Other important files include the document root, log file locations, and module locations (enabled in the configuration file).

The default document root is:

- **CentOS:**
  `/var/www/html/`
- **OpenSUSE:**
  `/srv/www/htdocs/`
- **Ubuntu:**
  `/var/www/html/`

The default log file location is:

- **CentOS:**
  `/var/log/httpd/`
- **OpenSUSE:**
  `/var/log/apache2/`
- **Ubuntu:**

The default log file location is:

- **CentOS:**
  `/var/log/httpd/`
- **OpenSUSE:**
  `/var/log/apache2/`
- **Ubuntu:**
  `/var/log/apache2/`

To load a module, use the following syntax:

```
LoadModule alias_module modules/mod_alias.so
```

7.7. Log Configuration

Apache has powerful logging features.

To create custom logs on an **Apache** server, you must first define a custom log format:

```
LogFormat "example-custom-format %h %l %u %t "%r" %>s %b" example-custom-format
```

Then you can create a log file, which uses your custom format:

```
CustomLog "logs/example-custom.log" example-custom-format
```

You can find a reference to all the available tokens at: http://httpd.apache.org/docs/2.2/mod/mod_log_config.html#formats

7.9. IP/Port Virtual Hosts

For multiple web sites using multiple addresses/ports, use **`VirtualHost`** stanzas.

To allow **Apache** to serve different sites on different **IP** addresses or ports, you should do the following:

1. Ensure all of the IP addresses and ports are defined in a **`Listen`** directive.
2. Add a stanza for each virtual host, as in the example below:

```
Listen 192.168.42.11:4374

<VirtualHost 192.168.42.11:4374>

     ServerAdmin webmaster@host1.example.com
       DocumentRoot /www/docs/host1.example.com
       ServerName host1.example.com
       ErrorLog logs/host1.example.com-error_log
       CustomLog logs/host1.example.com-access_log common

</VirtualHost>
```

For multiple web sites using a single address or port, use a **NameVirtualHost** directive or a **_default_** virtual host.

To enable a name-based virtual host for an IP/Port, create a **VirtualHost** stanza and modify the **DocumentRoot** and **ServerName** directives. It is not required to change the logging, however, it is a good idea to have **per-vhost** logs.

```
NameVirtualHost 10.0.0.20:80
Listen 10.0.0.20:80
<VirtualHost 10.0.0.20:80>
     DocumentRoot /www/docs/host1.example.com
     ServerName host1.example.com
     ErrorLog logs/host1.example.com-error_log
     CustomLog logs/host1.example.com-access_log common
</VirtualHost>
<VirtualHost 10.0.0.20:80>
     DocumentRoot /www/docs/secondhost.example.com
     ServerName secondhost.example.com
</VirtualHost>
```

Host names which are not defined in a **VirtualHost** stanza which match the **IP** address and port will be served by the

7.11. Name Vhosts and SSL

Name-based virtual hosts have some **SSL** limitations. Due to the way **SSL** works, the server has no way of knowing which host name is being sent by the client before the **SSL** session is started. The server cannot send the proper certificate back to the client without the proper host name. This has been worked around by using **Server Name Indication** (http://en.wikipedia.org/wiki/Server_Name_Indication). Support for **SNI** is still somewhat uncertain. **SNI** support exists on the following browsers:

- **IE 7** or higher (except on **Windows XP**).
- **Mozilla Firefox 2.0** or higher.
- **Opera 8.0** or higher.
- **Google Chrome.**
- **Safari 2.1** or higher.
- **Konqueror/KDE 4.7** or higher.

Apache was originally based on the **NSCA HTTPd** server:

- It is the most popular **HTTP** server today.
- It is extensible, using modules.

As you can see in Figure 7.1, the **NetCraft January 2016** web server survey shows that **Apache** runs on nearly 40% of all active web servers. You can learn more about the survey on the **Internet**: http://news.netcraft.com/archives/2016/01/26/january-2016-web-server-survey.html

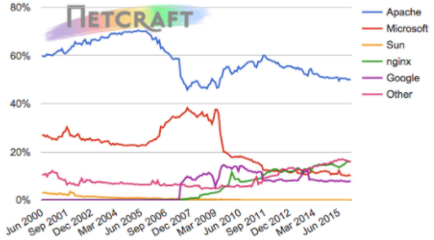

Fig. 7.1: Web Server Developers: Market Share of Active Sites (Retrieved from netcraft.com)

7.4.a. Main Configuration File

To access the main configuration file for **Apache**, which has a simple SGML-like format, use the following:

- **On CentOS:**
  `/etc/httpd/conf/httpd.conf`
- **On OpenSUSE:**
  `/etc/apache2/httpd.conf`
- **On Ubuntu**:
  `/etc/apache2/apache2.conf`
  which includes
  `/etc/apache2/mods-enabled/`
  `/etc/apache2/conf-enabled/`
  `/etc/apache2/sites-enabled/`

7.4.b. Main Configuration File (Cont'd)

Some of the more important configuration options are:

- The address and port to serve traffic from:

```
Listen 12.34.56.78:80
```

- Number of processes to start, maximum number of threads or forks (depending on the **MPM** module used):

```
MPM Config
<IfModule prefork.c>
StartServers 8
MinSpareServers 5
MaxSpareServers 20
ServerLimit 256
MaxClients 256
MaxRequestsPerChild 4000
</IfModule>
```

- The location on the filesystem where the served documents reside:

```
DocumentRoot "/var/www/html"
```

7.5. Include Configuration File

To allow for modification and flexibility in the **apache** configuration file, you can include other files and directories. This allows you to avoid one large configuration file and is useful for servers with multiple sites. Many distributions use this feature to enable or disable web server configurations by installing or removing packages.

The **OpenSUSE** distribution also allows for easy creation of additional **include** files and directories. To learn more, check out the `/etc/sysconfig/apache2` file.

Some of the default **include** directories are:

- **CentOS**:

  `/etc/httpd/conf.d/*.conf`

- **OpenSUSE**:

  `/etc/apache2/conf.d/`

- **Ubuntu**:

  `/etc/apache2/conf-enabled/`

  `/etc/apache2/sites-enabled/`

7.12. Access Control and .htaccess

Password-protected directories are protected via **SSL**. Per-User Access Control is activated using **AllowOverride** (**`.htaccess`** file).

Creating a password-protected directory is done in a stanza like this:

```
<Location /secure/>
AuthType Basic
AuthName "Restricted Files"
AuthUserFile secure.passwords
Require valid-user
</Location>
```

Create the user password file with the following command:

```
# htpasswd -c /etc/httpd/secure.passwords bob
```

- The **`-c`** switch is only needed the first time, to create the password database.

Limiting a certain directory to specific **HTTP** methods is done with the following stanza:

```
<Directory /var/www/html/get-only/>
```

Limiting a certain directory to specific **HTTP** methods is done with the following stanza:

```
<Directory /var/www/html/get-only/>
     <LimitExcept GET>
       Require valid-user
     </LimitExcept>
</Directory>
```

7.13. File Permissions

In addition to the access controls provided by the **Apache** configuration file, the filesystem permissions must allow access. This access should be for the user the **Apache httpd** daemon is running under (**`apache`** on most systems).

7.14. SELinux

A properly configured **SELinux** system can increase the security of an **Apache w**eb server. There are some settings you may need to change:

- To allow **Apache** to make outbound network connections (not directly related to serving a request), modify the **`httpd_can_network_connect`** boolean.
- The default **Document Root** and **CGI** root have the proper **SELinux** context. If you serve files from outside those locations, you need to use the **`chcon`** command.
- To enable the **`userdir`** module, modify the **`httpd_read_user_content`** boolean.
- To serve files from an **NFS** filesystem, modify the **`httpd_use_nfs`** boolean.

If your scripting languages supports it, a "taint" mode is a good idea. You should also sanitize inputs. For example, using a **CGI** script like the following is not a good idea:

```
$ echo -e "Content-type: text/plain "

$ echo -e "File is $1 "
$ cat "$1"
```

The above script would cat any file passed to the **URI** (i.e. `/cgi-bin/bad.sh?/etc/passwd`). **SQL** queries are another point of concern with sanitizing input. Many database libraries have methods for turning control characters into sanitized input. However, using stored or prepared statements with bound parameters is more secure.

When sanitizing inputs, do not blindly trust data from browsers. You should also use abstraction with database queries.

While **HTTP** is a clear text protocol, **SSL** is a port-based **vhost**. Types of **SSL** certificates include:

- Self-signed - for hobby or testing use.
- Certificate Authority Signed (**C**ertificate **S**igning **R**equest or **CSR**).

To generate a private key, use the following command:

```
$ openssl genrsa -aes128 2048>somefile.key
```

To generate a **CSR**, use the following command:

```
$ openssl req -new -key server.key -out server.csr
```

To generate a self-signed certificate, you should use the following command:

```
$ openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

It is vitally important to properly protect the private key. Keeping the key encrypted is a good idea, but it makes it difficult to restart the server. There are some ways around this, but most people just remove the encryption:

```
$ openssl rsa -in server.key -out server.key.unlocked
```

7.17. Install SSL Keys

To install your keys, place them in the proper place on your system, or change the configuration file (usually `ssl.config`):

- **On CentOS:**

  `/etc/pki/tls/certs/localhost.crt:` `Signed Certificate`

  `/etc/pki/tls/private/localhost.key`: `Private Key`

- **On OpenSUSE:**

  `/etc/apache2/ssl.crt/server.crt`: `Signed Certificate`

  `/etc/apache2/ssl.key/server.key`: `Private Key`

- **On Ubuntu:**

  `/etc/ssl/certs/ssl-cert-snakeoil.pem`: `Signed Certificate`

  `/etc/ssl/private/ssl-cert-snakeoil.key`: `Private Key`