**Deployment and Integration**

* **­­­­­JAX-RS Web Application can be deployed**

1. within a standalone servlet container, like

* Apache Tomcat
* Jetty
* JBossWeb, etc.

1. within the servlet container of Application Server, like

* JBoss
* WildFly
* Weblogic
* Websphere or
* Glassfish, etc.,
* How JAX-RS Web Applications is deployed within a servlet container varies between

1. JAX-RS-Aware Servlet Containers (like any JavaEE supported Application Servers or standalone Serrvlet 3.x containers like Tomcat)
2. Older JAXRS-Unaware Servlet Containers.

**Deployment within JAX-RS-Aware Containsers**

* Application servers that are certified under Java EE 6 required to have built-in support for JAXRS 1.1
* To deploy JAX-RS application in JAX-RS-Aware Containers, we should have
* at least one Java Class which extends “javax.es.rs.core.Appplication” and
* annotated with “@ApplicationPath” them JAX-RS-aware container will automatically deploy that Application
* For Example:

package com.mycomp.myapp;

import javax.ws.rs.ApplicationPath;

import javax.ws.rs.core.Application;

@ApplicationPath(“/services”)

public class MyRestApplication extends Application

{

….

}

* The @ApplicationPath annotation will set up a base path to whatever the WAR’s context root is. If WAR file name is “mywebapp” then URL will looks like,

http://<Domain-name>:<port>/<app-name>/services/<paths-of-root\_Resource\_Class\_and\_Method>

**javax.ws.rs.core.Application**

* **­­­**It’s a Concreate Class present in JAX-RS API
* The Application class is the only portable way of telling container that which web services (@Path annotated classes) we want to be deployed.
* This Class List “Classes and Objects” that JAX-RS is supposed to deploy.
* Application Class has below 2 methods & hence subclass of this class can override any/one/all of these methods

1. public Set<Class<?>> getClasses()
2. public Set<object> getSingletonss()
3. public Map<String, Object> getProperties()

* The getClasses() method returns a list of JAX-RS web service and provider classes. These Classes will get instantiated once request comes & garbage collected once response is given (non-singleton in nature). These Classes SHOULD have public default constructor. In this case, it’s a Container responsibility to Create an instance of resource class.
* The getSingletons() method return a list of pre-allocated JAX-RS web services and providers. These Classes will get instantiated during tshe server start-up & garbage collected during the server shut down. Hence they are “singleton” in nature. These Classes need not to have public default constructor. In this case, we as a developer, are responsible for creating these objects.
* The getProperties() method returns a map of custom application-wide properties
* The JAX-RS runtime will iterate thorugh the list of objects and register them internally.
* Along with creating a class which extends Application class we still neeed at least an empty web.xml file:

<?xml version = “1.0” encoding=”UTF-8”?>

<web-app …. 3.0>

</web-app>

**NOTE:**

* We can fully leverage “servlet class scanning abilities” of application server incase if we don’t override get Classes() and getSingletons() OR both return an empty set.

For example:-

@ApplicationPath(“/services”)

public class MyRestApplication extends Application{

//Empty Class

}

* When scanning the application server will look within WEB-INF/Classes and any JAR file within the WEB-INF/lib directory
* It will add any class annotated with @Path to the list of things that need to be deployed and registered with the JAX-RS runtime.
* You can also deploy as many Application classes as you want in WAR
* The scanner will also ignore any Application classes not annotated with @ApplicationPath
* You can also override the @ApplicationPath annotation via a simple servlet mapping within web.xml:

<servlet-mapping>

<servlet-name>

com.mycomp.myapp.MyRestApplication

</servlet-name>

<url-pattern>/\*</url-pattern>

<servlet-mapping>

* The servlet-name is the fully qualified class name of your application class.

In this case, we can also omit the @ApplicationPath annotation entirely.