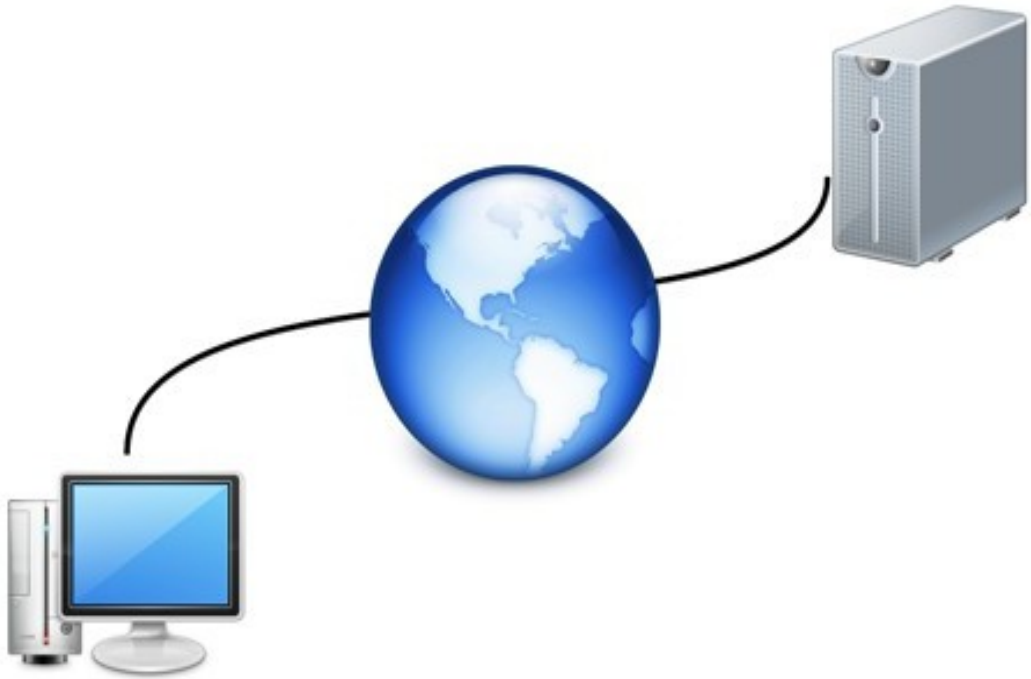
Topic 1: Implementation of web architectures

Web architecture and some models

A web application, or web in general, needs a structure that allows its access from different places (machines). This structure is called Web Architecture (actually this name is also given to the design of the whole structure).



The vast majority of web architectures today are based on a **client/server** model: an asymmetric communication in which one of the ends offers one or more services and the other makes use of it. This is the model on which the course will focus, but other models such as **P2P (peer-to-peer), B2B (business-to-business), etc.** should not be forgotten**.**

The term **service** is very broad and often confusing. For example, a website where we go to buy products can be considered a service in itself, but at the same time this service is composed of security services, session services, transaction services, and so on.

The structure of a current Web Architecture follows the following **model**:

1. A **client layer:** is usuall the browse We runnin at the

end user's computer. There are other more basic options, but nowadays the power and diversity of the existing browsers (as well as the fact that they are free) have relegated the other options to practical disappearance.

2. A Web server (**business layer**): The client layer can access different logic and procedures that exist in the business layer. Here the logic can be much more complex than in the previous layer. The components of this layer can range from simple HTML files to Java Servlets. There are many technologies that can be used in this layer: for example web scripting such as PHP, ASP or JSP to programming languages such as TCL, CORBA and PERL.
3. A **data layer**: It is composed of a storage system to access the data used to build the Web page. Generally it is a relational database manager (RDBMS) but it can be plain text files, XML files, etc. An increasingly used option is the creation of XML files from data stored in a database and its presentation by means of some of the options seen in the last course, such as XSLT.

The business layer may itself be divided into two parts if the system is sufficiently large or complex. It can be divided into a presentation layer and a business logic layer.

- The **presentation layer** is in charge of composing the pages by integrating the dynamic part into the static part. It also processes the pages sent by the client (e.g. form data). Some solutions for this sub-layer are Microsoft ASP or Java JSP. This part is usually performed by a web server.
- The **business logic layer** performs more complex operations. It corresponds to the implementation of an application server. It performs many types of operations, among which the following stand out:
    - Perform all operations and validations.
    - Manage workflow including control and management of sessions and data needed.
    - Manage all data access operations from the presentation layer.

In the case of using static web pages (they do not change depending on several variables) the data layer would not exist since these are incorporated in the markup files that will make up the web pages.

This assumption is becoming less and less common. Due to the introduction of dynamism in the pages, the structure previously seen has been noticeably altered:

- Web browsers are able to interpret different dynamic elements autonomously or through plugins (javascript, flash, etc.).
- Web servers can also interpret code to generate web pages. Thus, small programs can be introduced that alter the content or final appearance of a web page depending on different elements such as the user accessing the page or the information requested at any given moment. The web server needs some additional module to be able to interpret this code. Generally it is embedded in the web server itself for scripting languages or it is incorporated in a separate (application) server for more powerful languages. Some languages typically used in dynamic pages on the server are PHP, Python, Ruby or Java. These languages also allow access to the data layer and the interleaving of this data between the elements of the final page.

An example of the complete model would be composed of an Apache server and a Tomcat that connects to a database. An example of the simplified model would be a LAMP server.

Where the page dynamization code is executed will determine whether the programming language is client-side or server-side as will be studied in the other two computer content modules in this course of the cycle.

Although the Client/Server model is the most widespread, the W3C describes four models of web services architecture. We will see these concepts above because they do not have a great influence on the rest of the course:

1. The Message-Oriented Model s: It focuses on defining the messages, their structure, the way they are conveyed, etc. without reference to the why of each message or its meaning.

2. The Service-Oriented Model : It is the most widespread and the most complex of all. Although it uses messages, its definition is not based on them but on what is provided to the receivers of the service. The service is carried out by an agent and used by another agent, using messages to communicate. In this model metadata is used to form and agree on many aspects of the communication itself.

3. The Resource-Oriented Model : This model is based on the creation of resources and their accessibility through networks.

4. The Policy Model : is based on defining the behaviors of the agents that use the services by defining how they will access the resources.

These models are in many cases complementary and the definition of a web architecture can be done from different approaches. In a practical case, the most common is that our architecture is based on a standard definition.

A simple model for Web application deployment

Nowadays most of the information and logic of a business must be accessible from different places. This is where web applications come into play.

We all think of online stores as a model for a web application, but there are many others, such as an application for buying and selling assets between two banks in the business-to-business (B2B) sector.

You can immediately imagine that **security** is a key aspect in this type of application, but not the only one. Many times
The speed and **stability** of the communication and the service itself can be just as critical, if not more so.

When developing and implementing a web application, several factors must be taken into account. The first thing to do is to get a **general idea of the application** and the different **solutions** that can be used. The three layers must be taken into account. A very common mistake is to use only one set of technologies constantly. Of course, knowing a technology is a plus for its use, but many times we are going to implement a solution that is clearly better just because we have not considered using other technologies and learning how to use them.

The next aspect to consider would be the **cost**. How much is it going to cost us and what budget do we have.

These factors must be considered **before signing** any contract and even before giving an estimate, even if it is only a guideline.

For example, a company that sells industrial vehicles wants a web application to publish its sales data so that sales reps can access it remotely. We will need a database to store the different vehicles and their sales. We will also need a logic that keeps the whole system up to date and allows modifications. We will also need a client layer with authentication so that the different salespeople can access the system, consult and update the data.

After the evaluation, it can be decided not to undertake the project for many reasons. In addition to the costs already mentioned, it could be the case that we do not have the know-how or the infrastructure to carry out the project.

Once we are going to carry out the project we will have to put into practice the knowledge acquired during the study of this cycle, but it is often difficult to get an idea of how to do it; here we are guided with an example and this small guide can be very useful to organize the front-end.

What is a web application

It is an application that will run over the Internet. It will consist of two parts (at least) one on the server side and one that will run on the client machine in a web browser. Web applications fall under the higher concept of distributed applications. The server makes different resources available to the client. Examples of web applications are webmail, online stores, social networks, etc.

Phases of a web application project

Four phases can be considered in the project:

1. **Concept**: During this phase a clear and concrete idea of what the client wants must be obtained. We must also get a general idea of how it will be carried out and whether it is feasible or not. It is necessary to determine the real limitations that we can find. For example, the existing Internet connection in the area may not be sufficient to obtain the expected results. Another example of a problem may be that the technology required is too expensive. It is vital that at the end of this phase there is a documentation that clearly defines the limits and objectives of the project.

2. **Design**: This phase is focused on answering how we will make the application. We must specify the technologies (both software and hardware) that we will use and how they will communicate with each other. It is also necessary to determine the different modules we will use and their interfaces. It is very important to make a realistic project plan in which the tasks and responsibilities are divided and the following are calculated

the timing for each element as well as its sequence and dependencies. It is also necessary to obtain a functional specification detailing both the operation and the flow of the application.

3. **Development**: In this phase the project itself must be developed. It is very important to carry out unit and integration tests as well as to manage development documentation and version control.

4. **Testing and implementation**: When the project is completely finished, it is necessary to test it intensively before putting it into production. It is necessary to take into account both our application and its communication with other computer systems. The more the test system resembles the real one, the better. The last step is the installation and commissioning of the system. This is a critical moment.

This model is a simplification of those commonly used in software engineering.

> Research on the Internet the usual phases in a software development project and what are the different life cycles that are used. Discuss the findings in class with the help of the teacher.

A phase common to all IT projects and not included here is **maintenance**. This concept includes two parts. Service maintenance and bug fixes and enhancements. The first consists of making sure that everything continues to work and fixing possible errors and "crashes" in the service. The second is to extend the project. Both cases are usually considered as separate contracts and are therefore not included in the cycle.

The correct order for development is to start from the bottom up. That is, first the data layer, then the business layer and finally the presentation to the customer. Many times there is a temptation to do it the other way around. This is because the needs and objectives of the project have not been well identified or a concrete design has not been carried out. This is a mistake that will lead to many more modifications and errors in our application.

Web servers

A **web server** is a program or set of programs that provides a service over a network. Communication with a web server is usually done using the http protocol (hypertext transfer protocol), which is included in the application layer of the OSI model.

Often web server is also used as a reference to the **hardware** that hosts it, but this is inaccurate because the same hardware can host many other functionalities or it can be the case that the same hardware contains several web servers (sometimes simulated).

The purpose of a web server is to provide the means to enable communication between two or more programs or groups of software regardless of the technology used to create and operate each of them.

Currently the most widespread web server by far is **Apache**. That is why we will focus on it in this course. There are many other web servers. An easy way to consult the list and see a very general comparison is by visiting Wikipedia. In this other one you can read the main features of each one.

Web servers are included in a more general set of systems that is called a **distributed model** because the system is not unitary, it is distributed among different machines or sets of hardware. This model has to face some problems that must always be taken into account:

1. Latency and unreliability of the transport (e.g. network).
2. Lack of shared memory between the parties.
3. Problems arising from partial failures.
4. Management of concurrent access to remote resources.
5. Problems arising from updates of any of the parties.

What does each level of the OSI model stack do? What protocols are included in each layer? Identify them and explain very briefly what each one does.

Web Services

A **web service** is an abstract concept that must be implemented by an **agent**: a piece of software that sends, receives and processes messages while the service is the concept of what it does. The agent must only conform to the definition of an interface (two really, one inward (OSI stack) and one outward) and can be modified or even remade in another programming language without any problem. The design is made following modularity rules to allow these modifications.

It is of vital importance that the web service is well defined to enable communication between both ends. Therefore, there are many web service standards that allow a generic client (e.g. a web browser) to communicate with various services.

Generally the definition of a service is done in an API that specifies how to communicate with the service. The

process for using the service is as follows:

1. The client and the server must be aware of each other's existence. In the most common case it is the client that informs the server of its intention to use the service, but it can also be the server that initiates the contact. If it is the client that initiates, it can do so either by knowing in advance how to locate the server or by using the service discovery service (Web Service Discovery).
2. Both parties must agree on the parameters that will govern communication. This does not mean that they argue, only that the rules and protocols should be the same on both sides.
3. The agents on both sides start exchanging messages. The web server needs to compose the pages in case they contain multimedia elements and will even need to perform other actions if the page is created dynamically.

Alternatives

Before deciding to install our own web server, we must take into account that it is not always the best option. The first thing to know is what the customer wants. Depending on the size of the service we are going to provide and the importance of being able to control all aspects of the server, we may decide to use other possibilities.

On the other hand, the machine we need will have to have a lot of RAM and storage capacity and be able to support large workloads. The internet connection must also be powerful and we will need to hire a static IP address.

The first thing to consider is whether we are interested in having our own web server or hiring a web **hosting service**. Actually the term Web Hosting includes having your own server, but nowadays it is used to refer to renting space on another company's server. Generally this company is dedicated to it specifically. The advantages of this case are obvious: we do not have to worry about acquiring or maintaining the necessary hardware or software. In addition, the reliability of the service of a specialized company is usually very high.

There are cases in which there are even more specific technologies for our needs. It is becoming more and more common to have websites where the appearance does not change but the content is constantly updated. For these cases a **content manager** can be used. These allow the user to update the information on the site without the need for specific web knowledge. There are many web managers, some of them commercial and expensive but very powerful. There are also two widespread free alternatives: Joomla and Drupal.

Lately it is even considered a simpler alternative if you want an informative and quite simple website. Wordpress started as a platform for hosting blogs but increasingly includes more options and configuration possibilities allowing to carry out quite attractive pages. Currently it is a content management system that can even be installed autonomously as shown here.

http://www.frameworx.co.za/

http://www.webdesignerdepot.com/

http://designshack.net/

These are some examples of pages made in Wordpress.

> Look for comparisons between setting up your own server, hiring an external hosting or using wordpress. Which functionalities do you think are more important? Which one would you choose to create the website of a bakery? and of a municipal sports center? would you use the same system to create the website of the sports department of the Community of Madrid?
>
> Content management systems such as Joomla, Drupal or Wordpress are generating a lot of job offers nowadays. Search the Internet for the main features of each one and the reasons that may lead you to choose one or the other.

What do I need to set up a web server?

The first thing you need is a **machine** with a power capable of handling the requests it will process. This point is critical and difficult to manage because we do not know what the demand will be and it is often difficult to estimate the workload it will support. It is highly recommended that it be a dedicated machine or one that fulfills other functions related to the exchange of information on the Internet, such as managing e-mail or FTP.

It is also vital that the **operating system** we choose is stable. It makes no sense to choose an operating system that can easily become non-functional. It is desirable that it has some security and permissions control built in. The most common systems are different versions of UNIX (e.g. Solaris) but different Linux distributions are taking a strong position due to their low (or non-existent) cost. Windows is also a good option (especially its server versions) but it is more expensive than Linux and is more difficult to manage due to the diversity of configurations, options and functionality built in.

The next thing you will need to get is a **static IP address**. Of course it must be an internet address unless your goal is to set up an intranet. Our machine must be accessible from remote networks.

> Find out how to contract a static IP address and how much it costs.

The Internet names and addresses we know are based on a system called DNS that converts those addresses readable to us into IP addresses and vice versa. If our IP address changes frequently when someone were to access our page it would appear as unavailable even though all the rest of the system was working.

> What is DNS? Who manages it? How do I get a domain name?
>
> For alternatives such as Wordpress or external hosting, is a domain name necessary?

There is the possibility of working with a dynamic IP address through systems such as http://dyn.com/dns/ that always keep our address updated but it is only recommended (even on the page itself) for servers with very little load of connections and work.

The **24-hour Internet connection** is taken for granted, but it also implies certain problems such as the acquisition of network devices that can withstand this schedule without overheating or saturation.

The server **software** that we have already talked about and without which we could not work.

Configure our machine so that it is accessible but preventing the connection of strangers to critical parts of the system or that we do not want to publish. In short, security must be improved.

Installation and basic configuration of a web server: Apache

We are going to choose to install an Apache web server on a Linux operating system. It is one of the most widespread options and the possibility of obtaining this software for free greatly reduces costs, but it is not the only reason. Apache stands out from other servers for:

- It has a modular and highly configurable design.
- It is open source so there are many third-party extensions and tools.
- It works very well with Perl, PHP and other scripting languages.
- Versions are available for many operating systems including Windows, Linux and Mac OS X.

The Apache 2 family brought many improvements over the first version, especially in terms of flexibility, scalability and portability.

The most logical would be to install Apache on a server operating system (Ubuntu Server) but for didactic reasons, we are going to install it on a standard version with graphical interface. It is less secure so in a production system we should opt for the other option. Despite using a Linux with graphical interface we are going to install everything from the terminal window, so the steps can be applied to a server.

Apache can work autonomously, but nowadays (even more so when talking about web applications) it usually needs a database and more and more systems are using PHP. For this reason, AMP (Apache, MySQL and PHP) installers have proliferated, which install and configure the three systems to work together. This requires an in-depth knowledge of the configuration of the three elements in order to have a secure server. For PHP and MySQL security you will have to refer to their own modules.

XAMPP is the most widely used version for Windows and Linux. Depending on the system for which they are intended, these packages are generically called WAMP and LAMP. The installation of these packages is extremely simple.

NOTE: In previous versions of these notes, a guide to perform a manual installation of Apache was included. I have decided to remove it because this module is included in a cycle about development and not about administration, because it is too cumbersome, because we did not use it during the rest of the course and because it took too much time that we missed during the last topics. I consider that a student who finishes the course should be able to perform the manual installation following a guide with the Linux knowledge acquired in Operating Systems and the Apache knowledge explained in this module.

Checking and removing other versions

In most guides and tutorials on the internet you will see that they install MySQL first. This is because it makes installation easier, but what if we have to set up a database on a web server that is already running? We will install the most basic option on our web server first.

First of all, especially on a server that is going to publish to the web, we must make sure that we have full control. Many versions of Linux come with Apache packages **pre-installed** so the first thing to do is to **remove them** if they exist. This improves security because we make sure we are installing the latest version. Let's check together if there are any Apache, MySQL and PHP installations so we can leave a clean machine.

```
dpkg --list|grep -e httpd -e apache -e mysql -e php
```

What does the above line do?

In Ubuntu it is not installed by default. If it is preinstalled the above command will return something. If so we must check the dependencies to remove everything. If we see that *ii* appears next to the package it would be installed and we would have to remove it. Suppose it returns the following

```
     ii apache22                                .2.                    4-3build1Next generation,
scalable, extendable web se
     ii apache2-mpm-prefork                   2.2.                   4-3build1Traditional model for
     Apache
HTTPD
     ii apache2-utils2                       .2.4-3build1utility     programs for webservers ii
     apache2.2-common2                           .2.4-3build1Next          generation, scalable,
extendable web se
     ii libapache2-mod-php55                     .2.3-1ubuntu6server-side   , HTML-embedded scripting
langag
     ii libdbd-mysql-perl                      4.                       004-2A Perl5 database interface
     to
the MySQL data
     ii libmysqlclient15off5                 .0.45-1ubuntu3MySQL     database client library ii
     mysql-client-5.                         05.0.                   45-1ubuntu3MySQL database
     client binaries
     ii mysql-common                         5.0.                    45-1ubuntu3MySQL database common
     files
     ii mysql-server5                            .0.45-1ubuntu3MySQL      database server (meta
package dependin
     ii mysql-server-5.0                     5.0.                    45-1ubuntu3MySQL database server
     binaries
     ii php5-common5                          .2.                     3-1ubuntu6Common files for
packages built from the php
     ii php5-mysql                           5.2.                    3-1ubuntu6MySQL module for php5
```

We look for the apache processes.

```
     # ps -wef|grep apache2
```

We assume that it shows us

```
root      4006     1 0 09:41 ?          00:00:00 /usr/sbin/apache2 -k start
www-data  4025  4006  0 09:41 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  4026  4006  0 09:41 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  4027  4006  0 09:41 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  4028  4006  0 09:41 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  4029  4006  0 09:41 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  4140  4006  0 14:24 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  4141  4006  0 14:24 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  4142  4006  0 14:24 ?        00:00:00 /usr/sbin/apache2 -k start
root      4157  4125  0 14:36 pts/0    00:00:00 grep apache2
```

We stopped Apache

```
# apachectl stop
```

We look for it again to make sure it is no longer running.

```
# ps -wef|grep apache2
          root4162 4125 0 14:36        00:00:00 grep apache2
pts/0
```

Now we search for MySQL

```
# ps -wef|grep mysql
```

It shows us

```
      root      3857     1 0 09:41 ?         00:00:00 /bin/sh /usr/bin/mysqld_safe
      mysql          3897    3857    0  09:41 ?                00:00:00  /usr/sbin/mysqld  --basedir=/usr
--datadir=/var/lib/mysql   --user=mysql   --pid-file=/var/run/mysqld/mysqld.pid   --skip-external-locking
--port=3306 --socket=/var/run/mysqld/mysqld.sock
                root3898 3857 0 09:41 ?      00:00:00 logger -p daemon.err -t mysqld_safe -i -t mysqld
                root4355 4310 0 18:54 pts/100:00:00 grep mysql
```

We stop MySQL

```
      # /etc/init.d/mysql stop
       * Stopping MySQL database server mysqld      [ OK ]
```

And we found that it is no longer there.

```
      # ps -wef|grep mysql
                root4395 4310 0 18:55        00:00:00 grep mysql
      pts/1
```

PHP must not be stopped.

We eliminated all the packages that appeared.

```
      # 3 remove php5-mysql libapache2-mod-php5 php5-common
      # apt-get remove libdbd-mysql-perl mysql-server mysql-server-5.0 mysql-client-5.0
```

```
# apt-get remove libmysqlclient15off mysql-common
# apt-get remove apache2 apache2-mpm-prefork apache2.2-common apache2-utils
```

Now that we have seen how to do a manual installation and configuration of Apache 2, we are going to do another more standard installation to continue. In addition this installation will allow us to have a clean version.

Installation

Before installing something it is important to have the package list updated (this command does not install anything, it only updates the package list).

```
sudo apt-get update
```

then update the entire machine (we install the most recent versions of the installed machines)

```
sudo apt-get upgrade
```

and then we install

```
apt-get install apache2
```

The first thing we must do is to see that Apache is now installed in

```
cd /etc/apache2/
```

There are different versions of Apache 2, so it is interesting to check which one you are using.

```
apache2 -v
```

which will show us something like this

```
Server version: Apache/2.4.7(Ubuntu)
Server built:   Jul 22 2014 14:36:38
```

> Configure and install Apache in your virtual machine following the steps indicated and test the access from the virtual machine and from the host machine. Discuss the results in class.

We can see that there is already an apache2 file in /etc/init.d/ that causes Apache to start every time we turn on the machine.

```
sudo gedit /etc/init.d/apache2
```

It shows us a very complex content but we can take a look at it knowing that it is based on establishing what to do before several commands (the most important ones are start, stop, restart and graceful).

Take a look at the script without going into details, what is graceful, what is it used for here?

There are many ways to install Apache. You can choose modules, configuration and paths. We are going to use one that uses DSO (Dynamic Shared Objects). This is a configuration of Apache that allows to add modules later without the need to recompile the whole server and by default in the automatic installation is already activated. For our course it is vital. The only downside is an almost imperceptible decrease in performance so I consider highly recommended to use this method.

Now we open the web browser and load the page to see that everything went well. We write http://localhost and we should find the page that says "It Works! In the Ubuntu version this page has also been modified to explain the organization of the Apache configuration in this distribution.

In the standard distribution all the configuration goes in a single file but it can be distributed through calls to external configuration files. The configuration we find here is quite logical and we will see that the relationship between the two forms is easy to modify and adapt.

We can also place some HTML file and check that it is shown. To do this we will go to http://localhost/nombre_archivo.html

Configure and install Apache on your virtual machine following the steps below and check the access from the virtual machine and from the host machine.

Although the spirit of a server is to be active as long as possible, it is important to be able to stop, start and restart it in certain situations. All these options are explained in the Apache documentation. Many of these documentation pages can

I prefer to link to the English versions as they will always be up to date, which may not be the case for other languages.

> We have talked about the graceful option before. What happens if you use "apachectl -k graceful", is it logical, what is the difference with using "graceful-stop"?
>
> How should the server be stopped for a routine maintenance operation? What if we want to load some modifications to the configuration files?

Installing a server with LAMP

Now we are going to start with a **new virtual machine** to install a server with Apache, MySQL and PHP since as we have commented it is the most common option. Neither MySQL nor PHP are necessary to set up a web application server.

This time we are going to do it with the easiest option to see different installations. In the next chapter we will see the necessary configurations to secure our server.

For more security aspects in MySQL and PHP see the modules of the cycle about it. The first

thing is to download and install tasksel.

```
sudo apt-get install tasksel
```

Then we directly install the whole package and follow the instructions.

```
sudo tasksel install lamp-server
```

To test Apache we simply open the browser and query localhost.

To test PHP, we create a test.php file containing only "<?php phpinfo(); ?>". To test MySQL we

can use any method: connect, install PHPMyAdmin, etc. If we want to install PHPMyAdmin we

write

```
apt-get install phpmyadmin
```

and then we load http://localhost/phpmyadmin

The directory in which Apache is installed is

```
/etc/apache2
```

Copy a new virtual machine and install the LAMP server. Test the correct operation of everything.

Application Servers

An application server is a software package that provides services to applications such as security, data services, transaction support, load balancing and management of distributed systems.

The term was coined for servers of the Java platform in its Enterprise Edition version, but nowadays it is extended to many other technologies.

We will focus on Tomcat, a Java application server created by Apache. There are many others such as .NET integration on Microsoft servers, PHP integration on a server to have PHP application servers, Zend Server, also for PHP, Barracuda, IBM's WebLogic, etc.

Apache Tomcat is an application server created to host Servlets and Java Server Pages (JSP). Tomcat is free and open source but has nothing to envy to other commercial solutions. The version we will use is version 7.

The operation of an application server requires a web server. They often come in the same package, but they are really two distinct parts.

When a client makes a request to the web server, the web server tries to handle it, but there are many elements it does not know what to do with. This is where the application server comes into play, which offloads the web server from handling certain file types, in our case servlets and JSP.

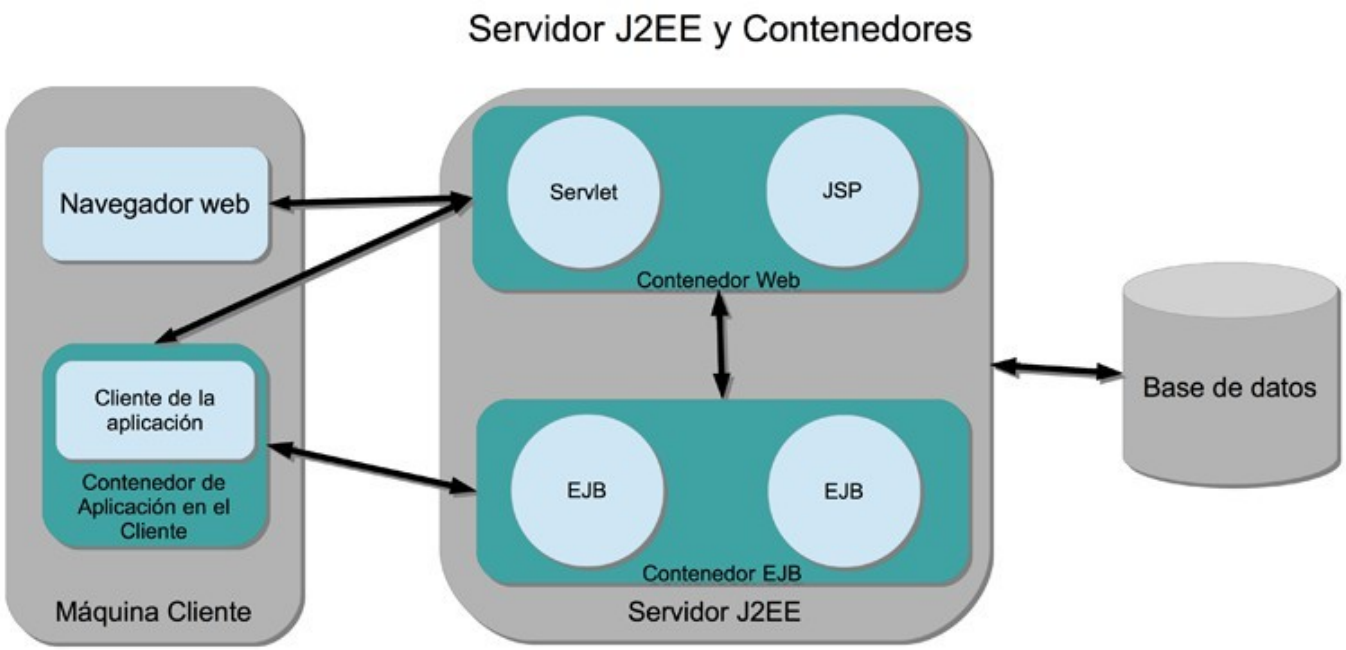
If a client makes a request to the server asking for a JSP, it arrives at the Web server, which reads an XML file provided by the application server and determines that the file will be managed by the application server.

The XML file also includes the address of the application server and the web server sends the request to it via HTTP.

Containers

The term *container* is another rather ambiguous term, as was the case with *server*. In many cases it is used to refer to the application server itself and even to the web server. However, the most widespread meaning is another one.

Keep in mind that the concepts we are talking about came mainly from J2EE.



Containers in application servers are a way to isolate the execution of each application or each **instance** of an application from other instances and other applications. For each execution they provide security, transaction support, remote connection and the management of the resources required for the execution of the application.

In reference to the previous image, some concepts should be specified:

- The J2EE server is the program that provides EJB and Web containers.
- The Enterprise JavaBeans (EJB) container is responsible for the execution of EJBs.
- The web container handles the execution of servlets and JSPs.
- The application client container is responsible for the execution of the application components on the client machine.
- The applet container is responsible for executing the applets on the client. It is composed of a web browser and a Java plugin.

Tomcat containers are called Catalina.

Tomcat

Tomcat is an application server that can run by itself. In fact it is able to process HTTP requests and serve HTML files quite efficiently, but not as well as Apache does. Generally if what we want is a web server with additional functionality it is best to have both servers working together, but for cases where almost everything is going to be java logic with Tomcat working autonomously is enough.

In this first approach to Tomcat we will install it in a new virtual machine. Later we will see how to integrate it with Apache so that each one is in charge of doing what it manages better. In that case, Apache will receive all the requests and will send to Tomcat what corresponds to it.

Only one instance of Tomcat can run in a Java virtual machine.

> What does the Java virtual machine do, and how is it different from virtual machines created with programs such as Oracle Virtual Box or VMware?

Installing Java

There are two versions of Java widely used in Linux environments. One is the official Oracle version and the other is called OpenJDK and is an open source version.

OpenJDK

This version is highly recommended when it is to be used with other open source or free software systems so that we **we will use this installation**.

We update the repositories and install Java

```
sudo apt-get update
sudo apt-get install openjdk-7-jdk
```

we check the Java version

```
java -version
```

which should display something like

```
java version "1.7.0_65".
OpenJDK Runtime Environment (IcedTea 2.5.2) (7u65-2.5.2-3~14.04)
OpenJDK 64-Bit Server VM (build 24.65-b04, mixed mode)
```

Java environment variables must be set

```
sudo gedit /etc/environment
```

And we add at the beginning the Java installation paths: the first one is the directory where java and javac are and the second one is the jre directory.

NOTE: Please note that the following are two examples, but the folder names in the paths depend on the architecture of the machine on which the system is running.

```
      JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386
      JRE_HOME=/usr/lib/jvm/java-7-openjdk-i386/jre
```

o

```
      JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64
      JRE_HOME=/usr/lib/jvm/java-7-openjdk-amd64/jre
```

And in the same file we add at the end of the path

```
      :$JAVA_HOME:$JRE_HOME
```

The entire file should look something like this:

```
      JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386
      JRE_HOME=/usr/lib/jvm/java-7-openjdk-i386/jre
      PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:$JAVA_HOME:
$JRE_HOME"
```

Although it is not strictly necessary, we restart the virtual machine.

Oracle Java

Tomcat is an application server programmed in Java so before installing Tomcat, we must have Java installed and running. The first thing to do is to add some repositories containing the Java installer

```
sudo add-apt-repository ppa:webupd8team/java
```

We update the software in the repositories with

```
sudo apt-get update
```

To then install Java

```
sudo apt-get install oracle-java7-installer
```

We can check that we have the correct version running

```
java -version
```

Something similar to this should appear on the screen:

```
java version "1.7.0.0_05".
```

```
Java(TM) SE Runtime Environment (build 1.7.0_05-b05)
Java HotSpot(TM) Client VM (build 23.1-b03, mixed mode)
```

Finally, we can check the correct operation by going to http://www.java.com/es/download/installed.jsp in the web browser. Now we have to set the environment variables so that Tomcat can find Java. To do this we edit the *environment* file

```
sudo gedit /etc/environment
```

And we add at the beginning the Java installation paths: the first one is the directory where java and javac are and the second one is the jre directory.

```
JAVA_HOME=/usr/lib/jvm/java-7-oracle/
JRE_HOME=/usr/lib/jvm/java-7-oracle/jre/
```

And in the same file we add at the end of the path

```
:$JAVA_HOME:$JRE_HOME
```

The entire file should look something like this:

```
JAVA_HOME=/usr/lib/jvm/java-7-oracle/
JRE_HOME=/usr/lib/jvm/java-7-oracle/jre/
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:$JAVA_HOME:
$JRE_HOME"
```

Although it is not strictly necessary, we restart the virtual machine.

Install Apache Tomcat

NOTE: Although Tomcat is on version 8 stably, it is not yet the most widespread version and has not been added to the standard Ubuntu repositories. Also for the purpose of this course it is not necessary for us, so I will keep Tomcat 7 as the version of the course notes one more time. If anyone wants to try Tomcat 8 you can find installation instructions on the Internet like these or these others.

It is necessary to have Java installed and configured for the correct operation of Tomcat. The steps are the same as we have already

seen. Now we install Tomcat

```
sudo apt-get install tomcat7
```

NOTE: In some cases, Tomcat will not work until we configure the environment variables as explained below. Now we can open a web

browser and access http://localhost:8080.

It shows the following screen in which the following paragraph is very important:

Tomcat7 veterans might be pleased to learn that this system instance of Tomcat is installed with CATALINA_HOME in /usr/share/tomcat7 and CATALINA_BASE in /var/lib/tomcat7, following the rules from /usr/share/doc/tomcat7-common/RUNNING.txt.gz.

In which we indicate the location of two very necessary environment variables in the configuration and use of Tomcat. In the manual installation, both would point to the directory where we unzip Tomcat. Let's set them as environment variables:

```
sudo gedit /etc/environment
```

in it we add (after the Java ones)

```
CATALINA_HOME=/usr/share/tomcat7
CATALINA_BASE=/var/lib/tomcat7
CATALINA_BASE=/var/lib/tomcat7
```

And we add them to the path, it would look like this:

```
JAVA_HOME=/usr/lib/jvm/java-7-openjdk-i386
JRE_HOME=/usr/lib/jvm/java-7-openjdk-i386/jre
CATALINA_HOME=/usr/share/tomcat7
CATALINA_BASE=/var/lib/tomcat7
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:$JAVA_HOME:
$JRE_HOME:$CATALINA_HOME:$CATALINA_BASE"
LANGUAGE="es:en"
LANG="en_US.UTF-8"
LC_NUMERIC="en_US.UTF-8"
LC_TIME="en_US.UTF-8"
LC_MONETARY="en_US.UTF-8"
LC_PAPER="en_US.UTF-8"
LC_IDENTIFICATION="en_EN_US.UTF
-8" LC_NAME="en_US.UTF-8"
LC_ADDRESS="en_US.UTF-8"
LC_TELEPHONE="en_US.UTF-8"
LC_MEASUREMENT="en_US.UTF-8"
```

CATALINA_HOME indicates the Tomcat installation directory.

CATALINA_BASE indicates the directory of a Tomcat instance. If we have more than one instance, CATALINA_BASE will be different for each one. In some installations (but not in this one) both environment variables point to the same directory.

We restart the machine so that the variables are loaded.

The file that Tomcat loads by default is */var/lib/tomcat7/webapps/ROOT/index.html*

Tomcat is installed in */etc/tomcat7*

As we can see this version of Tomcat already installs it as a service and makes it start automatically when the machine is turned on. The file where this is configured is much more complex than the one we did in the manual installation; you can consult it at

```
gedit /etc/init.d/tomcat7
```

Installing additional packages

We already have Tomcat installed and running, but as we could read in the default Tomcat home page, neither the documentation, nor the examples nor the administration application have been installed. On a production server this may be the correct configuration, but for our didactic purpose it is highly recommended to install the packages.

```
sudo apt-get install tomcat7-docs
sudo apt-get install tomcat7-examples
sudo apt-get install tomcat7-admin
```

Now we can access each of them through the corresponding link on the home page.

Install Tomcat and additional packages on a new virtual machine.

I have already commented before how it has been appearing in the technology news that Java is insecure on the net and that it should be disabled in browsers. This may seem like the end of Java, but... disable Java in your browser and try running the examples we just installed and now what?

If you click on the link that appears to see the code of an example you will see that it contains code but if you show the code of the page resulting from executing the example in the browser you will see that it does not, what does this indicate?

NOTE: As in Apache, there was a section on the manual installation of Tomcat that I have decided to remove for the same reasons.

Tomcat users

The first thing to do is to configure Tomcat users. To do this we must edit the *tomcat-users.xml* file that is in the *conf* directory.

```
sudo gedit /etc/tomcat7/tomcat-users.xml
```

What we are doing is adding an administrator user so the file, in the final part should look like the following, with the user name and password that we want, of course.

```
     <tomcat-users>
          <role rolename="manager-gui"/>
          <role rolename="manager-script"/>
          <role rolename="manager"/>
          <role rolename="admin-gui"/>
          <role rolename="admin-script"/>
          <role rolename="admin"/>

          <user username="sergio" password="sergio" roles="manager-gui,admin-gui,manager,admin,manager-
script,admin-script"/>
     </tomcat-users>
```

With this user and password that we have already created we can load the manager, from the Tomcat home page or directly from http://localhost:8080/manager/html.

It is recommended to test Tomcat examples to see if everything goes well by following the link on the home page or at http://localhost:8080/examples/.

Starting and stopping Tomcat

To start Tomcat we will use

```
     sudo /etc/init.d/tomcat7 start
     sudo /etc/init.d/tomcat7 stop
     sudo /etc/init.d/tomcat7 restart
     sudo /etc/init.d/tomcat7 restart
```

depending on what we want to do.

If we wanted it to work as a service, it would be:

```
sudo service tomcat7 start
```

and to stop it

```
sudo service tomcat7 stop
```

Web applications

A web application differs from a standard application in that it is accessed through a network such as the Internet or an intranet for example. In many cases it is an application that is written in languages supported by web browsers (e.g. Javascript + HTML) and needs a web browser to run.

The simplest example of a web application that comes to mind is a program that allows access to a company's data from outside the company. Here the main problem with web applications is already evident: **security**. The problem is often the need to find a compromise between security and efficiency in the application; too much security can slow down usage, increase traffic, etc.

The main **disadvantages** of this type of applications are those derived from the use of a network and the simultaneous access of several (sometimes many) users. The main **advantages** are being able to use a web browser as a client (something that all computers nowadays have) and the simplification of updates by not having to update the computers one by one. Let's go into more detail:

Advantages:

- No complex distribution, installation or update of the application is required. Simply the use of a compatible browser will allow us to use the application. Many times the application is created for a single web browser which in my opinion is a mistake. It is not necessary to cover the whole spectrum but giving at least two or three options would be advisable.
- No particularly powerful client machines are needed. Virtually any computer today is capable of running a web browser. If the application is heavier on the client, slightly higher resources may be required.
- They are easy to integrate with other server functionality such as e-mail.
- Generally, they eliminate problems arising from the use of different computer platforms (architectures, operating systems, etc.).

- The use of HTML5 greatly increases the functionality that can run natively in a web browser.

Disadvantages:

- Generally, the user interfaces of web applications are less intuitive and have a worse behavior than the classic ones.
- Web technologies are very dynamic and changing so we can use some functionality that disappears or is drastically modified forcing us to redo the interface (Flash in the not too distant future?).
- The absence of standards in "office" files can make it difficult to share data and information.
- They are totally dependent on the correct functioning of the network (Internet and/or intranet).
- From a user's point of view, the privacy and security of your data is a concern. One of the first examples of this is webmail, but nowadays all Google applications and many others such as Facebook control absolutely everything you do so the lack of privacy is remarkable.

> The list of advantages and disadvantages is a bit ambiguous. For each case think about what the statement implies and try to find an example that is an exception.

As we have already mentioned before, one of the main concerns when developing a web application must be security. In many cases, both **critical company information** and **users' private data** must be protected.

When implementing the security of a web application we must take into account five areas:

- User **authentication**: the use of an effective method to ensure that authorized users are logged in and to make it difficult to impersonate them is paramount. Is the application open to everyone? Can users register themselves?

- The **authorization** of each user: often there must be different types of users and not all users must be able to access all data or perform all operations (query, insert, modify or delete) on the data. The identification and creation of different roles is an aspect to be taken into account from the early stages of application design.
- **Resource management**: it is necessary to protect data both when it is in the database and when it is in transit between the client machine and the server. Direct remote connections (on an open network) to the database are generally avoided. In addition, encryption of data both in the database and in communications is very important. Data is used as an example of a resource, but is applicable to other types.
- **Data entry**: another aspect to take into account is what the user can type at each data entry point. The more we limit the characters and type of input that users can make, the easier it will be to prevent people from accessing unwanted points in our application and even other applications on the same network.
- **Audits and logs**: In order to control and correct security breaches it is very important to be able to know what has happened. Therefore, we must have methods of recording (e.g. *log* files) everything that happens in our application. The size of the files should allow us to review situations from a long time ago, since it often takes time for a security hole to be detected.

> Although we do not know how to deal with each problem in detail, we should already have the knowledge to propose solutions for the sensitive areas we have just discussed. Discuss a scenario that controls the problems generated in each area.

To control all these problems there are two basic **recommendations**:

- **Testing**: the more we test an application, the fewer bugs it will have. If the application is not very small, it is practically impossible to avoid all security holes but we must try to minimize them. The ideal is to have a person or a team

who know what they are doing trying to attack the application. This is why many hackers have ended up working for very powerful companies.

- Use a **Framework**: If we have a team, we do not want each member to "wage war on his own". All members should use one or more common methods to manage the security of the application. A common framework will greatly improve the security of the application. A framework for web applications should take into account the following aspects: data persistence, session management and user authentication, security, use of caches, use of templates (with data types, etc.) and include an administration interface.

Although it is beyond the scope of the course, in this page you can consult these aspects about web application frameworks and see a list with many of them.

Structure

The structure of a web application is **equivalent to that of a web server** as we saw at the beginning of the topic. The application must work at all levels so we will have a layered structure.

The most common is in three layers (although the middle layer may be subdivided as mentioned above): presentation, application and data.

Another way to look at a web application is in two layers: the client and the server. Depending on where most of the work is done the application will be either client or server heavy. Nowadays, with the proliferation of devices with low capacity and power (mobiles, tablets, netbooks, etc.) there are many heavy applications on the server, but when the application requires a lot of logic there is a tendency to distribute the work on the clients to improve performance.

The structure of a web application is also used to refer to the **directory layout** of all the elements that make up the application. Although we will return to this point later, it is important to emphasize that the directory structure we use must be contained within a starting point and must not reveal aspects about the distribution of the rest of the machine.

Deployment descriptor

The deployment descriptor of a web application is a configuration file in the container or application server. In J2EE this file is written using XML syntax. It describes how a component, module or application should be deployed by specifying aspects such as security, container options and configuration aspects.

There is also a deployment descriptor specific to each web server. For example in Tomcat, this other descriptor is located at <TOMCAT_HOME>/conf/web.xml. In the case of the installation seen above it is in /opt/tomcat/apache-tomcat-7.0.29/ conf/web.xml.

If we are using a development environment (for example Eclipse) the deployment descriptor is created by the environment itself, but it is always convenient to check it to make sure that it uses the options we want.

For J2EE web applications it must be called web.xml and located in the WEB_INF directory in the root directory of the application. The schema to which the deployment descriptor must conform can be consulted in the corresponding java page.

We can see a good example of web application deployment here. Note that if you click on a tag it will show you a brief explanation of its use.

More complete examples after installing Tomcat can be found in the following paths:

- <TOMCAT_HOME>/webapps/jsp-examples/WEB-INF/web.xml
- <TOMCAT_HOME>/server/webapps/manager/WEB-INF/web.xml

We are going to project the page seen before and discuss about the structure and different elements of the display descriptor that we are given as an example. To do so, determine beforehand what each of the second-level tags (the children of <web-app>) indicates