JBOSS DATA GRID EMBEDDED WAR POC NOTES

GLOBAL PARTNER STRATEGY & ENABLEMENT

Table of Contents

1.1 Document History

Version	Revision Date	Contributor	Revision Description
001	2012-04-12	Randy Thomas	Initial draft

Table 2-1: Revision History

2 WHAT IS IT?

This example demonstrates the deployment of an embedded infinispan cache usage from an EJB 3.1 bean bundled in a war archive for deployment to JBoss Enterprise Application Platform 6 (EAP6). The project also includes a set of Arquillian tests for the managed bean and EJB.

1. A JSF page asks the user for basic test information.

2. On clicking submit, the test parameters are sent to a managed bean named **DGPerfTestController**.

3. **DGPerfTestController** invokes the **DataGridEJB**, which was injected into the managed bean with the annotation **@EJB**.

4. The response from invoking the **DataGridEJB** is stored in a field called **message** of the managed bean.

5. The managed bean is annotated as **@SessionScoped**, so the same managed bean instance is used for the entire session. This ensures that the message is available when the page reloads and is displayed to the user.

2.1 System requirements

All you need to build this project is Java 6.0 (Java SDK 1.6) or better, Maven 3.0 or better. The application this project produces is designed to be run on JBoss Enterprise Application Platform 6.

2.2 Configure Maven

2.3 Modifications to the Standalone configuration

The standalone configuration has internal infinispan libraries that we will not be using. The modular classloading built into EAP6 will automatically use the infinispan libraries in our war. However, since our embedded cache will distribute it's load across other nodes in the network, JGroups must be added to the standalone configuration.

2.3.1 Add JGroups

1. Navigate to JBOSS_HOME/standalone/configuration
2. Open standalone.xml in a text editor
3. Towards the top of the file, find the extensions and add the jgroups extension as illustrated below:

```
<extensions>
        <extension module="org.jboss.as.clustering.infinispan"/>
        <extension module="org.jboss.as.clustering.jgroups"/>
```

4. The profile section of the file contains many subsection entries. Go to the bottom of the profile and add a subsystem entry for jgroups as illustrated below:

```
<subsystem xmlns="urn:jboss:domain:jgroups:1.0" default-stack="udp">
            <stack name="udp">
                <transport type="UDP" socket-binding="jgroups-udp" diagnostics-socket-
binding="jgroups-diagnostics"/>
                <protocol type="PING"/>
                <protocol type="MERGE2"/>
                <protocol type="FD_SOCK" socket-binding="jgroups-udp-fd"/>
                <protocol type="FD"/>
                <protocol type="VERIFY_SUSPECT"/>
                <protocol type="BARRIER"/>
                <protocol type="pbcast.NAKACK"/>
                <protocol type="UNICAST2"/>
                <protocol type="pbcast.STABLE"/>
                <protocol type="pbcast.GMS"/>
                <protocol type="UFC"/>
```

```
                <protocol type="MFC"/>
                <protocol type="FRAG2"/>
                <protocol type="pbcast.STATE_TRANSFER"/>
                <protocol type="pbcast.FLUSH"/>
            </stack>
            <stack name="tcp">
                <transport type="TCP" socket-binding="jgroups-tcp" diagnostics-socket-
binding="jgroups-diagnostics"/>
                <protocol type="MPING" socket-binding="jgroups-mping"/>
                <protocol type="MERGE2"/>
                <protocol type="FD_SOCK" socket-binding="jgroups-tcp-fd"/>
                <protocol type="FD"/>
                <protocol type="VERIFY_SUSPECT"/>
                <protocol type="BARRIER"/>
                <protocol type="pbcast.NAKACK"/>
                <protocol type="UNICAST2"/>
                <protocol type="pbcast.STABLE"/>
                <protocol type="pbcast.GMS"/>
                <protocol type="UFC"/>
                <protocol type="MFC"/>
                <protocol type="FRAG2"/>
                <protocol type="pbcast.STATE_TRANSFER"/>
                <protocol type="pbcast.FLUSH"/>
            </stack>
        </subsystem>
    </profile>
```

5. JGroups also requies a number of socket binding entries add those to the socket-binding-group section as illustrated below. The lines in bold are the additions for jgroups:

```
<socket-binding-group name="standard-sockets" default-interface="public" port-
offset="${jboss.socket.binding.port-offset:0}">
        <socket-binding name="http" port="8080"/>
        <socket-binding name="https" port="8443"/>
        <socket-binding name="jacorb" port="3528"/>
        <socket-binding name="jacorb-ssl" port="3529"/>
        <socket-binding name="jgroups-diagnostics" port="0" multicast-
address="224.0.75.75" multicast-port="7500"/>
        <socket-binding name="jgroups-mping" port="0" multicast-
address="${jboss.default.multicast.address:230.0.0.4}" multicast-port="45700"/>
        <socket-binding name="jgroups-tcp" port="7600"/>
        <socket-binding name="jgroups-tcp-fd" port="57600"/>
        <socket-binding name="jgroups-udp" port="55200" multicast-
address="${jboss.default.multicast.address:230.0.0.4}" multicast-port="45688"/>
        <socket-binding name="jgroups-udp-fd" port="54200"/>
        <socket-binding name="jmx-connector-registry" interface="management"
port="1090"/>
        <socket-binding name="jmx-connector-server" interface="management"
port="1091"/>
        <socket-binding name="management-native" interface="management"
port="${jboss.management.native.port:9999}"/>
        <socket-binding name="management-http" interface="management"
port="${jboss.management.http.port:9990}"/>
        <socket-binding name="messaging" port="5445"/>
        <socket-binding name="messaging-throughput" port="5455"/>
        <socket-binding name="osgi-http" interface="management" port="8090"/>
        <socket-binding name="remoting" port="4447"/>
        <socket-binding name="txn-recovery-environment" port="4712"/>
        <socket-binding name="txn-status-manager" port="4713"/>
        <outbound-socket-binding name="mail-smtp">
            <remote-destination host="localhost" port="25"/>
        </outbound-socket-binding>
    </socket-binding-group>
```

6. Save the configuration file and exit the editor

2.3.2 Duplicate the server

Navigate to the parent directory for JBOSS_HOME and copy the entire server to the same location with a different name. Name the second server **eap6_node2**. For example, on unix you may use the following:

```
cp -r jboss-eap-6.0 eap6_node2
```

2.3.3 Add a port offset for the second server

The second server must use different ports than the first if it is to run on the same machine.

1. To do this navigate to the **standalone.xml** file on the **eap6_node2** server
2. Towards the end of the configuration file, find the **socket-binding-group** element
3. Set the port-offset element to a 100 as illustrated below. The highlighted number is the only change.

```
<socket-binding-group name="standard-sockets" default-interface="public" port-
offset="${jboss.socket.binding.port-offset:100}">
```

2.4 Start JBoss Enterprise Application Platform 6

1. Open a command line and navigate to the root of the JBoss server directory.

2. The following shows the command line to start the server with the web profile:

For Linux: JBOSS_HOME/bin/standalone.sh

For Windows: JBOSS_HOME\bin\standalone.bat

2.5 Build the project

```
Mvn clean package
```

2.6 Start a jboss admin CLI for server 1

jboss-eap-6.0/bin/jboss-admin.sh

> connect

2.7 Start a jboss admin CLI for server 2

eap6_node2/bin/jboss-admin.sh

> connect localhost:10099

2.8 Use CLI to deploy the package on both servers

> deploy projectdir/target/dg-perf-test.war

Access the two war files from a browser:

http://localhost:8080/dg-perf-test

http://localhost:8180/dg-perf-test

3 OTHER VERSIONS

The initialPOC git branch uses the infinispan instance that comes with EAP6. Using this version requires the following addition to the Infinispan subsystem configuration:

```
<cache-container name="agilaireCacheContainer" default-cache="default"  jndi-
name="java:jboss/infinispan/agilaireCacheContainer">
                <distributed-cache owners="2" mode="SYNC" name="default"
start="EAGER">
                    <locking isolation="REPEATABLE_READ" striping="false"/>
                    <rehashing enabled="true" timeout="600000"/>
                </distributed-cache>
            </cache-container>
```

This approach has the advantage of allowing us to use CDI to inject our cache in the code like the following:

```
    @Resource(lookup="java:jboss/infinispan/agilaireCacheContainer")
    private org.infinispan.manager.CacheContainer container;
    private org.infinispan.Cache<String, Object> cache;

     @PostConstruct
     public void postContruct() {
           this.cache = this.container.getCache();
     }
```