CHAPTER 3

CONFIGURING A JBOSS EAP CLUSTER

Overview	
Goal	Describe basic clustering concepts.
Objectives	• Describe basic clustering concepts. • Describe and configure Infinispan cluster services in EAP 7. • Describe the JGroups Subsystem and its role in a server cluster. • Configure and deploy a highly available Singleton.
Sections	• Reviewing Clustering Concepts (and Guided Exercise) • Exploring Infinispan (and Guided Exercise) • Exploring JGroups (and Guided Exercise) • Deploying HA Singletons (and Guided Exercise)
Lab	• Configuring a JBoss EAP Cluster

Reviewing Clustering Concepts

Objectives

After completing this section, students will be able to describe basic clustering concepts.

Clustering Concepts in EAP 7

A cluster is a collection of EAP servers that communicate with each other to improve the availability of services by providing:

- *High Availability (HA)*: a service has a very high probability of being available.
- *Scalability*: a service can handle a large number of requests by spreading the workload across multiple servers.
- *Failover*: if a service fails, the client continues processing its tasks on another cluster member.
- *Fault Tolerance*: a server can guarantee correct behavior even if failover occurs.
- *Load Balancing*: requests are spread out over the cluster so that no one server in the cluster becomes over-burdened with connections.

The most common way to achieve scalability and high availability is to use a load balancer. Previous releases of EAP required an external load balancer, such as Apache httpd, but EAP 7 now allows users to customize the Undertow subsystem to act as a front-end load balancer.

Important

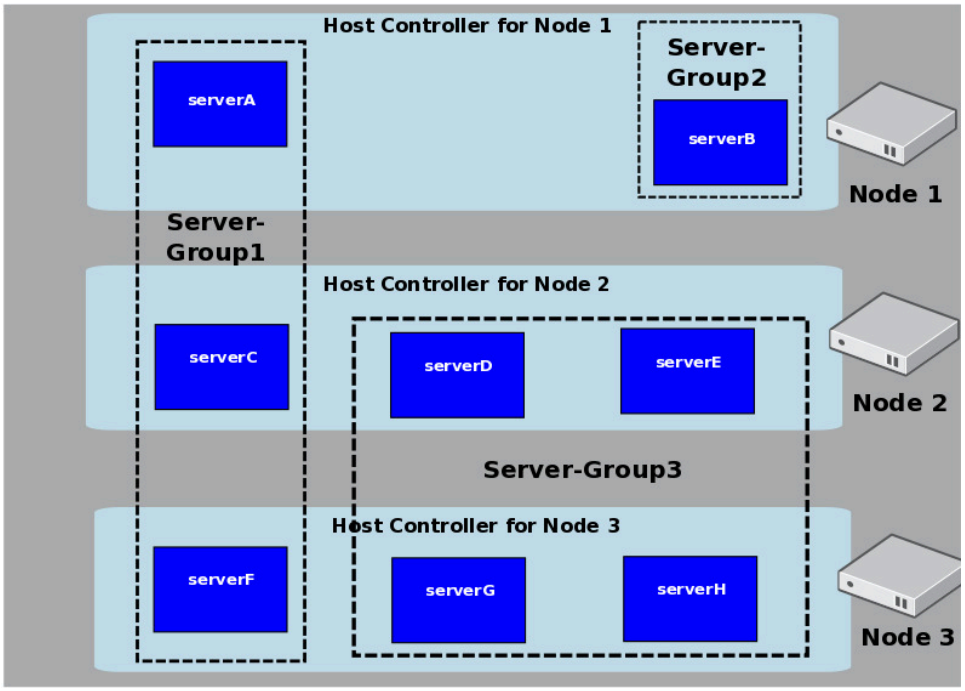
Clustering is made available to an EAP instance by three subsystems: **`jgroups`**, **`infinispan`** and **`modcluster`**. By default, the **`ha`** and **`full-ha`** profiles have these subsystems enabled.

Note

In EAP 7, as in EAP 6, the clustering services start up on demand, and they also shut down on demand, based on whether or not an application that is configured as **`distributable`** is deployed on the servers.

For EAP 7, a cluster is a group of identically-configured servers that communicate with each other to ensure that the cluster provides HA, failover and the other clustering capabilities. In a managed domain, a cluster is actually a collection of servers in a server group, with each server in the server group representing the nodes of the cluster.

Clustered Servers in Domain Mode



In a managed domain, a cluster is a collection of servers. A cluster can consist of two or more server groups, with all servers in the clustered server groups being the nodes of the cluster. As the above diagram shows, you can have multiple clusters within a domain, and you can have servers in a server group that do not form a cluster.

In the diagram **Clustered Servers in Domain Mode** there are three separate nodes. Each node represents a single host. Each node also has a corresponding EAP host controller. Each host controller defines multiple server instances, which are then organized into different server groups. The advantage of using server groups is the ability to execute configuration changes or deploy applications onto entire server groups.

In the diagram, the host controller running on Node 1 defines two servers, **`serverA`** and **`serverB`**. Meanwhile, the host controller on Node 2 has three servers, **`serverC`**, **`serverD`**, and **`serverE`**. Finally, the host controller on Node 3 defines three more servers, **`serverF`**, **`serverG`**, and **`serverH`**. Across the cluster, there are three server groups defined, **`Server-Group1`**, **`Server-Group2`**, and **`Server-Group3`**.

If the application **`helloWorld.war`** is marked as **`distributable`** and it is deployed on **`Server-Group1`**, then it is clustered with all of the servers in **`Server-Group1`**. Therefore if Node 3 were to fail, for example, then users are served the application from **`serverC`** or **`serverA`**.

Guided Exercise: Creating a Cluster

In this exercise, you will create a simple two-server cluster running in domain mode.

Eventually you are going to have a Domain Controller and two Host Controllers all running on your student workstation. In reality, you probably would run these three controllers on separate machines, so we are going to simulate separate machines by using subfolders named **machine1**, **machine2**, and **machine3**.

In this lab, you are going to configure **machine1** to run as the master controller. Also, you will create and configure **machine2** and **machine3** as slaves connecting to **machine1**.

Resources	
Files:	**/home/student/JB348/labs/create-cluster /home/student/JB348/apps/cluster.war**
Application URL:	**http://localhost:9990**, **http://localhost:8080/version**

Outcomes

You will be able to create and start a simple two-server cluster running in domain mode.

Before you begin

Use the following command to verify that an instance of EAP is installed in the **/opt/** directory and to download the server configuration files for this exercise:

```
[student@workstation ~]$ lab create-cluster setup
```

1. Create a New Domain Base Directory in the Workstation VM
   Open a terminal window from the workstation VM (**Applications** > **Favorites** > **Terminal**) and copy all of the contents from the folder **/opt/jboss-eap-7.0/domain** into a new folder **machine1** located in the lab directory **/home/student/JB348/labs/create-cluster/**. This creates a folder **machine1/**, which contains three subfolders: **configuration**, **data**, and **tmp**:

   ```
   [student@workstation ~]$ cd /home/student/JB348/labs/create-cluster
   [student@workstation create-cluster]$ mkdir machine1
   [student@workstation create-cluster]$ cp -r /opt/jboss-eap-7.0/domain/* machine1
   ```

2. Configure Files in **machine1** for a Domain Controller
   2.1. Using the editor of your choice, open the **host-master.xml** file in the **/home/student/JB348/labs/create-cluster/machine1/configuration** folder. This host configuration file configures a domain controller that does not manage any local servers.

   2.2. Look at the following line at the beginning of the **host-master.xml** file:

   ```
   <host xmlns="urn:jboss:domain:4.1" name="master">
   ```

   This line configures the name of this host to be "**master**".

2.3. There is only one interface defined in **host-master.xml**, named **management**.

```
...
    <interfaces>
        <interface name="management">
            <inet-address value="${jboss.bind.address.management:127.0.0.1}"/>
        </interface>
    </interfaces>
...
```

It is assumed that the host machine running the domain controller is not hosting servers. Consequently, it does not need to define a **public** network interface for the server instances to accept user requests.

2.4. Any slaves must be configured to point to the IP address of the domain controller.

2.5. The labs are going to simulate multiple machines, and binding to **127.0.0.1** does not make the domain controller on the **machine1** folder visible to outside machines.

Modify the **management** interface's **inet-address** to bind to the IP address of your workstation machine (172.25.250.254). The **<interfaces>** section of **host-master.xml** appears as follows:

```
<interfaces>
    <interface name="management">
        <inet-address value="${jboss.bind.address.management:172.25.250.254}"/>
    </interface>
</interfaces>
```

Note

Instead of manually editing the XML file, it is also possible to specify the **jboss.bind.address.management** property for the startup script.

2.6. Save your changes to **host-master.xml** and close the text editor.

2.7. Open the **domain.xml** in **/home/student/JB348/labs/create-cluster/machine1/configuration**.

2.8. Inside of the **messaging-activemq** subsystem of the **full-ha** profile, edit the **<cluster>** tag (line 1278):

```
<cluster password="${jboss.messaging.cluster.password:JBoss@RedHat123}" />
```

2.9. Save your changes to **domain.xml**.

3. Start the Domain Controller
Start the domain controller using the **host-master.xml** configuration file:

```
[student@workstation create-cluster]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./domain.sh \
```

```
-Djboss.domain.base.dir=/home/student/JB348/labs/create-cluster/machine1/ \
--host-config=host-master.xml
```

4. Create and Configure the host2 Host Controller
   Create a new host controller with the following properties:

   - **Host Controller server:** localhost
   - **Base directory: /home/student/JB348/labs/create-cluster/machine2**
   - **Host name:** host2
   - **Native interface port:** 2999
   - **Management IP:** 172.25.250.254
   - **Public IP**: 172.25.250.254
   - **Private IP**: 172.25.250.254

   Creating and configuring a host controller is a repetitive task and can be scripted to make the job easier. A custom native interface port is used to avoid port conflicts because the domain controller from the previous lab is already bound to port **9999**.

   Open a new terminal window and run the following commands to create a host controller:

```
[student@workstation ~]$ cd /home/student/JB348/labs/create-cluster
[student@workstation create-cluster]$ ./create-hc.sh localhost \
/home/student/JB348/labs/create-cluster/machine2 \
host2 2999 172.25.250.254 172.25.250.254 172.25.250.254
```

Note

The cluster communication by default use the **JGroups** technology. The **JGroups** technology requires an interface named **private**. The **create-hc.sh** script creates this new interface.

5. Start the **host2** Host Controller
   5.1. Start **host2** using the **host-slave.xml** configuration file that has its management interface bound to **172.25.250.254** on port **29999**.

   Run the following command from the **/opt/jboss-eap-7.0/bin** folder in a new terminal window:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB348/labs/create-cluster/machine2/ \
--host-config=host-slave.xml -Djboss.domain.master.address=172.25.250.254
```

Note

The prefix of each log entry in the terminal window is either [HostController] or the name of the server that caused the log event, which is either [Server:server-one] or [Server:server-two] in your deployment.

5.2. Carefully review the log output in the terminal window of the host controller of **machine2**. The log shows that the host controller connects to the master, and that server-one and server-two have started.

```
[Host Controller] 16:42:57,307 INFO  [org.jboss.as.host.controller]
 (Controller Boot Thread) WFLYHC0148: Connected to master host controller at
 remote://172.25.250.254:9999
[Host Controller] 16:42:57,367 INFO  [org.jboss.as.host.controller] (Controller
 Boot Thread) WFLYHC0023: Starting server server-one
```

5.3. Look in the terminal window of the domain controller for the following log entry showing the slave connecting:

```
[Host Controller] 11:42:16,348 INFO  [org.jboss.as.domain.controller] (Host
 Controller Service Threads - 36) WFLYHC0019: Registered remote slave host
 "host2", JBoss JBoss EAP 7.0.0.GA (WildFly 2.1.2.Final-redhat-1)
```

6. Delete all servers
It is very common to delete all servers from a new host controller to create new servers with custom names, groups, and port offset. Open a new terminal window and delete all servers using the following commands:

```
[student@workstation ~]$ cd /home/student/JB348/labs/create-cluster
[student@workstation create-cluster]$ ./delete-server.sh host2 \
server-one
[student@workstation create-cluster]$ ./delete-server.sh host2 \
server-two
```

7. Create and Configure the host3 Host Controller
Create a new host controller with the following properties:

- **Host Controller server:** localhost
- **Base directory: /home/student/JB348/labs/create-cluster/machine3**
- **Host name:** host3
- **Native interface port:** 3999
- **Management IP:** 172.25.250.254
- **Public IP**: 172.25.250.254
- **Private IP**: 172.25.250.254

Run the following script to create a new host controller:

```
[student@workstation create-cluster]$ ./create-hc.sh localhost \
/home/student/JB348/labs/create-cluster/machine3 \
host3 3999 172.25.250.254 172.25.250.254 172.25.250.254
```

8. Start the **host3** Host Controller
   8.1. Start **host3** using the **host-slave.xml** configuration file that has its management interface bound to **172.25.250.254** on port **39999**.

   Run the following command from your **/opt/jboss-eap-7.0/bin**:

   ```
   [student@workstation create-cluster]$ cd /opt/jboss-eap-7.0/bin
   [student@workstation bin]$ ./domain.sh \
   -Djboss.domain.base.dir=/home/student/JB348/labs/create-cluster/machine3/ \
   --host-config=host-slave.xml -Djboss.domain.master.address=172.25.250.254
   ```

   8.2. Carefully review the log output in the terminal window of the host controller for machine2. The log shows that the host controller connects to the master, and that server-one and server-two have started.

   ```
   [Host Controller] 16:42:57,307 INFO  [org.jboss.as.host.controller]
    (Controller Boot Thread) WFLYHC0148: Connected to master host controller at
    remote://172.25.250.254:9999
   [Host Controller] 16:42:57,367 INFO  [org.jboss.as.host.controller] (Controller
    Boot Thread) WFLYHC0023: Starting server server-one
   ```

   8.3. Look in the terminal window of the domain controller for the following log entry showing the slave connecting:

   ```
   [Host Controller] 11:42:16,348 INFO  [org.jboss.as.domain.controller] (Host
    Controller Service Threads - 36) WFLYHC0019: Registered remote slave host
    "host3", JBoss JBoss EAP 7.0.0.GA (WildFly 2.1.2.Final-redhat-1)
   ```

9. Delete all servers from host3
   Open a new terminal window and delete all servers:

   ```
   [student@workstation ~]$ cd /home/student/JB348/labs/create-cluster
   [student@workstation create-cluster]$ ./delete-server.sh host3 \
   server-one
   [student@workstation create-cluster]$ ./delete-server.sh host3 \
   server-two
   ```

10. Create new servers
    10.1. Create a new server on **host2** host controller with the following properties:

    - **Name:** my-server-one
    - **Server group:** main-server-group
    - **Port offset:** 0

- **Auto start:** true

```
[student@workstation create-cluster]$ ./create-server.sh host2 \
my-server-one main-server-group 0 true
```

10.2. Create a new server on **host2** host controller with the following characteristics:

- **Name:** my-server-two
- **Server group:** other-server-group
- **Port offset:** 150
- **Auto start:** true

```
[student@workstation create-cluster]$ ./create-server.sh host2 \
my-server-two other-server-group 150 true
```

10.3. Create a new server on **host3** host controller with the following characteristics:

- **Name:** my-server-three
- **Server group:** other-server-group
- **Port offset:** 1000
- **Auto start:** true

```
[student@workstation create-cluster]$ ./create-server.sh host3 \
my-server-three other-server-group 1000 true
```

11. Deploy an application

11.1. Open a new terminal window and connect to the CLI tool:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh --connect --user=jbossadm \
--password=JBoss@RedHat123 --controller=172.25.250.254:9990
```

11.2. Deploy the **cluster** application on the server group **main-server-group**:

```
[domain@172.25.250.254:9990 /]  deploy /home/student/JB348/apps/cluster.war \
--server-groups=other-server-group
```

12. Verify the Servers are Running

12.1. On the **workstation** VM, open a web browser and navigate to **http://172.25.250.254:8080/cluster**. The browser displays a **404** error page. Since **my-server-one** does not belong to the **other-server-group** group, the application was not deployed on this server.

12.2. In your web browser, navigate to **http://172.25.250.254:8230/cluster**. The browser displays the default **cluster** application served by **my-server-two**.

12.3. In your web browser, navigate to **`http://172.25.250.254:9080/cluster`**. The browser displays the default **`cluster`** application served by **`my-server-three`**.

13. Test the Cluster

13.1. In your web browser, navigate to **`http://172.25.250.254:9080/cluster`**. Refresh the page until you have the visitations count defined as 10.

13.2. Press **`Ctrl`**+**`C`** in the terminal window where you started the **`machine3`** host to stop new requests to this server.

13.3. In your web browser, refresh the page to confirm that the server is down.

13.4. In your web browser, navigate to **`http://172.25.250.254:8230/cluster`**. You have not lost the visitations count and that the new value is defined as 11.

14. Clean Up and Grading

14.1. Run the following command from the **`workstation`** VM to grade the exercise:

```
[student@workstation ~]$ lab create-cluster grade
```

14.2. Press **`Ctrl`**+**`C`** in each terminal windows where you started the cluster instances of EAP to stop the cluster.

This concludes the guided exercise.

Exploring Infinispan

Objectives

After completing this section, students will be able to describe and configure Infinispan cluster services in EAP 7.

Infinispan Cluster Services

The Infinispan subsystem provides caching support for JBoss EAP, facilitating the high availability features of clustered servers.

In a clustered environment, similar data is replicated onto each node in the cluster. This data is stored in a cache, and the caching mechanism and features are implemented by a framework called *Infinispan*. In addition to being able to configure how EAP caches data, Infinispan also provides the facilities to view runtime metrics for cache containers and caches.

A cache is defined within a *cache container*, or a repository for the caches. There are four preconfigured cache containers in the **ha** and **full-ha** profiles:

- **web**: for session replication
- **hibernate**: for entity caching
- **ejb**: for stateful session bean replication
- **server**: for singleton caching

The **web**, **hibernate** and **ejb** caches are used by developers to cache Java components. In clustering, the nodes use a cache in the **cluster** container configured for replicating objects efficiently and effectively over a large cluster of nodes.

There are four different types of caches:

- **Local**: Entries are not distributed to the rest of the cache and are instead stored only on the local node.
- **Invalidation**: Uses a cache store to store entries, pulling from the store when an entry needs it.
- **Replication**: All entries are replicated on each node.
- **Distribution**: Entries are replicated to only some of the nodes.

Accordingly, there are four different pages in the Management Console for defining each type of cache. These pages are locate in the **Infinispan** section of the **Subsystem** page in the Management Console.

The following XML excerpt displays the default Infinispan configuration:

```
<subsystem xmlns="urn:jboss:domain:infinispan:4.0">
  <cache-container name="server" aliases="singleton cluster" default-cache="default"
 module="org.wildfly.clustering.server">
```

```
    <transport lock-timeout="60000"/>
    <replicated-cache name="default" mode="SYNC">
      <transaction mode="BATCH"/>
    </replicated-cache>
  </cache-container>
  <cache-container name="web" default-cache="dist"
 module="org.wildfly.clustering.web.infinispan">
    <transport lock-timeout="60000"/>
    <distributed-cache name="dist" mode="ASYNC" l1-lifespan="0" owners="2">
      <locking isolation="REPEATABLE_READ"/>
      <transaction mode="BATCH"/>
      <file-store/>
    </distributed-cache>
  </cache-container>
  <cache-container name="ejb" aliases="sfsb" default-cache="dist"
 module="org.wildfly.clustering.ejb.infinispan">
    <transport lock-timeout="60000"/>
    <distributed-cache name="dist" mode="ASYNC" l1-lifespan="0" owners="2">
      <locking isolation="REPEATABLE_READ"/>
      <transaction mode="BATCH"/>
      <file-store/>
    </distributed-cache>
  </cache-container>
  <cache-container name="hibernate" default-cache="local-query"
 module="org.hibernate.infinispan">
    <transport lock-timeout="60000"/>
    <local-cache name="local-query">
      <eviction strategy="LRU" max-entries="10000"/>
      <expiration max-idle="100000"/>
    </local-cache>
    <invalidation-cache name="entity" mode="SYNC">
      <transaction mode="NON_XA"/>
      <eviction strategy="LRU" max-entries="10000"/>
      <expiration max-idle="100000"/>
    </invalidation-cache>
    <replicated-cache name="timestamps" mode="ASYNC"/>
  </cache-container>
</subsystem>
```

The subsystem defines the four default cache containers: web, hibernate, ejb, and server. Each cache container specifies the **`default-cache`**. For example, the **`hibernate`** cache container uses the **`local-query`** cache, which maps to the **`local-cache`**.

Configure a new cache with the EAP CLI:

1. Create a cache container:

```
/subsystem=infinispan/cache-container=<container-name>:add
```

2. Add a replicated cache:

```
/subsystem=infinispan/cache-container=<container-name>/replicated-cache=<cache-
name>:add(mode=<mode>)
```

3. Set the default cache:

```
/subsystem=infinispan/cache-container=<container-name>:write-attribute(name=default-
cache,value=<cache-name>)
```

Infinispan Architecture

Infinispan provides an implementation of JSR-107. As such, it provides services from the **`javax.cache`** packages. Administrators need to understand how Infinispan works so that they can troubleshoot and tune the service.

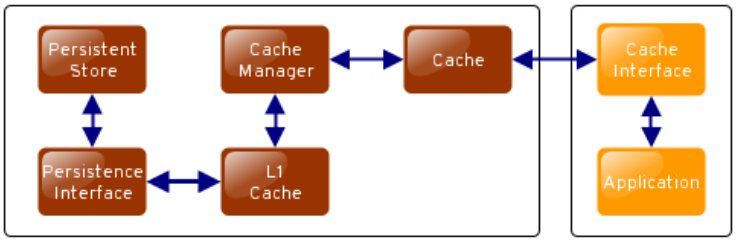


Figure 3.2: Infinispan Architecture

`L1 Cache` is also referred to as **`near cache`** in some caching products. It keeps a record of frequently accessed cache queries in the local memory. It is only used in cases where the caching mode is set to distributed. If it is not used, the **`Cache Manager`** communicates directly with the **`Persistence Interface`**, or **`Store`**.

The **`Cache Manager`** hands a **`Cache Interface`** to the application, which then communicates with the **`Cache`**. In general, unless a caching configuration differences at the wire level (TCP vs. UDP, for instance) is required, a single Cache Manager will suffice across all deployed applications on a server.

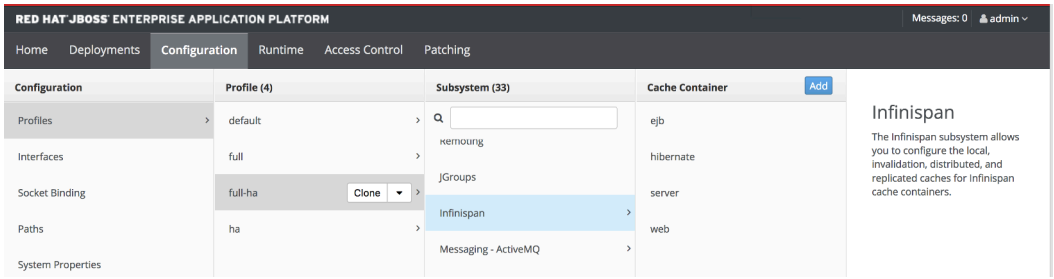
By using a Cache Container, it is possible to define multiple **`Cache Modes`**, and easily switch between them when the need arises.

A **`Persistent Store`** is a back end for the cache. It can be in memory, a flat file, or a database. In cases where it is left in memory, the store will not properly survive a system failure. Infinispan communicates with the persistent store through the **`Persistence Interface`**, which depends on the kind of store being used.

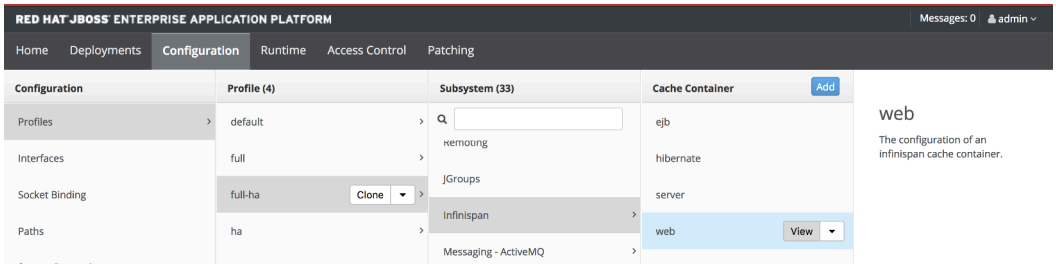
Configuring Infinispan

The Infinispan configuration is done by a desired profile. It means that all server groups that belongs to a profile will have the Infinispan configured. The following steps are required to configure the Infinispan subsystem:

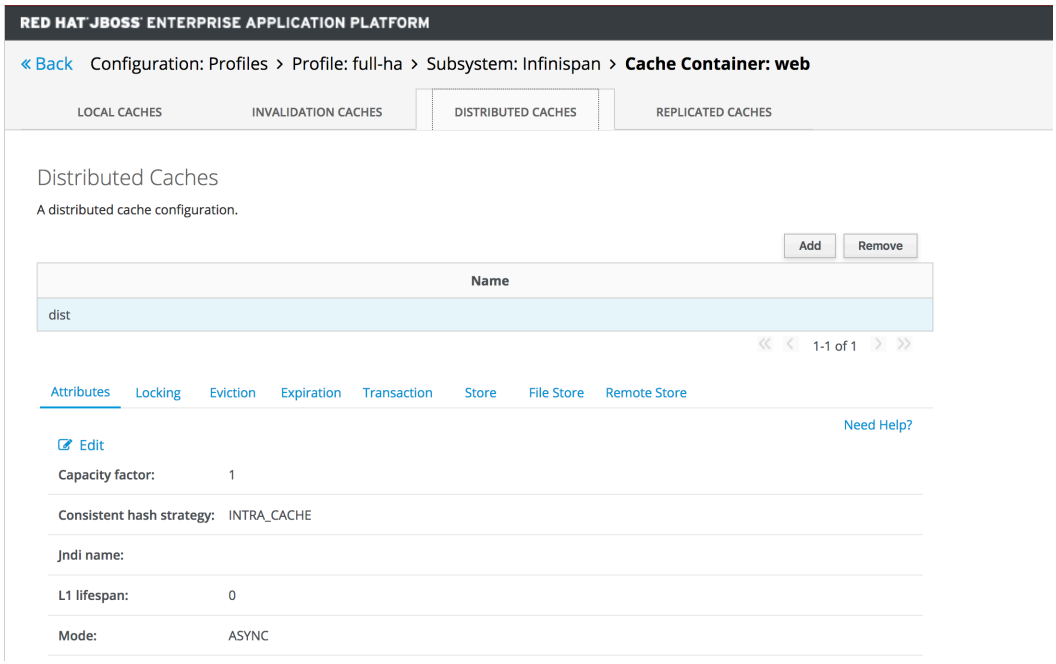
1. Open the Management console.
2. Click **Configuration** either from the top menu or from the home screen.
3. In the first column, click **Profiles**. In the second column, click the desired profile. In the third column, click **Infinispan**. The four preconfigured cache containers should be displayed.



4. To create a new cache container, click **Add** in the fourth column. To manage a cache, click the cache container and then click **View**.



5. Select the type of cache at the top of the page to view the configurations.



Demonstration: Configuring a Simple Cache

1. Open a terminal window from the workstation VM (**Applications** > **Favorites** > **Terminal**) and run the following command to create the lab directory and verify that EAP is installed and not currently running:

```
[student@workstation ~]$ demo simple-cache setup
```

2. Start the domain controller:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB348/labs/simple-cache/machine1/ \
--host-config=host-master.xml
```

3. Open a new terminal window and start **host2** using the **host-slave.xml** configuration file:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB348/labs/simple-cache/machine2/ \
--host-config=host-slave.xml -Djboss.domain.master.address=172.25.250.254
```

4. Open a new terminal window and start **host3** using the **host-slave.xml** configuration file:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB348/labs/simple-cache/machine3/ \
--host-config=host-slave.xml -Djboss.domain.master.address=172.25.250.254
```

5. Once the server finishes starting up, open a web browser and navigate to the management console at **http://172.25.250.254:9990**. Use the following preconfigured administrator credentials to log in:
   - User Name: **jbossadm**
   - Password: **JBoss@RedHat123**
6. Click **Configuration** either from the top menu or from the home screen.
7. In the first column, click **Profiles**. In the second column, click **ha**. In the third column, click **Infinispan**. In the fourth column, click **ejb** and then click **View**.
8. Make sure that **Local Caches** at the top of the page is selected.
9. Click **Add** to create a new cache. Name it **local_ejb** and click **Save**.
10. Under **Transaction**, click **Edit**. Update the **Locking** to **OPTIMISTIC**. With optimistic locking, a resource is not actually locked when it is first accessed by a transaction.
11. Click **Save** to enable the new configuration and then click **Back** in order to return to the main **Configuration** page.
12. Click **ejb** in the **Cache Container** column and click the down arrow next to **View** and then click **Container settings**.

13. Click **Edit** and update **Default cache** to **local_ejb** to define the new local cache as the default cache for the **ejb** cache container.

14. Click **Save** to enable the new configuration.

15. Press **Ctrl+C** in each terminal windows where the cluster instances of EAP was started to stop the cluster.

This concludes the demonstration.

Tuning Infinispan

Tuning Infinispan depends on each application that uses the cache. If the application works with data that changes often, for example, set the **Expiration** and **Eviction** so that expired entries are automatically purged, rather than waiting for a future get from the application to trigger the purge. **Eviction** is similar to **Garbage Collection** where **Expiration** is like marking objects as available for collection. Eviction defines the **maxEntries** attribute as a power of two. If the attribute is not defined as a power of two, the next highest power of two is selected. Thus, there is no point in setting **maxEntries** to 65. This bumps the **maxEntries** up to 128, regardless.

If Infinispan needs to start some caches immediately when EAP starts, rather than waiting for an application to use them, set the start mode of those caches to **EAGER** as opposed to the default, which is **LAZY**. When using **EAGER**, use a **Store** as well, to mitigate the impact of slow startup times.

References

Infinispan User Guide
http://infinispan.org/docs/stable/user_guide/user_guide.html

Guided Exercise: Tuning Infinispan

In this exercise, you will tune the Infinispan subsystem to improve the cache performance.

Resources	
Files:	**/home/student/JB348/labs/tuning-infispan /home/student/JB348/apps/airports.war**
Application URL:	**http://localhost:8080/airports**, **http://localhost:8180/airports**

Outcomes

You will be able to tune the Infinispan subsystem running in domain mode.

Before you begin

Use the following command to verify that an instance of EAP is installed in the **/opt/** directory and to download the server configuration files for this exercise:

```
[student@workstation ~]$ lab tuning-infinispan setup
```

1. Start the cluster
   1.1. Open a terminal window from the workstation VM (**Applications** > **Favorites** > **Terminal**) and start the domain controller:

```
[student@workstation create-cluster]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB348/labs/tuning-infinispan/machine1/ \
--host-config=host-master.xml
```

   1.2. Start **host2** using the **host-slave.xml** configuration file that has its management interface bound to **172.25.250.254** on port **29999**.

   Run the following command from your **/opt/jboss-eap-7.0/bin** folder in a new terminal window:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB348/labs/tuning-infinispan/machine2/ \
--host-config=host-slave.xml -Djboss.domain.master.address=172.25.250.254
```

   1.3. Start **host3** using the **host-slave.xml** configuration file that has its management interface bind to **172.25.250.254** on port **39999**.

   Run the following command from your **/opt/jboss-eap-7.0/bin** folder in a new terminal window:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB348/labs/tuning-infinispan/machine3/ \
--host-config=host-slave.xml -Djboss.domain.master.address=172.25.250.254
```

2. Create a new Infinispan Replicated Cache
   The **airports** application used in this lab requires a replicated cache to store the airports around the world. The first time that an airport is searched, the cache is populated.

   2.1. Open a new terminal window and connect to the CLI tool:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh --connect --user=jbossadm \
--password=JBoss@RedHat123 --controller=172.25.250.254:9990
```

   2.2. Navigate to the **infinispan** subsystem in the **ha** profile:

```
[domain@172.25.250.254:9990 /] cd /profile=ha/subsystem=infinispan
```

   2.3. Create a new cache container named **airport**. The cache container must have the **jndi-name** defined as **infinispan/airports_container**:

```
[domain@172.25.250.254:9990 subsystem=infinispan] ./cache-container=airport \
     :add(jndi-name=infinispan/airports_container)
```

   2.4. Infinispan synchronizes the cache with other instances of the cluster using the **JGroups** transport. Set the timeout value, when obtaining locks for the transport, to one minute:

```
[domain@172.25.250.254:9990 subsystem=infinispan] cd cache-container=airport
[domain@172.25.250.254:9990 cache-container=airport] ./transport=TRANSPORT/ \
:add(lock-timeout=60000)
```

   2.5. Create a synchronized replicated cache named **airports**. Presumably the replicated cache must have the **jndi-name** defined as **infinispan/airports_container/airports**.

```
[domain@172.25.250.254:9990 cache-container=airport] ./replicated-cache=\
airports:add(jndi-name=infinispan/airports_container/airports, mode=SYNC)
```

   2.6. Reload the servers to enable the new cache:

```
[domain@172.25.250.254:9990 cache-container=airport] /:reload-servers
```

3. Deploy the airports application
   Deploy the **airports** application on the server group **airport-group**:

```
[domain@172.25.250.254:9990 cache-container=airport] cd /
[domain@172.25.250.254:9990 /]  deploy /home/student/JB348/apps/airports.war \
--server-groups=airport-group
```

4. Test the application
   4.1. On the **workstation** VM, open a web browser and navigate to **http://172.25.250.254:8080/airports** to access the airports application.

4.2. The **airports** application loads the details about an airport based on the **ICAO** code. It is a four-character code designating aerodromes around the world. Fill the ICAO code with **sbbr** and click **Load Airport**.

This is the first time the application has been accessed, so the cache must be loaded before detailed information can be returned. Take a note about the time spent during the first request.

4.3. Next, fill the ICAO code with **krdu** and click **Load Airport**. This time all of the airports are available from the cache and the details about the airport is returned very quickly.

4.4. In your web browser, navigate to **http://172.25.250.254:8180/airports**. This is the second node on the cluster. Fill the ICAO code with **katl** and click **Load Airport**. Since you configured a replicated cache, the cache is already available to this node and the details about this airport is returned quickly.

5. Tune the cache

Every time that the cluster is restarted, the first request must be delayed since the cache will be loaded. To avoid this problem, configure the cache to save data to disk and to load the cache while the server is booting.

5.1. Go back to the terminal that is running CLI and create a new path to persist the cache file.

```
[domain@172.25.250.254:9990 /]  /path=airport.cache.destination\
:add(path=/home/student/JB348/labs/tuning-infinispan)
```

5.2. Add a persistent file to the cache container that should be used during the server booting with the following properties:

- **path**: airport-cache. This is the file that will persist the cache.
- **relative-to**: airport.cache.destination . This is the path to the cache file.
- **passivation**: false. False means that the cache store contains a copy of the contents in memory, so writes to cache result in cache store writes.
- **preload**: true. True means that when the cache starts, data in the cache store is preloaded into memory during the boot process.
- **purge**: false. False means that the cache store is not purged at startup time.

```
[domain@172.25.250.254:9990 /]  /profile=ha/subsystem=infinispan/\
cache-container=airport/replicated-cache=airports/\
file-store=FILE_STORE:add(path=airport-cache, \
relative-to=airport.cache.destination,\
passivation=false, preload=true, purge=false)
```

5.3. Reload the servers to enable the new configuration:

```
[domain@172.25.250.254:9990 /] :reload-servers
```

6. Test the new configuration
   6.1. On the **workstation** VM, open a web browser and navigate to **http://172.25.250.254:8080/airports**. Fill the ICAO code in with **kjfk**.

   Since this is the first time that the cache is being accessed after the tuning, it takes extra time to populate the cache and persist it.

   6.2. Go back to terminal that is running CLI and reload the cluster:

```
[domain@172.25.250.254:9990 /] :reload-servers
```

   6.3. Go back to the web browser and refresh the **http://172.25.250.254:8080/airports** page. Fill the ICAO code with **eglc**.

   The response time is much faster since the cache was loaded during the booting process.

7. Clean Up and Grading
   7.1. Run the following command from the workstation to grade the exercise:

```
[student@workstation ~]$ lab tuning-infinispan grade
```

   7.2. Press **Ctrl**+**C** in each terminal windows where you started the cluster instances of EAP to stop the cluster.

   This concludes the guided exercise.

Exploring JGroups

Objectives

After completing this section, students will be able to describe the JGroups Subsystem and its role in a server cluster.

JGroups Overview

The **`JGroups Subsystem`** provides all communication mechanisms allowing servers in a cluster to communicate with one another.

JBoss EAP 7 ships with **`JGroups`** and includes configurations for cluster communications in the **ha** and **`full-ha`** profiles which provide a good baseline example for clustering.

JGroups itself uses the concepts of nodes and clusters. A cluster is, to put it simply, a collection of nodes. A node broadly maps to an EAP server. Individual nodes can be part of multiple clusters, provided the node is running multiple EAP server instances. The following diagram outlines nodes in a cluster:

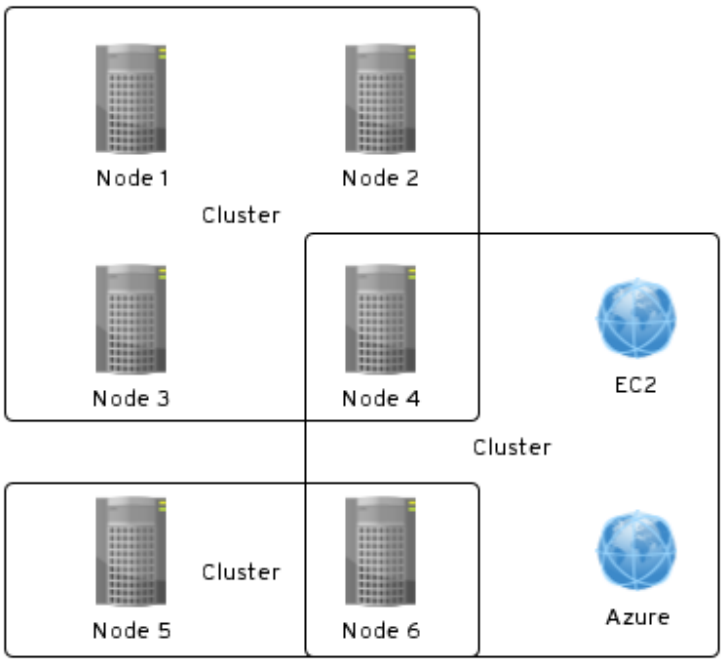


Figure 3.6: JGroups Cluster

JGroups in EAP 7 uses an address and port to define the nodes in a cluster. If a node needs a cluster address that does not exist, JGroups will create a new cluster to accommodate it. This simplifies cluster configuration immensely, since the servers themselves create clusters as needed.

In EAP 7, the JGroups sockets require a new interface named **`private`**, as listed on the **`sockets`** section from the desired profile:

```
<socket-binding name="jgroups-mping" interface="private" port="0" multicast-
address="${jboss.default.multicast.address:230.0.0.4}" multicast-port="45700"/>
<socket-binding name="jgroups-tcp" interface="private" port="7600"/>
<socket-binding name="jgroups-tcp-fd" interface="private" port="57600"/>
<socket-binding name="jgroups-udp" interface="private" port="55200" multicast-
address="${jboss.default.multicast.address:230.0.0.4}" multicast-port="45688"/>
<socket-binding name="jgroups-udp-fd" interface="private" port="54200"/>
<socket-binding name="modcluster" port="0" multicast-address="224.0.1.105" multicast-
port="23364"/>
```

The objective for the new interface is to separate the network traffic used by the cluster from the network traffic used by the application.

JGroups is designed to ensure reliable group communication. Individual nodes can send and receive messages from all or some of the other members of the group. As nodes join and leave the cluster, JGroups tracks and informs all nodes of changes in cluster membership, and JGroups provides a view of the nodes of a cluster at any given time. JGroups also handles retransmission in the case of a delivery failure, elimination of duplicate messages, and orders the messages so that nodes are getting the right messages at the right time.

Note

All cluster communication goes through JGroups. Infinispan manages its distributed cluster via JGroups. This gives an administrator a single configuration point to manage how a cluster communicates.

JGroups Architecture

JGroups consists of three parts:

- **`Channel:`** A channel provides a link to the JGroups system. A client joins a group by connecting the channel to a group and leaves it by disconnecting. Messages sent over the channel are received by all group members that are connected to the same group.
- **`Building blocks:`** Channels are simple socket-like constructs that does not provides a sophisticated interface. JGroups offers building blocks that provide more sophisticated APIs on top of a Channel. Building blocks either create and use channels internally, or require an existing channel to be specified when creating a building block.
- **`Protocol stack:`** The protocol stack is responsible for translating messages to transmit them across the network to other nodes.

Configure JGroups

EAP is preconfigured with two JGroups stacks:

1. **UDP**: the nodes in the cluster use User Datagram Protocol (UDP) multicasting to communicate with each other. This is the default stack.
2. **TCP**: the nodes in the cluster use Transmission Control Protocol (TCP) to communicate with each other.

Users can use one of these preconfigured stacks, or can define and use a new stack that suits the specific needs of the environment. By default, the UDP protocol is used to communicate between

clustered nodes in the default **ee** JGroups channel. The following EAP CLI command adjusts the **ee** JGroups channel to use a **tcp** stack instead of **UDP**:

```
/subsystem=jgroups/channel=ee:write-attribute(
name=stack,value=tcp)
```

Also, it is important to define the default stack attribute to use **tcp**:

```
/subsystem=jgroups:write-attribute(
name=default-stack,value=tcp)
```

By default, the **TCP** stack uses multicast for discovering other members of a cluster. Users can further customize the **TCP** stack by changing the protocol to either **TCPPING** or **TCPGOSSIP**.

- **TCPPING**: a protocol that uses a static list to define the cluster members and uses unicast as an alternative to multicast. The following configurations are specific to this protocol:
  - **initial_hosts**: a list of the hosts that are available and known to look up for cluster membership.
  - **port_range**: the range that the protocol uses to search for hosts based on the initial port. For example, a port range of two on an initial port of **7600** results in the **TCPPING** protocol searching for a viable host on ports **7600** and **7601** to be added to the membership.
- **TCPGOSSIP**: discovers members of a cluster by using an external gossip router.

The following configuration is an example of a full **TCP** cluster:

```
<stack name="tcpping">
  <transport type="TCP" socket-binding="jgroups-tcp"/>
    <protocol type="TCPPING">
      <property name="initial_hosts">
        servera[7600],servera[7700],serverb[7600],serverb[7700]
      </property>
      <property name="port_range">
          10
      </property>
    </protocol>
    <protocol type="MERGE3" (1)
    <protocol type="FD_SOCK" (2) socket-binding="jgroups-tcp-fd"/>
    <protocol type="FD" (3) />
    <protocol type="VERIFY_SUSPECT" (4)/>
    <protocol type="pbcast.NAKACK2" (5)/>
    <protocol type="UNICAST3" (6)/>
    <protocol type="pbcast.STABLE" (7)/>
    <protocol type="pbcast.GMS" (8)/>
    <protocol type="MFC" (9)/>
    <protocol type="FRAG2" (10)/>
</stack>
```

(1) In **MERGE3**, all members periodically send an **INFO** message with their address (UUID), logical name, physical address and ViewId.
(2) Failure detection protocol based on a ring of TCP sockets created between cluster members.

3. Failure detection based on heartbeat messages.
4. Verifies that a suspected member is really dead by pinging that member one last time before excluding it.
5. The pbcast.NAKACK protocol provides reliable delivery and First In First Out (FIFO) properties for messages sent to all nodes in a cluster.
6. UNICAST3 provides reliable delivery and FIFO properties for point-to-point messages between one sender and one receiver.
7. The pbcast.STABLE protocol garbage collects messages that have been seen by all members of a cluster.
8. Responsible for managing the group membership.
9. Flow control based on a credit based system.
10. Fragment large messages into smaller ones. Send the smaller ones, and at the receiver side, assemble them into larger messages again.

Tuning JGroups

Tuning JGroups depends on each environment. Here are a few key notes about tuning the JGroups:

- When running multiple clusters from a single node, change the port and, preferentially, the multicast address to keep cluster communications separate.
- JGroups uses sized packets to perform communications. The packet size can be controlled in the settings for the protocol, and should be set to the same size as the maximum sizes defined at the operating system level. This is one of a very few places where the operating system can also be tuned to better support EAP clusters. Either way, the sizes should match, if at all possible.
- Long JVM Garbage Collection (GC) may impact the FD (Failure Detection) timeout and retry. The GC may need to be tuned for shorter GC cycles or a longer FD timeout.
- The default TCP stack included with JBoss EAP 7 still uses MPING for server discovery, which runs over UDP. To completely remove UDP from the stack, replace MPING with TCPPING and configure appropriately.

JGroups in the Cloud

Some cloud providers charge less for traffic over internal IP addresses compared to public IP addresses. In fact, some cloud providers charge nothing for traffic over the internal network. In these circumstances, it's advantageous to configure the group communications in such way that replication traffic is sent via the internal network. However, the administrator may not know which internal IP address will be assigned.

JGroups, the underlying group communication library, provides a way for users to bind to a type of address rather than to a specific IP address. Administrators can configure **`bind_addr`** property in JGroups configuration file, or the **`-Djgroups.bind_addr`** system property to a keyword rather than a dotted decimal or symbolic IP address:

- **`GLOBAL:`** pick a public IP address. This should be avoided due to replication traffic.
- **`SITE_LOCAL:`** use a private IP address, for instance **`192.168.x.x`**. This avoids charges for bandwidth from providers who do not charge for internal network traffic.

- **`LINK_LOCAL:`** use a **`169.x.x.x`**, **`254.0.0.0`** address. This works only within one box, for instance, in the examples provided in this section, with two clusters on one machine.
- **`NON_LOOPBACK:`** use the first address found on an interface found on an active interface, which is not a **`127.x.x.x`** address.

When using JGroups with Amazon EC2 (a service provider supported by Red Hat OpenShift), it is required to include EC2-specific configurations. The three key ones are:

- **`jgroups.s3.access_key:`** The Amazon S3 access key used to access an S3 bucket.
- **`jgroups.s3.secret_access_key:`** The Amazon S3 secret key used to access an S3 bucket.
- **`jgroups.s3.bucket:`** Name of the Amazon S3 bucket to use. Must be unique and must already exist.

Troubleshooting JGroups with mcast

Some cluster environments do not support multicast and for this reason the nodes do not form a cluster.

There are two applications that can be used to detect this: **McastReceiverTest** and **McastSenderTest**.

The **McastReceiverTest** listens for all multicast messages from a specific IP and port. Use the following command to start the program:

```
# java -cp jgroups-3.6.8.Final-redhat-2.jar (1) org.jgroups.tests.McastReceiverTest -
mcast_addr (2) 230.0.0.4 -port (3) 45688
```

(1) The JGroups jar file can be found on **`JBOSS_HOME/modules/system/layers/base/org/jgroups/main/`** folder.
(2) The multicast address that JGroup should use for listening messages.
(3) The multicast port that JGroup should use for listening messages.

The **McastSenderTest** sends a multicast message to a specific ip and port. Use the following command to start the program:

```
# java -cp jgroups-3.6.8.Final-redhat-2.jar org.jgroups.tests.McastSenderTest -
mcast_addr (1) 230.0.0.4 -port (2) 45688
```

(1) The multicast address that JGroup should use for sending a message.
(2) The multicast port that JGroup should use for sending a message.

If a message is sent by the **McastSenderTest** application and is not displayed by the **McastReceiverTest**, it means that multicast is not allowed in that environment.

Guided Exercise: Troubleshooting JGroups

In this exercise, you will troubleshoot an issue with JGroups in an EAP cluster.

Resources	
Files:	**/home/student/JB348/labs/troub-jgroups**, **/home/student/JB348/apps/cluster.war**, **/tmp/new-tcp-stack.cli**
Application URL:	**http://localhost:9080/cluster**

Outcomes

You will able to troubleshoot a JGroups problem using the **mcast** program.

Before you begin

Use the following command to verify that an instance of EAP is installed in the **/opt/** directory and to download the server configuration files for this exercise:

```
[student@workstation ~]$ lab troub-jgroups setup
```

1. Configure the firewall
   This guided exercise starts a cluster using the **workstation** VM as domain controller and load balancer, and **server A** and **server B** VM as slave controllers. In addition, a firewall is configured to allow communication in the cluster between its members.

   1.1. Open a terminal window from the workstation VM (**Applications** > **Favorites** > **Terminal**) and apply the following firewall rules:

   - **Port**: **9990/tcp** - used to create a socket for the management interface.
   - **Port**: **9999/tcp** - used to create a socket for the management native interface.
   - **Port**: **9080/tcp** - used by the load balancer.

```
[student@workstation ~]$ sudo firewall-cmd --permanent --zone=public \
--add-port=9990/tcp
```

```
[student@workstation ~]$ sudo firewall-cmd --permanent --zone=public \
--add-port=9999/tcp
```

```
[student@workstation ~]$ sudo firewall-cmd --permanent --zone=public \
--add-port=9080/tcp
```

   1.2. Reload the firewall rules to enable the new configuration:

```
[student@workstation ~]$ sudo firewall-cmd --reload
```

   1.3. Access the **servera** VM using the ssh command:

```
[student@workstation ~]$ ssh servera
```

Apply the following firewall rules:

- **Port**: 8080/tcp - used to create a socket responsible for serving the application requests using the HTTP protocol.
- **Port**: 8009/tcp - used to create a socket responsible for serving the application requests using the AJP protocol.
- **Port**: 7600/tcp - used by JGroups to recognize the members that belongs to a cluster while JGroups is configured to use TCP instead of UDP.
- **Port**: 57600/tcp - This port is used by JGroups to detect failures based on sockets generating notifications if a member fails.

```
[student@servera ~]$ sudo firewall-cmd --permanent --zone=public \
--add-port=8080/tcp
```

```
[student@servera ~]$ sudo firewall-cmd --permanent --zone=public \
--add-port=8009/tcp
```

```
[student@servera ~]$ sudo firewall-cmd --permanent --zone=public \
--add-port=7600/tcp
```

```
[student@servera ~]$ sudo firewall-cmd --permanent --zone=public \
--add-port=57600/tcp
```

1.4. Reload the firewall rules to enable the new configuration:

```
[student@servera ~]$ sudo firewall-cmd --reload
```

1.5. Access the **serverb** VM using the ssh command:

```
[student@servera ~]$ ssh serverb
```

Apply the same rules from **server A**:

```
[student@serverb ~]$ sudo firewall-cmd --permanent --zone=public \
--add-port=8080/tcp
```

```
[student@serverb ~]$ sudo firewall-cmd --permanent --zone=public \
--add-port=8009/tcp
```

```
[student@serverb ~]$ sudo firewall-cmd --permanent --zone=public \
--add-port=7600/tcp
```

```
[student@serverb ~]$ sudo firewall-cmd --permanent --zone=public \
--add-port=57600/tcp
```

1.6. Reload the firewall rules to enable the new configuration:

```
[student@serverb ~]$ sudo firewall-cmd --reload
```

2. Start and configure a load balancer
The setup script created the load balancer configuration in the **/home/student/JB348/labs/troub-jgroups/lb** folder on **workstation** VM. Start the load balancer with the **standalone-ha.xml** and a port offset of 1000 to avoid port clashes with the domain controller running on the **workstation**. Because the load balancer is going to be the entry point for the application, it must be configured to listen on the public IP of the **workstation** (172.25.250.254).

2.1. Open a new terminal window from **workstation** and run the following command to start the load balancer:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./standalone.sh \
-Djboss.server.base.dir=/home/student/JB348/labs/troub-jgroups/lb \
-Djboss.bind.address=172.25.250.254 -Djboss.socket.binding.port-offset=1000 \
-c standalone-ha.xml
```

2.2. Launch the EAP CLI in a new terminal window and connect to the load balancer instance on workstation.

In a new terminal window on **workstation**, start the EAP CLI and connect to the load balancer:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh --connect --controller=localhost:10990
```

2.3. Configure the **modcluster** subsystem to act as a front-end load balancer. Set a unique password for the **advertise-security-key**.

```
[standalone@localhost:10990 /] /subsystem=modcluster/mod-cluster-config=\
configuration:write-attribute\
(name=advertise-security-key, value=redhat)
```

2.4. Configure the **mod_cluster** filter. Advertise the load balancer using the **modcluster** socket-binding and use the HTTP protocol for the management socket binding. Ensure that the **security-key** attribute matches the **advertise-security-key** you configured in the previous step.

```
[standalone@localhost:10990 /] /subsystem=undertow/configuration=\
filter/mod-cluster=modcluster:add\
(management-socket-binding=http, advertise-socket-binding=modcluster,\
security-key=redhat)
```

2.5. Bind the **modcluster** filter to the undertow **default-server**.

```
[standalone@localhost:10990 /] /subsystem=undertow/server=\
default-server/host=default-host/filter-ref=modcluster:add
```

2.6. Reload the load balancer configuration and exit from CLI:

```
[standalone@localhost:10990 /]:reload
            [standalone@localhost:10990 /] exit
```

3. Start and configure the domain controller

3.1. Start the domain controller:

```
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB348/labs/troub-jgroups/machine1/ \
--host-config=host-master.xml
```

3.2. In a new terminal window on **workstation**, start the EAP CLI and connect to the domain controller:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh -c --controller=172.25.250.254:9990
```

3.3. Configure the **mod_cluster** subsystem. By default, EAP is set up to advertise its status to load balancers using UDP multicasting. Disable advertising in the **mod_cluster** subsystem for the ha profile.

```
[domain@172.25.250.254:9990 /] /profile=ha/subsystem=modcluster\
/mod-cluster-config=configuration/\
:write-attribute(name=advertise,value=false)
```

3.4. Because you have disabled advertising, you must configure the back-end EAP nodes with a list of **proxies** (load balancers). Configure the EAP back-end nodes to communicate with the load balancer running on the **workstation** VM. Add an outbound socket binding that points to the load balancer IP address and port (**172.25.250.254:9080**).

```
[domain@172.25.250.254:9990 /] /socket-binding-group=ha-sockets\
/remote-destination-outbound-socket-binding=lb:\
add(host=172.25.250.254,port=9080)
```

3.5. Next, add the proxies to the **mod_cluster** configuration:

```
[domain@172.25.250.254:9990 /] /profile=ha/subsystem=modcluster\
/mod-cluster-config=configuration:list-add(name=proxies,value=lb)
```

3.6. Deploy the **cluster.war** application:

```
[domain@172.25.250.254:9990 /] deploy /home/student/JB348/apps/cluster.war \
--server-groups=cluster-group
```

4. Start the two host controllers
   4.1. Open a terminal window from **workstation** VM and access the **servera** VM using the **ssh** command:

```
[student@workstation ~]$ ssh servera
```

   Start the host controller:

```
[student@servera ~]$ cd /opt/jboss-eap-7.0/bin
[student@servera bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB348/labs/troub-jgroups/machine2/ \
--host-config=host-slave.xml -Djboss.domain.master.address=172.25.250.254
```

   4.2. Open a terminal window from the **workstation** VM and access the **serverb** VM using the ssh command:

```
[student@workstation ~]$ ssh serverb
```

   Start the host controller:

```
[student@serverb ~]$ cd /opt/jboss-eap-7.0/bin
[student@serverb bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB348/labs/troub-jgroups/machine3/ \
--host-config=host-slave.xml -Djboss.domain.master.address=172.25.250.254
```

5. Test the cluster
   On the **workstation** VM, open a web browser and navigate to **http://172.25.250.254:9080/cluster**. You will see the **cluster** application. Refresh the page until the number of visitations increases and that all requests are being handled by the same node.

   If your requests are being handled by **cluster-one** server, press **Ctrl**+**C** in the terminal window from **servera**, or press **Ctrl**+**C** in the terminal window from **serverb** to stop the server that is handling the requests.

   Refresh the page again and notice that the cluster is not working properly. The number of visitations does not persist, as it would in a normally functioning cluster. Instead, the number of visits is reset to "1".

   Press **Ctrl**+**C** in the terminal window from the other node to stop the cluster.

6. Check that JGroup is not working
   The problem with the cluster is that the nodes are not communicating correctly due to misconfigured JGroups. JGroups, by default, communicates using multicast and multicast is not allowed in this environment.

   6.1. Open a new terminal window from **workstation** and access the **servera** VM.

```
[student@workstation ~]$ ssh servera
```

6.2. JGroups provides a tool to verify whether multicast is working, given an environment listening for all multicast messages. The communication between the nodes in a JBoss EAP cluster uses the **230.0.0.4** address on the **45688** port using the **udp** protocol. You should start a client that will listen for all messages that are being send to this address:

```
[student@servera ~]$ cd /opt/jboss-eap-7.0/modules/system
[student@servera system]$ cd layers/base/org/jgroups/main/
[student@servera main]$ java -cp jgroups-3.6.8.Final-redhat-2.jar \
org.jgroups.tests.McastReceiverTest -mcast_addr 230.0.0.4 -port 45688
```

6.3. Open a new terminal window on **workstation** to access the **servera** VM.

```
[student@workstation ~]$ ssh servera
```

6.4. Start a client that will write a message using multicast:

```
[student@servera ~]$ cd /opt/jboss-eap-7.0/modules/system
[student@servera system]$ cd layers/base/org/jgroups/main/
[student@servera main]$ java -cp jgroups-3.6.8.Final-redhat-2.jar \
org.jgroups.tests.McastSenderTest -mcast_addr 230.0.0.4 -port 45688
```

6.5. Enter your name and verify that it was displayed in the terminal waiting for the messages:

```
> Your Name
```

The multicast is working for the same node. Type **exit** to quit the client.

6.6. Access the **serverb** VM using the ssh command:

```
[student@workstation ~]$ ssh serverb
```

JGroups also provides a tool to send a message using multicast. Start a client that writes a message using the same address and port used by JBoss EAP:

```
[student@serverb ~]$ cd /opt/jboss-eap-7.0/modules/system
[student@serverb system]$ cd layers/base/org/jgroups/main/
[student@serverb main]$ java -cp jgroups-3.6.8.Final-redhat-2.jar \
org.jgroups.tests.McastSenderTest -mcast_addr 230.0.0.4 -port 45688
```

6.7. Enter your name and verify that it was **not** displayed in the terminal waiting for the messages. This test proves that the multicast is not working between **servera** and **serverb**.

7. Fix the cluster
   7.1. The cluster must define a new **TCP** stack configuration. **TCP**, unlike **UDP**, can use the more reliable unicast method of communication. Open the file **/tmp/new-tcp-stack.cli** on the **workstation** VM and review the commands listed in it.

   Edit the **initial_hosts** property and replace the values with the host name and ports of the two EAP server instances that are part of the cluster.

```
..."initial_hosts":add(value="servera[7600],serverb[7600]")
```

   7.2. Execute the EAP CLI script file on the domain controller.

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh --connect \
--controller=172.25.250.254:9990 --file=/tmp/new-tcp-stack.cli
```

8. Start the host controllers and test to verify that the nodes are able to communicate. After stopping one node, the number of visitations is not restarted.

9. Clean Up and Grading
   9.1. Run the following command from the workstation to grade the exercise:

```
[student@workstation ~]$ lab troub-jgroups grade
```

   9.2. Press **Ctrl**+**C** in each terminal windows where you started the cluster instances of EAP to stop the cluster.

   This concludes the guided exercise.

Deploying HA Singleton Applications

Objectives

After completing this section, students will able to configure and deploy a high availability application.

The need for cluster singletons

A **singleton** is a popular software development **design pattern** in which there is a single shared instance of an object for the whole application. The usual implementation for the singleton design pattern creates one instance for each cluster member. But this may have unintended consequences for an application, such as data consistency issues.

Sometimes the application logic requires that only one instance of an object is running for the whole cluster. A developer could extract this instance to run as a service that is deployed into a single non-clustered EAP instance that is accessed by a clustered application, but then the service would become a single point of failure.

Many developers believe a **`@Singleton`** EJB works correctly as a clustered singleton but this is not true by the EJB specification. It states that each JVM in a distributed application server gets one instance. All ways to implement a singleton using only standard JVM or JEE features creates one instance per JVM, that is, one instance for each EAP server in a cluster.

When an application needs high availability and must work across the cluster, developers and EAP administrators have to work together to extract the singleton parts from the clustered application. They must adapt the application to use of the following proprietary EAP 7 approaches:

- Singleton deployments (similar to the feature from EAP 5)
- Singleton services (enhanced version of the EAP 6 feature)

Both approaches allow an application to work as an active-passive HA application inside an EAP cluster, but they differ in configuration details and the changes required to application packaging and source code.

The singleton subsystem

Both singleton deployments and singleton services are managed by the **`singleton`** subsystem. It employs an Infinispan cache to register all known singleton deployments and singleton services and defines which cluster member runs each application. The EAP server instance that runs a singleton application is called the **master** server for that application.

The **`singleton`** subsystem can be configured with different election policies that define the master. Two kinds of election policies are provided by EAP 7:

- **`simple`**: the first node to join the cluster runs the singleton application.
- **`random`**: a random node is selected to run the singleton application.

An election policy can optionally specify a preferred server which, when available, will be the master for ALL singleton applications under that policy. The policy refers to the node name as specified by the **`jboss.node.name`** system property.

The default EAP 7 configuration files provide a **simple** election policy named **default** with no preferred server. This policy is shown by the following listing:

```
...
        <subsystem xmlns="urn:jboss:domain:singleton:1.0">
            <singleton-policies default="default">
                <singleton-policy name="default" cache-container="server">
                    <simple-election-policy/>
                </singleton-policy>
            </singleton-policies>
        </subsystem>
...
```

The following commands show how to create a new election policy named **custom**, of type **random**, which prefers the server named **servera1**:

```
/subsystem=singleton/singleton-policy=custom/election-policy=random:add()
```

```
/subsystem=singleton/singleton-policy=custom/election-policy=random:list-add(\
name=name-preferences, value=servera1)
```

If the master server fails, the **singleton** subsystem runs a new election for all singleton applications that were using the failed server as their master. These applications are then restarted in new master servers. Each singleton application has to provide their own means to recover or recreate in-memory data lost in the failed master server.

A potential issue with a singleton application is when there is a network partition, also known as the **split-brain** scenario: Two sets of servers from the same cluster cannot connect to each other, but servers from one set have no issue connecting to servers from the same set. Each set of servers think all servers from the other set failed and continue to work as a surviving cluster.

A split-brain scenario has two (or more) independent clusters where there should be a single cluster. This can quickly lead to data consistency issues. To prevent that, an election policy can specify a quorum; that is, the minimum required number of cluster members. If a quorum is not reached, all remaining cluster members are shut down.

To configure a quorum of three servers, use the following command:

```
/subsystem=singleton/singleton-policy=custom:write-attribute(name=quorum, value=3)
```

The quorum should be at least **N/2+1** where **N** is the anticipated total number of cluster members.

Singleton deployments

Singleton deployments are similar to **HASingletonDeployer** from EAP 5 and earlier. There was no similar feature in EAP 6. It is a way to mark a deployment as a cluster-wide singleton without the need to use proprietary EAP 7 APIs.

An application package is considered to be a singleton deployment if it contains the proprietary deployment descriptor **/META-INF/singleton-deployment.xml**. This file refers to the election policy to be used for the singleton application, and the following example refers to the **custom** policy from the previous example:

```
<singleton-deployment xmlns="urn:jboss:singleton-deployment:1.0" policy="custom"/>
```

Although this approach does not require an application to use EAP 7 proprietary APIs, it still requires the original clustered application to access the extracted singleton application, for example, using remote EJB calls or JMS.

Singleton services

Singleton services are implemented the same way as internal EAP services, that is, their source code uses WildFly Core APIs. Although more intrusive than singleton deployments, singleton services provide the following advantages in very specific use cases:

- The election policy can be defined by the application, without the need to configure the **`singleton`** subsystem.
- It allows EAP modules to start cluster-wide singleton services similar to the old JBossMQ from EAP 4. That is, it allows an EAP module to work as an active-passive clustered service.

Teaching students how to program a singleton service is outside the scope of this book.

References

For more information about HA Singletons, see the upstream documentation:

> Wildfly 10 HA Singleton Features
> https://docs.jboss.org/author/display/WFLY10/HA+Singleton+Features

For information about singleton EJBs and clustered environments:

> JSR-345: EJB 3.2 specification
> https://jcp.org/aboutJava/communityprocess/final/jsr345/index.html

For information about how to develop singleton services:

> EAP 7 Developing EJB Applications
> https://access.redhat.com/documentation/en/red-hat-jboss-enterprise-application-platform/7.0/developing-ejb-applications/developing-ejb-applications

Guided Exercise: Deploying an HA Singleton

In this exercise, you will implement deploy an HA Singleton.

Resources	
Files:	**/home/student/JB348/labs/ha-singleton /home/student/JB348/apps/information-desktop.jar / home/student/JB348/apps/information-singleton-ejb.jar**
Application URL:	NA

Outcomes
You should be able to configure a HA Singleton application in a cluster.

Before you begin
Use the following command to verify that an instance of EAP is installed in the **/opt/** directory and to download the server configuration files for this exercise:

```
[student@workstation ~]$ lab ha-singleton setup
```

1. Start the Domain Controller
   Open a terminal window from the workstation VM (**Applications** > **Favorites** > **Terminal**) and start the domain controller:

   ```
   [student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
   [student@workstation bin]$ ./domain.sh \
   -Djboss.domain.base.dir=/home/student/JB348/labs/ha-singleton/machine1/ \
   --host-config=host-master.xml
   ```

2. Configure the HA Singleton
   The **information-singleton-ejb.jar** application is available in just one instance from the cluster. However, it is preferred for applications to have high availability, meaning that if a node that is handling this application dies, then other nodes star serving the application automatically.

   2.1. The application is configured to request an election policy named custom. Open a new terminal window and connect to CLI to create the election policy.

   ```
   [student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
   [student@workstation ~]$ ./jboss-cli.sh --connect --user=jbossadm \
   --password=JBoss@RedHat123 --controller=172.25.250.254:9990
   ```

   2.2. Create a new election policy named **custom**. The election policy uses a cache container named **server**. This cache is responsible for registering singleton provider candidates.

   ```
   [domain@172.25.250.254:9990 /] /profile=ha/subsystem=singleton/ \
   singleton-policy=custom:add(cache-container=server)
   ```

2.3. The election policy must be configured to select a random node to run the singleton application. Configure the **custom** election policy to achieve this goal.

```
[domain@172.25.250.254:9990 /] /profile=ha/subsystem=singleton/ \
singleton-policy=custom/election-policy=random:add()
```

2.4. It is possible to define a preferred server to run the singleton application. Configure the election policy to prefer the **cluster-two** server:

```
[domain@172.25.250.254:9990 /] /profile=ha/subsystem=singleton/\
singleton-policy=custom/election-policy=random\
:list-add(name=name-preferences, value=cluster-two)
```

With this configuration, if the **cluster-two** server is available on the cluster, it is responsible for handling new requests for the application.

3. Deploy the HA-singleton Application
Deploy the **/home/student/JB348/apps/information-singleton-ejb.jar** application into the **cluster-group** server group:

```
[domain@172.25.250.254:9990 /] deploy \
/home/student/JB348/apps/information-singleton-ejb.jar \
--server-groups=cluster-group
```

Exit the CLI application:

```
[domain@172.25.250.254:9990 /] exit
```

4. Configuring security
In EAP 7, a lookup to a remote EJB requires authentication. The **information-desktop.jar** client application is configured to authenticate using the **appsingleton** user name with password defined as **JBoss@RedHat123**.

Create credentials for the **host2** node:

```
[student@workstation bin]$ ./add-user.sh -a -u appsingleton -p JBoss@RedHat123 \
-dc /home/student/JB348/labs/ha-singleton/machine2/configuration/
```

Also create the credentials for the **host3** node:

```
[student@workstation bin]$ ./add-user.sh -a -u appsingleton -p JBoss@RedHat123 \
-dc /home/student/JB348/labs/ha-singleton/machine3/configuration/
```

5. Start the Host Controllers
5.1. Start **host2** using the **host-slave.xml** configuration file that has its management interface bound to **172.25.250.254**.

```
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB348/labs/ha-singleton/machine2/ \
```

```
--host-config=host-slave.xml -Djboss.domain.master.address=172.25.250.254
```

Wait until the startup process finishes. The log output shows that this node is selected to serve the singleton:

```
...OUTPUT OMITED...
[Server:cluster-one] 00:23:40,092 INFO  [org.wildfly.clustering.server]
 (notification-thread--p1-t1) WFLYCLSV0001: This node will now operate as
 the singleton provider of the jboss.deployment.unit."information-singleton-
ejb.jar".FIRST_MODULE_USE service
...OUTPUT OMITED...
```

5.2. Start **host3** using the **host-slave.xml** configuration file that has its management interface bind to **172.25.250.254**.

Run the following command from your **/opt/jboss-eap-7.0/bin** folder in a new terminal window:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./domain.sh \
-Djboss.domain.base.dir=/home/student/JB348/labs/ha-singleton/machine3/ \
--host-config=host-slave.xml -Djboss.domain.master.address=172.25.250.254
```

Wait until the startup process finishes. The log output shows that this node is now selected to serve the singleton. The reason for the change is that **host3** is the host controller responsible for the **cluster-two** server. The cluster identified that a preferred server is available on the cluster and changed the node that will operate as the singleton provider.

Also observe in the log output from **host2** that the singleton will no longer operate as the singleton provider.

```
...OUTPUT OMITED...
[Server:cluster-one] 00:30:44,031 INFO  [org.wildfly.clustering.server]
 (thread-17,ee,host2:cluster-one) WFLYCLSV0002: This node will no longer operate
 as the singleton provider of the jboss.deployment.unit."information-singleton-
ejb.jar".FIRST_MODULE_USE service
...OUTPUT OMITED...
```

6. Test the Singleton Using a Client

6.1. The **information-desktop.jar** client application requests informations about the HA-singleton application and displays on the screen. Open a terminal window from the workstation VM and run the application:

```
[student@workstation ~]$ java -jar \
/home/student/JB348/apps/information-desktop.jar
```

The following output is expected:

```
################################################################################
# Base dir: /home/student/JB348/labs/ha-singleton/machine3/servers/cluster-two
# Node name: host3:cluster-two
# Host name: workstation
```

```
################################################################################
```

As expected, the **cluster-two** server was responsible for the request.

6.2. Run the application multiple times. All responses should have the same information.

6.3. Stop the **host3** pressing **Ctrl**+**C** in the terminal window that started the host.

6.4. Run the client application. Since the **cluster-two** server is not more available, the **cluster-one** is responsible for handling the new requests.

Note

The client application is configured to load balancer the requests between nodes. Since you stopped the **host3**, you should see the following exception before the information:

```
WARN: Could not register a EJB receiver for connection to
 172.25.250.254:8180
java.lang.RuntimeException: java.net.ConnectException: Connection
 refused
```

7. Clean Up and Grading

7.1. Run the following command from the workstation to grade the exercise:

```
[student@workstation ~]$ lab ha-singleton grade
```

7.2. Press **Ctrl**+**C** in the terminal windows where you started the cluster instances of EAP to stop the cluster.

This concludes the guided exercise.

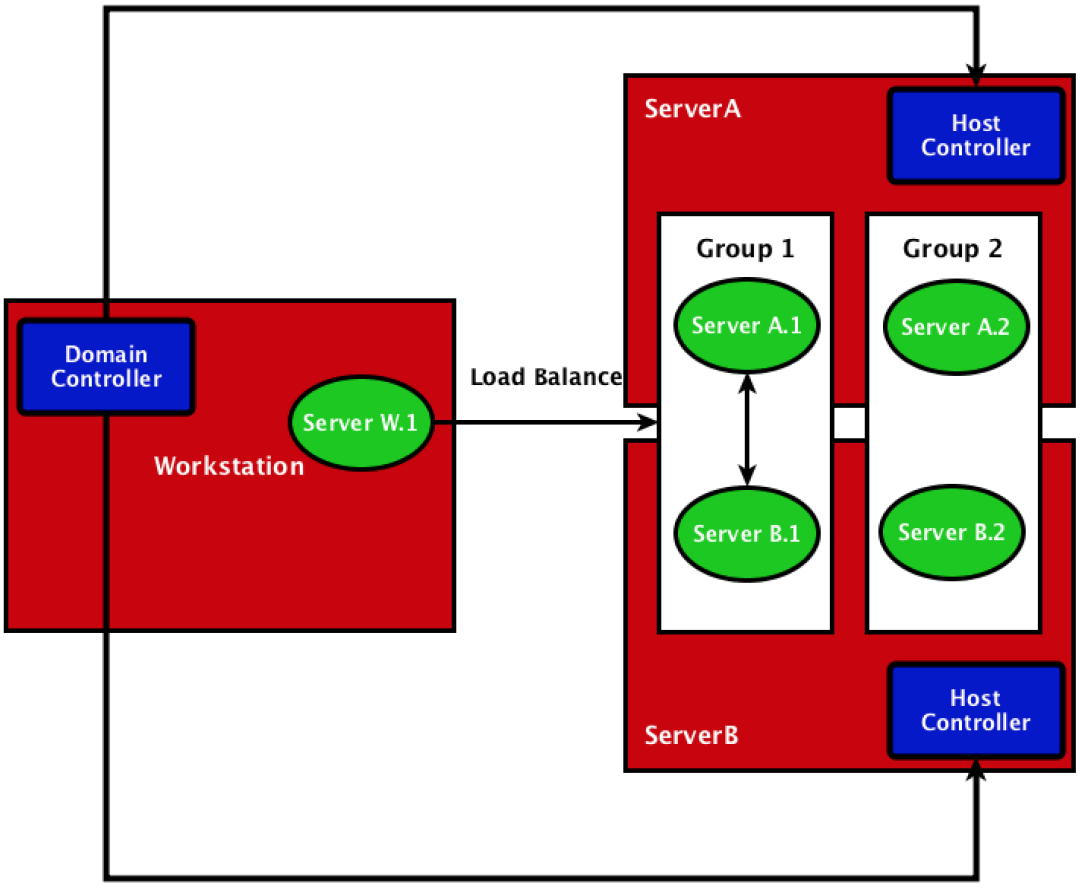
Lab: Configuring a JBoss EAP Cluster

In this lab, you will create a managed domain and a load balancer.

Resources	
Files:	`/home/student/JB348/labs/create-cluster`, `/home/student/JB348/apps/cluster.war`
Application URL:	`http://localhost:9990`, `http://localhost:8080/version`

Outcomes

You should be able to start a domain controller and a load balancer on the **`workstation`** VM and run a host controller on both **`server A`** and **`server B`**. The final solution should be in line with the following architecture with regards to the host controller and the domain controller.



Before you begin

Use the following command to verify that an instance of EAP is installed in the **`/opt/`** directory and to download the required files for this exercise:

```
[student@workstation ~]$ lab configure-cluster setup
```

1. To get started creating the domain controller, copy files from **JBOSS_HOME/domain** into the lab directory at **/opt/domain** on the workstation. Set the owner of the **/opt/domain** directory to user **jboss**.

2. The EAP instance on **workstation** is the domain controller, and exposes the management interface to an internal network. The network where all servers are connected to is the **172.25.250.X** network. Other network interfaces should not be exposed to guarantee that external host controllers may not get sensitive information from the domain controller.

   Update the address of the management interface for the domain controller to point to the workstation's IP address (**172.25.250.254**) using the **host-master.xml** configuration file. Remember to unlock the **9990/tcp** and **9999/tcp** ports to allow the communication between the domain controller and the host controllers:

3. Because the domain controller will run an application with high availability features, including the messaging subsystem, some extra requirements are needed. The domain controller is responsible for providing security information to allow server instances to be part of a clustered environment, for all subsystems.

   The messaging subsystem requires authentication to exchange data among cluster members, and the credentials are declared in the **domain.xml** file. The details about the subsystem are explained later, but for the environment to function correctly, the cluster password must be declared at the domain controller configuration file.

   Update **domain.xml** on **workstation** to set the cluster password to **JBoss@RedHat123**.

   3.1. As the **jboss** user, open the **/opt/domain/configuration/domain.xml** file on **workstation** .

   3.2. Inside of the **messaging-activemq** subsystem of the full-ha profile, edit the **<cluster>** tag (line 1278).

   3.3. Save your changes to **domain.xml**.

4. Start the domain controller on the workstation.

   4.1. In your terminal window, change directories to **/opt/jboss-eap-7.0/bin**:

   4.2. To start the domain controller using the **host-master.xml** file in your **/opt/domain/** folder.

5. An instance of EAP is already installed on both **server A** and **server B**. Create a new host controller with the following characteristics using the **/home/student/JB348/labs/configure-cluster/create-hc.sh** script file:

   - The base directory on **server A** is **/opt/domain** and sets the owner as **jboss**.
   - Set the name of the host to **servera**.
   - Use the IP address of Server A, **172.25.250.10**, for the management, public, and private interface.
   - Set the host controller to connect to the domain controller using the login **jbossadm** and the password **JBoss@RedHat123**. You can generate the password using the base64 algorithm with the following command:

```
[student@workstation ~]$ echo -n JBoss@RedHat123 | openssl base64
```

6. Start the host controller on **servera** with the configuration file **/opt/domain/configuration/host-slave.xml** and point to the domain controller running on **172.25.250.254**.

7. Create a new host controller with the following characteristics using the **/home/student/JB348/labs/configure-cluster/create-hc.sh** script file:

   - The base directory on **serverb** is **/opt/domain** and should set the owner as **jboss**.
   - Set the name of the host to **serverb**.
   - Use IP address of **serverb** (**172.25.250.11**) for the management, public, and private interface.
   - Set the host controller to connect to the domain controller using the login **jbossadm** and the password **JBoss@RedHat123**. You can generate the password using the base64 algorithm with the following command:

     ```
     [student@workstation ~]$ echo -n JBoss@RedHat123 | openssl base64
     ```

8. Start the host controller on **serverb** with the configuration file **/opt/domain/configuration/host-slave.xml** and point to the domain controller running on **172.25.250.254**.

9. Stop and remove all of the servers (**server-one**, **server-two**) using the **/home/student/JB348/labs/configure-cluster/delete-server.sh** script. In the next steps, you will create new servers to deploy applications on these hosts.

   9.1. Stop and remove the **server-one** server from **servera** host.

   9.2. Stop and remove the **server-two** server from **servera** host.

   9.3. Stop and remove the **server-one** server from **serverb** host.

   9.4. Stop and remove the **server-two** server from **serverb** host.

10. Remove the server-groups (**main-server-group**, **other-server-group**) using the **/home/student/JB348/labs/configure-cluster/delete-server-group.sh** script.

11. The cluster must define a new **TCP** stack configuration. The communication between the nodes uses unicast. For this environment, unicast is a requirement, since it is more reliable. Edit the file **/tmp/new-tcp-stack.cli** on the **workstation** VM and configure the cluster to use **TCP**.

12. The cluster provides the messaging subsystem and HA capabilities from EAP, and these features are available only in the **full-ha** profile. Create a new server group using the **/home/student/JB348/labs/configure-cluster/create-server-group.sh** script called **Group1** which is used to emulate a development environment with the following characteristics:

- **Name**: `Group1`
- **Profile**: `full-ha`
- **Socket Binding Group**: `full-ha-sockets`

13. Create another server group called **`Group2`** to use for deploying **bookstore** in a production environment with the same characteristics as the development environment:
    - **Name**: `Group2`
    - **Profile**: `full-ha`
    - **Socket Binding Group**: `full-ha-sockets`
14. You need to define four servers (two each on **`servera`** and **`serverb`**). Create the servers and assign them to the appropriate groups. Make sure that when you run multiple servers on a host, you configure the port offsets correctly to avoid port clashes. Also ensure that the servers are set to automatically start when the host controller is started or restarted. To create the servers, use the **`/home/student/JB348/labs/configure-cluster/create-server.sh`** script.
    14.1. Create and start a new server called **`servera.1`** on **`servera`** with the following characteristics:
    - **Name**: `servera.1`
    - **Server Group**: `Group1`
    - **Socket Binding Port Offset**: `Leave at default value of 0`
    - **Auto Start**: `true`

    14.2. Create and start a new server called **`servera.2`** on **`servera`** with the following characteristics:
    - **Name**: `servera.2`
    - **Server Group**: `Group2`
    - **Socket Binding Port Offset**: `100`
    - **Auto Start**: `true`

    14.3. Create and start a new server called **`serverb.1`** on **`serverb`** with the following characteristics:
    - **Name**: `serverb.1`
    - **Server Group**: `Group1`
    - **Socket Binding Port Offset**: `Leave at default value of 0`
    - **Auto Start**: `true`

14.4. Create and start a new server called **serverb.2** on **serverb** with the following characteristics:

- **Name**: **serverb.2**
- **Server Group**: **Group2**
- **Socket Binding Port Offset**: **100**
- **Auto Start**: **true**

15. You have now created four servers and associated them with the appropriate server groups. Because a firewall is running on **servera** and **serverb**, you must unlock all ports used by **servera.1**, **servera.2**, **serverb.1**, and **serverb.2** for public access. Unlock the following ports on **servera** and **serverb** using the **/home/student/JB348/labs/configure-cluster/firewall.sh** script:
    - 8080/tcp
    - 8180/tcp
    - 8009/tcp
    - 8109/tcp
    - 7600/tcp
    - 7700/tcp
    - 57600/tcp
    - 57700/tcp
16. To get started creating the load balancer, copy files from **JBOSS_HOME/standalone** into the lab directory at **/opt/lb** on the workstation. Set the owner of the **/opt/lb** directory to user **jboss**.
17. Start the load balancer on **workstation** with the **standalone-ha.xml** and a port offset of **1000** to avoid port clashes with the domain controller running on the **workstation**. Because the load balancer is going to be the entry point for the application, it must be configured to listen on the public IP of the **workstation** (172.25.250.254). Remember to unlock the **9080/tcp** port on the firewall.
18. Configure the load balancer for balancing requests using the **modcluster**. Set the password for the **advertise-security-key** to **redhat**.
19. Configure the **modcluster** subsystem from the **full-ha** profile to enable the load balancer. You must disable the advertise option and also configure the proxies list pointing to the load balancer address (**172.25.250.254:9080**).
20. You are now ready to deploy the **cluster** application WAR file in the managed domain. The **cluster.war** file is available in the **/home/student/JB348/apps** folder on the **workstation**.

20.1. Application deployments in a managed domain are always done at the server groups level. Each application belongs to one or more server groups. Deploy the **cluster.war** file to **Group1**.

Verify that the **cluster.war** file is deployed on **servera.1** and **serverb.1** because they are part of the **Group1** server group. Monitor the console window of **servera** and **serverb** for any error messages or warnings.

20.2. Verify that you can access the **cluster** application at **http://172.25.250.10:8080/cluster** and **http://172.25.250.11:8080/cluster**.

20.3. Verify that you can **NOT** access the **cluster** application at **http://172.25.250.10:8180/cluster** or **http://172.25.250.11:8180/cluster** because these servers are part of **Group2** and you have not deployed the application on **Group2**.

20.4. Verify that you can access the **cluster** application at **http://172.25.250.254:9080/cluster**. This access are being handled by the load balancer.

21. Clean Up and Grading

21.1. Press **Ctrl**+**C** in the terminal windows where you started the cluster instances of EAP to stop the cluster.

21.2. Run the following command from the **workstation** to grade the exercise:

```
[student@workstation ~]$ lab configure-cluster grade
```

This concludes the lab.

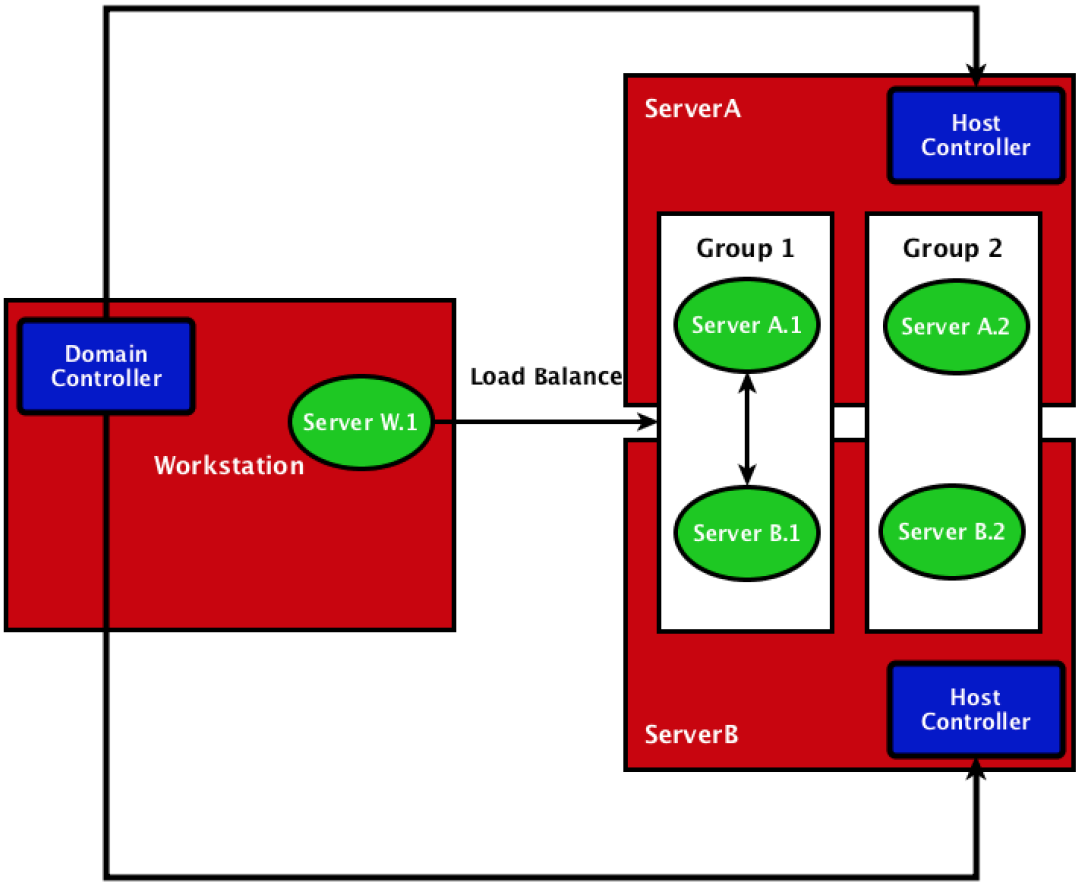
Solution

In this lab, you will create a managed domain and a load balancer.

Resources	
Files:	`/home/student/JB348/labs/create-cluster`, `/home/student/JB348/apps/cluster.war`
Application URL:	`http://localhost:9990`, `http://localhost:8080/version`

Outcomes

You should be able to start a domain controller and a load balancer on the **`workstation`** VM and run a host controller on both **`server A`** and **`server B`**. The final solution should be in line with the following architecture with regards to the host controller and the domain controller.



Before you begin

Use the following command to verify that an instance of EAP is installed in the **`/opt/`** directory and to download the required files for this exercise:

```
[student@workstation ~]$ lab configure-cluster setup
```

1. To get started creating the domain controller, copy files from **JBOSS_HOME/domain** into the lab directory at **/opt/domain** on the workstation. Set the owner of the **/opt/domain** directory to user **jboss**.

   1.1. Run the following command to copy the EAP domain configuration to the **/opt/domain** directory on the **workstation**.

   ```
   [student@workstation ~]$ sudo cp -r /opt/jboss-eap-7.0/domain /opt/
   ```

   1.2. Use the following command to set the directory owner as user **jboss**:

   ```
   [student@workstation ~]$ sudo chown -R jboss:jboss /opt/domain
   ```

2. The EAP instance on **workstation** is the domain controller, and exposes the management interface to an internal network. The network where all servers are connected to is the **172.25.250.X** network. Other network interfaces should not be exposed to guarantee that external host controllers may not get sensitive information from the domain controller.

   Update the address of the management interface for the domain controller to point to the workstation's IP address (**172.25.250.254**) using the **host-master.xml** configuration file. Remember to unlock the **9990/tcp** and **9999/tcp** ports to allow the communication between the domain controller and the host controllers:

   2.1. Open the file **/opt/domain/configuration/host-master.xml** with a text editor as user **jboss**.

   2.2. Modify the interface sections of the configuration file as follows:

   ```
   <interfaces>
       <interface name="management">
           <inet-address value="${jboss.bind.address.management:172.25.250.254}"/>
       </interface>
   </interfaces>
   ```

   2.3. Save your changes and exit the editor.

   2.4. To unlock the firewall ports on **workstation**, run the following commands in a terminal window:

   ```
   [student@workstation ~]$ cd /home/student/JB348/labs/configure-cluster
   [student@workstation configure-cluster]$ ./firewall.sh workstation 9990/tcp \
   9999/tcp
   ```

3. Because the domain controller will run an application with high availability features, including the messaging subsystem, some extra requirements are needed. The domain controller is responsible for providing security information to allow server instances to be part of a clustered environment, for all subsystems.

   The messaging subsystem requires authentication to exchange data among cluster members, and the credentials are declared in the **domain.xml** file. The details about the subsystem are explained later, but for the environment to function correctly, the cluster password must be declared at the domain controller configuration file.

Update **domain.xml** on **workstation** to set the cluster password to **JBoss@RedHat123**.

3.1. As the **jboss** user, open the **/opt/domain/configuration/domain.xml** file on **workstation** .

```
[student@workstation ~]$ sudo -u jboss vi /opt/domain/configuration/domain.xml
```

3.2. Inside of the **messaging-activemq** subsystem of the full-ha profile, edit the **<cluster>** tag (line 1278).

```
<cluster password="${jboss.messaging.cluster.password:JBoss@RedHat123}" />
```

3.3. Save your changes to **domain.xml**.

4. Start the domain controller on the workstation.

4.1. In your terminal window, change directories to **/opt/jboss-eap-7.0/bin**:

```
[student@workstation domain]$ cd /opt/jboss-eap-7.0/bin
```

4.2. To start the domain controller using the **host-master.xml** file in your **/opt/domain/** folder.

Enter the following command:

```
[student@workstation ~]$ sudo -u jboss ./domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
--host-config=host-master.xml
```

Look for the following output to confirm that the domain controller is running:

```
[Host Controller] 01:50:11,359 INFO  [org.jboss.as] (Controller Boot Thread)
 WFLYSRV0060: Http management interface listening on http://172.25.250.254:9990/
management
[Host Controller] 01:50:11,359 INFO  [org.jboss.as] (Controller Boot Thread)
 WFLYSRV0051: Admin console listening on http://172.25.250.254:9990
```

5. An instance of EAP is already installed on both **server A** and **server B**. Create a new host controller with the following characteristics using the **/home/student/JB348/labs/configure-cluster/create-hc.sh** script file:

- The base directory on **server A** is **/opt/domain** and sets the owner as **jboss**.
- Set the name of the host to **servera**.
- Use the IP address of Server A, **172.25.250.10**, for the management, public, and private interface.
- Set the host controller to connect to the domain controller using the login **jbossadm** and the password **JBoss@RedHat123**. You can generate the password using the base64 algorithm with the following command:

```
[student@workstation ~]$ echo -n JBoss@RedHat123 | openssl base64
```

Use the following commands on **workstation** to create the new host controller:

```
[student@workstation ~]$ cd /home/student/JB348/labs/configure-cluster
[student@workstation configure-cluster]$ ./create-hc.sh 172.25.250.10 /opt/domain \
servera jboss 172.25.250.10 172.25.250.10 172.25.250.10 jbossadm \
SkJvc3NAUmVkSGF0MTIz
```

6. Start the host controller on **servera** with the configuration file **/opt/domain/configuration/host-slave.xml** and point to the domain controller running on **172.25.250.254**.

   6.1. Access the **servera** VM using the **ssh** command:

   ```
   [student@workstation ~]$ ssh servera
   ```

   6.2. Run the following command to start the host controller and connect to the domain controller:

   ```
   [student@servera ~]$ cd /opt/jboss-eap-7.0/bin
   [student@servera bin]$ sudo -u jboss ./domain.sh \
   -Djboss.domain.base.dir=/opt/domain/ \
   --host-config=host-slave.xml \
   -Djboss.domain.master.address=172.25.250.254 \
   -Djboss.node.name=servera
   ```

   6.3. Look in the terminal window of the host controller on **servera**. Carefully review the log output and you should see the host controller connecting to the master, and also **server-one** and **server-two** starting up.

   ```
   [Host Controller] 16:42:57,307 INFO  [org.jboss.as.host.controller]
    (Controller Boot Thread) WFLYHC0148: Connected to master host controller at
    remote://172.25.250.254:9999
   [Host Controller] 16:42:57,367 INFO  [org.jboss.as.host.controller] (Controller
    Boot Thread) WFLYHC0023: Starting server server-one
   ```

   6.4. Look in the terminal window of the domain controller running on the workstation. You should see a log entry showing the slave connecting:

   ```
   [Host Controller] 11:42:16,348 INFO  [org.jboss.as.domain.controller] (Host
    Controller Service Threads - 36) WFLYHC0019: Registered remote slave host
    "servera", JBoss JBoss EAP 7.0.0.GA (WildFly 2.1.2.Final-redhat-1)
   ```

7. Create a new host controller with the following characteristics using the **/home/student/JB348/labs/configure-cluster/create-hc.sh** script file:

   - The base directory on **serverb** is **/opt/domain** and should set the owner as **jboss**.
   - Set the name of the host to **serverb**.

- Use IP address of **serverb** (**172.25.250.11**) for the management, public, and private interface.
- Set the host controller to connect to the domain controller using the login **jbossadm** and the password **JBoss@RedHat123**. You can generate the password using the base64 algorithm with the following command:

```
[student@workstation ~]$ echo -n JBoss@RedHat123 | openssl base64
```

Use the following commands on **workstation** to create the new host controller:

```
[student@workstation ~]$ cd /home/student/JB348/labs/configure-cluster
[student@workstation configure-cluster]$ ./create-hc.sh 172.25.250.11 /opt/domain \
serverb jboss 172.25.250.11 172.25.250.11 172.25.250.11 jbossadm \
SkJvc3NAUmVkSGF0MTIz
```

8. Start the host controller on **serverb** with the configuration file **/opt/domain/configuration/host-slave.xml** and point to the domain controller running on **172.25.250.254**.

   8.1. Access the **serverb** VM using the **ssh** command:

```
[student@workstation configure-cluster]$ ssh serverb
```

   8.2. Run the following command to start the host controller and connect to the domain controller:

```
[student@serverb ~]$ cd /opt/jboss-eap-7.0/bin
[student@serverb bin]$ sudo -u jboss ./domain.sh \
-Djboss.domain.base.dir=/opt/domain/ \
--host-config=host-slave.xml \
-Djboss.domain.master.address=172.25.250.254 \
-Djboss.node.name=serverb
```

   8.3. Look in the terminal window of the host controller of **serverb**. Carefully review the log output and you should see the host controller connecting to the master.

```
[Host Controller] 16:42:57,307 INFO  [org.jboss.as.host.controller]
 (Controller Boot Thread) WFLYHC0148: Connected to master host controller at
 remote://172.25.250.254:9999
```

   8.4. Look in the terminal window of the domain controller running on **workstation**. You should see a log entry showing the slave connecting:

```
[Host Controller] 11:42:16,348 INFO  [org.jboss.as.domain.controller] (Host
 Controller Service Threads - 36) WFLYHC0019: Registered remote slave host
 "serverb", JBoss JBoss EAP 7.0.0.GA (WildFly 2.1.2.Final-redhat-1)
```

9. Stop and remove all of the servers (**server-one**, **server-two**) using the **/home/student/JB348/labs/configure-cluster/delete-server.sh** script. In the next steps, you will create new servers to deploy applications on these hosts.

9.1. Stop and remove the **server-one** server from **servera** host.

```
[student@workstation bin]$ cd /home/student/JB348/labs/configure-cluster
[student@workstation configure-cluster]$ ./delete-server.sh servera server-one
```

9.2. Stop and remove the **server-two** server from **servera** host.

```
[student@workstation configure-cluster]$ ./delete-server.sh servera server-two
```

9.3. Stop and remove the **server-one** server from **serverb** host.

```
[student@workstation configure-cluster]$ ./delete-server.sh serverb server-one
```

9.4. Stop and remove the **server-two** server from **serverb** host.

```
[student@workstation configure-cluster]$ ./delete-server.sh serverb server-two
```

10. Remove the server-groups (**main-server-group**, **other-server-group**) using the **/home/student/JB348/labs/configure-cluster/delete-server-group.sh** script.

10.1. Run the following command to remove **main-server-group** on **workstation**:

```
[student@workstation configure-cluster]$ ./delete-server-group.sh \
main-server-group
```

10.2. Run the following command to remove **other-server-group** on **workstation**:

```
[student@workstation configure-cluster]$ ./delete-server-group.sh \
other-server-group
```

11. The cluster must define a new **TCP** stack configuration. The communication between the nodes uses unicast. For this environment, unicast is a requirement, since it is more reliable. Edit the file **/tmp/new-tcp-stack.cli** on the **workstation** VM and configure the cluster to use **TCP**.

11.1. Edit the **initial_hosts** property and replace the values with the host name and ports of the two EAP server instances that should be part of the cluster.

```
..."initial_hosts":add(value="servera[7600],servera[7700],serverb[7600],
 serverb[7700]")
```

Note

The **7600** port is used by JGroups to allow the communication between nodes.

11.2. Execute the EAP CLI script file on the domain controller.

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ sudo -u jboss ./jboss-cli.sh --connect \
--controller=172.25.250.254:9990 --file=/tmp/new-tcp-stack.cli
```

12. The cluster provides the messaging subsystem and HA capabilities from EAP, and these features are available only in the **full-ha** profile. Create a new server group using the **/home/student/JB348/labs/configure-cluster/create-server-group.sh** script called **Group1** which is used to emulate a development environment with the following characteristics:

    - **Name**: `Group1`
    - **Profile**: `full-ha`
    - **Socket Binding Group**: `full-ha-sockets`

```
[student@workstation ~]$ cd /home/student/JB348/labs/configure-cluster/
[student@workstation configure-cluster]$ ./create-server-group.sh Group1 \
full-ha full-ha-sockets
```

13. Create another server group called **Group2** to use for deploying **bookstore** in a production environment with the same characteristics as the development environment:

    - **Name**: `Group2`
    - **Profile**: `full-ha`
    - **Socket Binding Group**: `full-ha-sockets`

```
[student@workstation configure-cluster]$ ./create-server-group.sh Group2 \
full-ha full-ha-sockets
```

14. You need to define four servers (two each on **servera** and **serverb**). Create the servers and assign them to the appropriate groups. Make sure that when you run multiple servers on a host, you configure the port offsets correctly to avoid port clashes. Also ensure that the servers are set to automatically start when the host controller is started or restarted. To create the servers, use the **/home/student/JB348/labs/configure-cluster/create-server.sh** script.

    14.1. Create and start a new server called **servera.1** on **servera** with the following characteristics:

    - **Name**: `servera.1`
    - **Server Group**: `Group1`
    - **Socket Binding Port Offset**: `Leave at default value of 0`
    - **Auto Start**: `true`

```
[student@workstation configure-cluster]$ ./create-server.sh servera \
servera.1 Group1 0 true
```

14.2. Create and start a new server called **servera.2** on **servera** with the following characteristics:

- **Name**: servera.2
- **Server Group**: Group2
- **Socket Binding Port Offset**: 100
- **Auto Start**: true

```
[student@workstation configure-cluster]$ ./create-server.sh servera \
servera.2 Group2 100 true
```

14.3. Create and start a new server called **serverb.1** on **serverb** with the following characteristics:

- **Name**: serverb.1
- **Server Group**: Group1
- **Socket Binding Port Offset**: Leave at default value of 0
- **Auto Start**: true

```
[student@workstation configure-cluster]$ ./create-server.sh serverb \
serverb.1 Group1 0 true
```

14.4. Create and start a new server called **serverb.2** on **serverb** with the following characteristics:

- **Name**: serverb.2
- **Server Group**: Group2
- **Socket Binding Port Offset**: 100
- **Auto Start**: true

```
[student@workstation configure-cluster]$ ./create-server.sh serverb \
serverb.2 Group2 100 true
```

15. You have now created four servers and associated them with the appropriate server groups. Because a firewall is running on **servera** and **serverb**, you must unlock all ports used by **servera.1**, **servera.2**, **serverb.1**, and **serverb.2** for public access. Unlock the following ports on **servera** and **serverb** using the **/home/student/JB348/labs/configure-cluster/firewall.sh** script:

- 8080/tcp
- 8180/tcp
- 8009/tcp

- 8109/tcp
- 7600/tcp
- 7700/tcp
- 57600/tcp
- 57700/tcp

15.1. Unlock the ports on **servera**:

```
[student@workstation configure-cluster]$ ./firewall.sh servera 8080/tcp \
8180/tcp 8009/tcp 8109/tcp 7600/tcp 7700/tcp 57600/tcp 57700/tcp
```

15.2. Unlock the ports on **serverb**

```
[student@workstation configure-cluster]$ ./firewall.sh serverb 8080/tcp \
8180/tcp 8009/tcp 8109/tcp 7600/tcp 7700/tcp 57600/tcp 57700/tcp
```

15.3. Verify that you can access the default welcome page of all four servers you created in the above steps. Access the following URLs and verify that you can see the EAP 7 Welcome page:

- **servera.1**: **http://172.25.250.10:8080**
- **servera.2**: **http://172.25.250.10:8180**
- **serverb.1**: **http://172.25.250.11:8080**
- **serverb.2**: **http://172.25.250.11:8180**

16. To get started creating the load balancer, copy files from **JBOSS_HOME/standalone** into the lab directory at **/opt/lb** on the workstation. Set the owner of the **/opt/lb** directory to user **jboss**.

16.1. Run the following command to copy the EAP domain configuration to the **/opt/lb** directory on the **workstation**.

```
[student@workstation ~]$ sudo cp -r /opt/jboss-eap-7.0/standalone /opt/lb
```

16.2. Use the following command to set the directory owner as user **jboss**:

```
[student@workstation ~]$ sudo chown -R jboss:jboss /opt/lb
```

17. Start the load balancer on **workstation** with the **standalone-ha.xml** and a port offset of **1000** to avoid port clashes with the domain controller running on the **workstation**. Because the load balancer is going to be the entry point for the application, it must be configured to listen on the public IP of the **workstation** (172.25.250.254). Remember to unlock the **9080/tcp** port on the firewall.

17.1. To unlock the firewall port on **workstation**, run the following commands in a new terminal window from **workstation**:

```
[student@workstation ~]$ cd /home/student/JB348/labs/configure-cluster
[student@workstation configure-cluster]$ ./firewall.sh workstation 9080/tcp
```

17.2. Run the following command to start the load balancer:

```
[student@workstation configure-cluster]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ sudo -u jboss ./standalone.sh \
-Djboss.server.base.dir=/opt/lb -Djboss.bind.address=172.25.250.254 \
-Djboss.socket.binding.port-offset=1000 -c standalone-ha.xml
```

18. Configure the load balancer for balancing requests using the **modcluster**. Set the password for the **advertise-security-key** to **redhat**.

18.1. Launch the EAP CLI in a new terminal window and connect to the load balancer instance on workstation.

In a new terminal window on the **workstation**, start the EAP CLI and connect to the load balancer:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh --connect --controller=localhost:10990
```

18.2. Configure the **modcluster** subsystem to act as a front-end load balancer. Set the password for the **advertise-security-key** to **redhat**.

```
[standalone@localhost:10990 /] /subsystem=modcluster/mod-cluster-config=\
configuration:write-attribute\
(name=advertise-security-key, value=redhat)
```

18.3. Configure the **mod_cluster** filter. Advertise the load balancer using the **modcluster** socket-binding and use the HTTP protocol for the management socket binding. Ensure that the **security-key** attribute matches the **advertise-security-key** you configured in the previous step.

```
[standalone@localhost:10990 /] /subsystem=undertow/configuration=\
filter/mod-cluster=modcluster:add\
(management-socket-binding=http, advertise-socket-binding=modcluster,\
security-key=redhat)
```

18.4. As a final step, bind the **modcluster** filter to the undertow **default-server**.

```
[standalone@localhost:10990 /] /subsystem=undertow/server=\
default-server/host=default-host/filter-ref=modcluster:add
```

18.5. Reload the load balancer configuration and exit from CLI:

```
[standalone@localhost:10990 /] :reload
```

```
[standalone@localhost:10990 /] exit
```

19. Configure the **modcluster** subsystem from the **full-ha** profile to enable the load balancer. You must disable the advertise option and also configure the proxies list pointing to the load balancer address (**172.25.250.254:9080**).

    19.1. In a new terminal window on the **workstation**, start the EAP CLI and connect to the domain controller:

```
[student@workstation ~]$ cd /opt/jboss-eap-7.0/bin
[student@workstation bin]$ ./jboss-cli.sh -c --controller=172.25.250.254:9990
```

    19.2. Configure the **mod_cluster** subsystem. By default, EAP is set up for advertising its status to load balancers using UDP multicasting. Disable advertising in the **mod_cluster** subsystem for the **full-ha** profile.

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=modcluster\
/mod-cluster-config=configuration/\
:write-attribute(name=advertise,value=false)
```

    19.3. Because you have disabled advertising, you need to configure the back-end EAP nodes with a list of **proxies** (load balancers). Configure the EAP back-end nodes to communicate with the load balancer running on the **workstation** VM. Ensure you add an outbound socket binding that points to the load balancer IP address and port (**172.25.250.254:9080**).

```
[domain@172.25.250.254:9990 /] /socket-binding-group=full-ha-sockets\
/remote-destination-outbound-socket-binding=lb:\
add(host=172.25.250.254,port=9080)
```

    19.4. Next, add the proxies to the **mod_cluster** configuration:

```
[domain@172.25.250.254:9990 /] /profile=full-ha/subsystem=modcluster\
/mod-cluster-config=configuration:list-add(name=proxies,value=lb)
```

    19.5. Reload the hosts:

```
[domain@172.25.250.254:9990 /] reload --host=servera
[domain@172.25.250.254:9990 /] reload --host=serverb
```

20. You are now ready to deploy the **cluster** application WAR file in the managed domain. The **cluster.war** file is available in the **/home/student/JB348/apps** folder on the **workstation**.

    20.1. Application deployments in a managed domain are always done at the server groups level. Each application belongs to one or more server groups. Deploy the **cluster.war** file to **Group1**.

```
[domain@172.25.250.254:9990 /] deploy \
/tmp/cluster.war --server-groups=Group1
```

Verify that the **cluster.war** file is deployed on **servera.1** and **serverb.1** because they are part of the **Group1** server group. Monitor the console window of **servera** and **serverb** for any error messages or warnings.

20.2. Verify that you can access the **cluster** application at **http://172.25.250.10:8080/cluster** and **http://172.25.250.11:8080/cluster**.

20.3. Verify that you can **NOT** access the **cluster** application at **http://172.25.250.10:8180/cluster** or **http://172.25.250.11:8180/cluster** because these servers are part of **Group2** and you have not deployed the application on **Group2**.

20.4. Verify that you can access the **cluster** application at **http://172.25.250.254:9080/cluster**. This access are being handled by the load balancer.

21. Clean Up and Grading

21.1. Press **Ctrl**+**C** in the terminal windows where you started the cluster instances of EAP to stop the cluster.

21.2. Run the following command from the **workstation** to grade the exercise:

```
[student@workstation ~]$ lab configure-cluster grade
```

This concludes the lab.

Summary

In this chapter, you learned:

- A cluster should provide the following capabilities:
  - High Availability (HA)
  - Scalability
  - Failover
  - Fault Tolerance
  - Load Balancing
- The Infinispan subsystem provides caching support for JBoss EAP, facilitating the high availability features of clustered servers.
- There are four different types of caches on Infinispan:
  - Local
  - Invalidation
  - Replication
  - Distribution
- The JGroups Subsystem provides all the communication mechanisms for how the servers in a cluster communicate with each other.
- JGroups consists of three parts:
  - Channel
  - Building blocks
  - Protocol stack
- EAP 7 is preconfigured with two JGroups stacks:
  - UDP
  - TCP
- The singleton subsystem can be configured with different election policies that define the master. Two kinds of election policies are provided by EAP 7:
  - Simple
  - Random