

## Lab 9. Clustering in Tomcat 8



I would like to start this topic with a story. There were two teams; A and B, in an IT organization, managing different systems. Both teams consisted of highly qualified experts in middleware. One day, the CEO of that organization called a meeting for both teams, stating that they had to manage two different middleware environments, one middleware environment was individually assigned to teams A and B. Each team had to follow their own approaches to fix the environmental issues. After 3 months, each client performed a process review and the results surprised the higher management. Team A had maintained 50 percent uptime for the application and Team B had maintained 99 percent uptime for the application hosted in their environment. While comparing the approaches followed by each team, it was found that Team B had followed a high availability architecture using clustering while Team A had followed a single-server architecture.

In mid-year, the management announced the financial appraisal; Team A received no appraisal and, on the contrary, Team B members received high bonuses with promotions. Today, it's your chance to decide whether you want to join Team A or B. If you want to join Team B and follow a high availability architecture, read this lab carefully.

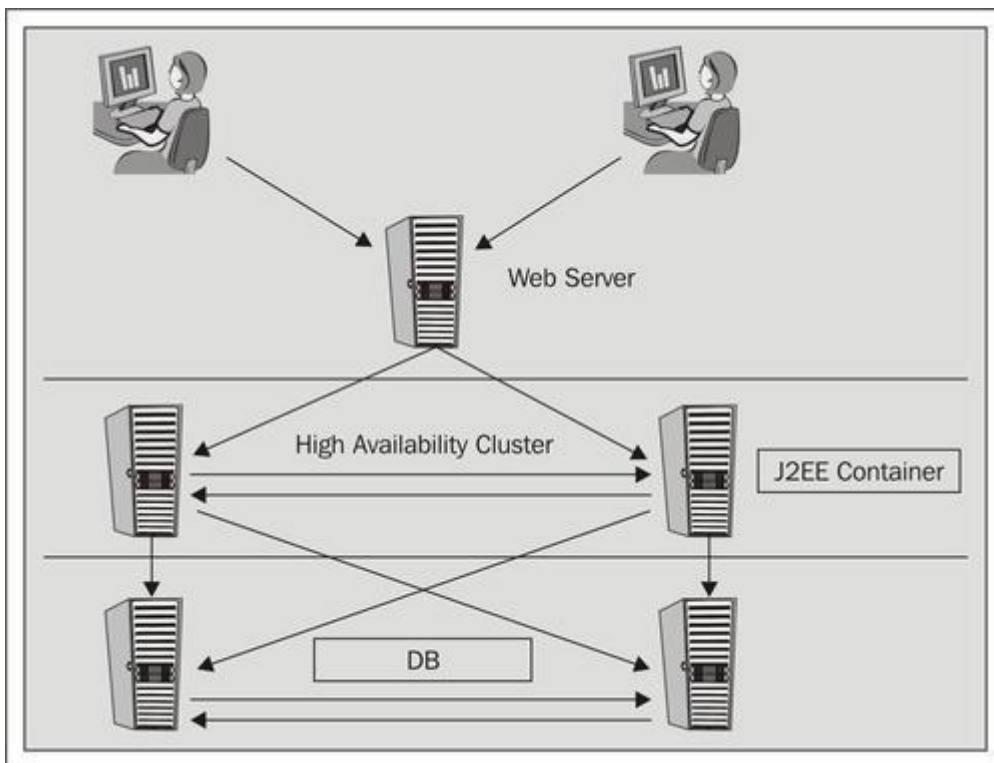
In this lab, we discuss:

- High availability architecture and its advantages
- Different types of high availability architectures including load balancing and clustering
- Approaches used by IT industries while building a high availability architecture in an enterprise setup
- How to do Apache Tomcat clustering
- Various clustering architectures
- How to solve common problems in clustering

### What is a cluster?

The cluster is a group of servers or computers connected together, which may perform similar functionality in the system. These systems are normally connected to each other through high speed Ethernet. Cluster systems are used where quick processing or system availability is required. Some of the examples where clusters are highly used are the financial sector, banking, security areas, and so on. The following figure shows the J2EE containers clustered in the environment:

---



## Note

Cluster topology varies from environment to environment, based on the application requirements.

## Benefits of clustering

There are many advantages of clustering in a middleware environment. It also depends on which cluster techniques we are using. We will discuss the various advantages of clustering:

- **Scalability:** It gives freedom to the system architect to accommodate future enhancements for applications and servers. Suppose the web application currently has 100 concurrent users, and at the time of an event, you are expecting 500 concurrent users. What would you do to make sure that the system runs as expected? Clustering is one of the best solutions.
- **High availability:** High availability systems are implemented in environments that need to be up 99.99 percent, such as banking, financial sectors, and so on where entire transactions need to be recorded on the systems. They cannot afford to have their websites down for a minute. Hence, they implement a high availability system to make sure the system will not be down at any time.
- **High performance:** One of the major advantages of clustering is that it boosts up the system performance by  $n$  times where,  $n$  = number of systems. For example, if you are running the system with a single server and the system supports 100 concurrent users, then just by adding another server system you can support 200 concurrent users. Also, if you want to decrease the response time for the application, you can use the JVM performance tuning.
- **Cloud computing:** Clustering is also very useful in a cloud computing environment. It is used while setting the grid computing architecture for cloud computing to improve performance.

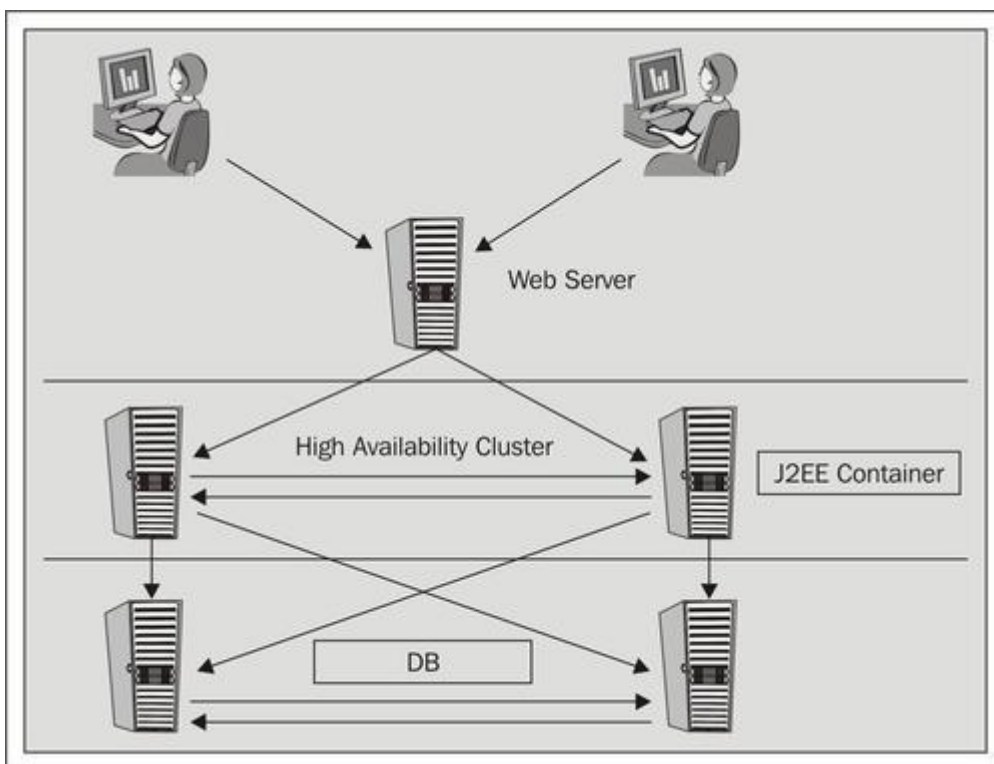
## Disadvantages of clustering

Until now, we have discussed how useful clustering is for a web environment. Let's discuss the disadvantages of clustering:

- **Cost:** It plays a major role in the implementation of a new environment. If we want to setup web clustering, then we need more servers. This, again, increases the cost of the project.
- **Monitoring:** With an increase in the number of servers, the monitoring of servers will also increase, making it difficult for the web administrator to manage the servers.

## What is a cluster?

The cluster is a group of servers or computers connected together, which may perform similar functionality in the system. These systems are normally connected to each other through high speed Ethernet. Cluster systems are used where quick processing or system availability is required. Some of the examples where clusters are highly used are the financial sector, banking, security areas, and so on. The following figure shows the J2EE containers clustered in the environment:



### Note

Cluster topology varies from environment to environment, based on the application requirements.

### Benefits of clustering

There are many advantages of clustering in a middleware environment. It also depends on which cluster techniques we are using. We will discuss the various advantages of clustering:

- **Scalability:** It gives freedom to the system architect to accommodate future enhancements for applications and servers. Suppose the web application currently has 100 concurrent users, and at the time of an event, you are expecting 500 concurrent users. What would you do to make sure that the system runs as expected? Clustering is one of the best solutions.
- **High availability:** High availability systems are implemented in environments that need to be up 99.99 percent, such as banking, financial sectors, and so on where entire transactions need to be recorded on the

systems. They cannot afford to have their websites down for a minute. Hence, they implement a high availability system to make sure the system will not be down at any time.

- **High performance:** One of the major advantages of clustering is that it boosts up the system performance by  $n$  times where,  $n$  = number of systems. For example, if you are running the system with a single server and the system supports 100 concurrent users, then just by adding another server system you can support 200 concurrent users. Also, if you want to decrease the response time for the application, you can use the JVM performance tuning.
- **Cloud computing:** Clustering is also very useful in a cloud computing environment. It is used while setting the grid computing architecture for cloud computing to improve performance.

## Disadvantages of clustering

Until now, we have discussed how useful clustering is for a web environment. Let's discuss the disadvantages of clustering:

- **Cost:** It plays a major role in the implementation of a new environment. If we want to setup web clustering, then we need more servers. This, again, increases the cost of the project.
- **Monitoring:** With an increase in the number of servers, the monitoring of servers will also increase, making it difficult for the web administrator to manage the servers.

## Clustering architecture

In this topic, we will discuss the various architectures of clustering used by IT industries. These architectures may vary on each implementation, depending on the application and business requirements. There are basically two types of clustering architectures implemented in a real-time IT infrastructure:

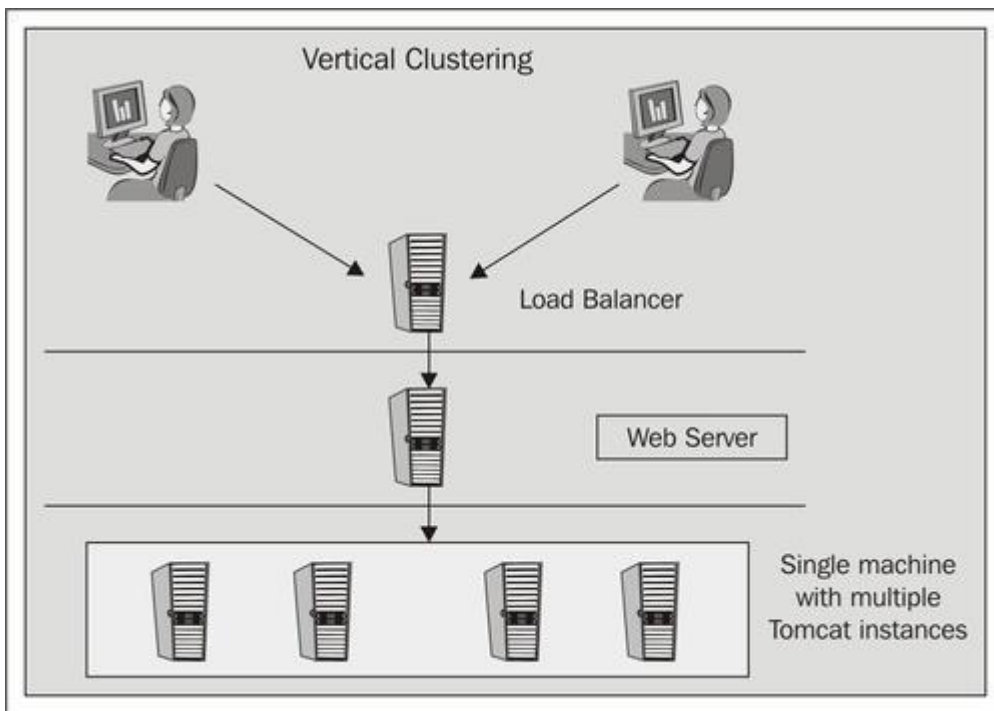
- Vertical clustering
- Horizontal clustering

By default, Apache Tomcat 8 supports both horizontal and vertical clustering. In the next section, we will discuss the implementation of both types of clustering in Apache Tomcat 8. Before that, let's discuss clustering architectures, where they can be implemented, and their advantages.

### Vertical clustering

Vertical clustering consists of a single hardware with multiple instances running, using shared resources from the system. This kind of setup is mainly done in development and quality systems for the developer to test the functionality of the application. Also, vertical clustering can be implemented in production in certain cases, where there is a resource crunch for the hardware. It uses the concept of a shared resource such as CPU, RAM, and so on. The following figure shows the pictorial presentation of vertical clustering:

---



Every architecture has its pros and cons. Let's discuss some of the pros and cons of vertical clustering.

#### **Advantages of vertical clustering**

Following are the advantages of vertical clustering:

- No network bandwidth issue, as instances are hosted on a single machine
- Hardware is shared by different Tomcat instances
- Addition of physical hardware is not required
- Single JVM can be shared by multiple instances

#### **Disadvantages of vertical clustering**

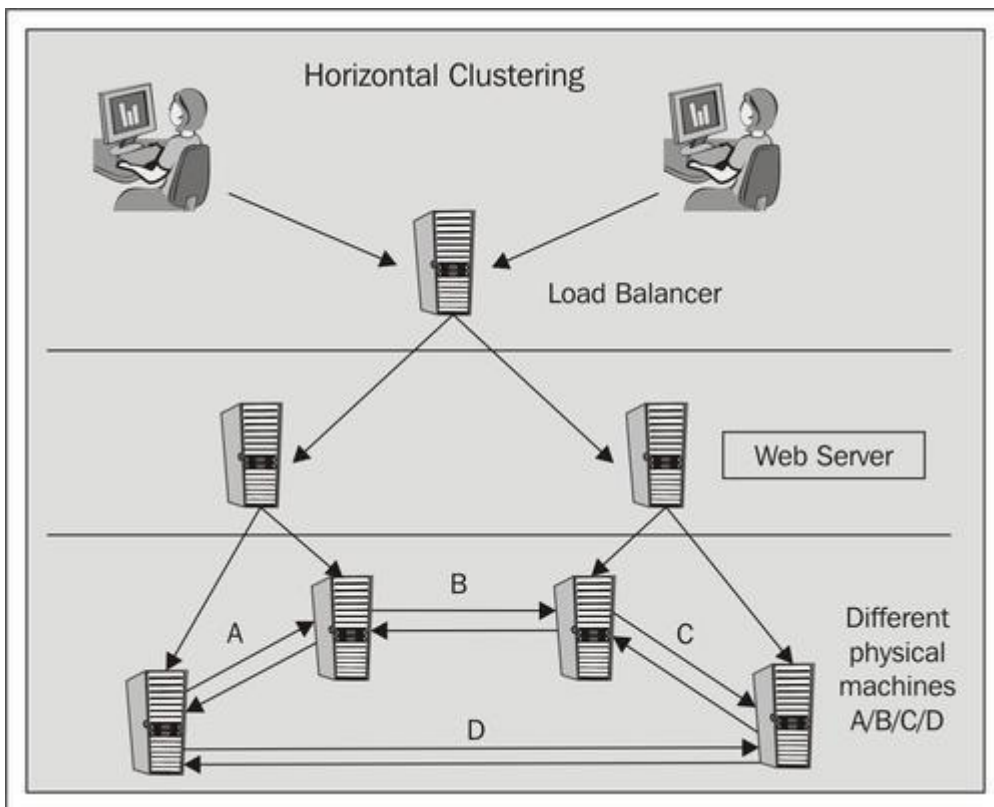
Following are the disadvantages of vertical clustering:

- No failover in case of hardware issues
- More maintenance issues
- High-end hardware used for implementation
- High cost

#### **Horizontal clustering**

In this type of clustering method, instances are configured separately on each physical machine and connected through high speed Ethernet. It's a very popular implementation technique in the production environment.

Resources of one machine are not shared with the other machine. Also, failover can be done in the case of hardware failure. The following figure shows the horizontal clustering for different Apache Tomcat instances using separate physical hardware:



Let's discuss some of the pros and cons of horizontal clustering:

#### Advantages of horizontal clustering

Following are the advantages of horizontal clustering:

- Failover is possible in the case of hardware failure
- A low-end system can be used, as a single instance runs for each physical or VM instance
- Low maintenance issues

#### Disadvantages of horizontal clustering

Following are the disadvantages of horizontal clustering:

- Network bandwidth issues
- Network connectivity issues between machines
- Each instance requires a separate physical hardware component

#### Note

Horizontal clustering is the most preferred method in a production environment.

## Vertical clustering in Apache Tomcat 8

In the previous topics, we have discussed the different types of cluster architecture, supported by Apache Tomcat 8. It's time to take a real-time challenge to implement clustering. Let's start with vertical clustering.

For vertical clustering, we have to configure at least two instances of Apache Tomcat and the complete process consists of three stages. Let's discuss and implement the steps for vertical cluster in Tomcat 7:

1. Installation of the Tomcat instance.
2. Configuration of the cluster.
3. Apache HTTP web server configuration for the vertical cluster.

## Installation of the Tomcat instance

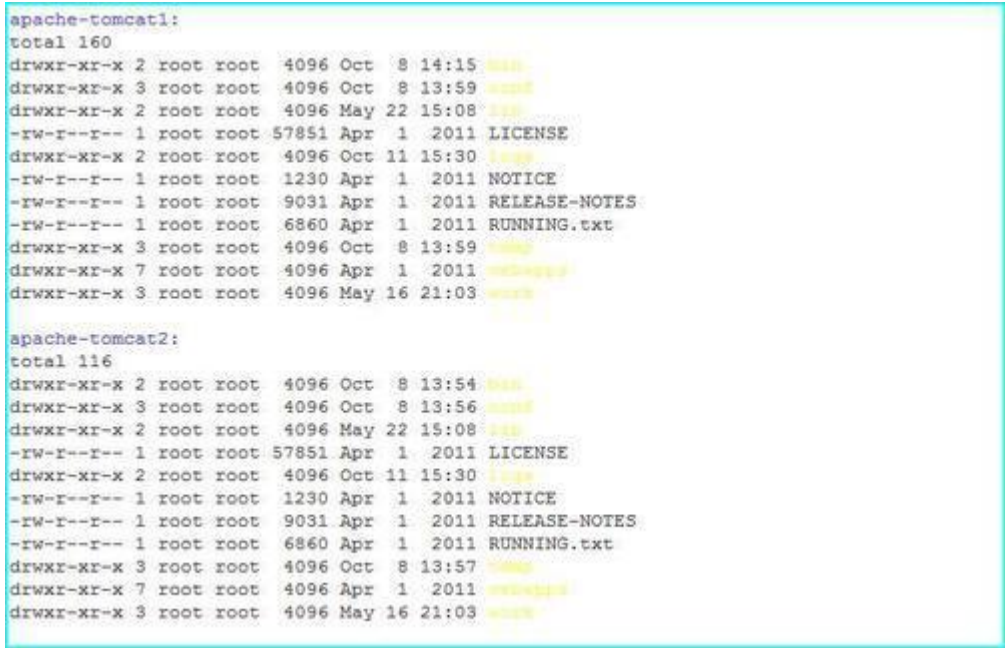
The installation of Apache Tomcat 8 can be done in three easy steps:

1. Download and unzip the software in the required directory.
2. Install the JDK and set the JAVA\_HOME.
3. Copy the Apache Tomcat source code in two different directories, for example, /opt/tomcatX (where X= instance number) and verify that the files are properly copied on both instances, using the following command:

```
[root@localhost opt]# ls -l apache-tomcat*
```

- The following screenshot shows the output of the previous

command:



```
apache-tomcat1:
total 160
drwxr-xr-x 2 root root 4096 Oct  8 14:15 bin
drwxr-xr-x 3 root root 4096 Oct  8 13:59 conf
drwxr-xr-x 2 root root 4096 May 22 15:08 lib
-rw-r--r-- 1 root root 57851 Apr  1 2011 LICENSE
drwxr-xr-x 2 root root 4096 Oct 11 15:30 logs
-rw-r--r-- 1 root root 1230 Apr  1 2011 NOTICE
-rw-r--r-- 1 root root 9031 Apr  1 2011 RELEASE-NOTES
-rw-r--r-- 1 root root 6860 Apr  1 2011 RUNNING.txt
drwxr-xr-x 3 root root 4096 Oct  8 13:59 webapp
drwxr-xr-x 7 root root 4096 Apr  1 2011 webapps
drwxr-xr-x 3 root root 4096 May 16 21:03 work

apache-tomcat2:
total 116
drwxr-xr-x 2 root root 4096 Oct  8 13:54 bin
drwxr-xr-x 3 root root 4096 Oct  8 13:56 conf
drwxr-xr-x 2 root root 4096 May 22 15:08 lib
-rw-r--r-- 1 root root 57851 Apr  1 2011 LICENSE
drwxr-xr-x 2 root root 4096 Oct 11 15:30 logs
-rw-r--r-- 1 root root 1230 Apr  1 2011 NOTICE
-rw-r--r-- 1 root root 9031 Apr  1 2011 RELEASE-NOTES
-rw-r--r-- 1 root root 6860 Apr  1 2011 RUNNING.txt
drwxr-xr-x 3 root root 4096 Oct  8 13:57 webapp
drwxr-xr-x 7 root root 4096 Apr  1 2011 webapps
drwxr-xr-x 3 root root 4096 May 16 21:03 work
```

## Configuration of a vertical cluster

This section is the most critical section for vertical clustering, as all the configurations are done in this section and a simple error can make the cluster non-functioning. So, be careful while carrying out the configuration. Let's do the step-by-step configuration on each node.

### Configuration of instance 1

For the first instance; node 1, we can use the default configuration, such as Connector, AJP, or shutdown port in `server.xml`. Let's discuss each component where the configuration needs to be done and why it is used:

1. Shutdown port: The following screenshot shows the configuration for the shutdown port for the Tomcat instance. While running multiple instances, if by any chance, you have skipped configuring the shutdown port, then the Tomcat instance will be unable to start it.

```
-->
<!-- Server port="8006" shutdown="SHUTDOWN">
<!-- Security listener. Documentation at /docs/config/listeners.html
<Listener className="org.apache.catalina.security.SecurityListener" />
-->
<!-- APR library loader. Documentation at /docs/apr.html -->
```

2. Connector port: The following screenshot shows the Connector port configuration for Tomcat 8. This port is used to access the Tomcat instance, for example, normally we access the Tomcat instance by using <http://localhost:8080>, the 8080 port is called the Connector port. While running multiple instances, if you have skipped configuring this port, then the Tomcat instance will be unable to start it and you will get the Port already in use exception.

```
<!-- A "Connector" represents an endpoint by which requests are received
and responses are returned. Documentation at :
Java HTTP Connector: /docs/config/http.html (blocking & non-blocking)
Java AJP Connector: /docs/config/ajp.html
APR (HTTP/AJP) Connector: /docs/apr.html
Define a non-SSL HTTP/1.1 Connector on port 8080
-->
<Connector port="8080" protocol="HTTP/1.1"
    connectionTimeout="20000"
    redirectPort="8443" />
<!-- A "Connector" using the shared thread pool-->
<!--
```

3. AJP port: The following screenshot shows the AJP port configuration for Tomcat 8. This port is used for AJP communication between the Apache HTTP server and the Tomcat instance. While running multiple instances, if you have skipped configuring port, then the Tomcat instance will be unable to start it and you will get the Port already in use exception.

```
<!-- Define an AJP 1.3 Connector on port 8009 -->
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

4. Cluster attributes: Enable the cluster attributes for clustering in `server.xml` and the following screenshot shows the cluster class used for clustering:

```
<!--For clustering, please take a look at documentation at:
/docs/cluster-howto.html (simple how to)
/docs/config/cluster.html (reference documentation) -->
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
```



5. Configuration test: Run the configtest.sh script from TOMCAT\_HOME/bin to check the configuration. The following screenshot shows the output for the following config.sh command:

```
[root@localhost bin]# ./configtest.sh
```

A terminal window showing the output of the configtest.sh script. The output displays various configuration variables like CATALINA\_BASE, CATALINA\_HOME, CATALINA\_TMPDIR, JRE\_HOME, and CLASSPATH. It also shows the initialization of the Catalina server, including the APR-based native library and the Coyote protocol handler. The final message indicates that the initialization process was successful in 8145 ms.

```
[root@localhost bin]# ./configtest.sh
Using CATALINA_BASE:   /opt/apache-tomcat1
Using CATALINA_HOME:   /opt/apache-tomcat1
Using CATALINA_TMPDIR: /opt/apache-tomcat1/temp
Using JRE_HOME:        /opt/jdk1.6.0_24
Using CLASSPATH:       /opt/apache-tomcat1/bin/bootstrap.jar:/opt/apache-tomcat1/bin/tomcat-juli.jar
Oct 11, 2011 4:55:58 PM org.apache.catalina.core.AprLifecycleListener init
INFO: The APR based Apache Tomcat Native library which allows optimal performance in production environments was not found on the java.library.path: /opt/jdk1.6.0_24/jre/lib/i386/server:/opt/jdk1.6.0_24/jre/lib/i386:/opt/jdk1.6.0_24/jre/../lib/i386/user/java/patches/lib/i386:/lib:/usr/lib
Oct 11, 2011 4:56:01 PM org.apache.coyote.AbstractProtocolHandler init
INFO: Initializing ProtocolHandler ["http-bio-8009"]
Oct 11, 2011 4:56:01 PM org.apache.coyote.AbstractProtocolHandler init
INFO: Initializing ProtocolHandler ["ajp-bio-8009"]
Oct 11, 2011 4:56:01 PM org.apache.catalina.startup.Catalina load
INFO: Initialization process completed in 8145 ms
```

6. Tomcat instance startup: Start the instance 1 configuration using the script startup.sh. The following screenshot shows the output for the following startup.sh script:

```
[root@localhost bin]# ./startup.sh
```

A terminal window showing the output of the startup.sh script. The output displays the same configuration variables as the configtest.sh script, including CATALINA\_BASE, CATALINA\_HOME, CATALINA\_TMPDIR, JRE\_HOME, and CLASSPATH. The script then proceeds to start the Catalina server.

```
[root@localhost bin]# ./startup.sh
Using CATALINA_BASE:   /opt/apache-tomcat1
Using CATALINA_HOME:   /opt/apache-tomcat1
Using CATALINA_TMPDIR: /opt/apache-tomcat1/temp
Using JRE_HOME:        /opt/jdk1.6.0_24
Using CLASSPATH:       /opt/apache-tomcat1/bin/bootstrap.jar:/opt/apache-tomcat1/bin/tomcat-juli.jar
```

- Check the Tomcat instance process using the following mentioned

command. The following screenshot shows the output for the `ps` command:

```
...
[root@localhost bin]# ps -ef |grep java
...

```

A terminal window showing the output of the ps -ef |grep java command. The output lists the Java process for the Tomcat instance, showing its PID (11766), PPID (1), and command line. The command line includes the Java executable, classpath, and various JVM options. A second line shows the grep process (PID 11902) that executed the command.

```
[root@localhost bin]# ps -ef |grep java
root      11766      1 11 17:00 pts/3    00:00:17 /opt/jdk1.6.0_24/bin/java -Djava
m -Xmx512m -XX:MaxPermSize=256m -Dorg.jboss.resolver.warning=true -Dsun.rmi.dgc.
.logging.manager=org.apache.juli.ClassLoaderLogManager -Djava.awt.headless=true
e.port=7091 -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.manageme
sspath /opt/apache-tomcat1/bin/bootstrap.jar:/opt/apache-tomcat1/bin/tomcat-juli
-Djava.io.tmpdir=/opt/apache-tomcat1/temp org.apache.catalina.startup.Bootstrap
root      11902 10149  0 17:02 pts/3    00:00:00 grep java
[root@localhost bin]#
```

## Configuration of instance 2

We cannot use the default configuration on node 2. There will be port conflicts as we are running the instance with single IP on the same physical machine. Let's configure instance 2 step-by-step:

1. Change the shutdown port for instance 2 in server.xml (increment it by 1). The following screenshot shows the configuration:

---

```
-->
<Server port="8007" shutdown="SHUTDOWN">
  <!-- Security listener. Documentation at /docs/config/listeners.html
  <Listener className="org.apache.catalina.security.SecurityListener" />
```

2. Change the Connector and redirect the port for instance 2 in server.xml (increment it by 1). The following screenshot shows the configuration:

---

```
-->
<Connector port="8081" protocol="HTTP/1.1"
  connectionTimeout="20000"
  redirectPort="8444" />
<!-- A "Connector" using the shared thread pool-->
```

3. Change the AJP and redirect the port for instance 2 in server.xml (increment it by 1). The following screenshot shows the configuration:

---

```
<!-- Define an AJP 1.3 Connector on port 8009 -->
<Connector port="8010" protocol="AJP/1.3" redirectPort="8444" />
```

4. Enable the cluster attributes for clustering in server.xml. The following screenshot shows the configuration:

---

```
<!--For clustering, please take a look at documentation at:
  /docs/cluster-howto.html (simple how to)
  /docs/config/cluster.html (reference documentation) -->
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
```

5. Save server.xml.
6. Run the configtest.sh script from TOMCAT\_HOME/bin to check the configuration. The following screenshot shows the output for the following startup.sh script:

```
[root@localhost bin]# ./configtest.sh
```

---

```
[root@localhost bin]# ./configtest.sh
Using CATALINA_BASE:   /opt/apache-tomcat2
Using CATALINA_HOME:   /opt/apache-tomcat2
Using CATALINA_TMPDIR: /opt/apache-tomcat2/temp
Using JRE_HOME:        /opt/jdk1.6.0_24
Using CLASSPATH:        /opt/apache-tomcat2/bin/bootstrap.jar:/opt/apache-tomcat2
Oct 11, 2011 5:21:07 PM org.apache.catalina.core.AprLifecycleListener init
INFO: The APR based Apache Tomcat Native library which allows optimal performance
1.6.0_24/jre/lib/1386/server:/opt/jdk1.6.0_24/jre/lib/1386:/opt/jdk1.6.0_24/jre/
Oct 11, 2011 5:21:13 PM org.apache.coyote.AbstractProtocolHandler init
INFO: Initializing ProtocolHandler ["http-bio-8081"]
Oct 11, 2011 5:21:14 PM org.apache.coyote.AbstractProtocolHandler init
INFO: Initializing ProtocolHandler ["ajp-bio-8010"]
Oct 11, 2011 5:21:14 PM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 14385 ms
```

7. Start the configuration of instance 2 using the script startup.sh. The following screenshot shows the output for the following startup.sh script:

```
[root@localhost bin]# ./startup.sh
```

```
[root@localhost bin]# ./startup.sh
Using CATALINA_BASE:   /opt/apache-tomcat2
Using CATALINA_HOME:   /opt/apache-tomcat2
Using CATALINA_TMPDIR: /opt/apache-tomcat2/temp
Using JRE_HOME:        /opt/jdk1.6.0_24
Using CLASSPATH:        /opt/apache-tomcat2/bin/bootstrap.jar:/opt/apache-tomcat2
/bin/tomcat-juli.jar
[root@localhost bin]#
```

8. Check the Tomcat instance process. The following screenshot shows the output for the ps command:

```
[root@localhost bin]# ps -ef |grep java
```

```
[root@localhost bin]# ps -ef |grep java
root      11744      1   3 17:50 pts/3    00:00:25 /opt/jdk1.6.0_24/bin/java -Djava.util.logging.config.file=/opt/apache-tomcat1/conf/logging.properties -Xmx128m -Xms128m -XX:MaxPermSize=256m -Dorg.apache.resolver.warning=true -Dsun.rmi.dgc.client.gcInterval=3600000 -Dsun.rmi.dgc.server.gcInterval=3600000 -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Djava.awt.headless=true -Xmx128m -server -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=7991 -Dcom.sun.management.jmxremote.ssl.enabled=false -Dcom.sun.management.jmxremote.ssl=false -Djava.endorsed.dirs=/opt/apache-tomcat1/endorsed -Djava.class.path=/opt/apache-tomcat1/bin/bootstrap.jar:/opt/apache-tomcat1/bin/tomcat-juli.jar -Dcatalina.base=/opt/apache-tomcat1 -Dcatalina.home=/opt/apache-tomcat1 -Djava.io.tmpdir=/opt/apache-tomcat1/temp org.apache.catalina.startup.Bootstrap start
root      12135      1   3 17:23 pts/3    00:00:17 /opt/jdk1.6.0_24/bin/java -Djava.util.logging.config.file=/opt/apache-tomcat2/conf/logging.properties -Xmx128m -Xms128m -XX:MaxPermSize=256m -Dorg.apache.resolver.warning=true -Dsun.rmi.dgc.client.gcInterval=3600000 -Dsun.rmi.dgc.server.gcInterval=3600000 -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -Djava.awt.headless=true -Xmx128m -server -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=7992 -Dcom.sun.management.jmxremote.ssl.enabled=false -Dcom.sun.management.jmxremote.ssl=false -Djava.endorsed.dirs=/opt/apache-tomcat2/endorsed -Djava.class.path=/opt/apache-tomcat2/bin/bootstrap.jar:/opt/apache-tomcat2/bin/tomcat-juli.jar -Dcatalina.base=/opt/apache-tomcat2 -Dcatalina.home=/opt/apache-tomcat2 -Djava.io.tmpdir=/opt/apache-tomcat2/temp org.apache.catalina.startup.Bootstrap start
root      12594 10149   0 17:24 pts/3    00:00:00 grep java
```

9. Now, check catalina.out for both the nodes.

The logs for node 1 are similar to the following:

```
Oct 11, 2011 5:00:24 PM org.apache.catalina.ha.tcp.SimpleTcpCluster
startInternal
INFO: Cluster is about to start
Oct 11, 2011 5:00:24 PM org.apache.catalina.tribes.transport.ReceiverBase bind
INFO: Receiver Server Socket bound to:/127.0.0.1:4000
Oct 11, 2011 5:00:24 PM org.apache.catalina.tribes.membership.McastServiceImpl
setupSocket
```

```

# Instance node 1 started on port 4000
INFO: Setting cluster mcast soTimeout to 500
Oct 11, 2011 5:00:24 PM
INFO: Sleeping for 1000 milliseconds to establish cluster membership, start
level:8
Oct 11, 2011 5:00:26 PM org.apache.catalina.tribes.membership.McastServiceImpl
waitForMembers
# waiting for other member to join the cluster
org.apache.catalina.ha.session.JvmRouteBinderValve startInternal
INFO: JvmRouteBinderValve started
Oct 11, 2011 5:00:37 PM org.apache.coyote.AbstractProtocolHandler start
INFO: Starting ProtocolHandler ["http-bio-8080"]
Oct 11, 2011 5:00:37 PM org.apache.coyote.AbstractProtocolHandler start
INFO: Starting ProtocolHandler ["ajp-bio-8009"]
Oct 11, 2011 5:00:37 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 13807 ms
Oct 11, 2011 5:23:42 PM org.apache.catalina.tribes.io.BufferPool getBufferPool
INFO: Created a buffer pool with max size:104857600 bytes of
type:org.apache.catalina.tribes.io.BufferPool15Impl
Oct 11, 2011 5:23:43 PM org.apache.catalina.ha.tcp.SimpleTcpCluster memberAdded
INFO: Replication member added:org.apache.catalina.tribes.membership.MemberImpl
[tcp://{127, 0, 0, 1}:4001,{127, 0, 0, 1},4001, alive=1043, securePort=-1, UDP
Port=-1, id={33 91 -59 78 -34 -52 73 -9 -99 124 -53 34 69 21 -40 -82 },
payload={}, command={}, domain={}, ]
#Instance 2 joined the cluster node.

```

The logs for node 2 are similar to the following:

```

...
INFO: Starting Servlet Engine: Apache Tomcat/8.5.61
Oct 11, 2011 5:23:41 PM org.apache.catalina.ha.tcp.SimpleTcpCluster startInternal
INFO: Cluster is about to start
Oct 11, 2011 5:23:42 PM org.apache.catalina.tribes.transport.ReceiverBase bind
INFO: Receiver Server Socket bound to:/127.0.0.1:4001
Oct 11, 2011 5:23:42 PM org.apache.catalina.tribes.membership.McastServiceImpl
setupSocket
# Instance node 2 started on port 4001
INFO: Setting cluster mcast soTimeout to 500
Oct 11, 2011 5:23:42 PM org.apache.catalina.tribes.membership.McastServiceImpl
waitForMembers
INFO: Sleeping for 1000 milliseconds to establish cluster membership, start level:4
Oct 11, 2011 5:23:43 PM org.apache.catalina.ha.tcp.SimpleTcpCluster memberAdded
INFO: Replication member added:org.apache.catalina.tribes.membership.MemberImpl
[tcp://{127, 0, 0, 1}:4000,{127, 0, 0, 1},4000, alive=1398024, securePort=-1, UDP
Port=-1, id={28 42 60 -68 -99 126 64 -35 -118 -97 7 84 26 20 90 24 }, payload={},
command={}, domain={}, ]
# Instance 1 joined the cluster node 2.
...

```

## Apache web server configuration for vertical clustering

Until now, we have done the Tomcat-level configuration to configure vertical clustering in the Tomcat instance. It's time to integrate the Apache web server to Tomcat 8. Let's enable the integration by performing the following steps:

1. We have to create a new file called `mod_jk.conf` in the `conf` directory of `APACHE_HOME/conf` using the following commands:

```
[root@localhost apache-2.0]# cd /opt/apache-2.2.19/conf
vi mod-jk.conf
```

- The contents of `mod_jk` include the following lines

```
of code:
```

```
...
LoadModulejk_module modules/mod_jk.so
JkWorkersFile conf/workers.properties
JkLogFile logs/mod_jk.log
JkLogLevel info
JkMount /sample/* loadbalancer
JkMount /* loadbalancer
...
```

2. Create a new file named as `workers.properties` in `conf` using the following command:

```
[root@localhost conf]# vi workers.properties
```

- `worker.list` lists all the nodes in Tomcat through

```
which Apache communicates using the AJP protocol. In our
example, it has two nodes, as shown in the following line of
code:
```

```
...
worker.list=tomcatnode1, tomcatnode2, loadbalancer
...
```

- Define the `worker.list` for the entire nodes in the

```
cluster:
```

```
...
worker.tomcatnode1.port=8009
worker.tomcatnode1.host=localhost
worker.tomcatnode1.type=ajp13
worker.tomcatnode1.lbfactor=1
...
```

- The previous lines of code define the `tomcatnode1`

```
properties. The highlighted code shows the AJP port and hostname
of `tomcatnode1`, which is essential for vertical
clustering:
```

```
...
worker.tomcatnode2.port=8010
worker.tomcatnode2.host=localhost
worker.tomcatnode2.type=ajp13
worker.tomcatnode2.lbfactor=1
...
```

- The previous lines of code define the `tomcatnode2`

```
properties. The highlighted code shows the AJP port and hostname
of `tomcatnode2`. This is essential for vertical
clustering.
```

```
...
worker.loadbalancer.type=lb
worker.loadbalancer.balanced_workers=tomcatnode1, tomcatnode2
worker.loadbalancer.sticky_session=1
...
```

- The previous lines of code define the load balancing properties

```
for `mod_jk`.
```

3. The last step is to include the `mod_jk.conf` in the main configuration file of `httpd`, that is `httpd.conf` and reload the Apache services:

```
[root@localhostconf]# vi httpd.conf
```

#### Note

```
Include ` conf/mod_jk.conf`.

in the end of the ` httpd.conf`.
```

## Horizontal clustering in Apache Tomcat 8

For horizontal clustering, we have to configure at least two instances of Apache Tomcat on two different physical or virtual systems. These physical machines can be on the same physical network. It also helps in providing a high-speed bandwidth to the system.

#### Note

If you want to configure clustering on different networks, then you have to open the firewall between the two networks for the AJP port and the clustering port.

There are prerequisites for configuring horizontal clustering. The following are the details:

- Time sync between the two servers
- Proper network connectivity between the two servers
- Firewall ports between the two servers (if you are connecting from a different network)

In order to configure horizontal clustering, you have to perform the following steps:

1. Installation of the Tomcat instance
2. Configuration of the cluster
3. Apache HTTP web server configuration for the horizontal cluster

## Installation of the Tomcat instance

We have already discussed the installation of Tomcat in the previous section. The steps for installation will remain the same here and, hence, we are skipping the installation. Let's move to step 2.

## Configuration of the cluster

This section is the most critical section for horizontal clustering, as all the configurations are done in this section and a simple error can make the cluster not work. So, be careful before carrying out the configuration. Let's do the step-by-step configuration on each node.

### Configuration of instance 1

For the first instance, we can use the default configuration such as the Connector, AJP, or shutdown port in `server.xml`. Let's discuss each component where configuration needs to be done and why it is used:

1. Shutdown port: The following screenshot shows the configuration for the shutdown port for the Tomcat instance. While running multiple instances, if you have skipped configuring the port, then the Tomcat instance will be unable to initiate it.

---

```
-->
<!-- A "Shutdown" listener that can be used to gracefully shutdown the
      server. Documentation at /docs/config/daemon.html -->
<Listener className="org.apache.catalina.core.StandardThreadExecutor" />
-->
<!-- APR library loader. Documentation at /docs/apr.html -->
```

2. Connector port: The following screenshot shows the configuration of the Connector port. This port is used to access the Tomcat instance, for example, normally you access the Tomcat instance by using <http://localhost:8080>. The 8080 port is called the connector port. While running multiple instances, if you have skipped configuring the port, then the Tomcat instance will be unable to start it and you will get the Port already in use exception.

---

```
<!-- A "Connector" represents an endpoint by which requests are received
      and responses are returned. Documentation at :
      Java HTTP Connector: /docs/config/http.html (blocking & non-blocking)
      Java AJP Connector: /docs/config/ajp.html
      APR (HTTP/AJP) Connector: /docs/apr.html
      Define a non-SSL HTTP/1.1 Connector on port 8080
-->
<Connector port="8080" protocol="HTTP/1.1"
      connectionTimeout="20000"
      redirectPort="8443" />
<!-- A "Connector" using the shared thread pool-->
<!--
```

3. AJP port: The following screenshot shows the AJP port configuration for Tomcat. This port is used for AJP communication between the Apache HTTP and the Tomcat instance. While running multiple instances, if you have skipped configuring this port, then the Tomcat instance will be unable to start and you will get a Port already in use exception.
- 

```
<!-- Define an AJP 1.3 Connector on port 8009 -->
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

4. Cluster attributes: Enable the cluster attributes for clustering in server.xml. The following screenshot shows the cluster class used for clustering:
- 

```
<!--For clustering, please take a look at documentation at:
/docs/cluster-howto.html (simple how to)
/docs/config/cluster.html (reference documentation) -->
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
```

- In horizontal clustering every machine has a separate IP. We

have to configure the broadcast address and port for the instance to connect with each other and create a cluster session. Add the following code to `server.xml` to enable broadcast setting and replication:

```
...
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
channelSendOptions="6">
  <Manager className="org.apache.catalina.ha.session.BackupManager"
expireSessionsOnShutdown="false" notifyListenersOnReplication="true"
mapSendOptions="6"/>
  <Channel className="org.apache.catalina.tribes.group.GroupChannel">
    <Membership className="org.apache.catalina.tribes.membership.McastService"
address="228.0.0.4" port="54446" frequency="500" dropTime="3500"/>
    <Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
address="auto" port="6000" selectorTimeout="100" maxThreads="6"/>
    <Sender className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
    <Transport className="org.apache.catalina.tribes.transport.nio.
PooledParallelSender"/>
  </Sender>
</Channel>
  <Deployer className="org.apache.catalina.ha.deploy.FarmWarDeployer"
tempDir="/opt/apachetomcat1/tomcat8-temp/" deployDir="/opt/apachetomcat1/tomcat8-
deploy/" watchDir="/opt/apachetomcat1/tomcat8-listen/" watchEnabled="false"/>
  <ClusterListener className=
"org.apache.catalina.ha.session.ClusterSessionListener"/>
</Cluster>
...
```



- The first highlighted code section shows the multicast IP.

Multicast creates a communication change for these two instances. The second highlighted section shows the deployment properties for the cluster instances.

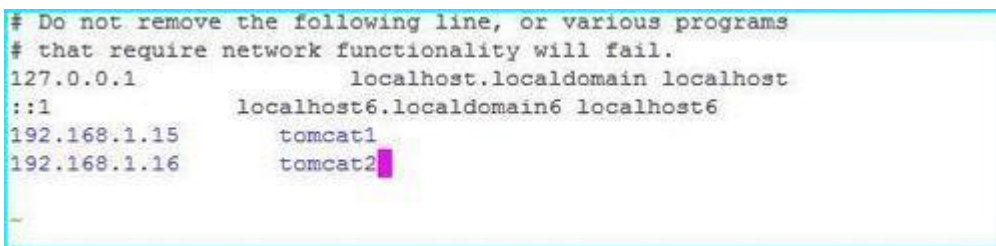
5. Configuration test: Run the configtest.sh script from TOMCAT\_HOME/bin to check the configuration. The following screenshot shows the output for the following config.sh command:

```
[root@localhost bin]# ./configtest.sh
```



A terminal window showing the output of the configtest.sh script. The output includes environment variables like CATALINA\_BASE, CATALINA\_HOME, CATALINA\_TMPDIR, JRE\_HOME, and CLASSPATH. It also shows the Java version (1.4.0\_24) and the path to the Java runtime. The script successfully initializes the ProtocolHandler and the Catalina load.

6. Host entry: Add the instance IP in the host files (/etc/hosts). The following screenshot shows the output for /etc/hosts file:



A screenshot of the /etc/hosts file. It shows the following entries: 127.0.0.1 localhost.localdomain localhost, ::1 localhost6.localdomain6 localhost6, 192.168.1.15 tomcat1, and 192.168.1.16 tomcat2.

7. Tomcat instance startup: Start the instance 1 configuration using the script startup.sh. The following screenshot shows the output for the following startup.sh script:

```
[root@localhost bin]# ./startup.sh
```



A terminal window showing the output of the startup.sh script. The output includes environment variables like CATALINA\_BASE, CATALINA\_HOME, CATALINA\_TMPDIR, JRE\_HOME, and CLASSPATH. It also shows the Java version (1.4.0\_24) and the path to the Java runtime. The script successfully initializes the ProtocolHandler and the Catalina load.

- Check the Tomcat instance process using the following mentioned

command. The following screenshot displays the output for the following `ps` command:

```
...  
[root@localhost bin]# ps -ef |grep java  
...
```

```
[root@localhost bin]# ps -ef |grep java  
root      11766      1 11 17:00 pts/3    00:00:17 /opt/jdk1.6.0_24/bin/java -Djava  
m -Xmx512m -XX:MaxPermSize=256m -Dorg.jboss.resolver.warning=true -Dsun.rmi.dgc.  
.logging.manager=org.apache.juli.ClassLoaderLogManager -Djava.awt.headless=true  
e.port=7091 -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.manageme  
sspath /opt/apache-tomcat1/bin/bootstrap.jar:/opt/apache-tomcat1/bin/tomcat-juli  
-Djava.io.tmpdir=/opt/apache-tomcat1/temp org.apache.catalina.startup.Bootstrap  
root      11902 10149  0 17:02 pts/3    00:00:00 grep java  
[root@localhost bin]#
```

## Configuration of instance 2

In order to start the configuration of instance 2, install Tomcat on the other machine and perform the same steps as performed on node 1.

Check `catalina.out` for both the nodes. The following mentioned logs show the activity performed during the startup of the Tomcat instance with the cluster instance. It also gives us complete visibility of the clustering functionality.

```
Oct 11, 2011 5:00:24 PM org.apache.catalina.ha.tcp.SimpleTcpCluster startInternal  
INFO: Cluster is about to start  
Oct 11, 2011 5:00:24 PM org.apache.catalina.tribes.transport.ReceiverBase bind  
INFO: Receiver Server Socket bound to:/192.168.1.15:4000  
Oct 11, 2011 5:00:24 PM org.apache.catalina.tribes.membership.McastServiceImpl  
setupSocket  
# Instance node 1 started on port 4000  
to establish cluster membership, start level:4  
Oct 11, 2011 5:00:25 PM org.apache.catalina.tribes.membership.McastServiceImpl  
waitForMembers  
INFO: Done sleeping, membership established, start level:4  
Oct 11, 2011 5:00:25 PM org.apache.catalina.tribes.membership.McastServiceImpl  
waitForMembers  
INFO: Sleeping for 1000 milliseconds to establish cluster membership, start level:8  
Oct 11, 2011 5:00:26 PM org.apache.catalina.tribes.membership.McastServiceImpl  
waitForMembers  
# waiting for other member to join the cluster  
INFO: Server startup in 13807 ms  
Oct 11, 2011 5:23:42 PM org.apache.catalina.tribes.io.BufferPool getBufferPool  
INFO: Created a buffer pool with max size:104857600 bytes of  
type:org.apache.catalina.tribes.io.BufferPool15Impl  
Oct 11, 2011 5:23:43 PM org.apache.catalina.ha.tcp.SimpleTcpCluster memberAdded  
INFO: Replication member added:org.apache.catalina.tribes.membership.MemberImpl  
[tcp://{192.168.1.16, 0, 0, 1}:4001,{192, 168, 1, 16},4001, alive=1043, securePort=-1,  
UDP Port=-1, id={33 91 -59 78 -34 -52 73 -9 -99 124 -53 34 69 21 -40 -82 }, payload=  
{}, command={}, domain={}, ]  
#Instance 2 joined the cluster node
```

The following are the logs for node 2:

```

INFO: Starting Servlet Engine: Apache Tomcat/8.5.61
Oct 11, 2011 5:23:41 PM org.apache.catalina.ha.tcp.SimpleTcpCluster startInternal
INFO: Cluster is about to start
Oct 11, 2011 5:23:42 PM org.apache.catalina.tribes.transport.ReceiverBase bind
INFO: Receiver Server Socket bound to:/192.198.1.16:4001
Oct 11, 2011 5:23:42 PM org.apache.catalina.tribes.membership.McastServiceImpl
setupSocket
# Instance node 1 started on port 4001
INFO: Setting cluster mcast soTimeout to 500
Oct 11, 2011 5:23:42 PM org.apache.catalina.tribes.membership.McastServiceImpl
waitForMembers
INFO: Sleeping for 1000 milliseconds to establish cluster membership, start level:4
Oct 11, 2011 5:23:43 PM org.apache.catalina.ha.tcp.SimpleTcpCluster memberAdded
INFO: Replication member added:org.apache.catalina.tribes.membership.MemberImpl
[tcp://{192,168, 1, 15}:4000,{127, 0, 0, 1},4000, alive=1398024, securePort=-1, UDP
Port=-1, id={28 42 60 -68 -99 126 64 -35 -118 - 97 7 84 26 20 90 24 }, payload={},
command={}, domain={}, ]
# Instance 1 joined the cluster node 2.

```

In the previously mentioned code, four sections are highlighted. Let's discuss each section briefly:

- The first section shows that `tomcatnode1` is started and ready to receive the cluster message on port 4000.

```
INFO: Receiver Server Socket bound to:/192.168.1.15:4000
```

- The second section shows that `tomcatnode2` had joined the cluster, and node 1 is getting the notification.

```
added:org.apache.catalina.tribes.membership.MemberImpl[tcp://{192.168.1.16, 0,
0, 1}:4001,{192, 168, 1, 16},4001, alive=1043
```

- The third section shows that `tomcatnode2` is started and ready to receive the cluster message on port 4000.

```
INFO: Receiver Server Socket bound to:/192.198.1.16:4001
```

- The fourth section shows that `tomcatnode1` had joined the cluster, and node 2 is getting the notification.

```
added:org.apache.catalina.tribes.membership.MemberImpl[tcp://{192,168, 1,
15}:4000,{127, 0, 0, 1},4000, alive=1398024
```

## Apache web server configuration for horizontal clustering

We have done the Tomcat level configuration to configure horizontal clustering on the Tomcat instance. It's time to integrate the Apache web server to Tomcat 8. Let's enable the integration by performing the following steps:

1. We have to create a new file called `mod_jk.conf` in the `conf` directory of `APACHE_HOME/conf` using the following commands:

```
[root@localhost apache-2.0]# cd /opt/apache-2.2.19/conf
vi mod-jk.conf
```

- The following mentioned code defines the configuration

```
parameters for `mod_jk.conf`:
```

```
```\nLoadModule jk_module modules/mod_jk.so\nJkWorkersFile conf/workers.properties\nJkLogFile logs/mod_jk.log\nJkLogLevel info\nJkMount /sample/* loadbalancer\nJkMount /* loadbalancer\n```\n
```

2. Create a new file named workers.properties in the conf directory using the following command:

```
[root@localhost conf]# vi workers.properties\nworker.list=tomcatnode1, tomcatnode2, loadbalancer
```

Define the `worker.list` for the entire nodes in the cluster:

```
```\nworker.tomcatnode1.port=8009\nworker.tomcatnode1.host=192.168.1.15\nworker.tomcatnode1.type=ajp13\nworker.tomcatnode1.lbfactor=1\n```\n
```

The previous lines of code define the `tomcatnode1` properties. The highlighted code shows the IP address of `tomcatnode1`. This is essential for horizontal clustering.

```
```\nworker.tomcatnode2.port=8009\nworker.tomcatnode2.host=192.168.1.16\nworker.tomcatnode2.type=ajp13\nworker.tomcatnode2.lbfactor=1\n```\n
```

The previous lines of code define the `tomcatnode2` properties. The highlighted code shows the IP address of `tomcatnode2`. This is essential for horizontal clustering.

```
```\nworker.loadbalancer.type=lb\nworker.loadbalancer.balanced_workers=tomcatnode1, tomcatnode2
```

```
worker.loadbalancer.sticky_session=1
...
```

The previous lines of code define the load balancing properties for `mod_jk``.

### Note

The only difference in the Apache configuration for `workers.properties`` for horizontal and vertical clustering is vertical hosting. (`worker.tomcatnode2.host`` is configured as `localhost`, whereas in horizontal clustering `worker.tomcatnode2.host`` is configured with the IP address of a different machine.)

3. The last step is to include the `mod_jk.conf` in the main configuration file of `httpd`, that is `httpd.conf` and reload the Apache services using the following command:

```
[root@localhost conf]# vi httpd.conf
```

### Note

Include `conf/mod_jk.conf`` in the end of `httpd.conf``.

## Testing of the clustered instance

To perform cluster testing, we are going to take you through a sequence of events. In the following event, we only plan to use two Tomcat instances--- `tomcatnode1` and `tomcatnode2` . We will cover the following sequence of events:

1. Start `tomcatnode1`.
2. Start `tomcatnode2` (wait for node 1 to start completely).
3. Node 1 crashes.
4. Node 2 takes over the user session of node 1 to node 2.
5. Start node 1 (wait for node 1 to start completely).
6. Node 2 and node 1 are in running state.

Now we have a good scenario with us, we will walk through how the entire process works:

1. Start instance 1: `tomcatnode1` starts up using the standard startup sequence. When the host object is created, a cluster object is associated with it. Tomcat asks the cluster class (in this case `SimpleTcpCluster`) to create a manager for the cluster and the cluster class will start up a membership service.

### Note

The membership service is a mechanism in the cluster instance through the cluster domain, which adds the member node in the

cluster. In simple terms, it is a service through which members are able to join the cluster.

2. Start instance 2: When Tomcat instance 2 starts up, it follows the same sequence as tomcatnode2 with one difference. The cluster is started and will establish a connection (tomcatnode1, tomcatnode2). tomcatnode2 will now send a request to the server that already exists in the cluster, which is now tomcatinstance2.

#### Note

In case the Tomcat instance does not respond within an interval of 60 seconds, then Tomcat instance 2 will update the cluster, and generate the entry in the logs.

3. Node 1 crashes: Once the Tomcat instance crashes, the cluster manager will send a notification to all the members, in our case it's tomcatnode2. The entire session of node 1 will be replicated to node 2, but the user will not see any issues while browsing the website.
4. Node 2 will take over the user session of node 1 to node 2: tomcatnode2 will process the request as with any other request. User requests are served with node 2.
5. Start instance 1: Upon start up, tomcatnode1 first joins the cluster, and then contacts tomcatnode2 for the current state of all the users in the session. It starts serving the user requests and shares the load for node 2.
6. Node 2 and node 1 are in running state: Now both the instances are in running state. Node 2 will continue to serve the user requests and once the request is served, it will terminate the user session.

If the previous mentioned test is working, it means clustering is working fine.

## Monitoring of Tomcat clustering

Once the cluster is up and working, the next stage is to set up the monitoring of the clustering. This can be done in the following ways:

- Various monitoring tools
- Scripts
- Manual

Following are the steps to manually monitor the clusters:

1. Check the Tomcat process using the following command:

```
root@localhost bin]# ps -ef |grep java
```

2. Check the logs to verify the connectivity of the cluster.
3. Verify the URL for both cluster members.

#### Summary

---

In this lab, we have discussed the clustering of Tomcat 8 and its implementation techniques. We have discussed clustering architecture, horizontal and vertical clusters and their benefits, the implementation of horizontal and vertical clustering on Tomcat 8, and the verification of clusters.

In the next lab, we will discuss the most awaited topic of Tomcat 8, that is, Tomcat 6 upgrade to Tomcat 8 and different techniques used during the upgrade.