Web Servers Cheatsheet

A quick reference guide to web servers, covering core concepts, popular servers like Apache and Nginx, and essential configurations.

Web Server Fundamentals

Core Concepts

Web Server: Software that responds to client requests over HTTP.

HTTP (Hypertext Transfer Protocol): The foundation of data communication on the web.

Client-Server Model: A client (e.g., a web browser) sends requests to a server, which processes them and returns a response.

Static Content: Web content that is pre-built and served as-is (e.g., HTML, CSS, JavaScript files, images).

Dynamic Content: Web content generated on the server-side, often using scripting languages (e.g., PHP, Python, Node.js).

Application Server: A server that hosts web applications and provides services for them to run.

Reverse Proxy: A server that sits in front of one or more web servers, handling client requests and forwarding them to the appropriate server.

Common Web Servers

Apache HTTP Server	A widely used, open-source web server known for its flexibility and module support.
Nginx	A high-performance web server and reverse proxy server, often used for its speed and efficiency.
Microsoft IIS (Internet Information Services)	A web server developed by Microsoft for use with Windows Server.
Lighttpd	Another open-source web server designed for speed-critical environments.

Key Features

Virtual Hosts	Hosting multiple websites on a single server.
Load Balancing	Distributing network traffic across multiple servers to improve performance and reliability.
SSL/TLS Encryption	Securing web traffic with encryption to protect sensitive data.
Caching	Storing frequently accessed content to reduce server load and improve response times.

Apache Configuration

Configuration Files

`httpd.conf` or `apache2.conf`: The main configuration file.

`Virtual Host` files: Configuration files for individual websites, often located in `/etc/apache2/sites-available/`.

Use `apachectl` or `systemctl` to manage Apache.

Example:
`sudo apachectl restart` or `sudo systemctl restart apache2`

Common Directives

`DocumentRoot`	Specifies the directory from which Apache serves files for a website. **Example:** `DocumentRoot /var/www/html`
`ServerName`	Specifies the domain name or IP address of the server. **Example:** `ServerName example.com`

`<Directory>`	Defines access control and other settings for a specific directory.
	
	Example:
	`<Directory /var/www/html>`
	`  Require all granted`
	`</Directory>`

`ErrorLog` and `CustomLog`	Specify the location of error and access log files.
	
	Example:
	`ErrorLog /var/log/apache2/error.log`
	`CustomLog /var/log/apache2/access.log combined`

`LoadModule`	Enables specific Apache modules. **Example:** `LoadModule rewrite_module modules/mod_rewrite.so`

Virtual Hosts

A virtual host configuration allows you to run multiple websites on a single Apache server.

Example Virtual Host Configuration:

```
<VirtualHost *:80>
  ServerName example.com
  DocumentRoot /var/www/example.com
  <Directory /var/www/example.com>
    Require all granted
  </Directory>
  ErrorLog /var/log/apache2/example.com-error.log
  CustomLog /var/log/apache2/example.com-access.log combined
</VirtualHost>
```

Enable a virtual host using `a2ensite` and disable using `a2dissite`.

Example:
`sudo a2ensite example.com`
`sudo systemctl restart apache2`

Nginx Configuration

Configuration Files

`nginx.conf`: The main Nginx configuration file, usually located in `/etc/nginx/`.

`sites-available/`: Directory for virtual host configuration files.
`sites-enabled/`: Directory for symlinks to enabled virtual host configuration files.

Use `nginx` or `systemctl` to manage Nginx.

Example:
`sudo nginx -t` (test configuration)
`sudo systemctl restart nginx`

Common Directives

`server` block	Defines a virtual server (similar to Apache's VirtualHost). **Example:** `server { listen 80; server_name example.com; root /var/www/example.com; index index.html index.htm; }`
`listen`	Specifies the port on which the server listens for connections. **Example:** `listen 80;`
`server_name`	Specifies the domain name or IP address of the server. **Example:** `server_name example.com;`
`root`	Specifies the directory from which Nginx serves files for a website. **Example:** `root /var/www/example.com;`
`location`	Defines how Nginx handles requests for specific URIs. **Example:** `location / { try_files $uri $uri/ =404; }`

Reverse Proxy Example

Nginx can be used as a reverse proxy to forward requests to backend servers.

Example Configuration:

```
server {
  listen 80;
  server_name example.com;

  location / {
    proxy_pass http://backend_server;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP
$remote_addr;
  }
}
```

Security Best Practices

General Security Measures

- Keep your web server software up to date with the latest security patches.
- Use a firewall to restrict access to your server.
- Disable unnecessary modules or features.
- Regularly audit your server configuration for security vulnerabilities.

SSL/TLS Configuration

Obtain an SSL/TLS Certificate	From a trusted Certificate Authority (CA) like Let's Encrypt, or purchase a certificate.
Configure SSL/TLS	Enable HTTPS by configuring your web server to use the SSL/TLS certificate.
Use Strong Cipher Suites	Configure your web server to use strong and secure cipher suites.
Redirect HTTP to HTTPS	Automatically redirect all HTTP traffic to HTTPS to ensure secure communication.

Access Control

Limit Directory Access	Restrict access to sensitive directories by configuring appropriate permissions.
Implement Authentication	Require users to authenticate before accessing certain areas of your website.
Use a Web Application Firewall (WAF)	A WAF can help protect your website from common web attacks like SQL injection and cross-site scripting (XSS).
Regularly Monitor Logs	Monitor your web server logs for suspicious activity.