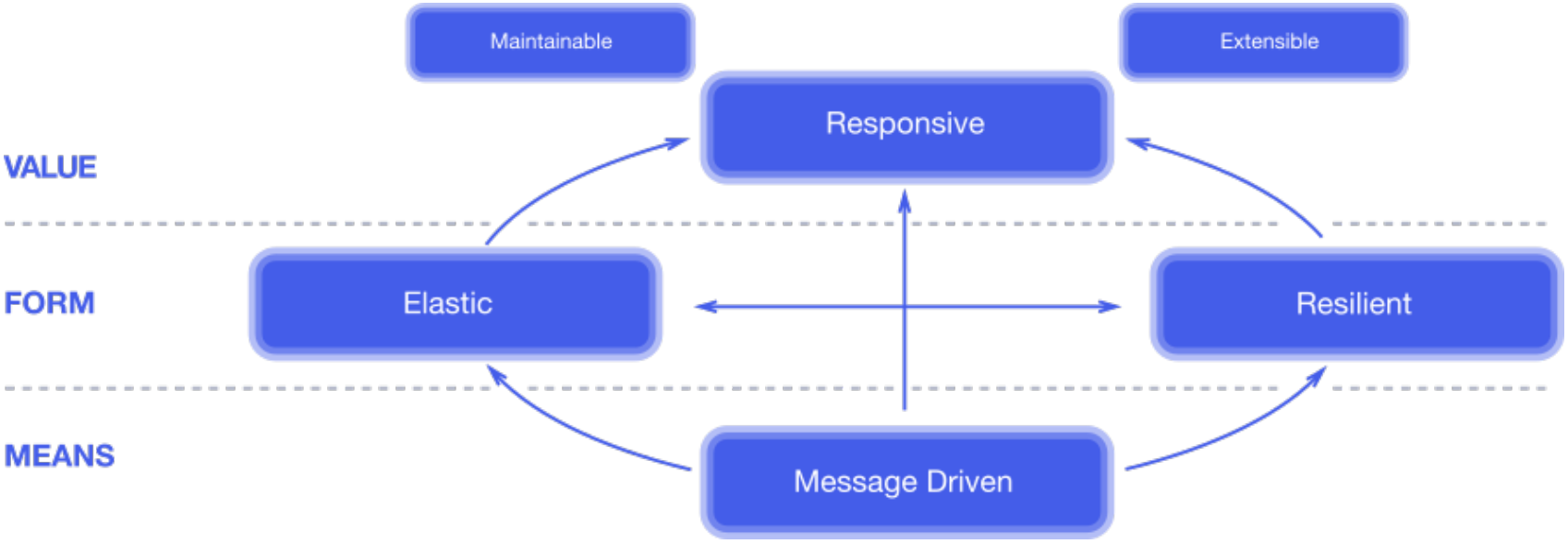
Docker para desarrolladores Java

Víctor Orozco

@tuxtor

11 de marzo de 2019

Aplicaciones reactivas



Escalabilidad

NABENIK

Monólito

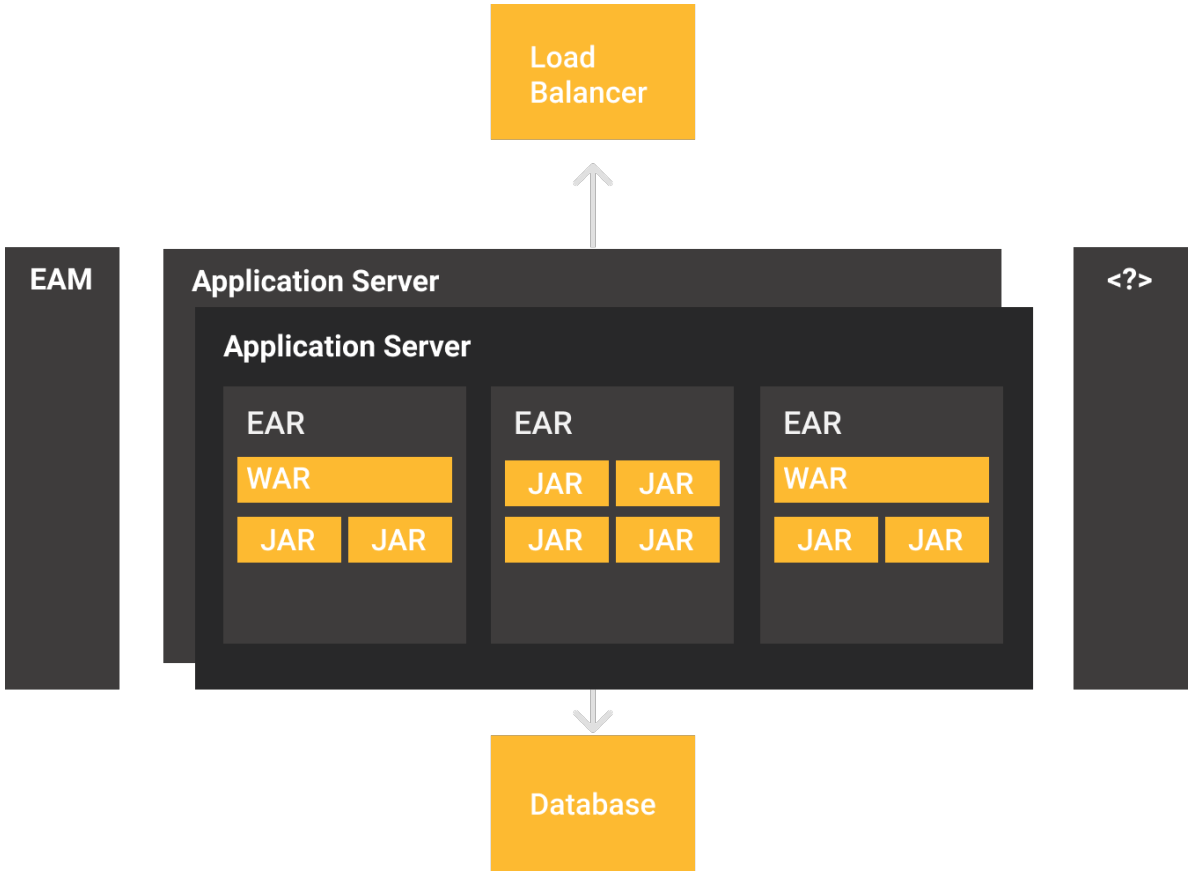


Figura 1: Monólito

ESB

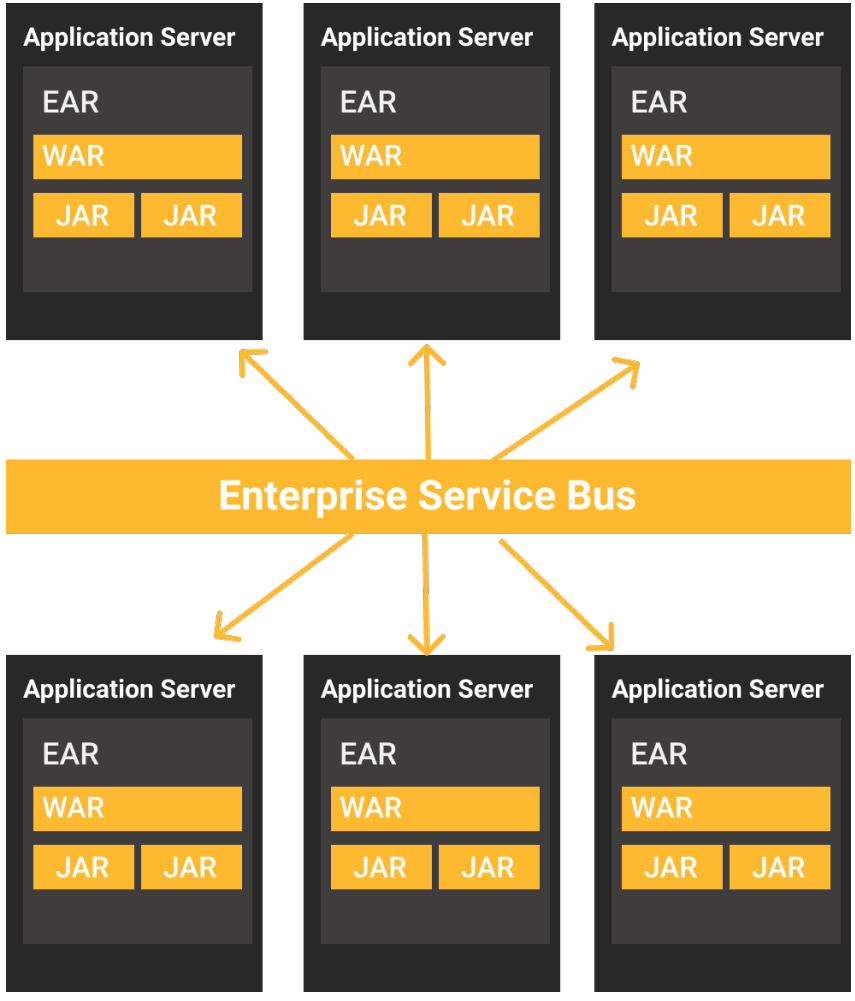


Figura 2: ESB

NABENIK

Microservicios

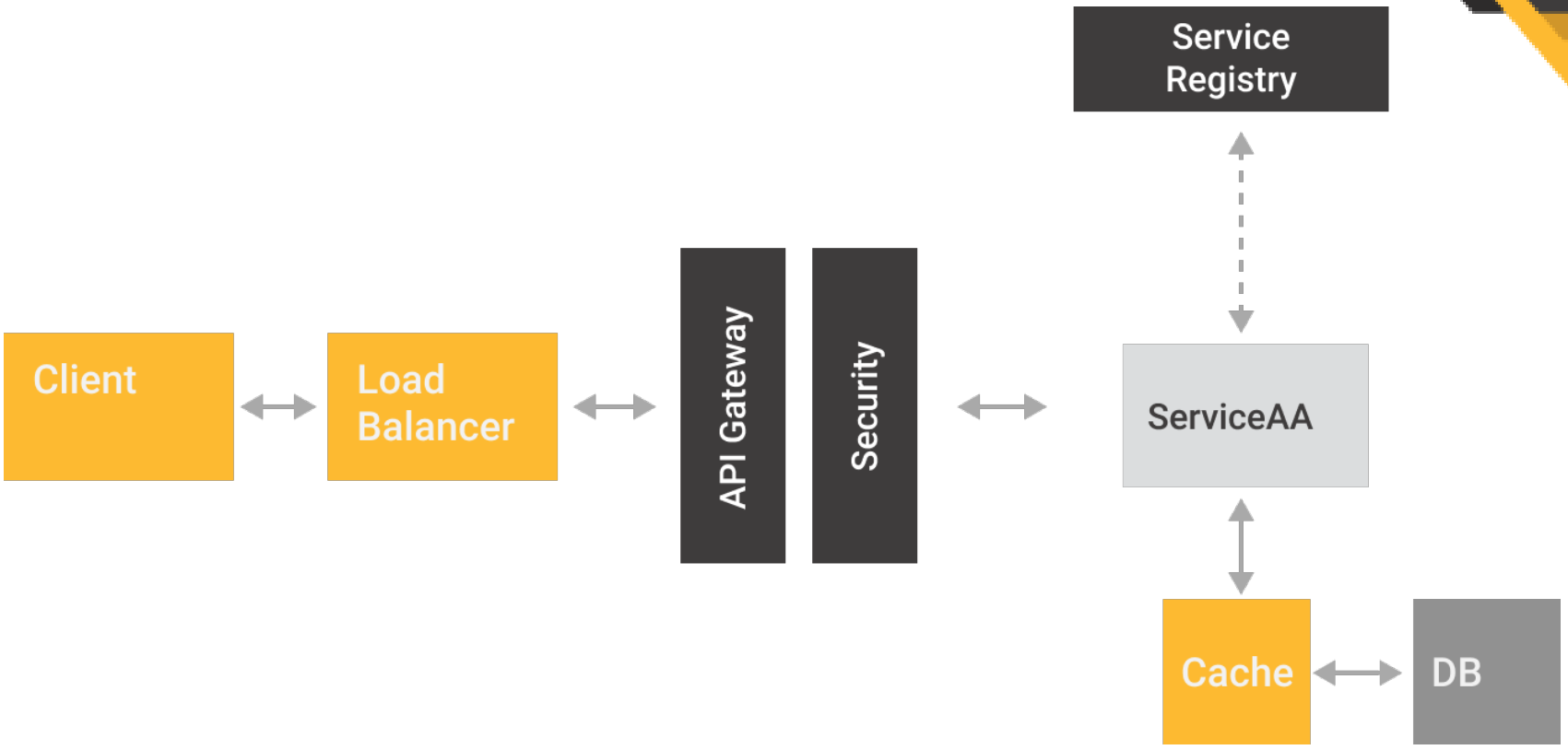
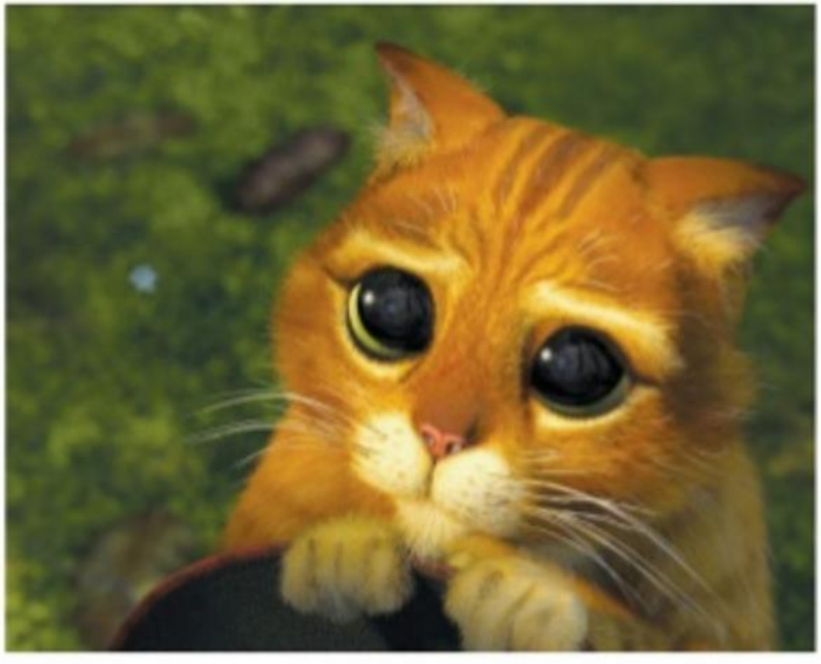


Figura 3: Microservicios

Monolito vs. Microservicios



NABENIK

Monolito vs. Microservicios

NABENIK

Docker

NABENIK

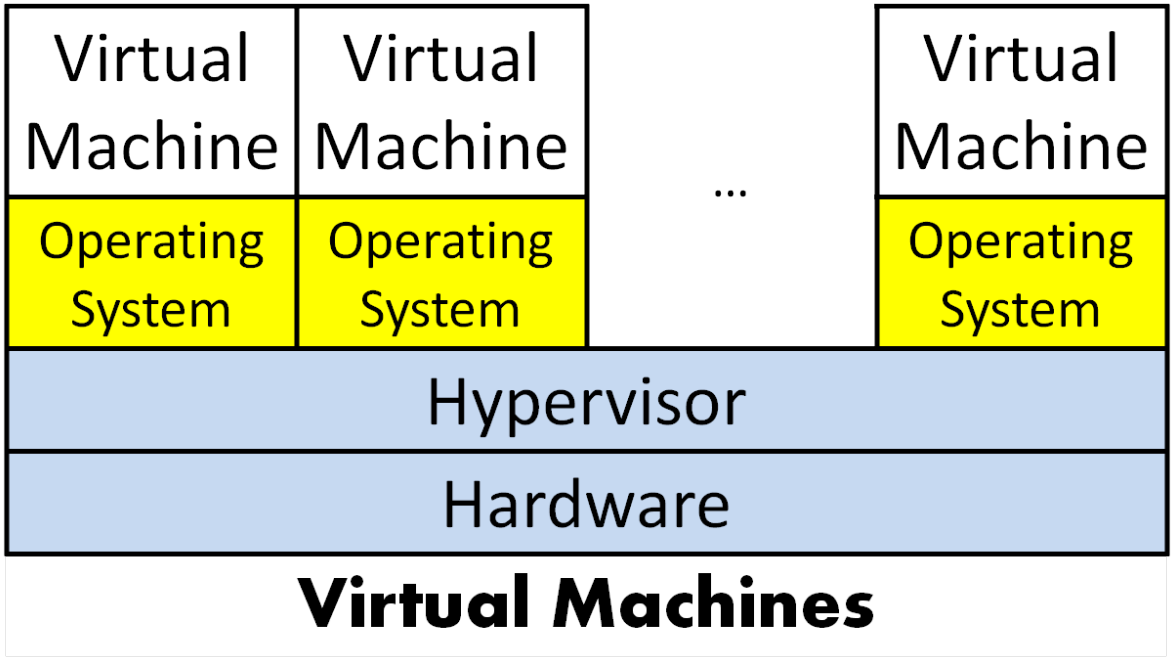
"Todo empezó el día que aprendí Linux"

NABENIK

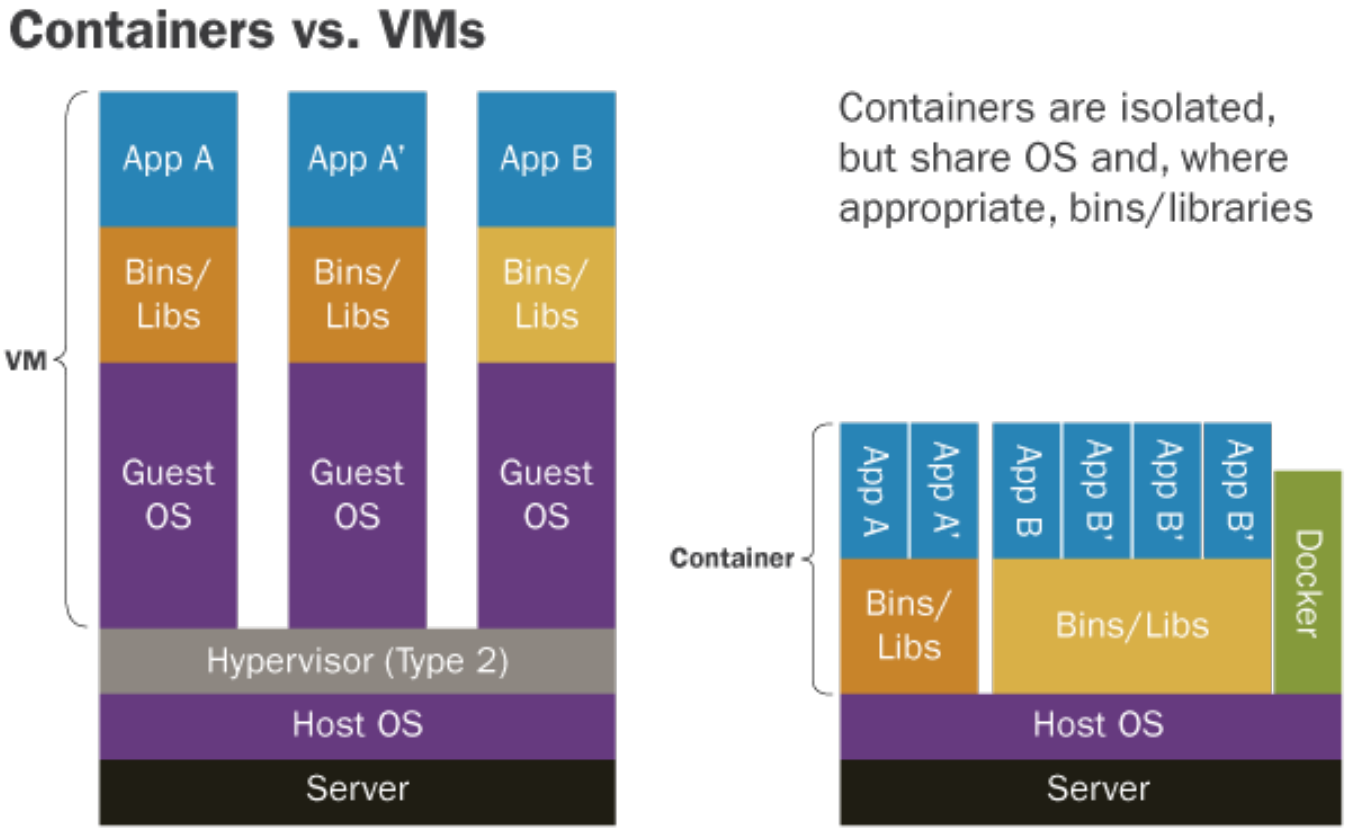
VMs

NABENIK

Hipervisores



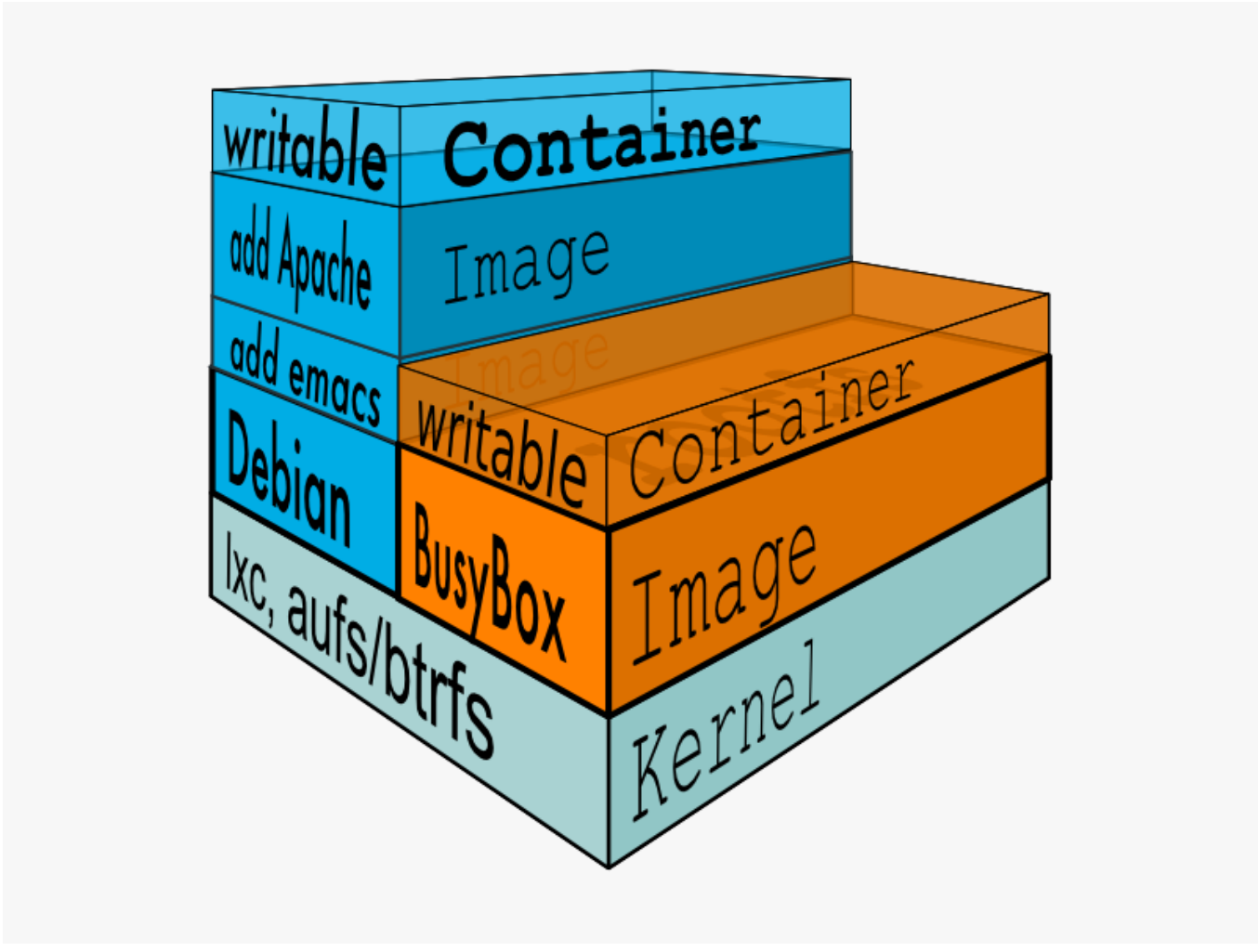
Despliegue contenedores



NABENIK

Contenedor

Contenedor = bibliotecas + app + shell



NABENIK

Contenedor

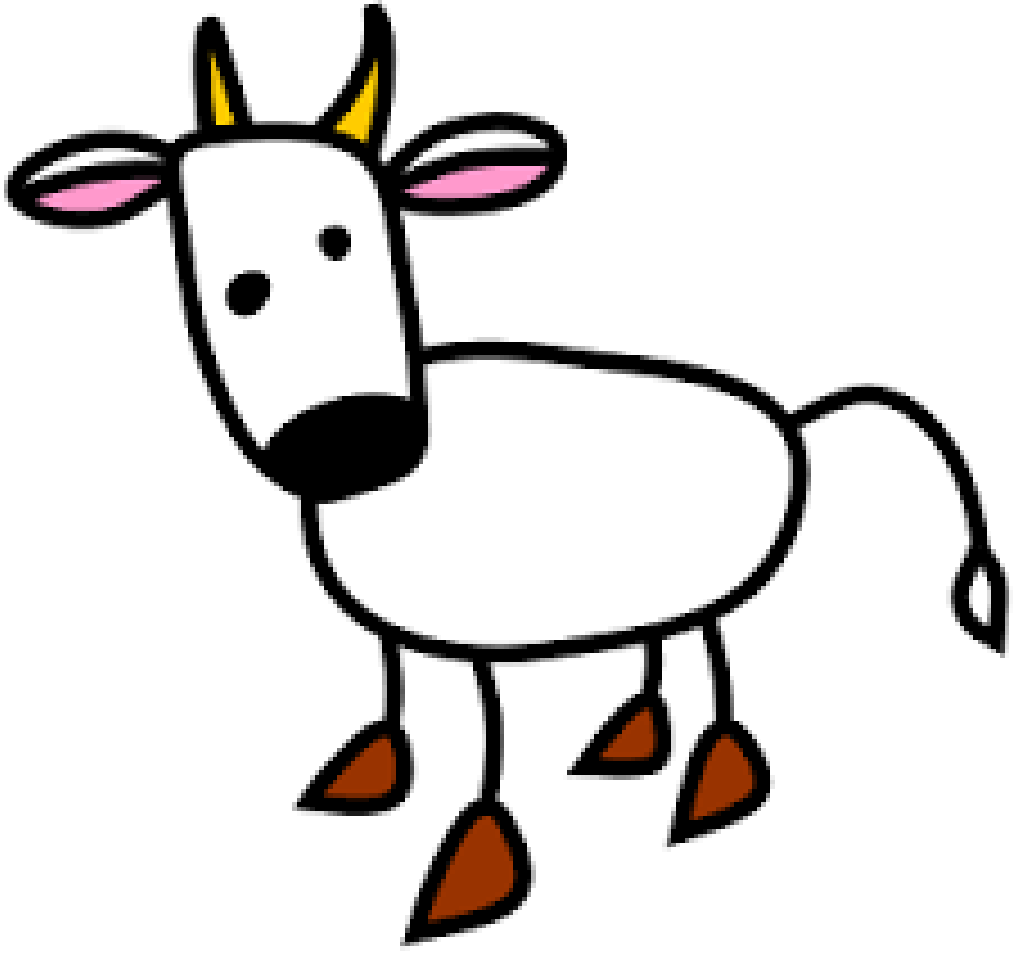
Contenedor = bibliotecas + app + shell

Ideas generales:

- Distribución
- Volatilidad
- Automatización

NABENIK

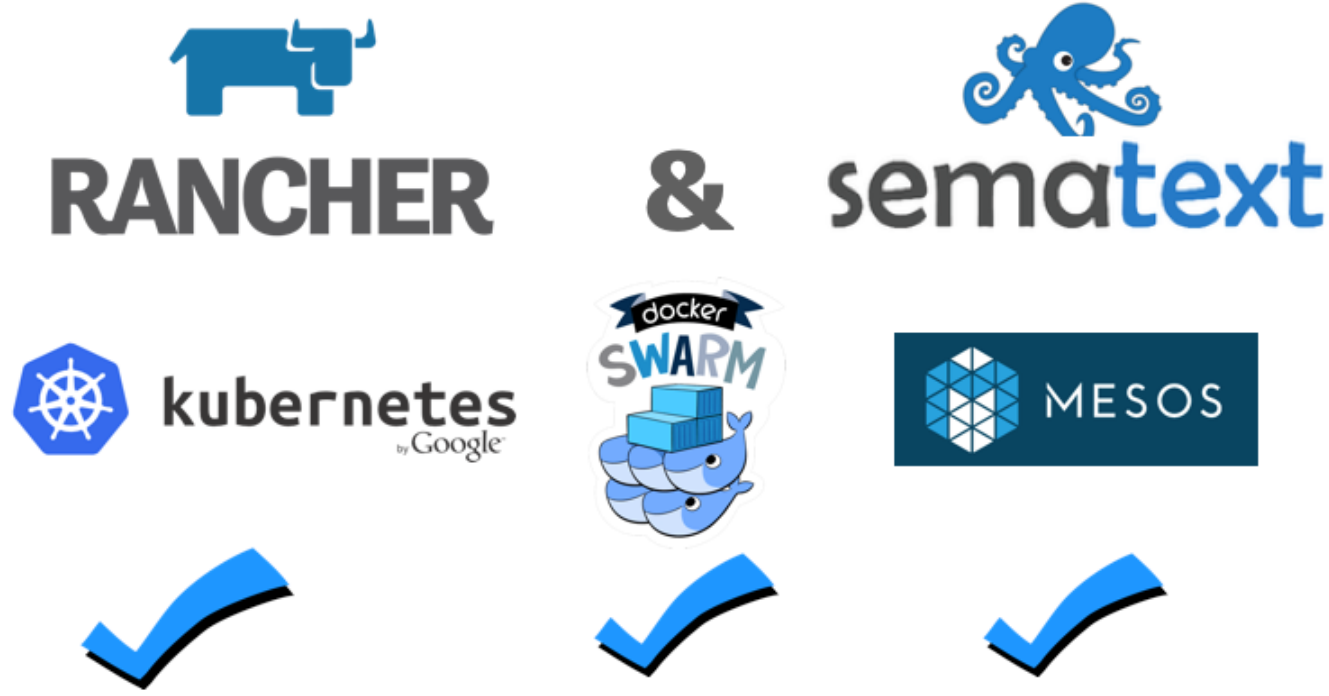
Docker



Swarm/Rancher/Kubernets/Mesos

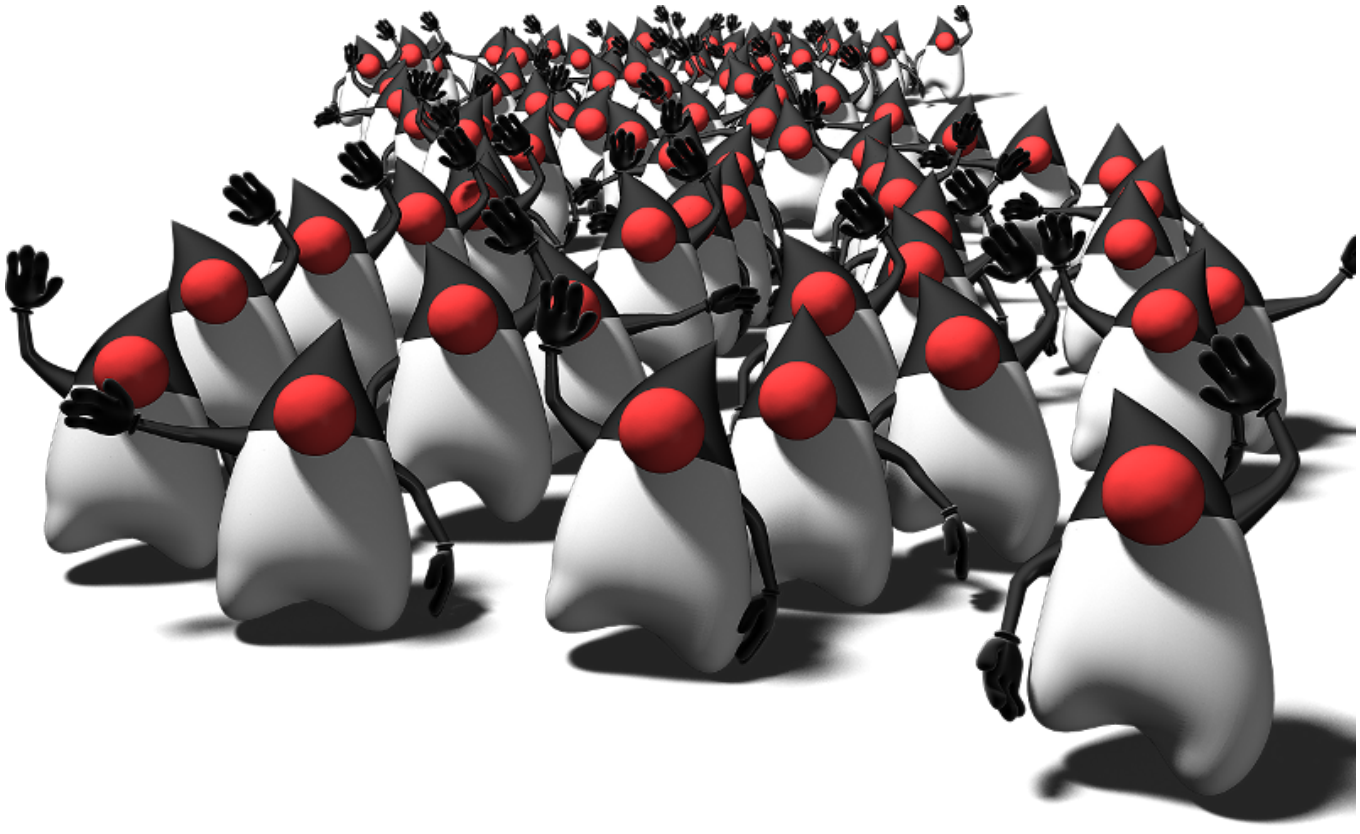
NABENIK

Swarm/Rancher/Kubernets/Mesos



NABENIK

Swarm/Rancher/Kubernets/Mesos + Java



NABENIK

Docker + Java

12 factores cloud native

1. Codebase

12 factores cloud native

1. Codebase (Git)
2. Dependencies

12 factores cloud native

1. Codebase (Git)
2. Dependencies (Maven/Graddle)
3. Config

12 factores cloud native

1. Codebase (Git)
2. Dependencies (Maven/Graddle)
3. Config (Framework/Service Mesh)
4. Backing services

12 factores cloud native

1. Codebase (Git)
2. Dependencies (Maven/Graddle)
3. Config (Framework/Service Mesh)
4. Backing services (RDBMS, NoSQL, JCA)
5. Build, release, run

12 factores cloud native

1. Codebase (Git)
2. Dependencies (Maven/Graddle)
3. Config (Framework/Service Mesh)
4. Backing services (RDBMS, NoSQL, JCA)
5. Build, release, run (CI/CD)
6. Processes

12 factores cloud native

1. Codebase (Git)
2. Dependencies (Maven/Graddle)
3. Config (Framework/Service Mesh)
4. Backing services (RDBMS, NoSQL, JCA)
5. Build, release, run (CI/CD)
6. Processes (Framework stateless)

1. Port binding

12 factores cloud native

1. Codebase (Git)
2. Dependencies (Maven/Graddle)
3. Config (Framework/Service Mesh)
4. Backing services (RDBMS, NoSQL, JCA)
5. Build, release, run (CI/CD)
6. Processes (Framework stateless)
1. Port binding (Service Mesh)
2. Concurrency

12 factores cloud native

1. Codebase (Git)
2. Dependencies (Maven/Graddle)
3. Config (Framework/Service Mesh)
4. Backing services (RDBMS, NoSQL, JCA)
5. Build, release, run (CI/CD)
6. Processes (Framework stateless)
1. Port binding (Service Mesh)
2. Concurrency (Service Mesh)
3. Disposability

12 factores cloud native

1. Codebase (Git)
2. Dependencies (Maven/Graddle)
3. Config (Framework/Service Mesh)
4. Backing services (RDBMS, NoSQL, JCA)
5. Build, release, run (CI/CD)
6. Processes (Framework stateless)

1. Port binding (Service Mesh)
2. Concurrency (Service Mesh)
3. Disposability (Framework/Service Mesh)
4. Dev / Prod parity

12 factores cloud native

1. Codebase (Git)
2. Dependencies (Maven/Graddle)
3. Config (Framework/Service Mesh)
4. Backing services (RDBMS, NoSQL, JCA)
5. Build, release, run (CI/CD)
6. Processes (Framework stateless)

1. Port binding (Service Mesh)
2. Concurrency (Service Mesh)
3. Disposability (Framework/Service Mesh)
4. Dev / Prod parity (Cultura)
5. Logs

12 factores cloud native

1. Codebase (Git)
2. Dependencies (Maven/Graddle)
3. Config (Framework/Service Mesh)
4. Backing services (RDBMS, NoSQL, JCA)
5. Build, release, run (CI/CD)
6. Processes (Framework stateless)

1. Port binding (Service Mesh)
2. Concurrency (Service Mesh)
3. Disposability (Framework/Service Mesh)
4. Dev / Prod parity (Cultura)
5. Logs (Metrics/Grafana, etc.)
6. Admin processes

12 factores cloud native

1. Codebase (Git)
2. Dependencies (Maven/Graddle)
3. Config (Framework/Service Mesh)
4. Backing services (RDBMS, NoSQL, JCA)
5. Build, release, run (CI/CD)
6. Processes (Framework stateless)

1. Port binding (Service Mesh)
2. Concurrency (Service Mesh)
3. Disposability (Framework/Service Mesh)
4. Dev / Prod parity (Cultura)
5. Logs (Metrics/Grafana, etc.)
6. Admin processes (Service Mesh)

Monolitos + Docker

Ventajas

- Migraciones rápidas
- SCM más ordenado
- Reconstrucción de monolito más rapida

Desventajas

- Tooling overhead
- Debug
- Más uso de memoria
- Complejidad en red

Ventajas

- Bases de código pequeñas
- Buenas practicas de programación
- Tolerancia a fallas
- Escalabilidad

Desventajas

- Tooling overhead
- Debug
- Transacciones distribuidas
- Latencia
- Interdependencia
- Falacias de la computación distribuida

NABENIK

Demo

Monolito

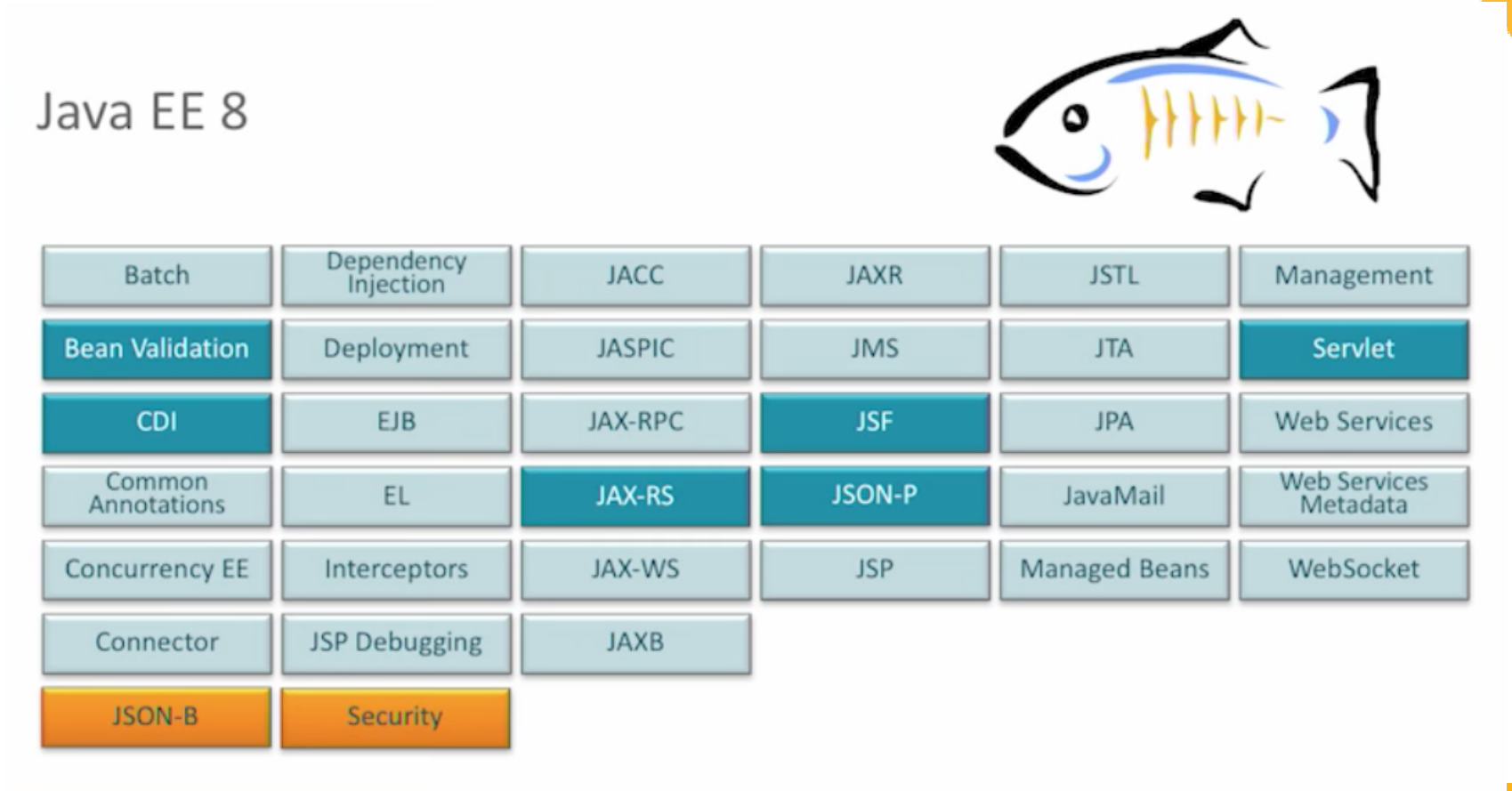
- WildFly (JBoss)
- Java EE 7
- Linode + Docker Hub

Microservicio

- Payara Micro
- Java EE 8 + MicroProfile
- Oracle Cloud

Java EE 8

Java EE 8



NABENIK

Java EE 8

- Mejor integración JSF-CDI
- Mejor integración JMS-CDI
- HTTP/2
- JSON-B
- Security
- **JAX-RS Reactivo**

Java EE 8 - Comunidad

JAKARTA EE

JAVA EE GUARDIANS

MICROPROFILE™

OPTIMIZING ENTERPRISE JAVA

NABENIK

Eclipse MicroProfile

Eclipse MicroProfile

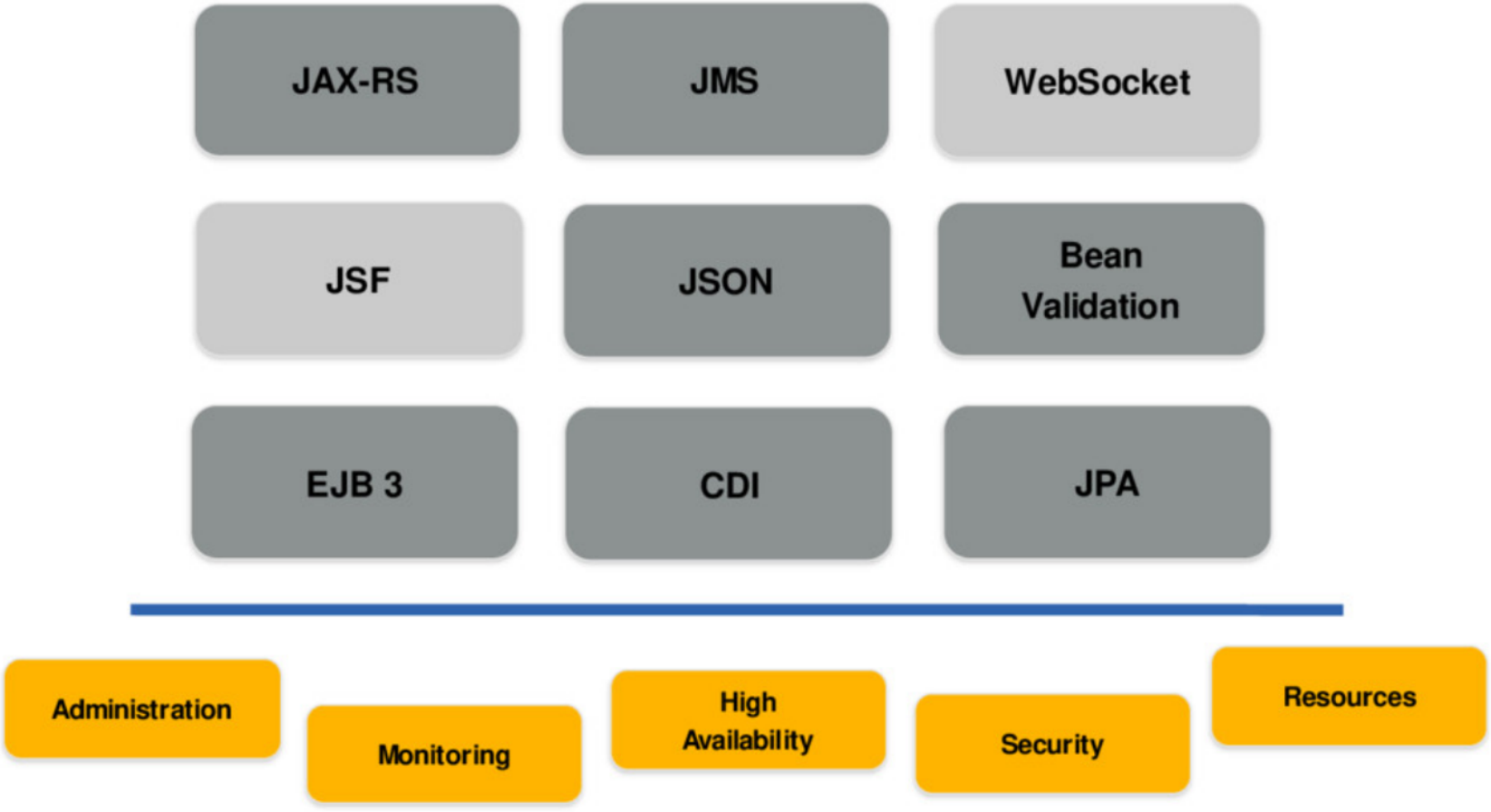
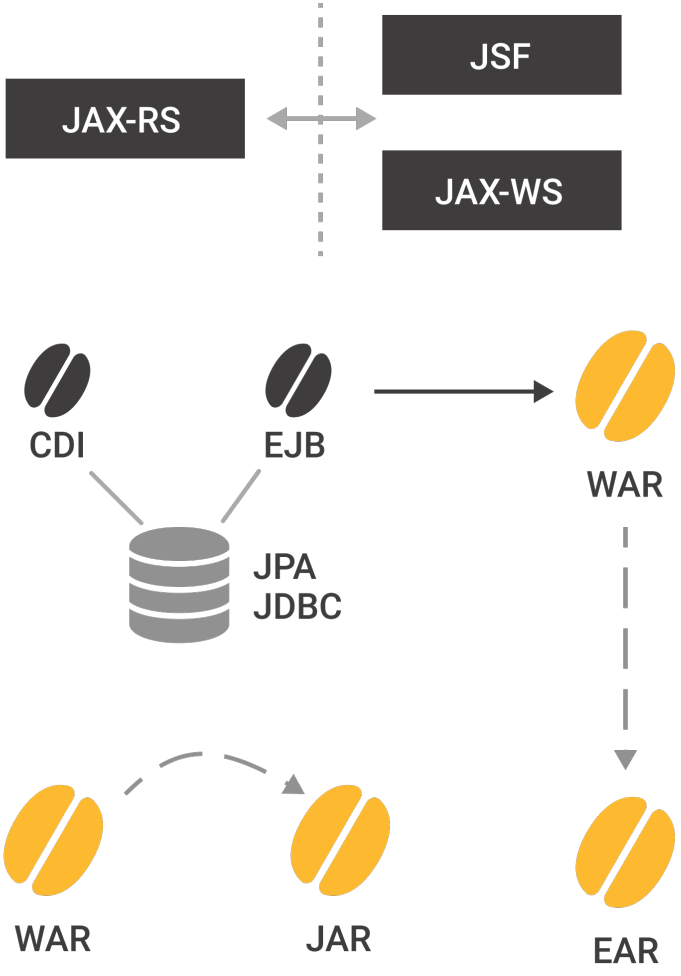


Figura 4: Credito: Reza Rahman

NABENIK

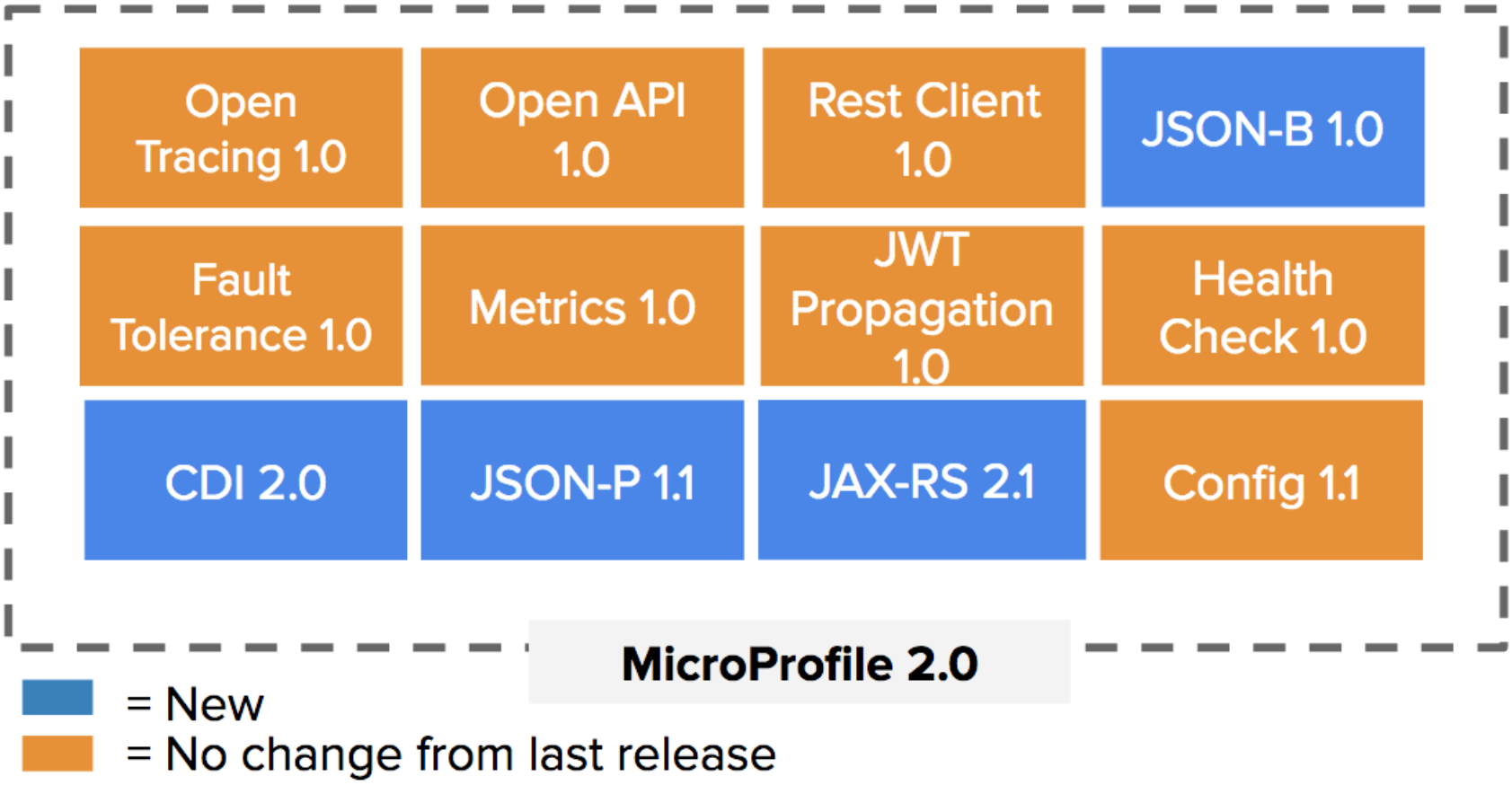
Eclipse MicroProfile



BV + JSON-P JSON-B + Security

NABENIK

Eclipse MicroProfile



NABENIK

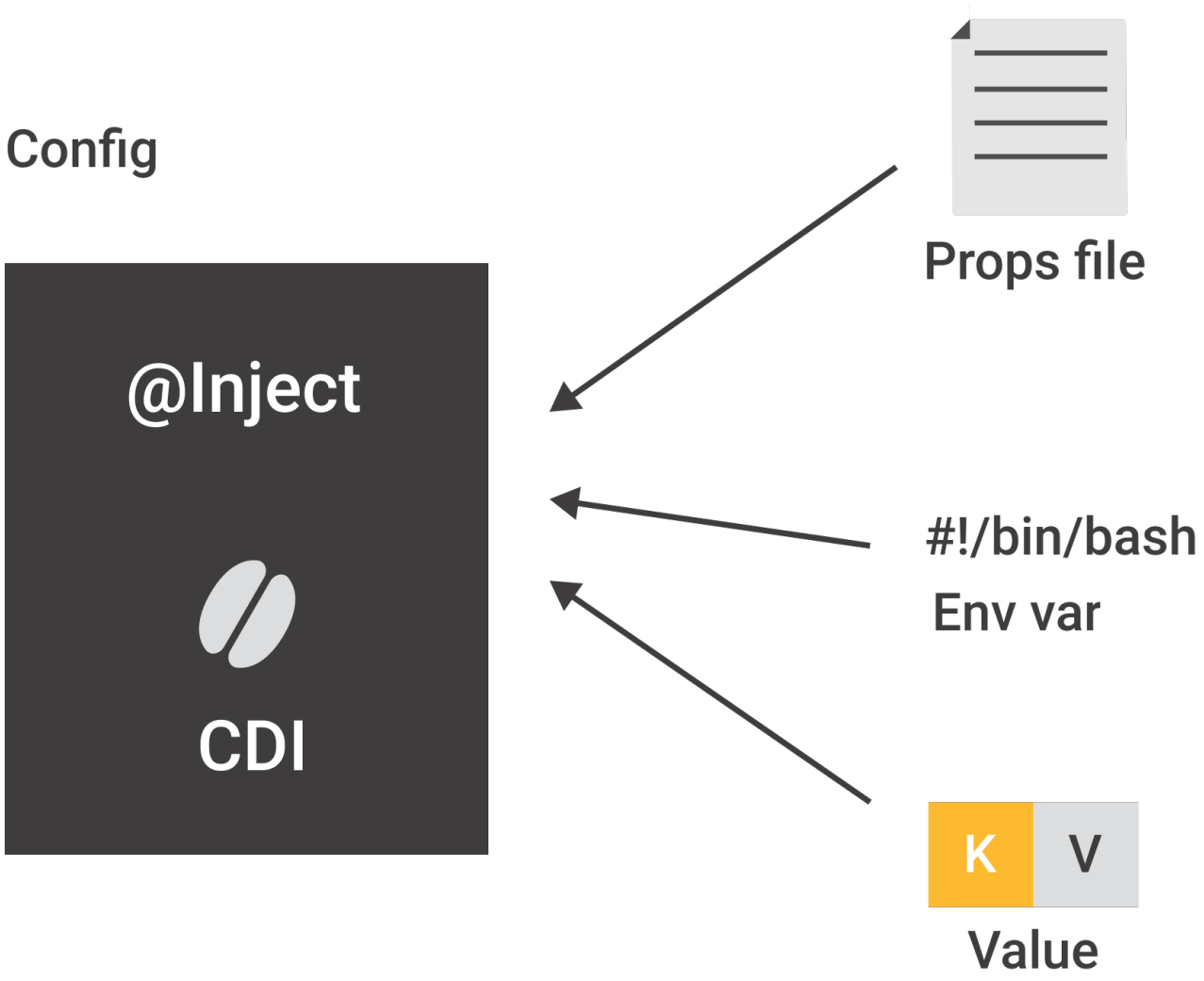
Eclipse MicroProfile - Implementações

- Thorntail (Red Hat)
- KumuluzEE
- Open Liberty (IBM)
- TomEE
- Helidon (Oracle)
- Hammock
- Apache Meecrowave
- **Payara Micro**

Eclipse MicroProfile en Payara 5

```
<dependency>
        <groupId>org.eclipse.microprofile</groupId>
        <artifactId>microprofile</artifactId>
        <type>pom</type>
        <version>2.0.1</version>
        <scope>provided</scope>
</dependency>
```

Config



NABENIK

Config

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

Externalización de la configuración (VM, Docker, Kubernetes)

Fault Tolerance

~~a ()~~ X

aFallback()

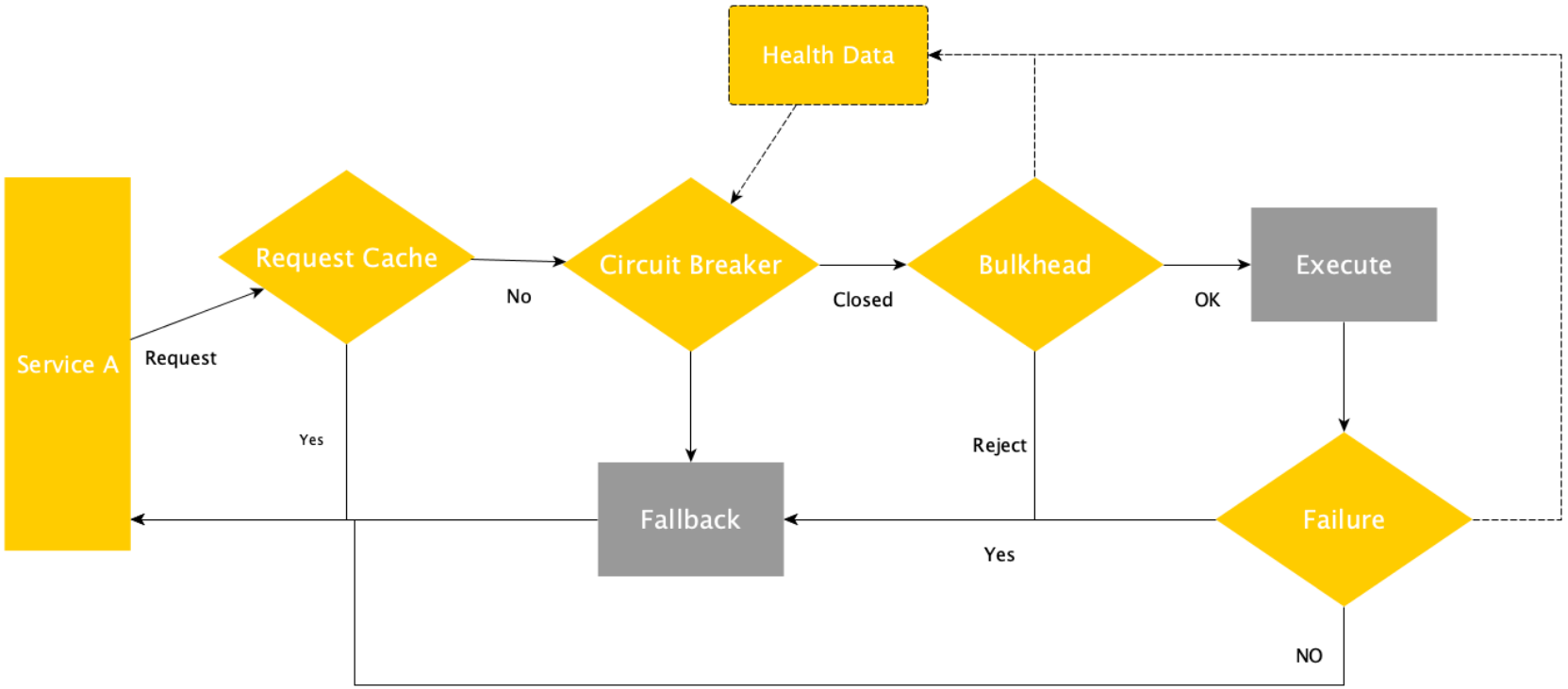
NABENIK

Metrics

NABENIK

Fault Tolerance + Metrics

- *Fault Tolerance* depende de metricas que pueden ser utilizadas para diagnosticar el estado de nuestro sistema mediante *Metrics*



Fault tolerance

Reglas de evaluación y alternativas

- Circuit Breaker
- Bulkhead
- Retry
- Timeout
- Fallback

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) {
...
}
```

NABENIK

Métricas

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

¿Cuales?

- 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 en tiempo real

```
@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

Desempeño y delay

```
@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
```

NABENIK

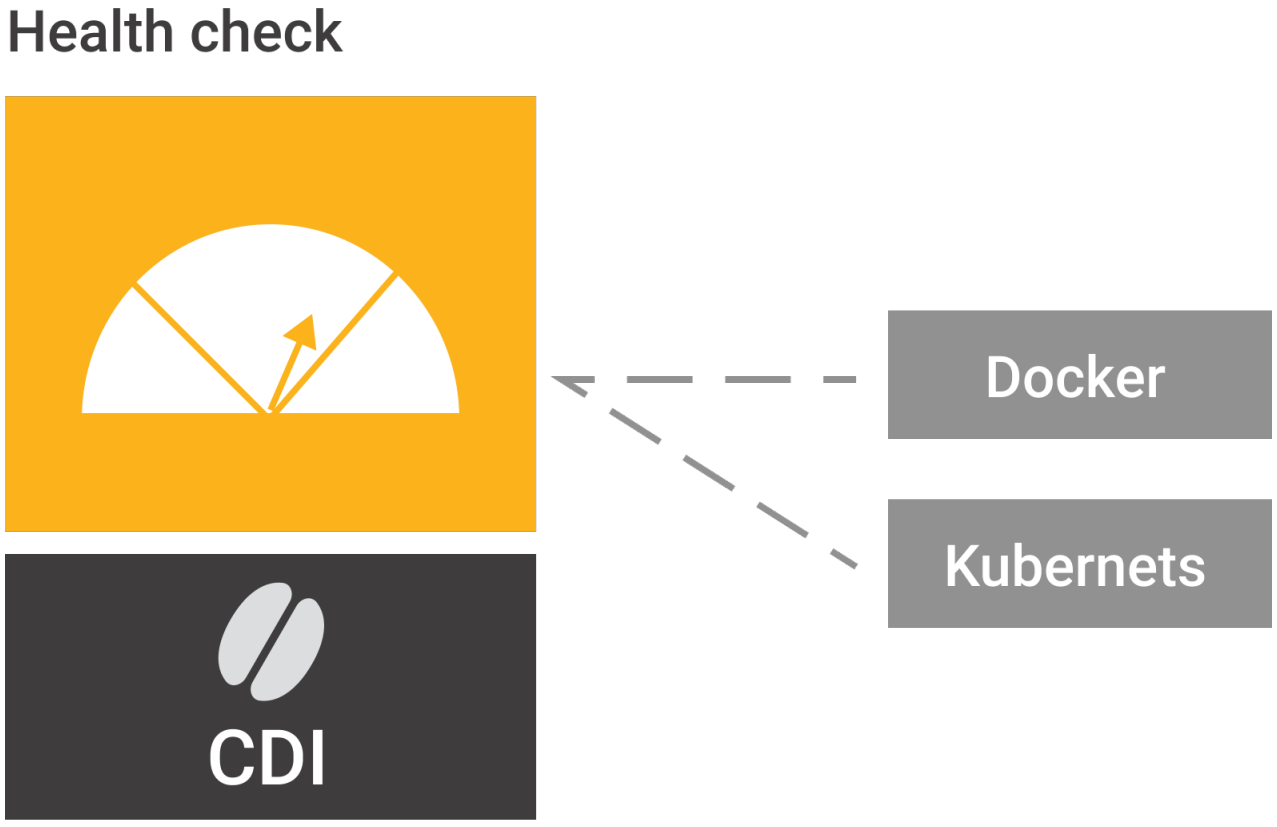
Metrics - Histogram

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



- OK?
- How much ok?

NABENIK

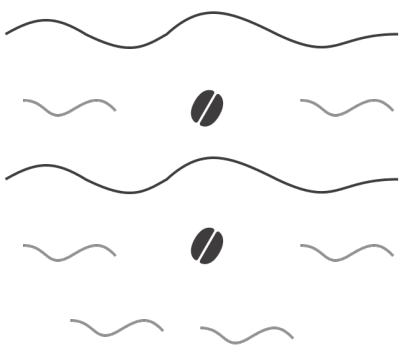
Health Check

¿Aun vivo?

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

}
```

JWT



@Inject
Principal

@Inject
Realm

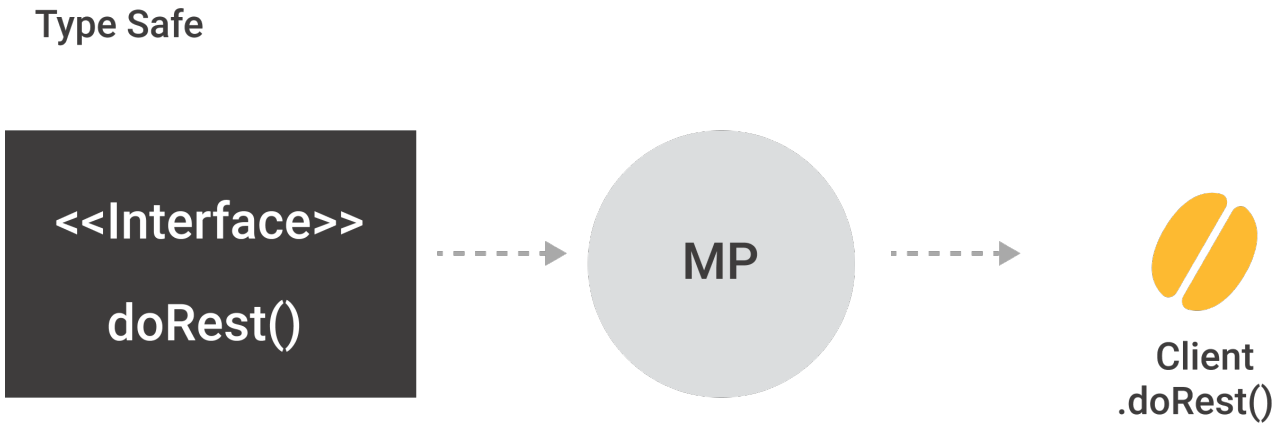
NABENIK

```
@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;
}
```

Demo

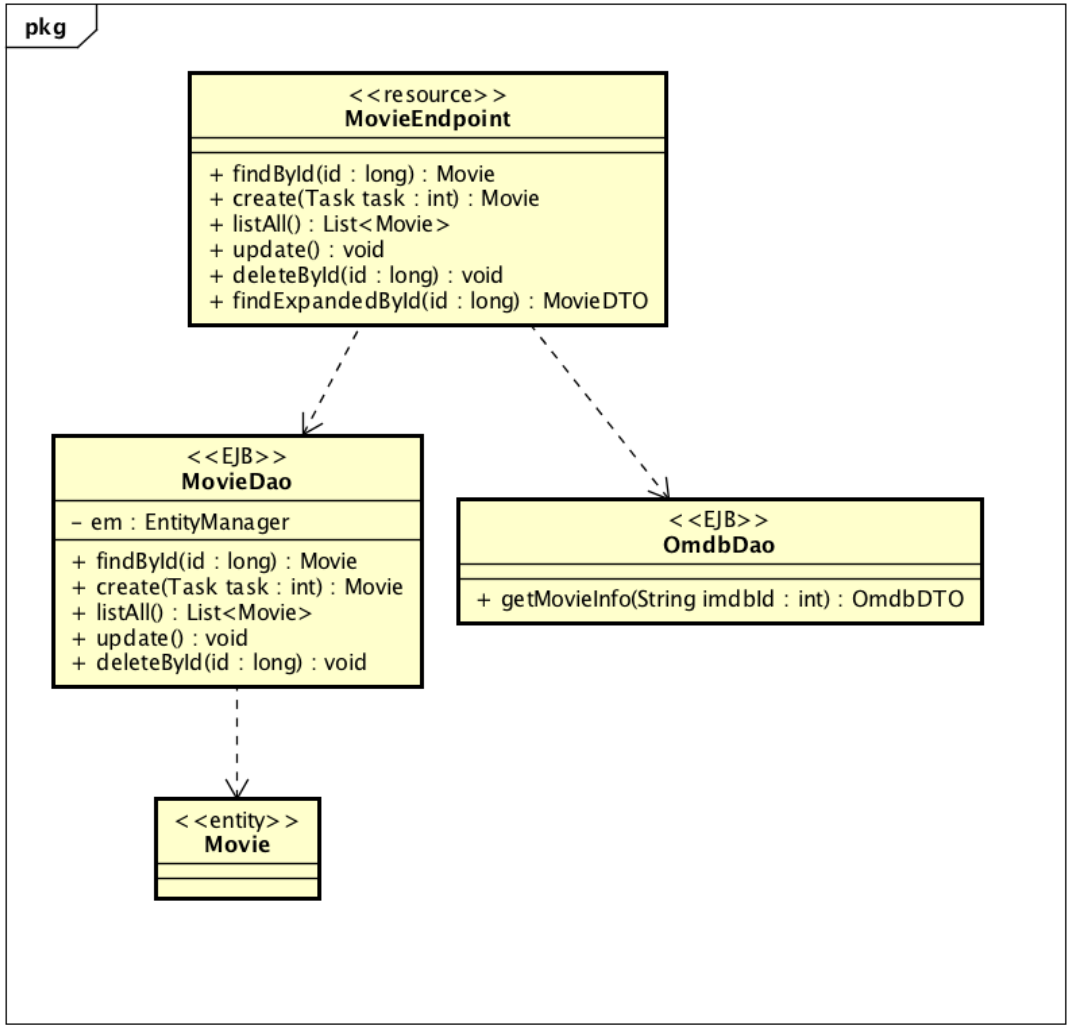
Java 8, JAX-RS, CDI, EJB, MicroProfile

`https://github.com/tuxtor/payara-demo`

`https://github.com/tuxtor/omdb-demo`

Stacks tradicionais

- EJB
- **JTA**
- JAX-RS
- CDI

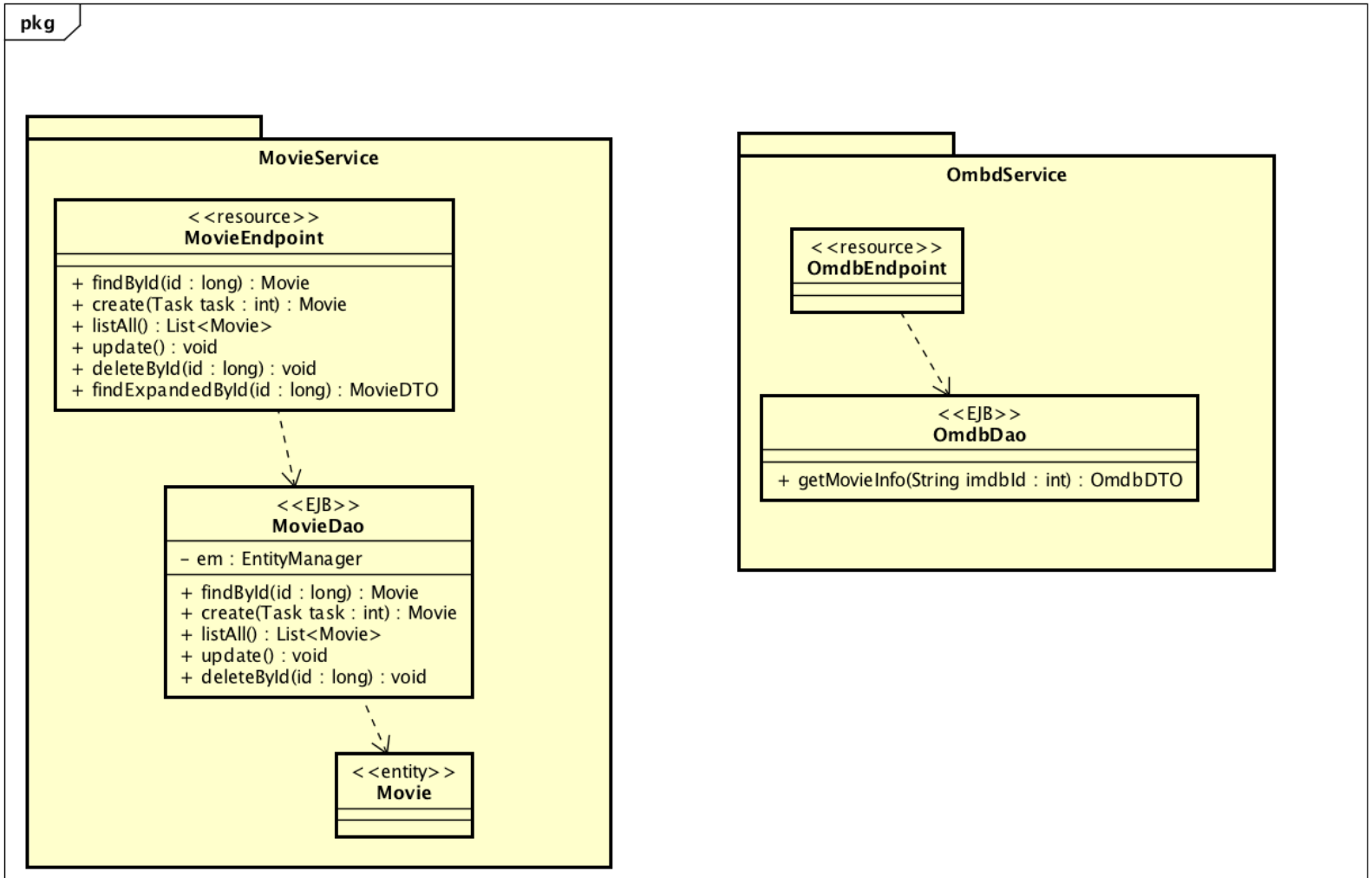


EE + MicroProfile - Demo

MicroProfile: JAX-RS, CDI, Config, Fault Tolerance, Metrics

Payara Micro: EJB, JTA

Fatores externos: Location, Deployment, Orchestation, Balancing, Consistency



BENIK

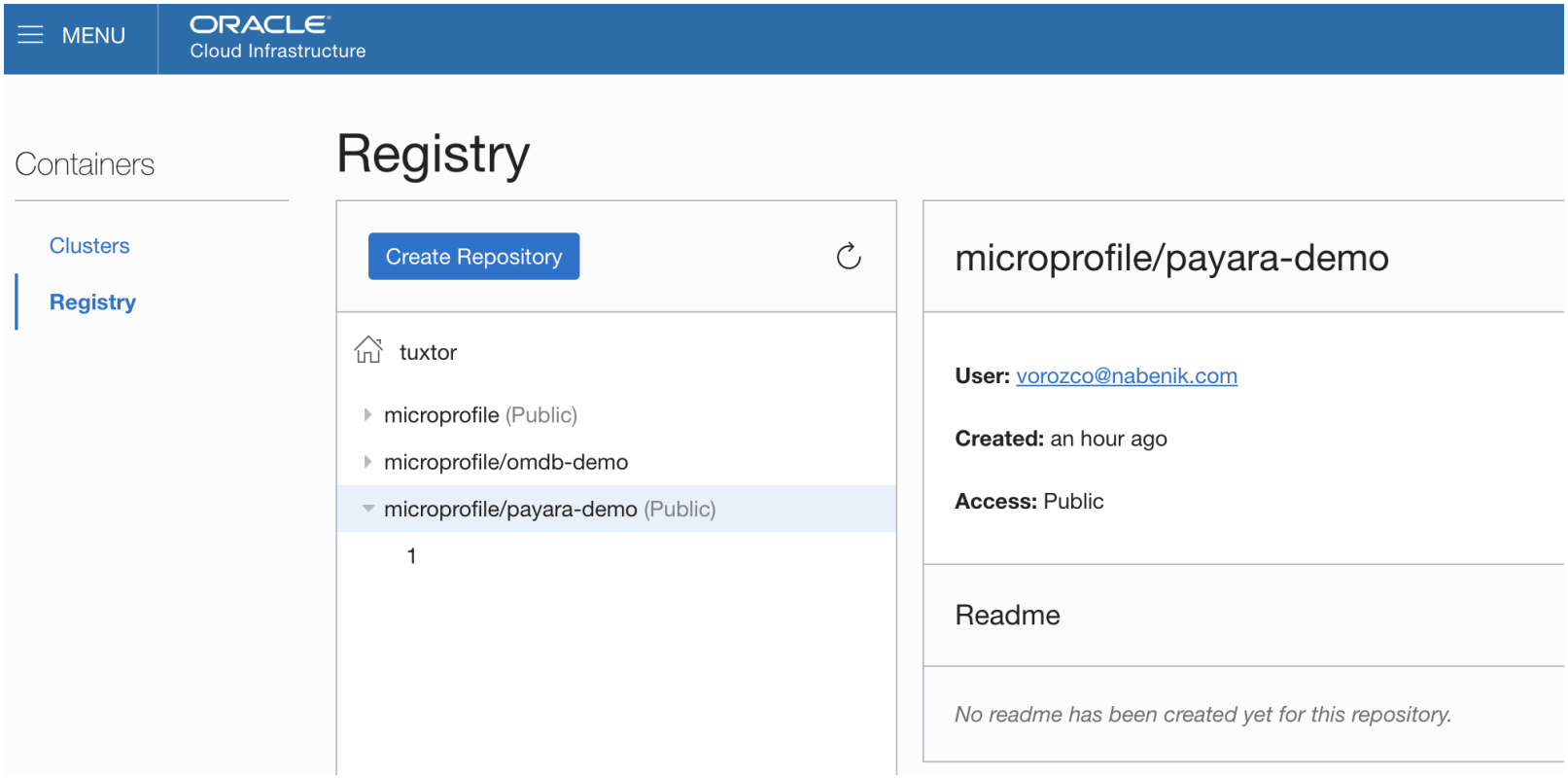
12 fatores

Microprofile

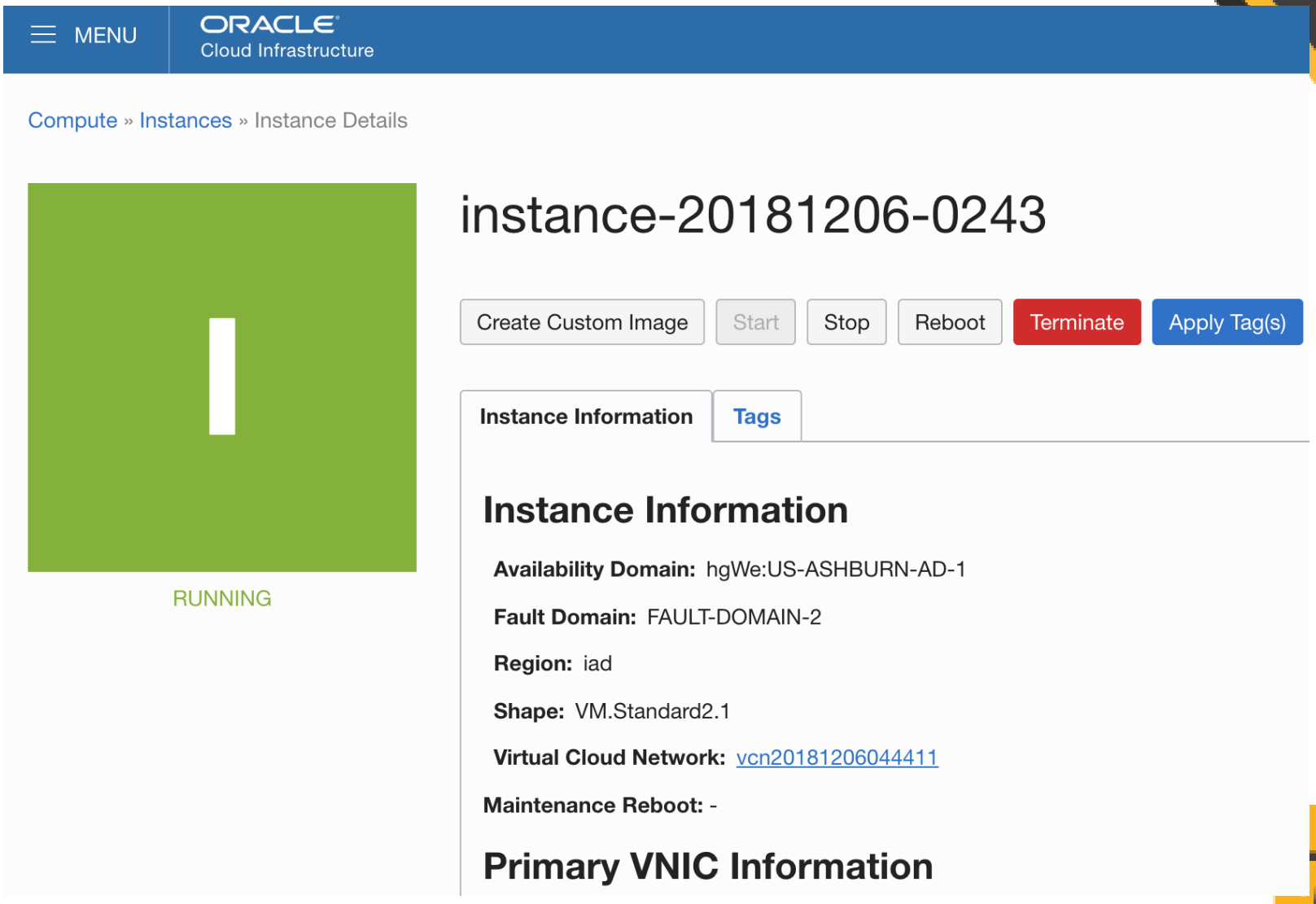
- Config
- Backing service
- Disposability

Cloud

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


MENU
ORACLE
Cloud Infrastructure
Containers
Clusters
Registry
Registry
Create Repository
tuxtor
microprofile (Public)
microprofile/omdb-demo
microprofile/payara-demo (Public)
1
microprofile/payara-demo
User: vorozco@nabenik.com
Created: an hour ago
Access: Public
Readme
No readme has been created yet for this repository.

NABENIK


MENU
ORACLE
Cloud Infrastructure
Compute » Instances » Instance Details
instance-20181206-0243
Create Custom Image
Start
Stop
Reboot
Terminate
Apply Tag(s)
Instance Information
Tags
RUNNING
Instance Information
Availability Domain: hgWe:US-ASHBURN-AD-1
Fault Domain: FAULT-DOMAIN-2
Region: iad
Shape: VM.Standard2.1
Virtual Cloud Network: vcn20181206044411
Maintenance Reboot: -
Primary VNIC Information

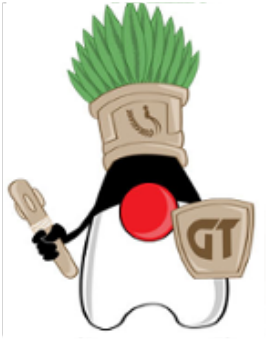
NABENIK

```
tuxtor@millenium-falcon-2:~$ docker tag omdb-demo iad.ocir.io/tuxtor/microprofile/omdb-demo:latest
```

```
tuxtor@millenium-falcon-2:~$ docker push iad.ocir.io/tuxtor/microprofile/omdb-demo
The push refers to repository [iad.ocir.io/tuxtor/microprofile/omdb-demo]
31d661ce120c: Pushing [==================================>                ]  952.3kB/1.46MB
5dd1fd455c13: Layer already exists
2aa3decaee68: Layer already exists
685fdd7e6770: Layer already exists
c9b26f41504c: Layer already exists
cd7100a72410: Layer already exists
```

Stateful Rules				
Source: 0.0.0.0/0	**IP Protocol:** TCP	**Source Port Range:** All	**Destination Port Range:** 22	**Allows:** TCP traffic for ports: 22 SSH Remote Login Protocol
Source: 0.0.0.0/0	**IP Protocol:** ICMP	**Type and Code:** 3, 4		**Allows:** ICMP traffic for: 3, 4 Destination Unreachable: Fragmentation Needed and Don't Fragment was Set
Source: 10.0.0.0/16	**IP Protocol:** ICMP	**Type and Code:** 3		**Allows:** ICMP traffic for: 3 Destination Unreachable
Source: 0.0.0.0/0	**IP Protocol:** TCP	**Source Port Range:** All	**Destination Port Range:** 8080-8085	**Allows:** TCP traffic for ports: 8080-8085

Víctor Orozco



Oracle Groundbreakers

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

NABENIK