OSGi productivity compared ...

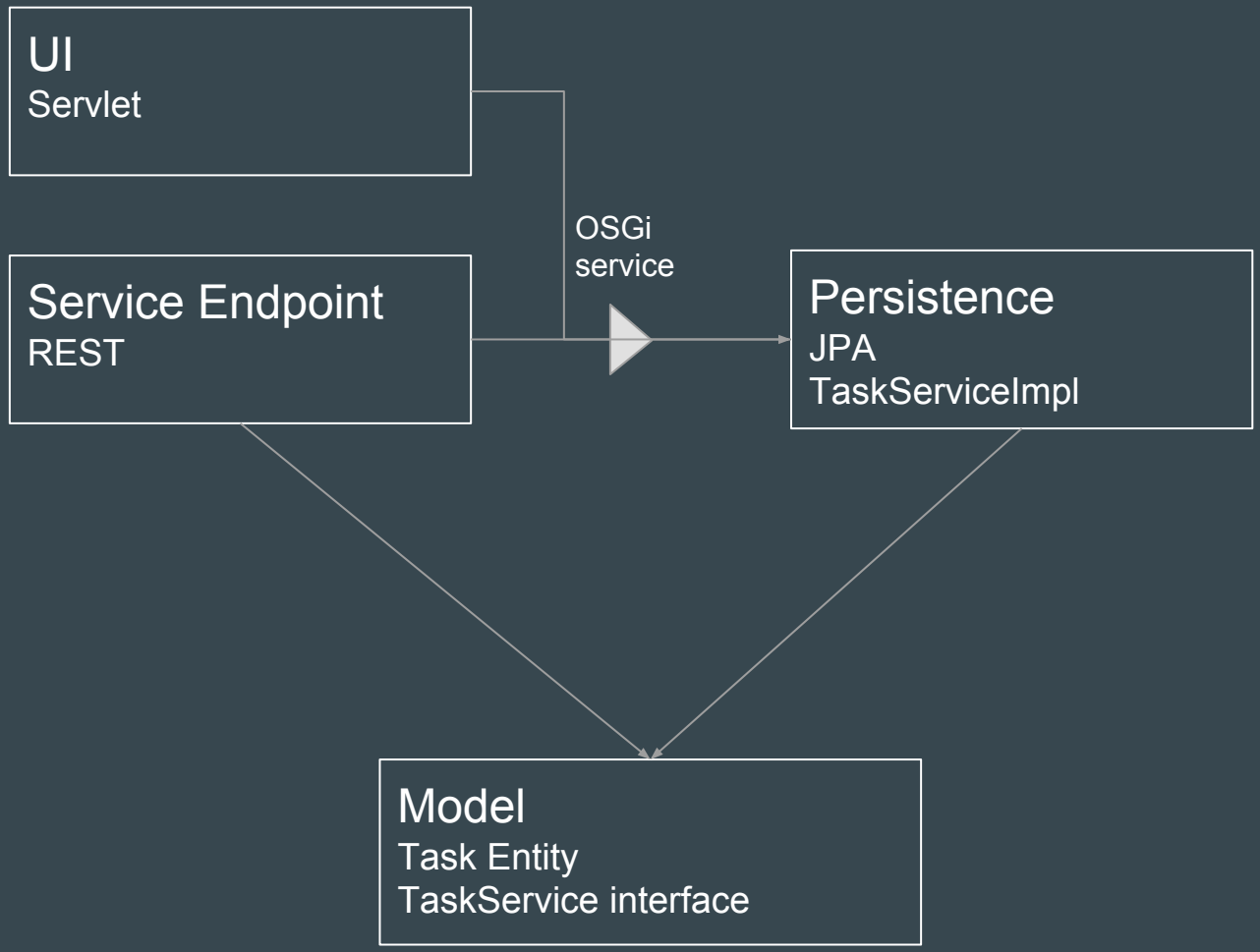
on Apache Karaf

Christian Schneider
Talend

Goals

- Introduce Declarative Services and Blueprint
- Model a simple enterprise application in both frameworks regarding
  - Configuration
  - JPA and Transactions
  - Rest and SOAP
- Packaging / Deployment for Apache Karaf
- Eclipse without PDE
- Recommendations
- Future enhancements

Sample Application Tasklist



Declarative Services (DS)

- Component <-> Class <-> XML descriptor
- Annotations -> XML at build time
- Dynamic Lifecycle

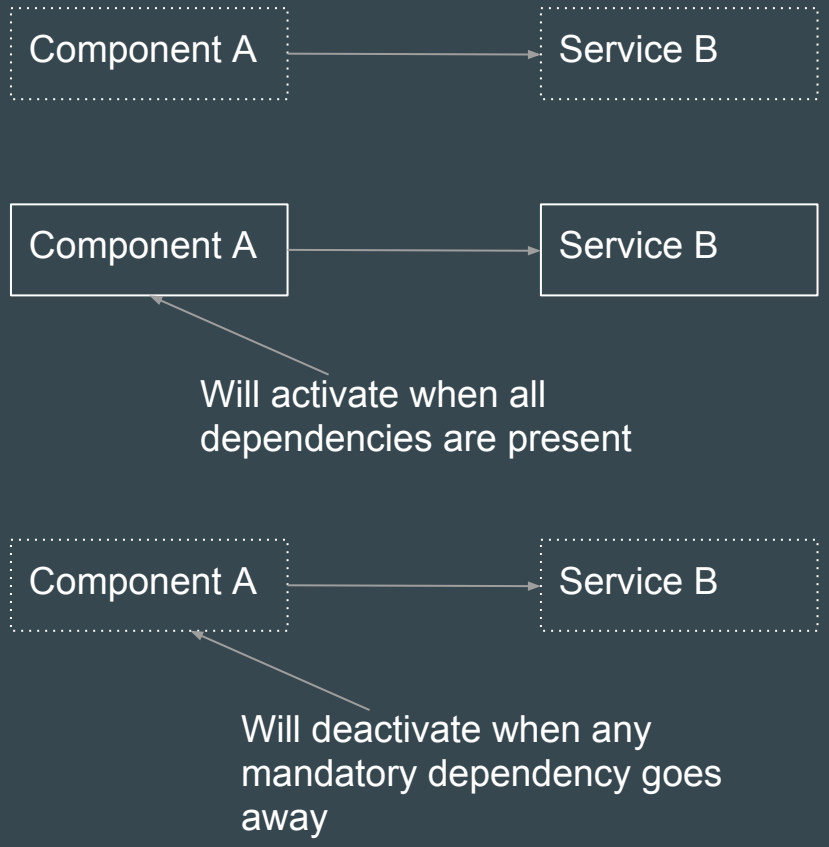
Pro

- Annotations standardized
- Very small at runtime (Just one bundle)

Con

- No support for interceptors
- No extension model

DS Component Lifecycle



DS Example

Create OSGi Service

```
@Component
public class InitHelper {
    TaskService taskService;

    @Activate
    public void addDemoTasks() {
        Task task = new Task(1, "Just a sample task", "Some more info");
        taskService.addTask(task);
    }

    @Reference
    public void setTaskService(TaskService taskService) {
        this.taskService = taskService;
    }
}
```

Called on start of component

Mandatory OSGi service reference

(Aries) Blueprint

- Beans described in a XML Context
- Annotations -> XML at build time
- Config per Context, injected into properties

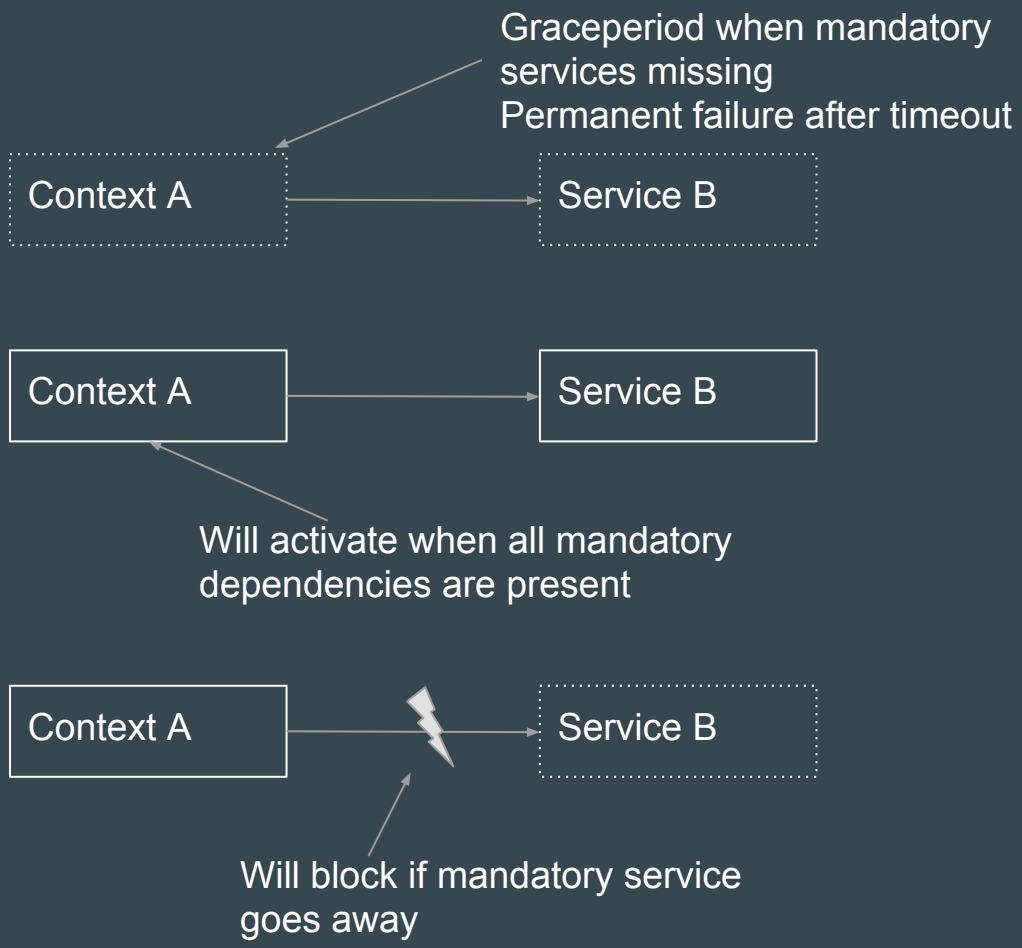
Pro

- Extension model using Namespaces
- Many Extensions (Aries JPA, Aries Transaction, CXF, Camel, Authz)
- Supports internal wiring (DI)
- Supports interceptors (e.g. for transactions)

Con

- No standard blueprint annotations
- Lifecycle per Context
- Service damping

Blueprint Lifecycle



Blueprint Example

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
<bean id="initHelper" class="net.lr.tasklist.persistence.impl.InitHelper" init-method="
addDemoTasks">
    <property name="taskService" ref="taskService"/>
</bean>
<reference id="taskService" interface="net.lr.tasklist.model.TaskService"/>
</blueprint>
```

Create bean

Called on start of bean

Inject another bean

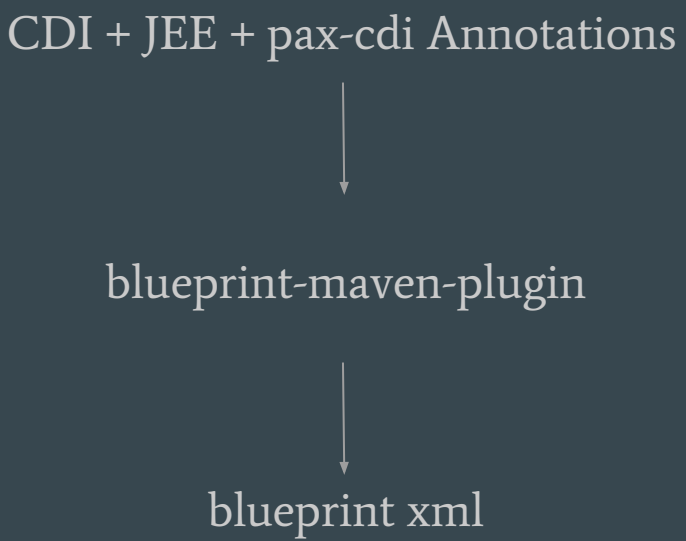
Mandatory OSGi service reference

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0" >
<bean id="taskServiceImpl" class="net.lr.tasklist.persistence.impl.TaskServiceImpl" ext:
field-injection="true"/>
<service ref="taskServiceImpl" interface="net.lr.tasklist.model.TaskService"/>
</blueprint>
```

Publish bean as OSGi service

Blueprint from CDI annotations

- Uses subset of CDI
- Aims for compatibility with real CDI runtime



Blueprint from CDI annotations Example

```
@OsgiServiceProvider(classes = {TaskService.class})
@Singleton
public class TaskServiceImpl implements TaskService {
}

@Singleton
public class InitHelper {

    @OsgiService @Inject TaskService taskService;

    @PostConstruct
    public void addDemoTasks() {
        Task task = new Task(1, "Just a sample task", "Some more info");
        taskService.addTask(task);
    }
}
```

Publish as OSGi service

Create bean

Inject OSGi Service

Inject another bean

Called on start of context

Configuration

Goals

- Configure objects from config admin configurations
- Support configuration changes at runtime
- Ideally support type safety and config validation (meta type spec)

Configuration in DS (1.3)

```
@ObjectClassDefinition(name = "Server Configuration")
@interface ServerConfig {
  String host() default "0.0.0.0";
  int port() default 8080;
  boolean enableSSL() default false;
}

@Component
@Designate(ocd = ServerConfig.class)
public class ServerComponent {
  @Activate
  public void activate(ServerConfig cfg) {
    ServerSocket sock = new ServerSocket();
    sock.bind(new InetSocketAddress(cfg.host(), cfg.port()));
  }
}
```

Meta type definition

link with config metadata

Type safe configuration
called when config changes

Example from http://njbartlett.name/2015/08/17/osgir6-declarative-services.html

Configuration in blueprint

In Apache Karaf config would be in /etc/com.example.config.cfg

```
<cm:property-placeholder persistent-id="com.example.config" update-strategy="reload">
    <cm:default-properties>
        <cm:property name="host" value="0.0.0.0" />
        <cm:property name="port" value="8080" />
    </cm:default-properties>
</cm:property-placeholder>
```

Blueprint context restarted on config change

Default value

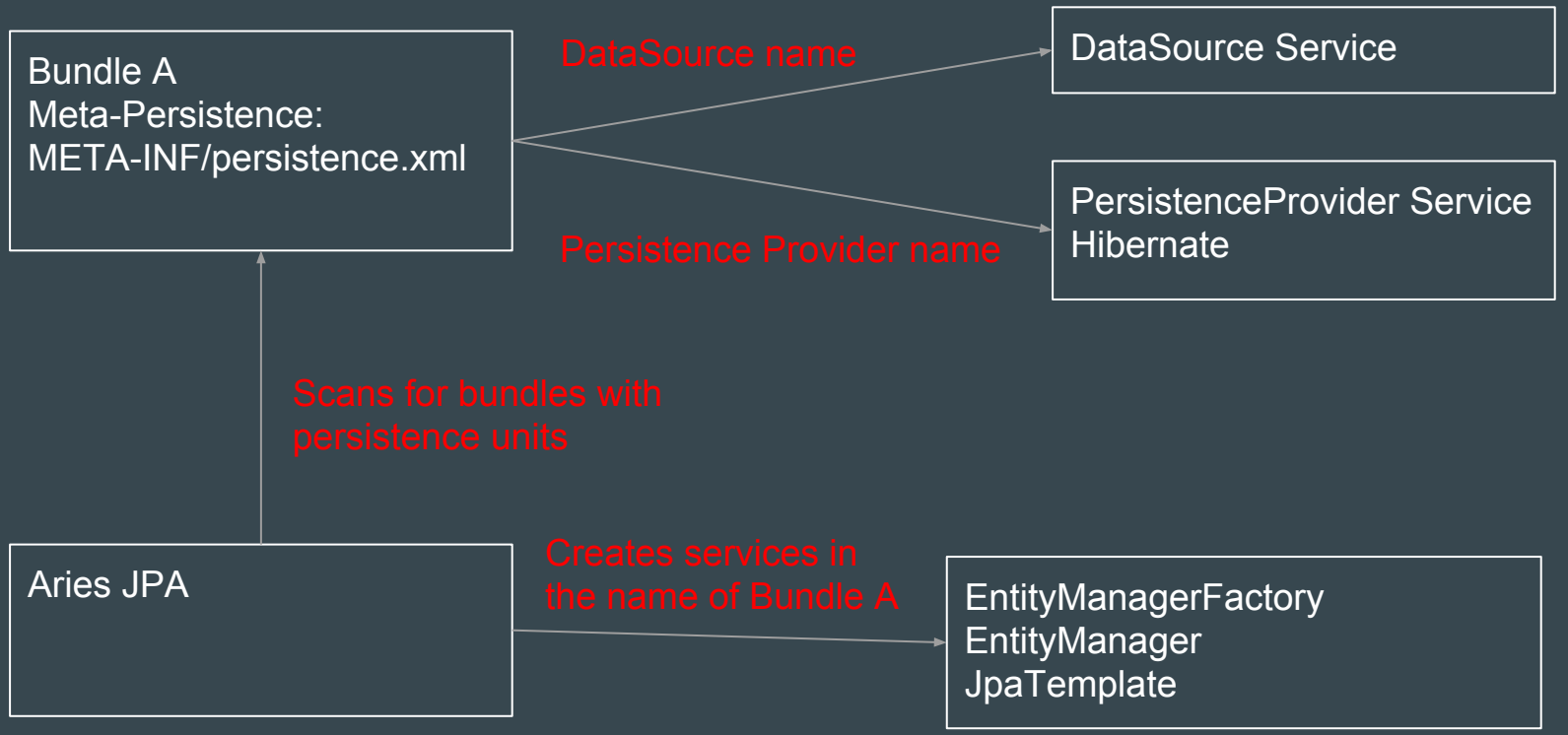
```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0" >
<bean id="taskServiceImpl" class="net.lr.tasklist.persistence.impl.TaskServiceImpl">
    <property name=”host” value=”${host}”/>
    <property name=”port” value=”${port}”/>
</bean>
</blueprint>
```

Inject into property with automatic type conversion

No real meta type support

JPA and Transactions - Persistence Unit

Aries JPA Container implements the OSGi JPA Service Specification.



Closure based transactions for DS or Blueprint

```
@Component
public class TaskServiceImpl implements TaskService {
    private JpaTemplate jpa;

    public Task getTask(Integer id) {
        return jpa.txExpr(TransactionType.Supports, em -> em.find(Task.class, id));
    }

    public void updateTask(Task task) {
        jpa.tx(em -> em.persist(task));
    }

    @Reference(target = "(osgi.unit.name=tasklist)")
    public void setJpaTemplate(JpaTemplate jpa) {
        this.jpa = jpa;
    }
}
```

Declarative transactions for Blueprint

```
@Singleton
@Transactional
public class TaskServiceImpl implements TaskService {
    @PersistenceContext(unitName = "tasklist")
    EntityManager em;

    @Transactional(TxType.SUPPORTS)
    public Task getTask(Integer id) {
        return em.find(Task.class, id);
    }

    public void updateTask(Task task) {
        em.persist(task);
    }
}
```

All methods run in transaction by default

Managed EM injected

No Transaction required here

REST and SOAP Services

Use OSGi remote services (Apache CXF DOSGi or Eclipse ECF)

- Annotate Class with @WebService or Rest annotations
- Export as OSGi service with special properties

JAX-RS connector by Eclipsesource (Holger Staudacher)

- Scans for OSGi services of JAX-RS classes
- Provides security and different serializations
- Easy to install in Apache Karaf with a feature

REST and SOAP Service in blueprint using Apache CXF

- Custom Aries blueprint namespace
- No pure Annotation based configs
- Several protocols http, servlet, jms, local, ...
- Extensive Security Features
- Enterprise level logging to Elasticsearch leveraging Karaf Decanter

REST Service in Blueprint using CXF

```
<bean id="personServiceImpl" class="net.lr.tutorial.karaf.cxf.personrest.impl.PersonServiceImpl"/>

<jaxrs:server address="/person" id="personService">
        <jaxrs:serviceBeans>
            <ref component-id="personServiceImpl" />
        </jaxrs:serviceBeans>
        <jaxrs:features>
            <cxf:logging />
        </jaxrs:features>
</jaxrs:server>
```

Class with JAXRS annoations

Additional CXF features

SOAP Service in Blueprint using CXF

```
<bean id="personServiceImpl" class="net.lr.tutorial.karaf.cxf.personservice.impl.PersonServiceImpl"/>
<jaxws:endpoint implementor="#personServiceImpl" address="/personService" />
```

Packaging for Apache Karaf

Pro

- User features can depend on other features to ease deployment
- Karaf provides features for most enterprise needs like (JPA, Transaction, CXF, Camel, ActiveMQ, Hibernate, OpenJPA, Eclipselink, Elastic Search)
- Feature validation at build time (Since Karaf 4)
- References to bundles and feature repos as maven coordinates
- Can leverage all bundles in any maven repo

Con

- Karaf feature creation still largely manual

Karaf Feature Repo Example

```
<features name="tasklist-cdi-1.0.0-SNAPSHOT" xmlns="http://karaf.apache.org/xmlns/features/v1.3.0">
  <repository>mvn:org.apache.aries.jpa/jpa-features/2.2.0-SNAPSHOT/xml/features</repository>
  <repository>mvn:org.ops4j.pax.jdbc/pax-jdbc-features/0.7.0/xml/features</repository>

   <feature name="example-tasklist-cdi-persistence" version="${pom.version}">
      <feature>pax-jdbc-h2</feature>
      <feature>pax-jdbc-config</feature>
      <feature>pax-jdbc-pool-dbcp2</feature>
      <feature>jndi</feature>
      <feature>transaction</feature>
      <feature version="[2.2, 2.3)">jpa</feature>
      <feature version="[4.3, 4.4)">hibernate</feature>
      <bundle>mvn:net.lr.tasklist.cdi/tasklist-model/${pom.version}</bundle>
      <bundle>mvn:net.lr.tasklist.cdi/tasklist-persistence/${pom.version}</bundle>
   </feature>
```

Reference other feature repos

Depend on features

Set version ranges for features

Define bundles to install

Deployment on Apache Karaf

Deployment options

- Deployment by hand using Console commands
- Remote triggered deployment using JMX
- Create self contained Archive from Karaf + Features + Bundles using karaf-maven-plugin
- Create lightweight deployment using Karaf Profiles

IDE: Plain Eclipse + m2e (No PDE)

Pros

- Familiar maven builds
- Can check out individual maven projects
- No target platform or similar
- maven-bundle-plugin does most of the work automatically
- Debugging in running Karaf quite simple

Cons

- No OSGi specific tooling
- Configuring deployments (Karaf features) mainly manual
- Pax Exam OSGi integration tests difficult to set up
- Needs remote debugging

Demo

Recommendations DS

- If OSGi dynamics needed
- Nicer config support
- If frameworks work with the stock DS features
- No special integration with most big open source libraries and frameworks

Recommendations (Aries) Blueprint

- If you need other Apache Frameworks like Camel, CXF
- Better JPA / XA support
- Annotations currently support only subset of CDI

Future Developments

- Extension model for Blueprint generation from Annotated Code
- Creation of OSGi indexes from maven dependencies
- Better bndtools maven support in Version 3.1.0
- Karaf features based on requirements like in bndtools
- Karaf boot: Simplified development / test / deployment lifecycle like in Spring boot

Questions ?

github.com/cschneider/Karaf-Tutorial

Examples tasklist-ds and tasklist-blueprint-cdi
use branch jpa-2.1.0 for newest changes

Backup

CDI (PAX-CDI)

- Mark bundle as CDI bundle by placing empty beans.xml in META-INF
- Use CDI + JEE + pax-cdi Annotations
- Annotations interpreted at runtime
- Uses pax-cdi + Openwebbeans or Weld
- Quite comprehensive support for CDI
- JPA support planned using Apache Deltaspike (not yet working)
- JSF not yet working
- Lifecycle per context (bundle)
- Service Damping like in Blueprint
- Not yet fully mature

CDI Example

Basically same as in Blueprint from CDI example.

Recommendations PAX CDI

Pro

- Leverage JEE knowledge

Con

- Not fully mature

Configuration in CDI

- No out of the box solution
- See https://java.net/jira/browse/GLASSFISH-15364 - unresolved since 2010
- Typically implemented using

```
@Produces
MyConfig loadConfig() {
// Read config explicitly
}
```

Deployment with bndtools

- Define launch configs using requirements and OSGi bundle resolver
- Create self contained archives from bndtools and gradle

IDE: Bndtools

Pros

- Deployments easy to define if OSGi indexes exist for the needed bundles
- Fast incremental Builds
- Tests and debugging easy. No remote debug

Cons

- OSGi ready repositories only cover very small subset of bundles in maven central
- Very limited maven integration (Version 3.0.0)