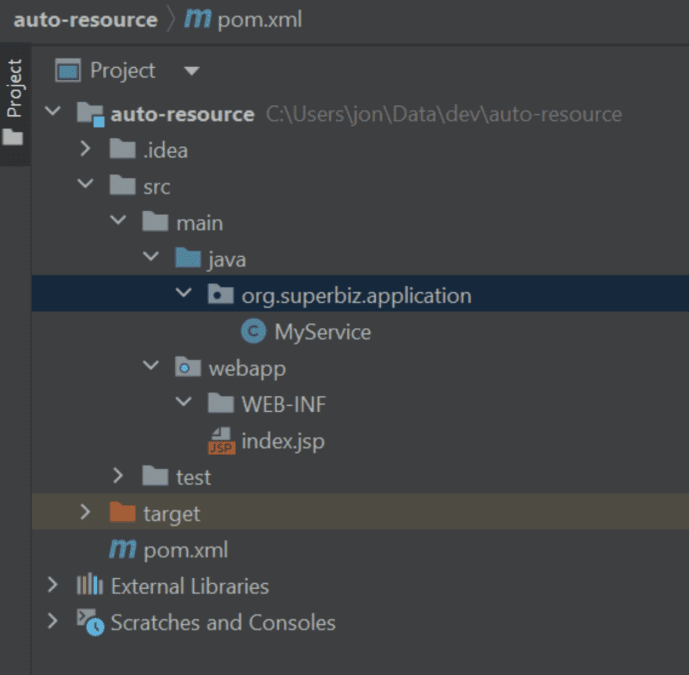
Demystifying Default Resources in Apache TomEE

The Problem

One of the design philosophies of Apache TomEE is to adapt the server to fit the user, rather than the other way around.  It can be incredibly frustrating to develop your application and then have the application server reject the deployment because a resource or another setting isn’t configured quite right. Apache TomEE will attempt to automatically create resources and containers with a reasonable set of defaults so you can concentrate on developing your application, rather than fighting with the server. In this article, I’ll set out how these defaults are applied, and how you can customize them.

If you consider the following simple project:



**MyService**.java

@Singleton

**public** **class** **MyService** {

    @Resource

**private** **ConnectionFactory** cf;

**public** void **doSomething**() {

        // don't actually use the connection factory

**System**.out.**println**("Hello, this is a message from the MyService.doSomething()");

    }

}

The source code for this project is available here: <https://github.com/tomitribe/tomee-activemq-vm-defaults/tree/main/sample>

If we run this application, either by deploying it to TomEE, or running the project with mvn clean install tomee:run, we see an embedded ActiveMQ Broker start, and listen on port 61616. The TCP port usage may conflict if you are also running a standalone ActiveMQ broker on the same machine.

Note that we didn’t configure an ActiveMQ broker, or a connection factory. None of our code actually sends or receives JMS messages – so what’s happening here? Let’s dig in, and have a look.

What happens at deploy time

There are 2 key things about this application:

1. It defines a Singleton Session Bean – the container must create an instance of this bean and make it available to use
2. The bean requires a ConnectionFactory resource – the container is required to identify this resource and inject it into the bean instance

This all takes place at deployment time. The container scans for the annotations, reads deployment descriptors and builds a plan of what to deploy, validates it, and creates the necessary objects before the application is then started and available for use.

But I didn’t configure a JMS Broker!

When an application requires a resource, but the resource isn’t explicitly configured, there are a couple of options:

* Fail the deployment

OR

* Attempt to create a suitable resource

By default, TomEE will attempt to create a suitable resource. This is in line with the philosophy I mentioned – adapting to fit the user.

As the MyService bean in the sample project requires a ConnectionFactory with it’s @Resource annotation on the cf field, TomEE will attempt to create a default ConnectionFactory. This in turn requires a ResourceAdapter, for which a default object will also be created.

So, where do the defaults come from?

This feature can often feel a bit “magical” for developers that are new to using TomEE. The good news is that the defaults for objects that the server automatically creates are quite well defined and documented.

The defaults themselves come from META-INF/org.apache.tomee/service-jar.xml from the classpath, which itself extends META-INF/org.apache.openejb/service-jar.xml.

If you take a look at the org.apache.openejb/service-jar.xml file here: <https://github.com/apache/tomee/blob/main/container/openejb-core/src/main/resources/META-INF/org.apache.openejb/service-jar.xml>, you’ll see the defaults for the resource adapter and connection factory:

  <ServiceProvider

          id="Default JMS Resource Adapter"

          service="Resource"

          types="ActiveMQResourceAdapter"

          class-name="org.apache.openejb.resource.activemq.ActiveMQResourceAdapter">

    # Broker configuration URI as defined by ActiveMQ

    # see http://activemq.apache.org/broker-configuration-uri.html

    # BrokerXmlConfig xbean:file:conf/activemq.xml - Requires xbean-spring.jar and dependencies

    BrokerXmlConfig broker:(tcp://localhost:61616)?useJmx=false

    # Broker address

    ServerUrl vm://localhost?waitForStart=20000&amp;async=true

    # DataSource for persistence messages

    DataSource Default Unmanaged JDBC Database

    # How long to wait for broker startup

    StartupTimeout 10 seconds

  </ServiceProvider>

  <ServiceProvider

          id="Default JMS Connection Factory"

          service="Resource"

          types="jakarta.jms.ConnectionFactory, jakarta.jms.QueueConnectionFactory, jakarta.jms.TopicConnectionFactory, QueueConnectionFactory, TopicConnectionFactory"

          class-name="org.apache.openejb.resource.activemq.jms2.TomEEManagedConnectionFactory">

    ResourceAdapter Default JMS Resource Adapter

    # Specifies if the connection is enrolled in global transaction

    # allowed values: xa, local or none

    TransactionSupport xa

    # Maximum number of physical connection to the ActiveMQ broker

    PoolMaxSize 10

    # Minimum number of physical connection to the ActiveMQ broker

    PoolMinSize 0

    # Maximum amount of time to wait for a connection

    ConnectionMaxWaitTime 5 seconds

    # Maximum amount of time a connection can be idle before being reclaimed

    ConnectionMaxIdleTime 15 Minutes

  </ServiceProvider>

Can I apply my own defaults?

Absolutely! Start off from this file: <https://github.com/apache/tomee/blob/main/tomee/tomee-webapp/src/main/resources/META-INF/org.apache.tomee/service-jar.xml>

You will need to decide a provider name – if for example, you wanted a provider name of “org.superbiz”, create a new jar file, and place the service-jar.xml file you just copied in META-INF/org.superbiz/service-jar.xml.

Make any changes you wish to the service-jar.xml, and then install your jar file to the TomEE lib directory. Finally set the system property openejb.provider.default to be the provider name you chose (org.superbiz in this case).

The full source code for this article, including an example of customizing the server defaults, and a sample application is available here: <https://github.com/tomitribe/tomee-activemq-vm-defaults>

Can I turn this off?

The auto-creation of resources can be turned off by setting openejb.offline to true. Please be aware that you will need to explicitly define *every*resource used by your application, including containers for enterprise beans (the auto config feature does do quite a lot of work for you!).