



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 |
| How To Install Apache Tomcat 10 on Ubuntu Server 22.04..... | 55 |
| Web Links..... | 63 |

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.

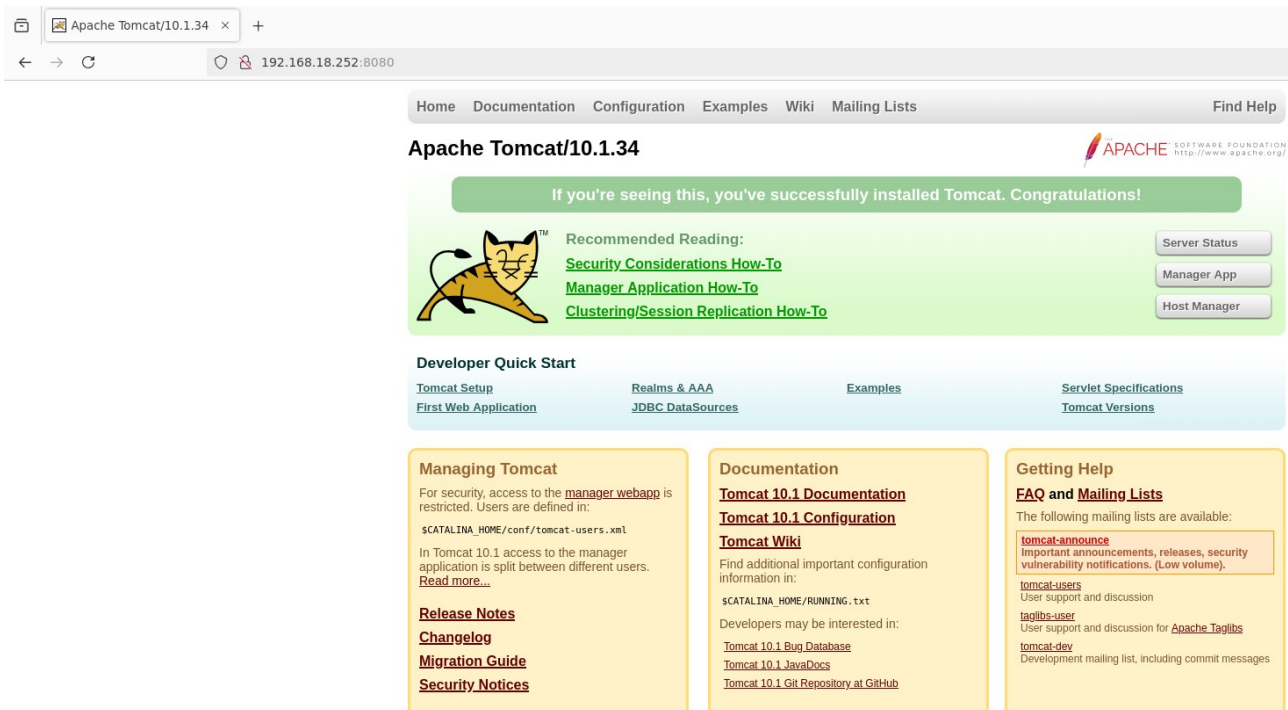
To achieve secure web applications, a series of mechanisms and tools must be used, among which we highlight:

- 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.



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:



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:



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: | <input type="text"/> |
| Version (for parallel deployment): | <input type="text"/> |
| XML Configuration file path: | <input type="text"/> |
| WAR or Directory path: | <input type="text"/> |
| <input type="button" value="Deploy"/> | |
| WAR file to deploy | |
| Select WAR file to upload | <input type="button" value="Browse..."/> No file selected. |
| <input type="button" value="Deploy"/> | |

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 same; 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.

<Manager>

<Context>

To configure persistent sessions, we will have to manage the element as a sub-element of 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

<CATALINA_HOME>/conf/context.xml

We would have to adapt the corresponding file 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.

Clustering

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

Clustering

Implementation 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).

failover

✓ High availability: Tomcat provides; 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 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>

RequestEntry " and writes them in JavaSpace

The cluster server connector receives the request from the clients, and the cluster server processor encapsulates the requests in the objects." The

Cluster Worker takes these requests and the processor of the cluster worker Connector meet requests.

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

java.io.Serializable

- ✓ All session attributes must be implemented.
- ✓ Uncomment the Cluster element in server.xml.

Valve

- ✓ Uncomment (ReplicationValve) in server.xml

tcpListenPort

- ✓ If multiple Tomcat instances are on the same machine, the parameter must be unique for each of the instances.

web.xml The <distributable/> or define it in a way that <Context

- ✓ Set to distributable file ="true"/>.

Engine " <Engine name="Catalina"

- ✓ The jvmRoutes attribute has to be defined in the " jvmRoute="nodeX"> setting its value to the name of the instance in the cluster.
- ✓ Synchronize the time of all nodes with an NTP service.

loadbalancer

- ✓ Set the 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.- The JBoss application server.

Case Study

The employees of BK Programming have heard about the importance of the 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 JBoss server, as well as each and every one of the steps necessary to be able to perform the

Application deployment.

The JBOSS server is an open source project, with which an application server based on J2EE is achieved, and implemented 100% in Java.

Servlet Container , JBoss is a Application Server

While the Tomcat is a , which supports J2EE functions, the most important are EJB's and clustering. Tomcat on its own simply works for JSP's and servlets.

JBoss is a Java-based application server while Tomcat is a servlet container.

One of the most important features of JBoss is its support for "hot" implementation. What it means is that implementing a new EJB is as simple as copying the corresponding file into the corresponding directory. If this is done while the bean is already loaded, JBOSS automatically downloads it, and then loads the new version.

Servlet Engine " and a " EJB Engine ", within the " Servlet

JBoss is composed of two parts: a "

Engine

(

EJB Engine Container

" exclusively run classic server applications (JSP's and Servlets), while the ")" is reserved for applications developed around EJB's or Enterprise Java Bean's.

JBoss is the first production-ready and J2EE 1.4 certified open source application server, offering a high-performance platform for e-business applications. Combining a revolutionary service-oriented architecture with an open-source license, JBoss can be downloaded, used, embedded, and distributed without license restrictions. For this reason, it is the most popular middleware platform for developers, independent software vendors, and also for large enterprises.

Among the outstanding features of JBoss we highlight the following:

- ✓ Open source license product at no additional cost.
- ✓ Meets standards.
- ✓ Reliable at the company level.
- ✓ Embeddable, service architecture oriented.
- ✓ Consistent flexibility.
- ✓ Middleware services for any Java object.

The creator of the first version of JBoss was Marc Fleury who founded a services company called JBoss Inc., acquired in 2006 by Red Hat.

For example, the Sims online game uses JBoss as well as other multi-user games.

Which of the following are features of the JBoss Application Server?

It is open source.

It is implemented entirely in Java.

It is only an "EJB Container".

It works only on Microsoft Windows servers.

It is oriented to service architecture.

3.1.- Basic installation and configuration.

We are going to start with a Debian 6.0.1 Squeeze machine, in which we will carry out the process of installation and basic configuration of the JBoss server and that we are going to structure in the following steps:

1. Download and install Java Development Kit (JDK): First of all, it should be noted that, in order to install any version of JBoss, it is necessary to have JDK (Java Development Kit) installed, since it is an application server based and implemented 100% in Java, as mentioned above, and can be run on any system in which a JDK in its version 1.5 or higher is operational. We start by looking for the Java package that we may be interested in. With the following command we would get the list of the Java environment because Debian provides several implementations, each of these packages has a development environment (JDK) and a known runtime (JRE or Java Virtual Machines JVM):

```
#aptitude search "?provides(java-runtime)"
```

2. Then we install these packages using the following command, this will not involve any kind of complication since they are accessible from the repository that we have by default:

```
#apt-get install default-jre openjdk-6-jdk
```

```
sources.list
```

To install the JDK version of Sun (now Oracle) in Debian 6 (Squeeze) we have to add a repository, for this we edit the file using the following command:

```
#nano /etc/apt/sources.list and add the following line: deb http://ftp.ch.debian.org/debian/  
squeeze main non-free
```

```
update
```

we save the file and then run the command: `#aptitude update` or `#apt-get` and, once the update is done, we will install the following Sun Java packages

using the following command:

```
#aptitude install sun-java6-jre sun-java6-jdk and, once installed, select it using:
```

```
#update-alternatives --config java that will display the available options and select the  
option number that contains the Sun/Oracle Java VM.
```

3. Download and installation of JBoss Application Server 6.0: You can download the different versions of the JBoss server from the following link, in this case we have decided to download the `jboss-as-distribution-6.0.0.Final.zip` package.
<http://www.jboss.org/jbossas/downloads/>

```
/usr/local/jboss/
```

To proceed with its installation we simply go to the folder where we want to install it, in our case we will do it in `""` and, once there, we unzip the package using:

```
#unzip jboss-as-distribution-6.0.0.Final.zip
```

4. Create the JBoss user who owns and runs JBoss: It is recommended that you run JBoss with a non-root, least-privilege user account. To do this, we will create a JBoss group and a user called JBoss to which we will put a password and add to the created group; We can do this as follows:

```
#groupadd jboss
```

```
#useradd -s /bin/bash -g jboss jboss
```

```
#passwd jboss
```

```
#usermod -d /usr/local/jboss/jboss-6.0.0.Final/ jboss
```


Set the Environment Variables `JAVA_HOME` and `JBOSS_HOME` : These variables are interesting

5.

to indicate the paths where Java and JBoss have been installed. These paths will be used in the configuration files of these applications, for this we simply add, in our case, the following content to the `/etc/profile` file:

```
JAVA_HOME=/usr/lib/jvm/java-6-sun/jre
```

```
JBOSS_HOME=/usr/local/jboss/jboss-6.0.0.Final      PATH=$PATH:$JAVA_HOME/bin:  
$JBOSS_HOME/bin export PATH JAVA_HOME JBOSS_HOME
```

then we would run `#source /etc/profile` so that the system collects the content of the created variables without having to restart the computer.

6. Create one script for automate Jboss with the Parameters/functionalities

```
jboss_init_redhat.sh " in the folder $JBOSS_HOME/bin
```

"start/stop/restart" and configure JBoss to run as a service: There is a script called " that will help us create the script that manages the JBoss server; To do this, we copy the script to `/etc/init.d` and rename it to `jboss` `#cp $JBOSS_HOME/bin/jboss_init_redhat.sh /etc/init.d/jboss` , then edit the copied file, where we have to replace the following lines, adapting them to our configuration, in our case:

```
JBOSS_HOME=${JBOSS_HOME:-"/usr/local/jboss/jboss-6.0.0.Final"}
```

```
JAVAPATH=${JAVAPATH:-"/usr/java/jdk1.6.0_24"}
```

and add the line `JBOSS_HOST="0.0.0.0"` thus allowing access to JBoss from any IP.

```
http://ip_equipo:8080
```

7. Access the JBoss management console: Make sure that JBoss has been launched and that we are able to access the JBoss console from the following addresses:

```
http://localhost:8080
```

and also if it is accessed from the server itself.

/usr/local/jboss-

8. Change JBoss administrator password: We edit the "

6.0.0.Final/server/default/conf/props/jmx-console-

users.properties " in where Introduce

the

admin=

password that we decide next.

To perform the basic installation and configuration of the JBoss 6.0 server, we must follow, sequentially, each and every one of the following steps:

1. Download and install the Java Development Kit (JDK), an essential requirement to be able to operate the server.
2. Download and install JBoss Application Server 6.0: This is free software.
3. Create the JBoss user, who owns and runs JBoss, because it is advisable not to work with the root user of a machine, to manage a web server.
4. Set the environment variables JAVA_HOME and JBOSS_HOME: they will speed up the server configuration process since they are used by many configuration files.
5. Create a script to automate JBoss with the "start/stop/restart" parameters/functionalities and configure JBoss to run as a service, although it is not necessary, but it is more convenient to start the server as another service.
6. Access the JBoss administration console, from where we can manage the server from a web environment.

3.2.- Deployment of business applications.

JBoss, acquired by Red Hat in 2006, is a market leader in offering enterprise-grade Open Source middleware solutions. JBoss Middleware Enterprise is a set of certified and

supported platforms and frameworks with the level of professional quality offered by Red Hat.

JBoss Enterprise Middleware solutions are delivered via "JBoss Subscription," which includes certified software and updates, management tools, long-term maintenance policies, and industry-leading technical support. Subscriptions are available for both production and development use.

The JBoss Enterprise platforms, detailed below, integrate multiple projects and components, the most popular in the community JBoss.org in certified, stable and secure distributions, with a single path of patches and updates.

- ✓ JBoss Enterprise Application Platform.
 - ➔ Designed to build, deploy and host Java applications and services.
 - ➔ It integrates the JBoss AS application server in cluster, an O/R mapping and persistence system, and a powerful framework for building next-generation Web 2.0 applications.
- ✓ JBoss Enterprise Web Platform.
 - ➔ For web applications in Java and rich Internet-based applications (RIAs).
- ✓ JBoss Enterprise Web Server.
 - ➔ A single enterprise solution based on open source for web servers based on Apache and Tomcat technology.
- ✓ JBoss Enterprise Portal Platform.
 - ➔ Designed to build and deploy portals for SOA user interaction and personalized presentation.
 - ➔ It integrates a portal framework, CMS functionalities with workflow and JBoss Enterprise Application Platform.
- ✓ JBoss Enterprise SOA Platform.

- ➔ Integrate applications and orchestrate services to automate business processes in a service-oriented architecture.
- ➔ It is built on a service bus and integrates a rules engine, business process automation, and JBoss Enterprise Application Platform.
- ✓ JBoss Enterprise BRMS.
- ➔ An enterprise open source-based system for managing business rules that makes it easy to develop, access, and manage policy and business rule changes.
- ✓ JBoss Enterprise Data Services Platform.
- ➔ Break down the disconnect between the diverse enterprise data sources that exist and the innovative data formats required by new projects, applications, and architectures.

The structure of a web application in its simplest form should contain the following directory structure:

META-INF/ manifest.mf WEB-INF/ classes/ src/ lib/ web.xml

META-INF , in applications .Jar , the archive manifest.mf

WEB-INF

Classes that contains the classes compiled for the application, Lib

containing the folder, which contains the list of contents of the application, and which are generated at the time of creating it. The directory contains all the files needed to run the application, and structures its contents in folders with the libraries necessary for the application and src, with the source code of the application.

Once the JEE application is properly built, packaging is done with the command:

```
#jar CVF nombre_aplicacion.jar folders/ficheros_a_empaquetar
```

Once we have the .jar application to deploy it, we only copy it to the folder
deploy,

"\$JBOSS_HOME/server/default/deploy" and JBoss itself will give us a message similar to
ctxPath = /nombre_aplicacion, which means that the application has been deployed
correct; This is known as hot deployment.

3.3.- Folder structure of a business application. EAR file.

In the Java EE world we have three possible types of applications: web applications,
distributed objects EJBs and business applications, which are nothing more than a set of
the two previous applications.

EAR

.war

.Jar

A Java EE enterprise application (file .) is a set of modules, with a module being a
complete web application (packaged in a file) or set of distributed objects EJBs (packaged
in a file).

EAR

We can summarize that the structure of the file is:

✓

✓

/*war : War files.

: Files (ejb) jar.

/*Jar

/META-INF/application.xml

EAR

EJB

✓ : Module deployment descriptor, where the name and description of the application that is deployed, and the different web modules that make up the application, are registered and declared.

app.war

Let's assume a structure that is as simple as possible for a web application like the following, and that is the one that constitutes the

"".

index.html " and a deployment descriptor " web.xml

EAR

WAR with one page HTML

where we observe a static page "", from this structure we intend to build our own file that will contain a single static file.

Application ", using the #jar cvf app.war *

. WAR

.zip

Once located in the folder " we will generate the file corresponding to the application; We can verify that it is a format similar to the files by trying to open it with a compressor program.

temporary

To construct the . EAR, at least, we will have to create a deployment descriptor that we will call "aplicacion.xml", for this we create a folder called "" where we place the file

"aplicacion.war"; in the same path we create a folder called "META-INF" where we are going to create the descriptor; the structure is as follows:

We are located inside the folder "temporary " and created the .ear Using the command:

```
#jar cvf app.ear *
```

.ear

and thus we will have the file corresponding to the created application.

Which of the following statements are correct?

An archive .war can be made up of several files .ear. An archive .ear can be made up of several files .war.

The command #jar cvf Allows you to generate files .war.

The command #jar cvf Allows you to generate files .ear.

An .ear file can contain .jar files.

4.- Automatic construction and deployment with Ant.

Case Study

In order to speed up the process of building web applications, BK Programming has thought of automating the process with the help of the Ant tool that is used to perform mechanical and repetitive tasks, usually during the compilation and construction phase.

When implementing this tool, it has also been proposed to document the procedure for installation, configuration and commissioning of this tool.

ANT (Another Neat Tool), which in English stands for "Another Neat Tool", which in English means

"ant") was created by James Duncan Davidson while transforming Sun Microsystems' Solar project into open source (specifically the implementation of Sun's JSP/Servlet engine, which would later be called Jakarta Tomcat).

Make

shell

Apache Ant is a tool used in programming to perform mechanical and repetitive tasks, usually focused on the build phase. It is similar to the "" used in Linux, but developed in Java; it has the advantage of not depending on the commands of each operating system, since it is based on XML configuration files and Java classes, being ideal as a multi-platform solution.

We can highlight aspects and/or functions that Ant is going to take care of:

- ✓ Compilation.
- ✓ Generation of documentation.
- ✓ Packaging.
- ✓ Execution, etc.

build.xml

It is used in most Java development projects and works from an assembly script, in XML format () which will be explained in more detail later; it is also easily extensible and integrates with many tools used by developers, for example the Jedit editor or the Netbeans IDE.

.Java

classpath

.class

Working without Ant implies a manual compilation of all the files (without a control of those that have been modified and those that have not), including the appropriate relatives, having the files mixed with the source code...; however, with Ant, in the end, you are nothing more than automating tasks, so that, in the end, with a single command, you can build your project from scratch, run unit tests, generate the documentation, package the program...

As limitations to be taken into account:

- ✓ Being an XML-based tool, Ant files must be written in XML.

`<javac>` , `<exec>` and `<java>`

- ✓ Most old tools like have bad default settings, values for options that are not consistent with the latest tasks.

- ✓ When properties are expanded in a string or text element, undefined properties are not posed as an error, but are left as an unexpanded reference.

To work with Ant you need:

- ✓ JDK version 1.4 or higher, as Ant is still a Java application.

- ✓ An XML parser. It doesn't matter which one, if you have downloaded the binary version of Ant there is no need to worry, because it already includes one.

ANT (Another Neat Tool)

What is it? It is a tool that allows you to automate the assembly process of

Web Apps

Assembly = build + deployment

Similar to the linux make tool

What is it for? It deals with:

- ✓ Compilation
- ✓ Documentation generation
- ✓ Packaging
- ✓ Execution...

Advantages It automates tasks, so that in the end with a single command:

- ✓ You can build your project from scratch,
- ✓ run unit tests,
- ✓ generate the documentation,
- ✓ Package the program...

It does not depend on the Shell commands of each operating system, since it is based on XML files and Java classes, being ideal as a multi-platform solution.

How does it work? It works from an assembly script in XML format called build.xml, defined based on project, targets, and tasks

- ✓ Project
 - ➔ One per file and contains targets
- ✓ Target
 - ➔ With a name and dependencies on other targets
 - ➔ Contains a set of tasks
- ✓ Tasks
 - ➔ Basic operations (javac, java, jar, etc.)

On this page we can find all the information that may interest us to start working with the Ant tool. <http://ant.apache.org/>

4.1.1.- Installation and configuration of Ant.

We are going to start from a machine with the Debian 6.0.1 operating system where we will perform the installation of Ant, first of all we check if we have Java installed, we can do it using the following command:

```
#java -version
```

remember that, as a requirement for the installation of Ant, a JDK version 1.4 or higher is essential.

Later we will proceed to download the Ant binary package, which we can download as follows:

```
#wget http://ant.apache.org/bindownload.cgi/apache-ant-1.8.2-bin.tar.gz
```

and once we have downloaded the binary file we unzip it using the instruction:

```
#tar -zxvf apache-ant-1.8.2-bin.tar.gz
```

```
apache-ant-1.8.2  " created to " /usr/local
```

Then we move the "" folder.

ANT_HOME and update the variable PATH

The only thing left is to create the variable.

ANT_HOME

/usr/local/apache-ant-1.8.2

✓ : Indicates the root directory of Ant installation, according to the instructions above this path would be: .

PATH

✓ : Defines the path for the system binaries; modifying this variable allows you to access Ant executables from any directory.

We can do this by adding to the "/etc/profile" the following content:

```
ANT_HOME=/usr/local/apache-ant-1.8.2/
```

```
PATH=$PATH:$ANT_HOME/bin
```

```
#source
```

and then, for the system to collect the changes made, we use the command:

```
/etc/profile
```

```
.
```

To check that ant has been installed correctly from a shell console we run the following command: `#ant` And we should get a message similar to:

```
Buildfile: build.xml does not exist! Build failed
```

so the ANT tool would be correctly installed and configured to perform its function in our machine.

In the following video we can see that it shows how to install the Ant package on a computer with Microsoft Windows 7 operating system.

http://www.youtube.com/watch?feature=player_embedded&v=bcY4ZF1jt4o

The first part of the video explains how to download the Ant package from its download website, on this page we can see several formats and revisions for the Ant package and, in our case, the .zip is selected and the download is carried out.

Then the downloaded .zip file is extracted in the folder that interests us, in this case "c:\kwit\apache-ant-1.8.2", where we see the folder structure that Ant contains, among others, bin, docs, etc, lib...

Once the package is installed, you go on to configure the application's environment variables; to do this, you go to "Control Panel, advanced configurations and environment variables", select the PATH variable and update its value with the path where Ant has been installed followed by \bin, that is, for this specific case it would be: "c:\kwit\apache-ant-1.8.2\bin" and, finally, a command interpreter is opened using the cmd command and, by means of the ant-version command, it is verified that the installation has been successful and that the application is still operational.

4.2.- The build.xml.

build.xml

As we have said, Ant is based on XML files, we usually configure the work to be done with our application in a file called , so let's see some of the tags with which we can form the content of this file.

Project

✓ : This is the root element of the XML file and, as such, there can only be one in the entire file, the one that corresponds to our Java application.

Target :One Target

✓ Your goal is a set of tasks that we want to apply to our application at some point. You can make some objectives dependent on others, so that Ant treats that automatically.

Task :One Task

✓ o Task is an executable code that we will apply to our application, and that can contain different properties (such as the classpath). Ant already includes many basic ones, such as compilation and deletion of temporary files, but we can extend this mechanism if we need to. Then we will see some of the available ones.

property : A property or property

BaseDir

build.xml

ant.java.version

prev.file

✓ it is simply some parameter (in the form of a name-value pair) that we need to process our application, such as the name of the compiler, etc. Ant already includes the most basic ones, such as for the base directory of our project, for the absolute path of the file, and for the version of the JVM.

Let's look at a simple example of a file `build.xml` :

```
<?xml version="1.0"?>
```

```
<project name="ProbandoAnt" default="compile" basedir="." >
```

```
<!-- global properties of the project -->
```

```
<property name="source" value="." />
```

```
<property name="destination" value="classes" />
```

```
<target name="compile">
```

```
<javac srcdir="${source}" destdir="${destination}" />
```

```
</target>
```

```
</project>
```

default="compile"

This simple file requires little explanation, we simply declare the project indicating the default action to be performed (), and indicate that the base directory is the current one

basedir="."

().

property

Then we indicate the source and destination directories in property tags (

name="source" value="."

and property name="target" value="classes").

Target

Finally we declare a call compile, which is the one we have declared as the default.

javac

Srcdir and Disconnect

\${source} and \${destination}

In this objective we have a single task, the compilation task, to which by means of the attributes we indicate the source and destination directories, which we collect from the properties previously declared with .

build.xml

The only thing left is to compile our code, so simply, being located in the directory where we have our , from an MS-DOS window or GNU/Linux terminal, we can do:

#[PATH_TO_ANT]ant

This works like this because we have declared compile as the default goal, although it could be another one so as a general rule we would put:

#[PATH_TO_ANT]ant nombre_objetivo

Ant is based on XML files, we usually configure the work to be done with our application in

a file called `build.xml` .

4.3.- The objective `.jar`.

To explain the content of this section we are going to do it through an example. First of all

We create a file `build.xml` at the root of our project and we define its name:

```
<project name="Project">
</project>
```

Targets

Ant, like other construction tools, is based on the concept of objectives or whose definition encompasses both the previous dependencies and the steps to follow to achieve them.

Init

We're going to start by defining a preparation target called that will be in charge of creating a classes directory where we'll save the ".class" files resulting from the build and the build directory for the final `.jar`. To do this, it is sufficient to include within `<project>` the following lines:

```
<target name="init">
  <mkdir dir="classes" />
  <mkdir dir="build" />
</target>
```

As we can see, the objectives are delimited with tags `<target>` and a name. Within them are listed the steps that must be followed to achieve the objective, in this case you have to create directories.

If we want to achieve the objective `Init` It is enough to carry out:

```
#ant init
```


Buildfile: build.xml

Init:

[mkdir] Created dir: /home/teacher/project/classes

[mkdir] Created dir: /home/profeosr/proyecto/build

BUILD SUCCESSFUL

Total time: 0 seconds

Compile

Classes

It's time to compile our project, let's define the goal. However, the compilation depends on the creation of the "" directory that is done in the previous objective. With this in mind, it is enough to include:

```
<target name="compile" depends="init">  
  <javac srcdir="src" destdir="classes" />  
</target>
```

Target

Src

Classes

The dependency is fixed in the declaration of the in such a way that its compliance is guaranteed before it begins. Our code is in the "" directory and the result of the compilation is taken to the "" directory.

```
<javac>
```

Important to note that this time we are using this is what Ant calls task. There are many predefined tasks.

.Jar

With our compiled project we are going to generate the one that we will distribute using a New target called `build` .

```
<target name="build" depends="compile">  
  <jar destfile="build/proyecto.jar" basedir="classes" /> </target>
```

`Compile` and the task is used `Jar`
Classes in the file `proyecto.jar` .

We check that there is a dependency that is responsible for packaging all the contents of the directory

Finally we'll include a new goal to clean up the entire environment, the goal `Clean` :

```
<target name="clean">  
  <delete dir="classes" />  
  <delete dir="build" />  
</target>
```

Removes working directories leaving the environment clean of the build process.
Summarizing

Our file is:

`build.xml`

```
<project name="Project">
```

```
<target name="init">
  <mkdir dir="classes" />
  <mkdir dir="build" />
</target>
<target name="compile" depends="init">
  <javac srcdir="src" destdir="classes" />
</target>
<target name="build" depends="compile">
  <jar destfile="build/proyecto.jar" basedir="classes" />
</target>
<target name="clean">
  <delete dir="classes" />
  <delete dir="build" />
</target>
</project>
```

4.4.- Deployment of a WAR file.

In Java EE architecture, web components and files with static content, such as images, are called web resources.

A web module is the smallest unit of a web resource that can be used and deployed. A Java EE web module corresponds to a web application, as defined in the Java Servlet specification.

In addition to web components and web resources, a web module can contain other files:

JavaBeans

- ✓ Server-side utility classes (beans (software component that has the particularity of being reusable and thus avoiding the tedious task of programming the different components one by one) for databases, shopping carts and others). Often these classes meet architecture.
- ✓ Client-side classes (applets (component of an application that runs in the context of another program, such as a web browser) and utility classes).

A web module has a specific structure. The highest directory in a web module's directory hierarchy is the application's document root. It's where JSP pages, client-side classes and files, and static resources like images are stored.

WEB-INF

The root directory of the documents contains a subdirectory called , which contains the following files and directories:

web.xml

Classes

- ✓ : The application deployment descriptor.
- ✓ : A directory that contains the server-side classes: Servlets components, utility classes, and JavaBean.

Tags

- ✓ : A directory that contains tag files, which are implementations of tag libraries.

Lib

- ✓ : A directory that contains the JAR files of the libraries called by the server-side classes.

WAR

WAR

Jar in a directory located in the format of a module, using the Ant

A web module must be packaged in a in certain deployment scenarios and when you want to distribute the web module. A web module is packaged into a web module by running the command or using the IDE tool of your choice.

JAR known as a web file (WAR

WAR differ from those files JAR , the name of the file WAR

A web module can be deployed as an unpackaged filestructure or it can be packaged into a file. Since the content and use of the files uses a

. WAR

. The described web module is portable, it can be deployed in any web container that complies with the Java Servlet specification.

WAR

To deploy a to an application server, the file must contain a run-time deployment descriptor. The deployment descriptor is an XML file that contains information such as the root context of the web application and the relationship of the portable names of the application resources to the resources of the application server.

Ant

There are a series of tasks that we can use for application management, among which we highlight:

- ✓ <deploy>: Deploys a web application.
- ✓ <start>: Launches a web application.
- ✓ <stop>: For one application.

- ✓ <undeploy>: Redeploys (uninstalls) an application.
- ✓ <trycatch>: Prevents a build from failing even if a task fails.

Various types of web application servers can be used in conjunction with the Ant tool, for example JBoss or Tomcat.

To deploy a WAR with the Ant tool, we open a terminal or command line window in the directory where the WAR has been built and packaged and run ant deploy.

Fill in the gaps with the right concepts:

In Java EE architecture, web components and files with static content, such as images, are called web resources . A web module is the smallest unit of a web resource that can be used and deployed.

The deployment descriptor is an XML file that contains information such as the root context of the web application and the relationship of the portable names of the application resources to the resources of the application server.

To deploy a WAR with the Ant tool, we open a terminal or command line window in the directory where the WAR has been built and packaged and run ant deploy .

5.- Tomcat's Web application manager.

Case Study

In the company BK programming they have a Tomcat web application server. Due to the options it provides, they have decided to deepen the operation of

the latter, but focusing on the administration of applications to be deployed from the web interface that Tomcat provides, the "Tomcat Web Application Manager".

catalina.sh

/Bin

Once started on the computer server the Tomcat by means of the script "" that is Find in The folder of the Tomcat installation directory, in our case

"/usr/local/apacheTomcat-6.0.32/", from a browser we can access Tomcat through the URL:

http://localhost:8080 yes

http://ip_servidor:8080

- ✓ we access it from the very machine on which Tomcat is running.
- ✓ if we access it from any other machine on the network.

Tomcat Manager

War

WebApps

Through the "" link we access the Tomcat Web application manager. This page allows you to deploy a project contained in an extension file, as we have already seen in the point "2.2 Deploying a web application" of this topic, or simply copy the folder it contains from the application to the folder found in the Tomcat installation directory.

Tomcat Manager

We go to and there we can see a list of the web applications that are available on the server. We can check, in our case, that if we have in the folder

usr/local/apache-Tomcat-6.0.32/webapps/ the application folder" Aplic_Web

" that we developed at the beginning of this topic, would already be shown in the list that the Tomcat application manager offers us, or simply Accessing since one

browser to the URL: `http://ip_servidor:8080/nombre_aplicacion` (in the generic case), for our case we can

Try `http://localhost:8080/Aplic_Web`.

If you are working as a system administrator in a company (let's say it isBK programming), in which you are in charge of managing, among others, a machine in which there is a Tomcat web application server.

How would you ask application developers to send you applications to be deployed on that server?

5.1.- Manager configuration.

Conf

All configuration files are located in the folder in the installation path of

Tomcat. We referenced this route earlier with the `CATALINA_HOME` environment variable, that is,

`$CATALINA_HOME/conf`

. In this path we find a folder named `catalina/localhost/` in

where the Tomcat web configuration is stored in two .xml files: `host-manager.xml` and `manager.xml`

.

#apt-get install Tomcat6-admin

To perform server administration since the `entornoweb`, we will install an additional package, `Tomcat6-admin`, we can do it by the statement:

and, to access the administration, it is necessary to create the "manager" role and a user with that role, for this we can follow the following procedure:


```
#nano $CATALINA_HOME/conf/Tomcat-users.xml
```

- ✓ We edit the Tomcat user file: .
- ✓ We add the lines establishing a content for and :

```
<role rolename="manager">
```

```
<user username="<user>" password="<key>" roles="manager"/>
```

We restart the Tomcat and, through the URL `http://ip_servidor:8080/manager/html`
Can

✓

Uninstall, reload, and install apps.

Admin , and from `http://ip_servidor:8080/host-manager/html`

- ✓ To enable the host-manager we would have to perform the same steps but by establishing the role we would have the service operational.

We can ensure Tomcat by establishing that access to this context is allowed only to the IP addresses of the computers from which the administrators operate, this can be configured

On file: `$CATALINA_HOME/work/Catalina/localhost/manager/context.xml` .

```
<Context path="/manager" privileged="true" antiResourceLocking="false" docBase="/opt/apachetomcat6/webapps/manager">
```

```
<Valve className="org.apache.catalina.valves.RemoteAddrValve" allow="127.0.0.1, direccion_ip1, direccion_ip2" />
```

```
</Context>
```

To access the Tomcat administration it is necessary to create the role " user/s with this role, for this it is necessary to edit the

Manager

", " of

Admin

" and

Users Tomcat

\$CATALINA_HOME/conf/Tomcat-users.xml .

5.2.- Connection to Tomcat's web application manager remotely.

An Apache-Tomcat server consists of 3 main components:

Catalina

✓ : is the Tomcat Servlet container. Implements Sun specifications for servlets and Java Server Pages (JSP).

Coyote

✓ : is the HTTP connector that supports the HTTP1.1 protocol for the web server or for the application container.

Coyote listens for incoming connections on a given TCP port and redirects the requests to the Tomcat engine to process the requests and send a response back to the client.

Jasper

✓ : is Tomcat's JSP engine; compiles the JSP pages in java code into servlets that can be handled by Catalina.

At runtime, any changes to a JSP file are detected and recompiled by Jasper.

Tomcat modes of operation can be:

1. Application Server:

- ✓ Tomcat needs a server that acts as a frontend (Apache, IIS...).
- ✓ Static content is served by the frontend.
- ✓ Requests to servlets and JSPs are redirected to Tomcat by the web server.

✓ It receives requests in specific protocols such as AJP that are sent by the frontend.

2. Standalone:

✓ There is no web server to act as a frontend.

✓ All content is served by Tomcat.

✓ Receive HTTP requests.

Coyote

Connectors are the components that provide the external interface to the server, specifically the HTTP1.1-based connector is the default connector for Tomcat. The connectors are defined in the file:

due to establishing security measures for web connections to the server, we can configure the following for an HTTP/1.1 connector with SSL:

```
<Connector port="8080" protocol="HTTP/1.1" maxTherads="150" scheme="https"
secure="true" clientAuth="false" sslProtocol="TLS"/>
```

scheme for the protocol, and Secure

where we see that the attributes have been established to establish that it is an SSL connector.

Fill in the gaps with the right concepts:

An Apache-Tomcat server consists of 3 main components:

Catalina

✓ : is the Tomcat Servlet container. Implements Sun specifications for servlets and Java Server Pages (JSP).

Coyote

✓ : is the HTTP connector that supports the HTTP1.1 protocol for the web server or for the application container.

TCP

Tomcat

Coyote listens for incoming connections on a given port and redirects the requests to the engine in order to process the requests and send a response back to the client.

Tomcat

Jasper

Catalina

✓ : It is the JSP engine of ; compiles JSP pages in java code into servlets that can be handled by .

Application Server Standalone

Tomcat's modes of operation can be and .

5.3.- Include Ant tasks in Tomcat.

Ant

As we have already seen above, it is a software construction tool that allows you to automate repetitive tasks in the process of compiling, linking, deploying, etc.

Ant

Tomcat defines a series of libraries that allow you to automate tasks such as deploying and retracting web applications, using .

To integrate the two previous tools we can follow the following operations:

Ant

✓ Download.

- ✓ Unzip the file.

ANT_HOME

- ✓ Configure the environment variables to point to the root of the distribution.

PATH to add the path to the directory <ANT_HOME>/bin

- ✓ Configure the .

<Tomcat_HOME>/lib/catalina-ant.jar in <ANT_HOME>/lib

- ✓ Copy the .

. WAR

To install a web application, Tomcat Manager is told that a new context is available by using the command `#ant install` that works both with files and if the path to the directory of the unpackaged application is indicated. It is necessary to note that the above command does not imply a permanent deployment; if Tomcat is restarted, previously installed applications will not be available.

Permanent deployment of web applications:

*. WAR

- ✓ It only works with .
- ✓ Unpackaged directories cannot be deployed.

*. WAR

- ✓ You get on the Tomcat and you start.
- ✓ Allows for remote deployment.
- ✓ A remote web container cannot access the local machine's directory.

`#ant deploy`

The command is used for the permanent deployment of applications, and for this it is necessary to:

Tomcat Manager is running in the location specified by the URL

✓ That the .

✓ The deployment of an application in the context specified by the path attribute and the location contained in the files of the web application specified with the war attribute.

We can establish the following example:

```
<deploy url="http://localhost:8080/manager"
path="mywebapp"
war="file:/path/to/mywebapp.war" username="username" password="password" />
```

The archive build.xml from an application called " Hello " for " Ant deploy " could be the following:

```
<target name="deploy" description="Deploy web application" depends="build">
  <deploy url="${url}" username="${username}"
    password="${password}"
    path="${path}" war="file:${build}/${example}.war"/> </target>
<taskdef name="deploy" classname="org.apache.catalina.ant.DeployTask" />
<property name="url" value="http://localhost:8080/manager" />
<property name="path" value="/${example}" />
<property name="example" value="hello" />
```

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 `wget` 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 `wget` 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
```

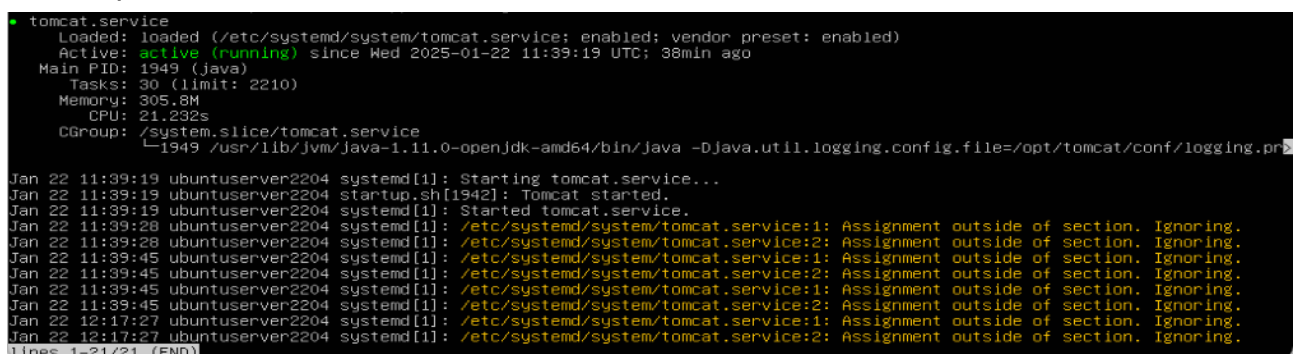
You can then start the Tomcat service by typing:

```
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 ubuntu204 systemd[1]: Starting tomcat.service...
Jan 22 11:39:19 ubuntu204 startup.sh[1942]: Tomcat started.
Jan 22 11:39:19 ubuntu204 systemd[1]: Started tomcat.service.
Jan 22 11:39:28 ubuntu204 systemd[1]: /etc/systemd/system/tomcat.service:1: Assignment outside of section. Ignoring.
Jan 22 11:39:28 ubuntu204 systemd[1]: /etc/systemd/system/tomcat.service:2: Assignment outside of section. Ignoring.
Jan 22 11:39:45 ubuntu204 systemd[1]: /etc/systemd/system/tomcat.service:1: Assignment outside of section. Ignoring.
Jan 22 11:39:45 ubuntu204 systemd[1]: /etc/systemd/system/tomcat.service:2: Assignment outside of section. Ignoring.
Jan 22 11:39:45 ubuntu204 systemd[1]: /etc/systemd/system/tomcat.service:1: Assignment outside of section. Ignoring.
Jan 22 11:39:45 ubuntu204 systemd[1]: /etc/systemd/system/tomcat.service:2: Assignment outside of section. Ignoring.
Jan 22 12:17:27 ubuntu204 systemd[1]: /etc/systemd/system/tomcat.service:1: Assignment outside of section. Ignoring.
Jan 22 12:17:27 ubuntu204 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.

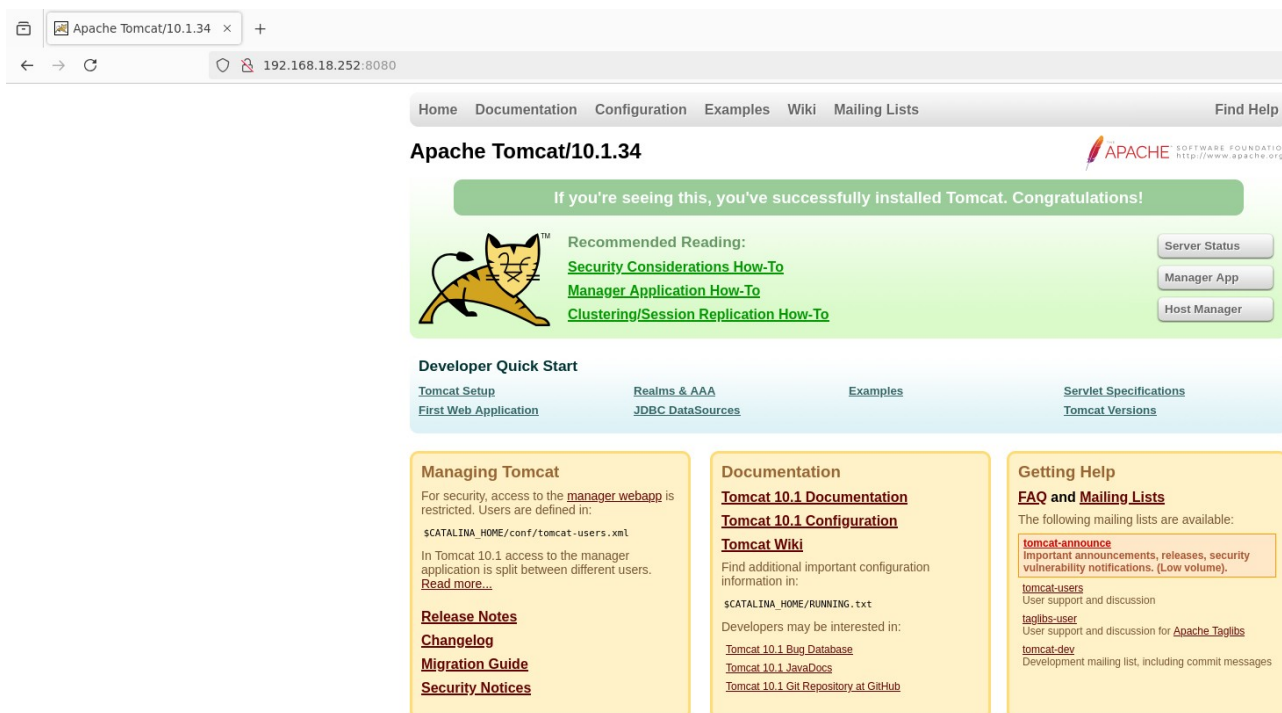
Tomcat uses port `8080` to accept HTTP requests. Run the following command to allow traffic to that port:

```
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
```

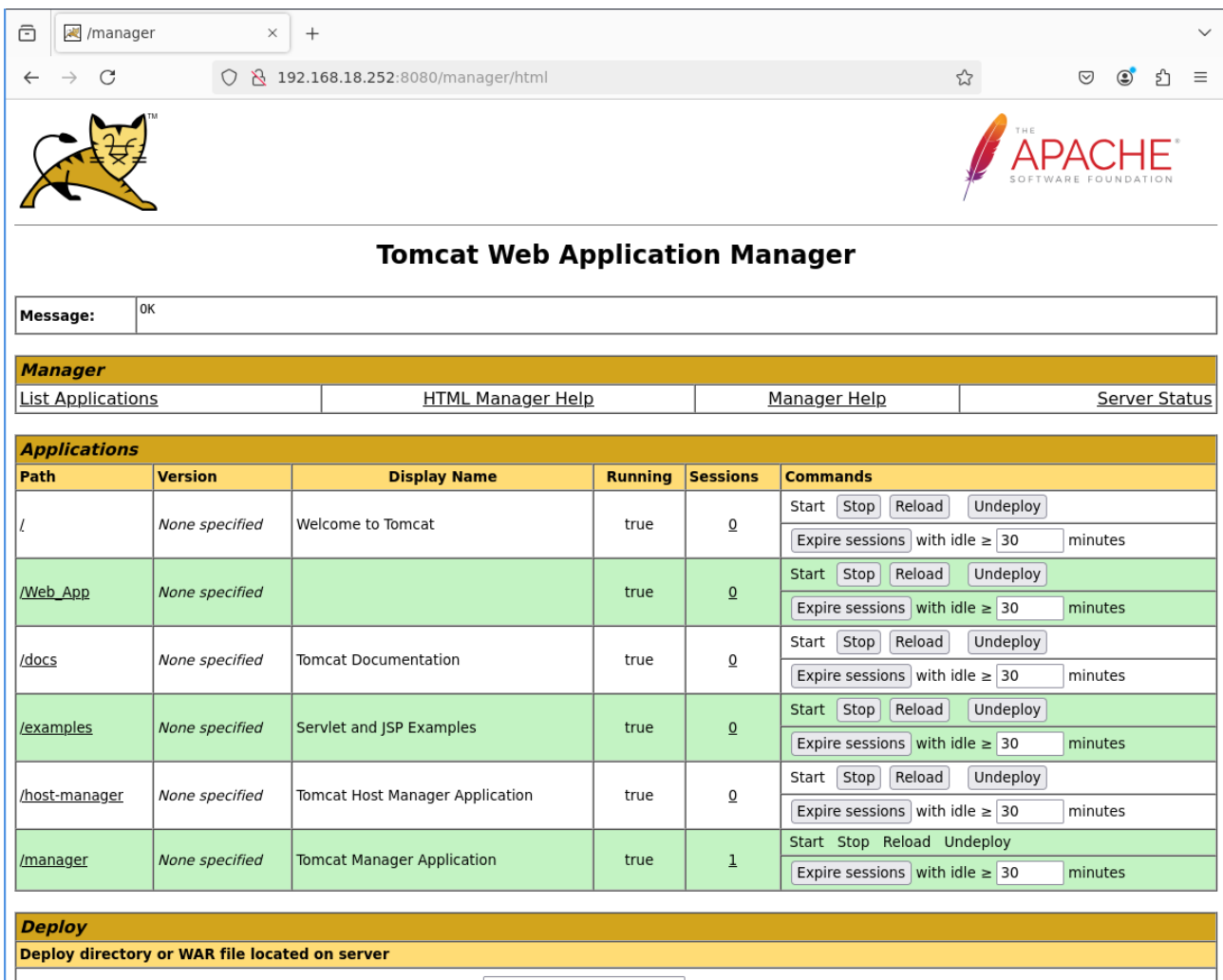
You'll see the default Tomcat welcome page:



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 |

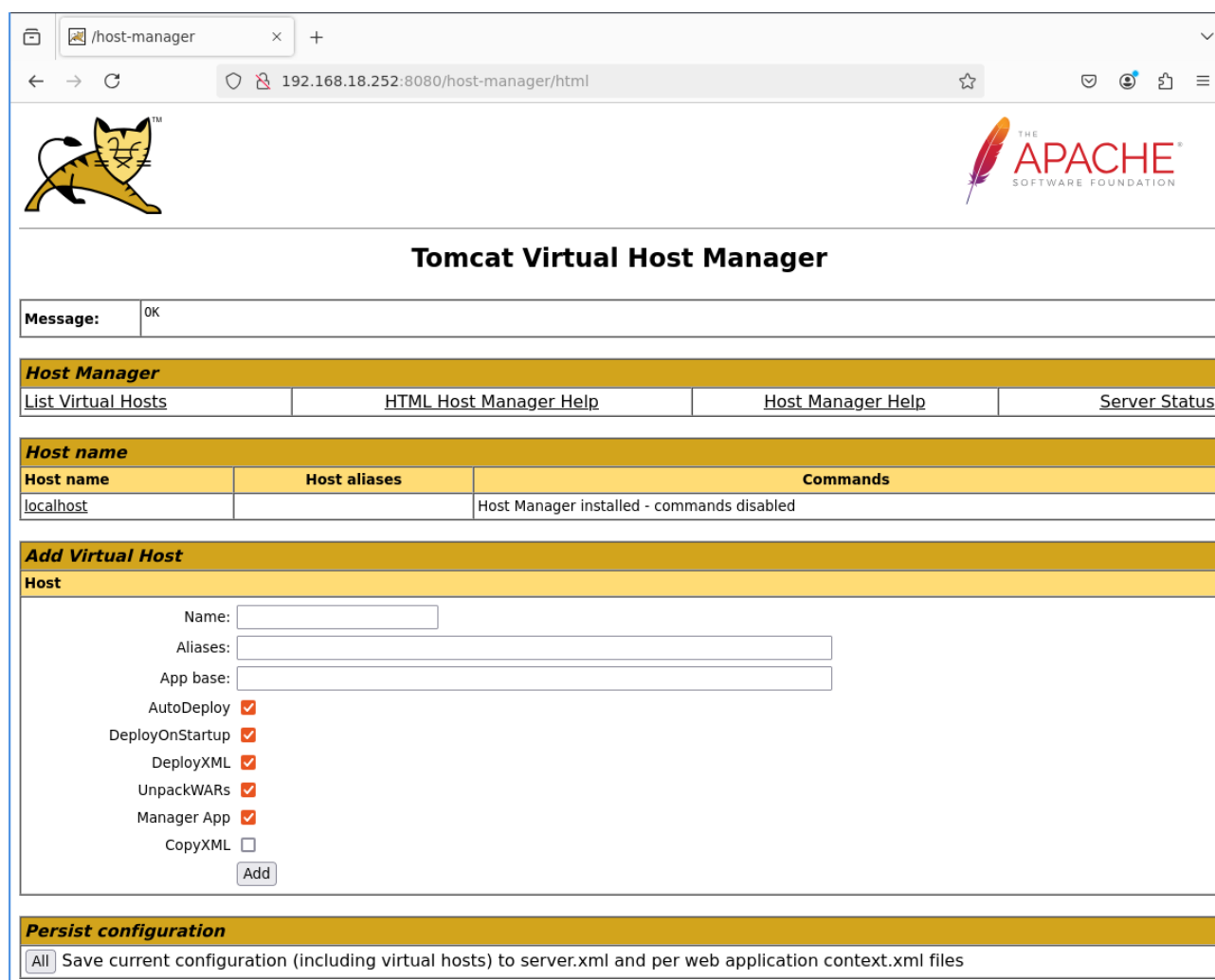
Deploy

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`.



The screenshot shows the Tomcat Virtual Host Manager web interface in a browser. The browser's address bar shows the URL `192.168.18.252:8080/host-manager/html`. The page features the Tomcat logo (a yellow cat) on the left and the Apache Software Foundation logo on the right. The main heading is "Tomcat Virtual Host Manager". Below this, there is a "Message:" box with the text "OK". A navigation bar contains links: "List Virtual Hosts", "HTML Host Manager Help", "Host Manager Help", and "Server Status". The "Host name" section displays a table with one entry for "localhost". Below this is the "Add Virtual Host" section, which includes form fields for "Name:", "Aliases:", and "App base:", and a list of checkboxes for "AutoDeploy", "DeployOnStartup", "DeployXML", "UnpackWARs", "Manager App", and "CopyXML". An "Add" button is at the bottom of this section. The "Persist configuration" section at the bottom has a button labeled "All" and a text description: "Save current configuration (including virtual hosts) to server.xml and per web application context.xml files".

| 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 ☐

Persist configuration

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>
-