Patrones de Cloud Native con MicroProfile

Víctor Orozco

11 de septiembre de 2021

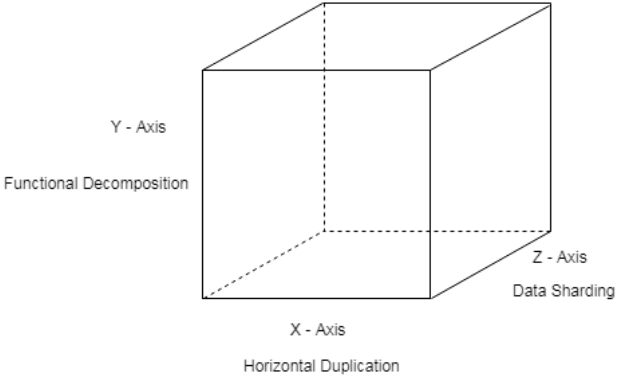
Nabenik

ACADEMIK

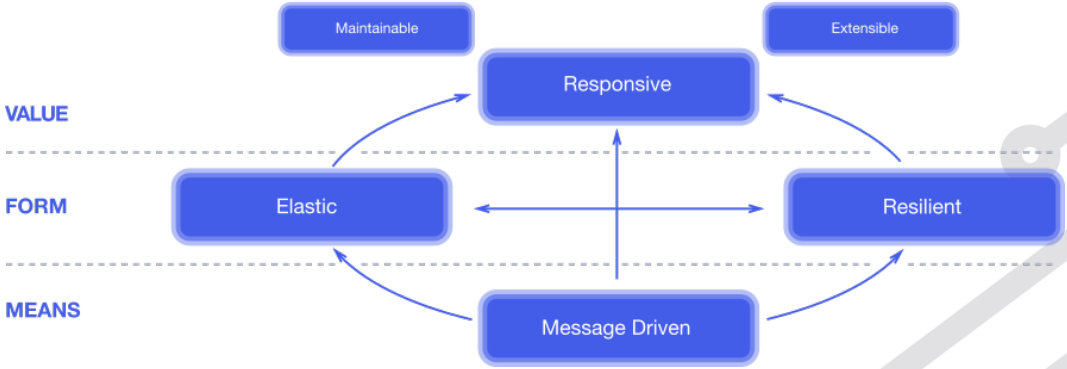
Patrones de microservicios

Microservicios = Metapatron arquitectural

Arquitectura que estructura una aplicación como un conjunto de servicios débilmente acoplados. Corresponde al eje Y del scale cube cuyo objetivo final son sistemas reactivos



@tuxtor



@tuxtor

Application Server

- Transacionalidad distribuida (JTA/XA)
- Contratos (JNDI)
- Service discovery (JNDI)
- Deployment (EAR/Class Loaders/Dashboards)
- Métricas (JMX)
- Seguridad (SoteriaRI/JACC)

Microservicios

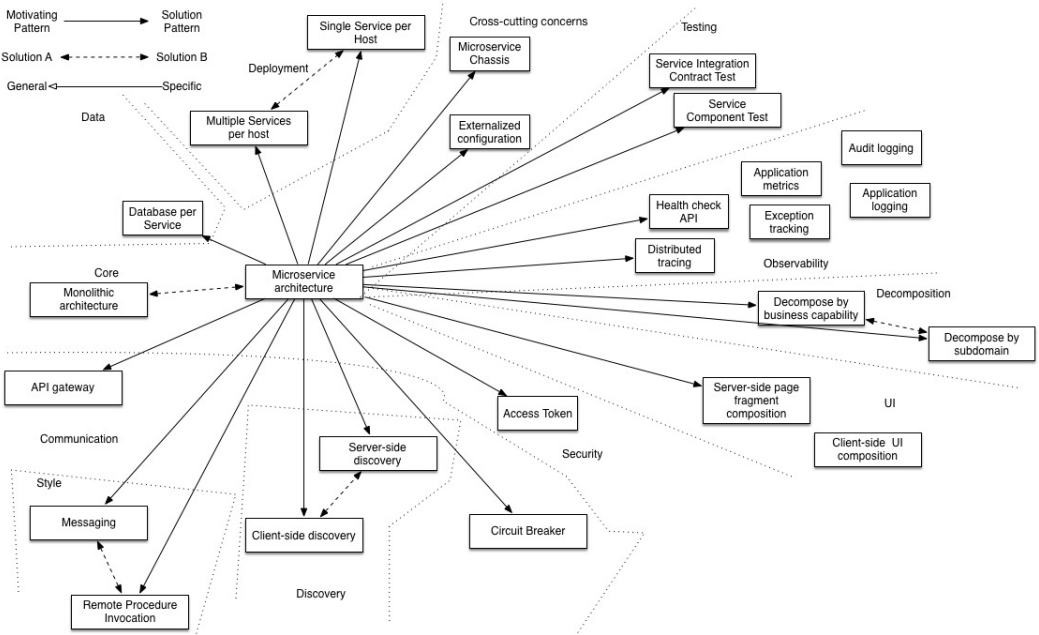
Aplicaciones Cloud Native

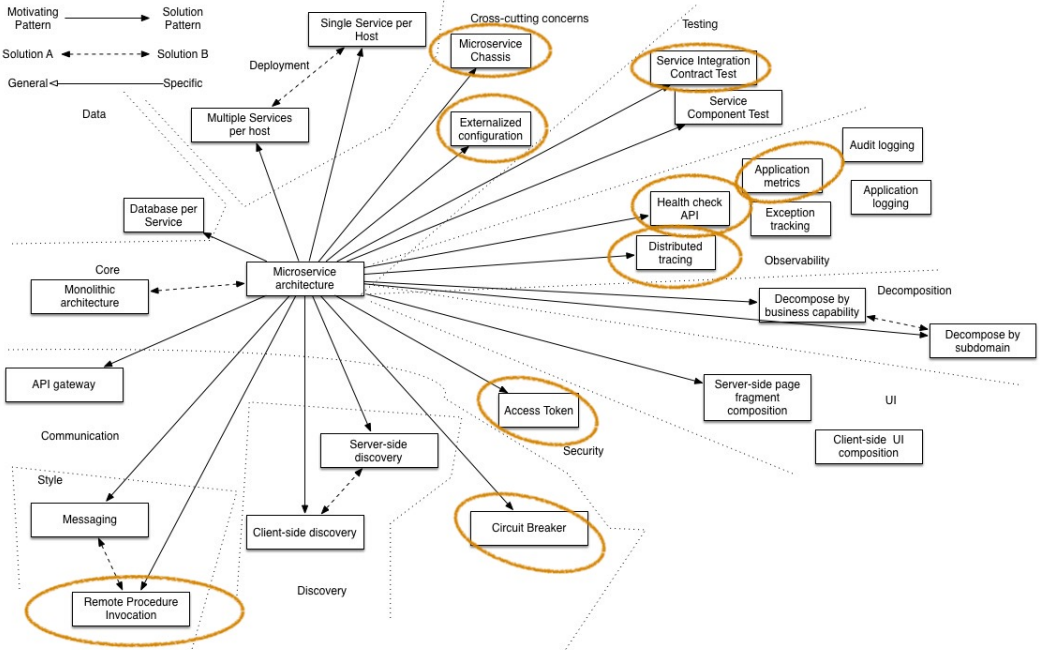
- Sistemas reactivos
- 12 factores Cloud Native
- Design patterns
- Domain Driven Design
- Microservice chassis y service mesh
- Orquestación de contenedores

Microservicios = Metapatron arquitectural

Cloud Native

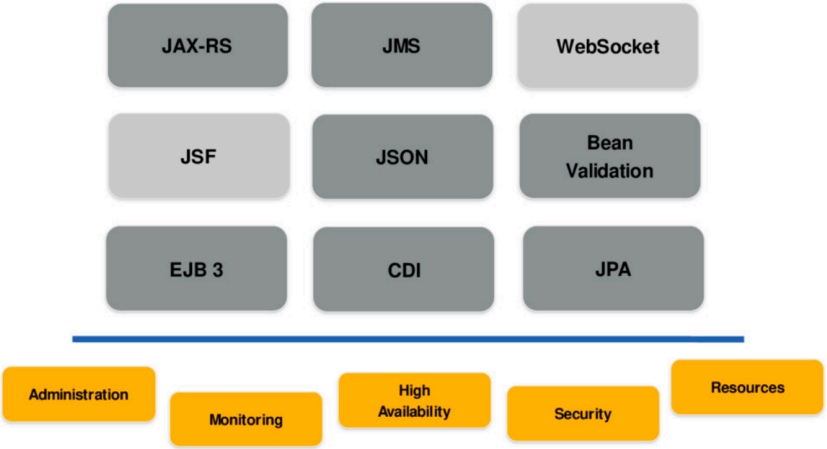
- (Nos gustan) Sistemas reactivos
- (Es posible mediante los) 12 factores Cloud Native
- (Usamos soluciones probada con) design patterns
- (Fragmentamos le sistema mediante) Domain Driven Design
- **(Implementamos los servicios com frameworks) microservice chassis y service mesh**
- (Hacemos despliegue) mediante orquestación de contenedores


Motivating Pattern
Solution Pattern
Solution A
Solution B
General
Specific
Deployment
Data
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


Motivating Pattern
Solution Pattern
Solution A
Solution B
General
Specific
Deployment
Single Service per Host
Multiple Services per host
Data
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

Eclipse MicroProfile

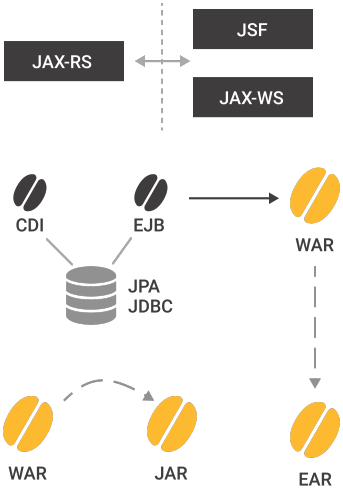
Eclipse MicroProfile



@tuxtor

Figura 1: Credito: Reza Rahman

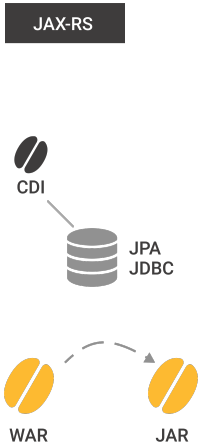
Eclipse MicroProfile



@tuxtor

JSON-P

Eclipse MicroProfile

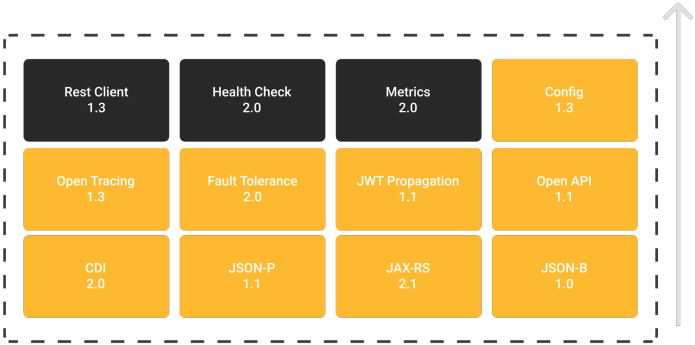


@tuxtor

JSON-P

Eclipse MicroProfile

APIS OF MICROPROFILE 3.0 APIS



MicroProfile3.0

= Uptaded

= No change from last release (MicroProfile 2.2)

@tuxtor

Eclipse MicroProfile - Implementaciones

Bibliotecas

- SmallRye (Red Hat)
- Hammock
- Apache Geronimo
- Fujitsu Launcher

JEAS - Fat Jar

- Dropwizard
- KumuluzEE
- Helidon (Oracle)
- Open Liberty (IBM)
- Quarkus (Red Hat)

Eclipse MicroProfile - Implementaciones

Micro server

- Payara Micro
- TomEE MicroProfile

Full server

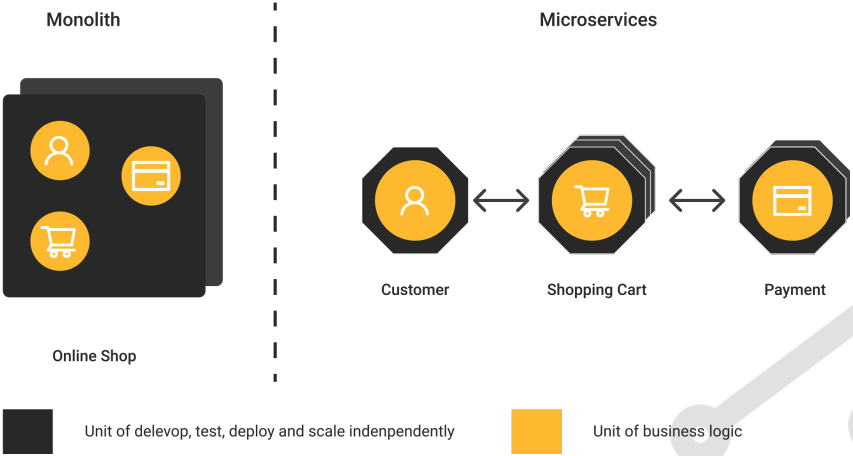
- Payara Application Server
- Apache TomEE
- JBoss Application Server / Wildfly Application Server
- WebSphere Liberty (IBM)

https://wiki.eclipse.org/MicroProfile/Implementation

Eclipse MicroProfile - Objetivos

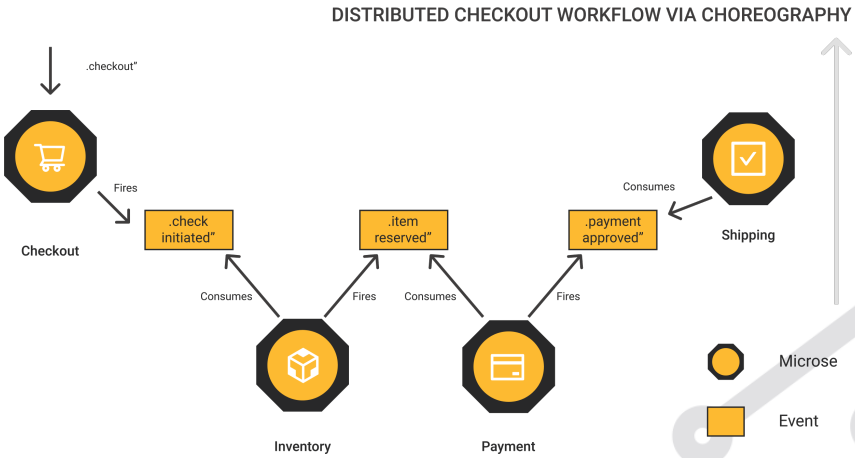
Demo MicroProfile Helidon

Eclipse MicroProfile



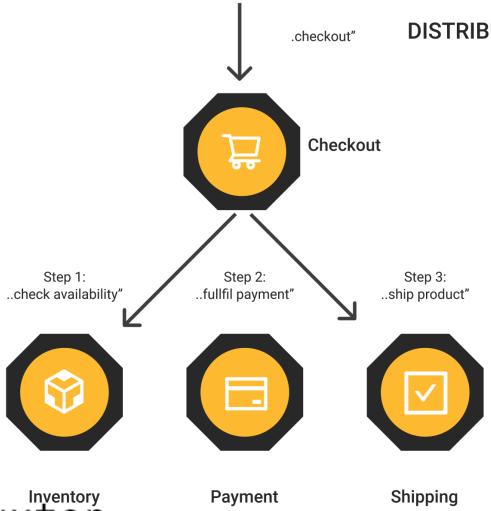
@tuxtor

Eclipse MicroProfile - Coreografia



@tuxtor

Eclipse MicroProfile - Orquestación



DISTRIBUTED CHECKOUT WORKFLOW VIA ORCHESTRATION

Microservice

Command

@tuxtor

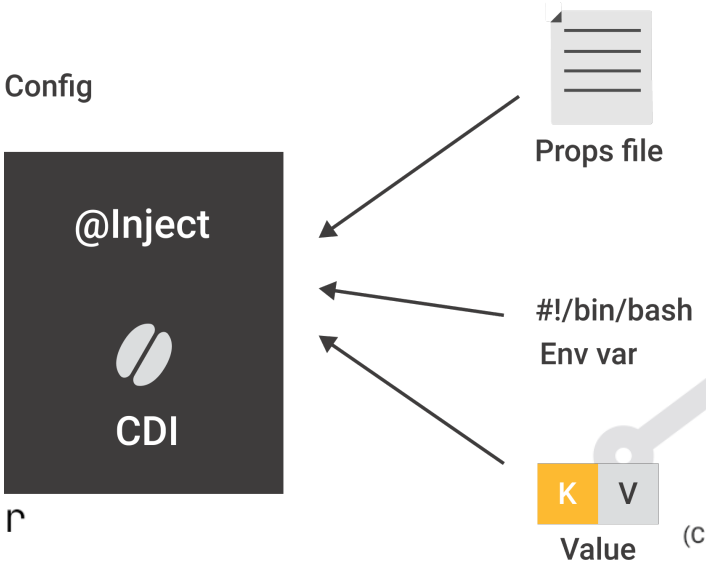
Eclipse MicroProfile - Cross-cutting concerns

- Health checks & Metrics
- Resilence & Fault Tolerance
- Configuration
- Authentication & Authorization
- Standarized documentation
- Tracing

@tuxtor

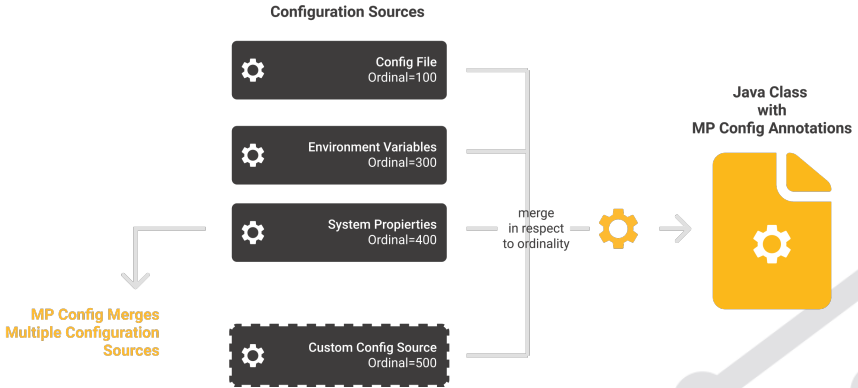
Eclipse MicroProfile - APIs

Config



@tuxtor

Config



@tuxtor

Config

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

Ext. de la configuración (VM, Docker, Kubernetes)

Config

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

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

Config

No CDI, no hay problema

```
final Config config = ConfigProvider.getConfig();
config.getValue("application.curreny", String.class);
config.getOptionalValue("application.list.maxSize",
^^IInteger.class);
```

@tuxtor

Config

Propiedades dinámicas

```
@Inject @ConfigProperty(name="userId")
Provider<String> userId;
```

Config

Inyección global

```
@Inject Config config;
```

OpenAPI - Aplicación

APIs documentadas en formato estandard

- @APIResponses - Respuestas multiples de una API
- @APIResponse - Respuesta unica de una API
- @Content - Esquema y ejemplo
- @Schema - I/O data types
- @Operation - Describe la operación
- @Parameter - Describe el parámetro de una operación

OpenAPI - Aplicación

APIs documentadas en formato estandard

```
@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 = {
    ^^I@Server(url = "/example",
    ^^Idescription = "localhost")
    }
)
public class ApplicationConfig extends Application {
```

@tuxtor

OpenAPI

APIs documentadas en formato estandard

```
@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

localhost:8080/openapi/

localhost:8080/

```
openapi: 3.0.0
info:
  title: Deployed Resources
  version: 1.0.0
servers:
- url: http://localhost:8080/micro-sample-1.0-SNAPSHOT
  description: Default Server.
paths:
  /data/hello:
    get:
      description: Un metodo de hola mundo
      operationId: hello world
      responses:
        default:
          content:
            '*/*':
              schema:
                type: string
          description: Default Response.
components: {}
```

OpenAPI - Swagger UI

APIs documentadas en formato estandard

```
<dependency>
    <groupId>org.microprofile-ext.openapi-ext</groupId>
    <artifactId>swagger-ui</artifactId>
    <version>1.0.2</version>
</dependency>
```

@tuxtor

Fault Tolerance

Fault Tolerance

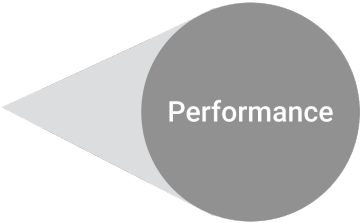
~~a ()~~ X

aFallback()

@tuxtor

Metrics

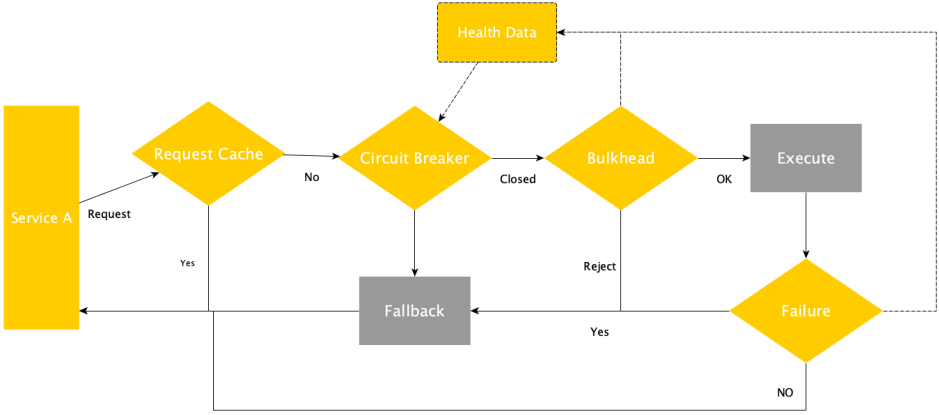
Metrics



@tuxtor

Fault Tolerance + Metrics

- *Fault Tolerance* depende de la existencia de metricas, las metricas se exponen mediante *Metrics*



Fault tolerance

Reglas de evaluación y 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() {
    ...
}
```

@tuxtor

Fault tolerance - CircuitBreaker

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

@tuxtor

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);
}
```

@tuxtor

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) {
...
}
```

@tuxtor

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) {
    ^^Ireturn Stream.of("Star Wars",
    ^^I"The Matrix", "Cantinflas").collect(toList());
    }
}
```

@tuxtor

Métricas

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

¿Cuales?

- Counted
- Gauge
- Metered
- Timed
- Histogram

@tuxtor

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

@tuxtor

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 retraso

```
@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

@tuxtor

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

Health check

Docker

Kubernets

Health Check

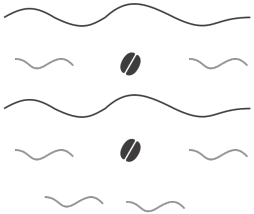
¿Estas vivo?

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

}
```

@tuxtor

JWT

JWT



@Inject
Principal

@Inject
Realm

@tuxtor

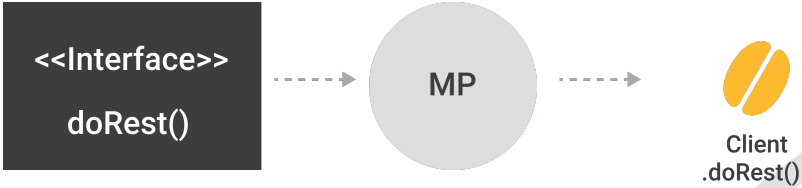
JWT

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

@Inject
private JsonWebToken jwtPrincipal;

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

@tuxtor

TypeSafe

Type Safe



@tuxtor

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;
}
```

@tuxtor

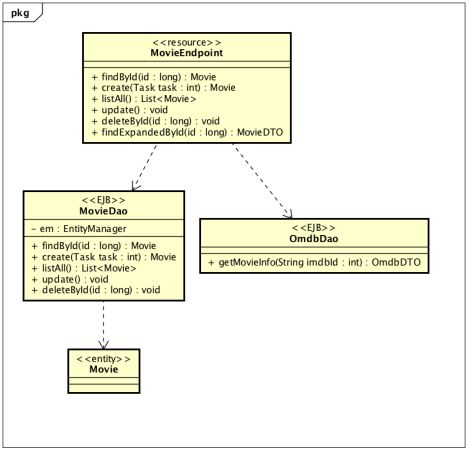
Demo

Java 11, JAX-RS, CDI, EJB, MicroProfile

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

Stacks tradicionales

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



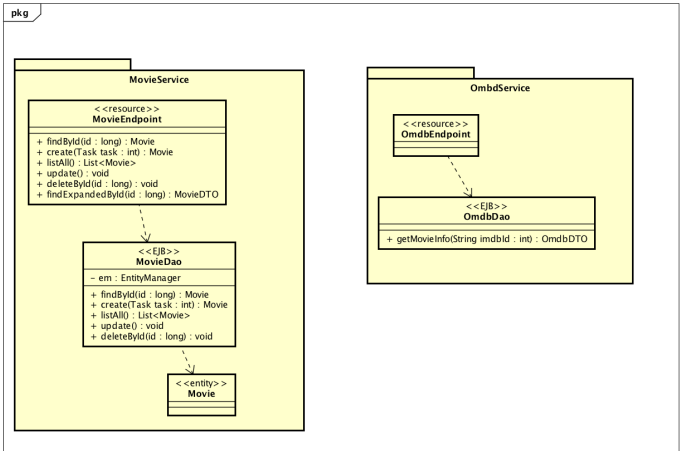
@tuxtor

EE + MicroProfile - Demo

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

Payara Micro: EJB, JTA

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



@tuxto

12 factores 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

@tuxtor

ACADEMIK

Escríbenos a cursos@academik.io

www.academik.io