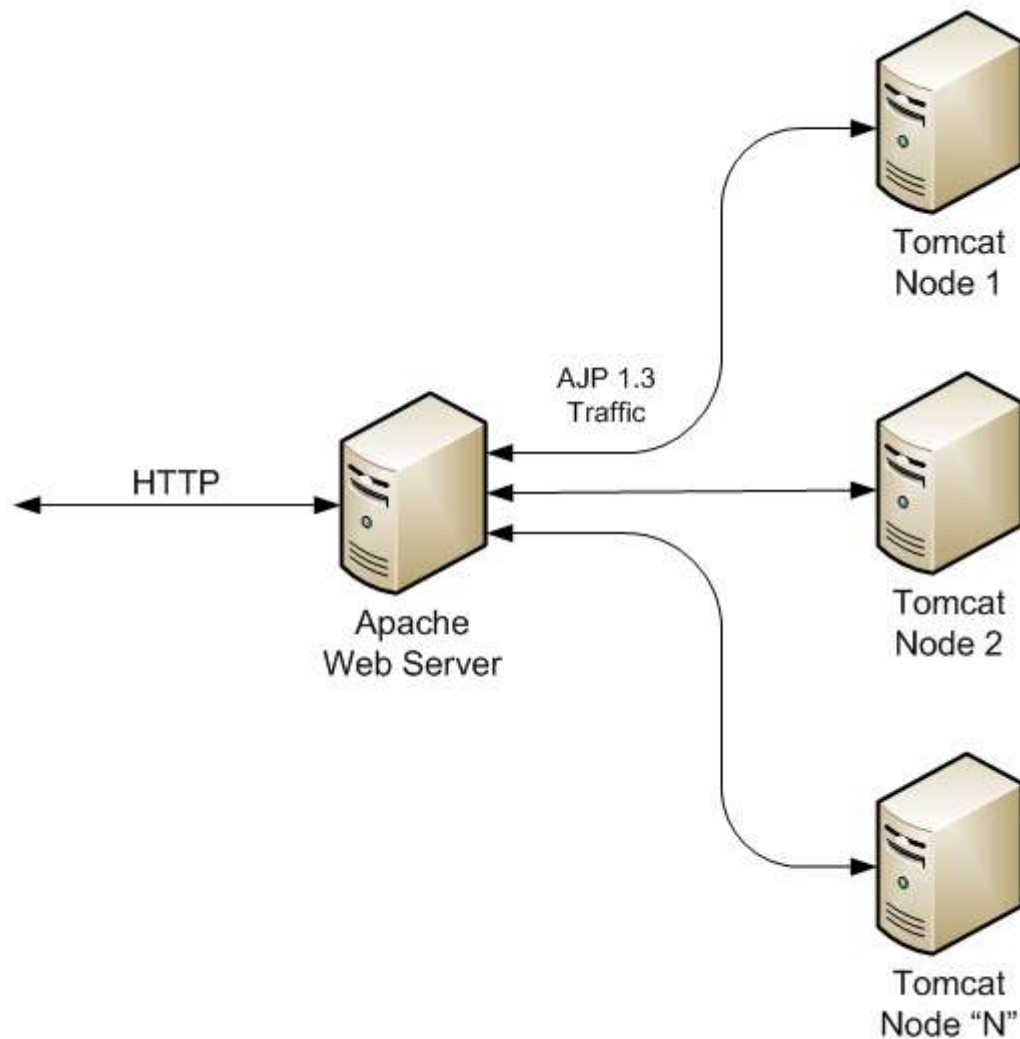


TomEE Clustering Setup using mod_proxy

There are pretty much two ways to set up basic clustering, which use two different Apache modules. The architecture for both, is the same. Apache sits in front of the Tomcat nodes and acts as a load balancer.



Traffic is passed between Apache and Tomcat(s) using the binary AJP 1.3 protocol. The two modules are *mod_jk* and *mod_proxy*.

mod_jk stands for “*jakarta*” the original project under which Tomcat was developed. It is the older way of setting this up, but still has some advantages.

mod_proxy is a newer and more generic way of setting this up. The rest of this guide will focus on **mod_proxy**, since it ships “out of the box” with newer versions of Apache.

You should be able to follow this guide by downloading Apache and TomEE default distributions and following the steps.

Clustering Background

You can cluster at the request or session level. Request level means that each request may go to a different node – this is the ideal since the traffic would be balanced across all nodes, and if a node goes down, the user has no idea. Unfortunately this requires session replication between all nodes, not just of HttpSession, but ANY session state.

Session level clustering means if your application is one that requires a login or other forms of session-state, and one or more your Tomcat nodes goes down, on their next request, the user will be asked to log in again, since they will hit a different node which does not have any stored session data for the user.

This is still an improvement on a non-clustered environment where, if your node goes down, you have no application at all!

And we still get the benefits of load balancing across nodes, which allows us to scale our application out horizontally across many machines. Anyhow without further ado, let's get into the how-to.

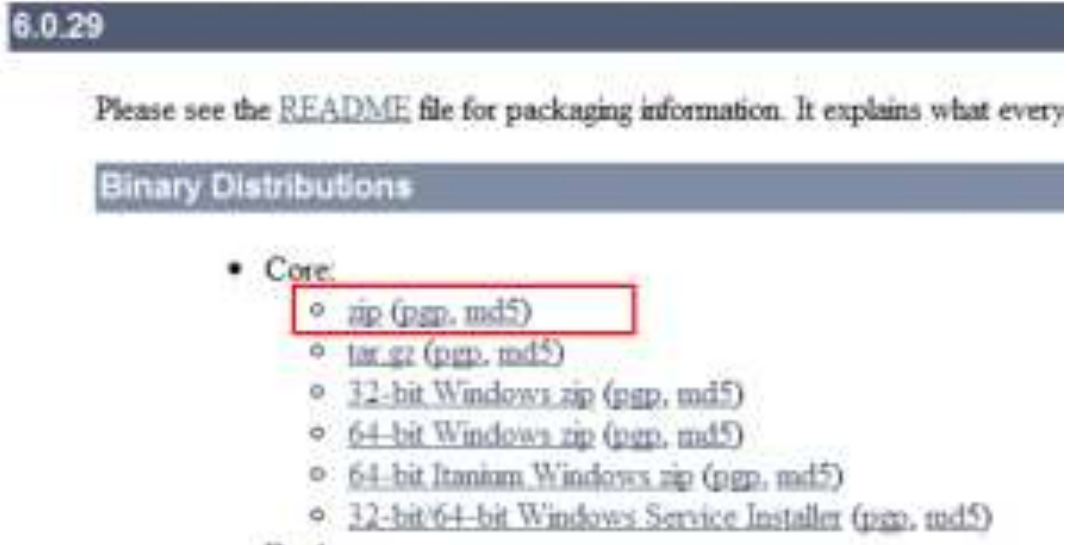
Setting Up The Nodes

In most situations you would be deploying the nodes on physically separate machines, but in this example we will set them up on a single machine, but on different ports. This allows us to easily test this configuration.

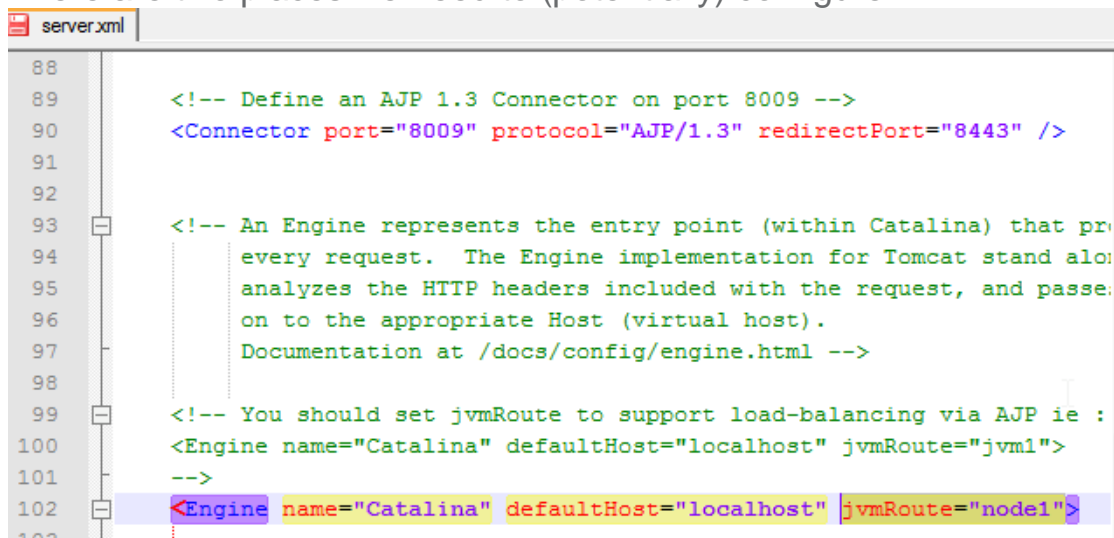
Nothing much changes for the physically separate set up – just the Hostnames of the nodes as you would expect.

Oh and I'm working on Windows – but aside from the installation of Apache and Tomcat nothing is different between platforms since the configuration files are standard on all platforms.

1.



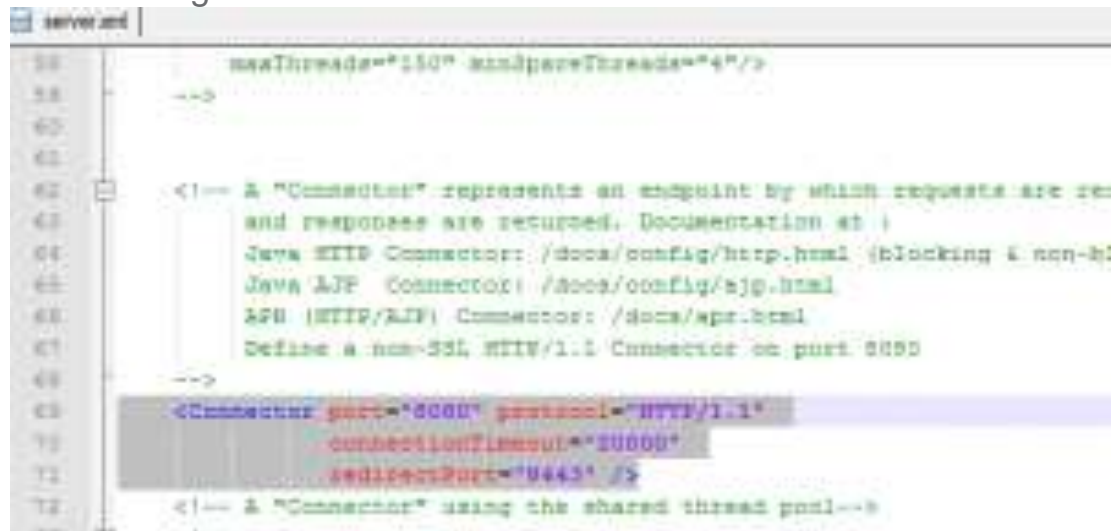
2. We'll use a folder to install all this stuff in. Let's say it's "C:\cluster" for the purposes of the exercise.
3. Unzip the Tomcat distro twice, into two folders –
C:\cluster\tomcat-node-1
C:\cluster\tomcat-node-2
4. Start up each of the nodes, using the bin/startup.bat / bin/startup.sh scripts. Ensure they start. If they don't you may need to point Tomcat to the JDK installation on your machine.
5. Open up the server.xml configuration on
c:\cluster\tomcat-node-1\conf\server.xml
6. There are two places we need to (potentially) configure –



The first line is the connector for the AJP protocol. The “port” attribute is the important part here. We will leave this one as is, but for our second (or subsequent) Tomcat nodes, we will need to change it to a different value. The second part is the “engine” element. The “jvmRoute” attribute has to be added – this configures the name of

this node in the cluster. The “jvmRoute” must be unique across all your nodes. For our purposes we will use “node1” and “node2” for our two node cluster.

7. This step is optional, but for production configs, you may want to remove the HTTP connector for Tomcat – that’s one less port to secure, and you don’t need it for the cluster to operate. Comment out the following lines of the server.xml –



8. Add those lines to the “Engine” node:

<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="45564"
frequency="500"
dropTime="3000" />

```

<Receiver
className="org.apache.catalina.tribes.transport.nio.NioReceiver"
address="auto"
port="5000"
selectorTimeout="100"
maxThreads="6" />

<Sender
className="org.apache.catalina.tribes.transport.ReplicationTransmitter">

<Transport
className="org.apache.catalina.tribes.transport.nio.PooledParallelSender" />

</Sender>

</Channel>

<Valve className="org.apache.catalina.ha.tcp.ReplicationValve"
filter=".*\.gif|.*\js|.*\jpeg|.*\jpg|.*\png|.*\htm|.*\html|.*\css|.*\txt" />

<ClusterListener
className="org.apache.catalina.ha.session.ClusterSessionListener" />
</Cluster>

```

9. Save and close your file, then restart your TomEE. If everything is ok, your log file should have something like this:

```

org.apache.catalina.ha.tcp.SimpleTcpCluster startInternal
Cluster is about to start
org.apache.catalina.tribes.transport.ReceiverBase bind
Receiver Server Socket bound to:/192.168.0.104:5000
org.apache.catalina.tribes.membership.McastServiceImpl setupSocket
Setting cluster mcast soTimeout to 500
org.apache.catalina.tribes.membership.McastServiceImpl waitForMembers
Sleeping for 1000 milliseconds to establish cluster membership, start level:4
org.apache.catalina.tribes.membership.McastServiceImpl waitForMembers
Done sleeping, membership established, start level:4
org.apache.catalina.tribes.membership.McastServiceImpl waitForMembers
Sleeping for 1000 milliseconds to establish cluster membership, start level:8
org.apache.catalina.tribes.membership.McastServiceImpl waitForMembers
Done sleeping, membership established, start level:8

```

10. Now you have the first node of your cluster up and running. Do it to another server, could be another machine or even a VM. Do it to as

many servers as you want according to your needs. Each time you add a new node, the log file of other nodes should contain something like this:

```
org.apache.catalina.tribes.io.BufferPool getBufferPool
Created a buffer pool with max size:104857600 bytes of
type:org.apache.catalina.tribes.io.BufferPool15Impl
org.apache.catalina.ha.tcp.SimpleTcpCluster memberAdded
Replication member
added:org.apache.catalina.tribes.membership.MemberImpl[tcp://{192, 168, 0,
107}:5000,{192, 168, 0, 107},5000, alive=1016, securePort=-1, UDP Port=-1,
id={111 -38 59 -66 -68 -29 72 -69 -103 12 -121 -120 -13 -25 -90 17 },
payload={}, command={}, domain={}, ]
```

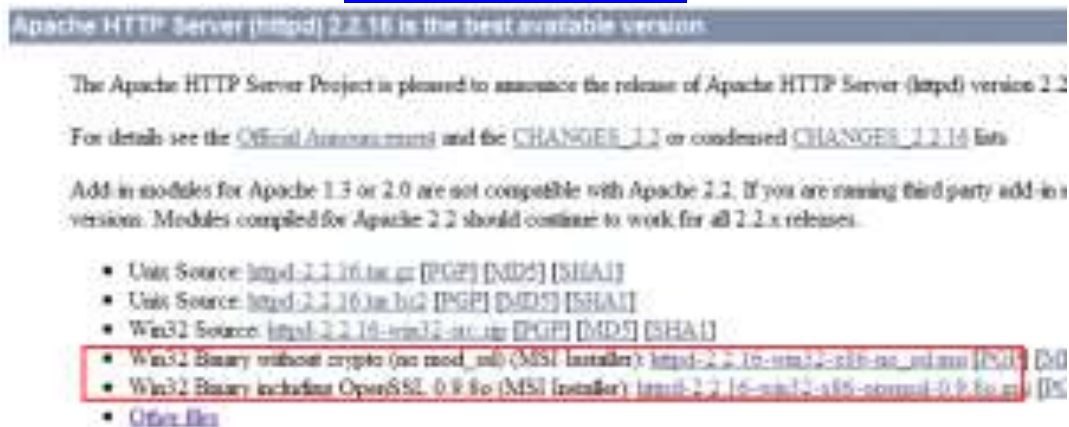
11. Now you have your beautiful cluster fully happy and working!
12. Now repeat this for `C:\cluster\tomcat-node-2\conf\server.xml`
Change the `jvmRoute` to “node2” and the AJP connector port to “8010”.

We’re done with Tomcat. Start each node up, and ensure it still works.

Setting Up The Apache Cluster

Okay, this is the important part.

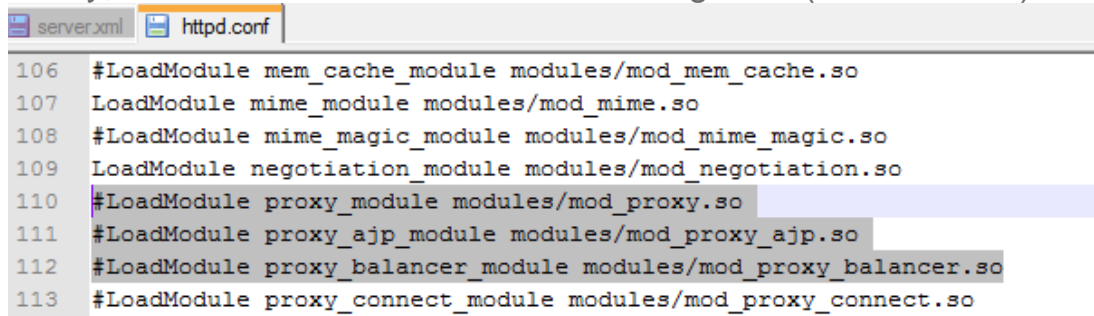
1. Download and install [Apache HTTP Server](#).



Use the custom option to install it into `C:\cluster\apache2.2`

2. Now open up `c:\cluster\apache2.2\conf\httpd.conf` in your favourite text editor.

3. Firstly, we need to uncomment the following lines (delete the '#') –



The screenshot shows a text editor with two tabs: 'server.xml' and 'httpd.conf'. The 'httpd.conf' tab is active, showing lines 106 through 113. Lines 106, 108, 110, 111, 112, and 113 are highlighted in blue, indicating they are the lines to be uncommented. The lines are:

```
106 #LoadModule mem_cache_module modules/mod_mem_cache.so
107 LoadModule mime_module modules/mod_mime.so
108 #LoadModule mime_magic_module modules/mod_mime_magic.so
109 LoadModule negotiation_module modules/mod_negotiation.so
110 #LoadModule proxy_module modules/mod_proxy.so
111 #LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
112 #LoadModule proxy_balancer_module modules/mod_proxy_balancer.so
113 #LoadModule proxy_connect_module modules/mod_proxy_connect.so
```

These enable the necessary **mod_proxy** modules in Apache.

4. Finally, go to the end of the file, and add the following:



The screenshot shows the configuration for a Proxy balancer. The configuration is as follows:

```
<Proxy balancer://testcluster stickysession=JSESSIONID>

BalancerMember ajp://127.0.0.1:8009 min=10 max=100 route=node1 loadfactor=1

BalancerMember ajp://127.0.0.1:8019 min=20 max=200 route=node2 loadfactor=1

</Proxy>

ProxyPass /examples balancer://testcluster/examples
```

The above is the actual clustering configuration.

The first section configures a load balancer across our two nodes.

The *loadfactor* can be modified to send more traffic to one or the other node. i.e. how much load can this member handle compared to the others?

This allows you to balance effectively if you have multiple servers which have different hardware profiles.

Note also the “route” setting which must match the names of the “*jvmRoutes*” in the Tomcat server.xml for each node. This in conjunction with the “*stickysession*” setting is key for a Tomcat cluster, as this configures the session management. It tells *mod_proxy* to look for the node’s *route* in the given session cookie to determine which node that session is using. This allows all requests from a given client to go to the node which is holding the session state for the client.

The *ProxyPass* line configures the actual URL from Apache to the load balanced cluster. You may want this to be “/”

e.g. “*ProxyPass / balancer://testcluster/*”

In our case we’re just configuring the Tomcat **/examples** application for our test.

5. Save it, and restart your Apache server.

Test It Out

With your Apache server running you should be able to go to <http://localhost/examples>

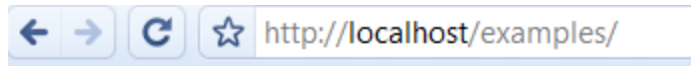
You should get a 503 error page as per below –
Service Temporarily Unavailable

The server is temporarily unable to service your request due to maintenance downtime or capacity problems. Please try again later.

This is because both Tomcat nodes are down.

Start up *node1* (*c:\cluster\tomcat-node-1\bin\startup*) and reload <http://localhost/examples>

You should see the examples application from the default Tomcat installation –



Apache Tomcat Examples

- [Servlets examples](#)
- [JSP Examples](#)

Shut down **node1**, and then start up **node2**. Repeat the test. You should see the same page as above. We have transparently moved from node1 to node2 since node1 went down.

Start both nodes up and your cluster is now working.

You’re done!

Optional: Set Up Apache Balancer Manager

mod_proxy has an additional “balancer manager” component which provides a nice web interface to the load balanced cluster. It’s worthwhile setting this up if you want to remotely administer / monitor the cluster.

To do so is easy –

1. Add the following to the bottom of your *C:\cluster\apache2.2\conf\httpd.conf*

```
<Location /balancer-manager>

SetHandler balancer-manager

AuthType Basic

AuthName "Balancer Manager"

AuthUserFile "C:/cluster/apache2.2/conf/.htpasswd"

Require valid-user

</Location>
```

This configures the balancer manager at <http://localhost/balancer-manager>

2. We need to create a password file to secure it. At the command prompt you can use –

```
c:\cluster\apache2.2\bin\htpasswd -c c:\cluster\apache2.2\conf\htpasswd admin
```

Then set a password when prompted. This password would be used by the balancer-manager URL to authenticate.

Restart your Apache web server, and go to <http://localhost/balancer-manager>

You should be prompted for a username/password as you set before, and see the balancer manager tool as below:

Load Balancer Manager for localhost

Server Version: Apache/2.2.16 (Win32)

Server Built: Jul 30 2010 16:15:37

LoadBalancer Status for balancer://testcluster

StickySession Timeout FailoverAttempts Method

JSESSIONID 0 1 byrequests

Worker URL	Route	RouteRedir	Factor	Set	Status	Elected	To	From
ajp://127.0.0.1:8009	node1		1	0	Err	1	0	0
ajp://127.0.0.1:8019	node2		1	0	Err	1	0	0

Dynamic and Static Discovery cluster with Apache TomEE

Now let's see quickly how we can set them up at the Apache TomEE. So we will do just a little zoom to see where the things are done.

Dynamic Discovery

At the "Engine" node you should put this code:

```
<Cluster
className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
channelSendOptions="6">
```

```
<Manager...
```

```
<Channel...
```

```
<Valve...
```

```
<Deployer...
```

```
<ClusterListener...
```

```
</Cluster>
```

Static Discovery

At the same “Engine” node you should put this:

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
channelSendOptions="6">

<Channel className="org.apache.catalina.tribes.group.GroupChannel">

<Interceptor
className="org.apache.catalina.tribes.group.interceptors.StaticMembershipIn
terceptor">

<Member className="org.apache.catalina.tribes.membership.StaticMember"
port="4000" host="server1" uniqueId="{0,0,0,0,0,0,0,0,0,0,0,0,0,0,1}" />
<Member className="org.apache.catalina.tribes.membership.StaticMember"
port="4000" host="server2" uniqueId="{0,0,0,0,0,0,0,0,0,0,0,0,0,0,2}" />
<Member className="org.apache.catalina.tribes.membership.StaticMember"
port="4000" host="server3" uniqueId="{0,0,0,0,0,0,0,0,0,0,0,0,0,0,3}" />

</Interceptor>

</Channel>

</Cluster>
```

So as you can see, with the Static Discovery you use the StaticMember at the Member node instead of McastService used at the Dynamic Discovery.