5 Agile Steps to building Elastic and Cloud-ready apps

Ondro Mihályi, Payara, http://www.payara.fish

@OMIHALYI

What is cloud ready?

- **Spring, Java EE / Jakarta EE, MicroProfile or Lagom**
- **AWS, Azure or Openshift**
- **SQL or NoSQL**
- **REST or EJB**

Is it really about technology?

kubernetes

Even cool tech can be painful

Cloud ready requirements

- Pluggable persistence
- Scalable according to the load
- Low coupling
- External configuration
- Failure recovery
- Security
- Monitoring

There are even more according to the 12 factor applications manifesto

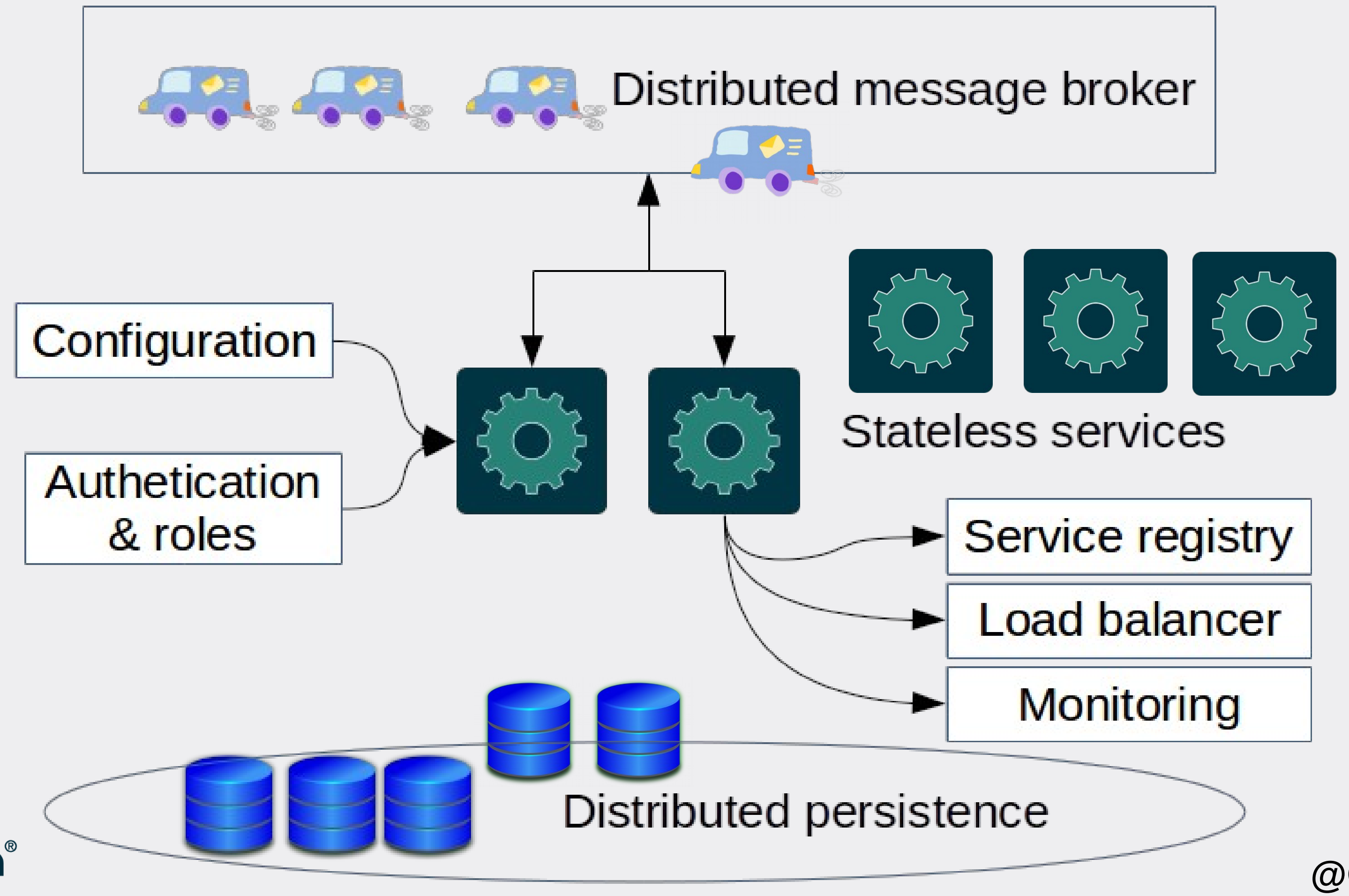
Solution?

@OMIHALYI

Solution: agile evolution

- **Simple API abstractions**
- **Flexible implementations**
- **Application logic first, against a solid platform**
- **Abstract the technology, prepare for refactoring**
- **Choose final technology later**

Cloud-ready architecture

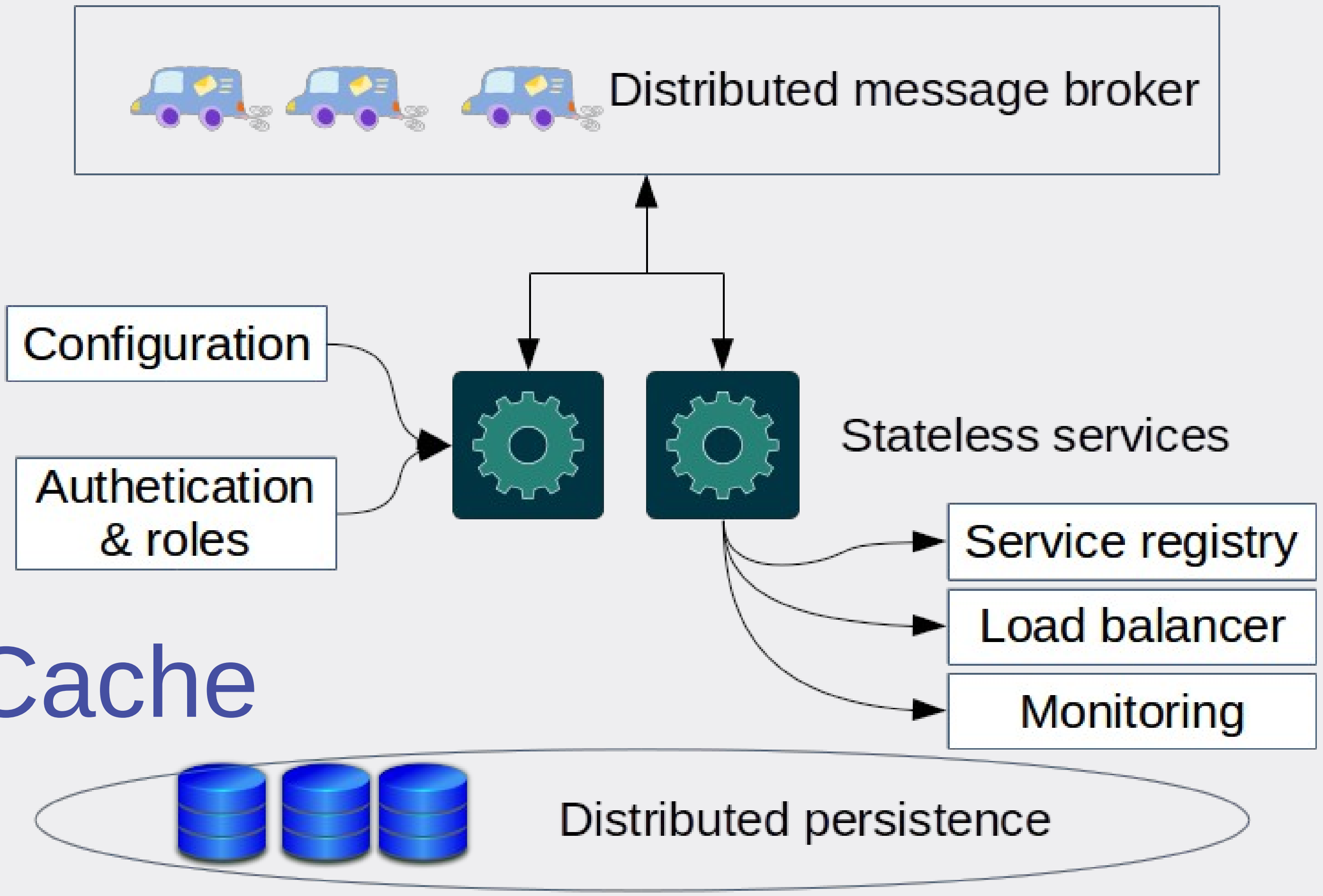


payara®

@OMIHALYI

1. JCACHE

@OMIHALYI



JCache

@OMIHALYI

JCache

- **Temporary cache → optimize reads**
- **Cache data-retrieval method calls**
- **Temporary key-value store, extensible to permanent with a read/write-through policy**
- **More than 10 implementations (also in Payara Micro and Spring)**
- **Distributed caches allow scalable storage**

JCache API

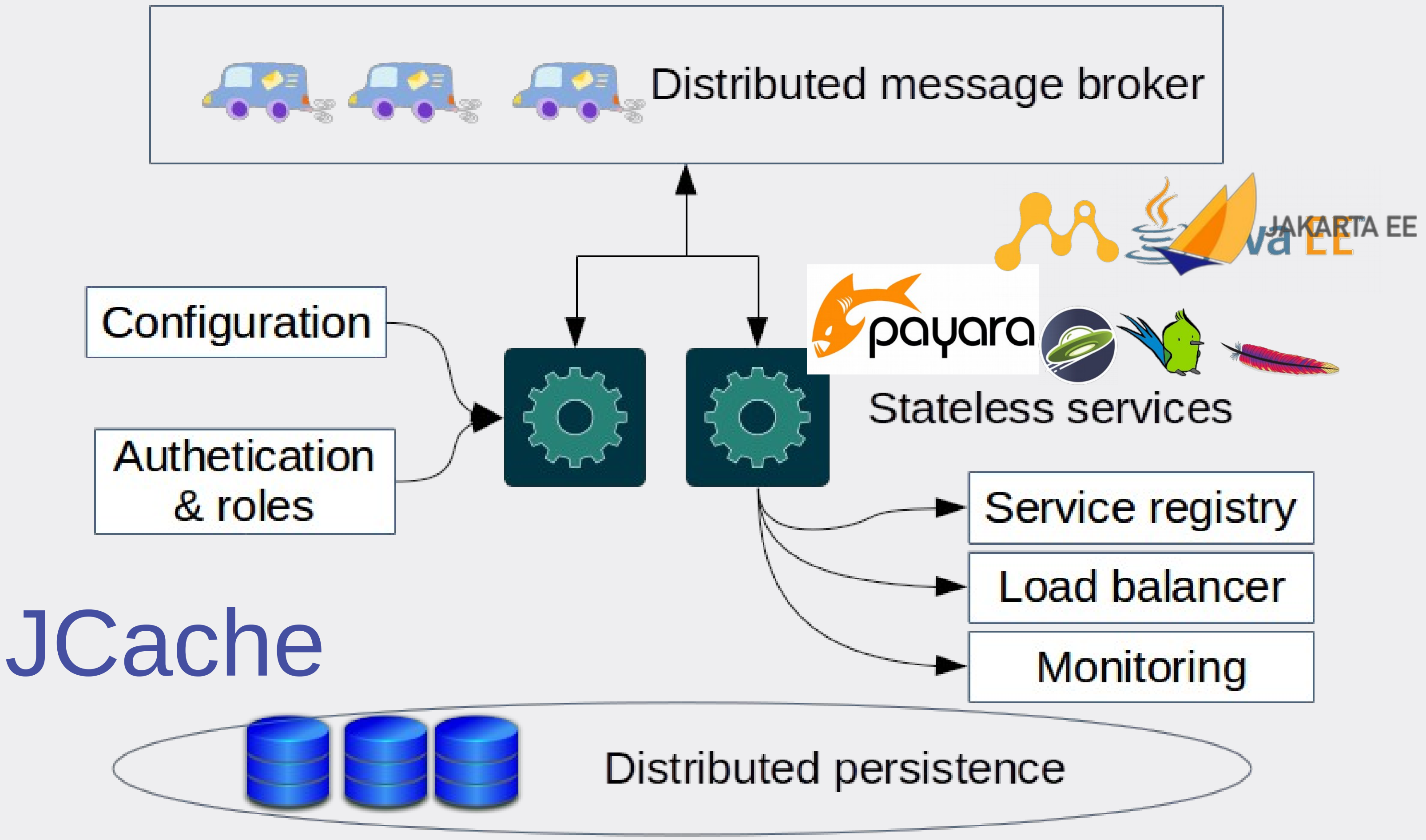
```
@CacheResult
User getUserForName(String name) { /*do if not cached*/ }

@Inject
Cache<String, User> users;

users.put(user.getName(), user);

User user = users.get(name);

StreamSupport.stream(users.spliterator(), false)
.filter( e -> e.getValue().getAge() > 50)
.count()
```

JCache in your app container

- **JCache widely available (in Payara Micro, Open Liberty, Spring Boot, …)**
- **In Java EE containers integrated with CDI**
- **Often powered by Hazelcast**
  - Distributed, auto-discovery of nodes
  - Data replication, even data distribution
  - Lite nodes possible without data
  - More features via Hazelcast extensions

2. SCALABLE RUNTIME

@OMIHALYI


Distributed message broker
JAKARTA EE
Java EE
payara
Stateless services
Configuration
Authetication & roles
Service registry
Load balancer
Monitoring
JCache
Distributed persistence

What is Payara Micro?

- **Executable JAR (~70MB disk size, ~30 MB RAM)**
- **Runs WAR and EAR from command line**
  - Also Uber JAR, embeddable (run from your app)
- **Forms dynamically scalable cluster**
- **Web Profile "plus", MicroProfile**
- **Opensource, Maven dep, release each 3 months**

Scale dynamically

- Run multiple instances with the same command

```
java -jar payara-micro.jar application.war
  --autoBindHttp
```

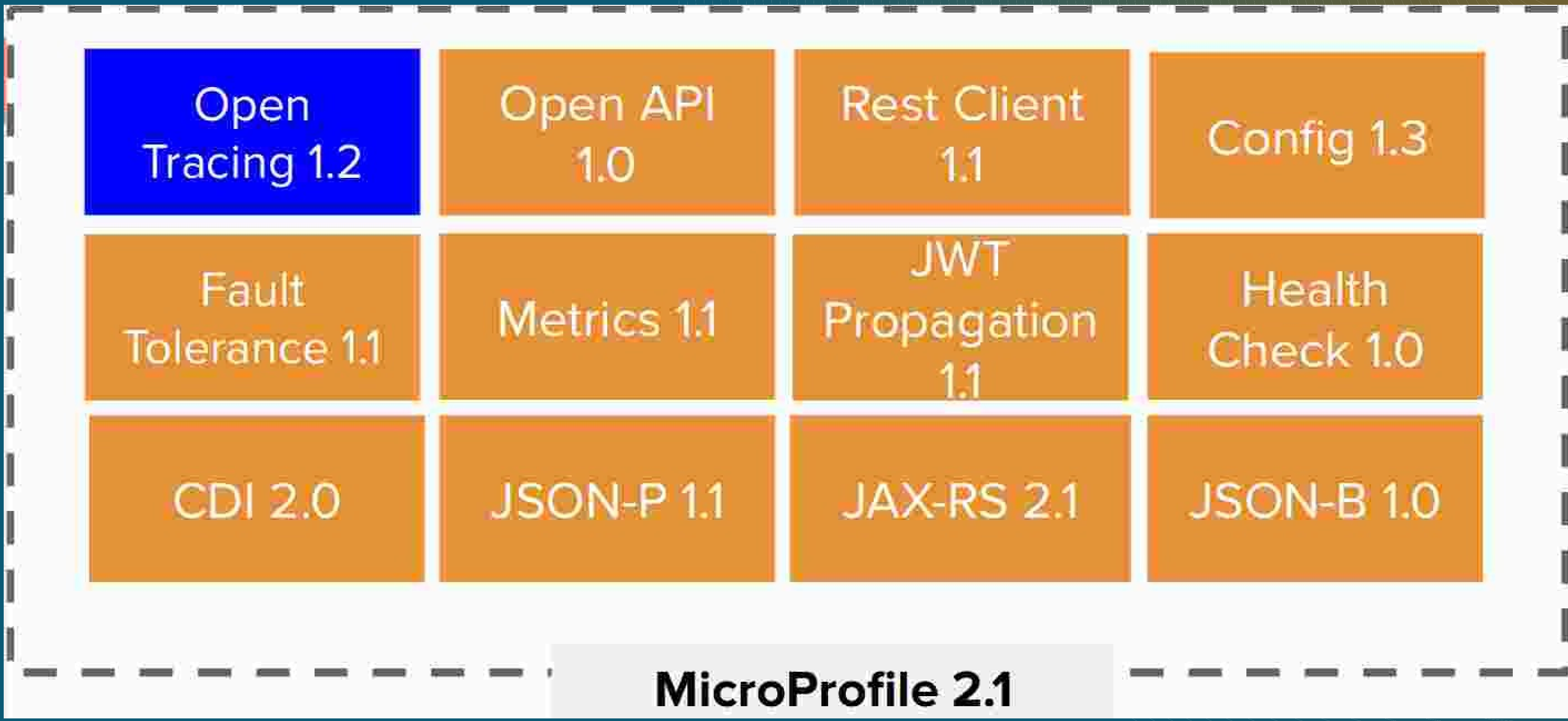
- Package as a single executable Uber JAR

```
java -jar payara-micro.jar application.war
  --outputUberJar application.jar
```

- Run embedded: `PayaraMicro.getInstance().bootStrap()`
- Run using Maven plugin: `mvn payara-micro:start`

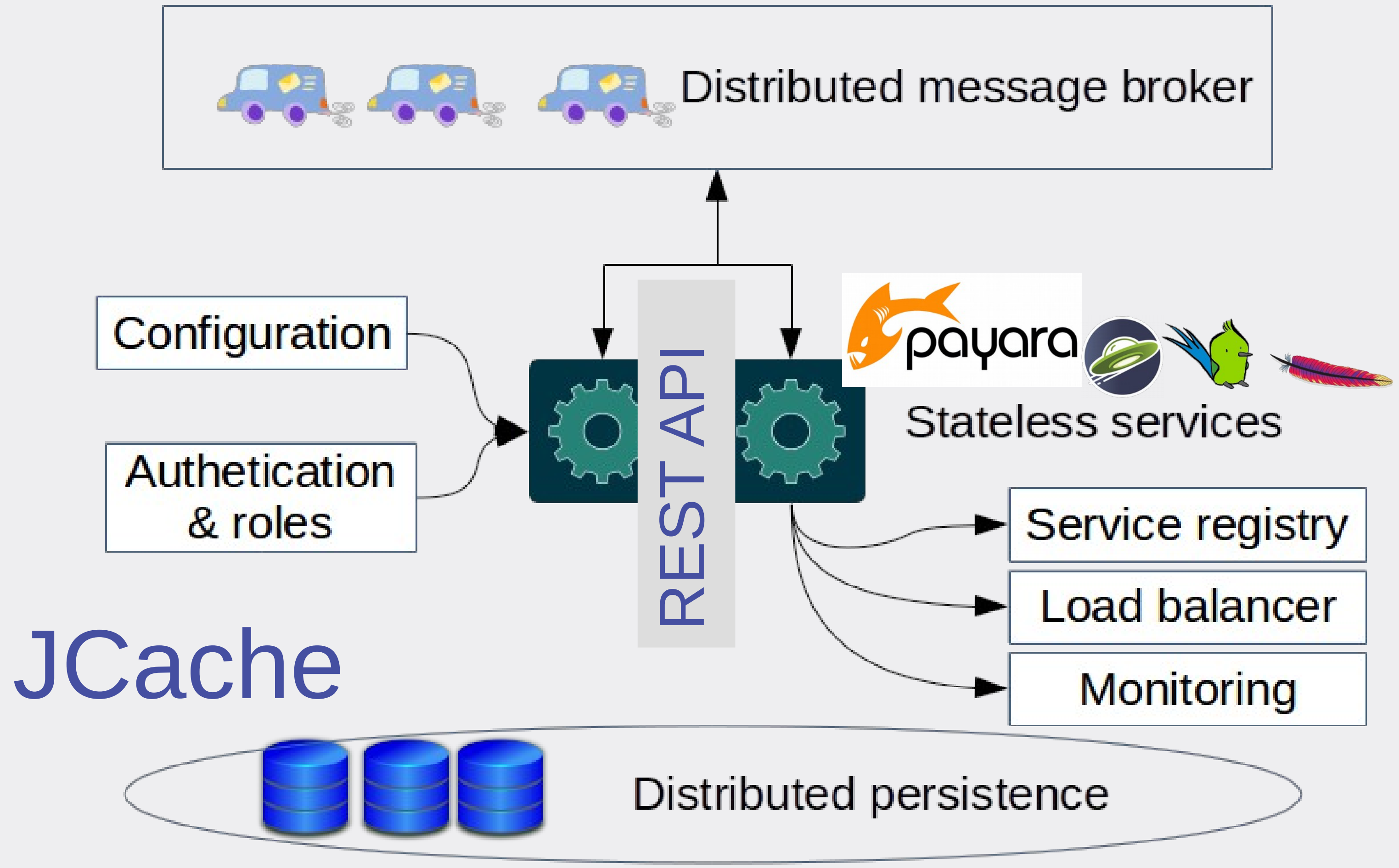
What is MicroProfile?

- **Project at Eclipse Foundation**
- **Common API, multiple implementations**
- **Extends Java EE**
- **Modern patterns:**
  - Microservices, Reactive, …
- **http://microprofile.io - open for everybody**


Open Tracing 1.2
Open API 1.0
Rest Client 1.1
Config 1.3
Fault Tolerance 1.1
Metrics 1.1
JWT Propagation 1.1
Health Check 1.0
CDI 2.0
JSON-P 1.1
JAX-RS 2.1
JSON-B 1.0
MicroProfile 2.1

3. RESTFUL

@OMIHALYI


Distributed message broker
Configuration
Authetication & roles
REST API
payara
Stateless services
Service registry
Load balancer
Monitoring
JCache
Distributed persistence

REST services API (server)

- JAX-RS endpoint

```
@GET
@Path("/{id}")
@Produces(MediaType.APPLICATION_JSON)
public User getUser(@PathParam("id")
    Integer id) {
        return userById(id);
}
```

REST services API (client)

- JAX-RS client

```
User user = client.target(url)
                  .path("all")
                  .request().get(User.class)
```

- MicroProfile client (much better abstraction)

```
User admin = userService.getUser("admin")
```

JSON binding

```
@JsonbNillable
public class User implements Serializable {

    private String name;

    @JsonbTransient
    private String description;

    @JsonbProperty("userId")
    private long id;
}
```

- new in Java EE 8 and MicroProfile 2.0

More about JSON-B: http://json-b.net

payara®

4. MESSAGING

payara®

@OMIHALYI

CDI events, really?

- **Part of Java EE API already**
- **Easy to send and observe messages**

- **Is it enough? What about:**
  - Sending events to other services
  - Message broker to decouple services
  - Transactions

CDI events, really?

What about:

- **Sending events to other services**
  - Nothing else is important in initial dev stage
- ~~**Message broker for reliable delivery**~~
- ~~**Transactions**~~

Payara CDI event bus

- **Out of the box in Payara Micro**
- **Uses embedded Hazelcast**
- **No config needed, events dispatched to all matching services**

```
@Inject @Outbound
Event<Payload> event;
```

```
void onMessage(
  @Observes @Inbound
  Payload event)
```

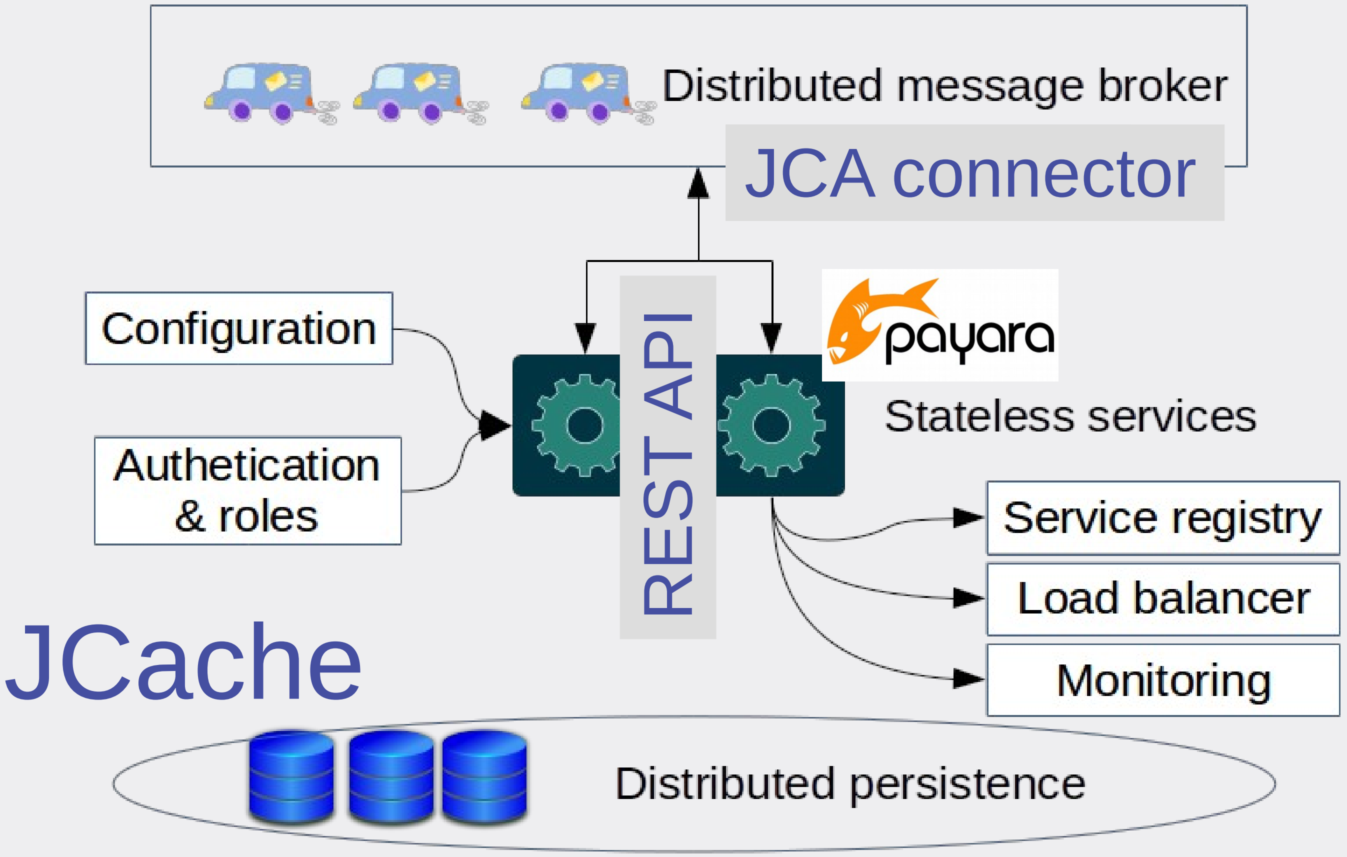
Events as an abstraction

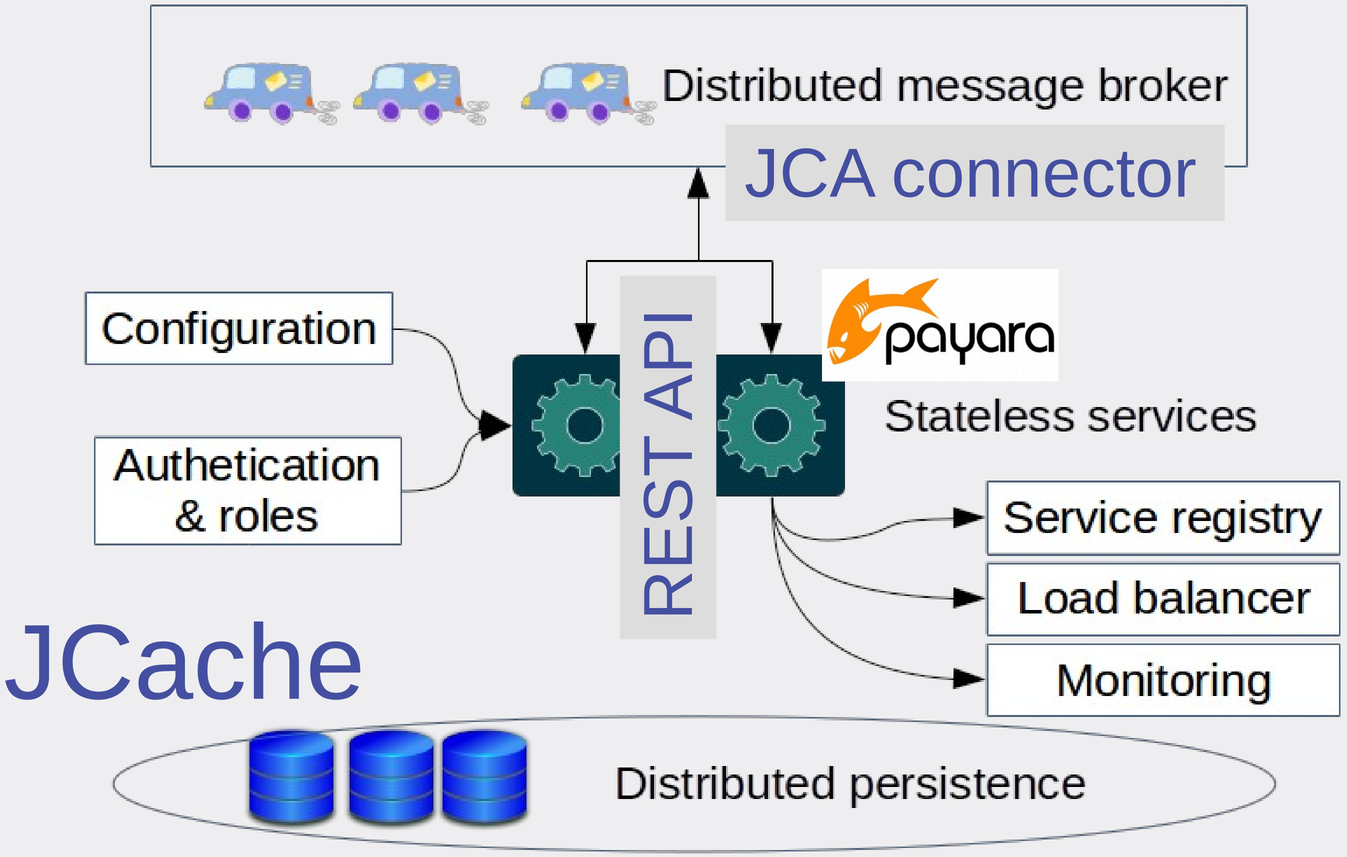
- **Transfer events to other services in an event handler**
  - Using distributed queues
  - Using any message broker
- **Distribute incoming messages as events**
- **Start simple, extend to robust**

One more option… JCA connector

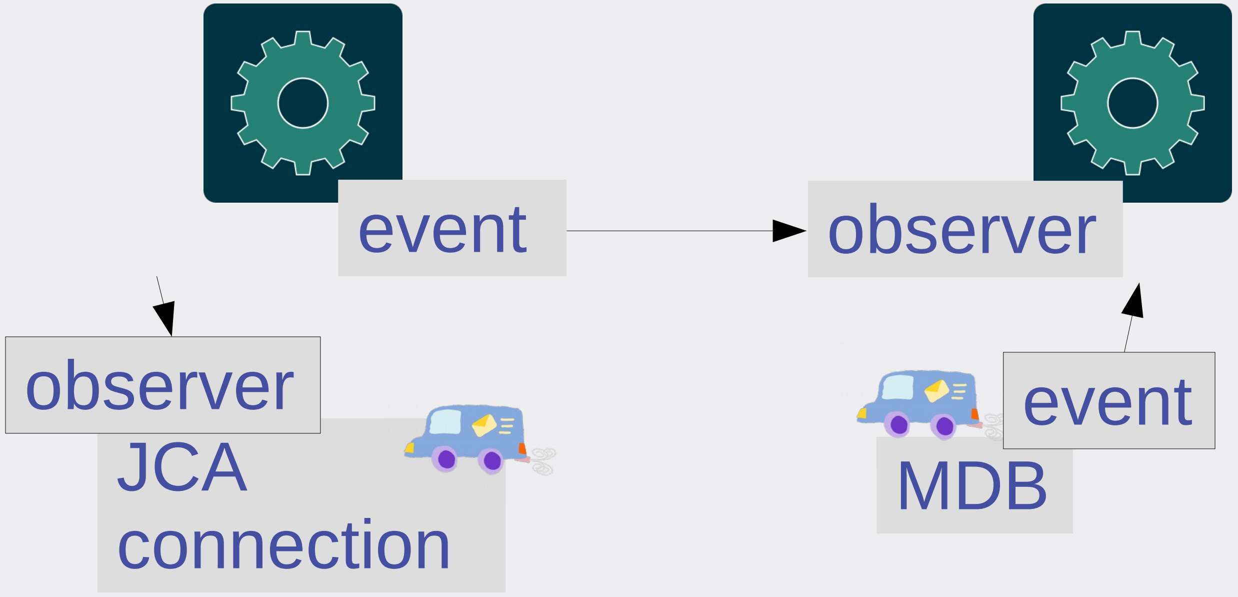
- **Message-driven beans, does it ring the bell?**
  - Not only for JMS but for any messaging infrastructure
- **Connetors on Github for AWS, Azure, Kafka, MQTT**

```
@MessageDriven(activationConfig = { … })
public class KafkaMDB implements KafkaListener {

    @OnRecord( topics={"my-topic"} )
    public void onMsg(ConsumerRecord record) {
        …
```


Distributed message broker
JCA connector
Configuration
Authetication & roles
REST API
payara
Stateless services
Service registry
Load balancer
Monitoring
JCache
Distributed persistence


Distributed message broker
JCA connector
Configuration
Authetication & roles
REST API
payara
Stateless services
Service registry
Load balancer
Monitoring
JCache
Distributed persistence

Or evolution to avoid refactoring



Evolutionary architecture

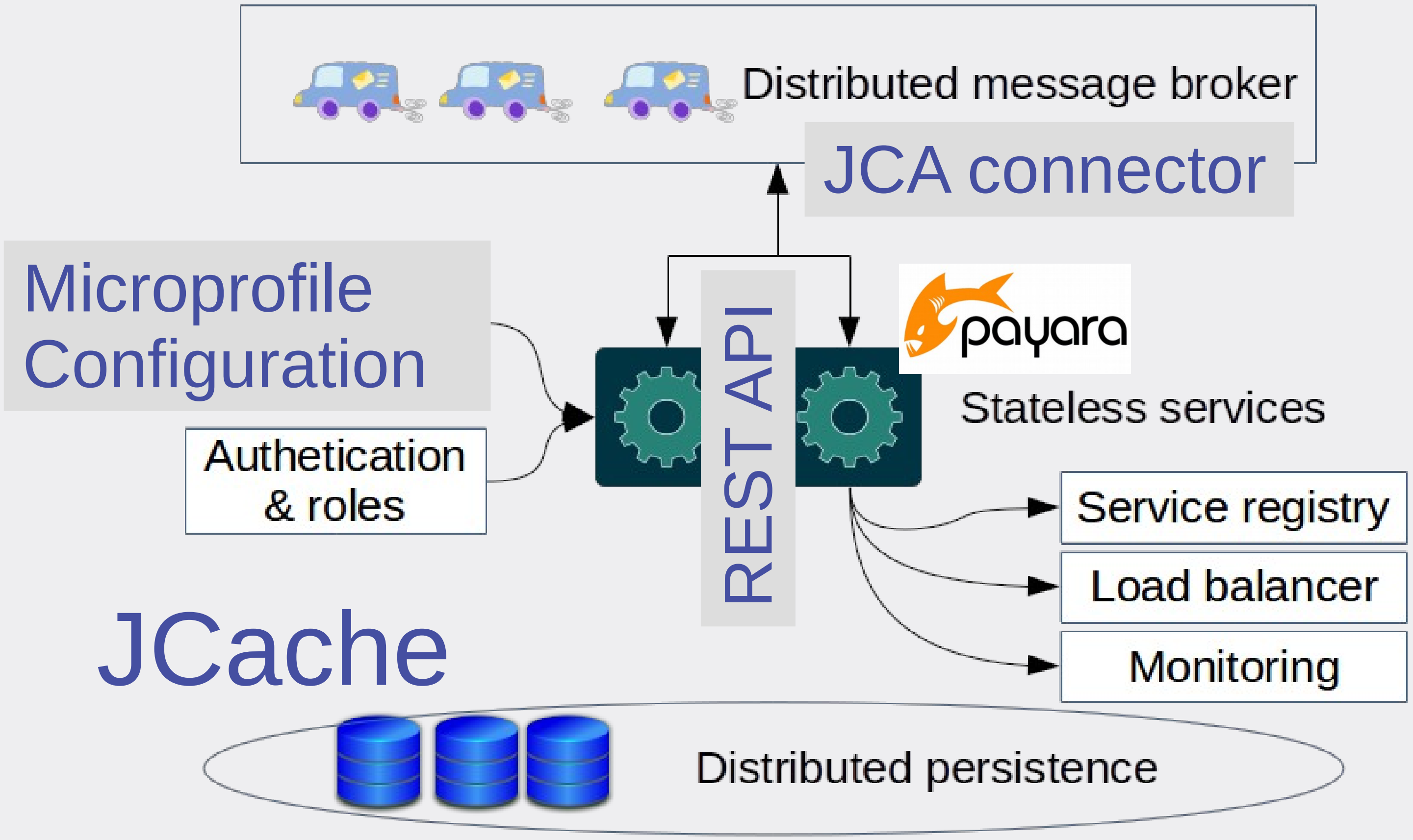
„An evolutionary architecture supports continual and incremental change as a first principle along multiple dimensions.“

„Microservices meet this definition.“

Neal Ford, Rebecca Parsons

http://evolutionaryarchitecture.com/

5. CONFIGURATION FACADE



payara®

@OMIHALYI

Microprofile Configuration

- **Standard config sources**
  - Env. variables
  - System properties
- **Pluggable sources**
  - Database?, secrets?
- **More sources in Payara Micro**
  - **Cluster-wide**
  - **Directory, secrets**
  - **Scoped** (server, app, module)

```
@Inject
@ConfigProperty(name =
            "myservice.url")
URL myService;
```

```
URL myService =
    ConfigProvider.getConfig()
    .getValue("myservice.url",
              URL.class);
```

payara®

@OMIHALYI

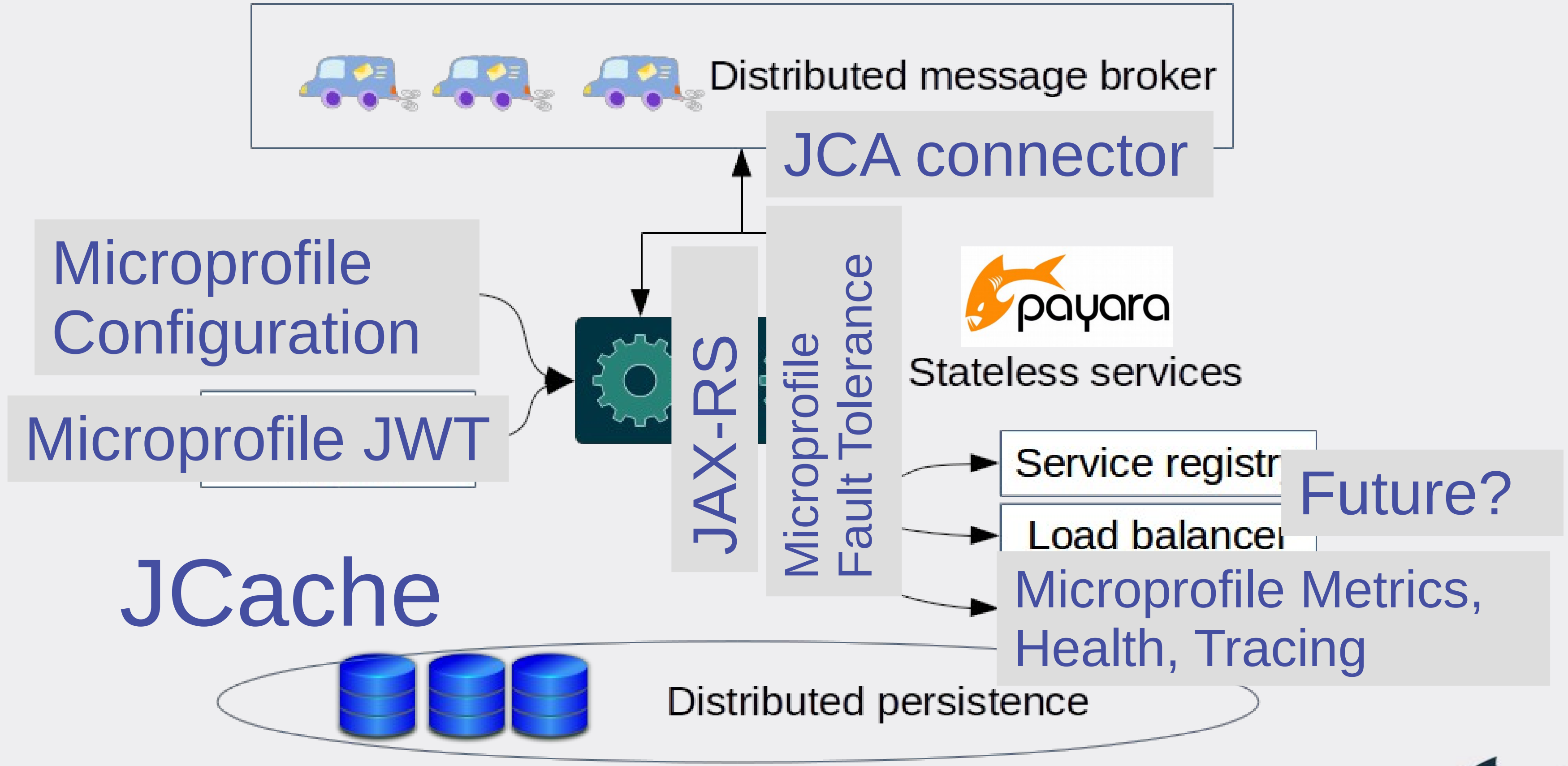
DEMO

BONUS: MONITORING

Is there a free lunch?

Microprofile provides a lot out of the box

- Metrics – monitoring data, statistics
- Health – problem detection and autorecovery
- Opentracing – connects related requests



Distributed message broker
JCA connector
Microprofile Configuration
Microprofile JWT
JAX-RS
Microprofile Fault Tolerance
payara
Stateless services
Service registr
Load balance
Future?
Microprofile Metrics, Health, Tracing
JCache
Distributed persistence

Thank you!

- https://microprofile.io/
- https://www.payara.fish/
- http://evolutionaryarchitecture.com/
- https://github.com/payara/Cloud-Connectors
- https://www.microprofile-ext.org/
- https://github.com/OndrejM-demonstrations/elastic-cloud-ready-apps

payara®

@OMIHALYI