KARLSRUHER ENTWICKLERTAG

Microservices with Apache Karaf and Apache CXF: practical experience

Andrei Shakirin, Talend

Agenda

- Microservices and OSGi
- Core ideas of OSGi
- Apache Karaf
- Design and develop in OSGi: the history of one project
- Conclusions and lessons learned

About Me

- Software architect in Talend Team
- PMC in Apache CXF
- Contributions in Apache Syncope, Apache Aries and Apache Karaf

Microservices

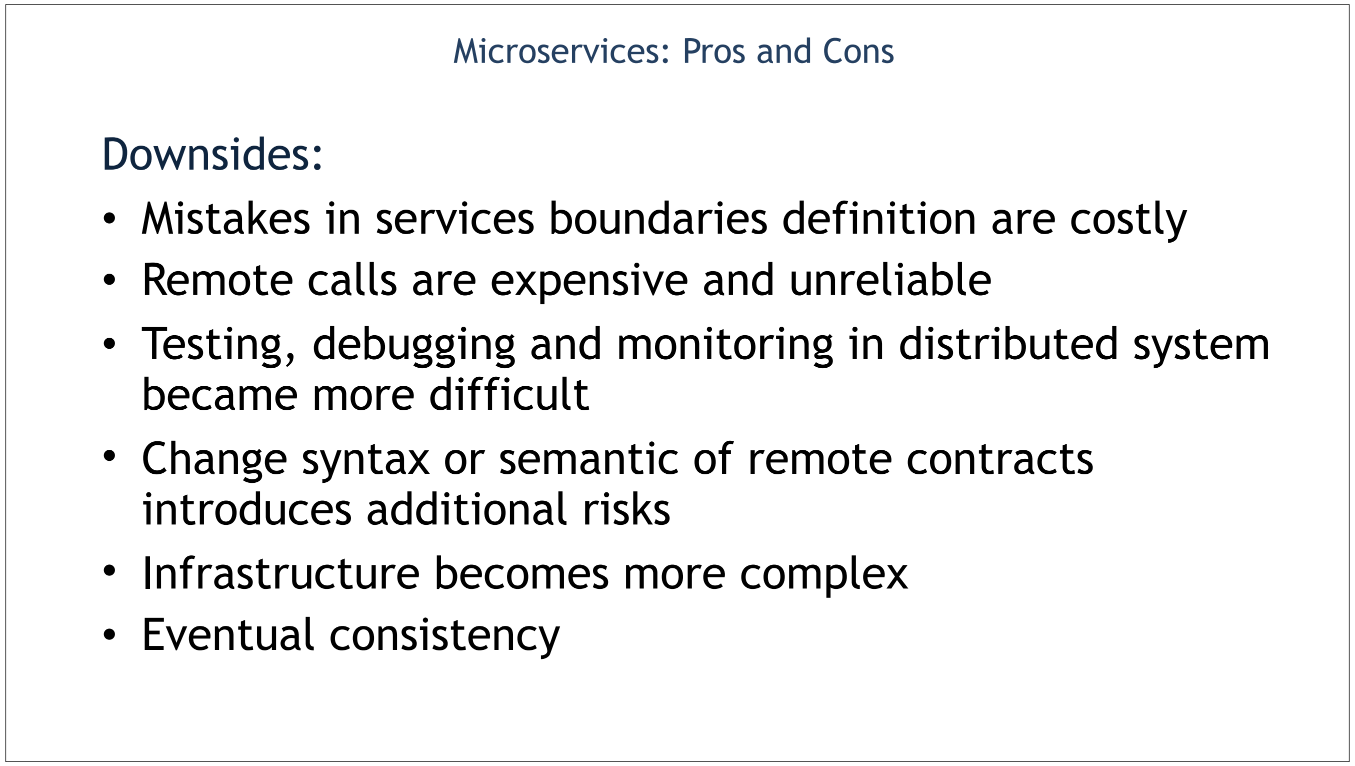
(James Lewis and Martin Fowler)

- Application as suite of small services
- Organization around business capabilities
- Each service runs in own process
- Smart endpoints and dumb pipes
- Decentralized data management and technologies
- Infrastructure automation

Microservices: Pros and Cons

Benefits:

- Services themselves are simple, focusing on doing one thing well
- Systems are loosely coupled
- Services and can be (relatively) independently developed and deployed by different teams
- Services can be scaled differently
- Service team can use the most appropriate technologies and programming languages



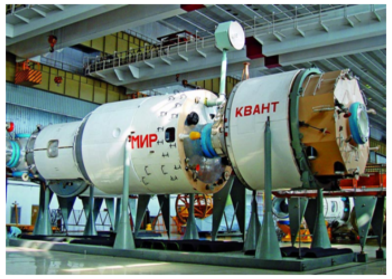
Microservices: Pros and Cons

Downsides:

- Mistakes in services boundaries definition are costly
- Remote calls are expensive and unreliable
- Testing, debugging and monitoring in distributed system became more difficult
- Change syntax or semantic of remote contracts introduces additional risks
- Infrastructure becomes more complex
- Eventual consistency

OSGi => Modular Applications

What is the module?



OSGi: Modules and Modularity

OSGi: Modules and Modularity

OSGi: software modules

- Implements a specific function
- Can be used alone or combined with others
- Provides functionality to be reused or replaced
- Has well defined name
- Has a version

The Central Repository

jars modules

OSGi: software modules

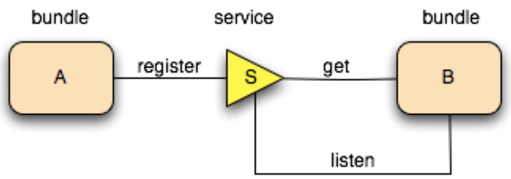
But:

- It is hard to achieve loosely coupling between the modules (only partial solutions: Class.forName; ServiceLoader; log-appenders)
- You cannot encapsulate functionality in the module
- Missing runtime control which version of the dependencies functionality will be used
- Self-describing module contract is missing

- Keep the name and version of JAR file
- Add explicit package dependencies (requirements)
- Add explicit package exports (capabilities)
- Provide API as external contract (OSGi services)

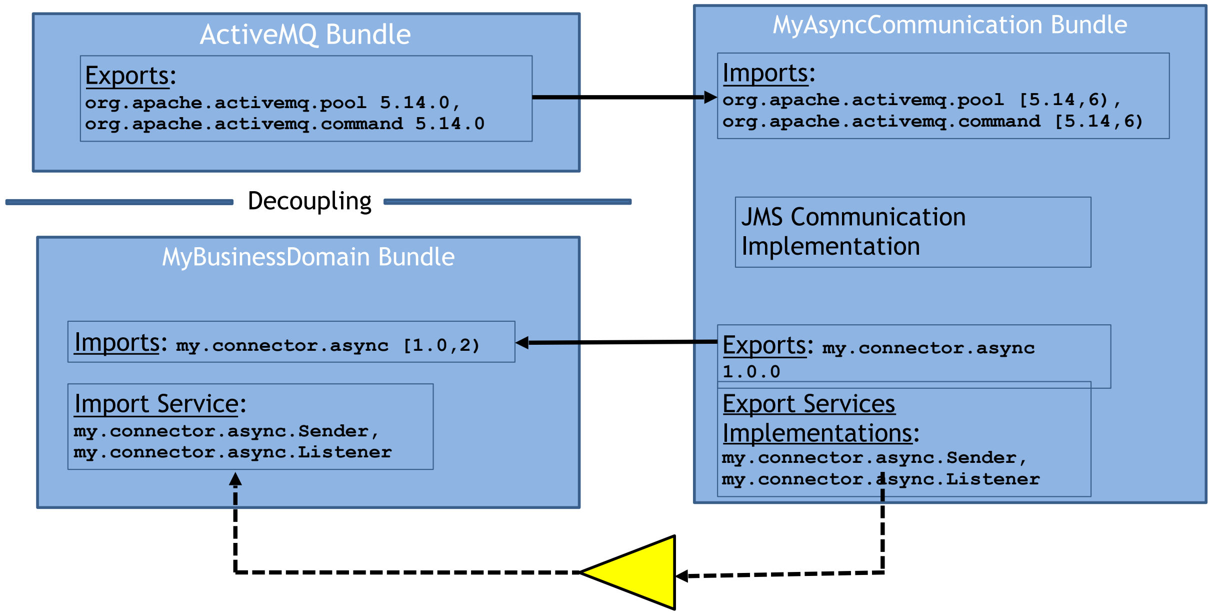
OSGi bundle

OSGi Services



- Service Contract is one or more java interfaces
- Bundle can register the service in registry
- Other bundle can get and listen for the service
- Multiple registered services can be distinguished using properties
- No any coupling between bundles except Service Contract: neither in code, no on the classpath (different to java ServiceLoader)

OSGi Decoupling



ActiveMQ Bundle

Exports:
`org.apache.activemq.pool 5.14.0,`
`org.apache.activemq.command 5.14.0`

MyAsyncCommunication Bundle

Imports:
`org.apache.activemq.pool [5.14,6),`
`org.apache.activemq.command [5.14,6)`

JMS Communication Implementation

Exports: `my.connector.async 1.0.0`

Export Services Implementations:
`my.connector.async.Sender,`
`my.connector.async.Listener`

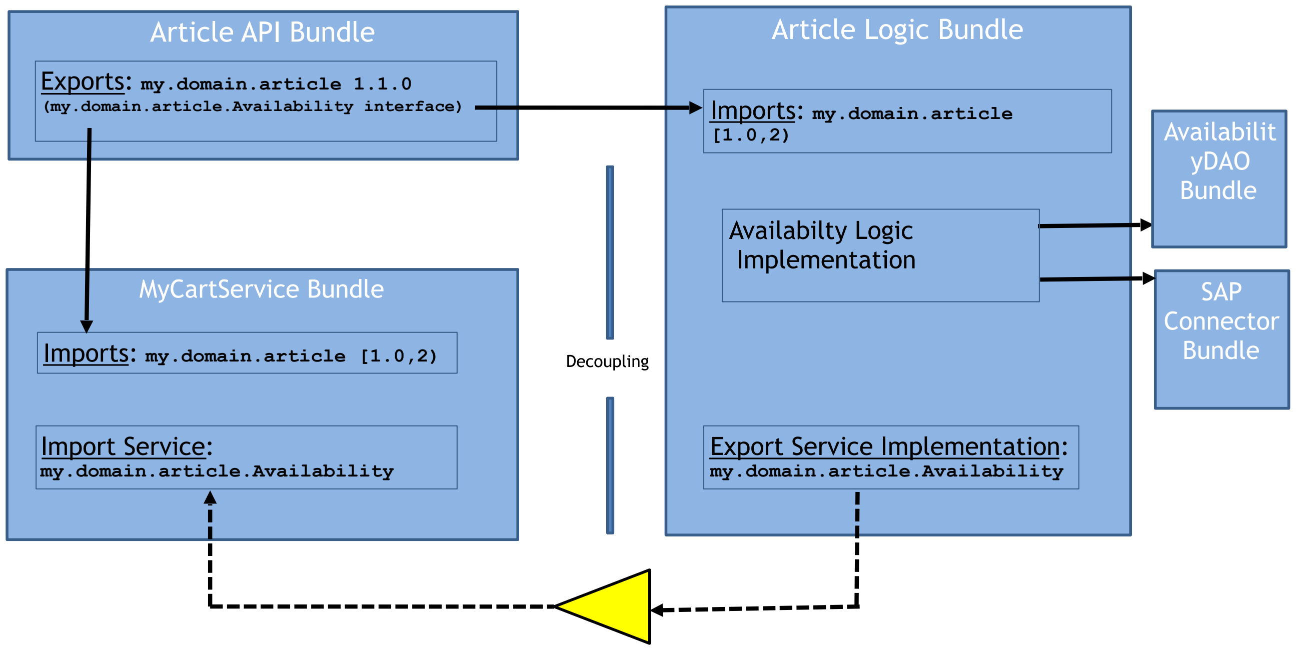
Decoupling

MyBusinessDomain Bundle

Imports: `my.connector.async [1.0,2)`

Import Service:
`my.connector.async.Sender,`
`my.connector.async.Listener`

OSGi Decoupling



Article API Bundle

Exports: `my.domain.article 1.1.0`
`(my.domain.article.Availability interface)`

Article Logic Bundle

Imports: `my.domain.article`
`[1.0,2)`

Availabilty Logic
Implementation

Availabilit
yDAO
Bundle

SAP
Connector
Bundle

MyCartService Bundle

Imports: `my.domain.article [1.0,2)`

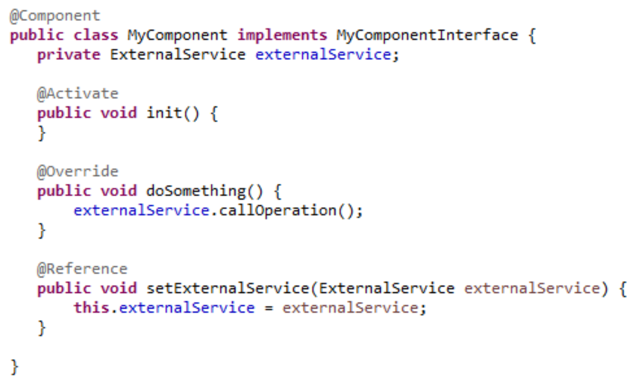
Import Service:
`my.domain.article.Availability`

Decoupling

Export Service Implementation:
`my.domain.article.Availability`

Declare OSGi Services: Option 1

- Declarative Services



```
@Component
public class MyComponent implements MyComponentInterface {
    private ExternalService externalService;

    @Activate
    public void init() {
    }

    @Override
    public void doSomething() {
        externalService.callOperation();
    }

    @Reference
    public void setExternalService(ExternalService externalService) {
        this.externalService = externalService;
    }

}
```

Christian Schneider Blog: "Apache Karaf Tutorial part 10 - Declarative services"

Declare OSGi Services: Option 2

- Blueprint

```
<?xml version="1.0" ?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
   xsi:schemaLocation="
            http://www.osgi.org/xmlns/blueprint/v1.0.0
            http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd>

   <bean id="myComponent" class="my.company.MyComponent">
     <property name="externalService" ref="externalService" />
   </bean>

   <reference id="externalService" interface="my.company.ExternalService" />
   <service ref="myComponent" interface="my.company.MyComponentInterface" />

</blueprint>
```

```
@Component
public class MyComponent implements MyComponentInterface {
   private ExternalService externalService;

   @PostConstruct
   public void init() {
   }

   @Override
   public void doSomething() {
       externalService.callOperation();
   }

   @Inject
   public void setExternalService(@OsgiService ExternalService externalService) {
       this.externalService = externalService;
   }

}
```

```
<plugin>
    <groupId>org.apache.aries.blueprint</groupId>
    <artifactId>blueprint-maven-plugin</artifactId>
    <executions>
        <execution>
            <phase>process-classes</phase>
            <goals>
                <goal>blueprint-generate</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <scanPaths>
            <scanPath>my.company</scanPath>
        </scanPaths>
    </configuration>
</plugin>
```

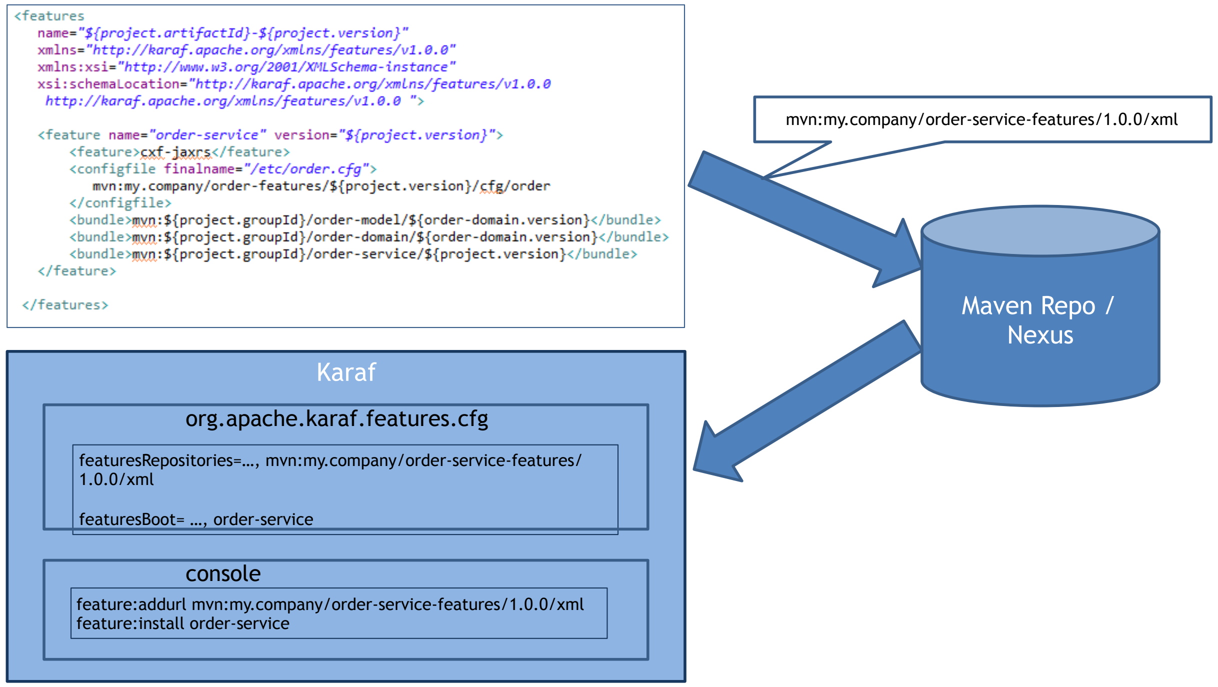
Classic Microservices vs OSGi

Aspect	Microservices	OSGi
Application structure	Suite of small services	Suite of bundles / modules
Boundaries	Around business capabilities	Modularization around business and technical aspects
Communication	Lightweight remote	Flexible: local or remote
Contract	Remote API	Local java interfaces or remote API
Decentralized Data Management	Desired	Depends on requirements for single process, desired for multiple processes
Infrastructure Automation	Desired	Desired

Apache Karaf

- OSGi based Container using Apache Felix or Eclipse Equinox implementations
- Runs as Container, Docker Image, embedding (karaf-boot)
- Provisioning (maven repository, file, http, …)
- Configuration
- Console
- Logging, Management, Security

Karaf Features



```
<features
   name="${project.artifactId}-${project.version}"
   xmlns="http://karaf.apache.org/xmlns/features/v1.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://karaf.apache.org/xmlns/features/v1.0.0
    http://karaf.apache.org/xmlns/features/v1.0.0 ">

   <feature name="order-service" version="${project.version}">
       <feature>cxf-jaxrs</feature>
       <configfile finalname="/etc/order.cfg">
          mvn:my.company/order-features/${project.version}/cfg/order
       </configfile>
       <bundle>mvn:${project.groupId}/order-model/${order-domain.version}</bundle>
       <bundle>mvn:${project.groupId}/order-domain/${order-domain.version}</bundle>
       <bundle>mvn:${project.groupId}/order-service/${project.version}</bundle>
   </feature>

 </features>
```

mvn:my.company/order-service-features/1.0.0/xml

Maven Repo / Nexus

Karaf

org.apache.karaf.features.cfg

featuresRepositories=..., mvn:my.company/order-service-features/1.0.0/xml

featuresBoot= ..., order-service

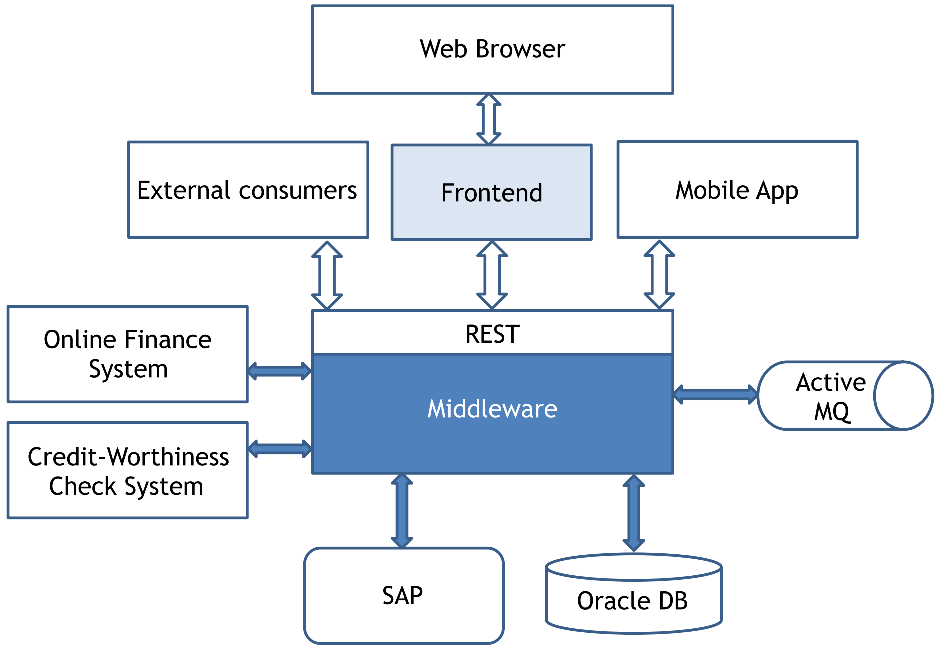
console

feature:addurl mvn:my.company/order-service-features/1.0.0/xml
feature:install order-service

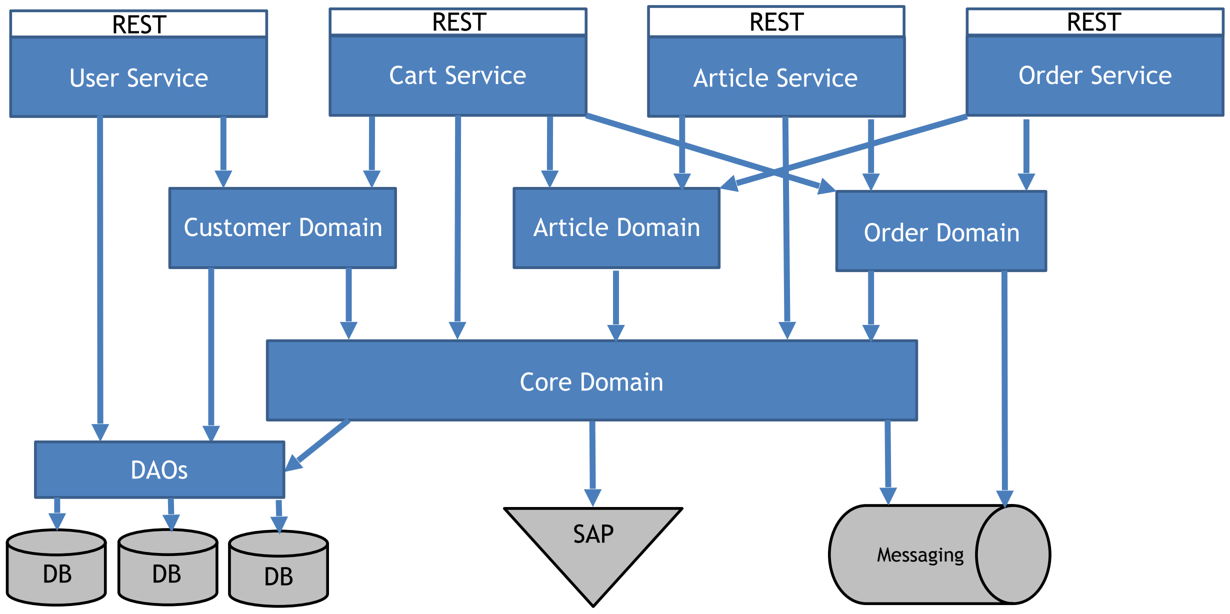
Migration to OSGi in eCommerce Project

- Business Domain: WebShop, eCommerce
- Team: 20 – 30 persons
- Initial technologies: Java, Spring, Hibernate, Apache CXF, Apache Camel, ActiveMQ, Tomcat
- Current technologies: Java, Hibernate, Apache CXF, ActiveMQ, OSGi + Apache Karaf, SpringBoot, MongoDB

Online Shop Architecture



Online Shop Design



REST

User Service

REST

Cart Service

REST

Article Service

REST

Order Service

Customer Domain

Article Domain

Order Domain

Core Domain

DAOs

DB

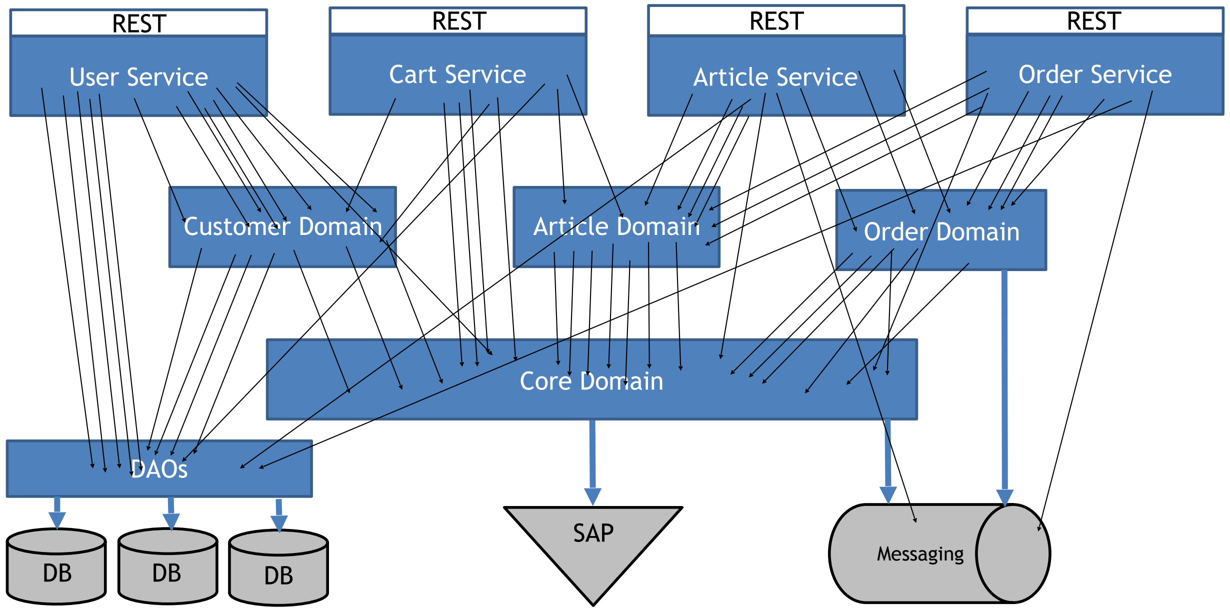
DB

DB

SAP

Messaging

Online Shop Design



REST

User Service

REST

Cart Service

REST

Article Service

REST

Order Service

Customer Domain

Article Domain

Order Domain

Core Domain

DAOs

DB

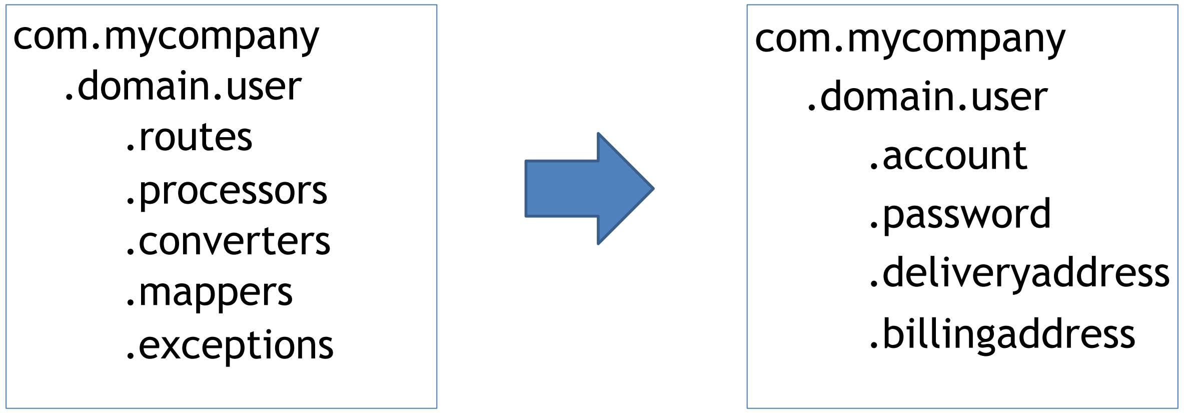
DB

DB

SAP

Messaging

Step 1: Packages Refactoring

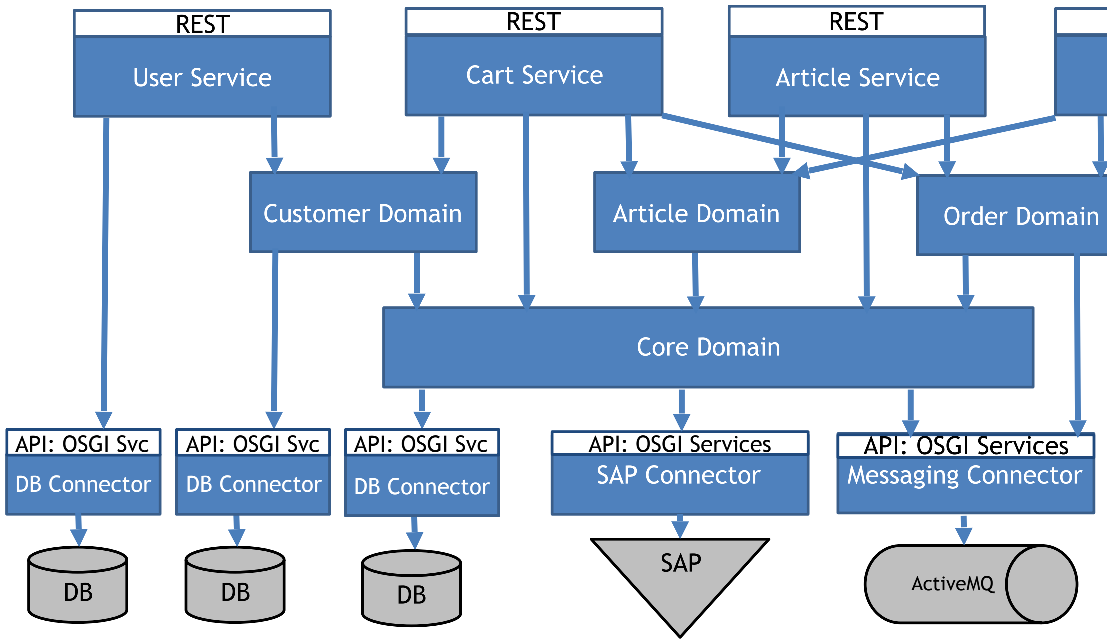


```
com.mycompany
   .domain.user
       .routes
       .processors
       .converters
       .mappers
       .exceptions
```

```
com.mycompany
   .domain.user
       .account
       .password
       .deliveryaddress
       .billingaddress
```

- Classes for business function are grouped together -> high cohesion
- Less dependencies between packages -> low coupling
- Private and public packages are easily recognizable (`model, api, impl`)

Step 2: Connectors API



REST

User Service

REST

Cart Service

REST

Article Service

REST

Order Service

Customer Domain

Article Domain

Order Domain

Core Domain

API: OSGI Svc

DB Connector

API: OSGI Svc

DB Connector

API: OSGI Svc

DB Connector

API: OSGI Services

SAP Connector

API: OSGI Services

Messaging Connector

DB

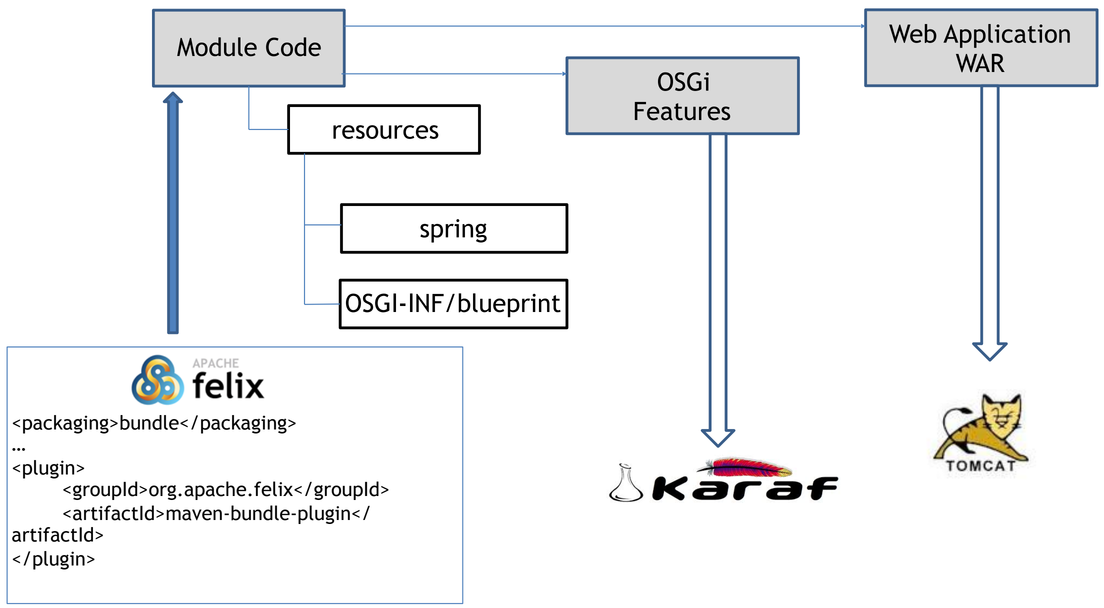
DB

DB

SAP

ActiveMQ

Step 3: Parallel Web And OSGi Deployment



```
<packaging>bundle</packaging>
...
<plugin>
        <groupId>org.apache.felix</groupId>
        <artifactId>maven-bundle-plugin</
artifactId>
</plugin>
```

Step 4: Refactor Complex Domain Logic (Camel Routes)

```
from(„vm:PriceAndAvailability“)
 .bean(availabilityOptionsMapper)
 .multicast(hdrAggregationStrategy
)
 .parallelProcessing().timeout(100)
    .to(„direct:getPrice“)
    .to („direct:getAvailability“)
 .end
.validate(availabilityValidator)
.bean(priceAvailResponseMapper)
```

1

```
PriceAndAvailResult getPriceAndAvail (Cart cart,
          AvailabilityOptions options);
```

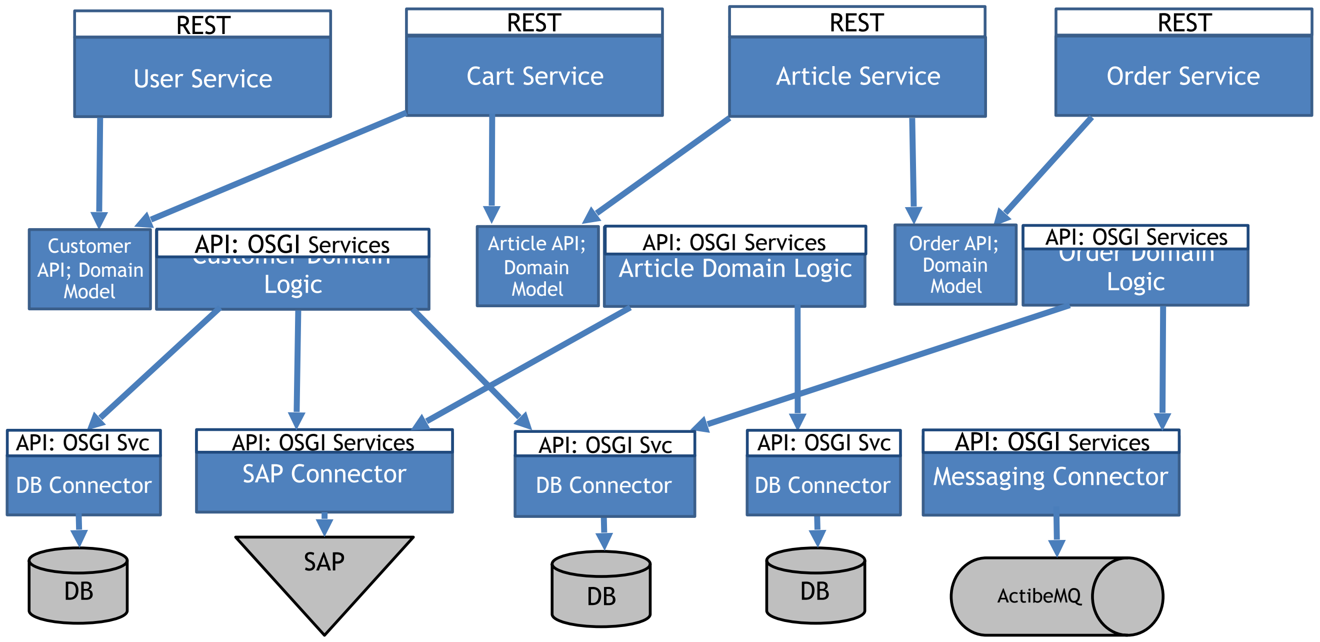
2

```
ATPOptions atpOptions = mapAvailabilityOptions(options);
…
final Future<AvailabilityReturner> availabilityFuture =
     executorService.submit(availabilityTask);
final Future<PriceReturner> priceFuture =
     executorService.submit(priceTask);
…
validateAvailability(availability);
PriceAndAvailResult result = new
PriceAndAvailabilityResult(availability,
      price);
```

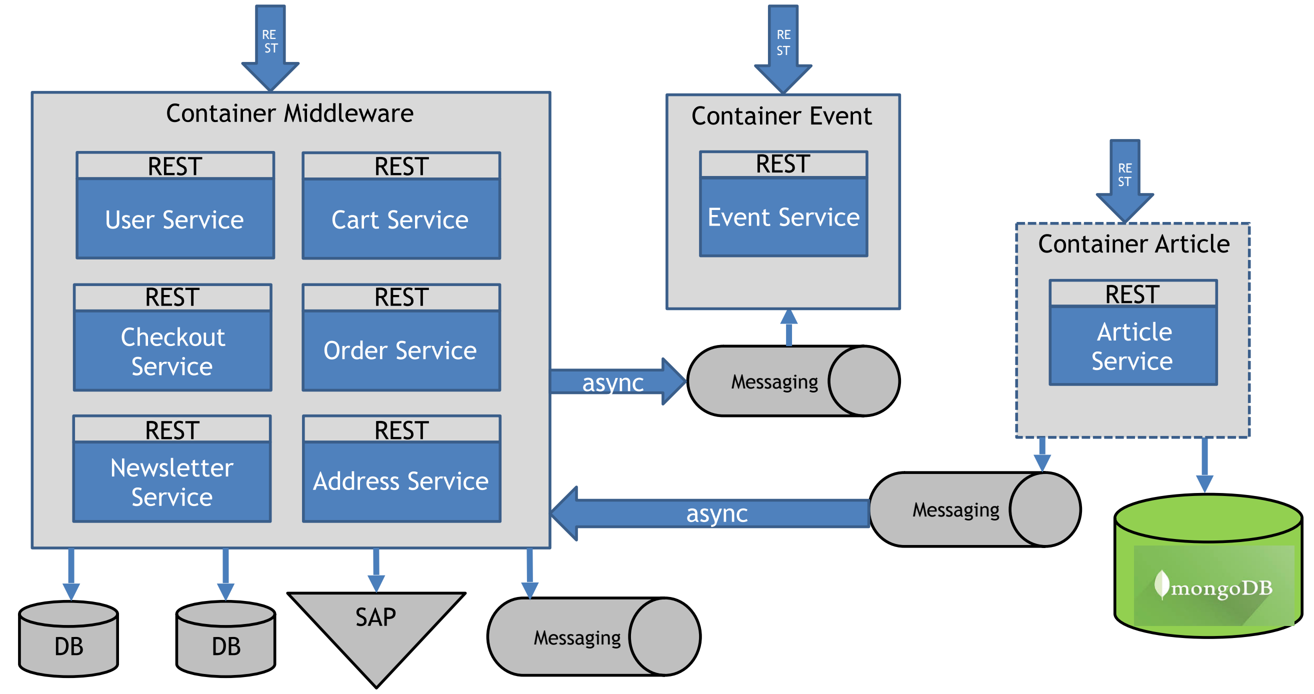
- What type of data is transmitted?
- Debug me ☺️
- Would it be harder in plain Java?

- Type safe interfaces
- Clearly shows what data is proceed
- Not essentially verbose as Camel route
- Easy to debug and understand for team

Step 4: Domains APIs And Refactoring



Step 5: Separate containers for some services

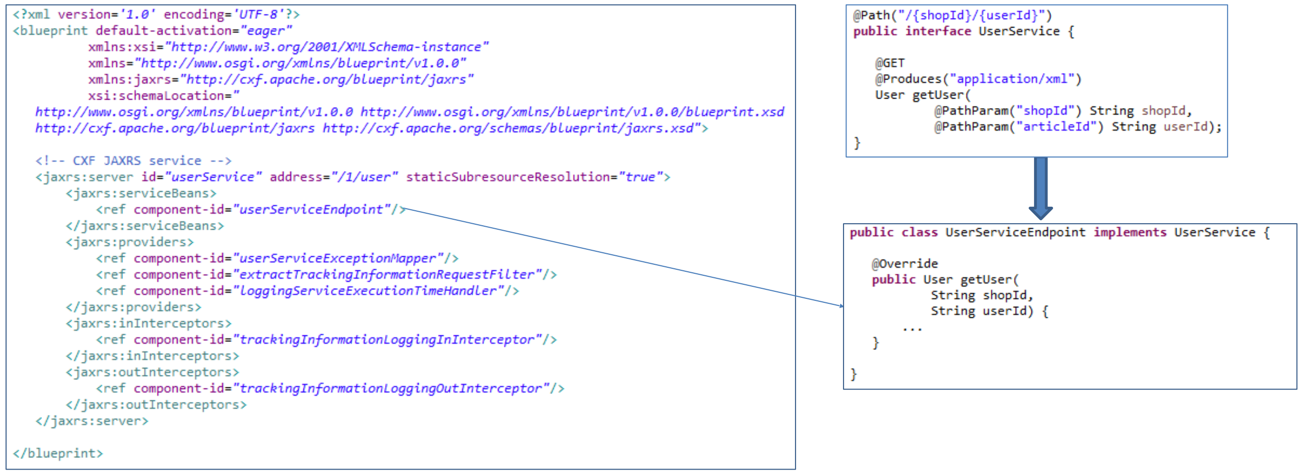


REST Communication in OSGi

- DOSGi and Aries Remote Service Admin (RSA)

  Christian Schneider “Lean Microservices on OSGi“, ApacheCon EU 2016

- Explicit via CXF

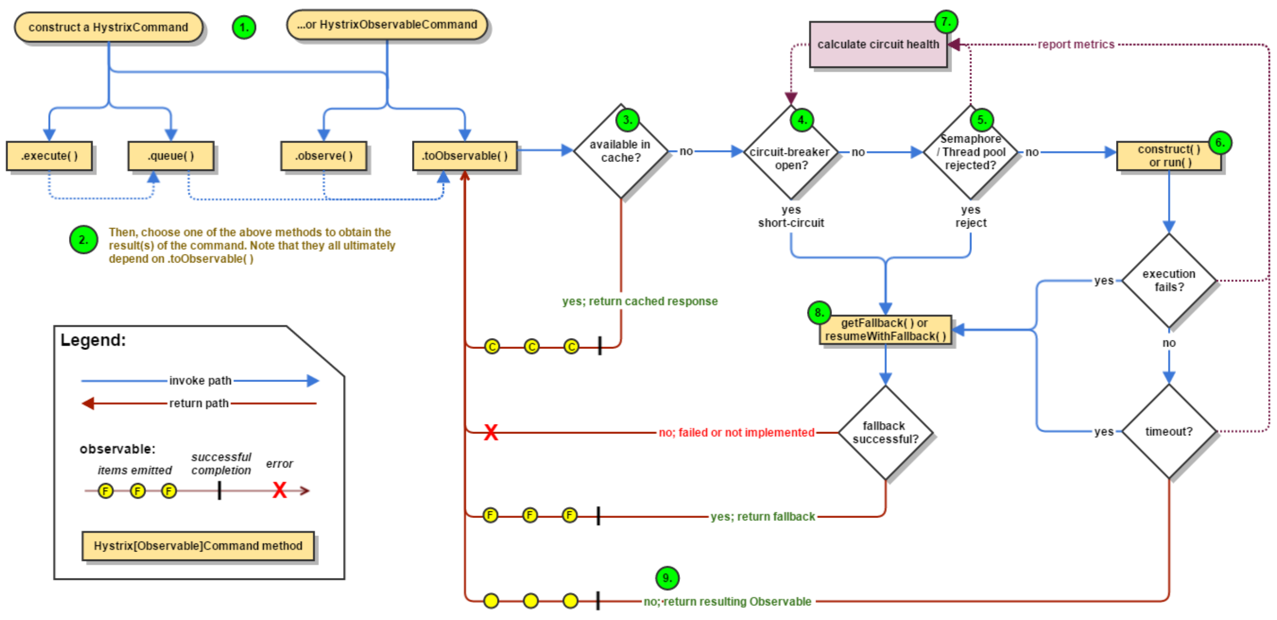


```
<?xml version='1.0' encoding='UTF-8'?>
<blueprint default-activation="eager"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
          xmlns:jaxrs="http://cxf.apache.org/blueprint/jaxrs"
          xsi:schemaLocation="
   http://www.osgi.org/xmlns/blueprint/v1.0.0 http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd
   http://cxf.apache.org/blueprint/jaxrs http://cxf.apache.org/schemas/blueprint/jaxrs.xsd">

   <!-- CXF JAXRS service -->
   <jaxrs:server id="userService" address="/1/user" staticSubresourceResolution="true">
       <jaxrs:serviceBeans>
           <ref component-id="userServiceEndpoint"/>
       </jaxrs:serviceBeans>
       <jaxrs:providers>
           <ref component-id="userServiceExceptionMapper"/>
           <ref component-id="extractTrackingInformationRequestFilter"/>
           <ref component-id="loggingServiceExecutionTimeHandler"/>
       </jaxrs:providers>
       <jaxrs:inInterceptors>
           <ref component-id="trackingInformationLoggingInInterceptor"/>
       </jaxrs:inInterceptors>
       <jaxrs:outInterceptors>
           <ref component-id="trackingInformationLoggingOutInterceptor"/>
       </jaxrs:outInterceptors>
   </jaxrs:server>

</blueprint>
```

```
@Path("/{shopId}/{userId}")
public interface UserService {

   @GET
   @Produces("application/xml")
   User getUser(
           @PathParam("shopId") String shopId,
           @PathParam("articleId") String userId);
}
```

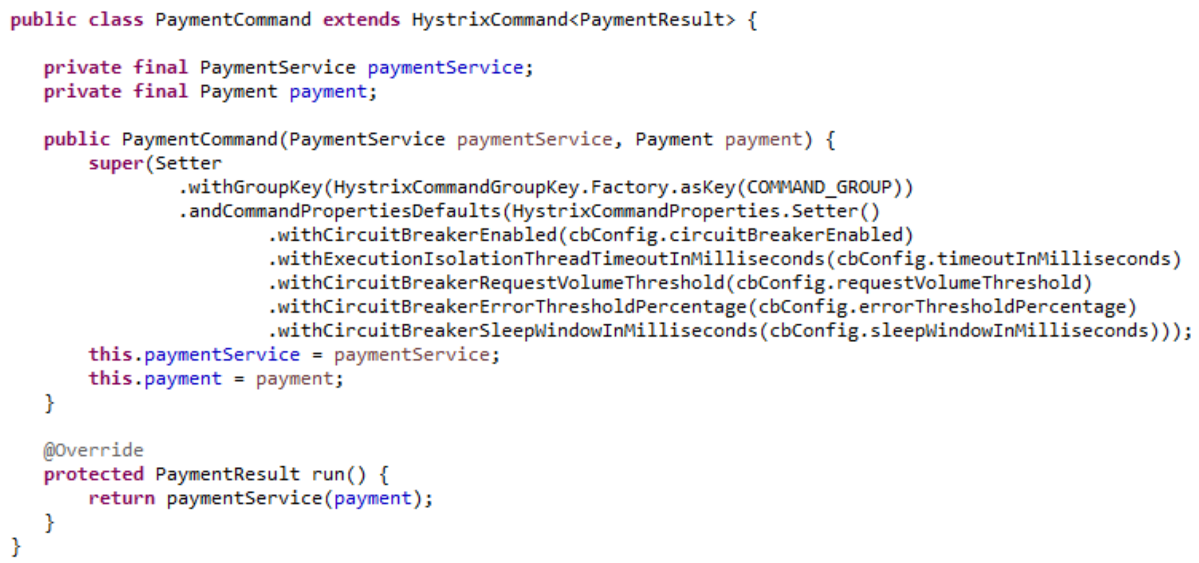
```
public class UserServiceEndpoint implements UserService {

   @Override
   public User getUser(
           String shopId,
           String userId) {
       ...
   }

}
```

Design For Failure With Hystrix(Netflix)



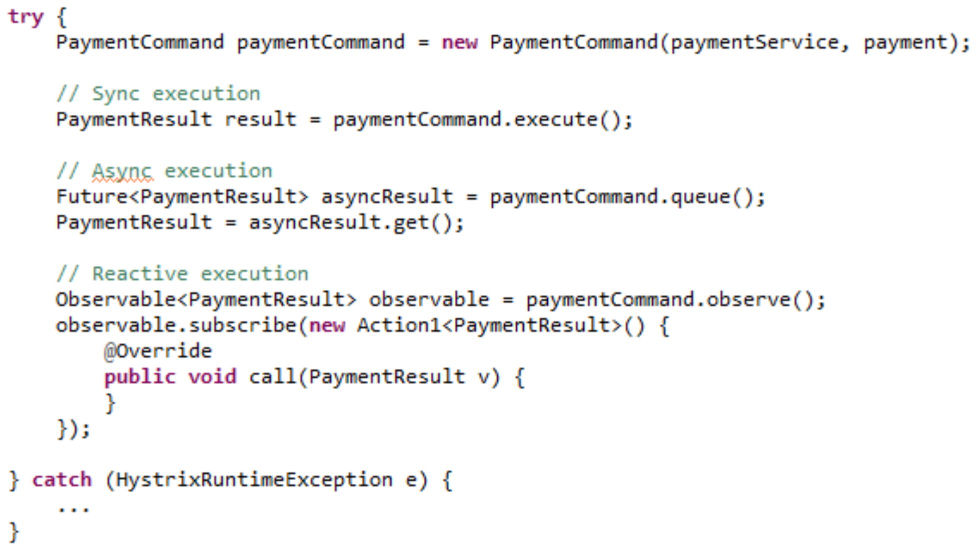
Then, choose one of the above methods to obtain the result(s) of the command. Note that they all ultimately depend on .toObservable( )

Resilience With Hystrix



```
public class PaymentCommand extends HystrixCommand<PaymentResult> {

    private final PaymentService paymentService;
    private final Payment payment;

    public PaymentCommand(PaymentService paymentService, Payment payment) {
        super(Setter
                .withGroupKey(HystrixCommandGroupKey.Factory.asKey(COMMAND_GROUP))
                .andCommandPropertiesDefaults(HystrixCommandProperties.Setter()
                        .withCircuitBreakerEnabled(cbConfig.circuitBreakerEnabled)
                        .withExecutionIsolationThreadTimeoutInMilliseconds(cbConfig.timeoutInMilliseconds)
                        .withCircuitBreakerRequestVolumeThreshold(cbConfig.requestVolumeThreshold)
                        .withCircuitBreakerErrorThresholdPercentage(cbConfig.errorThresholdPercentage)
                        .withCircuitBreakerSleepWindowInMilliseconds(cbConfig.sleepWindowInMilliseconds)));
        this.paymentService = paymentService;
        this.payment = payment;
    }

    @Override
    protected PaymentResult run() {
        return paymentService(payment);
    }
}
```

Resilience With Hystrix



```java
try {
    PaymentCommand paymentCommand = new PaymentCommand(paymentService, payment);

    // Sync execution
    PaymentResult result = paymentCommand.execute();

    // Async execution
    Future<PaymentResult> asyncResult = paymentCommand.queue();
    PaymentResult = asyncResult.get();

    // Reactive execution
    Observable<PaymentResult> observable = paymentCommand.observe();
    observable.subscribe(new Action1<PaymentResult>() {
        @Override
        public void call(PaymentResult v) {
        }
    });

} catch (HystrixRuntimeException e) {
    ...
}
```

Conclusions and Lessons Learned

- Design your application modular (either in OSGi or not)
- Care about decoupling between modules, high cohesion inside the module and modules dependencies
- Continuously refactor your modules to achive optimal boundaries
- Stay on single process at the beginning, split application into different processes only if it is required and brings benefits
- Define your remote and async APIs carefully, design remote calls for failure

OSGi Critic and Myths

OSGi is complex: in understanding, in build, in deployment and in debugging and has poor tooling support

OSGi Alliance

The most understandable specification in the world (inclusive HTTP, ConfigAdmin, DS, RSA, JTA, JMX, JPA)

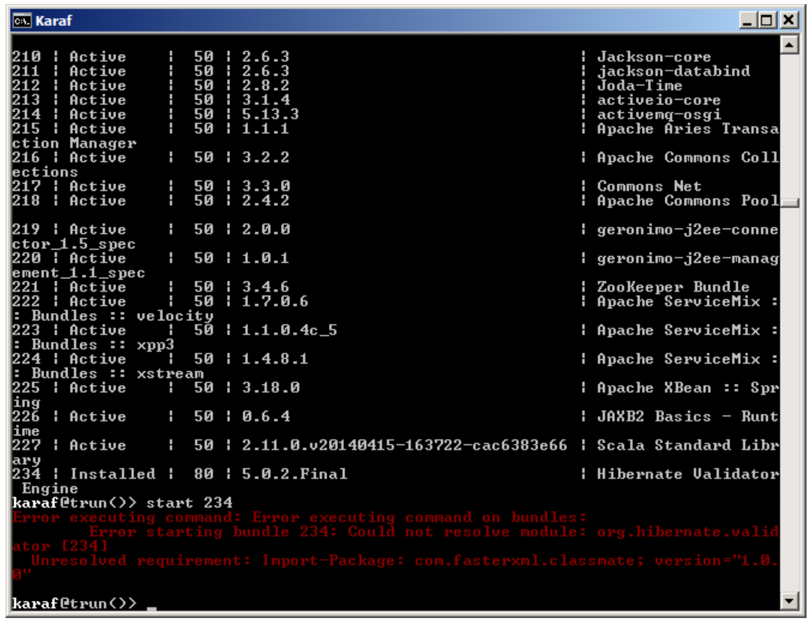
APACHE felix

```
<packaging>bundle</packaging>
…
<plugin>
    <groupId>org.apache.felix</groupId>
    <artifactId>maven-bundle-plugin</artifactId>
</plugin>
```

Karaf

```
Features, configuration, security, console
```

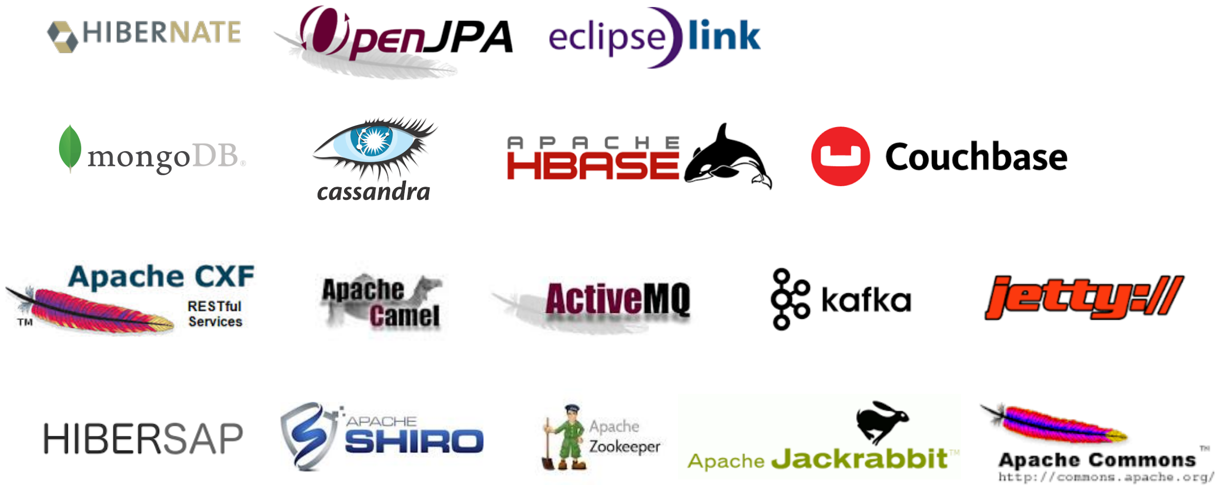
OSGi Critic and Myths



```
210 ¦ Active    ¦  50 ¦ 2.6.3                              ¦ Jackson-core
211 ¦ Active    ¦  50 ¦ 2.6.3                              ¦ jackson-databind
212 ¦ Active    ¦  50 ¦ 2.8.2                              ¦ Joda-Time
213 ¦ Active    ¦  50 ¦ 3.1.4                              ¦ activeio-core
214 ¦ Active    ¦  50 ¦ 5.13.3                             ¦ activemq-osgi
215 ¦ Active    ¦  50 ¦ 1.1.1                              ¦ Apache Aries Transa
ction Manager
216 ¦ Active    ¦  50 ¦ 3.2.2                              ¦ Apache Commons Coll
ections
217 ¦ Active    ¦  50 ¦ 3.3.0                              ¦ Commons Net
218 ¦ Active    ¦  50 ¦ 2.4.2                              ¦ Apache Commons Pool

219 ¦ Active    ¦  50 ¦ 2.0.0                              ¦ geronimo-j2ee-conne
ctor_1.5_spec
220 ¦ Active    ¦  50 ¦ 1.0.1                              ¦ geronimo-j2ee-manag
ement_1.1_spec
221 ¦ Active    ¦  50 ¦ 3.4.6                              ¦ ZooKeeper Bundle
222 ¦ Active    ¦  50 ¦ 1.7.0.6                            ¦ Apache ServiceMix :
: Bundles :: velocity
223 ¦ Active    ¦  50 ¦ 1.1.0.4c_5                         ¦ Apache ServiceMix :
: Bundles :: xpp3
224 ¦ Active    ¦  50 ¦ 1.4.8.1                            ¦ Apache ServiceMix :
: Bundles :: xstream
225 ¦ Active    ¦  50 ¦ 3.18.0                             ¦ Apache XBean :: Spr
ing
226 ¦ Active    ¦  50 ¦ 0.6.4                              ¦ JAXB2 Basics - Runt
ime
227 ¦ Active    ¦  50 ¦ 2.11.0.v20140415-163722-cac6383e66 ¦ Scala Standard Libr
ary
234 ¦ Installed ¦  80 ¦ 5.0.2.Final                        ¦ Hibernate Validator
 Engine
karaf@trun()> start 234
Error executing command: Error executing command on bundles:
        Error starting bundle 234: Could not resolve module: org.hibernate.valid
ator [234]
  Unresolved requirement: Import-Package: com.fasterxml.classmate; version="1.0.
0"

karaf@trun()> _
```

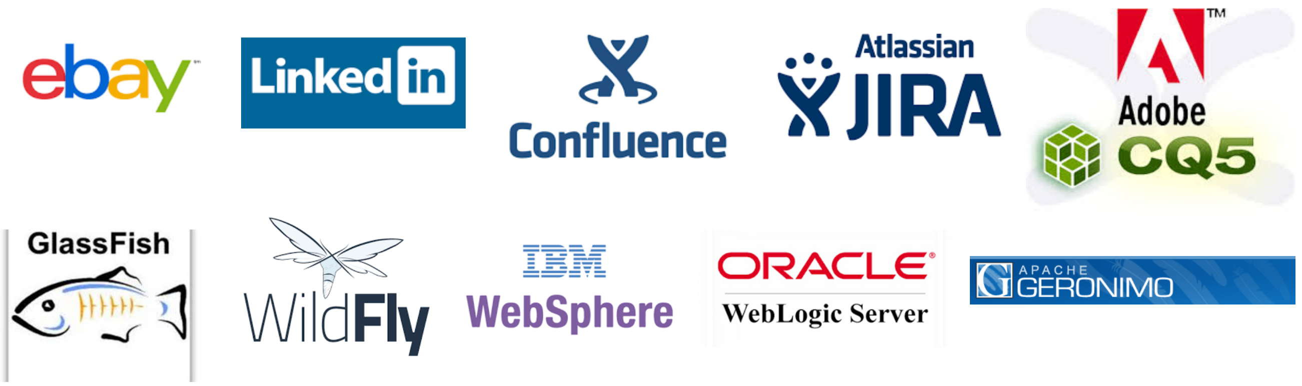
OSGi Critic and Myths

OSGi is not supported by frameworks and libraries



OSGi Critic and Myths

OSGi is not supported by frameworks and libraries



OSGi Critic and Myths

The most important OSGi feature is hot updates: install, delete or replace the bundle on the fly

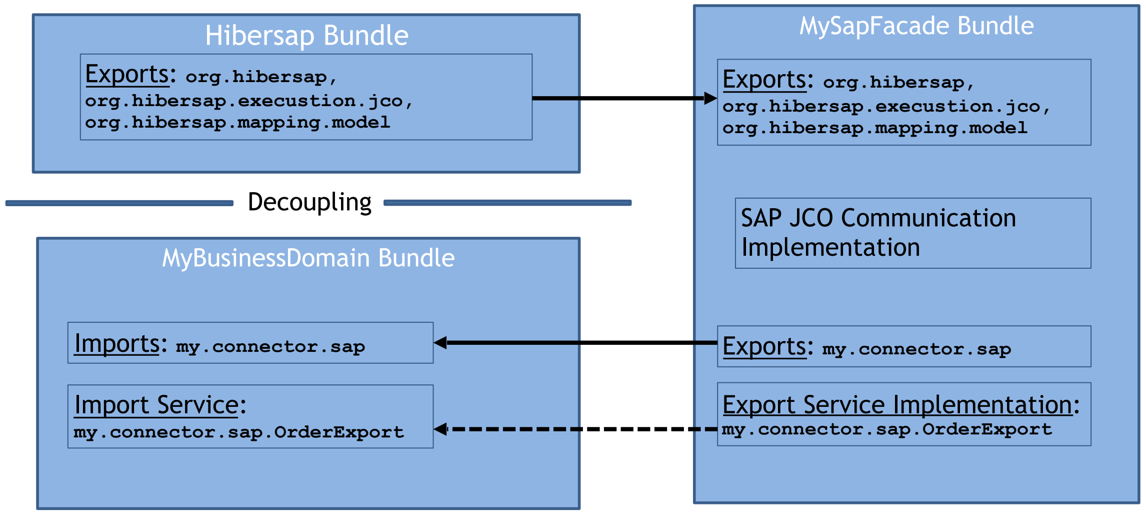
Yes, OSGi is designed for updates without restarting the whole application, but:

1. Normally it is safer to restart the whole Container to have reproducible state in production
2. Hot deployment is not a free lunch: application have to be designed and tested for that
3. The main OSGi gain is not a hot deployment, but clean modular application design, isolation and decoupling of modules. Hot deployment is more derivative feature
4. Can be useful in developer environment, special use cases, partly restarts

REST Communication in OSGi

- Consider REST Architectural Style principles (resources design, verbs contracts, response codes, statelessness)
- Reuse CXF providers, features and interceptors (logging, security)
- Customize (if necessary) through own JAX-RS Filters and Interceptors, MessageBodyReaders and Writers, ParamConverters, CXF Interceptors
- Consider to use Swagger to document and test your API
- Make your external calls resilient

OSGi Decoupling



Hibersap Bundle

Exports: org.hibersap, org.hibersap.execustion.jco, org.hibersap.mapping.model

Decoupling

MyBusinessDomain Bundle

Imports: my.connector.sap

Import Service: my.connector.sap.OrderExport

MySapFacade Bundle

Exports: org.hibersap, org.hibersap.execustion.jco, org.hibersap.mapping.model

SAP JCO Communication Implementation

Exports: my.connector.sap

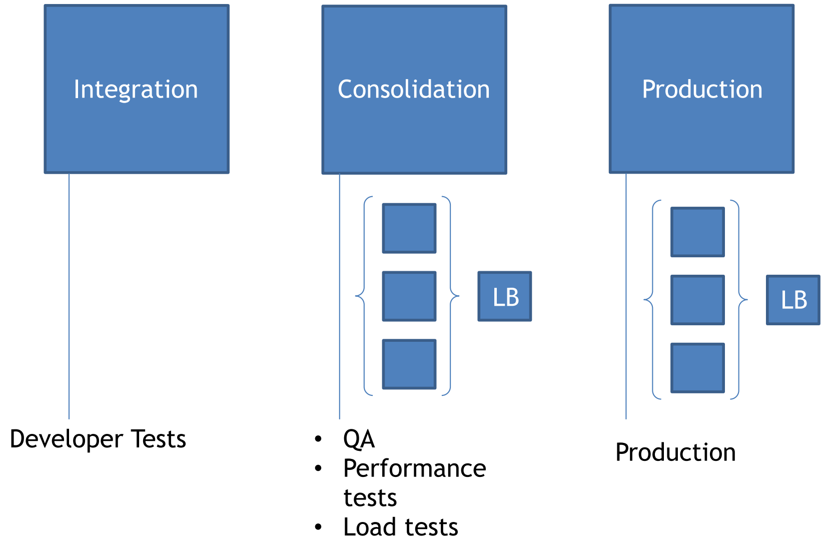
Export Service Implementation: my.connector.sap.OrderExport

Karaf Deployment

Configured as Jenkins JOBs with folwoing steps:

1. Stop Karaf Instance
2. Replace org.apache.karaf.features.cfg
3. Start Karaf Instance
4. Waiting for AvailabilityService

System Environments



Integration

Consolidation

Production

LB

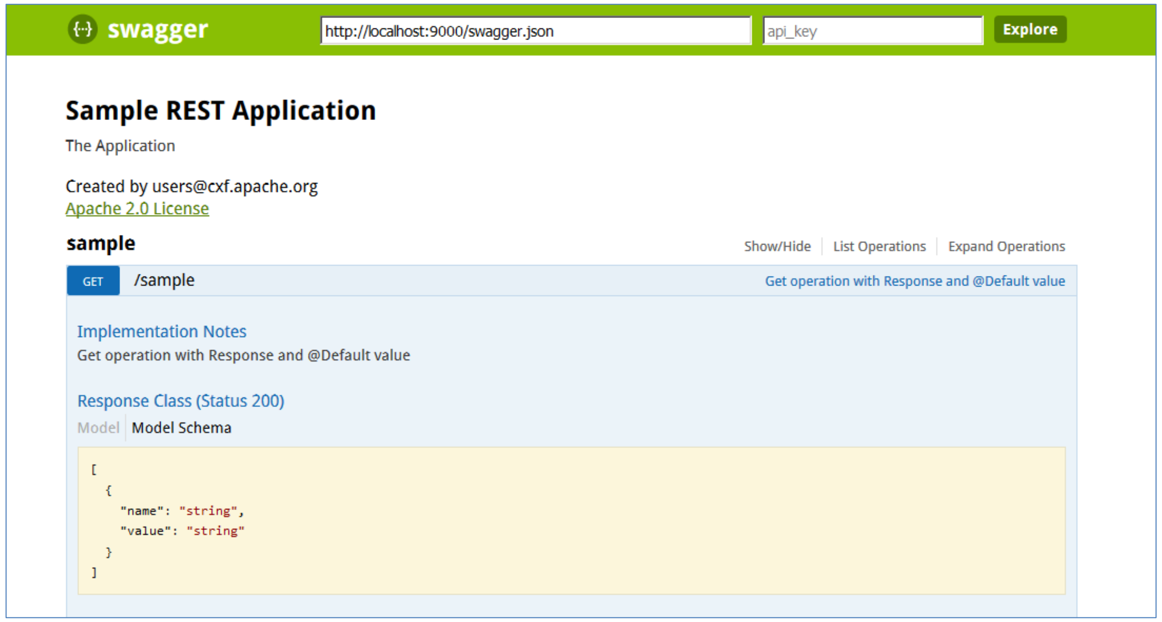
LB

Developer Tests

- QA
- Performance tests
- Load tests

Production

Swagger in JAXRS API



Sample REST Application

The Application

Created by users@cxf.apache.org
Apache 2.0 License

sample

/sample

Get operation with Response and @Default value

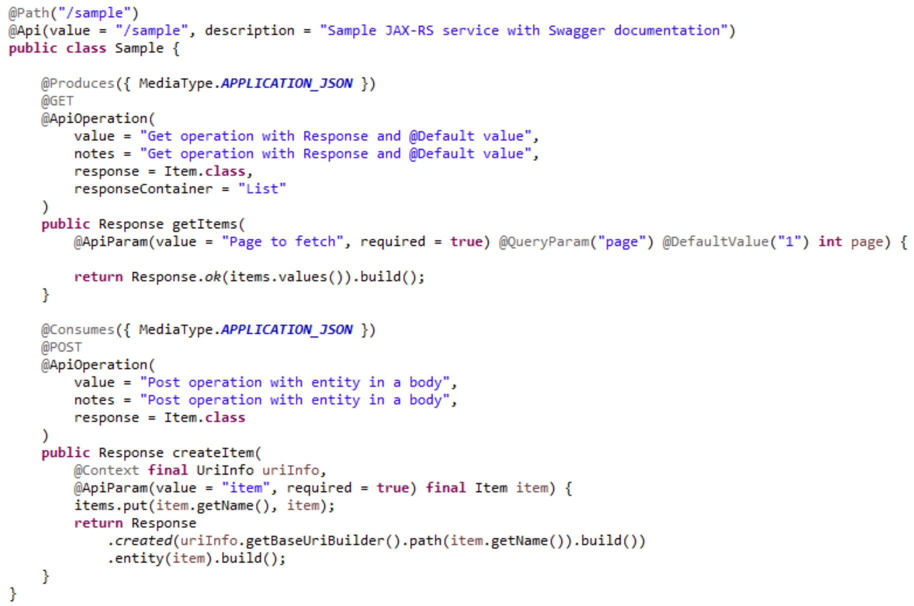
Implementation Notes

Get operation with Response and @Default value

Response Class (Status 200)

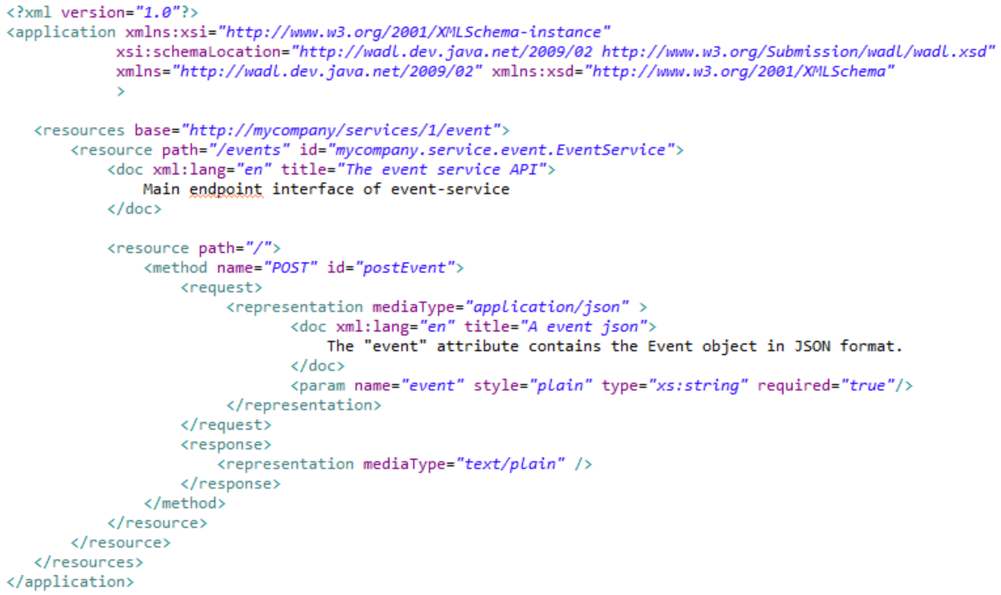
```
[
  {
    "name": "string",
    "value": "string"
  }
]
```

Swagger in JAXRS API: Java First



```java
@Path("/sample")
@Api(value = "/sample", description = "Sample JAX-RS service with Swagger documentation")
public class Sample {

    @Produces({ MediaType.APPLICATION_JSON })
    @GET
    @ApiOperation(
        value = "Get operation with Response and @Default value",
        notes = "Get operation with Response and @Default value",
        response = Item.class,
        responseContainer = "List"
    )
    public Response getItems(
        @ApiParam(value = "Page to fetch", required = true) @QueryParam("page") @DefaultValue("1") int page) {

        return Response.ok(items.values()).build();
    }

    @Consumes({ MediaType.APPLICATION_JSON })
    @POST
    @ApiOperation(
        value = "Post operation with entity in a body",
        notes = "Post operation with entity in a body",
        response = Item.class
    )
    public Response createItem(
        @Context final UriInfo uriInfo,
        @ApiParam(value = "item", required = true) final Item item) {
        items.put(item.getName(), item);
        return Response
            .created(uriInfo.getBaseUriBuilder().path(item.getName()).build())
            .entity(item).build();
    }
}
```

Swagger in JAXRS API: WADL First



```xml
<?xml version="1.0"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://wadl.dev.java.net/2009/02 http://www.w3.org/Submission/wadl/wadl.xsd"
            xmlns="http://wadl.dev.java.net/2009/02" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            >

   <resources base="http://mycompany/services/1/event">
       <resource path="/events" id="mycompany.service.event.EventService">
           <doc xml:lang="en" title="The event service API">
               Main endpoint interface of event-service
           </doc>

           <resource path="/">
               <method name="POST" id="postEvent">
                   <request>
                        <representation mediaType="application/json" >
                               <doc xml:lang="en" title="A event json">
                                   The "event" attribute contains the Event object in JSON format.
                               </doc>
                               <param name="event" style="plain" type="xs:string" required="true"/>
                        </representation>
                   </request>
                   <response>
                       <representation mediaType="text/plain" />
                   </response>
               </method>
           </resource>
       </resource>
   </resources>
</application>
```