5. Application Server Administration

Miquel Àngel París i Peñaranda

Web Application Deployment

2nd C-VET Web Application Development

Index of contents

1. Goals.......... 3

Application server configuration and management.......... 4

1.- Application server protection.......... 6

2.- Deployment of applications in Tomcat.......... 8

2.1. Creation of a web application.......... 10

2.2. Deployment of a web application.......... 12

2.3. Implement access registration.......... 14

2.4. Persistent sessions.......... 16

2.5. Configure Tomcat in cluster.......... 17

3. WildFly Application Server.......... 20

3.1. Getting Started with WildFly 35.......... 21

3.2. Getting Started with WildFly.......... 25

How To Install Apache Tomcat 10 on Ubuntu Server 22.04.......... 28

Web Links.......... 36

1. Goals.

1. Install and configure web servers on virtual machines and/or the cloud.
2. Perform functional tests on web and application servers.
3. Document the installation and configuration processes performed on web and application servers.

Application server configuration and management.

Case Study

At the company BK programming, Ada, together with her employees Juan and María has met to evaluate the possibility of configuring one or two application servers to install demos, or beta versions (also called "betatest", indicates a period in which a software is technically finished, which means that no more functions will be added to it for the time being, and presumably it will be stable enough to work normally. In contrast, the alpha version, the version prior to the beta, is more unstable and not complete), of the applications they develop, so that customers, or potential customers, could try BK programming products before purchasing them.

As a result of this meeting, they have concluded that, prior to the installation and commissioning of application servers, it would be very important to evaluate many parameters that would affect the correct operation of the servers, in addition to their needs.

Among the parameters to be evaluated, the following should be highlighted:

- Application server security: security measures to be applied to prevent possible attacks or intrusions.
- Server sizing where the physical needs of the server equipment are studied.
- Type of server to be installed, specific features of the selected server software (Tomcat, Jboss, etc.).
- Deployment of applications on the server where it would be necessary to establish which tools should be used.
- Managing remote connections to servers.
- Server scalability, to be taken into account depending on the number of simultaneous connections that can be established.
- On-server task automation tools (Ant, etc.).

Due to the number of parameters that must be managed to put the application servers in good operation, Ada has decided that its employees should document each and every one of them and, if possible, the possibility of taking a training course on the administration of application servers.

1.- Application server protection.

Case Study

One of the first concerns that server administrators encounter is the security and protection of the same against possible attacks or uncontrolled access, for this reason, María has started to investigate the options to configure, and tools to use, to block possible vulnerabilities of web servers along with security problems in web applications.

An application server is usually software that provides a series of application services to an indeterminate number of client computers that access these services via the web; The main advantages of this type of technology is the centralization and reduction of complexity in the development of applications, however web applications are thus more exposed to attacks.

Nowadays there are web applications for almost everything and that have access to very valuable information such as, for example, credit card numbers, bank accounts, medical records, personal information, etc. Therefore, they represent an interesting target to attack; These attacks can be classified based on three levels:

- Attacks on the user's computer (client).
- Server attacks.
- Attacks on the flow of information that is transmitted between client and server.

In each of the above levels, it is necessary to guarantee a minimum security to achieve the safety of the entire process. At the user level, they must have secure browsers and platforms, free of viruses; At the server level, it must be ensured that data is not modified without authorization (integrity) and that it is only distributed to authorized persons (access control) and, as far as the transit of information is concerned, it must not be read (confidentiality), modified or destroyed by third parties, while at the same time ensuring a reliable communication channel that is not interrupted relatively easily.

To achieve secure web applications, mechanisms must be established to guarantee:

- **Authentication**: allows you to identify, at all times, who the user who is accessing it is. There are several methods to achieve this:

  - Basic authentication: user and password request.
  - Authentication with certificates.
    - `HTTP DIGEST AUTH` (HTTP Digest Authentication).
    - `HTTP NTLM AUTH` (HTTP Autentication Microsoft NT Lan Manager).
- **Authorization**: allows you to determine which data and modules of the application the user can access, once authenticated.
- **Ticket validation**, as the validation code can be manipulated on the client-side.
- **SQL command injection**: A technique for exploiting web applications that do not validate client-supplied information to generate dangerous SQL queries.

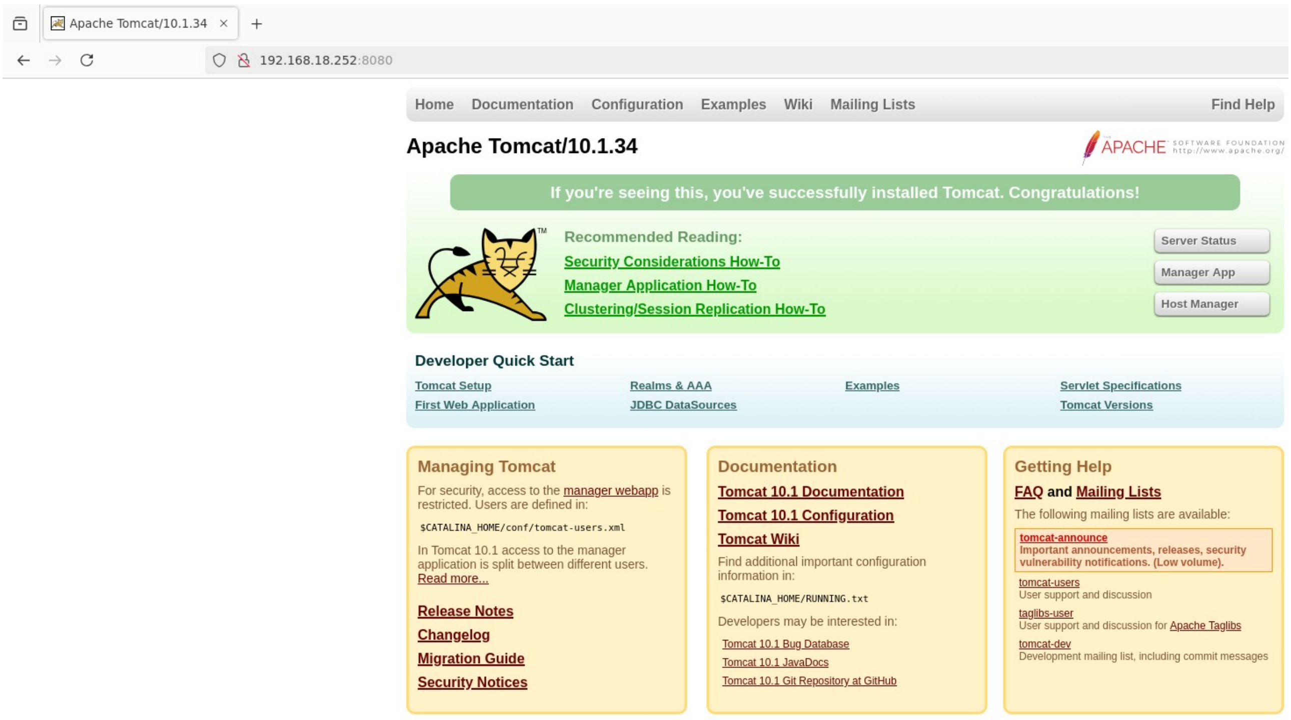
To achieve secure web applications, a series of mechanisms and tools must be used, among which we highlight:

- Disabling unused accounts and services.
- Updating the operating system and applications (**patches** (When applied in conjunction with software, it is a set of files in addition to the original software of a tool or computer program. They are usually used to solve a possible shortcoming, vulnerability, or malfunctioning)).
- Strength in passwords.
- Use of Firewalls.
- Periodic back-ups.
- Periodic log analysis (official log of events during a particular time range. For computer security professionals, it is used to record data or information about who, what, when, where, and why an event occurs for a particular device or application.
- Periodic verification of active services.
- Traffic encryption.
- Establishment of security policies.

2.- Deployment of applications in Tomcat.

Case Study

María has set up a Ubuntu 22.04 server machine with the Tomcat application server so that the members of BK programming can deploy, on that server, the web applications they consider necessary. Juan has carried out a first practice of deploying web applications and has documented each and every one of the steps that must be carried out so that the web application is fully operational on the server, so that any customer of the company can enjoy the functionality of the application.



Apache Tomcat/10.1.34

If you're seeing this, you've successfully installed Tomcat. Congratulations!

Recommended Reading:
Security Considerations How-To
Manager Application How-To
Clustering/Session Replication How-To

Developer Quick Start
Tomcat Setup
First Web Application
Realms & AAA
JDBC DataSources
Examples
Servlet Specifications
Tomcat Versions

Managing Tomcat
For security, access to the manager webapp is restricted. Users are defined in:

```
$CATALINA_HOME/conf/tomcat-users.xml
```

In Tomcat 10.1 access to the manager application is split between different users. Read more...

Release Notes
Changelog
Migration Guide
Security Notices

Documentation
Tomcat 10.1 Documentation
Tomcat 10.1 Configuration
Tomcat Wiki
Find additional important configuration information in:

```
$CATALINA_HOME/RUNNING.txt
```

Developers may be interested in:
Tomcat 10.1 Bug Database
Tomcat 10.1 JavaDocs
Tomcat 10.1 Git Repository at GitHub

Getting Help
FAQ and Mailing Lists
The following mailing lists are available:
tomcat-announce
Important announcements, releases, security vulnerability notifications. (Low volume).
tomcat-users
User support and discussion
taglibs-user
User support and discussion for Apache Taglibs
tomcat-dev
Development mailing list, including commit messages

Deploying a servlet consists of placing a series of files in a web container so that clients can access its functionality; A web application is a set of servlets, HTML pages, JSPs, classes, and other resources that can be packaged in a certain way.

A web application can be deployed in Different Servers web maintaining his functionality and No modification to your code due to the Servlet 2.2 specification. Web applications should be organized according to the following directory structure:

- **Main (root) directory** : It will contain static files (HTML, images, etc...) and JSPs.

- `WEB-INF` folder: contains the "`web.xml`" file (application descriptor), responsible for configuring the application.
  - Subfolder `classes`: contains the compiled files (servlets, beans).
  - Subfolder `lib`: additional libraries.
- Other folders for static files.

A web application can be deployed using one of the following methods:

- Through `WAR` files.
- Editing the files `web.xml` and `server.xml`, this method is the one discussed below.
- The directories that make up a compiled application are usually: `www`, `bin`, `src`, `tomcat`, `gwt-cache.`

The `www` folder also contains a folder, with the name and path of the project, which contains the files that make up the interface (HTML, js, css...). The `bin` folder contains the java classes of the application.

To deploy the application to Tomcat, the following steps must be performed:

1. Copy the folder contained in de `www` (with the name of the project) in the Tomcat directory `webApps`
2. .Rename the new folder thus created in Tomcat with a simpler name. That will be the application folder in Tomcat.
3. Create, within this folder, a new one, and give it the name `WEB-INF` (respecting the capital letters).
4. Create, within `WEB-INF`, two other subdirectories, called `lib` and `classes`
5. Copy to `lib` all libraries (`.jar`) that the application needs for its operation.
6. Copy the contents of the `bin` folder of the application in the Tomcat subdirectory `WEB-INF/classes`

7. Create in `WEB-INF` a text file called `web.xml`, with the paths of the servlets used in the application.
8. The application can now be accessed on the server, the way to do it is by entering the path of the input HTML file in the browser, which will be located in the application folder in Tomcat.

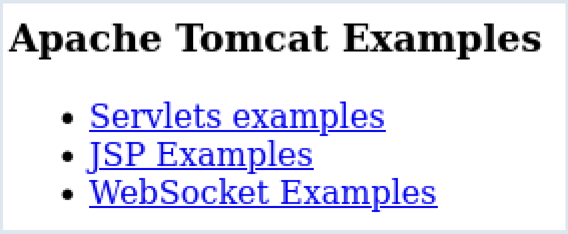
Let's start from our machine with the Ubuntu Server operating system in which we have the Tomcat server running to show the process of creating and deploying applications. Because we intend to set up a `LAMP` platform, due to its advantages derived from the characteristics of free software, we will also install the following components: `MySQL` and `PHP`.

Let's remember, first of all, that in order to install any version of Tomcat it is necessary to have JDK (Java Development Kit) installed, since the objective is that requests to Apache are redirected to Tomcat using a connector provided by Java in this case.

2.1. Creation of a web application.

Case Study

In the company BK programming, Juan has decided to document the most useful and simple methods to follow for the creation of a web application, so that it can be deployed without any difficulty on the Tomcat application server that María has assembled. In this way, customers will have all the functionalities of the applications developed in the company available.


Apache Tomcat Examples

- Servlets examples
- JSP Examples
- WebSocket Examples

The Tomcat application server has a number of examples, both servlets and JSP, that help you learn how to perform the tasks of building and deploying web applications.

It is very interesting to create two environment variables: `JAVA_HOME` one that indicates the location of the Java binary files and `CATALINA_HOME` that points to the location of the scripts (command file or batch command file processing, is a usually simple program, usually stored in a plain text file) from Tomcat, to do this, we can add the following code to the `/etc/profile`.

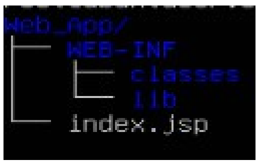
```
CATALINA_HOME=/usr/local/apache-Tomcat-6.0.32/
JAVA_HOME=/usr/lib/jvm/java-6-openjdk/jre/
PATH=$PATH:$JAVA_HOME/bin:$CATALINA_HOME
export PATH JAVA_HOME CATALINA_HOME
```

We update the environment variables using the command:

```
source /etc/profile
```

The Javascript language runs on the client side, it is an interpreted scripting language that does not allow access to local information of the client nor can it connect to other network computers.

First of all we will create a folder with the name that we are interested in to identify the application, in this example we have opted for `Web_App` a structure like the one in the following image:



```
Web_App/
├── WEB-INF
│   ├── classes
│   └── lib
└── index.jsp
```

The application that we intend to develop contains a file that we will call `index.jsp` very simple with the following content:

```
<html>
     <head><title>WEB APPLICATION DEVELOPMENT</title>
          <script language="Javascript">
               function popup(){
                    alert("CONFIGURATION AND ADMINISTRATION OF
                    APPLICATION SERVERS");
               }
          </script>
     </head>
     <body>
          <h1 align=center>WEB APPLICATION DEPLOYMENT</h1>
          <div align=center>
               <form>
                    <input   type="button"   value="UNIT   5"
                    onclick="popup()">
               </form>
          </div>
     </body>
```

```
</html>
```

Finally, we would only have to make a copy of the folder of our application in **`$CATALINA_HOME/webapps`**. Later, from a browser, we access http://192.168.18.252:8080/Web_App we would have the application working.

If the computer on which we have developed the previous application, and where it has been put to work, belongs to a computer network and has the IP: 192.168.10.1. Could we access the web application from other computers? If so, what would be the URL we should type?

2.2. Deployment of a web application.

One of the objectives pursued when developing web applications is that they can be deployed on different web servers, maintaining their functionality and without any code modification.

WARs are simply Java files from a web application with a different extension to differentiate them from the commonly used JARs.

Prior to the Servlet 2.2 specification, it was quite different to deploy servlets between different servlet containers, previously also called servlet engines. Specification 2.2 standardized deployment across containers, taking Java code portability a step further.

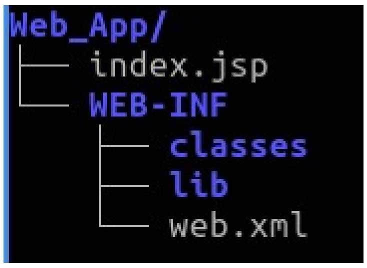
The simplest method to deploy an application, which is mainly used during its development stage, is the one done in the previous point, that is, copying the folder corresponding to our application in the folder `$CATALINA_HOME/webapps`, taking into account that the variable `$CATALINA_HOME` is the path of the scripts used by Tomcat.

Continuing with the application developed in the previous point (`Web_App`), we are going to create a deployment descriptor file **`web.xml` that is responsible for describing the deployment characteristics of the application**.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
     version="2.5">
```

```
        <display-name>Descriptor Application Web_App</display-
        name>
        <description>
            My first descriptor web.xml.
        </description>
    </web-app>
```

This file will be placed in the folder `WEB-INF` belonging to the application under development, so that the structure of the resulting folder would be the one shown in this image:



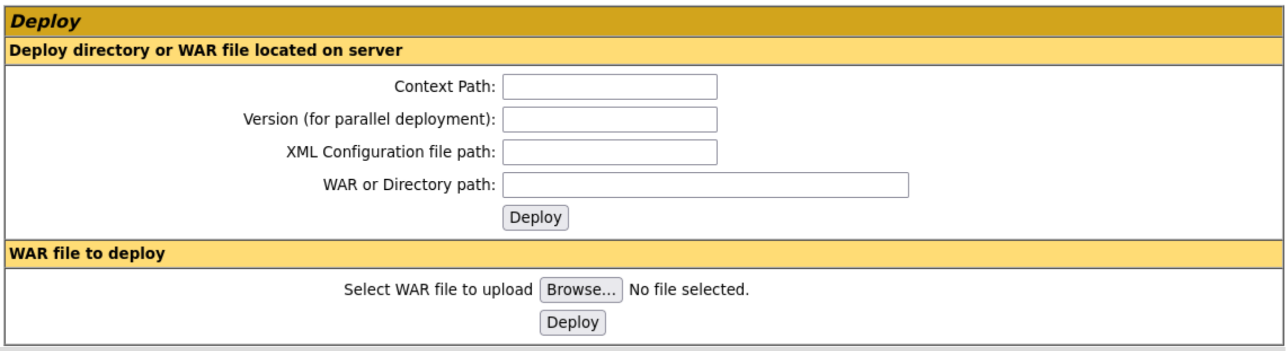
```
Web_App/
├── index.jsp
└── WEB-INF
    ├── classes
    ├── lib
    └── web.xml
```

Once we consider our web application finished, we can generate the file `WAR` belonging to the application, for this we can apply the following commands:

`javac -d WEB-INF/classes *.java` .This command is intended to compile the Java classes of our application.

`jar cvf Aplic_Web.war WEB-INF` to create the `WAR` file.

Once the above has been done, we could access via the web to: `http://127.0.0.1:8080` and, in the home page, access to the option "`Manager App`" and from the resulting window we have the options that appear in the following image to deploy the file `.WAR`:



Deploy

Deploy directory or WAR file located on server

Context Path:	
Version (for parallel deployment):	
XML Configuration file path:	
WAR or Directory path:	

WAR file to deploy

Select WAR file to upload Browse... No file selected.

This website provides a comprehensive overview of the operation, configuration, installation, administration, etc. of the Tomcat application server, and also provides information on how to deploy applications.

http://tomcat.apache.org/

2.3. Implement access registration.

Case Study

Regarding the web applications that have been developed by the company BK Programming and that are already accessible to its customers, it has been considered to carry out some follow-up, so that it can be verified the accesses they have had, at what time and which resources are most demanded; for this Juan, together with María, have configured the Tomcat server to be able to adapt the logs, so that they can obtain information about the accesses to their applications.

To obtain and be able to configure the access logs to a Tomcat application server, as is our case, we will start by talking about Tomcat's access log valves, since it will be the method we will use.

Tomcat valves are a technology introduced from Tomcat 4 that allows an instance of a Java class to be associated with a "`Catalina`" container. This configuration allows the associated class to act as a pre-processor of the requests. These classes are called valves, and they must implement the interface "`org.apache.catalina.valve`" or extend the class "`org.apache.catalina.valves.ValveBase`". The valves are Tomcat's own and cannot be used in other servlet containers.

The valves available are:

- **Access Log Valve**: is implemented by the "`org.apache.catalina.valves.AccessLogValve`". Create `log` files to track access to customer information, recording information such as user session activity, user authentication information, and more. For example, the following code:

  ```
  <Valve className="org.apache.catalina.valves.AccessLogValve"
  directory="logs" prefix="localhost_access_log." suffix=".txt"
  pattern="common"/>
  ```

  It will indicate that the access logs will be stored in the `$CATALINA_HOME/logs` directory and the log files will have the nomenclature with prefix:

localhost_access_log and suffix .txt probably between suffix and prefix the date on which the file is created will be added.

- **Remote Address Filter**: Allows you to compare the client's IP address with one or more regular expressions and, as a result, deny or allow the request submitted by the client. An example of use could be the following:

  ```
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
  deny="127.*">
  ```

  where customers who have an IP that begins with 127 will have their request denied.

- **Remote Host Filter**: is very similar to the previous one but with the difference that it allows you to compare by computer name instead of IP.

  ```
  <Valve className="org.apache.catalina.valves.RemoteHostValve"
  deny="pc_fp.*">
  ```

- **Request Dumper**: is a debugging tool that writes to the `log` the detail of each request made.

  ```
  <Valve
  className="org.apache.catalina.valves.RequestDumperValve"/>
  ```

  Any access to `localhost:8080` it will have a series of entries associated with it in the logs.

- **Single Sign On**: When we want users to be able to identify themselves in any application on our virtual host, and for their identity to be recognized by any application that is on that host.

  ```
  <Valve
  className="org.apache.catalina.authenticator.SingleSignOn"/>
  ```

  We can implement the above examples in `$CATALINA_HOME/conf/server.xml`, so such changes will affect any applications deployed on the server.

"New systems create new problems."

Murphy's Law

2.4. Persistent sessions.

Case Study

Regarding the web applications that have been developed by the company BK programming and that are already accessible to its customers, Ada has asked Juan and María how to be able, in some way, to guarantee the sessions, establishing in the Tomcat configuration persistent sessions that ensure reliable sessions to the applications in case of server failure or loss of connection.

Active sessions by clients to web applications hosted on Tomcat web servers, by default, are configured to be maintained in case of possible loss of connection with the server or possible restarts of the server itself; Despite all this, it is possible to establish greater control over these sessions.

For sessions that are inactive (but not yet expired), you can configure them to be stored on disk, thereby freeing up the associated memory resources. When Tomcat stops, active sessions are dumped to disk so that when you restart it, they can be restored.

Sessions with a lifetime that exceeds a limit are automatically copied to disk for security to prevent potential session lockouts.

To configure persistent sessions, we will have to manage the element `<Manager>` as a sub-element of `<Context>` a way that we can act on two levels depending on whether we want the established configuration to apply to all the applications on the server or to a specific application.

If we configure the persistent sessions globally we have to manipulate the file `/conf/context.xml`, while if we want to configure the sessions locally to an application we would have to adapt the corresponding file `<CATALINA_HOME>/conf/context.xml` to the application.

An example of a configuration could be the following (comments are used to explain each of the parameters):

```
<Context>
```

```
        <!-- classname specifies the class of the server that the
        manager implements, it is recommended to use the
        org.apache.catalina.session.PersistentManager -->
        <Manager
        className="org.apache.catalina.session.PersistentManager">
          <!--saveOnRestart=true to indicate that all sessions are
          saved when the server is restarted -->
          saveOnRestart="true"
          <!--maxActiveSession: when the limit established here is
          exceeded, the new sessions begin to be sent to disk. A
          value of -1 is set to indicate unlimited sessions-->
          maxActiveSession="3"
          <!--minIdleSwap sets the minimum number of seconds that
          elapse before a session can be copied to the hard disk --
          >
          minIdleSwap="0"
          <!--maxIdleSwap indicates the maximum number of seconds
          that elapse before a session can be copied to the hard
          disk -->
          maxIdleSwap="60"
          <!--maxIdleBackup to indicate the number of seconds from
          when a session was last active until it is sent to disk.
          The session is not deleted from memory. Allows session
          restoration in case of server failure. -->
          maxIdleBackup="5">
          <!--Store indicates how and where to store the session,
          the following implementations are available:
          org.apache.catalina.session.FileStore and
          org.apache.catalina.session.JDBCStore -->
          <Store
          className="org.apache.catalina.session.FileStore"/>
        </Manager>
     </Context>
```

2.5. Configure Tomcat in cluster.

Case Study

Once the applications that BK programming has finished developing on the Tomcat server have been placed, an exponential increase in the number of clients that access the services of these applications has been observed, which is why it has been thought to establish some type of cluster on the server to be able to efficiently attend to user requests.

Due to the increase in web applications, scalability and availability become a transcendental resource to guarantee efficient service to web clients; Implementing `clustering` for Web application servers is an effective and relatively simple solution.

Implementation of `clustering` with Tomcat provides:

- **Scalability**: if a web server invests a "T" time to provide a service requested by a web client, to satisfy a large number of services, it is worth asking how much time is invested. The ideal answer to the previous question would be for the time spent to be as close as possible to the time spent on a single request, i.e. as close as possible to "T".

  There are two possible solutions to do this: horizontal scaling (involves increasing the number of servers), vertical scaling (involves increasing the resources of the server itself).

- **High availability**: Tomcat provides `failover`; in the server engine there are two types of `failover` provided by `clustering`:
  - `Request-level failover`: If one server goes down, the following requests will be redirected to other active servers.
  - `Session-level failover`: In the event that a server stops providing service, another server in the cluster should provide the session to the clients in order to minimize the loss of connection, this implies replicating the session in the cluster on the new machine in the shortest possible time.

- **Load balancing**: Establishing a method of distributing the request load among the servers in the cluster, so that the response time to client requests is minimized; this is achieved by using load distribution algorithms.

Clustering

Typical `clustering` solutions offer a server paradigm that consists of offering a system based on distributed execution, although there is a limitation regarding scalability, we can observe the Jakarta Tomcat server engine works scheme.

http://tomcat.apache.org/tomcat-6.0-doc/cluster-howto.html

The cluster server connector receives the request from the clients, and the cluster server processor encapsulates the requests in the objects “**`RequestEntry`**" and writes them in `JavaSpace`. The `Cluster Worker` takes these requests and the processor of the `cluster worker` meet the requests.

To establish a cluster configuration in Tomcat we can follow the following steps:

- All session attributes must implement `java.io.Serializable`.
- Uncomment the **`Cluster`** element in `server.xml`.
- Uncomment `Valve`(`ReplicationValve`) in `server.xml`
- If multiple Tomcat instances are on the same machine, the parameter `tcpListenPort` must be unique for each of the instances.
- Set the `<distributable/>` element in `web.xml` file or define it in a way that `<Context distributable="true"/>`.
- The `jvmRoutes` attribute has to be defined in the "`Engine`” `<Engine name="Catalina" jvmRoute="nodeX">` setting its value to the name of the instance in the cluster.
- Synchronize the time of all nodes with an `NTP` service.
- Set the `loadbalancer` parameter to "`sticky session`" mode.

This website documents the steps to follow to set up a horizontal cluster made up of two servers with a Tomcat instance running on each of them.

http://es.wikibooks.org/wiki/Cluster_Tomcat_HOWTO

3. WildFly Application Server.

Case Study

The employees of BK Programming have heard about the importance of the WildFly (formerly JBoss) application server, since it is an open source server oriented to ebusiness applications; being, for all these reasons, a platform that has acquired great importance in the market, both for individuals and large companies, and that it is worth studying its behavior to be able to implement.

In the same way, it is interesting to establish the mode to operate for the installation and configuration of the WildFly server, as well as each and every one of the steps necessary to be able to perform the Application deployment.

WildFly is a flexible, lightweight, managed application runtime that helps you build amazing applications.

SAVE TIME WITH FASTER DEVELOPMENT

WildFly supports the latest standards for REST based data access, including JAX-RS 2, and JSON-P.

Building on Jakarta EE provides rich enterprise capabilities in easy to consume frameworks that eliminate boilerplate and reduce technical burden.

The quick boot of WildFly combined with the easy-to-use Arquillian framework allows for test driven development using the real environment your code will be running in. Your test code is separate and simply deployed along side your application where it has full access to server resources.

SAVE RESOURCES WITH EFFICIENT MANAGEMENT

WildFly takes an aggressive approach to memory management and is based on pluggable subsystems that can be added or removed as needed.

The quick boot of WildFly combined with the easy-to-use Arquillian framework allows for test driven development using the real environment your code will be running in. Your test code is separate and simply deployed along side your application where it has full access to server resources.

Configuration in WildFly is centralized, simple and user-focused. The configuration file is organized by subsystems that you can easily comprehend and no internal server wiring is exposed. Subsystems use intelligent defaults, but can still be customized to best fit your needs. If you are running in domain mode, the configuration for all servers participating in the domain is specified in a well-organized manner within the same file.

SAVE MONEY WITH OPEN SOURCE

WildFly is an open source community project sponsored by Red Hat. WildFly is fully open. All components are open source and follow a community-focused development process. This allows organizations like yours to develop amazing new technologies and federates the tech world to allow successful startups to come from anywhere.

When your organization needs guaranteed stability and support and is ready for the big-leagues, you should check out the Red Hat JBoss Enterprise Application Platform (EAP). A no-cost subscription to Red Hat JBoss EAP is available for 1-year at no-cost for a non-production development environment.

3.1. Getting Started with WildFly 35

The Getting Started Guide shows you how to install, start and configure the server.

To download on no GUI server:

```
$ wget
https://github.com/wildfly/wildfly/releases/download/35.0.0.F
inal/wildfly-35.0.0.Final.tar.gz
$ tar xvf wildfly-35.0.0.Final.tar.gz
$ mv wildfly-WILDFLY_RELEASE /opt/wildlfy
```

Let’s now create a system user and group that will run WildFly service.

```
sudo groupadd --system wildfly
sudo useradd -s /sbin/nologin --system -d /opt/wildfly  -g
wildfly wildfly
```

Create WildFly configurations directory.

```
sudo mkdir /etc/wildfly
```

Copy WildFly systemd service, configuration file and start scripts templates from the /opt/wildfly/docs/contrib/scripts/systemd/ directory.

```
sudo cp 
/opt/wildfly/docs/contrib/scripts/systemd/wildfly.conf 
/etc/wildfly/
sudo cp 
/opt/wildfly/docs/contrib/scripts/systemd/wildfly.service 
/etc/systemd/system/
sudo cp 
/opt/wildfly/docs/contrib/scripts/systemd/launch.sh 
/opt/wildfly/bin/
sudo chmod +x /opt/wildfly/bin/launch.sh
```

Set `/opt/wildfly` permissions.

```
sudo chown -R wildfly:wildfly /opt/wildfly
```

Reload systemd service.

```
sudo systemctl daemon-reload
```

Start and enable WildFly service:

```
sudo systemctl start wildfly
sudo systemctl enable wildfly
```

Confirm WildFly Application Server status.

```
sudo systemctl status wildfly
```

Service should bind to port 8080.

```
$ ss -tunelp | grep 8080
```

By default WildFly is now distributed with security enabled for the management interfaces. We need to create a user who can access WildFly administration console or remotely use the CLI. A script is provided for managing users.

Run it by executing the command:

```
sudo /opt/wildfly/bin/add-user.sh
```

You will be asked to choose type of user to add. Since this the first user, we want to make it admin. So choose `a`.

```
What type of user do you wish to add?
  a) Management User (mgmt-users.properties)
  b) Application User (application-users.properties)
```

```
 (a):  a
```

Provide desired username for the user.

```
Enter the details of the new user to add.
 Using realm 'ManagementRealm' as discovered from the
existing property files.
 Username : admin
```

Set password for the user:

```
Password recommendations are listed below. To modify these
restrictions edit the add-user.properties configuration file.
The password should be different from the username
The password should not be one of the following restricted
values {root, admin, administrator}
The password should contain at least 8 characters, 1
alphabetic character(s), 1 digit(s), 1 non-alphanumeric
symbol(s)
Password : <Enter Password>
Re-enter Password :  <Confirm Password>
```

Press enter and agree to subsequent prompts to finish user creation.

```
What groups do you want this user to belong to? (Please enter
a comma separated list, or leave blank for none)[  ]: <Enter>
 About to add user 'computingforgeeks' for realm
'ManagementRealm'
 Is this correct yes/no? yes
 Added user 'computingforgeeks' to file
'/opt/wildfly/standalone/configuration/mgmt-users.properties'
 Added user 'computingforgeeks' to file
'/opt/wildfly/domain/configuration/mgmt-users.properties'
 Added user 'computingforgeeks' with groups  to file
'/opt/wildfly/standalone/configuration/mgmt-groups.properties'
 Added user 'computingforgeeks' with groups  to file
'/opt/wildfly/domain/configuration/mgmt-groups.properties'
 Is this new user going to be used for one AS process to
connect to another AS process?
 e.g. for a slave host controller connecting to the master or
for a Remoting connection for server to server EJB calls.
 yes/no? yes
 To represent the user add the following to the server-
identities definition
```

Notice that:

User information is kept on: `/opt/wildfly/domain/configuration/mgmt-users.properties`

Group information is kept on: `/opt/wildfly/standalone/configuration/mgmt-groups.properties`

Accessing WildFly Admin Console

To be able to run WildFly scripts from you current shell session, add `/opt/wildfly/bin/` to your $PATH.

```
cat >> ~/.bashrc <<EOF
export WildFly_BIN="/opt/wildfly/bin/"
export PATH=\$PATH:\$WildFly_BIN
EOF
```

Source the bashrc file.

```
source ~/.bashrc
```

Now test by connecting to WildFly Admin Console from CLI with `jboss-cli.sh` command.

```
$ jboss-cli.sh --connect
Authenticating against security realm: ManagementRealm
Username: computingforgeeks
Password:
[standalone@localhost:9990 /] version
JBoss Admin Command-line Interface
JBOSS_HOME: /opt/wildfly
Release: 29.0.1.Final
Product: WildFly Full 29.0.1.Final
JAVA_HOME: null
java.version: 11.0.13
java.vm.vendor: Ubuntu
java.vm.version: 11.0.13+8-Ubuntu-0ubuntu1
os.name: Linux
os.version: 5.13.0-19-generic
[standalone@localhost:9990 /] exit
```

Accessing WildFly Admin Console from Web Interface

By default, the console is accessible on localhost IP on port 9990.

```
$ ss -tunelp | grep 9990
```

We can start it on a different IP address accessible from outside the local server. Edit `/opt/wildfly/bin/launch.sh` to look like this:

```
$ sudo vim /opt/wildfly/bin/launch.sh
```

```
#!/bin/bash

if [ "x$WILDFLY_HOME" = "x" ]; then
    WILDFLY_HOME="/opt/wildfly"
fi

if [[ "$1" == "domain" ]]; then
    $WILDFLY_HOME/bin/domain.sh -c $2 -b $3
else
    $WILDFLY_HOME/bin/standalone.sh -c $2 -b $3 
-bmanagement=0.0.0.0
fi
```

We added `-bmanagement=0.0.0.0` to start script line. This binds "**management**" interface to all available IP addresses.

Restart wildfly service

```
sudo systemctl restart wildfly
```

Check service status to confirm it was successful

```
$ systemctl status  wildfly
```

Confirm port 9990 is listening:

```
$ ss -tunelp | grep 9990
```

Open your browser and URL `http://serverip:9990` to access WildFly Web console.

Use username created earlier and password to authenticate. WildFly console will be the next window to show.

3.2. Getting Started with WildFly.

Build and run a Jakarta EE application with WildFly in a few minutes.

Step 0. Install Java & Maven

You need Java (at least version `11`, and preferably `17`) and Maven installed on your machine to create a Maven project that contains the source code of the Jakarta EE application.

You can verify they are installed by executing the commands:

```
java -version
mvn -version
```

Step 1. Create the Application

You can create the Jakarta EE application as a Maven project by executing the commands:

```
mvn archetype:generate \
    -DarchetypeGroupId=org.wildfly.archetype \
    -DarchetypeArtifactId=wildfly-getting-started-archetype
```

The `getting-started` project that is generated contains a simple "Hello World" application that exposes a HTTP endpoint with the Jakarta-RS API.

The Maven project is configured to "provision" (install and configure) the WildFly that hosts your application.

Step 2. Build the Application

You can build the application by executing the commands:

```
cd getting-started
mvn package verify
```

This Maven command compiles the Jakarta EE application, provisions WildFly, deploys the application into WildFly and runs integration tests against it. When this command is finished, you have a fully functional, tested application running on WildFly.

Step 3. Run the Application

The `target/server` contains a fully functional WildFly server with your application. You start it by executing the command:

```
./target/server/bin/standalone.sh
```

The application is accessible at http://localhost:8080/.

To stop the application, type `Ctrl + C` in the terminal where you started WildFly.

Step 4. Continuous Development

You can develop your application and see the updates in the running application immediately by using the `wildfly:dev` goal from the root of your project:

```
mvn clean wildfly:dev
```

The application is accessible at http://localhost:8080 and will be continuously updated when its code changes.

Open your favorite code editor and change the `hello` method in the `GettingStartedService.java` file:

```
public String hello(String name) {
  return String.format("Hello '%s'.", name.toUpperCase());
}
```

Save the file and the application will be recompiled and updated in WildFly. If you access the application at http://localhost:8080, it will now return the name in upper case.

What’s next?

To learn more about WildFly, you can read its documentation. If you want to learn how to use WildFly on OpenShift, read the Getting Started with WildFly on OpenShift Guide. In addition, you can also watch the talk from our first mini conference about getting started with WildFly. Finally you can browse more guides on a wide range of topics relating to WildFly.

How To Install Apache Tomcat 10 on Ubuntu Server 22.04

Introduction

Apache Tomcat is a web server and servlet container that is used to serve Java applications. It’s an open source implementation of the Jakarta Servlet, Jakarta Server Pages, and other technologies of the Jakarta EE platform.

Step 1 — Installing Tomcat

In this section, you will set up Tomcat 10 on your server. To begin, you will download its latest version and set up a separate user and appropriate permissions for it. You will also install the Java Development Kit (JDK).

For security purposes, Tomcat should run under a separate, unprivileged user. Run the following command to create a user called tomcat:

```
sudo useradd -m -d /opt/tomcat -U -s /bin/false tomcat
```

By supplying /bin/false as the user’s default shell, you ensure that it’s not possible to log in as tomcat.

You’ll now install the JDK. First, update the package manager cache by running:

```
sudo apt update
```

Then, install the JDK by running the following command:

```
sudo apt install default-jdk
```

Answer `y` when prompted to continue with the installation.

When the installation finishes, check the version of the available Java installation:

```
java -version
```

To install Tomcat, you’ll need the latest Core Linux build for Tomcat 10, which you can get from the downloads page. Select the latest Core Linux build, ending in `.tar.gz`. At the time of writing, the latest version was `10.1.34`.

First, navigate to the `/tmp` directory:

```
cd /tmp
```

Download the archive using `wge`t by running the following command:

```
wget 
https://dlcdn.apache.org/tomcat/tomcat-10/v10.1.34/bin/apache
-tomcat-10.1.34.tar.gz
```

The `wge`t command downloads resources from the Internet.

Then, extract the archive you downloaded by running:

```
sudo tar xzvf apache-tomcat-10*tar.gz -C /opt/tomcat --strip-
components=1
```

Since you have already created a user, you can now grant `tomcat` ownership over the extracted installation by running:

```
sudo chown -R tomcat:tomcat /opt/tomcat/
sudo chmod -R u+x /opt/tomcat/bin
```

Both commands update the settings of your `tomcat` installation.

In this step, you installed the JDK and Tomcat. You also created a separate user for it and set up permissions over Tomcat binaries. You will now configure credentials for accessing your Tomcat instance.

Step 2 — Configuring Admin Users

To gain access to the Manager and Host Manager pages, you'll define privileged users in Tomcat's configuration. You will need to remove the IP address restrictions, which disallows all external IP addresses from accessing those pages.

Tomcat users are defined in `/opt/tomcat/conf/tomcat-users.xml`. Open the file for editing with the following command:

```
sudo nano /opt/tomcat/conf/tomcat-users.xml
```

Add the following lines before the ending tag:

```
<role rolename="manager-gui" />
<user username="manager" password="manager_password" 
roles="manager-gui" />
```

```
<role rolename="admin-gui" />
<user username="admin" password="admin_password"
roles="manager-gui,admin-gui" />
```

Replace highlighted passwords with your own. When you’re done, save and close the file.

Here you define two user roles, `manager-gui` and `admin-gui`, which allow access to **Manager** and **Host Manager** pages, respectively. You also define two users, `manager` and `admin`, with relevant roles.

By default, Tomcat is configured to restrict access to the admin pages, unless the connection comes from the server itself. To access those pages with the users you just defined, you will need to edit config files for those pages.

To remove the restriction for the **Manager** page, open its config file for editing:

```
sudo nano /opt/tomcat/webapps/manager/META-INF/context.xml
```

Comment out the `Valve` definition, as shown:

```
...
<Context antiResourceLocking="false" privileged="true" >
  <CookieProcessor
className="org.apache.tomcat.util.http.Rfc6265CookieProcessor"
                   sameSiteCookies="strict" />
<!--  <Valve
className="org.apache.catalina.valves.RemoteAddrValve"
         allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" /> -->
  <Manager sessionAttributeValueClassNameFilter="java\.lang\.
(?:Boolean|Integer|Long|Number|String)|
org\.apache\.catalina\.filters\.Csr>
</Context>
```

Save and close the file, then repeat for Host Manager:

```
sudo nano
/opt/tomcat/webapps/host-manager/META-INF/context.xml
```

You have now defined two users, `manager` and `admin`, which you will later use to access restricted parts of the management interface. You’ll now create a `systemd` service for Tomcat.

Step 3 — Creating a `systemd` service

The `systemd` service that you will now create will keep Tomcat quietly running in the background. The `systemd` service will also restart Tomcat automatically in case of an error or failure.

Tomcat, being a Java application itself, requires the Java runtime to be present, which you installed with the JDK in step 1. Before you create the service, you need to know where Java is located. You can look that up by running the following command:

```
sudo update-java-alternatives -l
```

The output will be similar to this:

```
java-1.11.0-openjdk-amd64      1111
/usr/lib/jvm/java-1.11.0-openjdk-amd64
```

Note the path where Java resides, listed in the last column. You'll need the path momentarily to define the service.

You'll store the `tomcat` service in a file named `tomcat.service`, under `/etc/systemd/system`. Create the file for editing by running:

```
sudo nano /etc/systemd/system/tomcat.service
```

Add the following lines:

```
[Unit]
Description=Tomcat
After=network.target

[Service]
Type=forking

User=tomcat
Group=tomcat

Environment="JAVA_HOME=/usr/lib/jvm/java-1.11.0-openjdk-
amd64"
Environment="JAVA_OPTS=-Djava.security.egd=file:///dev/
urandom"
Environment="CATALINA_BASE=/opt/tomcat"
Environment="CATALINA_HOME=/opt/tomcat"
Environment="CATALINA_PID=/opt/tomcat/temp/tomcat.pid"
Environment="CATALINA_OPTS=-Xms512M -Xmx1024M -server -XX:
+UseParallelGC"
```

```
ExecStart=/opt/tomcat/bin/startup.sh
ExecStop=/opt/tomcat/bin/shutdown.sh

RestartSec=10
Restart=always

[Install]
WantedBy=multi-user.target
```

Modify the highlighted value of `JAVA_HOME` if it differs from the one you noted previously.

Here, you define a service that will run Tomcat by executing the startup and shutdown scripts it provides. You also set a few environment variables to define its home directory (which is `/opt/tomcat` as before) and limit the amount of memory that the Java VM can allocate (in `CATALINA_OPTS`). Upon failure, the Tomcat service will restart automatically.

When you're done, save and close the file.

Reload the systemd daemon so that it becomes aware of the new service:

```
sudo systemctl daemon-reload
```

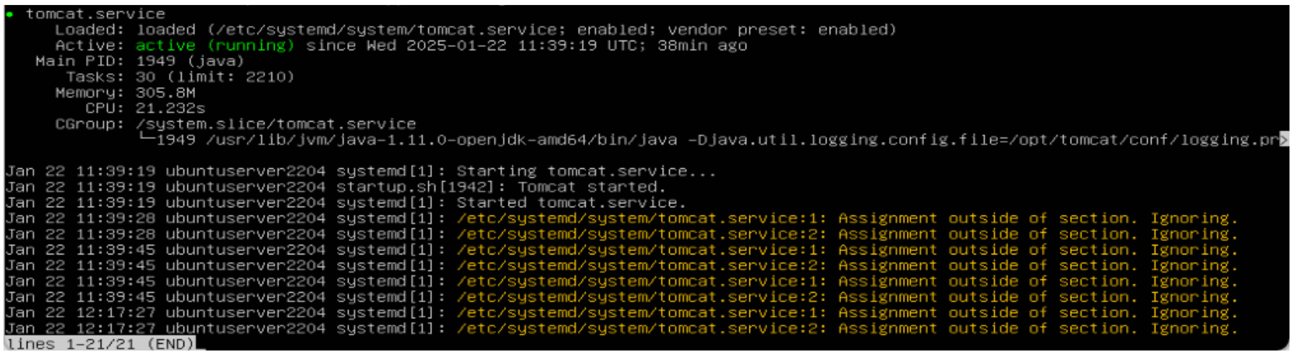
You can then start the Tomcat service by typing:

```
sudo systemctl start tomcat
```

Then, look at its status to confirm that it started successfully:

```
sudo systemctl status tomcat
```

The output will look like this:



```
● tomcat.service
     Loaded: loaded (/etc/systemd/system/tomcat.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2025-01-22 11:39:19 UTC; 38min ago
   Main PID: 1949 (java)
      Tasks: 30 (limit: 2210)
     Memory: 305.8M
        CPU: 21.232s
     CGroup: /system.slice/tomcat.service
             └─1949 /usr/lib/jvm/java-1.11.0-openjdk-amd64/bin/java -Djava.util.logging.config.file=/opt/tomcat/conf/logging.pr>

Jan 22 11:39:19 ubuntuserver2204 systemd[1]: Starting tomcat.service...
Jan 22 11:39:19 ubuntuserver2204 startup.sh[1942]: Tomcat started.
Jan 22 11:39:19 ubuntuserver2204 systemd[1]: Started tomcat.service.
Jan 22 11:39:28 ubuntuserver2204 systemd[1]: /etc/systemd/system/tomcat.service:1: Assignment outside of section. Ignoring.
Jan 22 11:39:28 ubuntuserver2204 systemd[1]: /etc/systemd/system/tomcat.service:2: Assignment outside of section. Ignoring.
Jan 22 11:39:45 ubuntuserver2204 systemd[1]: /etc/systemd/system/tomcat.service:1: Assignment outside of section. Ignoring.
Jan 22 11:39:45 ubuntuserver2204 systemd[1]: /etc/systemd/system/tomcat.service:2: Assignment outside of section. Ignoring.
Jan 22 11:39:45 ubuntuserver2204 systemd[1]: /etc/systemd/system/tomcat.service:1: Assignment outside of section. Ignoring.
Jan 22 11:39:45 ubuntuserver2204 systemd[1]: /etc/systemd/system/tomcat.service:2: Assignment outside of section. Ignoring.
Jan 22 12:17:27 ubuntuserver2204 systemd[1]: /etc/systemd/system/tomcat.service:1: Assignment outside of section. Ignoring.
Jan 22 12:17:27 ubuntuserver2204 systemd[1]: /etc/systemd/system/tomcat.service:2: Assignment outside of section. Ignoring.
lines 1-21/21 (END)
```

To enable Tomcat starting up with the system, run the following command:

```
sudo systemctl enable tomcat
```

In this step, you identified where Java resides and enabled systemd to run Tomcat in the background. You'll now access Tomcat through your web browser.

Step 4 — Accessing the Web Interface

Now that the Tomcat service is running, you can configure the firewall to allow connections to Tomcat. Then, you will be able to access its web interface.

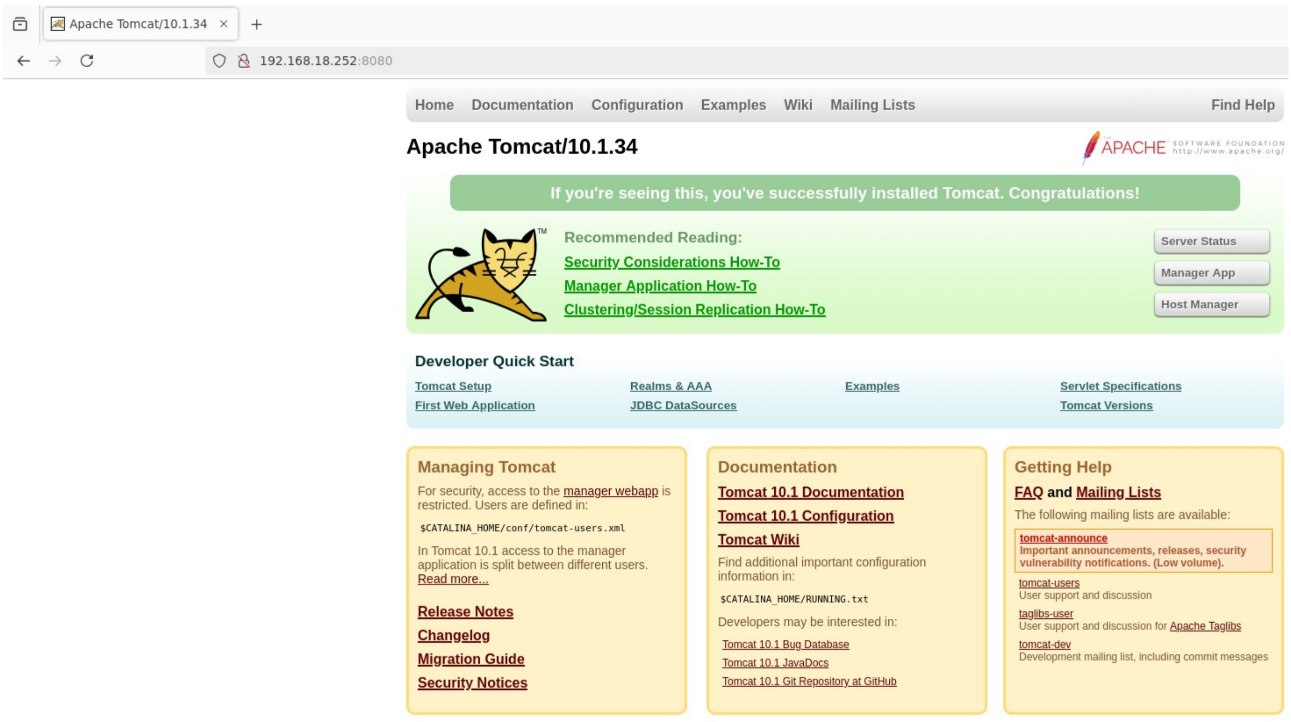
Tomcat uses port `8080` to accept HTTP requests. Run the following command to allow traffic to that port:

```
sudo ufw allow 8080
```

In your browser, you can now access Tomcat by navigating to the IP address of your server:

```
http://your_server_ip:8080
```

You'll see the default Tomcat welcome page:



Apache Tomcat/10.1.34

If you're seeing this, you've successfully installed Tomcat. Congratulations!

Recommended Reading:
Security Considerations How-To
Manager Application How-To
Clustering/Session Replication How-To

Developer Quick Start
Tomcat Setup
First Web Application
Realms & AAA
JDBC DataSources
Examples
Servlet Specifications
Tomcat Versions

Managing Tomcat
For security, access to the manager webapp is restricted. Users are defined in:
$CATALINA_HOME/conf/tomcat-users.xml
In Tomcat 10.1 access to the manager application is split between different users. Read more...
Release Notes
Changelog
Migration Guide
Security Notices

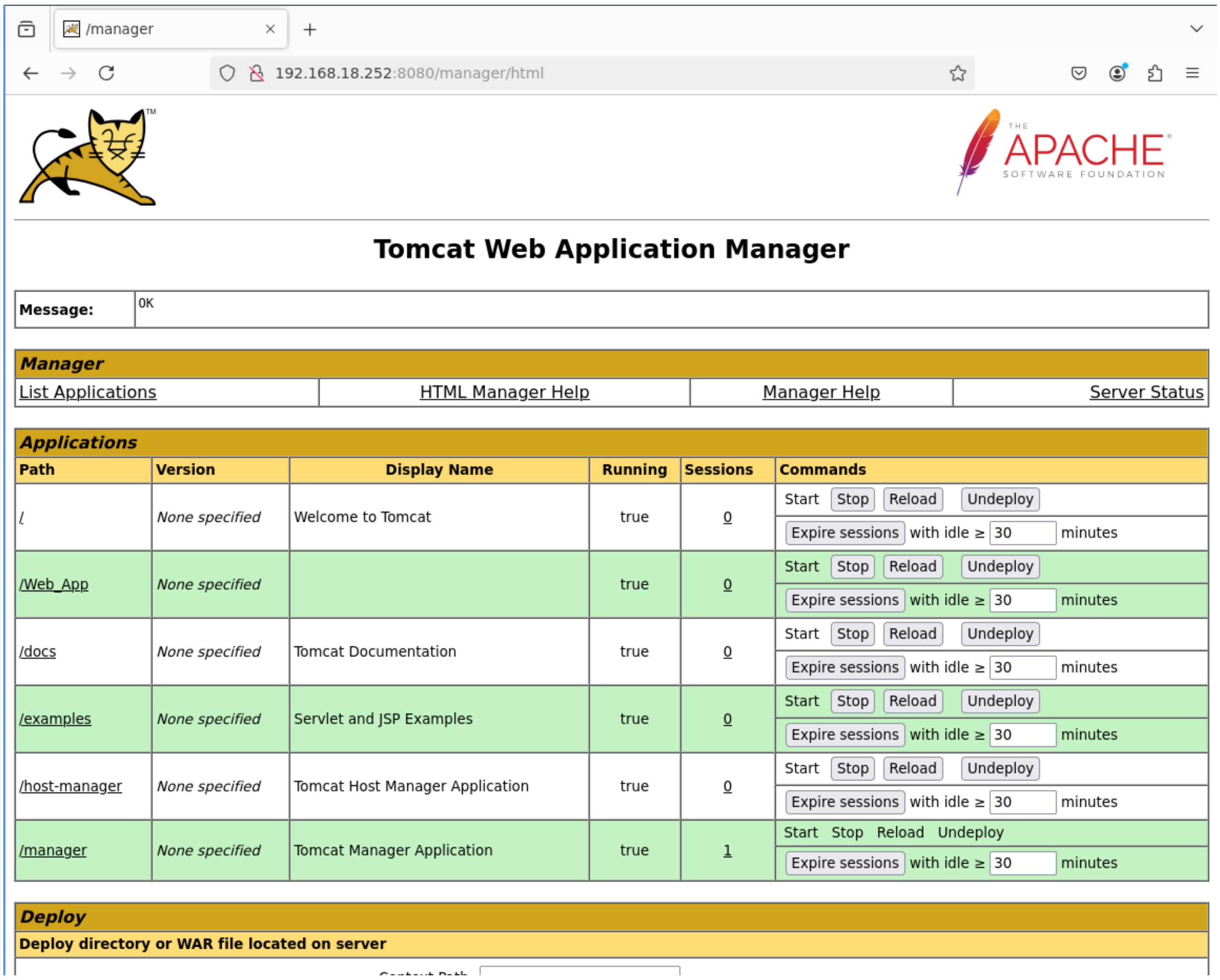
Documentation
Tomcat 10.1 Documentation
Tomcat 10.1 Configuration
Tomcat Wiki
Find additional important configuration information in:
$CATALINA_HOME/RUNNING.txt
Developers may be interested in:
Tomcat 10.1 Bug Database
Tomcat 10.1 JavaDocs
Tomcat 10.1 Git Repository at GitHub

Getting Help
FAQ and Mailing Lists
The following mailing lists are available:
tomcat-announce
Important announcements, releases, security vulnerability notifications. (Low volume).
tomcat-users
User support and discussion
taglibs-user
User support and discussion for Apache Taglibs
tomcat-dev
Development mailing list, including commit messages

You've now verified that the Tomcat service is working.

Press on the **Manager App** button on the right. You'll be prompted to enter the account credentials that you defined in a previous step.

You should see a page that looks like this:



Tomcat Web Application Manager

Message:	OK

Manager

List Applications	HTML Manager Help	Manager Help	Server Status

Applications

Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes
/Web_App	None specified		true	0	Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy / Expire sessions with idle ≥ 30 minutes

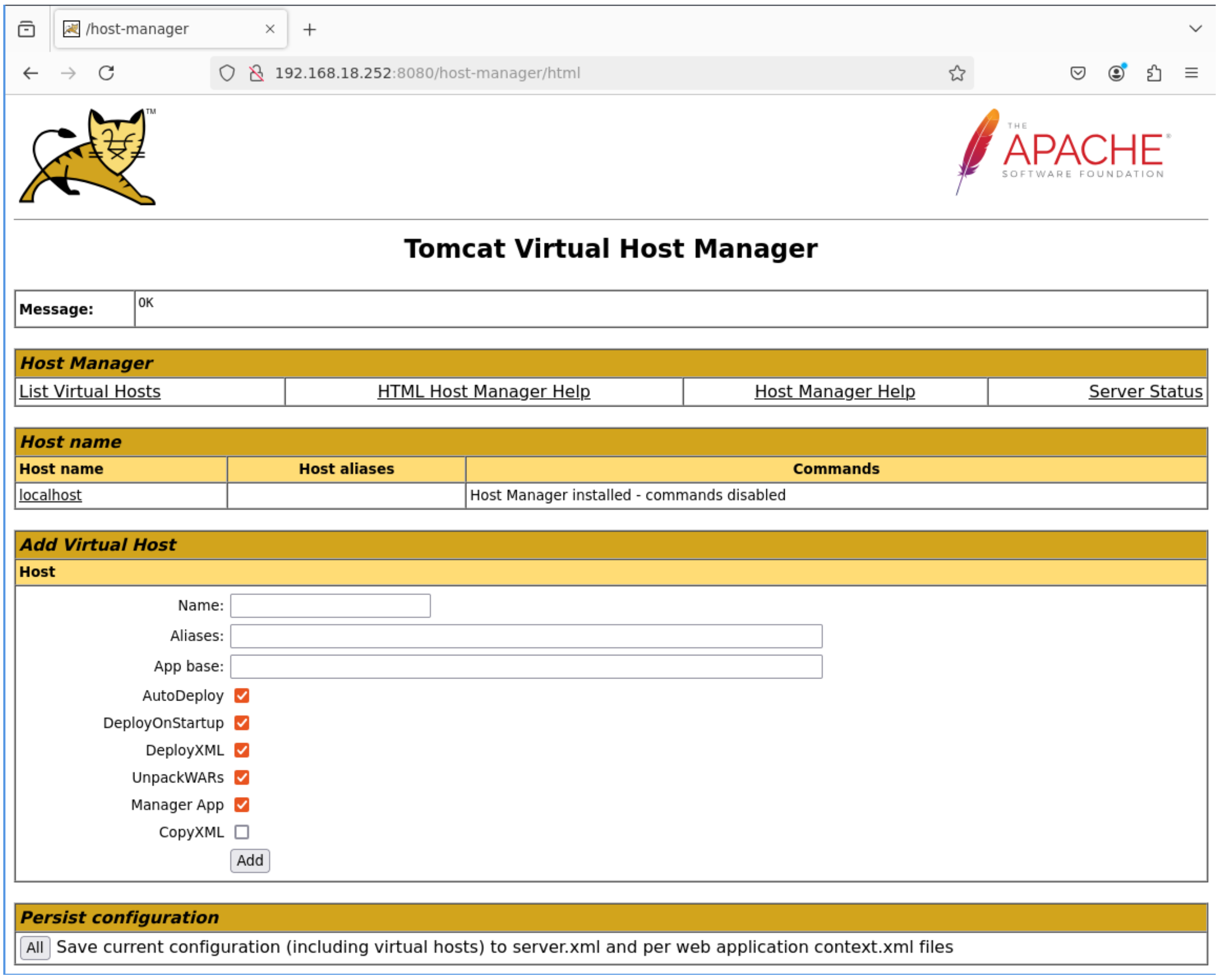
Deploy

Deploy directory or WAR file located on server

The Web Application Manager is used to manage your Java applications. You can start, stop, reload, deploy, and undeploy them from here. You can also run some diagnostics on your apps (for example, to find memory leaks). Information about your server is available at the very bottom of this page.

Now, take a look at the **Host Manager**, accessible by pressing its button on the main page:

Here, you can add virtual hosts to serve your applications from. Keep in mind that this page is not accessible by users who don't have the `admin-gui` role assigned, such as `manager`.


/host-manager
192.168.18.252:8080/host-manager/html
APACHE SOFTWARE FOUNDATION

Tomcat Virtual Host Manager

Message:	OK

Host Manager

List Virtual Hosts	HTML Host Manager Help	Host Manager Help	Server Status

Host name

Host name	Host aliases	Commands
localhost		Host Manager installed - commands disabled

Add Virtual Host

Host

Name:	
Aliases:	
App base:	
AutoDeploy	✓
DeployOnStartup	✓
DeployXML	✓
UnpackWARs	✓
Manager App	✓
CopyXML	☐

Add

Persist configuration

All Save current configuration (including virtual hosts) to server.xml and per web application context.xml files

Conclusion

You installed Tomcat 10 on your Ubuntu 22.04 server and configured it to be accessible remotely with management accounts. You can now use it to deploy your Java applications, based on Jakarta EE technologies. You can learn more about Java apps by visiting the official docs.

Web Links

In this section, you will find the relevant links of interest necessary to expand and explore the contents of the unit.

- Seguridad en la red. This website arises with the aim of raising awareness and helping people to increase security on the network, it appears, in an updated way, threats, attacks, security recommendations, etc.
- INCIBE | INCIBE
- Apache Tomcat® - Welcome! This website shows, in a broad way, the operation, configuration, installation, administration, etc. of the Tomcat application server, where we can also find how to deploy applications.
- https://ubuntu.com/server/docs/install-and-configure-a-mysql-server
- https://dev.mysql.com/doc/refman/8.4/en/default-privileges.html
- https://ubuntu.com/server/docs/how-to-install-and-configure-php
-