# Web Hosting

# The future is www

- A web server must respond to a large number of requests at the same time. It should also provide security and reliability. Linux and UNIX systems are best suited for this task because of the way they are designed.
- With the introduction of Web 2.0, and modern technologies like AJAX, HTML5 and CSS3, a lot of applications now are using the web as a service-platform
- For example, most enterprise solutions like Web Logic, Web Sphere, OFSSA, among others are using web programming languages like Java combined with a web application server without the need of a sophisticated client, only a browser (thin client).
- Cloud services is a relatively new buzz word that takes advantage of this: serve applications on a central, powerful web server and let your clients access it from anywhere on any Internet-aware device (browsers, smartphones, tablets, smart TV sets…etc).

# The HTTP protocol

- HTTP refers to Hyper Text Transfer Protocol. It is the web "language". Any request made to a website is done through HTTP (FTP is another protocol used exclusively for file transfer).
- The GET request is one of the most common HTTP request types. It simply asks the web server to fetch an HTML (Hyper Text Markup Language) document. In its simplest form, the web server examines the URL (Uniform Resource Locator), and serves the requested document. For example: http://www.linuxadmin.dev/training/index.html will grab a document with the name of index.html that is supposed to exist in a folder called training under the root directory of the web server.
- A GET request will receive a response that contains both the HTML of the requested document and some "metadata" about the document such as the type, length, and last modification date.

# Dynamic content now and then

- In early days of the web, the concept of "dynamic content" started to emerge. For example, a forum that accepts content (posts) from a user, saves them in a database and later displays them on request cannot work with mere HTML, a programming language like Perl or PHP must be used here.
- Back then, CGI (Common Gateway Interface) was used as an interface between the webserver and the programming language. But this posed an additional performance penalty as the webserver had to create a separate process for each new request handled by the script.
- Later on, plugins were introduced. They enabled a language like PHP or Python to be embedded inside the webserver itself. This enhanced performance because the webserver will not fork new process for each request.
- When a language like PHP is configured, the webserver examines the file extension of the requested file. If it is one of the PHP extensions (*.php, *.php5, .phtml …etc) the webserver does not serve the file as it is; however, it *parses* the file using the PHP interpreter to generate the approroate HTML content to be served.
- Recently a module called FastCGI was introduced. It acheived even more performance enhancement by letting the script (PHP for example) to start once and keep running in order to serve multiple requests.

# Scaling and load balancing

- Careful planning should be made before deploying a heavy-traffic web site/application
- Not only the bandwidth should be taken care of, but also, more importantly the machine physical resources like CPU, memory, and I/O as those are more likely to get saturated before the network bandwidth.
- Horizontal scaling is sometimes preferred to vertical scaling. In other words, load could be distributed over a number of machines with modest configuration instead of investing in upgrading the current single machine
- Load balancing can be achieved in a number of ways, the most common of which is DNS load balancing. This refers to adding more than one record of the web machine (typically www) in the DNS zone. When asked for an IP, the DNS serves the recorded IP addresses one after another per request in a round-robin fashion, which is some sort of primitive load balancing (it is used by Google).
- Load balancing can also be achieved using hardware products like F5, and also software products like Linux Virtual Server (free), and Zeus products (commercial)

# The Apache web server

- It serves now (as of Jan. 2016) about 50% of the total websites on the Internet, 10% for Microsoft and 16% for Nginx.
- It can be installed using `yum install httpd` or `apt-get install apache2`
- The configuration file is placed in `/etc/httpd/conf/httpd.conf` on Red Hat machines and in `/etc/apache2.conf` (among other files for specific settings) on Ubuntu and Debian based machines.
- On the file contains three sections:
  - The first section includes global server settings like the port (80 by default)
  - The second section contains configuration options for the default web site. That is, the content the would be served if no virtual hosts are configured. It also has some security settings for directories and files, in addition to the user and group accounts that own the apache daemon.
  - The third section includes configuration for virtual hosts
- Once you make any changes to the configuration file, it's a good idea to run httpd –t to ensure that the file is having a correct syntax before attempting to restart Apache daemon

# Hosting multiple sites on the same server

- Apache webserver allows you to host more than one website on the same machine. This can be accomplished either by adding virtual interfaces or by using name-based virtual hosts.
- A virtual interface is an additional IP configured on the same NIC. It can be done as follows: `ifconfig eth0:0 192.168.0.249 netmask 255.255.255.0 up`
- To make those changes persistent across reboots:
  - On Red Hat: create a file called `/etc/sysconfig/networ-scripts/ifcfg-eth0:0` that would have the same content as the ifcfg-eth0 file except that you omit the MAC address part, and you name the device eth0:0
  - On Ubuntu and Debian: create a similar file in the /etc/network/interfaces with changes similar to he one created in Red Hat.
- In the previous examples we assumed that the network interface on which the alias is created in eth0. You can add as many aliases (eth0:0, eth0:1…etc.) as you want on any configured network interface

# LAB: configuring IP-based virtual hosts

- Target: configure the webserver to serve different websites according to the IP address used in the web request.
- Add a virtual IP to the eth0 interface:
  `ifconfig eth0:0 192.168.0.249 netmask 255.255.255.0 up`
- Create a web directory in /var/www/linuxadmin.dev and add some HTML documents there
- Create an HTML document in the default web directory `/var/www/html`
- In the configuration file `/etc/httpd/conf/httpd.conf` add a virtual host as follows:

```
<VirtualHost 192.168.0.249>
    DocumentRoot /var/www/linuxadmin.dev/
    ServerName www.linuxadmin.dev
    ErrorLog logs/www.linuxadmin.dev-error_log
    CustomLog logs/www.linuxadmin.dev-access_log combined
</VirtualHost>
```

- Restart Apache to effect the changes:
  `apachectl restart`
- Navigate to the IP of the machine http://192.168.0.252 then to the virtual IP http://192.168.0.249 and observe the changes in the displayed content

# LAB: configure name-based virtual hosts

- Target: using only one IP address, an Apache webserver can serve multiple websites depending to the website name (address)
- In /etc/httpd/conf/httpd.conf, uncomment the line
  #NameVirtualHost *:80
  This instructs Apache to start using name-based virtual hosts
- Add the following stanza to define a virtual host to serve documents in /var/www/linuxadmin.dev directory:

```
<VirtualHost *.80>
    DocumentRoot /var/www/linuxadmin.dev/
    ServerName www.linuxadmin.dev
    ErrorLog logs/www.linuxadmin.dev-error_log
    CustomLog logs/www.linuxadmin.dev-access_log combined
</VirtualHost>
```

- Restart the webserver to effect the changes:

```
apachectl restart
```

- In your client (whatever it was Linux, Windows or MAC), add the following line to the appropriate hosts file:
  192.168.0.252 www.linuxadmin.dev
- Navigate to the website once using http://www.linuxadmin.dev and another using http://192.168.0.252 and observe the difference in the served content

# Secure HTTP (https)

- Using SSL (secure sockets layer), communication between the client browser and the webserver is encrypted to prevent any network sniffing or eavesdropping
- This is accomplished using certificate-based authentication: the website presents a certificate the client, this certificate is *signed* by a well known certificate authority (like VeriSign) to guarantee that the website identity is what it claims to be.
- All major certificates have their signatures shipped with browsers. When the client receives the certificate, it compares the signature with the built-in signatures to verify the website identity. Some browsers give the user the option to bypass a failed certificate check and continue navigation.
- SSL used to be a challenge earlier before because – due to US restrictions – had to be no more than 40 bits. However, those restrictions have been lifted now and the keys can be as secure as 128 bits

# LAB: Configuring Apache to use a self-signed certificate

- Ensure that `mod_ssl` is loaded in Apache: `apachectl -M | grep ssl`
  if not you can install it using `yum -y install mod_ssl`
- Ensure that the openssl package is installed `yum -y install openssl`
- Create the directory that is going to host the SSL certificate files:
  `mkdir /etc/httpd/ssl`
- Generate the self-signed certificate:
  `openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/httpd/ssl/cert.key -out /etc/httpd/ssl/cert.crt`
  Go through the steps and supply the appropriate information
- Add the certificate to Apache by editing the line `/etc/httpd/conf.d/ssl.conf` ensuring that the following lines are uncommented and have appropriate information:
  `SSLCertificateFile /etc/httpd/ssl/cert.crt`
  `SSLCertificateKeyFile /etc/httpd/ssl/cert.key`
- Restart Apache to effect the changes `apachectl restart`
- Try to navigate to https://www.linuxadmin.dev you should see a warning that the certificate could not be verified, ignore it and you should be able to see the site content