# Basics of clustering with Apache Tomcat

A Tomcat worker is a Tomcat instance that is waiting to execute servlets on behalf of some web application. For example, we can have a web server such as Apache Http forwarding requests to a Tomcat process (the worker) running behind it.

The scenario described above is a very simple one; in fact one can configure multiple Tomcat workers to serve servlets on behalf of a certain web server. The reasons for such configuration can be:

* We want different contexts to be served by different Tomcat workers to provide a development environment where all the developers share the same web server but own a Tomcat worker of their own.
* We want different virtual hosts served by different Tomcat processes to provide a clear separation between sites belonging to different companies.
* We want to provide load balancing, meaning run multiple Tomcat workers each on a machine of its own and distribute the requests between them.

The package we prepared for this training contains a ready to use installation for Apache Tomcat 6.0.36 with multiple instances that can be used as master and/or slaves depending on the type of clustering we want to perform.

## Apache Tomcat configuration for vertical clustering

The first step to perform in order to create a vertical cluster with Apache Tomcat to run multiple instances on the same box (physical or virtual) is actually have a single Tomcat HOME folder containing the standard Apache Tomcat binaries but then configuring additional Tomcat BASE folders containing the configurations for each instance (We are assuming the reader is familiar with Tomcat and relative concepts and terminology).

First of all we should download a version of the Apache Tomcat binaries (in this document we are using 6.0.36) and extract them into a common folder, which will become the CATALINA\_HOME default folder.

Next step, we should start to create a new folder for each instance we want to have. This folder will contain the following sub-folders, copied from the original Tomcat folders that are part of the download:

* bin/ : we should empty this directory after we copy it. It will contain scripts to set environmental variables
* conf/ :
* logs/ : empty; will contain the Tomcat logs for the specific instance
* temp/ : empty; will contain the temp files for the specific instance
* work/ : empty; will contain the compiled JSPs for the specific instance

You need to repeat this step for each instance we want to have. In our sample package we created multiple directories which we’ll allow us to run multiple instances of Tomcat on the same machine. On Windows you can find the instances under:

%TRAINING\_ROOT%\tomcat-7.0.72\instances\

On Linux you can find the instances under:

\opt\tomcat\_server\\opt\tomcat\_server2\

## Tomcat connectors

Now, we need to tweak the ports used by each Tomcat instance. This means opening up the **conf/server.xml** file inside the conf directory for each instance and making sure the various instances use different ports for each service. As an instance we did the following:

1. Instance 1
2. shutdown port 8005
3. HTTP connector port 8083
4. AJP connector port 8009
5. HTTPS port 8443
6. Instance 2
7. shutdown port 8006
8. HTTP connector port 8082
9. AJP connector port 8010
10. HTTPS port 8444

### HTTP

The HTTP connector is setup by default with Tomcat, and is ready to use. This connector features the lowest latency and best overall performance. For clustering, a HTTP load balancer with support for web sessions stickiness must be installed to direct the traffic to the Tomcat servers. Tomcat supports mod\_proxy (on Apache HTTP Server 2.x, and included by default in Apache HTTP Server 2.2) as the load balancer. It should be noted that the performance of HTTP proxying is usually lower than the performance of AJP, so AJP clustering is often preferable.

### AJP

When using a single server, the performance when using a native webserver in front of the Tomcat instance is most of the time significantly worse than a standalone Tomcat with its default HTTP connector, even if a large part of the web application is made of static files. If integration with the native webserver is needed for any reason, an AJP connector will provide faster performance than proxied HTTP. AJP clustering is the most efficient from the Tomcat perspective. It is otherwise functionally equivalent to HTTP clustering.

## Prepare Tomcat connectors

Note also that we have configured a route for each slave, all of the above settings should be done into the ${CATALINA\_BASE}/conf/server.xml file of each tomcat slave instance:

### HTTP connector

Edit the **conf/server.xml** file located into your CATALINA\_BASE instance folder and modify the connector as following:

<!-- 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 8083 -->

<Connector port="8083" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />

.. note:: To add multiple instances you may modify the port attributes.

### AJP connector

The following steps show you how to set the ajp connector port.

Edit the **conf/server.xml** file located into your CATALINA\_BASE instance folder and modify the connector as following:

<!-- Define an AJP 1.3 Connector on port 8009 -->

<Connector port="8009" protocol="AJP/1.3" redirectPort="8443"/>

and the route name into the Engine node:

<!-- You should set jvmRoute to support load-balancing via AJP ie :<Engine name="Catalina" defaultHost="localhost" jvmRoute="jvm1">-->

<Engine defaultHost="localhost" name="Catalina" jvmRoute="route1">

# Personalize Server instances

In the following sections we will go over the steps needed to configure multiple Tomcat instances to run Server in a perfect Active-Active clustered configuration with Apache HTTP.

## Step 1: Basic Customization for instances’ environment variables

* Create a setenv.bat file in the bin folder for each instance
* Make sure the JAVA\_HOME and JRE\_HOME contains the full path of the JDK you intend to use
* Make sure CATALINA\_BASE contains the full path of the Apache Tomcat base folder
* Set the instance name to use the same name of the directory that contains the config for such an instance
* In the provided training **Linux environment** the variable is configured in *setenv.sh* located in the conf folder of each tomcat instance: /opt/tomcat\_server/conf/setenv.sh and

/opt/tomcat\_server2/conf/setenv.sh

The setenv.bat so far looks like the following:

REM Setting Java

set JAVA\_HOME=%TRAINING\_ROOT%\jdk

set JRE\_HOME=%JAVA\_HOME%/jre

rem basic Tomcat configuration

set INSTANCE\_NAME=instance1

set CATALINA\_HOME=%TRAINING\_ROOT%\tomcat-7.0.72set CATALINA\_BASE=%CATALINA\_HOME%\instances\%INSTANCE\_NAME%

Notice how we initialized the CATALINA\_BASE environmental variable.

This set up works since the various instances have been created in a path contained within the CATALINA\_HOME directory where the parent Tomcat instances resides.

It is also crucial to remark that the instance name must match the name of the enclosing directory for the instance’s configuration.

## Step 2: Customization for instances’ environment variables related to Server

We are now going to provide some basic hints on how to customize environmental variable in order to run two instance of Server in parallel on the same machine sharing the same data directory in order to create a perfect cluster in Active-Active mode. Notice that we are assuming that the data directory to use sits right next to the instances directories.

All the instructions below refer again to the setenv.bat file and must be replicated on all instances.

1. Make sure the server\_data directory is properly set for all instances in order to have them point to the same location
2. Make sure the SERVER\_LOG\_LOCATION points to different locations for each instances in order to prevent them for writing on the same log file
3. If your Server comes with integrated WebCache:
   1. Make sure the WEBCACHE\_CACHE\_DIR is the same on all instances
   2. Disable the diskquota or enable support for clustered disk quota