



[Home](#) [About](#) [Search](#) [Changes](#) [Unused Pages](#) [Undefined Pages](#) [Index](#) [License](#)

[Edit this page](#)

# JBossESBDeploymentStrategies

Your trail:

## **JBossESB - JBossESBDocumentation - JBossESB-4.2 Deployment Strategies**

### **Introduction**

The JBossESB-4.2 code is packaged up and deployed as a Service ARchive (SAR) called jbossesb.sar. This service archive contains the core code needed for the message transport protocols and the various listener capabilities.

To make the ESB 'do' something, you can deploy services onto the ESB. The ESB is packaged with base services that ship with the ESB. A service consist of action code + configuration. Both of which should be deployed in an .esb archive. You can deploy as many .esb archives as you please. You can specify the deployment order of the archives using the deployment.xml. Typically you would deploy the .esb archive to the 'deploy' directory.

### **The .esb archive**

The idea behind the .esb archive is that it is a deployable service unit. It should enable you to move a service between servers simply by moving the .esb archive.

The standard layout of an esb archive looks like

META-INF		
	MANIFEST.MF	
	jboss-esb.xml	
	deployment.xml	
classes		
jars		
queue-service.xml		

where,

- the jboss-esb.xml contains the service configuration (listener and actions), as well as provider configuration.
- the deployment.xml is optional, but can be used for 2 reasons:

- make this .esb archive depend on other archives, to specify classloading order.
- make the deployment of this .esb archive scoped.
- you custom action classes.
- additional jar archives your actions depend on.
- queue-service.xml, if your 'provider' section references queues or topics you can deploy their configuration in the .esb archive. Note that any other way to deploy these queues is fine too, we just recommend this to keep your deployments as self-contained as possible and therefor to keep dependency management simple.

JBossESB ships with a number of standard service archives:

- [jbossesb.esb](#) - internal services like the [DeadLetterService](#),
- [jbrules.esb](#) - needed for rules evaluation using [JBossRules](#), for services like the [Content-Based Router](#),
- [jbpm.esb](#) - needed for default provider of Business Process Management [jBPM](#), and
- [smooks.esb](#) - default message transformation engine [Smooks](#).

These services are deployed by default, but you should be able to remove them if you don't need these service deployments.

## Hot Redeployment

---

The jbossesb-server will hot redeploy an .esb archive when the timestamp on the jar file changes, or -if the archive is exploded- when the timestamp on the META-INF/jboss-esb.xml changes .

## Examples

---

### 1. jbossesb.esb

The jbossesb.esb service archive contains internal services. For now it actually deploys just one service, like the [DeadLetterService](#). Most service archives depend on this service to be there. Since there is not code dependency "being there" really only means that it needs to be deployed at least once in your distributed environment. If you want to make sure this service archive is loaded before your service add the following section to your deployment.xml:

```
<jbossesb-deployment>
  <depends>jboss.esb:deployment=jbossesb.esb</depends>
</jbossesb-deployment>
```

which will deploy the jbossesb.esb before it deploys myproject.esb in on the current jvm that is. So you have to have to make sure the jbossesb.esb is deployed to the node you are currently deploying the myproject.esb to.

### 2. jbrules.esb

By default the [Content-based Router](#) uses JBoss Rules to evaluate routing rules. So you need to deploy JBoss Rules to the ESB.

- The recommended way to deploy it is to deploy the jbrules.esb archive, which actually is deployed by default (so just check for it in the deploy directory). Next you can deploy your own 'myproject.esb' archive, and make sure to make reference the jbrules.esb in the deployer.xml by adding the following section:

```
<jbossesb-deployment>
  <depends>jboss.esb:deployment=jbrules.esb</depends>
</jbossesb-deployment>
```

This will make sure the jbrules.esb gets deployed *before* your myproject.esb, and prevents class-not-found exceptions. If you want to move your myproject.esb to another server, make sure this server also has the jbrules.esb deployed.

Note that if you need to scope your myproject.esb, then you can add the following section to your deployment.xml:

```
<loader-repository>
  org.jboss.soa.esb:loader=myproject-scope
```

```
<loader-repository-config>
  java2ParentDelegaton=false
</loader-repository-config>
</loader-repository>
```

- Alternative 1 - self-contained deployment. Add your rules based actions and configurations to the jbrules.esb archive. This may make sense if you want to keep the number of .esb archive to a minimum, or if you want to make your deployment self-contained. You should now be able to move the jbrules.esb archive between servers without dependency issues. Basically you are creating a rules-based services archive. Using the per-archive scoping you can sun multiple versions of jbrules simultaneously.
- Alternative 2 - appserver deployment. Deploy JBoss Rules to the 'AppServer' by adding all the jars in jbrules/lib to the jbossesb.sar, or deploying it any other way supported by [JBossRules](#). The advantage of that is that you now only have to worry about deploying your .esb archive(s). The downside is that it gets harder to keep all the jar dependencies straight, and it becomes harder to move services around in your farm, or to specialize a [JBossESB](#) node.

See also the [simple\\_cbr](#) and the [fun\\_cbr](#).

### 3. jbpm.esb

If you want Business Process Management support, either for service orchestration or for human workflow support.

- The recommended deployment is to have the jbpm.esb archive deployed, and to make your myproject.esb depend on jbpm.esb by adding the following section to your deployment.xml:

```
<jbossesb-deployment>
  <depends>jboss.esb:deployment=jbpm.esb</depends>
</jbossesb-deployment>
```

- Alternative 1 - self-contained deployment: Same pros and cons as under the Content-Based Routing example, but it should be noted that packaging up the jbpm jars and config in each of your scoped .esb archives allows you to run multiple jbpm engines pointing to *\*different\** data stores.
- Alternative 2 - appserver deployment: Same pros and cons as under the Content-Based Routing example, resulting in one jbpm engine per jvm, which will by default all interact with the same data store.

See also the [jbpm\\_simple\\_1](#) and the [bpm\\_orchestration1](#).

---

[Go to top](#) [Edit this page](#) [More info...](#) [Attach file...](#)

*This page last changed on 29-May-2007 17:04:23 EDT by kurt.stam@jboss.com.*

