WHITE PAPER

Apache Tomcat Best Practices

How to Deploy for Lasting Performance, Security, and Resiliency

Executive Summary

In this white paper, we give an overview of the best practices to ensure secure, performant, and resilient Apache Tomcat deployments, including sections on clustering, load balancing, performance, security, and more.

Contents

Tomcat Deployment Best Practices

Running Apache Tomcat as an application container for mission-critical applications is a decision many companies make every day. Unfortunately, this is often where the decision making stops.

Unoptimized Tomcat deployments can suffer from increased hosting overhead, increased risk of data breaches, and even put the applications at risk of catastrophic failures.

An optimized Tomcat deployment features a system configuration that will protect these applications even in the event of catastrophic system failure. This is where applying best practices in Tomcat clustering, load balancing, performance, and security become paramount.

CLUSTERING

In the next sections, we look at some of the best practices in these areas, and how teams can configure Tomcat for lasting stability and success.

By implementing a solid clustering design, you protect your company from systems failure. This section will walk through the core concepts of clustering and why you should cluster your Tomcat application servers. It will look at the multiple options for clustering setups, and help you identify which one is the best for your IT infrastructure. Finally, we will review some detailed example configurations and common issues when clustering Apache Tomcat.

WHAT IS CLUSTERING?

Clustering, when referring to information technology systems, is two or more independent interconnected systems (nodes), interlinked to provide reliability. Reliability can come in the form of high-availability, improved scalability, improved application availability, and ease of maintenance.

Independent interconnected systems sound complicated, although they are not. In the case of this paper, we are referring to Tomcat systems.

An instance of Tomcat is an independent system. Clustering instances of Tomcat makes them interconnected. Tomcat instances in a Tomcat cluster are often referred to as a node. Individual components in any network configuration can normally be referred to as nodes, but for the duration of this paper, we are referring to nodes as Tomcat instances.

A Tomcat cluster is a group of Tomcat instances that are connected. There are different ways that they can be connected. The Tomcat instances can be running on the same physical device, same virtual device, or disparate systems.

WHY CLUSTER YOUR TOMCAT DEPLOYMENT?

Clustering can solve different problems. For instance, you have a web application, serving approximately five thousand concurrent requests, running on your server. Under this load, your single server is maxed out. New users are receiving 404 errors. Supporting larger numbers of concurrent requests is one of the advantages of high-availability clustering. The goal of high-availability clustering is 99.999 percent (“five nines”).

Tomcat clustering is also useful for engineering failovers. If your business is running a web application that earns income for your business and this web application is running in a non-clustered environment, you are at risk. If your application is on a Tomcat server that is not clustered and the Tomcat server fails, that source of revenue stops generating money every second the system is down. By setting up a simple Tomcat cluster containing two instances, this issue is preventable. In a properly configured cluster, all requests to the failed server will be directed to the remaining working instance. This will preserve your revenue stream even if there is performance degradation from losing 50 percent of the nodes in the cluster.

These are just some examples of why it pays off to cluster Tomcat, or at least research a little more. In addition to these examples above, Tomcat improves your systems availability. High-availability is a goal that many companies seek to improve the appearance and availability of their services.

A normal system's yearly average uptime is called its availability. High-availability is a pre-arranged, contracted level of performance that will be maintained during the contract length. Granted, that is not very easy to understand. An example of high-availability could be: your web server is guaranteed to be available "five nines." This means that in a given year the server will have a maximum of 5.26 minutes of unscheduled downtime.

To achieve high-availability you need to implement geographic separation. Geographic separation, as it applies to server configuration, is installing nodes of the cluster in geographically different locations. This provides safety against regional power outages and other locational risks like storms and floods.

FINDING THE BEST CLUSTERING APPROACH FOR YOUR ARCHITECTURE

Everyone wants to build a reliable, stable, and available application container platform. But, in order to do so you need to determine which clustering configuration fits you and your business the best.

In determining your configuration, you must evaluate the resources at hand. This section will discuss possible options for your resources, without actually taking your resources into consideration. The next section will make suggestions as to which configurations your company may leverage depending on the resources available.

VERTICAL CLUSTERING

A vertical cluster expands vertically. A horizontal cluster expands, you guessed it, horizontally. What does this mean? A vertically expanding cluster has a limited horizontal layout. A horizontal layout would consist of multiple systems and/or resources.

A vertical cluster is on a single machine. A machine can be many things, including a physical device or a virtual host. As need increases, Tomcat instances are spawned on the same machine, using configuration tweaks that allow multiple instances to run on the same system.

HORIZONTAL CLUSTERING

A horizontal cluster contains Tomcat instances running on separate machines. If demand for processing increases and you had a pure, horizontal cluster configuration, the network technician (or you) would install a new machine, virtual or physical, and on that machine is a new Tomcat instance.

HYBRID CLUSTERING

Real life is often very different from dictate. Companies rarely have a pure horizontal or vertical cluster configuration. Most systems are hybrids. A hybrid cluster is a mixture of vertical and horizontal clustering to facilitate a specific need and/or to match the hardware provided.

HOMOGENOUS VS. HETEROGENOUS CLUSTERING

Is your setup going to be for multiple applications, or just a few, or just one? Do you have applications that require specific hardware? This determines whether or not you decide to use a heterogeneous or homogeneous setup.

A homogeneous setup is very common. Companies will often duplicate their Tomcat environment, launching servers on many devices with a simple copy of the Tomcat directory. A Tomcat cluster that has the same web applications deployed on all nodes is considered homogeneous.

Homogeneous setups can be hard to keep truly identical. Sometimes, especially after node failure and replacement, it can be hard to synchronize the Tomcat instances. The best way to do this is to create an image of the Tomcat setup from a node designated as the primary node. As long as this image stays up to date you can distribute it over as many Tomcat setups as you prefer.

Heterogenous setups, on the other hand, can have different web applications deployed on different nodes of a cluster. With a handful of applications, heterogenous clustering can be fairly simple. When an organization has a large number of unique web applications, setting up, configuring, and maintaining that cluster can be complicated.

LOAD BALANCING

Load balancing happens outside of the Tomcat cluster but is still an important consideration for teams deploying Tomcat. For the purposes of this document, we are concerned with Apache Httpd server and the built-in load balancing/gateway features, as this is a free and common solution within many enterprise systems. However, the open source version of NGINX — a common and free load balancing option — can be implemented in similar fashion to Apache HTTPD.

APACHE HTTPD

When load balancing with Apache HTTPD the main consideration is which protocol will be used. The two options to consider here are either AJP or HTTP(s). The most common Tomcat configuration that we see is Tomcat running behind some sort of Web based proxy like Apache HTTPD or NGINX. However, for decades the only option to do this was to use AJP, so a lot of current AJP usage is based on industry inertia more than anything else.

There are definitely some security (no TLS/SSL support) and support (no HTTP 1.2 support) considerations that need to be made here, but from a performance perspective the fact that AJP is a binary protocol means it can provide some performance benefits.

In most environments, it is not nearly as important as it used to be. Back when 10/100 ethernet was the norm and bandwidth came at a premium, using a binary protocol over a text-based protocol like HTTP(s) was a no brainer.

Today, where gigabyte ethernet is the standard for any network backplane, the binary aspect of AJP has become a lot less important. Bandwidth constrained environments still exist, so it is common to see small Tomcat environments running on remote IoT installations. In such cases choosing a binary based protocol like AJP in spite of its security and support issues might make sense.

Regardless of the method, we want to make sure we are tuning Tomcat to handle the number of connections that will be thrown its way — whether that's from AJP or HTTP(s). If your web proxy is configured to handle 1000 connections but your Tomcat server is only configured to handle 500, you are in for a bad time.

After deciding which protocol to use, the next decision is which Apache HTTPD module to use. If AJP is being used then it would be mod_jk or mod_proxy_ajp. If the decision was made to utilize HTTP(s) then mod_proxy and mod_proxy_balacer would be used.

mod_proxy/mod_proxy_balancer	
Pros	**Cons**
Required if secure communication over HTTPS is needed	Limited load balancing configurations, only basic load balancing available
Can still be uses with the AJP protocol (mod_proxy_ajp)	Does not support domain- based clustering (introduced in AJP 1.2.8)
No need to compile a separate module, comes default with Apache 2.2 and higher	

mod_jk	
Pros	**Cons**
Support for more advanced load balancing options	Insecure protocol, no support for TLS. AJP is a binary protocol, but it is not encrypted, even when the shared secret is configured
Better node failure detections	Has to be built and maintained separately from Apache HTTPD)
Support large AJP packets	

OTHER LOAD BALANCING OPTIONS

One also might consider whether or not your environments even need a proxy service at all. The Coyote web server that ships with Tomcat is a more than capable HTTP server and can handle serving up web content just fine. This would be a common configuration found when enterprises implement hardware based load balancer like an F5 LTM.

HARDWARE LOAD BALANCING

Another common enterprise configuration option for load balancing is the hardware load balancer. A hardware load balancer (HLB) performs the same tasks as a software balancer (like the one in the Apache Httpd server). The main difference between a software balancer and a hardware balance (besides price), is resources.

An HLB has dedicated hardware resources (RAM), processor, network adapters, etc. This allows hardware balancers to perform at a much more efficient rate, while providing more features. This is also an infinitely more expensive method, as you can find many free open source load balancing solutions. In most cases, these software-based open source solutions are perfectly acceptable. However, some use cases may require a dedicated hardware solution. Applications that see sustained high usage, (e.g., with concurrent users in the 100,000+ range) will most likely want to consider a hardware based solution.

In addition to ultra-high usage profiles, applications that require global load balancing would need to consider this type of hardware-based solution as well. Though in cases where ultra-high usage performance is not a requirement, but global load balancing services are still needed, paid software based load balancing services such as NGINX-Plus are available.

PERFORMANCE

Most Tomcat performance tuning is specific to the application and takes place in setting up the right JVM options and sizing. Making sure you have an optimal system environment, the appropriate amount of compute cycles available, enough memory to support the JVM and system OS overhead, enough disk and enough I/O throughput to support the given application) is a must. To properly tune your Tomcat environment, here are some best practices to make sure you understand the needs of the application Tomcat will be running.

THE IMPORTANCE OF PROFILING YOUR APPLICATION

The first place to start with any performance tuning on any platform is with the application itself. If you don't profile your application, you can't answer questions like:

- Do you know how much time your web application spends in garbage collection?
- Do you know how many threads your app uses?
- Do you know what its longest running query is?
- How many disk operations does your app perform at any given time?
- Do you know the answers to these questions when you have 10 concurrent users? How about 100? 1000?

Profiling your web application is a must! It also happens to be one of the most overlooked aspects of application development we run into.

When we start a Tomcat professional services engagement and ask if they know what their average garbage collection times are under load, the answer is all too often "no". In way too many cases, garbage collection logs are not even being collected.

Building an application profile does not only have to occur when doing load/performance testing (though it should; more on that later). We can build a real-world application profile over time by making sure we are collecting the right data and metrics.

By making sure we are collecting garbage collection logs, capturing regular thread dumps, performing access logging, gathering database call times, etc., we can make sure we have the data to review over time to get a better understanding of how applications are running and what they look like in a real-world scenario.

Of course, there is not much value in collecting this data if no one is looking at it. A review of an application's performance profile should be baked into every release cycle.

AUTOMATE YOUR LOAD AND STRESS TESTING

As mentioned in the previous section, you don't have to do load testing to build an accurate application profile, but you should.

In a perfect world, we would have a 1 to 1 replica of our production environment to perform load and stress testing, but rarely do we find this to be the case. Granted it is a lot easier to spin up a production like environment these days compared to 20 years ago when we were doing this all on bare metal, but you don't have to have a perfect production replica to perform worthwhile load and performance stress testing.

Taking a scaled-down model of your production environment and pushing it to its breaking point with a utility like JMeter will still provide your team with invaluable information and create a baseline that the rest of your performance tuning can be based upon.

Wondering what your JVM memory sizing should be? Wondering if your application would benefit from enabling compressibleMimeType? Load and performance testing will tell you.

We all know how tempting it is (and how often it happens) to skip load and performance testing, but just like the previous tip, this should be baked into every release cycle. With modern CI/CD tools like Jenkins and GitLab, automating your load and stress testing to have them run in your CI/CD pipeline is easier than ever.

NEEDS BASED TUNING

Most Tomcat tuning is "needs" based tuning; in other words, we are tailoring the Tomcat configuration to the given application it is running. Now that we know what our application and performance profiles look like we can start building out environments to match these profiles. Does the application churn CPU? Does it need a ton of memory for a large JVM? Does it have a lot of disk I/O? Now that we know these things, we know how to build out the compute environment in a way that matches the needs of the application and the projected users that are going to be using the system.

We can also start making fact-based decisions on specific Tomcat settings like number of threads, max keepalive times etc. Does your application have a lot of long running queries? If so, then we may need to set longer keepalive timeouts than we normally would. Or, on the other end of the spectrum, does the application perform a lot of short duration queries but fire off a lot of them at one time? In that case we would want to consider short "keepalive" timeouts and a larger number of connections in the connection pool.

OTHER PERFORMANCE CONSIDERATIONS

Be careful to not over-tune your environment. While not as common, we do run into environments that have way too many unnecessary Java arguments, or they are set incorrectly, and the default values would have provided better performance to begin with.

For the naturally curious, it is tempting to tinker with your environments — and that is good, but just not in production. We want to be sure we are using fact-based evidence when deciding which parameters or Java arguments we are introducing into our productions environments, and not just playing the guessing game.

Also, when considering system resources, the 70/30 rule is always a good general rule to follow. Hitting 70% system utilization while reserving 30% system free resources insure you always have breathing room for unexpected spikes in utilization. While not a fixed rule, when an environment starts seeing consistent utilization of resources sitting in the 80% plus range it is a good idea to start planning to add additional resources. When system utilization is consistently sitting in the 90% plus range that plan should be executed sooner rather than later.

SECURITY BEST PRACTICES

Following known security best practices for any piece of software is important. And, as we show below, many of the basic security best practices for Tomcat are ones you can apply to other pieces of software within your web applications.

That said, Tomcat does carry some default exposures that need to be corrected on new deployments. The best practices listed below, while far from comprehensive, are a good place to start when deploying Tomcat.

DO NOT RUN TOMCAT AS THE ROOT USER

One of the most basic security best practices for Tomcat is to not run Tomcat as the root user. Creating a user with minimum OS permissions and running the Tomcat server as that user should be the first thing you do.

REMOVE DEFAULT SAMPLES AND TEST APPLICATIONS

Tomcat also comes with some default samples and test applications. These samples are known to contain some vulnerabilities of their own and should be removed from your environment.

SET YOUR TOMCAT PERMISSIONS CAREFULLY

Tomcat itself should be set to only have the necessary permissions, should your server ever be hijacked.

DISABLE SUPPORT FOR TRACE REQUESTS

Disabling support for TRACE requests prevents browsers from being exposed to a cross-site scripting attack. To prevent information about your Tomcat server from being broadcast, you will want to disable the X-Powered-By HTTP header. This header broadcasts information such as what version of Tomcat you are running and other sensitive information. This can be disabled in the server.xml file.

DISABLE SSLV3 PROTOCOLS

POODLE was a well-publicized attack that targeted the SSLv3 protocols, so you will need to disable that in Tomcat before you get it up and running. Maintaining detailed logs is also key to ensuring your Tomcat server and environment security. This applies to user access, application traffic, Tomcat internals, the OS/firewall, etc.

LOG YOUR NETWORK TRAFFIC

To enable logging of network traffic in Tomcat, use the AccessLogValve component. This can be configured on a host, engine, or context basis and will create a standard web server log file for traffic to any resources associated with it. The Access Log Valve supports a variety of attributes to control the output of the valve.

BE CAREFUL WITH THE TOMCAT MANAGER APP

The Tomcat Manager app is a built in webapp used to manage Tomcat instances, application deployment, and other various settings. For security purposes this console is disabled by default, so if you enable it, be sure you treat it appropriately.

USE REALMS TO CONTROL RESOURCE ACCESS

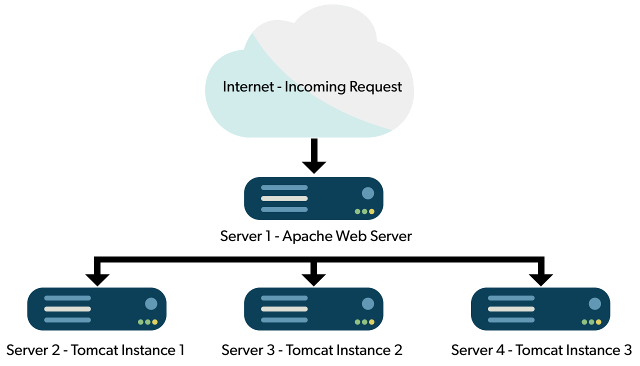
Realms are another method of controlling access to resources in Tomcat. Realms are components that access databases of users that should have access to a given application or group of apps, and the roles and privileges they have within the application once logged in. The most secure of the realms is the LockOut realm which places a limit on the number of times a user can attempt to authenticate themselves.

Applied Use Cases

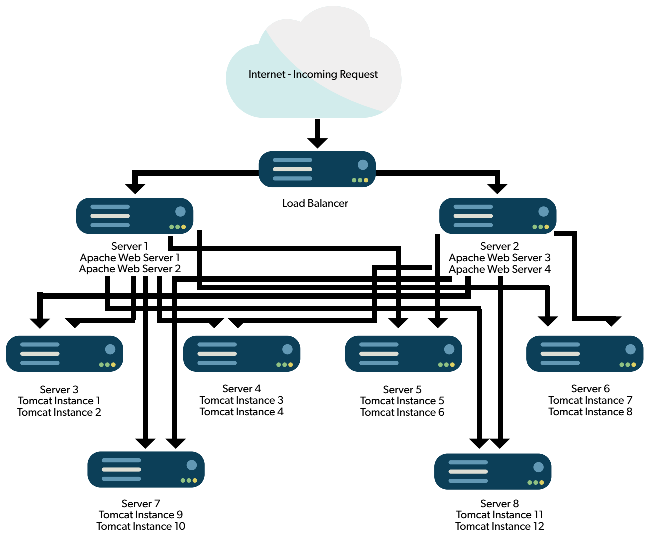
Tomcat is used successfully in many types of web applications, with different deployment patterns able to meet needs for many enterprise use cases. In the sections below, we look at two of those use cases, including horizontal scaling and enterprises managing several applications.

USE CASE #1: SCALING

In this use case, the client has 4 low-end servers, meaning they have one processor with 1- 4 gigabytes of RAM. This would be an ideal situation for a horizontal cluster. Each member of the cluster would be able to run one instance of Tomcat efficiently. One of the servers could be used as a balancer running Apache Httpd server.


Internet - Incoming Request
Server 1 - Apache Web Server
Server 2 - Tomcat Instance 1
Server 3 - Tomcat Instance 2
Server 4 - Tomcat Instance 3

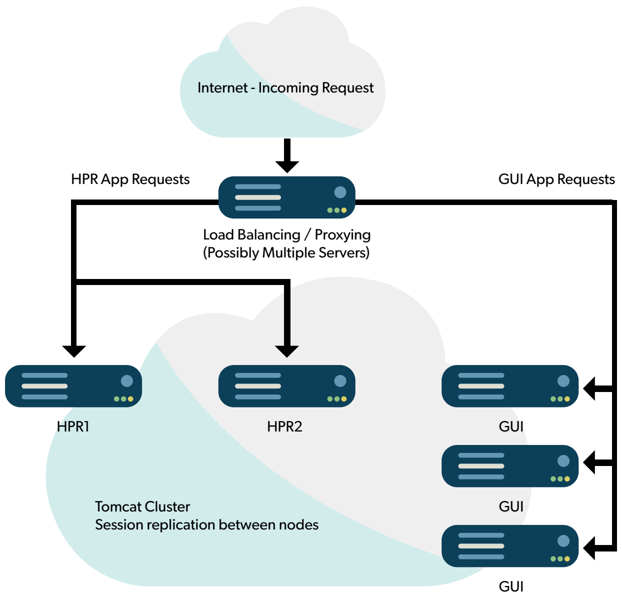
If your situation was a bit different, and you had better servers, you could consider a hybrid cluster. If there are servers available with two or more processors and a large amount of ram (8 gigabytes or more) this would be ideal for multiple Tomcat instances. In this configuration you can set up a hybrid cluster by running multiple instances of Tomcat on multiple machines, and multiple instances of Apache Httpd to handle the load of load balancing. This configuration could look something like this:



USE CASE #2: MULTIPLE APPLICATIONS

Tomcat can also accommodate organizations that want to deploy multiple applications. How the applications are divided into Tomcat nodes is up to the user. However, the configuration of these nodes in Tomcat will be a little more complicated.

If an organization has an application that requires heavy processing and substantial amounts of RAM (HPR1,) you can set up this application on two nodes by itself. After this, take the remaining applications (GUI) and place them on two different nodes in the cluster. This will prevent the GUI application from being bogged down when HPR1 is consuming the CPU and RAM. This cluster might look like this:


Internet - Incoming Request
HPR App Requests
GUI App Requests
Load Balancing / Proxying (Possibly Multiple Servers)
HPR1
HPR2
GUI
GUI
GUI
Tomcat Cluster Session replication between nodes

There are many things to take into consideration when designing and building your cluster. If a large company is relying on you to provide a reliable, highly-available application implementation, then clustering and load balancing is the right choice. Regardless of if you are new to clustering, or an old hand, purchase a support contract. There are companies that will provide open source software support for your Tomcat and Apache Httpd configuration. This will allow you to offer your customers an extremely reliable, available service while at the same time providing someone to turn to if you run into problems.

Cluster Setup and Configuration Overview

This is not a complete step-by-step tutorial on cluster creation, but we will provide you with the tools you can use to implement a cluster rapidly and effectively. Whether you have created many clusters in the past or this is your first attempt, we hope that you will be able to learn something, whether it be basic or advanced, from the information discussed in this paper. To limit the liability of your attempt at creating a cluster, you can set up a machine, virtual or physical, just for this task.

Please note that you can run multiple Tomcat instances on a single virtual/physical machine by tweaking just a few settings, mainly port numbers so the instances don't interfere with each other. The configuration of these Tomcat instances is well outside the scope of this document, although it is not difficult to accomplish.

Below are two server configurations that you can use to run a simple cluster, just start with two instances of Tomcat 10, and replace the corresponding server.xml file with the .xml information provided below.

SERVER.XML 1

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="50005" shutdown="SHUTDOWN">
 <Listener className="org.apache.catalina.startup.VersionLoggerListener" />
 <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
 <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
 <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
 <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />
<GlobalNamingResources>
 <Resource name="UserDatabase" auth="Container"
 type="org.apache.catalina.UserDatabase"
 description="User database that can be updated and saved"
 factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
 pathname="conf/tomcat-users.xml" />
 </GlobalNamingResources>
<Service name="Catalina">
 <Connector port="51112" protocol="HTTP/1.1"}
 connectionTimeout="20000"
 redirectPort="51114 " />
<Engine name="Catalina" defaultHost="localhost">
 <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
 <Realm className="org.apache.catalina.realm.LockOutRealm">
 <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
 resourceName="UserDatabase"/>
 </Realm>
 <Host name="localhost" appBase="webapps"
 unpackWARs="true" autoDeploy="true">
 <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
 prefix="localhost_access_log" suffix=".txt"
 pattern="%h %l %u %t &quot;%r&quot; %s %b" />
 </Host>
 </Engine>
 </Service>
</Server>
```

SERVER.XML 2

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="50006" shutdown="SHUTDOWN">
 <Listener className="org.apache.catalina.startup.VersionLoggerListener" />
 <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
 <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
 <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
 <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />
 <GlobalNamingResources>
 <Resource name="UserDatabase" auth="Container"
 type="org.apache.catalina.UserDatabase"
 description="User database that can be updated and saved"
 factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
 pathname="conf/tomcat-users.xml" />
 </GlobalNamingResources>
 <Service name="Catalina">
 <Connector port="51112" protocol="HTTP/1.1"
 connectionTimeout="20000"
 redirectPort="51114 " />
 <Engine name="Catalina" defaultHost="localhost">
 <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
 <Realm className="org.apache.catalina.realm.LockOutRealm">
 <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
 resourceName="UserDatabase"/>
 </Realm>
 <Host name="localhost" appBase="webapps"
 unpackWARs="true" autoDeploy="true">
 <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
 prefix="localhost_access_log" suffix=".txt"
 pattern="%h %l %u %t &quot;%r&quot; %s %b" />
 </Host>
 </Engine>
 </Service>
</Server>
```

CREATING A TOMCAT CLUSTER

Tomcat clustering is quite simple to set up. However, if you wish to leverage clustering in your enterprise environment the default configuration is not going to be the best route for you. To turn on clustering in your Tomcat server all you must do is add one line of code to your server.xml.

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

Adding this line to your configuration enables clustering with all of the default settings. This would be great if you were not in an enterprise setting.

You have created a clustered Tomcat instance, but you only have one instance, so it is not a very big cluster. Before we create the next instance, we should install the application we want to test on this cluster.

MAKING YOUR WEB APPLICATION DISTRIBUTABLE

With your cluster running, placing a normal application on one server will not trigger propagation to other servers. The idea behind propagation is that an application is placed on one node in the cluster, it is migrated (copied) automatically to other nodes in the cluster. To achieve this we add the following code to the web.xml:

```
<distributable/>
```

This tells Tomcat that this application is designed to run on multiple nodes in this cluster.

SETTING UP SESSION REPLICATION

The default session replication mode is "All to All," meaning any session data created on a server will be duplicated to all other servers in the cluster. If your application creates session data for a user, and you have a heterogeneous cluster, the session data will still be replicated across the other nodes.

A heterogeneous configuration is one that does not have all of the same applications on every node. Therefore, if application A stores session data for a user, and application A is running on server A, but not server B, session data will replicate to server B, even though there is no use for it there.

CONFIGURING MULTICAST SETUP

The cluster is discovered and maintained via multicast heartbeats. The server will be set up with a default multicast IP address of 228.0.0.4 and a multicast port of 45564. This means that any other nodes that are using the same multicast address and port will see this cluster/node. It is important to ensure your network supports multicast. This is commonly blocked for security reasons.

ADDITIONAL CONFIGURATION STEPS

After creating the cluster object and making your web applications distributable, we need to move on to configuring other settings.

THE MANAGER OBJECT

The Manager object controls session replication.

```
<Manager
 className="org.apache.catalina.ha.session.
DeltaManager"…/>
```

The DeltaManager replicates all changed session data to all nodes of the cluster. The BackupManager backs up session data to a specific backup node. For large clusters the BackupManager is the option to go with, for smaller clusters it is common to just use the default DeltaManager.

In Tomcat 5, you could not choose the specific session manager for your application. In Tomcat 10, you can define a manager in the cluster configuration, as you could in earlier versions, but you can also define a manager in a web application's context.

Defining the Manager in your clustering configuration provides a default setting for applications that do not provide their own Manager configuration. For instance, the following code will set all applications in your cluster to use the BackupManager for session replication.

```
<Manager
 className="org.apache.catalina.ha.session.
BackupManager".../>
```

CHANNEL SEND OPTIONS

After setting the Manager, you might need to apply a non-default channel send options value. Channel send options is a setting specified on the cluster object. For example:

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

Channel send options control how messages are sent between cluster nodes. Are these messages sent synchronously? Or, in basic terms, does the thread that sends the message need to wait until the message has sent before continuing to work, in turn, potentially making the users request wait on this message to be sent? Sending the messages asynchronously is when the thread generates and sends the message but does not stop and wait for this to happen. Instead, it does this by spawning a worker thread.

As you can see, this is just one aspect of the channel send options, and it is a lot of information. To go over channel send options in detail will require a whole default channel send mode that is asynchronous.

WRAPPING UP

Tomcat clustering is a powerful tool that can provide the high availability, reliability, and dependability that your company requires and all of it can be set up with little effort. That said, this paper barely scratches the surface of clustering. What we have provided is a starting point for your cluster. With the information provided here you can start a cluster containing two or one thousand nodes, it is just a matter of determining your company's needs.

That said, clustering adds complexity. Teams should expect minor problems along the way, like nodes not joining a cluster, session information being lost, random node crashes, and configuration issues.

Final Thoughts

From clustering and load balancing to performance and security, teams that put in the time to optimize their Tomcat deployments will reap the rewards of secure and performant web applications. But it is also important to note that optimizing Tomcat is not a one-time exercise. Ensuring your Tomcat deployments are kept up to date and configured to match the changing needs of your application(s), is an ongoing and often-overlooked commitment.

While we went over many of the basic best practices for Tomcat in this paper, there is a lot of ground we left untouched. Ultimately each web application will have its own needs, and each Tomcat deployment will need to be configured and maintained to support those needs in different ways.

Finding Dependable Technical Support for Your Tomcat Deployments

Whether you are planning, deploying, or even supporting a legacy Tomcat deployment, OpenLogic can provide the dependable technical support you need to find success. Learn more about what we can offer your team by visiting our Tomcat Support and Services page today.

SEE WHAT WE OFFER