Apache and Tomcat Clustering Configuration

Table of Contents

Introduction

Idea behind this document is to consolidate the whole process required to run tomcat in the cluster mode. This document also provides the information related to integration of Apache HTTPD web server with tomcat and then run tomcat in the cluster mode and Tuning Apache an Tomcat.

Version Information: -

Apache HTTPD Web Server	:	2.0
Tomcat Server	:	5.5.25

We will connect Apache Server with Tomcat using mod_jk.so library. Mod_jk.so is a dynamically loaded module that Apache uses to recognize JSP, Servlet, or XML (using Cocoon) requests that need to be handled by Tomcat. The communication between Apache and Tomcat is coordinated by mod_jk.so over TCPIP port 8007.

Revision History

Revision No	Description	Revision Date	Author
1.0	This document was created.	08-24-2009	HeadStream Team

Download and Install JDK

Make sure you have installed JDK 1.5 on your system and corresponding path is set. Set the JAVA_HOME environment variable pointing to the location where you have installed the JDK. JDK can be downloaded from the following location:-
http://java.sun.com/j2se/1.5.0/download.jsp.

Download and Install Apache Web Server (httpd)

Visit http://httpd.apache.org/download.cgi to download the Apache 2.0 version and unzip it on to your file system. Set %APACHE_HOME% environment variable where you have unzipped the apache server.

Download and Install Tomcat server

Go to http://tomcat.apache.org/download-55.cgi and download the tomcat 5.5 version and unpack the zip file on your file system. Set the TOMCAT_HOME environment variable to the directory where you have unpacked the zip file.

Apache Web Server Configuration

Now we are required to configure Apache Web Server, which will be redirecting our request to the different tomcat servers, which are connected to Apache Server. Apache Server will act as a load balancer and depending upon the load balancing strategy configured, it will redirect the request to available clusters.

- Download Mod_jk.so from http://tomcat.apache.org/download-connectors.cgi
  Or the source can be downloaded and compiled but source compilation for this connector is out of scope of this document.
- Copy Mod_jk.so to %APACHE_HOME%\modules directory.
- Edit %APACHE_HOME%/conf/httpd.conf file and put the following entry:-

```
<IfModule mod_jk.c>
        # following line tells the apache that it should use workers.properties file which
        #contains information/configuration about the workers (cluster).
          JkWorkersFile conf/workers.properties
         # This entry provides the log file path information of the connector
          JkLogFile logs/mod_jk.log
          JkLogLevel info
        # this entry is mounted for the load balancer and '/*' indicates that all the request
        coming to apache should be redirected to tomcat through load balancer.
          JkMount /* loadbalancer
</IfModule>
```

- Create a new property file with the name of workers.properties in the same directory where httpd.conf is placed and add the following entries.

```
worker.list=loadbalancer

# working nodes
worker.worker1.type=ajp13
worker.worker1.host=localhost
worker.worker1.port=8009
worker.worker1.lbfactor=1
worker.worker1.cachesize=10
worker.worker1.cache_timeout=600
worker.worker1.socket_keepalive=1
worker.worker1.socket_timeout=300

# similarly add all the tomcat nodes where load balancing is required and then add the
following entry:-
worker.loadbalancer.type=lb
worker.loadbalancer.sticky_session=1
worker.loadbalancer.balance_workers=worker1, worker2 (if created)
```

# Change the host entry to the corresponding machine name where worker / cluster is running.

# Sticky session means that whether we require session replication or not on different clusters. #This is for mission critical application where is any server fails while catering to the particular # user request then other cluster can take care of the request as session is replicated.

Tomcat Nodes Configuration.

Enable AJP connector listener for each tomcat instance

The mod_jk load balancing module talks to tomcat using AJP 1.3 protocol. Therefore we need to have associated listener configured in tomcat. The port number of ajp connector should be same as the one configured in workers.properties file for this server instance (i.e. <TOMCAT_NODE1_AJP_PORT> or <TOMCAT_NODE2_AJP_PORT>)

For server instance 1 TOMCAT_NODE_AJP_PORT = 8009

```
<!-- Define an AJP 1.3 Connector  -->
 <Connector port="8009"
 enableLookups="false" redirectPort="8443" protocol="AJP/1.3"
URIEncoding="UTF-8" />
```

For server instance 2 TOMCAT_NODE_AJP_PORT = 8029

```
<Connector port="8029"
 enableLookups="false" redirectPort="8443" protocol="AJP/1.3"
URIEncoding="UTF-8" />
```

Sticky Session Setting

Set jvmRoute attribute for the tomcat container Engine. By default this attribute is not set. This attribute is used by load balancer module to work out sticky session load balancing.

For server instance 1

```
<!-- Define the top level container in our container hierarchy -->
   <Engine name="Catalina" defaultHost="localhost" jvmRoute="worker1">
```

For server instance 2

```
<!-- Define the top level container in our container hierarchy -->
   <Engine name="Catalina" defaultHost="localhost" jvmRoute="worker2">
```

You need to modify %TOMCAT_HOME%\conf\server.xml file of each tomcat instance to include that instance in cluster.
Following configuration needs to be added in **Engine** element of server.xml file

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

<Manager className="org.apache.catalina.ha.session.DeltaManager"
                                  expireSessionsOnShutdown="false"
                                  notifyListenersOnReplication="true"/>

<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="4000"
                              autoBind="100"
                              selectorTimeout="5000"
                              maxThreads="6"/>

<Sender className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
<Transport className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
</Sender>
<Interceptor className="org.apache.catalina.tribes.group.interceptors.TcpFailureDetector"/>
<Interceptor
className="org.apache.catalina.tribes.group.interceptors.MessageDispatch15Interceptor"/>
</Channel>

<Valve className="org.apache.catalina.ha.tcp.ReplicationValve"
```

```
            filter=""/>
<Valve className="org.apache.catalina.ha.session.JvmRouteBinderValve"/>

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

Adding this configuration in each tomcat will make that instance to be added in a cluster when it starts up (using multicast IP and port)

Now start tomcat and apache. On tomcat console, you should be able to see some clustered messages.

Example :

```
# workers.properties
ps=/

# list the workers by name
worker.list=tomcat1,tomcat2,tomcat3,loadbalancer

#
# Specifies the load balance factor when used with # a load balancing
worker.
# Note:
# ----> lbfactor must be > 0
# ----> Low lbfactor means less work done by the worker.

# -----------------------
# First tomcat server
# -----------------------
worker.tomcat1.port=18009
worker.tomcat1.host=192.168.70.101
worker.tomcat1.type=ajp13
worker.tomcat1.lbfactor=33

# -----------------------
# Second tomcat server
# -----------------------
worker.tomcat2.port=19009
worker.tomcat2.host=192.168.70.102
worker.tomcat2.type=ajp13
worker.tomcat2.lbfactor=33

# -----------------------
# Third tomcat server
# -----------------------
worker.tomcat3.port=20009
worker.tomcat3.host=192.168.70.103
worker.tomcat3.type=ajp13
worker.tomcat3.lbfactor=33
```

Apache and Tomcat configuration tuning

Apache Server works on threads to know that the consumption of memory each process is done you should see the operating system from the stack memory the server is set:

- We must run the command ulimit:

** For Linux:*

```
$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
file size               (blocks, -f) unlimited
pending signals                 (-i) 1024
max locked memory       (kbytes, -l) 32
max memory size         (kbytes, -m) unlimited
open files                      (-n) 10240
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
stack size              (kbytes, -s) 10240
cpu time               (seconds, -t) unlimited
max user processes              (-u) 114688
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

You can see that the stack is defined to 10240 Kb

When we look at the http configuration file (usually httpd.conf) in the default installation comes in the form of parameterized:

```
ThreadLimit         25
ServerLimit         64
StartServers         2
MaxClients         600
MinSpareThreads     25
MaxSpareThreads     75
ThreadsPerChild     25
MaxRequestsPerChild  0
```

You can see that ThreadsPerChild defined in 25

This means that for every thread that is launched is consumed (ThreadsPerChild) x (stack):

- For example, in Linux: 10240 x 25 = 256000

Therefore each thread can consume up to 250 MB

If you also look at the value of the maximum number of servers (ServerLimit 64) can be calculated the maximum memory that can be consumed by processes http:

(ThreadsPerChild) x (stack) x (ServerLimit)

To avoid a very large memory consumption should parameterize the Apache so that parameters do not consume many resources:

```
ThreadLimit 25
ServerLimit 24
StartServers 2
```

```
MaxClients 600
MinSpareThreads 25
MaxSpareThreads 600
ThreadsPerChild 25
MaxRequestsPerChild 0
```

In this case the parameter ServerLimit have fallen from 64 to 24, and adjusted the number of ThreadsPerChild x ServerLimit is equal to the value of MaxClients settings and get a maximum consumption of 6 GB for the Linux server.

Another way to adjust the consumption would decrease the stack memory that has defined the operating system:

- We must run the command ulimit (as we have done above):

  ** For Linux:*

```
$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
stack size              (kbytes, -s) 10240
```

  You can see that in 10240 Kb

It should change the ulimit -S-s *newdate* , for example to a value of 4096 would be:

```
- ulimit -S –s 4096
```

This change can be dangerous as it is changed for all the operating system, so it is recommended to change the ulimit value (eg 4096) in the Apachec apachectl executable by adding an entry form:

```
ULIMIT_STACK_SIZE="ulimit -S –s 4096 "
$ ULIMIT_STACK_SIZE
```

Tomcat Tuning

Connector tuning

maxThreads

- maximum number of concurrent requests
- for BIO, maximum number of open/active connections
- typical values 200 to 800
- 400 is a good starting value
- heavy CPU usage → decrease
- light CPU usage → increase

maxKeepAliveRequests

- typical values 1, 100
- maximum number of HTTP requests per TCP connection
- set to 1 to disable keep alive
- enable for SSL, APR/NIO, layer 7 load balancer

connectionTimeout

- typical value 3000
- default of 20000 is too high for production use
- also used for keep alive time-out
- increase for slow clients
- increase for layer 7 load balancer with connection pool and keep alive on
- decrease for faster time-outs

Content cache tuning

Dynamic content is not cached
Static content is cached
Configured using the <Context .../> element
cacheMaxSize
- 10240

cacheTTL
- 5000

CacheMaxFileSize
- 12
- from 6.0.19 onwards

JVM tuning

Two key areas
- Memory
- Garbage collection

Remember to follow the tuning process

Java heap (Xmx, Xms) is not the same as the process heap

Process heap includes
- Java Heap
- Permanent Generation
- Thread stacks
- Native code
- Directly allocated memory
- Code generation
- Garbage collection
- TCP buffers

Read OutOfMemory exception messages carefully

Xms/-Xmx

- Used to define size of Java heap
- Aim to set as low as possible
- Setting too high can cause wasted memory and long GC cycles
-

-XX:NewSize/-XX:NewRatio

- Set to 25-33% of total Java heap
- Setting too high or too low leads to inefficient GC

Scaling Tomcat

Load balancing

- Routing requests to multiple Tomcat instances
-

Clustering

- Sharing state between Tomcat instances for fail-over

Simplest configuration

- 1 * httpd
- 2 * Tomcat instances
- mod_proxy_http
-

Considerations

- state management
- fail over

Stateless

- Requests routed to Tomcat instances based purely on load balancing algorithm
- HTTP sessions will not work

Adding HTTP session support

- Tomcat instance maintains HTTP session state
- 'Sticky sessions'
- All requests for a session routed to same Tomcat instance

Fail over

- Add session replication - clustering
- Asynchronous by default so usually used with sticky sessions
- Single line configuration for defaults
- Uses multicast for node discovery
- Will need additional configuration for production use

Reference:

http://oreilly.com/catalog/9780596101060/chapter/ch04.pdf
http://tomcat.apache.org/connectors-doc/webserver_howto/apache.html