Reference Architectures 2017 Configuring a Red Hat JBoss EAP 7 Cluster

Calvin Zhu Babak Mozaffari

Reference Architectures 2017 Configuring a Red Hat JBoss EAP 7 Cluster

Calvin Zhu

Babak Mozaffari
refarch-feedback@redhat.com

Legal Notice

Abstract

This reference architecture provides offline CLI scripts to fully automate the configuration of two EAP 7 Clusters, each set up as a separate domain - one active and another passive - to eliminate any downtime due to maintenance and upgrades. Each cluster consists of three EAP instances running a modified version of the provided full-ha profile. Through in-memory data replication, each cluster makes the HTTP and EJB sessions available on all nodes. A second-level cache is set up for the JPA layer with the default option to invalidate cache entries as they are changed. The JMS messaging subsystem, based on ActiveMQ Artemis, is configured to use a shared file store, with one live and one backup messaging server configured on each EAP node for redundancy. An instance of the Red Hat JBoss Core Services Apache HTTP Server sits in front of the each cluster, balancing web load with sticky behavior, while also ensuring transparent failover in case a node becomes unavailable.

Table of Contents

COMMENTS AND FEEDBACK

In the spirit of open source, we invite anyone to provide feedback and comments on any reference architecture. Although we review our papers internally, sometimes issues or typographical errors are encountered. Feedback allows us to not only improve the quality of the papers we produce, but allows the reader to provide their thoughts on potential improvements and topic expansion to the papers. Feedback on the papers can be provided by emailing refarch-feedback@redhat.com. Please refer to the title within the email.

CHAPTER 1. EXECUTIVE SUMMARY

Ensuring the availability of production services is critical in today's IT environment. A service that is unavailable for even an hour can be extremely costly to a business. In addition to the need for redundancy, large servers have become less and less cost-efficient in recent years. Through its clustering feature, **Red Hat® JBoss® Enterprise Application Platform (EAP) 7** allows horizontal scaling by distributing the load between multiple physical and virtual machines and eliminating a single point of failure. Efficient and reliable group communication built on top of TCP or UDP enables replication of data held in volatile memory, thereby ensuring high availability, while minimizing any sacrifice in efficiency. The domain configuration capability of JBoss EAP 7 mitigates governance challenges and promotes consistency among cluster nodes.

This reference architecture stands up two JBoss EAP 7 clusters, each set up as a separate domain, one active and another passive, to eliminate any downtime due to maintenance and upgrades. Each cluster consists of three JBoss EAP instances running the provided full-ha profile, while customizing it based on the application requirements. Through in-memory data replication, each cluster makes the HTTP and Enterprise Java™ Beans (EJB) sessions available on all nodes. A second-level cache is set up for the Java Persistence API (application programming interface)(JPA) layer with the default option to invalidate cache entries as they are changed. The ActiveMQ Artemis subsystem is configured to use shared store high availability (HA) policy. An instance of Red Hat JBoss Core Services Apache HTTP Server 2.4 sits in front of the JBoss EAP domain, balancing web load with sticky behavior while also ensuring transparent failover in case a node becomes unavailable.

The goal of this reference architecture is to provide a thorough description of the steps required for such a setup, while citing the rationale and explaining the alternatives at each decision juncture, when applicable. Within time and scope constraints, potential challenges are discussed, along with common solutions to each problem. JBoss EAP command line interface (CLI) scripts, configuration files and other attachments are provided to facilitate the reproduction of the environment and to help the reader validate their understanding, along with the solution.

CHAPTER 2. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM 7

2.1. OVERVIEW

Red Hat JBoss Enterprise Application Platform (EAP) 7 is a fast, secure and powerful middleware platform built upon open standards and compliant with the Java Enterprise Edition (Java EE) 7 specification. It provides high-availability clustering, powerful messaging, distributed caching and other technologies to create a stable and scalable platform.

The modular structure allows for services to be enabled only when required, significantly increasing start-up speed. The Management Console and Management CLI remove the need to edit XML configuration files by hand, adding the ability to script and automate tasks. In addition, it includes APIs and development frameworks that can be used to develop secure, powerful, and scalable Java EE applications quickly.

2.2. CLUSTERING

Clustering refers to using multiple resources, such as servers, as though they were a single entity.

In its simplest form, horizontal scaling can be accomplished by using load balancing to distribute the load between two or more servers. Such an approach quickly becomes problematic when the server holds non-persistent and in-memory state. Such a state is typically associated with a client session. Sticky load balancing attempts to address these concerns by ensuring that client requests tied to the same session are always sent to the same server instance. For web requests, sticky load balancing can be accomplished through either the use of a hardware load balancer, or a web server with a software load balancer component. This architecture covers the use of a web server with a specific load balancing software but the principles remain largely the same for other load balancing solutions, including those that are hardware based.

While this sticky behavior protects callers from loss of data due to load balancing, it does not address the potential failure of a node holding session data. Session replication provides a copy of the session data, in one or several other nodes of the cluster, so they can fully compensate for the failure of the original node.

Session replication, or any similar approach to provide redundancy, presents a tradeoff between performance and reliability. Replicating data through network communication or persisting it on the file system is a performance cost and keeping in-memory copies on other nodes is a memory cost. The alternative, however, is a single point of failure where the session is concerned.

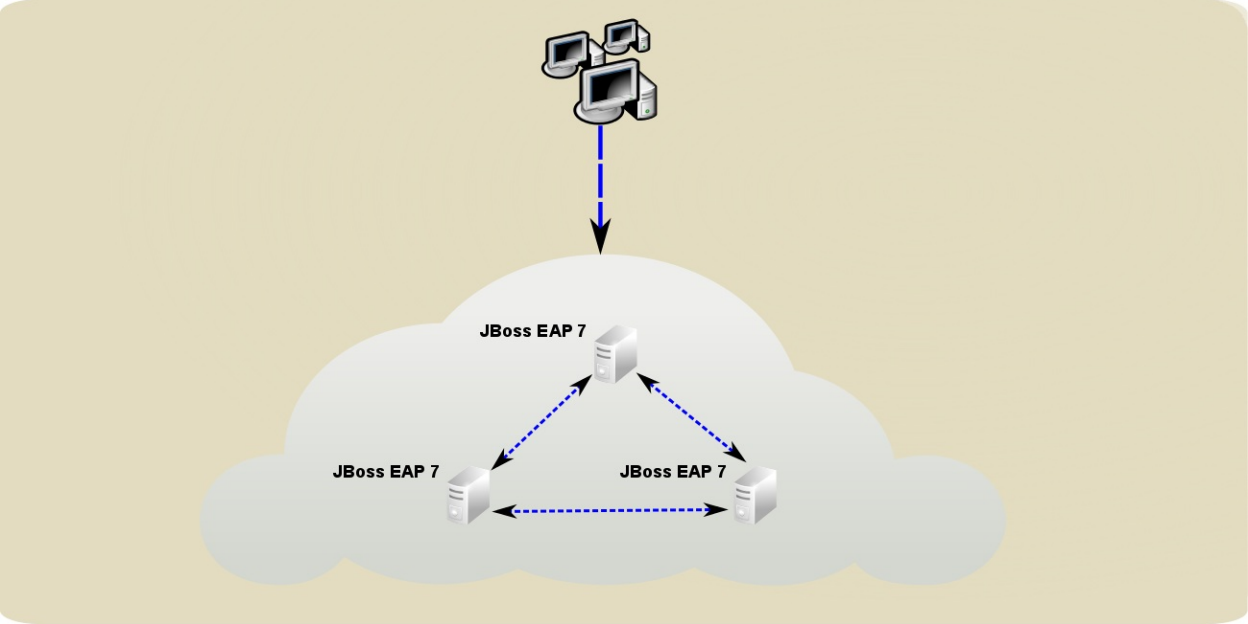
JBoss EAP 7 supports clustering at several different levels and provides both load balancing and failover benefits. Some of the subsystems that can be made highly available include:

- Instances of the Application Server
- The Web Subsystem / Servlet Container
- EJB, including stateful, stateless, and entity beans
- Java Naming and Directory Interface (JNDI) services
- Single Sign On (SSO) Mechanisms
- Distributed cache
- HTTP sessions

- Java Message Service (JMS) and message-driven beans (MDBs)

Ideally, a cluster of JBoss EAP 7 servers is viewed as a single EAP 7 instance and the redundancy and replication is transparent to the caller.

Figure 2.1. Red Hat JBoss EAP 7 cluster



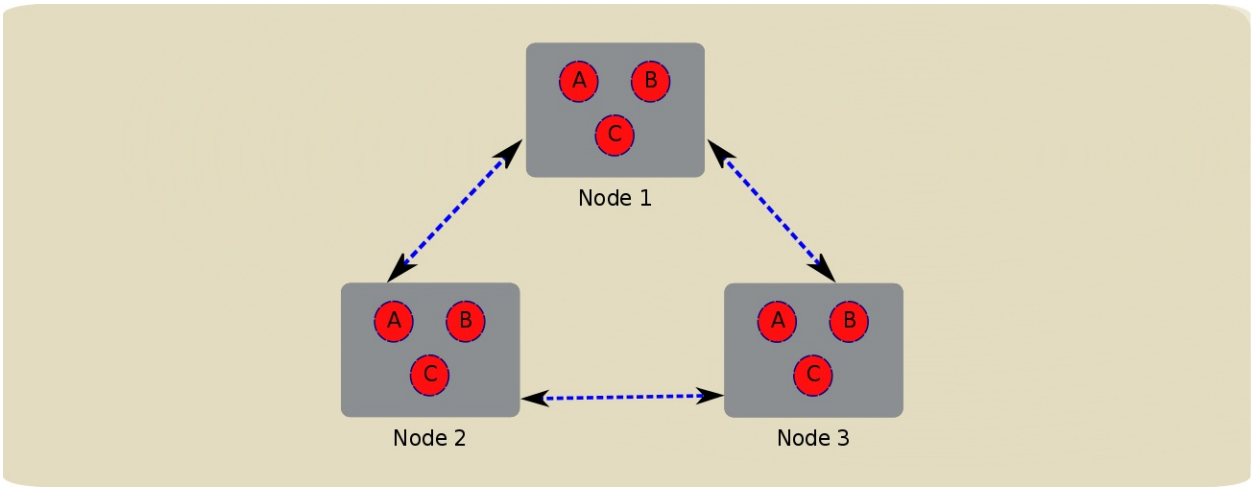
2.3. HTTP SESSIONS

An HTTP session is implicitly or explicitly created for an HTTP client and it is maintained until such time that it is either explicitly invalidated, or naturally times out. The existence of an HTTP session makes a web application stateful and leads to the requirement of an intelligent clustering solution.

Once a JBoss EAP 7 cluster is configured and started, a web application simply needs to declare itself as distributable to take advantage of the EAP 7 session replication capabilities. JBoss EAP 7 uses Infinispan to provide session replication. One available mode for this purpose is replication mode. A replicated cache replicates all session data across all nodes of the cluster asynchronously. This cache can be created in the web cache container and configured in various ways. The replication mode is a simple and traditional clustering solution to avoid a single point of failure and allow requests to seamlessly transition to another node of the cluster. It works by simply creating and maintaining a copy of each session in all other nodes of the cluster.

Assuming a cluster of three nodes, with an average of one session per node, sessions A, B and C are created on all nodes; the following diagram depicts the effect of session replication in such a scenario:

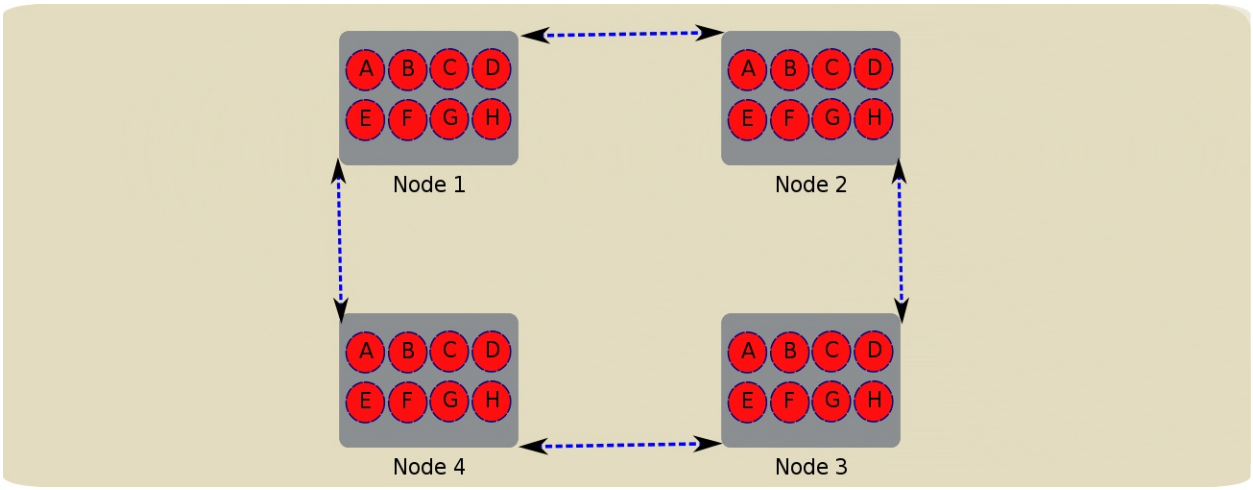
Figure 2.2. HTTP Session Clustering, Replication



The replication approach works best in small clusters. As the number of users increases and the size of the HTTP session grows, keeping a copy of every active session on every single node means that horizontal scaling can no longer counter the increasing memory requirement.

For a larger cluster of four nodes with two sessions created on each node, there are eight sessions stored on every node:

Figure 2.3. HTTP Session Replication in Larger Clusters



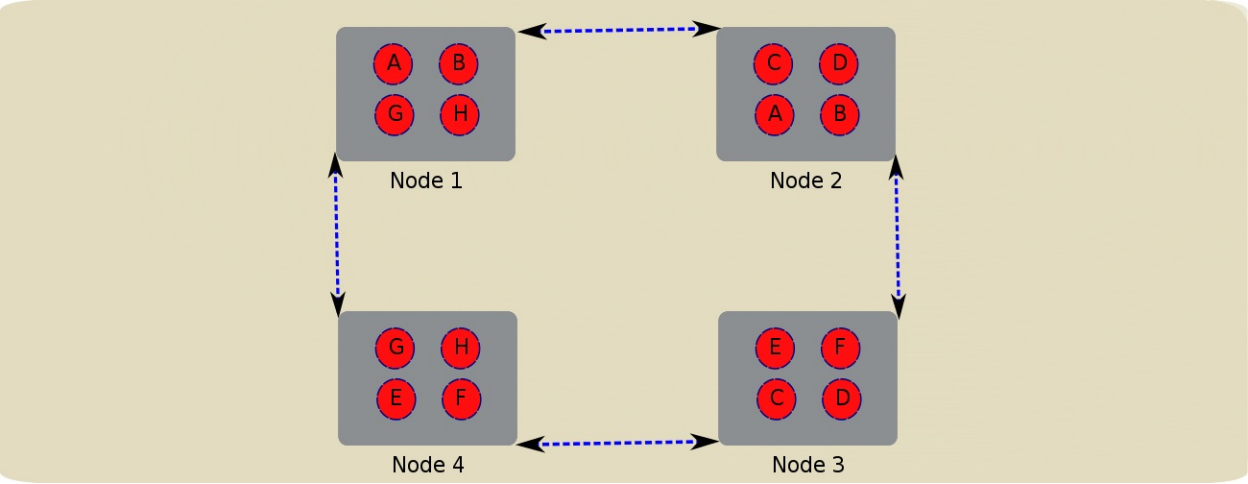
In such scenarios, the preferred strategy is the distribution mode. Under the distribution mode, Infinispan allows the cluster to scale linearly as more servers are added. Each piece of data is copied to a number of other nodes, but this is different from the buddy replication system used in older versions of JBoss EAP, as the nodes in question are not statically designated and proven grid algorithms are used to scale the cluster more effectively.

The web cache container is configured with a default distributed cache called **dist**, which is initially set up with 2 owners. The number of owners determines the total number of nodes that will contain each data item, so an owners count of 2, results in one backup copy of the HTTP session.

Sticky load balancing behavior results in all HTTP requests being directed to the original owner of the session; if that server fails, a new session owner is designated and the load balancer is automatically associated with the new host, so that a remote call results in a transparent redirection to a node that now contains the session data and new backup data is created on the remaining servers.

Reproducing the previous example of a larger cluster of four nodes, with two sessions created on each node, there would only be four sessions stored on every node with distribution:

Figure 2.4. HTTP Session Distribution in Larger Clusters



Note

Refer to Red Hat JBoss EAP 7 documentation for details on configuring the web clustering mode, including the number of copies in the distribution mode.

Administrators can also specify a limit for the number of currently active HTTP sessions and result in the passivation of some sessions, to make room for new ones, when this limit is reached. Passivation and subsequent activation follows the Least Recently Used (LRU) algorithm. The maximum number of active sessions or the idle time before passivation occurs can be configured through CLI, as described in the JBoss EAP documentation.

2.4. STATELESS SESSION BEANS

By definition, a stateless session bean avoids holding data on behalf of its client. This makes it easier to cluster stateless sessions beans, and removes any concern about data loss resulting from failure. Contrary to stateful HTTP conversations and stateful session beans, the lack of state in a stateless bean means that sequential calls made by the same client through the same stub can be load balanced across available cluster nodes. This is in fact the default behavior of stateless session beans in JBoss EAP 7, when the client stub is aware of the cluster topology. Such awareness can either be achieved by designating the bean as clustered or through EJB client configuration that lists all server nodes.

The JNDI lookup of a stateless session bean returns an intelligent stub, which has knowledge of the cluster topology and can successfully load balance or fail over a request across the available cluster nodes. This cluster information is updated with each subsequent call so the stub has the latest available information about the active nodes of the cluster.

2.5. STATEFUL SESSION BEANS

With a stateful session bean, the container dedicates an instance of the bean to each client stub. The sequence of calls made through the stub are treated as a conversation and the state of the bean is maintained on the server side, so that each call potentially affects the result of a subsequent call. Once again, the stub of the stateful session bean, returned to a caller through a JNDI lookup, is an intelligent stub with knowledge of the cluster. However, this time the stub treats the conversation as a sticky session and routes all requests to the same cluster node, unless and until that node fails. Much like HTTP session replication, JBoss EAP 7 uses an Infinispan cache to hold the state of the

stateful session bean and enable failover in case the active node crashes. Once again, only a distributed cache is preconfigured under the ejb cache container and set up as the default option in JBoss EAP 7. The ejb cache container can be independently configured to use a new cache that is set up by an administrator.

2.6. TRANSACTION SUBSYSTEM

A transaction consists of two or more actions which must either all succeed or all fail. A successful outcome is a commit, and a failed outcome is a roll-back. In a roll-back, each member's state is reverted to its state before the transaction attempted to commit. The typical standard for a well-designed transaction is that it is Atomic, Consistent, Isolated, and Durable (ACID). Red Hat JBoss EAP 7 defaults to using the **Java Transaction API (JTA)** to handle transactions. **Java Transaction Service (JTS)** is a mapping of the **Object Transaction Service (OTS)** to Java.

Note

When a node fails within a transaction, recovery of that transaction will be attempted only when that node is restarted again

While common storage may be used among EAP cluster nodes, the transaction log store of each server cannot be distributed and each node maintains its own object store. The node-identifier attribute of the transaction subsystem identifies avoids conflict between nodes by clearly associating transactions with their owner nodes and avoiding potential conflict. This attribute is then used as the basis of the unique transaction identifier. Refer to the Red Hat documentation on configuring the transaction manager for further details.

2.7. JAVA PERSISTENCE API (JPA)

The **Java Persistence API (JPA)** is the standard for using persistence in Java projects. Java EE 7 applications use the Java Persistence 2.1 specification. Hibernate EntityManager implements the programming interfaces and life-cycle rules defined by the specification. It provides JBoss EAP 7 with a complete Java persistence solution. JBoss EAP 7 is 100% compliant with the Java Persistence 2.1 specification. Hibernate also provides additional features to the specification.

When using JPA in a cluster, the state is typically stored in a central database or a cluster of databases, separate and independent of the JBoss EAP cluster. As such, requests may go through JPA beans on any node of the cluster and failure of an JBoss EAP instance does not affect the data, where JPA is concerned.

First-level caching in JPA relates to the persistence context, and in JBoss EAP 7, the hibernate session. This cache is local, short-lived and bound to the transaction. As long as the transaction is handled by the server, there is no concern in terms of horizontal scaling and clustering of JPA. However, in instances where the calling client controls the transaction and invokes EJBs resulting in multiple calls to the same JPA entities, there is a risk that different nodes of an EAP cluster would attempt to update and then read the same data, resulting in stale reads from the 1L cache. This can be avoided by ensuring that all such calls are directed to the same node. For further information on this issue, refer to the following Red Hat knowledgebase article.

JPA second-level caching refers to the more traditional database cache. It is a local data store of JPA entities that improves performance by reducing the number of required roundtrips to the database. With the use of second-level caching, it is understood that the data in the database may only be modified through JPA and only within the same cluster. Any change in data through other

means may leave the cache stale, and the system subject to data inconsistency. However, even within the cluster, the second-level cache suddenly introduces a stateful aspect to JPA that must be addressed.

Red Hat JBoss EAP 7 uses Infinispan for second-level caching. This cache is set up by default and uses an **invalidation-cache** called **entity** within the **hibernate** cache container of the server profile.

To configure an application to take advantage of this cache, the value of **hibernate.cache.use_second_level_cache** needs to be set to true in the persistence XML file. In JBoss EAP, the Infinispan cache manager is associated with JPA by default and does not need to be configured.

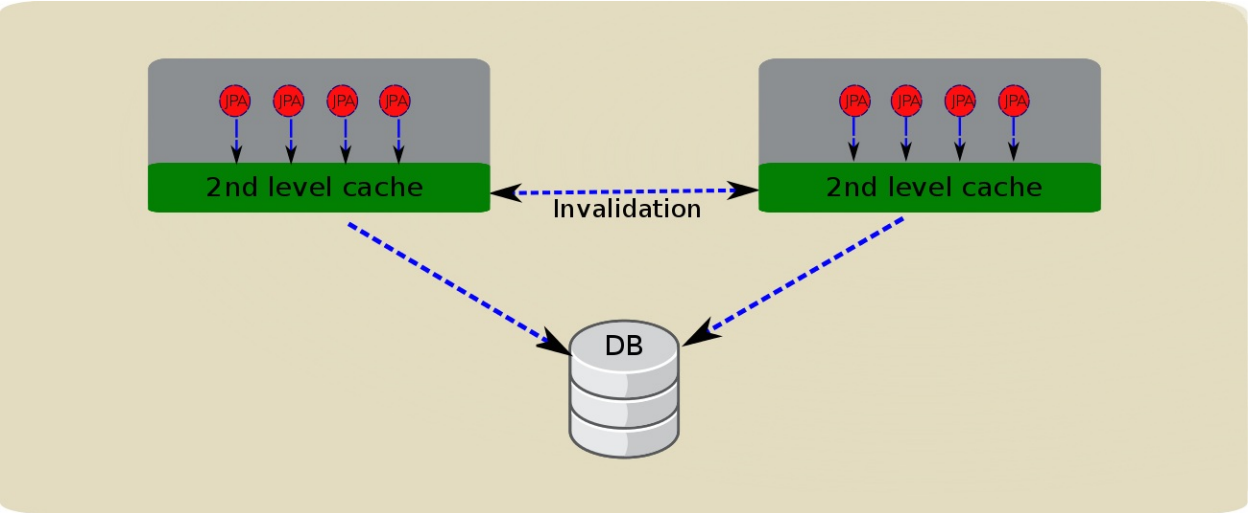
Once a persistence unit is configured to use second-level caching, it is the shared-cache-mode property in the application's persistence configuration file that determines which entities get cached. Possible values for this property include:

- **ENABLE_SELECTIVE** to only cache entities explicitly marked as cacheable
- **DISABLE_SELECTIVE** to cache all entities, unless explicitly excluded from caching
- **ALL** to cache all entities, regardless of their individual annotation
- **NONE** to not cache any entity, regardless of its annotation

Individual Entity classes may be marked with **@Cacheable(true)** or **@Cacheable(false)** to explicitly request caching or exclusion from caching.

The entity cache is an invalidation cache, which means that entities are cached on each node only when they are loaded on that node. Once an entity is changed on one node, an invalidation message is broadcast to the cluster to invalidate and remove all cached instances of this entity on any node of the cluster. That results in the stale version of the entity being avoided on other nodes and an updated copy being loaded and cached on any node that requires it, at the time when it's needed.

Figure 2.5. JPA Second-Level Cache with Invalidation



2.8. ACTIVEMQ ARTEMIS

The messaging broker in JBoss EAP 6 was called HornetQ, a JBoss community project. The HornetQ codebase was donated to the Apache ActiveMQ project, and the HornetQ community joined that project to enhance the donated codebase and create a next-generation messaging

broker. The result is Apache ActiveMQ Artemis, the messaging broker for JBoss EAP 7, providing messaging consolidation and backwards compatibility with JBoss EAP 6.

JBoss EAP 7 uses Apache ActiveMQ Artemis as its JMS broker and is configured using the messaging-activemq subsystem. This fully replaces the HornetQ broker but retains protocol compatibility with JBoss EAP 6.

Default configuration for the messaging-activemq subsystem is included when starting the JBoss EAP server with the full or full-ha configuration. The full-ha option includes advanced configuration for features like **clustering** and **high availability**.

ActiveMQ's HA feature allows JMS servers to be linked together as live - backup groups where each live server has a backup. Live servers receive messages from clients, while a backup server is not operational until failover occurs. A backup server can be owned by only one live server. In cases where there is more than one backup server in a backup group, only one will announce itself as being ready to take over the live server. This server will remain in passive mode, waiting to take over the live server's work.

When a live server crashes or is brought down gracefully, the backup server currently in passive mode will become the new live server. If the new live server is configured to allow automatic failback, it will detect the old live server coming back up and automatically stop, allowing the old live server to start receiving messages again.

ActiveMQ Artemis supports two different HA strategies for backing up a server: data replication and shared store.

<server>
…
<replication-master … />
OR
<replication-slave … />
OR
<shared-store-master … />
OR
<shared-store-slave … />
…
<server>

Data Replication

When using replication, the live and the backup servers do not share the same data directories, all data synchronization is done over the network. Therefore all (persistent) data received by the live server will be duplicated to the backup.

If the live server is cleanly shutdown, the backup server will activate and clients will failover to backup. This behavior is pre-determined and is therefore not configurable when using data replication.

Upon startup the backup server will first need to synchronize all existing data from the live server before replacing it. Unlike shared storage, a replicating backup will not be fully operational immediately after startup. The time it will take for the synchronization to happen depends on the amount of data to be synchronized and the network speed.

Shared Store

This style of high availability differs from data replication in that it requires a shared file system which is accessible by both the live and backup nodes. This means that the servers use the same location for their paging, message journal, bindings journal, and large messages in their configuration.

Typically this shared file system will be some kind of high performance storage area network (SAN). Red Hat does not recommend using network-attached storage (NAS) for your storage solution.

The advantage of shared-store high availability is that no replication occurs between the live and backup nodes, this means it does not suffer any performance penalties due to the overhead of replication during normal operation.

The disadvantage of shared store replication is that when the backup server activates it needs to load the journal from the shared store which can take some time depending on the amount of data in the store. Also, it requires a shared storage solution supported by JBoss EAP.

2.9. HTTP CONNECTORS

JBoss EAP 7 can take advantage of the load-balancing and high-availability mechanisms built into external web servers, such as **Apache Web Server, Microsoft IIS, and Oracle iPlanet, as well as through Undertow**. JBoss EAP 7 communicates with an external web server using an HTTP Connector. These HTTP connectors are configured within the web subsystem of JBoss EAP 7. Web Servers include software modules which control the way HTTP requests are routed to JBoss EAP 7 worker nodes. Each of these modules works differently and has its own configuration method. Modules may be configured to balance work loads across multiple JBoss EAP 7 server nodes, move work loads to alternate servers in case of a failure event, or do both.

JBoss EAP 7 supports several different HTTP connectors. The one you choose depends on both the Web Server you connect to and the functionality you require. The table below lists the differences between various available HTTP connectors, all compatible with JBoss EAP 7. For the most up-to-date information about supported HTTP connectors, refer to the official Red Hat documentation.

Table 2.1. HTTP Connectors

Connector	Web Server	Supported Operating Systems	Supported Protocols
mod_cluster	Apache HTTP Server in Red Hat JBoss Web Server, Red Hat JBoss Core Services	Red Hat Enterprise Linux®, Microsoft Windows Server, Oracle Solaris	HTTP, HTTPS, AJP
mod_jk	Apache HTTP Server in JBoss Web Server, JBoss Core Services	Red Hat Enterprise Linux®, Microsoft Windows Server, Oracle Solaris	AJP
mod_proxy	Apache HTTP Server in JBoss Web Server, JBoss Core Services	Red Hat Enterprise Linux®, Microsoft Windows Server, Oracle Solaris	HTTP, HTTPS, AJP

Connector	Web Server	Supported Operating Systems	Supported Protocols
ISAPI connector	Microsoft IIS	Microsoft Windows Server	AJP
NSAPI connector	Oracle iPlanet Web Server	Oracle Solaris	AJP

2.9.1. mod_cluster

mod_cluster is an HTTP load balancer that provides a higher level of intelligence and control over web applications, beyond what is available with other HTTP load balancers. Using a special communication layer between the JBoss application server and the web server, mod_cluster can not only register when a web context is enabled, but also when it is disabled and removed from load balancing. This allows mod_cluster to handle full web application life cycles. With mod_cluster, the load of a given node is determined by the node itself. This allows load balancing using a wider range of metrics including CPU load, heap usage and other factors. It also makes it possible to use a custom load metric to determine the desired load balancing effect. mod_cluster has two modules: one for the web server, which handles routing and load balancing, and one for the JBoss application server to manage the web application contexts. Both modules must be installed and configured for the cluster to function. The mod_cluster module on JBoss EAP is available by default but may be further configured to auto-discover the proxy through multicast (**advertise**) or contact it directly through IP address and port.

CHAPTER 3. REFERENCE ARCHITECTURE ENVIRONMENT

3.1. OVERVIEW

This reference architecture consists of two JBoss EAP 7 clusters, each containing three nodes. The two clusters provide active/passive redundancy in an effort to approach a zero-downtime architecture. Having two distinct and separate clusters allows for upgrades and other types of maintenance, without requiring an operational maintenance window.

While it is also possible to use JBoss EAP itself as a front-end load balancer, this paper opts for the more common configuration of Apache HTTP Server. Two instances of **Red Hat JBoss Core Services Apache HTTP Server 2.4** are deployed, each to front-end one of the EAP clusters for incoming HTTP requests.

A single instance of **PostgreSQL database** is used for JPA persistence in the back-end.

At any given time, the active setup simply includes an HTTP server that redirects to a logical JBoss EAP 7 application server for all functionality, which in turn uses the **PostgreSQL database** instance for its persistence.

3.2. RED HAT JBOSS CORE SERVICES APACHE HTTP SERVER 2.4

Two instances of **Red Hat JBoss Core Services Apache HTTP Server** are installed on a separate machine to front-end the active and passive EAP clusters, providing sticky-session load balancing and failover. The HTTP servers use mod_cluster and Apache JServ Protocol (AJP) to forward HTTP requests to an appropriate JBoss EAP node. The advertise feature of mod_cluster is turned off and the proxy's host and port are configured on the mod_cluster plugin of the application servers, so that servers can directly contact and update the proxy.

Where requests over HTTP are concerned, the one instance of HTTP Server front-ending an EAP cluster can be considered a single point of failure. The HTTP Server itself can also be clustered but the focus is the enterprise application platform and clustering of the HTTP Server is beyond the scope of this reference architecture.

3.3. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM

Two JBoss EAP 7 clusters are deployed and configured with nearly identical settings to provide active/passive redundancy. The clusters are front-ended by different HTTP Servers and as such, use a different URL in their mod_cluster **proxy-list** configuration.

In an effort to strike a balance in proposing an economical solution, while still providing high availability, each of the three nodes of the passive domain is co-located with a node of the active domain on the same machine. In a basic non-virtualized environment, this allows whichever cluster is active at any given time to take full advantage of the available hardware resources. The trade off is that certain maintenance tasks may affect both clusters and require downtime.

Alternatively, having two virtual machines on each physical server, one to host the active domain node and another for the passive domain node, allows certain OS-level maintenance work to be performed independently for each cluster. With such a setup, the efficiency loss from a resource usage perspective is insignificant. In this virtualized environment, certain maintenance tasks may require the shutdown of the host OS, which would still impact both clusters. To tilt yet further towards reliability at the expense of cost-efficiency, a distinct physical machine may be used for each node of each cluster, resulting in the use of 6 physical servers for the JBoss EAP 7 instances in this architecture.

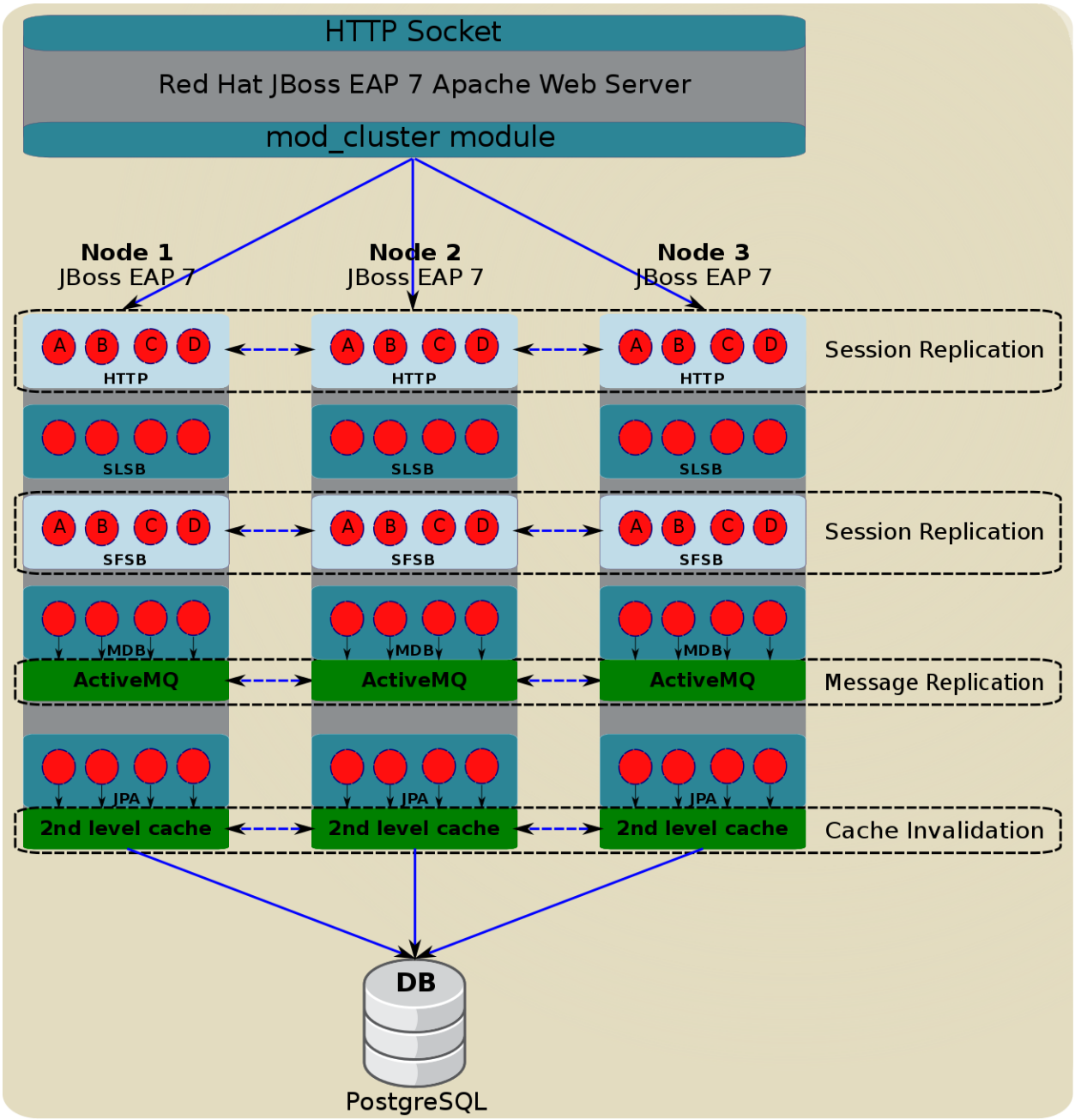
3.4. POSTGRESQL DATABASE

To configure and demonstrate the functionality of JPA second-level caching in an EAP 7 cluster, an external database instance must be installed and configured. JPA is largely portable across various RDBMS vendors and switching to a different database server can easily be accomplished with minor configuration changes. This reference architecture uses **PostgreSQL Database Server**. JBoss EAP 7 uses hibernate for its JPA implementation, which has the ability to create the necessary schema, so the only required configuration on the database is the creation of a database as well as a user with privileges to create tables and modify data in that database. A single PostgreSQL database instance can serve both the active and passive EAP 7 clusters.

For client requests that result in JPA calls, the one instance of PostgreSQL Database can be considered a single point of failure. The Database itself can also be clustered, but the focus of this effort is JBoss EAP 7 and clustering of the Database is beyond the scope of this reference architecture.

The following diagram attempts to provide a simplified architectural diagram of this reference environment:

Figure 3.1. Reference Architecture - Single Cluster



Note

Message replication, as portrayed in this diagram, is particularly simplified. This reference architecture using a shared store to create an HA cluster for messaging. Each node has a live and a backup server, where each backup server shares a store with the live server of another node. The final result is an HA cluster with full failover and load balancing capability, as per the above diagram.

CHAPTER 4. CREATING THE ENVIRONMENT

4.1. PREREQUISITES

Prerequisites for creating this reference architecture include a supported Operating System and JDK. Refer to Red Hat documentation for JBoss EAP 7 supported environments.

With minor changes, almost any relational database management system (RDBMS) may be used in lieu of PostgreSQL Database, but if Postgres is used, the details of the download and installation are also considered a prerequisite for this reference architecture. On a Red Hat Enterprise Linux system, installing PostgreSQL can be as simple as running:

```
# yum install postgresql-server.i686
```

4.2. DOWNLOADS

Download the attachments to this document. These scripts and files will be used in configuring the reference architecture environment:

https://access.redhat.com/node/2359241/40/0

If you do not have access to the Red Hat customer portal, See the Comments and Feedback section to contact us for alternative methods of access to these files.

Download the following Red Hat JBoss Enterprise Application Platform and Red Hat JBoss Core Services Apache HTTP Server versions from Red Hat's Customer Support Portal:

- Red Hat JBoss Core Services Apache HTTP Server 2.4.6 for RHEL 7 x86_64
- Red Hat JBoss Enterprise Application Platform 7.0.0

Warning

It is strongly recommended to set up clusters on dedicated virtual local area networks (VLAN). If a VLAN is not available, use a version of JBoss EAP 7 that includes the fix for CVE-2016-2141, or download and apply the patch for Red Hat JBoss Enterprise Application Platform 7.0, while also encrypting the communication.

4.3. INSTALLATION

4.3.1. Red Hat JBoss Core Services Apache HTTP Server 2.4

Red Hat JBoss Core Services Apache HTTP Server is available for download and installation in both RPM and ZIP form. This reference architecture uses modified scripts to run two instances of Apache HTTP Server. Follow the ZIP installation process to allow more customizability and support for other Operating Systems.

4.3.2. Red Hat JBoss Enterprise Application Platform 7

Red Hat JBoss EAP 7 does not require any installation steps. The archive file simply needs to be extracted after the download. This reference architecture installs two sets of binaries for EAP 7 on each machine, one for the active domain and another for the passive domain:

```
# unzip jboss-eap-7.0.0.zip -d /opt/ # mv /opt/jboss-eap-7.0
/opt/jboss-eap-7.0_active # unzip jboss-eap-7.0.0.zip -d /opt/ # mv
/opt/jboss-eap-7.0 /opt/jboss-eap-7.0_passive
```

4.4. CONFIGURATION

4.4.1. Overview

Various other types of configuration may be required for UDP and TCP communication. For example, Linux operating systems typically have a low maximum socket buffer size configured, which is lower than the default cluster JGroups buffer size. It may be important to correct any such warnings observed in the EAP logs. For example, in this case for a Linux operating system, the maximum socket buffer size may be configured as follows. Further details are available in the official documentation.

```
# sysctl -w net.core.rmem_max=26214400 # sysctl -w
net.core.wmem_max=1048576
```

4.4.2. Security-Enhanced Linux

This reference environment has been set up and tested with **Security-Enhanced Linux** (SELinux) enabled in ENFORCING mode. Once again, please refer to the Red Hat documentation on SELinux for further details on using and configuring this feature. For any other operating system, consult the respective documentation for security and firewall solutions to ensure that maximum security is maintained while the ports required by your application are opened.

When enabled in ENFORCING mode, by default, SELinux prevents Apache web server from establishing network connections. On the machine hosting Apache web server, configure SELinux it to allow httpd network connections:

```
# /usr/sbin/setsebool httpd_can_network_connect 1
```

4.4.3. Ports and Firewall

In the reference environment, several ports are used for intra-node communication. This includes ports 6661 and 6662 on the web servers' mod-cluster module, being accessed by all three cluster nodes, as well as the 5432 Postgres port. Web clients are routed to the web server through ports 81 and 82. The EAP cluster nodes also require many different ports for access, including 8080 or 8180 for HTTP access and EJB calls, 8009 and 8109 for AJP and so on.

This reference architecture uses **firewalld**, the default Red Hat firewall, to block all network packets by default and only allow configured ports and addresses to communicate. Refer to the Red Hat documentation on firewalld for further details.

Check the status of firewalld on each machine and make sure it is running:

```
# systemctl status firewalld
```

This reference environment starts with the default and most restrictive firewall setting (zone=public) and only opens the required ports.For example open the 81 port for the active HTTP Apache server for client to connect.

```
sudo firewall-cmd --permanent --zone=public --add-rich-rule='rule
family=ipv4 source address="10.10.0.1/8" port port=81 protocol=tcp
accept'
```

Using the permanent flag persists the firewall configuration but also requires a reload to have the changes take effect immediately:

```
# firewall-cmd --reload
```

Please see the appendix on **firewalld** configuration for the firewall rules used for the active domain in this reference environment.

Note, please make sure the network is properly configured for multicast and the multicast address would be verified during configuration stage of the enivornment, otherwise the HA function might not work properly.

4.4.4. Red Hat JBoss Core Services Apache HTTP Server 2.4

A freshly installed version of the Apache HTTP Server includes sample configuration under *httpd/conf/httpd.conf*. This reference architecture envisions running two instances of the web server using the same installed binary. The configuration file has therefore been separated into the provided *cluster1.conf* and *cluster2.conf* files, with an index variable that is set to 1 and 2 respectively to allow a clean separation of the two Apache HTTP Server instance.

The provided *cluster1.conf* and *cluster2.conf* files assume that Apache HTTP Server has been unzipped to */opt/jbcs-httpd*. To avoid confusion, it may be best to delete the sample *httpd.conf* file and place these two files in the *httpd/conf* directory instead. Edit *cluster1.conf* and *cluster2.conf* and modify the root directory of the web server installation as appropriate for the directory structure. The ServerRoot file will have a default value such as:

ServerRoot "/opt/jbcs-httpd/httpd"

Review the cluster configuration files for other relevant file system references. This reference architecture does not use the web server to host any content, but the referenced file locations can be adjusted as appropriate if hosting other content is required. Notice that the two cluster files use a suffix of 1 and 2 to separate the process ID file, listen port, *MemManagerFile*, *mod_cluster* port, error log and custom log of the two clusters.

To allow starting and stopping Apache HTTP Server separately for each cluster, modify the *OPTIONS* variable in the *apachectl* script to point to the correct configuration file and output to the appropriate error log. The modified *apachectl* script is provided in the attachments:

```
OPTIONS="-f /opt/jbcs-httpd/httpd/conf/cluster${CLUSTER_NUM}.conf
      -E /opt/jbcs-httpd/httpd/logs/httpd${CLUSTER_NUM}.log"
```

It is enough to simply set the environment variable to 1 or 2 before running the script, to control the first or second cluster's load balancer. The provided *cluster1.sh* and *cluster2.sh* scripts set this variable:

```
#!/bin/sh export CLUSTER_NUM=1 ./httpd/sbin/apachectl $@
```

These scripts can be used in the same way as *apachectl*, for example, to start the first Apache HTTP Server instance:

```
# ./cluster1.sh start
```

4.4.5. PostgreSQL Database setup

After a basic installation of the PostgreSQL Database Server, the first step is to initialize it. On Linux systems where proper installation has been completed through **yum** or **RPM** files, a service will be configured, and can be executed to initialize the database server:

```
# service postgresql initdb
```

Regardless of the operating system, the control of the PostgreSQL database server is performed through the **pg_ctl** file. The **-D** flag can be used to specify the data directory to be used by the database server:

```
# pg_ctl -D /usr/local/pgsql/data initdb
```

By default, the database server may only be listening on *localhost*. Specify a listen address in the database server configuration file or set it to “*”, so that it listens on all available network interfaces. The configuration file is located under the data directory and in version 9.3 of the product, it is available at */var/lib/pgsql/data/postgresql.conf*.

Uncomment and modify the following line:

```
listen_addresses = '*'
```

Note

PostgreSQL Database Server typically ships with XA transaction support turned off by default. To enable XA transactions, the *max_prepared_transactions* property must be uncommented and set to a value equal or greater than *max_connections*:

```
max_prepared_transactions = 0 # zero disables the feature
                              (change requires restart)
# Note: Increasing max_prepared_transactions costs ~600 bytes of shared memory
# per transaction slot, plus lock space (see max_locks_per_transaction).
# It is not advisable to set max_prepared_transactions nonzero unless you
# actively intend to use prepared transactions.
```

Another important consideration is the ability of users to access the database from remote servers, and authenticate against it. Java applications typically use password authentication to connect to a database. For a PostgreSQL Database Server to accept authentication from a remote user, the corresponding configuration file needs to be updated to reflect the required access. The configuration file in question is located under the data directory and in version 9.3 of the product, it is available at */var/lib/pgsql/data/pg_hba.conf*.

One or more lines need to be added, to permit password authentication from the desired IP addresses. Standard network masks and slash notation may be used. For this reference architecture, a moderately more permissive configuration is used:

host all all 10.19.137.0/24 password

PostgreqSQL requires a non-root user to take ownership of the database process. This is typically achieved by a *postgres* user being created. This user should then be used to start, stop and configure databases. The database server is started using one of the available approaches, for example:

```
$ /usr/bin/pg_ctl start
```

To create a database and a user that JBoss EAP instances use to access the database, perform the following:

```
# su – postgres
$ /usr/bin/psql -U postgres postgres
CREATE USER jboss WITH PASSWORD 'password';
CREATE DATABASE eap6 WITH OWNER jboss;
\q
```

Making the *jboss* user the owner of the *eap7* database ensures the necessary privileges to create tables and modify data are assigned.

4.4.6. Red Hat JBoss Enterprise Application Platform

This reference architecture includes six distinct installations of JBoss EAP 7. There are three machines, where each includes one node of the primary cluster and one node of the backup cluster. The names *node1*, *node2* and *node3* are used to refer to both the machines and the EAP nodes on the machines. The primary cluster will be called the *active* domain while the redundant backup cluster will be termed the *passive* domain. This reference architecture selects *node1* to host the domain controller of the active domain and *node2* for the passive domain.

4.4.6.1. Adding Users

The first important step in configuring the EAP 7 clusters is to add the required users. They are Admin Users, Node Users and Application Users.

1) Admin User

An administrator user is required for each domain. Assuming the user ID of *admin* and the password of *password1!* for this *admin* user:

On *node1*:

```
# /opt/jboss-eap-7.0_active/bin/add-user.sh admin password1!
```

On *node2*:

```
# /opt/jboss-eap-7.0_passive/bin/add-user.sh admin password1!
```

This uses the non-interactive mode of the *add-user* script, to add management users with a given username and password.

2) Node Users

The next step is to add a user for each node that will connect to the cluster. For the active domain, that means creating two users called node2 and node3 (since node1 hosts the domain controller

and does not need to use a password to authenticate against itself), and for the passive domain, two users called node1 and node3 are required. This time, provide no argument to the add-user script and instead follow the interactive setup.

The first step is to specify that it is a management user. The interactive process is as follows:

What type of user do you wish to add?
a) Management User (mgmt-users.properties)
b) Application User (application-users.properties)
(a): **a**

- Simply press enter to accept the default selection of *a*

Enter the details of the new user to add.
Realm (ManagementRealm) :

- Once again simply press enter to continue

Username : **node1**

- Enter the username and press enter (node1, node2 or node3)

Password : **password1!**

- Enter *password1!* as the password, and press enter

Re-enter Password : **password1!**

- Enter *password1!* again to confirm, and press enter

About to add user 'node_X_' for realm 'ManagementRealm'
Is this correct yes/no? **yes**

- Type *yes* and press enter to continue

Is this new user going to be used for one AS process to connect to another AS process?
e.g. for a slave host controller connecting to the master or for a Remoting connection for server to server EJB calls.
yes/no?

- Type *yes* and press enter

To represent the user add the following to the server-identities definition <secret value="cGFzc3dvcmQxIQ==" />

- Copy and paste the provided secret hash into *slaves.properties* before running the CLI scripts. The host XML file of any servers that are not domain controllers need to provide this password hash along with the associated username to connect to the domain controller.

This concludes the setup of required management users to administer the domains and connect the slave machines.

3) Application User

An application user is also required on all 6 nodes to allow a remote Java class to authenticate and invoke a deployed EJB. Creating an application user requires the interactive mode, so run *add-user.sh* and provide the following six times, once on each EAP 7 installation node:

What type of user do you wish to add?
a) Management User (mgmt-users.properties)
b) Application User (application-users.properties)
(a): **b**

- Enter *b* and press enter to add an application user

Enter the details of the new user to add.
Realm (ApplicationRealm) :

- Simply press enter to continue

Username : **ejbcaller**

- Enter the username as *ejbcaller* and press enter

Password : **password1!**

- Enter *password1!* as the password, and press enter

Re-enter Password : **password1!**

- Enter *password1!* again to confirm, and press enter

What roles do you want this user to belong to? (Please enter a comma separated list, or leave blank for none)[ ]:

- Press enter to leave this blank and continue

About to add user 'ejbcaller' for realm 'ApplicationRealm'
Is this correct yes/no? **yes**

- Type *yes* and press enter to continue

…[confirmation of user being added to files]
Is this new user going to be used for one AS process to connect to another AS process?
e.g. for a slave host controller connecting to the master or for a Remoting connection for server to server EJB calls.
yes/no? **no**

- Type *no* and press enter to complete user setup

At this point, all the required users have been created; move forward to setup the server configuration files using CLI scripts, before starting the servers.

4.4.6.2. Domain configuration using CLI scripts

After the users are added, run the CLI scripts on domain controller machines to set up the configuration and deployments (*node1* hosts the domain controller of the active domain and *node2*, the passive domain). It also sets up the slave machine's configuration file (*host-slave.xml*), which you need to manually copy to other machines.

For both active and passive domains, the CLI domain setup script is similar, just with different values for environment properties.

Below are the files used for the CLI domain setup script:

- *configure.sh*: main shell script to set up the whole cluster environment
- *environment.sh*: configures the environment variables
- *setup-master.cli*: CLI script that configures the main server's *domain.xml* and *host.xml*
- *cluster.properties*: property file used by *setup-master.cli*
- *setup-slave.cli*: CLI script configures the slave servers' *host-slave.xml*
- *slaves.properties*: property file used by *setup-slave.cli*

Before running for the first time, please update these three files with proper values:

- *environment.sh*
- *cluster.properties*
- *slaves.properties*

After update, execute:

```
# configure.sh
```

Also please note that it's not mandatory to have a configuration backup folder, but it is recommended that before running for the first time, you make a copy of the *$JBOSS_HOME/domain/configuration* folder which ships with JBoss EAP 7. The reason is that the CLI domain setup script will make changes to the default configuration files. Having the configuration backup folder will help ensure that the CLI will start from the same default configuration every time, thus allowing multiple retries.

Detailed explanations of these three modified files:

1) *environment.sh*

Sample values:

JBOSS_HOME=/opt/jboss-eap-7.0
CONF_HOME=$JBOSS_HOME/domain/configuration
CONF_BACKUP_HOME=$JBOSS_HOME/domain/configuration_bck

Explanation:

- JBOSS_HOME: JBoss EAP 7 binary installation location
- CONF_HOME: Domain configuration folder, which will be the target of the CLI scripts
- CONF_BACKUP_HOME: The location from which *configure.sh* tries to restore the default configuration. Please make a backup of the configure folder before first running the script.

2) *cluster.properties*

Sample values:

```
lb.host=10.19.137.37
lb.port=6661
master.name=node1
psql.connection.url=jdbc:postgresql://10.19.137.37:5432/eap7
port.offset=0
management.http.port=9990
management.native.port=9999
multicast.address=230.0.0.1
cluster.user=admin
cluster.password=password
share.store.path=/mnt/activemq/sharefs
messaging.port=5545
messaging.backup.port=5546
```

Explanation:

- lb.host: JSON Web Signature (JWS) apache http server host address
- lb.port: JWS apache http server port
- master.name: Server name in *host.xml* for the DMC box
- psql.connection.url: JDBC URL for postgres DB
- port.offset: Server port offset value
- management.http.port: Listen port for EAP management console
- management.native.port: Listen port for EAP management tools, including CLI
- multicast.address: UDP address used across the cluster
- cluster.user: ActiveMQ cluster user name
- cluster.password: ActiveMQ cluster password
- share.store.path: the path to the ActiveMQ shared store location
- messaging.port: messaging port used in ActiveMQ remote connector/acceptor in default jms server
- messaging.backup.port: messaging port used in ActiveMQ remote connector/acceptor in backup jms server

3) *slaves.properties*

Sample values:

```
slaves=slave1,slave2
slave1_name=node2
slave1_secret="cGFzc3dvcmQxIQ=="
slave1_live_group=2
slave1_backup_group=3
slave2_name=node3
slave2_secret="cGFzc3dvcmQxIQ=="
slave2_live_group=3
slave2_backup_group=1
```

Explanation:

- slaves: the property names for the slave servers, the configure.sh does a for loop based on the names here, it is configurable for allowing more servers.
- slaveX_name: this name need to match the name added using $JBOSS_HOME/bin/add-user.sh for the slave boxes
- slaveX_secret: this is the secret hash from add node users, for AS process to connect to another AS process.
- slaveX_live_group: Live jms server group number
- slaveX_backup_group: Backup jms server group number

After running configure.sh, each slave machine's host file is generated under its own folder under the DMC box's domain/configuration folder.

$JBOSS_HOME/domain/configuration/slaveX/host-slave.xml

These new generated *host-slave.xml* needs to be manually copied to each slave box under *$JBOSS_HOME/domain/configuration/*

4.5. STARTUP

4.5.1. Starting Red Hat JBoss Core Services Apache HTTP Server

Log on to the machine where Apache HTTP Server is installed and navigate to the installed directory:

```
# cd /opt/jbcs-httpd
```

To start the web server front-ending the active domain:

```
# ./cluster1.sh start
```

To start the web server front-ending the passive domain:

```
# ./cluster2.sh start
```

To stop the active domain web server:

```
# ./cluster1.sh stop
```

To stop the passive domain web server:

```
# ./cluster2.sh stop
```

4.5.2. Starting the EAP domains

4.5.2.1. Active Domain startup

To start the active domain, assuming that *10.19.137.34* is the IP address for the *node1* machine, *10.19.137.35* for *node2* and *10.19.137.36* for *node3*:

On node1:

```
# /opt/jboss-eap-7.0_active/bin/domain.sh -b 10.19.137.34 -
bprivate=10.19.137.34 -Djboss.bind.address.management=10.19.137.34
```

Wait until the domain controller on node1 has started, then start node2 and node3

```
# /opt/jboss-eap-7.0_active/bin/domain.sh -b 10.19.137.35 -
bprivate=10.19.137.35 -Djboss.domain.master.address=10.19.137.34 --
host-config=host-slave.xml # /opt/jboss-eap-7.0_active/bin/domain.sh -b
10.19.137.36 -bprivate=10.19.137.36 -
Djboss.domain.master.address=10.19.137.34 --host-config=host-slave.xml
```

4.5.2.2. Passive Domain startup

Now start the servers in the passive domain with the provided sample configuration. Once again, assuming that *10.19.137.34* is the IP address for the *node1* machine, *10.19.137.35* for *node2* and *10.19.137.36* for *node3*:

On node2:

```
# /opt/jboss-eap-7.0_passive/bin/domain.sh -b 10.19.137.35 -
bprivate=10.19.137.35 -Djboss.bind.address.management=10.19.137.35
```

Wait until the domain controller on node2 has started. Then on node1 and node3

```
# /opt/jboss-eap-7.0_passive/bin/domain.sh -b 10.19.137.34 -
bprivate=10.19.137.34 -Djboss.domain.master.address=10.19.137.35 --
host-config=host-slave.xml # /opt/jboss-eap-7.0_passive/bin/domain.sh -
b 10.19.137.36 -bprivate=10.19.137.36 -
Djboss.domain.master.address=10.19.137.35 --host-config=host-slave.xml
```

CHAPTER 5. DESIGN AND DEVELOPMENT

5.1. RED HAT JBOSS CORE SERVICES APACHE HTTP SERVER 2.4

This reference architecture assumes that Red Hat JBoss Core Services Apache HTTP Server has been downloaded and installed at */opt/jbcs-httpd*.

Providing two separate *httpd* configurations with distinct ports makes it possible to start two separate instances of the web server. The attached *cluster1.sh* and *cluster2.sh* scripts help start and stop each instance, for example:

```
#!/bin/sh export CLUSTER_NUM=1 ./httpd/sbin/apachectl $@
```

Both *cluster1.sh* and *cluster2.sh* scripts result in a call to the standard *apachectl* script, but target the corresponding web server instance and either *cluster1.conf* or *cluster2.conf* is used as the main web server configuration file. Accordingly, the error file is created as *httpd1.log* or *httpd2.log*.

The first argument passed to either script may have be *start* or *stop* to respectively start up or shut down the web server, or any other parameter accepted by the standard *apachectl* script.

The first and perhaps most important required configuration in an httpd configuration file is to specify the path of the web server directory structure. Some of the future references, particularly those to loading modules, will be relative to this path:

ServerRoot "/opt/jbcs-httpd/httpd"

Once an instance of the web server is started, its process id is stored in a file so that it can be used to determine its status and possibly kill the process in the future:

PidFile "/opt/jbcs-httpd/httpd/run/httpd${index}.pid"

The file name is appended with 1 or 2 for the first and second instance of the web server, respectively.

This example follows a similar pattern in naming various log files:

#
For a single logfile with access, agent, and referer information
(Combined Logfile Format), use the following directive:
#
CustomLog logs/access_log${index} combined

#
ErrorLog: The location of the error log file.
If you do not specify an ErrorLog directive within a <VirtualHost>
container, error messages relating to that virtual host will be
logged here. If you **do** define an error logfile for a <VirtualHost>
container, that host's errors will be logged there and not here.
#
ErrorLog logs/error_log${index}

The next step in configuring the web server is to set values for a large number of parameters, and load a number of common modules that may be required. In the reference architecture, these

settings remain largely unchanged from the defaults that were downloaded in the archive files. One important common configuration is the document root, where the server can attempt to find and serve content to clients. Since the reference web server is exclusively used to forward requests to the application server, this parameter is pointed to an empty directory:

```
#
# DocumentRoot: The directory out of which you will serve your
# documents. By default, all requests are taken from this directory, but
# symbolic links and aliases may be used to point to other locations.
#
DocumentRoot "/var/www/html"
```

Now that the basic modules and some of the dependencies have been configured, the next step is to load **mod_cluster** modules and configure them for each instance. The required modules are:

```
# mod_proxy_balancer should be disabled when mod_cluster is used
LoadModule cluster_slotmem_module modules/mod_cluster_slotmem.so
LoadModule manager_module modules/mod_manager.so
LoadModule proxy_cluster_module modules/mod_proxy_cluster.so
LoadModule advertise_module modules/mod_advertise.so
```

MemManagerFile is the base name for the file names mod_manager uses to store configuration, generate keys for shared memory, or lock files. It must be an absolute path name; the referenced directories are created if required. It is highly recommended that those files be placed on a local drive and not an NFS share. This path must be unique for each instance of the web server, so again append 1 or 2 to the file name:

```
MemManagerFile /var/cache/mod_cluster${index}
```

The final remaining configuration for **mod_cluster** is its listen port, which is set to 6661 for the first instance and modified to 6662 for the second web server instance. Also add rules to the module configuration to permit connections from required IP addresses, in this case nodes 1, 2 and 3 of the EAP 7 cluster, or the internal network where they reside:

```
<IfModule manager_module>
  Listen *:666${index}
  <VirtualHost *:666${index}>
    <Directory />
      Require ip 10
    </Directory>
    ServerAdvertise off
    EnableMCPMReceive
    <Location /mod_cluster_manager>
      SetHandler mod_cluster-manager
      Require ip 10
   </Location>
  </VirtualHost>
</IfModule>
```

The reference architecture does not require any additional configuration files. It is particularly important, as noted above, that conflicting modules such as *mod_proxy_balancer* would not be loaded. If present, the *httpd/conf.d/mod_cluster.conf* sample configuration can introduce such a conflict, so comment out the line that loads all configuration files in the provided folder:

```
#
# Load config files from the config directory "/etc/httpd/conf.d".
#
#IncludeOptional conf.d/*.conf
```

Simultaneous execution of two web server instances requires that they are configured to use different HTTP and HTTPS ports. For the regular HTTP port, use *81* and *82* for the two instances, so the cluster configuration files are configured as follows:

```
# HTTP port
#
Listen 10.19.137.37:8${index}
```

Everything else in the web server configuration remains largely untouched. It is recommended that the parameters are reviewed in detail for a specific environment, particularly when there are changes to the reference architecture that might invalidate some of the accompanied assumptions.

This reference architecture does not make use of Secure Sockets Layer (SSL). To use SSL, appropriate security keys and certificates have to be issued and copied to the machine. To turn SSL off, the documentation recommends renaming *ssl.conf*. It is also possible to keep the SSL configuration file while setting *SSLEngine* to *off*:

```
# SSL Engine Switch:
# Enable/Disable SSL for this virtual host.
SSLEngine off
…
```

The *cluster2.conf* file is almost identical to *cluster1.conf*, while providing a different *index* variable value to avoid any port or file name conflict.

5.2. POSTGRESQL DATABASE

The PostgreSQL Database Server used in this reference architecture requires very little custom configuration. No files are copied to require a review, and there is no script-based configuration that changes the setup.

To enter the interactive database management mode, enter the following command as the postgres user:

```
$ /usr/bin/psql -U postgres postgres
```

With the database server running, users can enter both SQL and postgres instructions in this environment. Type help at the prompt to access to some of the documentation:

```
psql (9.2.15)
Type "help" for help.
```

Type \l and press enter to see a list of the databases along with the name of each database owner:

```
postgres=# \l
                              List of databases
   Name    |  Owner   | Encoding |  Collation  |    Ctype    |
Access privileges
-----------+----------+----------+-------------+-------------+------
-----------------
```

```
 eap7      | jboss    | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres  | postgres | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
…
```

To issue commands against a specific database, enter *c* followed by the database name. In this case, use the *eap7* database for the reference architecture:

```
postgres=# \c eap7
psql (9.2.15)
You are now connected to database "eap7".
```

To see a list of the tables created in the database, use the *dt* command:

```
postgres-# \dt
 List of relations
 Schema |  Name  | Type  | Owner
--------+--------+-------+-------
 public | person | table | jboss
(1 row)
```

The results for the above *dt* command are different depending on whether the EAP servers are started, the cluster testing application is deployed and any data is created through JPA. To further query the table, use standard SQL commands.

5.3. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM

5.3.1. Configuration Files

The CLI script in this reference architecture uses an embedded and offline domain controller for each of the domains to issue CLI instructions. The result is that the required server profiles are created and configured as appropriate for the cluster. This section reviews the various configuration files. The few manual steps involve copying the files from the master server to slave servers and adding application users, which were explained in the previous section.

This reference architecture includes two separate clusters, set up as two separate domains, where one is active and the other passive. The passive domain is a mirror image of the active one, other than minor changes to reflect the machines that are hosting it and avoid conflicting ports. The excerpts below reflect the active domain.

To start the active domain, first run the following command on *node1*, identified by the IP address of *10.19.137.34*:

```
# /opt/jboss-eap-7.0_active/bin/domain.sh -b 10.19.137.34 -
bprivate=10.19.137.34 -Djboss.bind.address.management=10.19.137.34
```

The *jboss.bind.address.management* system property indicates that this same server, running on *node1*, is the domain controller for this active domain. As such, all management users have been created on this node. This also means that the configuration for the entire domain is stored in a *domain.xml* file on this node. Furthermore, in the absence of a *--host-config=* argument, the configuration of this host is loaded from the local *host.xml* file. This section reviews each of these files.

To see a list of all the management users for this domain, look at the content of *domain/configuration/mgmt-users.properties*:

```
# Users can be added to this properties file at any time, updates after the server has started
# will be automatically detected.
#
# By default the properties realm expects the entries to be in the format: -
# username=HEX( MD5( username ':' realm ':' password))
#
# A utility script is provided which can be executed from the bin folder to add the users: -
# - Linux
# bin/add-user.sh
#
# - Windows
# bin\add-user.bat
#
# On start-up the server will also automatically add a user $local - this user is specifically
# for local tools running against this AS installation.
#
# The following illustrates how an admin user could be defined, this
# is for illustration only an does not correspond to a usable password.
#
#admin=2a0923285184943425d1f53ddd58ec7a
admin=e61d4e732bbcbd2234553cc024cbb092
node1=a10447f2ee947026f5add6d7366bc858
node2=5259fe18080faee6b9aafe3dc0e0edfe
node3=edd25e74df6c078685576891b5b82b48
```

Four users called *admin*, *node1*, *node2* and *node3* are defined in this property file. As the comments explain, the hexadecimal number provided as the value is a hash of each user's password (along with other information) used to authenticate the user. In this example, *password1!* is used as the password for all four users; the hash value contains other information, including the username itself, and that is why an identical hash is not generated for all users.

The *admin* user has been created to allow administrative access through the management console and the CLI interface.

There is a user account configured for each node of the cluster. These accounts are used by slave hosts to connect to the domain controller.

The application users are created and stored in the same directory, under *domain/configuration/*. The user name and password are stored in *application-users.properties*:

```
#
# Properties declaration of users for the realm 'ApplicationRealm' which is the default realm
# for application services on a new AS 7.1 installation.
#
# This includes the following protocols: remote ejb, remote jndi, web, remote jms
#
# Users can be added to this properties file at any time, updates after the server has started
# will be automatically detected.
#
# The format of this realm is as follows: -
# username=HEX( MD5( username ':' realm ':' password))
#
# …
#
# The following illustrates how an admin user could be defined, this
# is for illustration only and does not correspond to a usable password.
#
```

```
#admin=2a0923285184943425d1f53ddd58ec7a
ejbcaller=ef4f1428b4b43d020d895797d19d827f
```

There is only one application user *ejbcaller* and it is used to make remote calls to EJB. The provided EJBs do not use fine-grained security and therefore, no roles need to be assigned to this user to invoke their operations. This makes the *application-roles.properties* file rather simple:

```
# Properties declaration of users roles for the realm 'ApplicationRealm'.
#
# This includes the following protocols: remote ejb, remote jndi, web, remote jms
#
# …
#
# The format of this file is as follows: -
# username=role1,role2,role3
#
# The following illustrates how an admin user could be defined.
#
#admin=PowerUser,BillingAdmin,
#guest=guest
ejbcaller=
```

Application users are required on each server that hosts the EJB, these two files are therefore generated on all six EAP installations.

5.3.1.1. Host files

As previously mentioned, the domain controller is started with the default host configuration file. This file is modified to change the server name by the CLI script. Each section of this file is inspected separately.

```
<?xml version='1.0' encoding='UTF-8'?>
<host name="node1" xmlns="urn:jboss:domain:4.1">
```

The security configuration remains unchanged from its default values, but a review that provides a better understanding of the authentication section is still useful. As seen below under authentication, the management realm uses both the *mgmt-users.properties* file, previously reviewed, as well as a local default user configuration. This local user enables silent authentication when the client is on the same machine. As such, the user *node1*, although created, is not necessary for the active domain. It is created for the sake of consistency across the two domains. For further information about silent authentication of a local user, refer to the product documentation.

```
<management>
  <security-realm name="ManagementRealm">
    <authentication>
      <local default-user="$local" skip-group-loading="true"/>
      <properties path="mgmt-users.properties" relative-to="jboss.domain.config.dir"/>
    </authentication>
    <authorization map-groups-to-roles="false">
      <properties path="mgmt-groups.properties" relative-to="jboss.domain.config.dir"/>
    </authorization>
 </security-realm>
```

The rest of the security configuration simply refers to the application users, and once again, it is unchanged from the default values.

```
    <security-realm name="ApplicationRealm">
        <authentication>
            <local default-user="$local" allowed-users="*" skip-group-loading="true"/>
            <properties path="application-users.properties" relative-to="jboss.domain.config.dir"/>
        </authentication>
        <authorization>
            <properties path="application-roles.properties" relative-to="jboss.domain.config.dir"/>
        </authorization>
    </security-realm>
</security-realms>
```

Management interfaces are also configured as per the default and are not modified in the setup.

```
    <management-interfaces>
        <native-interface security-realm="ManagementRealm">
            <socket interface="management" port="${jboss.management.native.port:9999}"/>
        </native-interface>
        <http-interface security-realm="ManagementRealm">
            <socket interface="management" port="${jboss.management.http.port:9990}"/>
        </http-interface>
    </management-interfaces>
</management>
```

With node1 being the domain controller, the configuration file explicitly reflects this fact and points to <local/>. This is the default setup for the *host.xml* in EAP 7:

```
<domain-controller>
    <local/>
</domain-controller>
```

Interfaces and JVM configuration are also left unchanged.

The automated configuration scripts remove the sample servers that are configured by default in JBoss EAP 7 and instead create a new server as part of this cluster. This server is named in a way that reflects both the node on which it runs as well as the domain to which it belongs. All servers are configured to auto start when the host starts up:

```
<server name="node1" group="demo-cluster" auto-start="true">
    <socket-bindings port-offset="0"/>
</server>
```

To start the other nodes of the active domain, after the domain controller has been started, run the following commands on nodes 2 and 3, identified by IP addresses of *10.19.137.35* and *10.19.137.36*:

```
# /opt/jboss-eap-7.0_active/bin/domain.sh -b 10.19.137.35 -
bprivate=10.19.137.35 -Djboss.domain.master.address=10.19.137.34 --
host-config=host-slave.xml # /opt/jboss-eap-7.0_active/bin/domain.sh -b
10.19.137.36 -bprivate=10.19.137.36 -
Djboss.domain.master.address=10.19.137.34 –host-config=host-slave.xml
```

These two nodes are configured similarly, so it suffices to look at the host configuration file for one of them:

```
<?xml version='1.0' encoding='UTF-8'?>
<host name="node2" xmlns="urn:jboss:domain:4.1">
```

Once again, the host name is modified by CLI to reflect the host name. In the case of nodes 2 and 3, the name selected for the host is not only for display and informational purposes. This host name is also picked up as the user name that is used to authenticate against the domain controller. The hash value of the the user name and its associated password is provided as the secret value of the server identity:

```
<management>
<security-realms>
   <security-realm name="ManagementRealm">
      <server-identities>
         <secret value="cGFzc3dvcmQxIQ==">/>
      </server-identities>
```

So in this case, *node2* authenticates against the domain controller running on *node1* by providing *node2* as its username and the above secret value as its password hash. This value is copied from the output of the *add-user* script when *node2* is added as a user.

The rest of the security configuration remains unchanged. Management interface configuration also remains unchanged.

The domain controller is identified by a Java system property (-Djboss.domain.master.address) that is passed to the node2 Java virtual machine (JVM) when starting the server:

```
<domain-controller>
   <remote security-realm="ManagementRealm">
      <discovery-options>
         <static-discovery name="primary" protocol="${jboss.domain.master.protocol:remote}"
host="${jboss.domain.master.address}" port="${jboss.domain.master.port:9999}"/>
      </discovery-options>
   </remote>
</domain-controller>
```

Automated configuration scripts remove the sample servers of JBoss EAP 7 and instead create a new server as part of this cluster:

```
<servers>
  <server name="node2" group="demo-cluster" auto-start="true">
     <socket-bindings port-offset="0"/>
  </server>
</servers>
```

5.3.1.2. Domain configuration file

The bulk of the configuration for the cluster and the domain is stored in its *domain.xml* file. This file can only be edited when the servers are stopped, and is best modified through the management capabilities, which include the management console, CLI and various scripts and languages available on top of CLI.

The first section of the domain configuration file lists the various extensions that are enabled for this domain. This section is left unchanged and includes all the standard available extensions:

```
<?xml version="1.0" ?>
```

```
<domain xmlns="urn:jboss:domain:4.1">
   <extensions>
      …
   </extensions>
```

System properties are also left as default:

```
<system-properties>
   <property name="java.net.preferIPv4Stack" value="true"/>
</system-properties>
```

The next section of the domain configuration is the profiles section. JBoss EAP 7 ships with four pre-configured profiles. Since this reference architecture only uses the full-ha profile, the other three profiles have been removed to make the domain.xml easier to read and maintain.

```
<profile name="default">
<profile name="ha">
<profile name="full">
<profile name="full-ha">
```

The configuration script updates parts of the full-ha profile, but most of the profile remains unchanged from the *full-ha* baseline. For eample, the *ExampleDS* datasource, using an in-memory H2 database, remains intact but is not used in the cluster.

The configuration script deploys a JDBC driver for postgres, and configures a connection pool to the *eap7* database using the database owner's credentials:

```
<datasource jndi-name="java:jboss/datasources/ClusterDS" pool-name="eap7" use-ccm="false">
  <connection-url>jdbc:postgresql://10.19.137.37:5432/eap7</connection-url>
  <driver>postgresql-9.4.1208.jar</driver>
  <new-connection-sql>set datestyle = ISO, European;</new-connection-sql>
  <pool>
     <max-pool-size>25</max-pool-size>
  </pool>
  <security>
     <user-name>jboss</user-name>
     <password>password</password>
  </security>
  <timeout>
     <blocking-timeout-millis>5000</blocking-timeout-millis>
  </timeout>
</datasource>
```

The driver element includes the name of the JAR (Java ARchive) file where the driver class can be found. This creates a dependency on the JDBC driver JAR file and assumes that it has been separately deployed. There are multiple ways to make a JDBC driver available in EAP 7. The approach here is to deploy the driver, as with any other application.

The *ExampleDS* connection pool uses a different approach for its driver. The JDBC driver of the H2 Database is made available as a module. As such, it is configured under the drivers section of the domain configuration:

```
      <drivers>
         <driver name="h2" module="com.h2database.h2">
            <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
```

```
            </driver>
        </drivers>
    </datasources>
</subsystem>
```

After datasources, the profile configures the Java EE and EJB3 subsystems. This configuration is derived from the standard *full-ha* profile and unchanged.

The *Infinispan* subsystem is also left unchanged. This section configures several standard cache containers and defines the various available caches for each container. It also determines which cache is used by each container. Clustering takes advantage of these cache containers and as such, uses the configured cache of that container.

The cache container below is for applications that use a cluster singleton service:

```
<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>
```

Next in the Infinispan subsystem is the web cache container. It is used to replicate HTTP session data between cluster nodes:

```
    <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>
```

Next is the *ejb* cache and used to replicate stateful session bean data.

```
<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>
```

Finally, the last cache container in the subsystem is the *hibernate* cache and is used for replicating second-level cache data. Hibernate makes use of an entity cache to cache data on each node and invalidate other nodes of the cluster when the data is modified. It also has an optional query cache:

```
<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>
```

Several other subsystems are then configured in the profile, without any modification from the baseline *full-ha* profile:

The next subsystem configured in the profile is messaging, Activemq. Two JMS servers are configured using the shared-store HA policy, one is a live JMS server (named "default").

```
<server name="default">
    <cluster password="password" user="admin"/>
    <shared-store-master failover-on-server-shutdown="true"/>
    <bindings-directory path="/mnt/activemq/sharefs/group-${live-group}/bindings"/>
    <journal-directory path="/mnt/activemq/sharefs/group-${live-group}/journal"/>
    <large-messages-directory path="/mnt/activemq/sharefs/group-${live-group}/largemessages"/>
    <paging-directory path="/mnt/activemq/sharefs/group-${live-group}/paging"/>
    <security-setting name="#">
        <role name="guest" delete-non-durable-queue="true" create-non-durable-queue="true"
consume="true" send="true"/>
    </security-setting>
    <address-setting name="#" redistribution-delay="0" message-counter-history-day-limit="10" page-
size-bytes="2097152" max-size-bytes="10485760" expiry-address="jms.queue.ExpiryQueue" dead-
letter-address="jms.queue.DLQ"/>
    <http-connector name="http-connector" endpoint="http-acceptor" socket-binding="http"/>
    <http-connector name="http-connector-throughput" endpoint="http-acceptor-throughput" socket-
binding="http">
        <param name="batch-delay" value="50"/>
    </http-connector>
    <remote-connector name="netty" socket-binding="messaging">
        <param name="use-nio" value="true"/>
        <param name="use-nio-global-worker-pool" value="true"/>
    </remote-connector>
    <in-vm-connector name="in-vm" server-id="0"/>


    <http-acceptor name="http-acceptor" http-listener="default"/>
    <http-acceptor name="http-acceptor-throughput" http-listener="default">
        <param name="batch-delay" value="50"/>
        <param name="direct-deliver" value="false"/>
    </http-acceptor>
    <remote-acceptor name="netty" socket-binding="messaging">
        <param name="use-nio" value="true"/>
    </remote-acceptor>
    <in-vm-acceptor name="in-vm" server-id="0"/>
    <broadcast-group name="bg-group1" connectors="netty" jgroups-channel="activemq-cluster"/>
    <discovery-group name="dg-group1" jgroups-channel="activemq-cluster"/>
    <cluster-connection name="amq-cluster" discovery-group="dg-group1" connector-name="netty"
address="jms"/>
    <jms-queue name="ExpiryQueue" entries="java:/jms/queue/ExpiryQueue"/>
```

```
    <jms-queue name="DLQ" entries="java:/jms/queue/DLQ"/>
    <jms-queue name="DistributedQueue" entries="java:/queue/DistributedQueue"/>
    <connection-factory name="InVmConnectionFactory" entries="java:/ConnectionFactory"
connectors="in-vm"/>
    <connection-factory name="RemoteConnectionFactory" reconnect-attempts="-1" block-on-
acknowledge="true" ha="true" entries="java:jboss/exported/jms/RemoteConnectionFactory"
connectors="netty"/>
    <pooled-connection-factory name="activemq-ra" transaction="xa" entries="java:/JmsXA
java:jboss/DefaultJMSConnectionFactory" connectors="in-vm"/>
</server>
```

And the other is a backup JMS server (named "backup").

```
<server name="backup">
    <cluster password="password" user="admin"/>
    <shared-store-slave failover-on-server-shutdown="true"/>
    <bindings-directory path="/mnt/activemq/sharefs/group-${backup-group}/bindings"/>
    <journal-directory path="/mnt/activemq/sharefs/group-${backup-group}/journal"/>
    <large-messages-directory path="/mnt/activemq/sharefs/group-${backup-
group}/largemessages"/>
    <paging-directory path="/mnt/activemq/sharefs/group-${backup-group}/paging"/>
    <remote-connector name="netty-backup" socket-binding="messaging-backup"/>
    <remote-acceptor name="netty-backup" socket-binding="messaging-backup"/>
    <broadcast-group name="bg-group-backup" connectors="netty-backup" broadcast-period="1000"
jgroups-channel="activemq-cluster"/>
    <discovery-group name="dg-group-backup" refresh-timeout="1000" jgroups-channel="activemq-
cluster"/>
    <cluster-connection name="amq-cluster" discovery-group="dg-group-backup" connector-
name="netty-backup" address="jms"/>
</server>
```

The live JMS server uses the *shared-store-master* tag and the backup JMS server use the *shared-store-slave* tag. With the *shared-store* HA policy, when the live and backup JMS servers share the same file path, they form a live-backup group.

```
<shared-store-master failover-on-server-shutdown="true"/>

<shared-store-slave failover-on-server-shutdown="true"/>
```

To have a JMS HA Cluster and to set up 3 nodes, one solution is to set up 3 copies of the full-ha profiles, almost identical to each other, but different in their messaging-activemq subsystem. This would allow the 3 profiles to be modified indiviually, following the diagrom below to form a 3 node JMS cluster: each profile will have 1 active jms server and 1 backup server.

However a simpler setup is possible that is both easier to understand and maintain. This reference architecture is set up with a single full-ha profile, with 2 JMS servers inside, one live and one backup JMS server. The same profile will apply to all 3 nodes, thus 1 live server and 1 backup server on each node. The key point is using variables to make sure that at runtime, the live and backup servers belong to different groups and form a live-backup pair with a different JVM.

Please note the use of variables in this reference architecture’s shared-store file path, which 's the way to make sure a backup JMS server won’t be in the same JVM as the live JMS server. Of course with a live-backup pair within the same JVM, failure at JVM or OS level will cause outage to both the live and its backup JMS server at same time and the backlog of messages being processed by that server will not be picked up by any other node, something that must be avoided in a true HA environment.

```
<bindings-directory path="/mnt/activemq/sharefs/group-${live-group}/bindings"/>

<bindings-directory path="/mnt/activemq/sharefs/group-${backup-group}/bindings"/>
```

These 2 variables, live-group and backup-group, are runtime values are from the host.xml of the DMC node and host-slave.xml of slave nodes, defined inside each server tag.

```
<server name="node1" group="demo-cluster" auto-start="true">
  <system-properties>
    <property name="live-group" value="3"/>
    <property name="backup-group" value="4"/>
  </system-properties>
```

Sample values for the active domain: in this setup, the second JMS server of node1 will be the backup server for the live server of node2, the second JMS server of node2 will be the backup server for the live server of node3, and the second JMS server of node3 will be the backup server for the live server of node1.

```
node1 (DMC node):
    live-group: 1
    backup-group: 2
node2:
    live-group: 2
    backup-group: 3
node3:
    live-group: 3
    backup-group: 1
```

The cluster-connection for both JMS servers use separately defined remote-connector configurations instead of the default http-connector, to avoid conflict on the http port.

```
<cluster-connection name="amq-cluster" discovery-group="dg-group1" connector-name="netty"
address="jms"/>

<remote-acceptor name="netty" socket-binding="messaging"/>

<cluster-connection name="amq-cluster" discovery-group="dg-group-backup" connector-
name="netty-backup" address="jms"/>

<remote-connector name="netty-backup" socket-binding="messaging-backup"/>
```

After messaging, the next subsystem configuration in the domain configuration file is *mod_cluster*, which is used to communicated with JWS apache http server:

```
<subsystem xmlns="urn:jboss:domain:modcluster:2.0">
   <mod-cluster-config advertise-socket="modcluster" proxies="mod-cluster" advertise="false"
connector="ajp">
       <dynamic-load-provider>
          <load-metric type="cpu"/>
       </dynamic-load-provider>
   </mod-cluster-config>
</subsystem>
```

The rest of the profile remains unchanged from the baseline *full-ha* configuration.

The domain configuration of the available interfaces is also based on the defaults provided in the *full-ha* configuration:

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

The four socket binding groups provided by default, remain configured in the setup, but only *full-ha-sockets* is used, and the other items are ignored.

The *full-ha-sockets* configures a series of port numbers for various protocols and services. Most of these port numbers are unchanged in the profiles and only three are modified. *Jgroups-mping* and *jgroups-udp* are changed to for cluster communication. *mod-cluster* affects full-ha profile's modcluster subsystem, which is used to communicate with the JWS Apache http server.

```
  <socket-binding-group name="full-ha-sockets" default-interface="public">
  …
     <socket-binding name="jgroups-mping" port="0" multicast-
address="${jboss.default.multicast.address:230.0.0.1}" multicast-port="45700"/>
  …
     <socket-binding name="jgroups-udp" port="55200" multicast-
address="${jboss.default.multicast.address:230.0.0.1}" multicast-port="45688"/>
  …
     <outbound-socket-binding name="mod-cluster">
    <remote-destination host="10.19.137.37" port="6661"/>
     </outbound-socket-binding>
  </socket-binding-group>
```

Finally, the new server group is added and replaces the default ones:

```
   <server-groups>
      <server-group name="demo-cluster" profile="full-ha">
         <jvm name="default">
            <heap size="66m" max-size="512m"/>
         </jvm>
         <socket-binding-group ref="full-ha-sockets"/>
         <deployments>
            <deployment name="postgresql-9.4.1208.jar" runtime-name="postgresql-9.4.1208.jar"/>
            <deployment name="clusterApp.war" runtime-name="clusterApp.war"/>
            <deployment name="chat.war" runtime-name="chat.war"/>
         </deployments>
      </server-group>
   </server-groups>
```

This server group has three applications deployed:

- The first deployment, *postgresql-9.4.1208.jar* is the JDBC driver JAR file for PostgreSQL Database. The configured datasource called *ClusterDS* relies on this driver, and declares its dependency on it.

- The second deployment *clusterApp.war* is a clustered application deployed by the configuration script, to validate and demonstrate clustering capabilities. It includes a distributable web application, stateless and clustered stateful sessions beans, a JPA bean with second-level caching enabled and MDBs to consume messages from a distributed queue.
- The third deployment is *chat.war*, which is used to demonstrate the WebSocket feature of Java EE 7.

The second and passive cluster/domain is configured through the same script with a slightly modified property file. The only properties that is changed for the second domain are as follows:

domainController: the IP address of the domain controller, running on node2
domainName: *passive* for the passive domain
offsetUnit: *1* for the passive domain
modClusterProxy: The *host:port* value of the proxy for the passive cluster

5.4. CONFIGURATION SCRIPTS (CLI)

5.4.1. Overview

The **management Command Line Interface (CLI)** is a command line administration tool for JBoss EAP 7. The Management CLI can be used to start and stop servers, deploy and undeploy applications, configure system settings, and perform other administrative tasks.

A CLI operation request consists of three parts:

- an address, prefixed with a slash “/”.
- an operation name, prefixed with a colon “:”.
- an optional set of parameters, contained within parentheses “()”.

The configuration is presented as a hierarchical tree of addressable resources. Each resource node offers a different set of operations. The address specifies which resource node to perform the operation on. An address uses the following syntax:

/node-type=node-name

- node-type is the resource node type. This maps to an element name in the configuration XML.
- node-name is the resource node name. This maps to the name attribute of the element in the configuration XML.
- Separate each level of the resource tree with a slash “/”.

Each resource may also have a number of attributes. The read-attribute operation is a global operation used to read the current runtime value of a selected attribute.

5.4.2. Command-line interface (CLI) Scripts

This reference architecture uses CLI script to setup the whole domain structure.

JBoss EAP 7 introduced embed-host-controller for CLI scripts, also referred to as **Offline CLI**, which allows the setup of domain environments without the need of starting a domain first.

There are two ways to invoke the embed-host-controller to configure the domain.

1) To set up *domain.xml* on DMC box:

```
# /opt/jboss-eap-7.0/bin/jboss-cli.sh embed-host-controller --domain-
config=domain.xml --host-config=host.xml --std-out=echo
```

2) To set up *host-slave.xml* on DMC box, so that after running the CLI, the *host-slave.xml* can be manually copied to slave boxes:

```
# /opt/jboss-eap-7.0/bin/jboss-cli.sh -
Djboss.domain.master.address=10.19.137.34 embed-host-controller --
domain-config=domain.xml --host-config=host-slave.xml --std-out=echo
```

5.4.2.1. Shell Scripts

For this reference architecture, the CLI is invoked through the *configure.sh* shell script in both cases.

The configure.sh script performs the following tasks in order:

- Stop all running jboss threads

```
# pkill -f 'jboss'
```

- Clean up the environment by removing the CONF_HOME and server folder
- Restore the last good configuration by copying it from CONF_BACKUP_HOME

```
if [ -d $CONF_BACKUP_HOME ]
then
   echo "========configuration to remove: " $CONF_HOME
   rm -rf $CONF_HOME
   echo "========restore from: " $CONF_BACKUP_HOME
   cp -r $CONF_BACKUP_HOME $CONF_HOME
else
   echo "========No configuration backup at " $CONF_BACKUP_HOME
fi
ls -l  $JBOSS_HOME/domain
echo "========clean servers folder"
rm -rf $JBOSS_HOME/domain/servers/*
```

- Build the test clients and cluster application using maven

```
echo "========build deployments if needed"
if [ -f ${SCRIPT_DIR}/code/webapp/target/clusterApp.war ]
then
   echo "========clusterApp.war is already built"
else
   mvn -f ${SCRIPT_DIR}/code/pom.xml install
fi
```

- Set up the master server's *domain.xml* and *host.xml* by calling *setup-master.cli*

```
$JBOSS_HOME/bin/jboss-cli.sh --
properties=$SCRIPT_DIR/cluster.properties --file=$SCRIPT_DIR/setup-
master.cli -Dhost.file=host.xml
THIS_HOSTNAME=`$JBOSS_HOME/bin/jboss-cli.sh  -
```

```
Djboss.domain.master.address=127.0.0.1 --commands="embed-host-
controller --domain-config=domain.xml --host-config=host-
slave.xml,read-attribute local-host-name"`
```

- Set up the slave servers' *host-slave.xml* by calling *setup-slave.cli*

```
. $SCRIPT_DIR/slaves.properties
for slave in $(echo $slaves | sed "s/,/ /g")
do
        #Variable names for each node's name and identity secret:
        name="${slave}_name"
        secret="${slave}_secret"
        echo Will configure ${slave}
        #Back up host-slave as it will be overwritten by CLI:
        cp $CONF_HOME/host-slave.xml $CONF_HOME/host-slave-orig.xml

        $JBOSS_HOME/bin/jboss-cli.sh --file=$SCRIPT_DIR/setup-
slave.cli -Djboss.domain.master.address=127.0.0.1 -Dhost.file=host-
slave.xml -Dold.host.name=$THIS_HOSTNAME -Dhost.name=${!name} -
Dserver.identity=${!secret} -Dlive.group=${!liveGroup} -
Dbackup.group=${!backupGroup} --
properties=$SCRIPT_DIR/cluster.properties


        #Copy final host-slave file to a directory for this node,
restore the original
        mkdir $CONF_HOME/$slave
        mv $CONF_HOME/host-slave.xml $CONF_HOME/$slave/
        mv $CONF_HOME/host-slave-orig.xml $CONF_HOME/host-slave.xml
done
```

5.4.2.2. CLI Scripts

1) *setup-master.cli*

- Remove default *server-group* and servers that belong it, also remove the 3 profiles that are not needed in here.

```
/host=master/server-config=server-three/:remove
/server-group=other-server-group/:remove
/host=master/server-config=server-one/:remove
/host=master/server-config=server-two/:remove
/server-group=main-server-group/:remove

/profile=default:remove
/profile=full:remove
/profile=ha:remove
```

- Configure multicast address

```
/socket-binding-group=full-ha-sockets/socket-binding=jgroups-
mping:write-attribute(name=multicast-
address,value="${jboss.default.multicast.address:"$multicastAddress}
/socket-binding-group=full-ha-sockets/socket-binding=jgroups-udp:write-
```

```
attribute(name=multicast-
address,value="${jboss.default.multicast.address:"$multicastAddress}
```

- Set up mod-cluster for apache

```
/socket-binding-group=full-ha-sockets/remote-destination-outbound-
socket-binding=mod-cluster:add(host=$lbHost,port=$lbPort)
/profile=full-ha/subsystem=modcluster/mod-cluster-
config=configuration:write-attribute(name=advertise,value=false)
/profile=full-ha/subsystem=modcluster/mod-cluster-
config=configuration:list-add(index=0,name=proxies,value=mod-cluster)
```

- Create demo-cluster server group

```
/server-group=demo-cluster:add(profile=full-ha,socket-binding-
group=full-ha-sockets)
/server-group=demo-cluster/jvm=default:add(heap-size=66m, max-heap-
size=512m)
```

- Add server to this server group

```
/host=master/server-config=$serverName:add(group=demo-cluster,auto-
start=true)
/host=master/server-config=$serverName:write-attribute(name=socket-
binding-port-offset,value=$portOffset)
```

- Configure host management interface

```
/host=master/core-service=management/management-interface=http-
interface:write-
attribute(name=port,value="${jboss.management.http.port:"$mgmtHttpPort}
/host=master/core-service=management/management-interface=native-
interface:write-
attribute(name=port,value="${jboss.management.native.port:"$mgmtNativeP
ort}
```

- Add live-group and backup-group system parameters for ActiveMQ backup grouping

```
/host=master/server-config=$serverName/system-property=live-
group:add(value=$liveGroup)
/host=master/server-config=$serverName/system-property=backup-
group:add(value=$backupGroup)
```

- Add postgresql datasource

```
deploy ${env.SCRIPT_DIR}/postgresql-9.4.1208.jar  --server-groups=demo-
cluster
data-source add --profile=full-ha --name=eap7 --driver-name=postgresql-
9.4.1208.jar --connection-url=$psqlConnectionUrl --jndi-
name=java:jboss/datasources/ClusterDS --user-name=jboss --
password=password --use-ccm=false --max-pool-size=25 --blocking-
timeout-wait-millis=5000 --new-connection-sql="set datestyle = ISO,
European;"
holdback-batch deploy-postgresql-driver
```

- Add 2 socket bindings for ActiveMQ

```
 /socket-binding-group=full-ha-sockets/socket-binding=messaging:add
 /socket-binding-group=full-ha-sockets/socket-binding=messaging:write-
attribute(name=port,value=$messagingPort)
 /socket-binding-group=full-ha-sockets/socket-binding=messaging-
backup:add
 /socket-binding-group=full-ha-sockets/socket-binding=messaging-
backup:write-attribute(name=port,value=$messagingBackupPort)
```

- Set up ActiveMQ live jms server

```
 /profile=full-ha/subsystem=messaging-activemq/server=default:write-
attribute(name=cluster-user,value=$clusterUser)
 /profile=full-ha/subsystem=messaging-activemq/server=default:write-
attribute(name=cluster-password,value=$clusterPassword)

add shared-store-master /profile=full-ha/subsystem=messaging-
activemq/server=default/ha-policy=shared-store-master:add(failover-on-
server-shutdown=true) #change the value from 1000 (default) to 0
/profile=full-ha/subsystem=messaging-activemq/server=default/address-
setting=:write-attribute(name=redistribution-delay,value=0)


#add shared files path
 /profile=full-ha/subsystem=messaging-
activemq/server=default/path=bindings-
directory:add(path=$shareStorePath/group-${live-group}/bindings)
 /profile=full-ha/subsystem=messaging-
activemq/server=default/path=journal-
directory:add(path=$shareStorePath/group-${live-group}/journal)
 /profile=full-ha/subsystem=messaging-
activemq/server=default/path=large-messages-
directory:add(path=$shareStorePath/group-${live-group}/largemessages)
 /profile=full-ha/subsystem=messaging-
activemq/server=default/path=paging-
directory:add(path=$shareStorePath/group-${live-group}/paging)
#add remote connector netty
 /profile=full-ha/subsystem=messaging-activemq/server=default/remote-
connector=netty:add(socket-binding=messaging)
 /profile=full-ha/subsystem=messaging-activemq/server=default/remote-
connector=netty:write-attribute(name=params,value={use-nio=true,use-
nio-global-worker-pool=true})
# /profile=full-ha/subsystem=messaging-activemq/server=default/remote-
connector=netty:write-attribute(name=params,value={name=,value=true})
#add remote acceptor netty
 /profile=full-ha/subsystem=messaging-activemq/server=default/remote-
acceptor=netty:add(socket-binding=messaging)
 /profile=full-ha/subsystem=messaging-activemq/server=default/remote-
acceptor=netty:write-attribute(name=params,value={use-nio=true})

 /profile=full-ha/subsystem=messaging-activemq/server=default/jms-
queue=DistributedQueue/:add(entries=["java:/queue/DistributedQueue"])
 /profile=full-ha/subsystem=messaging-activemq/server=default/cluster-
connection=my-cluster:remove
 /profile=full-ha/subsystem=messaging-activemq/server=default/cluster-
```

```
connection=amq-cluster:add(discovery-group=dg-group1,connector-
name=netty,cluster-connection-address=jms)
#update the connectors from http-connector to netty
 /profile=full-ha/subsystem=messaging-
activemq/server=default/broadcast-group=bg-group1:write-
attribute(name=connectors,value=["netty"])
 /profile=full-ha/subsystem=messaging-
activemq/server=default/connection-
factory=RemoteConnectionFactory:write-attribute(name=connectors,value=
["netty"])
```

- Set up ActiveMQ backup jms server

```
 /profile=full-ha/subsystem=messaging-
activemq/server=backup:add(cluster-user=$clusterUser,cluster-
password=$clusterPassword)
#add shared-store-slave
 /profile=full-ha/subsystem=messaging-activemq/server=backup/ha-
policy=shared-store-slave:add(allow-failback=true,failover-on-server-
shutdown=true)

#add shared files path
 /profile=full-ha/subsystem=messaging-
activemq/server=backup/path=bindings-
directory:add(path=$shareStorePath/group-${backup-group}/bindings)
 /profile=full-ha/subsystem=messaging-
activemq/server=backup/path=journal-
directory:add(path=$shareStorePath/group-${backup-group}/journal)
 /profile=full-ha/subsystem=messaging-
activemq/server=backup/path=large-messages-
directory:add(path=$shareStorePath/group-${backup-group}/largemessages)
 /profile=full-ha/subsystem=messaging-
activemq/server=backup/path=paging-
directory:add(path=$shareStorePath/group-${backup-group}/paging)
#add remote connector netty
 /profile=full-ha/subsystem=messaging-activemq/server=backup/remote-
connector=netty-backup:add(socket-binding=messaging-backup)
#add remote acceptor netty
 /profile=full-ha/subsystem=messaging-activemq/server=backup/remote-
acceptor=netty-backup:add(socket-binding=messaging-backup)
#add broadcast-group
 /profile=full-ha/subsystem=messaging-activemq/server=backup/broadcast-
group=bg-group-backup:add(connectors=["netty-backup"],broadcast-
period="1000",jgroups-channel="activemq-cluster")
#add discovery-group
 /profile=full-ha/subsystem=messaging-activemq/server=backup/discovery-
group=dg-group-backup:add(refresh-timeout="1000",jgroups-
channel="activemq-cluster")
#add cluster-connection
 /profile=full-ha/subsystem=messaging-activemq/server=backup/cluster-
connection=amq-cluster:add(discovery-group=dg-group-backup,connector-
name=netty-backup,cluster-connection-address=jms)
```

- Update transactions subsytem’s attribute node-identifier to supress warning

```
 /profile=full-ha/subsystem=transactions:write-attribute(name=node-
identifier,value=node-identifier-${live-group})
```

- Deploy WAR files

```
deploy ${env.SCRIPT_DIR}/code/webapp/target/clusterApp.war  --server-
groups=demo-cluster
deploy ${env.SCRIPT_DIR}/chat.war  --server-groups=demo-cluster
```

2) *setup-slave.cli*

- Update *ManagementRealm serverIdentity* with password hash from *add-user.sh*

```
/host=$oldHostName/core-service=management/security-
realm=ManagementRealm/server-identity=secret:write-
attribute(name=value,value=$serverIdentity)
```

- Set management port

```
/host=$oldHostName/core-service=management/management-interface=native-
interface:write-
attribute(name=port,value="${jboss.management.native.port:"$mgmtNativeP
ort}
```

- Add correct server and remove two default servers

```
/host=$oldHostName/server-config=$hostName-server:add(auto-
start=true,group=demo-cluster)
/host=$oldHostName/server-config=$hostName-server:write-
attribute(name=socket-binding-port-offset,value=$portOffset)
/host=$oldHostName/server-config=server-one:remove()
/host=$oldHostName/server-config=server-two:remove()
```

- Add live-group and backup-group system parameters for ActiveMQ backup grouping

```
/host=$oldHostName/server-config=$hostName-server/system-property=live-
group:add(value=$liveGroup)
/host=$oldHostName/server-config=$hostName-server/system-
property=backup-group:add(value=$backupGroup)
```

- Update host name in the host-slave.xml

```
/host=$oldHostName:write-attribute(name=name,value=$hostName)
```

CHAPTER 6. CLUSTERING APPLICATIONS

6.1. OVERVIEW

While much of the HA behavior provided by a container is transparent, developers must remain aware of the distributed nature of their applications and at times, might even be required to annotate or designate their applications as distributable.

This reference architecture includes and deploys a web application called *clusterApp.war*, which makes use of the cluster capabilities of several different container components.

The web application includes a servlet, a stateful session bean, a JPA bean that is front-ended by a stateless session bean and an MDB. The persistence unit is configured with a second-level cache.

6.2. HTTP SESSION CLUSTERING

The *ClusteredServlet*, included and deployed as part of *clusterApp*, is a simple Java servlet that creates an HTTP session and saves and retrieves data from it.

```
import java.io.IOException;
import java.io.PrintWriter;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Enumeration;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@WebServlet("/*")
public class ClusteredServlet extends HttpServlet
{
 private static final long serialVersionUID = 1L;
...
```

This servlet handles both *GET* and *POST* requests. Upon receiving a request, it looks for an HTTP session associated with the user. If an existing session is not found, it creates a new session and stores the current time, along with the name of the current EAP server, in that session:

```
HttpSession session = request.getSession( false );
if( session == null )
{
 session = request.getSession( true );
 session.setAttribute( "initialization", new Date() );
 session.setAttribute( "initial_server", System.getProperty(
"jboss.server.name" ) );
}
```

While these two parameters are sufficient to cause and demonstrate replication of session data, the servlet also allows clients to add other key/value pairs of data to the session:

```
if( request.getParameter( "save" ) != null )
{
 String key = request.getParameter( "key" );
 String value = request.getParameter( "value" );
 if( key.length() > 0 )
 {
  if( value.length() == 0 )
  {
   session.removeAttribute( key );
  }
  else
  {
   session.setAttribute( key, value );
  }
 }
}
```

The servlet uses the *jboss.server.name* property to determine the name of the JBoss EAP server that is being reached on every invocation:

```
PrintWriter writer = response.getWriter();
writer.println( "<html>" );
writer.println( "<head>" );
writer.println( "</head>" );
writer.println( "<body>" );
StringBuilder welcomeMessage = new StringBuilder();
welcomeMessage.append( "HTTP Request received " );
welcomeMessage.append( TIME_FORMAT.format( new Date() ) );
welcomeMessage.append( " on server <b>" );
welcomeMessage.append( System.getProperty( "jboss.server.name" ) );
welcomeMessage.append( "</b>" );
writer.println( welcomeMessage.toString() );
writer.println( "<p/>" );
```

An HTML form is rendered to facilitate the entry to additional data into the session:

```
writer.println( "<form action='' method='post'>" );
writer.println( "Store value in HTTP session:<br/>" );
writer.println( "Key: <input type=\"text\" name=\"key\"><br/>" );
writer.println( "Value: <input type=\"text\" name=\"value\"><br/>" );
writer.println( "<input type=\"submit\" name=\"save\" value=\"Save\">"
);
writer.println( "</form>" );
```

All session attributes are displayed as an HTML table by the servlet, so that they can be inspected and verified every time:

```
writer.println( "<table border='1'>" );
Enumeration<String> attrNames = session.getAttributeNames();
while( attrNames.hasMoreElements() )
{
 writer.println( "<tr>" );
 String name = (String)attrNames.nextElement();
 Object value = session.getAttribute( name );
 if( value instanceof Date )
```

```
 {
  Date date = (Date)value;
  value = TIME_FORMAT.format( date );
 }
 writer.print( "<td>" );
 writer.print( name );
 writer.println( "</td>" );
 writer.print( "<td>" );
 writer.print( value );
 writer.println( "</td>" );
 writer.println( "</tr>" );
}
writer.println( "</table>" );
writer.println( "</body>" );
writer.println( "</html>" );
```

Finally, the response content type is set appropriately, and the content is returned:

```
response.setContentType( "text/html;charset=utf-8" );
writer.flush();
```

An HTTP POST request is treated identically to a GET request:

```
@Override
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
{
 this.doGet( request, response );
}
```

The servlet is configured using a standard web application deployment descriptor, provided as WEB-INF/web.xml in the WAR file content:

```
    <?xml version="1.0"?>
    <web-app xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" version="3.0">
     <distributable/>
    </web-app>
```

By designating the web application as distributable, the container is instructed to allow replication of the HTTP session data, as configured in the server profile:

```
 <distributable />
```

6.3. STATEFUL SESSION BEAN CLUSTERING

The application includes a stateful session bean. Following the latest Java EE Specification, under JBoss EAP 7, a stateful session bean can simply be included in a WAR file with no additional descriptor. The *com.redhat.refarch.eap7.cluster.sfsb* package of the application contains both the remote interface and the bean implementation class of the stateful session bean.

The interface declares a getter and setter method for its state, which is a simple alphanumeric name. It also provides a *getServer()* operation that returns the name of the EAP server being reached. This can help identify the node of the cluster that is invoked:

```
public interface StatefulSession
{
 public String getServer();
 public String getName();
 public void setName(String name);
}
```

The bean class implements the interface and declares it as it remote interface:

```
import javax.ejb.Remote;
import javax.ejb.Stateful;
import org.jboss.ejb3.annotation.Clustered;
@Stateful
@Remote(StatefulSession.class)
public class StatefulSessionBean implements StatefulSession
{

 private String name;

 @Override
 public String getServer()
 {
  return System.getProperty( "jboss.server.name" );
 }

 public String getName()
 {
  return name;
 }

 public void setName(String name)
 {
  this.name = name;
 }
}
```

The class uses annotations to designate itself as a stateful session bean:

```
@Stateful
```

Remote invocation of a clustered session bean is demonstrated in the BeanClient class, included in the clientApp JAR file and placed in the *com.redhat.refarch.eap7.cluster.sfsb.client* package. This class calls both a stateful and a stateless session bean:

```
import java.util.Hashtable;
import javax.jms.JMSException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
   import com.redhat.refarch.eap7.cluster.jpa.Person;
   import com.redhat.refarch.eap7.cluster.sfsb.StatefulSession;
```

```
    import com.redhat.refarch.eap7.cluster.sfsb.StatefulSessionBean;
    import com.redhat.refarch.eap7.cluster.slsb.StatelessSession;
    import com.redhat.refarch.eap7.cluster.slsb.StatelessSessionBean;

 public class BeanClient
 {

  private StatefulSession sfsb;
  private Context context;
  private String applicationContext;
 ...
```

The *sfsb* field is created to hold a stub reference for the stateful session bean, and context is the naming context used to look up session beans. The *applicationContext* is the root context of the deployed web application, assumed to be *clusterApp* for this reference architecture.

The following code looks up and stores a reference to the stateful session bean:

```
 Hashtable<String, String> jndiProps = new Hashtable<String, String>();
 jndiProps.put( Context.URL_PKG_PREFIXES, "org.jboss.ejb.client.naming"
);
 context = new InitialContext( jndiProps );
 String sfsbName = StatefulSessionBean.class.getSimpleName() + "!" +
StatefulSession.class.getName() + "?stateful";
 sfsb = (StatefulSession)context.lookup( "ejb:/" + applicationContext +
"//" + sfsbName );
```

Any subsequent interaction with the stateful session bean is assumed to be part of the same conversation and uses the stored stub reference:

```
 private StatefulSession getStatefulSessionBean() throws
NamingException
 {
  return sfsb;
 }
```

In contrast, every call to the stateless session bean looks up a new stub. It is assumed that calls are infrequent and are never considered part of a conversation:

```
 private StatelessSession getStatelessSessionBean() throws
NamingException
 {
  String slsbName = StatelessSessionBean.class.getSimpleName() + "!" +
StatelessSession.class.getName();
  String lookupName = "ejb:/" + applicationContext + "//" + slsbName;
  return (StatelessSession)context.lookup( lookupName );
 }
```

To look up enterprise Java beans deployed on EAP 7 through this approach, the required EJB client context is provided. This context is provided by including a property file called *jboss-ejb-client.properties* in the root of the runtime classpath. In this example, the file contains the following:

```
endpoint.name=client-endpoint
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=node1,node2,node3
```

```
remote.connection.node1.host=10.19.137.34
remote.connection.node1.port=4547
remote.connection.node1.connect.timeout=500
remote.connection.node1.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
remote.connection.node1.username=ejbcaller
remote.connection.node1.password=password1!
remote.connection.node2.host=10.19.137.35
remote.connection.node2.port=4547
remote.connection.node2.connect.timeout=500
remote.connection.node2.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
remote.connection.node2.username=ejbcaller
remote.connection.node2.password=password1!
remote.connection.node1.host=10.19.137.36
remote.connection.node1.port=4547
remote.connection.node1.connect.timeout=500
remote.connection.node1.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
remote.connection.node1.username=ejbcaller
remote.connection.node1.password=password1!

remote.clusters=ejb

remote.cluster.ejb.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
remote.cluster.ejb.username=ejbcaller
remote.cluster.ejb.password=password1!
```

The *endpoint.name* property represents the name that will be used to create the client side of the endpoint. This property is optional and if not specified in the *jboss-ejb-client.properties* file, its values defaults to *config-based-ejb-client-endpoint*. This property has no functional impact.

```
endpoint.name=client-endpoint
```

The *remote.connectionprovider.create.options.* property prefix can be used to pass the options that will be used while create the connection provider which handle the *remote:* protocol. In this example, the *remote.connectionprovider.create.options.* property prefix is used to pass the *org.xnio.Options.SSL_ENABLED* property value as *false*, as EJB communication is not taking place over SSL in this reference environment.

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
```

It is possible to configure multiple EJB receivers to handle remote calls. In this reference architecture, the EJB is deployed on all three nodes of the cluster so all three can be made available for the initial connection. The cluster is configured separately in this same file, so listing all the cluster members is not strictly required. Configuring all three nodes makes the initial connection possible, even when the first two nodes are not available:

```
remote.connections=node1,node2,node3
```

This means providing three separate configuration blocks, one for each node. For *node1*, the configuration in this reference architecture is as follows:

```
remote.connection.node1.host=10.19.137.34
remote.connection.node1.port=4547
remote.connection.node1.connect.timeout=500
remote.connection.node1.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
```

remote.connection.node1.username=ejbcaller
remote.connection.node1.password=password1!

The first two properties identify the host address and its remoting port. In this example, the remoting port has an offset of 100, since the client is targeting the passive and secondary domain. The timeout has been set to 500ms for the connection and indicates that SASL mechanisms which accept anonymous logins are not permitted.

Credentials for an application user are also provided. This user must be configured in the application realm of the EAP server.

This configuration block can be duplicated for nodes 2 and 3, where only the IP address is changed:

remote.connection.node2.host=10.19.137.35
remote.connection.node2.port=4547
…

remote.connection.node3.host=10.19.137.36
remote.connection.node3.port=4547
…

The cluster itself must also be configured. When communicating with more than one cluster, a comma-separated list can be provided.

remote.clusters=ejb

The name of the cluster must match the name of the cache container that is used to back the cluster data. By default, the Infinispan cache container is called ejb. Similar security configuration for the cluster also follows:

remote.cluster.ejb.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
remote.cluster.ejb.username=ejbcaller remote.cluster.ejb.password=password1!

6.4. DISTRIBUTED MESSAGING QUEUES

It is common practice to use an MDB to consume from a JMS queue. This application includes the *MessageDrivenBean* class in the *com.redhat.refarch.eap7.cluster.mdb* package:

```
import java.io.Serializable;
import java.util.HashMap;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.ObjectMessage;
@MessageDriven(activationConfig =
{@ActivationConfigProperty(propertyName = "destinationType",
propertyValue = "javax.jms.Queue"),
@ActivationConfigProperty(propertyName = "destination", propertyValue =
"queue/DistributedQueue"), @ActivationConfigProperty(propertyName =
"maxSession", propertyValue = "1")})
public class MessageDrivenBean implements MessageListener
{
 @Override
```

```
 public void onMessage(Message message)
 {
  try
  {
      @SuppressWarnings("unchecked")
      HashMap<String, Serializable> map = (HashMap<String,
Serializable>) message.getBody(HashMap.class);
      String text = (String)map.get( "message" );
      int count = (Integer)map.get( "count" );
      long delay = (Long)map.get( "delay" );
      System.out.println( count + "  :  " + text );
      Thread.sleep( delay );
  }
  catch( JMSException e )
  {
   e.printStackTrace();
  }
  catch( InterruptedException e )
  {
   e.printStackTrace();
  }
 }
}
```

The class is designated as an MDB through annotation:

```
 @MessageDriven
```

The annotation includes an *activationConfig* property, which describes this bean as a consumer of a queue, provides the Java Naming and Directory Interface (JNDI) name of that queue, and configures a single session for message consumption, thereby throttling and slowing down the processing of messages for the purpose of testing:

```
 activationConfig = {
   @ActivationConfigProperty(propertyName = "destinationType",
propertyValue = "javax.jms.Queue"),
   @ActivationConfigProperty(propertyName = "destination",
propertyValue = "queue/DistributedQueue"),
   @ActivationConfigProperty(propertyName = "maxSession", propertyValue
= "1")})
```

This MDB expects to read an Object Message that is in fact a Map, containing a string message, a sequence number called count, and a numeric delay value, which will cause the bean to throttle the processing of messages by the provided amount (in milliseconds):

```
  String text = (String)map.get( "message" );
  int count = (Integer)map.get( "count" );
  long delay = (Long)map.get( "delay" );
  System.out.println( count + ": " + text );
  Thread.sleep( delay );
```

6.5. JAVA PERSISTENCE API, SECOND-LEVEL CACHING

Web applications in JBoss EAP 7 can configure persistence units by simply providing a persistence.xml file in the classpath, under a META-INF folder:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
  <persistence-unit name="jpaTest">
    <jta-data-source>java:jboss/datasources/ClusterDS</jta-data-source>
    <shared-cache-mode>DISABLE_SELECTIVE</shared-cache-mode>
    <properties>
      <property name="hibernate.hbm2ddl.auto" value="update" />
      <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
      <property name="hibernate.cache.use_second_level_cache" value="true"/>
    </properties>
  </persistence-unit>
</persistence>
```

This persistence unit associates itself with a datasource configured in EAP 7 and is attached to a JNDI name of *java:jboss/datasources/ClusterDS*:

```
<jta-data-source>java:jboss/datasources/ClusterDS</jta-data-source>
```

Caching is made implicitly available for all cases, other than those where the class explicitly opts out of caching by providing an annotation:

```
<shared-cache-mode>DISABLE_SELECTIVE</shared-cache-mode>
```

This means that if JPA second-level caching is enabled and configured, as it is in this reference architecture, a JPA bean will take advantage of the cache, unless it is annotated to not be cacheable, as follows:

```
  @Cacheable(false)
```

Provider-specific configuration instructs hibernate to create SQL statements appropriate for a Postgres database, create the schema as required and enable second-level caching:

```
<property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
<property name="hibernate.hbm2ddl.auto" value="update" />
<property name="hibernate.cache.use_second_level_cache" value="true"/>
```

An Entity class called Person is created in the com.redhat.refarch.eap7.cluster.jpa package, mapping to a default table name of Person in the default persistence unit:

```
  package com.redhat.refarch.eap7.cluster.jpa;
  import java.io.Serializable;
  import javax.persistence.Entity;
  import javax.persistence.GeneratedValue;
  import javax.persistence.GenerationType;
  import javax.persistence.Id;
  @Entity
  public class Person implements Serializable
```

```
{

 private static final long serialVersionUID = 1L;

 @Id
 @GeneratedValue(strategy = GenerationType.IDENTITY)
 private Long id;

 private String name;

 public Person(String name)
 {
  this.name = name;
 }

 public String getName()
 {
  return name;
 }

 public void setName(String name)
 {
  this.name = name;
 }

 @Override
 public String toString()
 {
  return "Person [id=" + id + ", name=" + name + "]";
 }
}
```

In the absence of a *Cacheable(false)* annotation, *Person* entities will be cached in the configured second-level hibernate cache.

A Stateless Session bean called *StatelessSessionBean* is created in the *com.redhat.refarch.eap7.cluster.slsb* package to create, retrieve and modify *Person* entities. The stateless session bean also sends messages to the configured JMS Queue.

Similar to the stateful bean, the stateless session beans also uses a remote interface:

```
import java.util.List;
import javax.jms.JMSException;
import com.redhat.refarch.eap7.cluster.jpa.Person;
public interface StatelessSession
{

 public String getServer();

 public void createPerson(Person person);

 public List<Person> findPersons();

 public String getName(Long pk);

 public void replacePerson(Long pk, String name);
```

```
  public void sendMessage(String message, Integer messageCount, Long
processingDelay) throws JMSException;
 }
```

The bean implementation class is called *StatelessSessionBean* and declared in the same package as its interface:

```
import java.io.Serializable;
import java.util.HashMap;
import java.util.List;
import javax.annotation.Resource;
import javax.ejb.Remote;
import javax.ejb.Stateless;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.JMSException;
import javax.jms.MessageProducer;
import javax.jms.ObjectMessage;
import javax.jms.Queue;
import javax.jms.Session;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import org.jboss.ejb3.annotation.Clustered;

@Stateless
@Remote(StatelessSession.class)
public class StatelessSessionBean implements StatelessSession
{
   ...
```

The class uses annotations to designate itself as a stateless session bean:

```
@Stateless
```

The bean both implements its remote interface and declares it as *Remote*:

```
@Remote(StatelessSession.class)
... implements StatelessSession
```

The stateless session bean can simply inject an entity manager for the default persistence unit:

```
@PersistenceContext
private EntityManager entityManager;
```

To interact with a JMS queue, both the queue and a connection factory can also be injected:

```
@Resource(mappedName = "java:/ConnectionFactory")
private ConnectionFactory connectionFactory;

@Resource(mappedName = "java:/queue/DistributedQueue")
private Queue queue;
```

To highlight the load balancing taking place, the name of the server that has been reached may be

returned by calling an operation on the bean:

```
@Override
public String getServer()
{
 return System.getProperty( "jboss.server.name" );
}
```

Creating a new entity object is simple with the injected entity manager. With JPA configured to map to a database table which is created by hibernate on demand, a new entity results in a new row being inserted in the RDBMS.

```
@Override
public void createPerson(Person person)
{
 entityManager.persist( person );
}
```

The JPA 2 Specification provides the Criteria API for queries. The findPersons() operation of the class returns all the available entities:

```
@Override
public List<Person> findPersons()
{
 CriteriaBuilder builder = entityManager.getCriteriaBuilder();
 CriteriaQuery<Person> criteriaQuery = builder.createQuery(
Person.class );
 criteriaQuery.select( criteriaQuery.from( Person.class ) );
 List<Person> persons = entityManager.createQuery( criteriaQuery
).getResultList();
 return persons;
}
```

Each entity has an automated sequence ID and a name. To look up an entity by its sequence ID, which is the primary key of the table in the database, the find operation of the entity manager may be used:

```
@Override
public String getName(Long pk)
{
 Person entity = entityManager.find( Person.class, pk );
 if( entity == null )
 {
  return null;
 }
 else
 {
  return entity.getName();
 }
}
```

The bean also provides a method to modify an entity:

```
@Override
public void replacePerson(Long pk, String name)
```

```
 {
  Person entity = entityManager.find( Person.class, pk );
  if( entity != null )
  {
   entity.setName( name );
   entityManager.merge( entity );
  }
 }
```

Finally, the stateless bean can also be used to send a number of JMS messages to the configured queue and request that they would be processed sequentially, and throttled according to the provided delay, in milliseconds:

```
 @Override
 public void sendMessage(String message, Integer messageCount, Long
processingDelay) throws JMSException
 {
  HashMap<String, Serializable> map = new HashMap<String, Serializable>
();
  map.put( "delay", processingDelay );
  map.put( "message", message );
  Connection connection = connectionFactory.createConnection();
  try
  {
   Session session = connection.createSession( false,
Session.AUTO_ACKNOWLEDGE );
   MessageProducer messageProducer = session.createProducer( queue );
   connection.start();
   for( int index = 1; index <= messageCount; index++ )
   {
    map.put( "count", index );
    ObjectMessage objectMessage = session.createObjectMessage();
    objectMessage.setObject( map );
    messageProducer.send( objectMessage );
   }
  }
  finally
  {
   connection.close();
  }
 }
```

APPENDIX A. REVISION HISTORY

Revision	Release Date	Author(s)
1.3	Mar 2017	Babak Mozaffari
1.2	Jul 2016	Babak Mozaffari
1.1	Jun 2016	Babak Mozaffari
1.0	Jun 2016	Calvin Zhu & Babak Mozaffari

APPENDIX B. CONTRIBUTORS

We would like to thank the following individuals for their time and patience as we collaborated on this process. This document would not have been possible without their many contributions.

Contributor	Title	Contribution
Bilge Ozpeynirci	Senior Product Manager - Technical	Requirements, Technical Content Review
Michael Musgrove	Senior Software Engineer	Technical Content Review
Tom Ross	Senior Software Maintenance Engineer	Technical Content Review
Scott Marlow	Principal Software Engineer	Technical Content Review
Miroslav Novak	Quality Engineer	Messaging Configuration, Technical Content Review
Justin Bertram	Senior Software Engineer	Technical Content Review
Jan Stefl	Associate Manager, Quality Engineering	Technical Content Review
Ondrej Chaloupka	Quality Engineer	Technical Content Review
Wolf Fink	Senior Software Maintenance Engineer	Technical Content Review

APPENDIX C. FIREWALLD CONFIGURATION

An ideal firewall configuration constraints open ports to the required services based on respective clients. This reference environment includes a set of ports for the active cluster along with another set used by the passive cluster, which has an offset of 100 over the original set. Other than the TCP ports accessed by callers, there are also a number of UDP ports that are used within the cluster itself for replication, failure detection and other HA functions. The following firewalld configuration opens the ports for known JBoss services within the set of IP addresses used in the reference environment, while also allowing UDP communication between them on any multicast address. This table shows the ports for the active domain. The passive domain would include an offset of 100 over many of these ports and different usage and configuration of components may lead to alternate firewall requirements.

Below is the node1, node2 and node3 firewalld configuration:

```
firewall-cmd --permanent --zone=public --list-all
public (default)
  interfaces:
  sources:
  services: dhcpv6-client ssh
  ports: 45700/udp 55200/udp 45688/udp 23364/udp
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:
    rule family="ipv4" source address="10.19.137.34" port port="3528" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.35" port port="3528" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.36" port port="3528" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.34" port port="3529" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.35" port port="3529" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.36" port port="3529" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.34" port port="4712" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.35" port port="4712" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.36" port port="4712" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.34" port port="4713" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.35" port port="4713" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.36" port port="4713" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.34" port port="5545" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.35" port port="5545" protocol="tcp" accept

    rule family="ipv4" source address="10.19.137.36" port port="5545" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.34" port port="5546" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.35" port port="5546" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.36" port port="5546" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.34" port port="7600" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.35" port port="7600" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.36" port port="7600" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.37" port port="8009" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.34" port port="8080" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.35" port port="8080" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.36" port port="8080" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.35" port port="9999" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.36" port port="9999" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.34" port port="54200" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.36" port port="54200" protocol="tcp" accept
```

```
        rule family="ipv4" source address="10.19.137.35" port port="54200" protocol="tcp" accept
        rule family="ipv4" source address="10.19.137.34" port port="55200" protocol="tcp" accept
        rule family="ipv4" source address="10.19.137.35" port port="55200" protocol="tcp" accept
        rule family="ipv4" source address="10.19.137.36" port port="55200" protocol="tcp" accept
        rule family="ipv4" source address="10.19.137.34" port port="57600" protocol="tcp" accept
        rule family="ipv4" source address="10.19.137.35" port port="57600" protocol="tcp" accept
        rule family="ipv4" source address="10.19.137.36" port port="57600" protocol="tcp" accept
```

Red Hat JBoss Core Services Apache HTTP Server 2.4 has the following firewalld configuration:

```
firewall-cmd --permanent --zone=public --list-all
public (default)
  interfaces:
  sources:
  services: dhcpv6-client ssh
  ports: 443/tcp
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:
    rule family="ipv4" source address="10.19.137.34" port port="5432" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.35" port port="5432" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.36" port port="5432" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.34" port port="6661" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.35" port port="6661" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.36" port port="6661" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.34" port port="6662" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.35" port port="6662" protocol="tcp" accept
    rule family="ipv4" source address="10.19.137.36" port port="6662" protocol="tcp" accept
    rule family="ipv4" source address="10.10.0.1/8" port port="81" protocol="tcp" accept
    rule family="ipv4" source address="10.10.0.1/8" port port="82" protocol="tcp" accept
```

APPENDIX D. REVISION HISTORY

Revision 1.3-0	**March 2017**	**CZ**