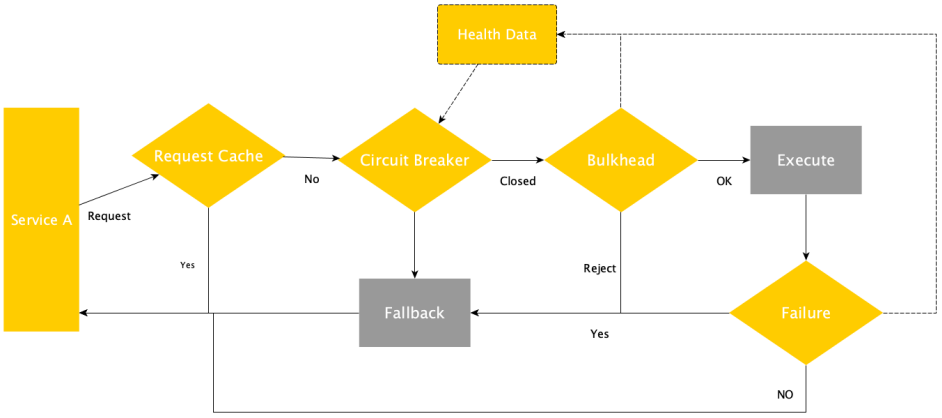
~~a ( )~~ X

aFallback()

Metrics



Fault Tolerance + Metrics



Fault tolerance

Regras e alternativas

- Circuit Breaker
- Bulkhead
- Retry
- Timeout
- Fallback

Fault tolerance - Retry

```
@Retry(delay = 400, maxDuration= 3200, jitter= 400, maxRetries = 10)
public Connection serviceA() {
        ...
}

@Retry(retryOn = {IOException.class})
public void serviceB() {
        ...
}
```

Fault tolerance - CircuitBreaker

```
@CircuitBreaker(successThreshold = 10,
        requestVolumeThreshold = 4,
        failureRatio=0.75,
        delay = 1000)
public Connection serviceA() {
        Connection conn = null;
        conn = connectionService();
        return conn;
}
```

Fault tolerance - Bulkhead

```
@Bulkhead(5)
public Connection serviceA() {
        Connection conn = null;
        conn = connectionService();
        return conn;
}

@Asynchronous
@Bulkhead(value = 5, waitingTaskQueue = 8)
public Future<Connection> serviceA() {
        Connection conn = null;
        conn = connectionService();
        return CompletableFuture.completedFuture(conn);
}
```

Fault tolerance - Fallback, Timeout

```
@GET
@Path("/{id:[a-z]*[0-9][0-9]*}")
@Fallback(fallbackMethod = "findByIdFallBack")
@Timeout(TIMEOUT)
public Response findById(@PathParam("id")
final String imdbId) {
...
}

public Response findByIdFallBack(@PathParam("id")
final String imdbId) {
...
}
```

Fault tolerance - Fallback Handler, Timeout

```
@GET
@Path("/{id:[a-z]*[0-9][0-9]*}")
@Fallback(MovieFindAllFallbackHandler.class)
@Timeout(TIMEOUT)
public Response findById(@PathParam("id")
final String imdbId) {
...
}

public class MovieFindAllFallbackHandler
        implements FallbackHandler<List> {
        @Override
        public List handle(final ExecutionContext context) {
                return Stream.of("Star␣Wars",
                "The␣Matrix", "Cantinflas").collect(toList());
        }
}
```

Metrics

- JSON or OpenMetrics (Prometheus)
- Vendor
- Base
- Application

Opções

- Counted
- Gauge
- Metered
- Timed
- Histogram

Metrics - Counted

```
@Inject
@Metric
Counter failedQueries;

@GET
@Path("/{id:[a-z]*[0-9][0-9]*}")
@Fallback(fallbackMethod = "findByIdFallBack")
@Timeout(TIMEOUT)
public Response findById(@PathParam("id")
final String imdbId) {
...
}

public Response findByIdFallBack(@PathParam("id")
final String imdbId) {
        ...
        failedQueries.inc();
}
```

Metrics - Gauge

Inc-dec

```
@Gauge(unit = "ExternalDatabases",name = "movieDatabases", absolute = true)
public long getDatabases() {
        return 99; //Any value
}
```

```
/metrics/application/movieDatabases
```

Metrics - Metered

Events rate

```
@Metered(name = "moviesRetrieved",
        unit = MetricUnits.MINUTES,
        description = "Metrics␣to␣monitor␣movies",
        absolute = true)
public Response findExpandedById(
        @PathParam("id") final Long id)

/metrics/application/movieDatabases
```

Metrics- Timed

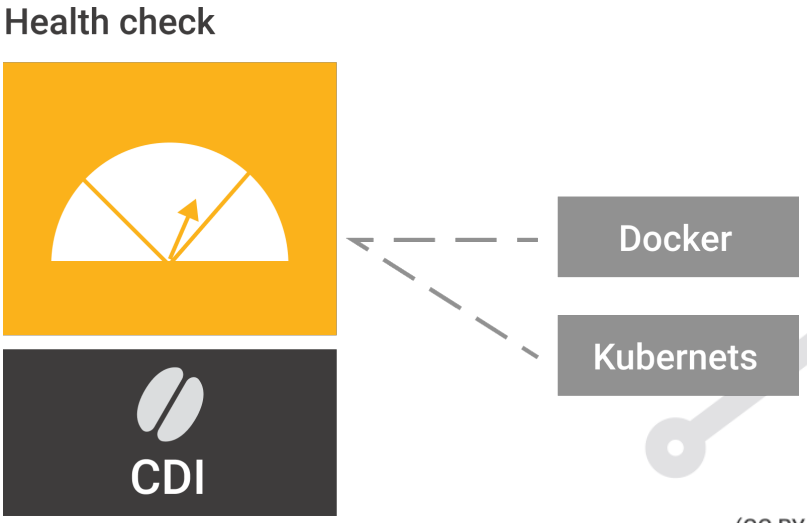
Performance

```
@Timed(name = "moviesDelay",
        description = "Time␣to␣retrieve␣a␣movie",
        unit = MetricUnits.MINUTES,
        absolute = true)
public Response findExpandedById(
        @PathParam("id") final Long id)

/metrics/application/moviesDelay
```

Metrics - Histogram

Distribuciones

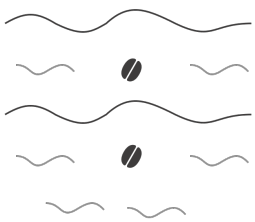
```
@Inject
MetricRegistry registry;

@POST
@Path("/add/{attendees}")
public Response addAttendees(
        @PathParam("attendees") Long attendees) {
        Metadata metadata =
                new Metadata("matrix␣attendees",
                        MetricType.HISTOGRAM);
        Histogram histogram =
                registry.histogram(metadata);
        histogram.update(attendees);
        return Response.ok().build();
}
```



Health Check

```
@Override
public HealthCheckResponse call() {
        return HealthCheckResponse.named("TaVivoAinda")
                .withData("key1", "val1")
                .withData("key2", "val2")
                .up()
                .build();

}
```

JWT

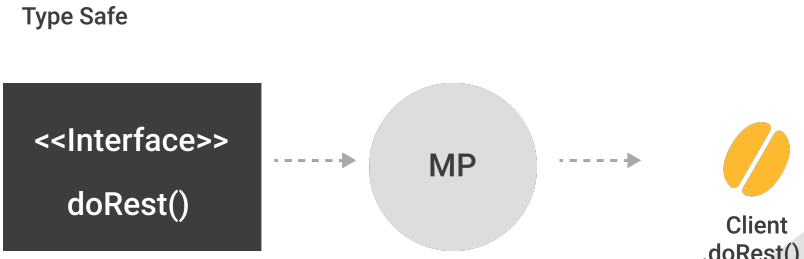


@Inject
Principal

@Inject
Realm

```
@LoginConfig(authMethod = "MP-JWT")
public class ApplicationConfig extends Application {
}

@Inject
private JsonWebToken jwtPrincipal;

@Inject
@Claim("email")
private String email;
```

TypeSafe



TypeSafe

```
@Path("/playlist")
@Consumes("application/json")
public interface MusicPlaylistService {

        @GET
        List<String> getPlaylistNames();


        @PUT
        @Path("/{playlistName}")
        long updatePlayList(@PathParam("playlistName")
                String name,
                List<Song> playlist)
                throws UnknownPlaylistException;
}
```

12 fatores cloud native (Heroku)

Microprofile

- Config
- Backing service
- Disposability

Cloud

- Codebase (Git-Flow)
- Dependencies (Maven)
- Build, Release, Run
- Processes (Pipelines)
- Port binding
- Concurrency (Docker - k8s)
- Dev / Prod parity
- Logs
- Admin process

Víctor Orozco

- vorozco@nabenik.com
- @tuxtor
- http://vorozco.com
- http://tuxtor.shekalug.org