Apache Webserver For Beginners

In This Video:

- Installing Apache (httpd)
- Custom Domain
- HTTPS/SSL
- Multiple Websites
- Reverse Proxy
- Load Balancing

In This Video:

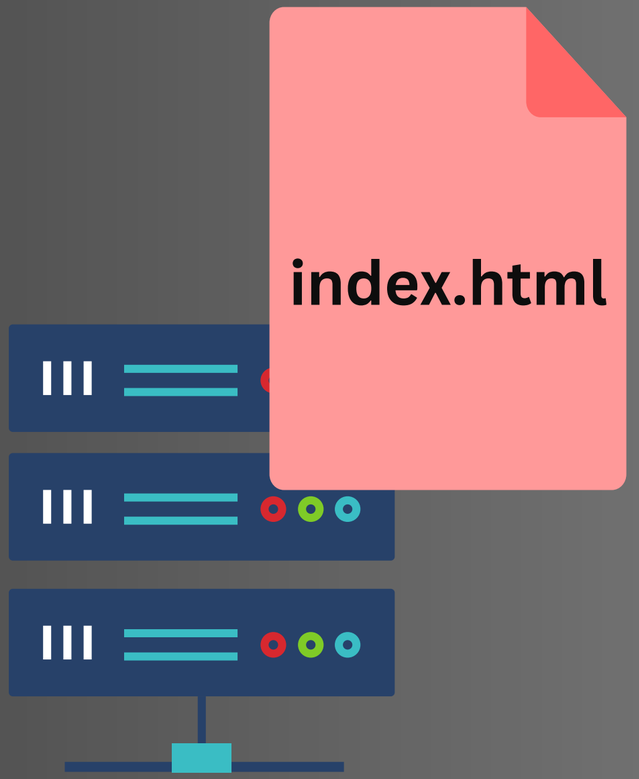
- 🔧 Installing Apache (`httpd`)
- 🌐 Setting up a **Custom Domain**
- 🔒 Enabling **HTTPS/SSL with Let's Encrypt**
- 🏠 Hosting **Multiple Websites (Virtual Hosts)**
- 🔄 Setting up a **Reverse Proxy**
- ⚖️ Configuring **Load Balancing**
- 🛠️ Real-world Linux (EC2) Setup

Introduction

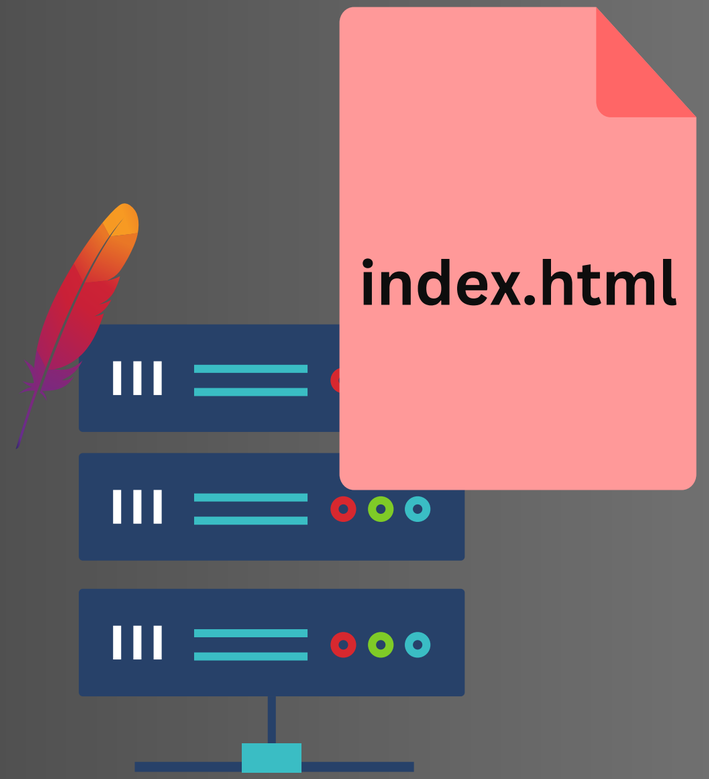
- What is Apache HTTP Server?
- Use cases and popularity in web hosting
- Apache vs NGINX – quick overview

Apache Web Server (Apache HTTPD) is an open-source web server that helps deliver web pages to users over the internet.

It processes requests from browsers and serves websites using the HTTP protocol.

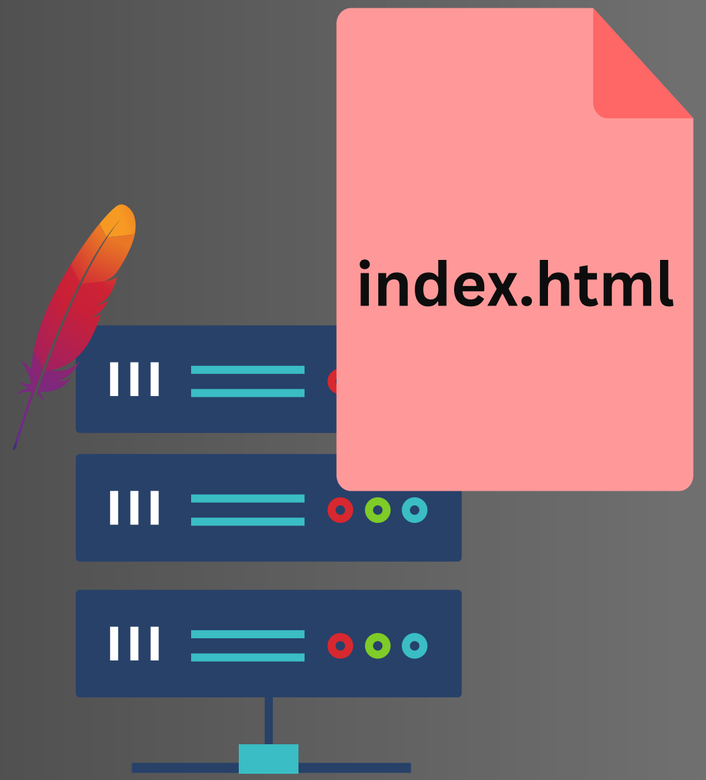


server



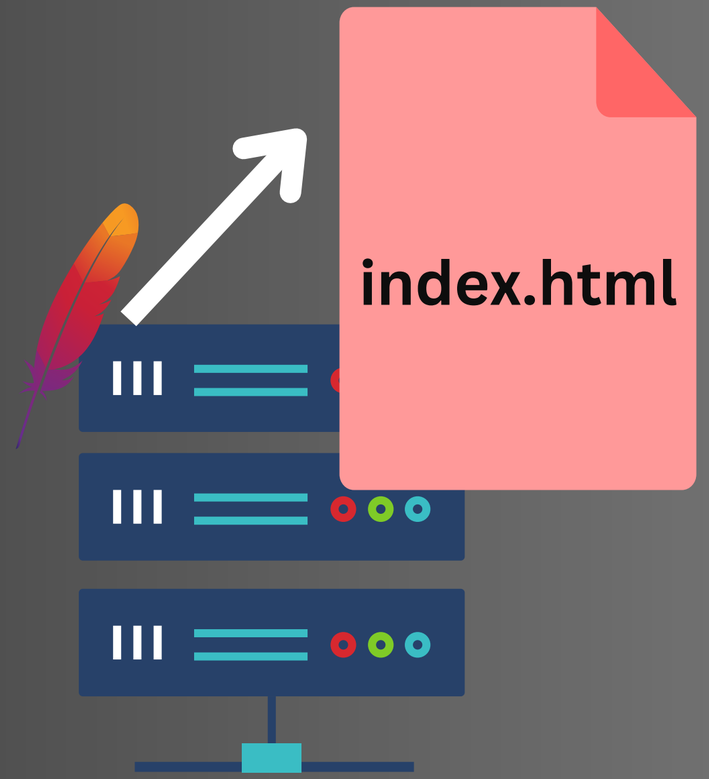
server

www.example.com

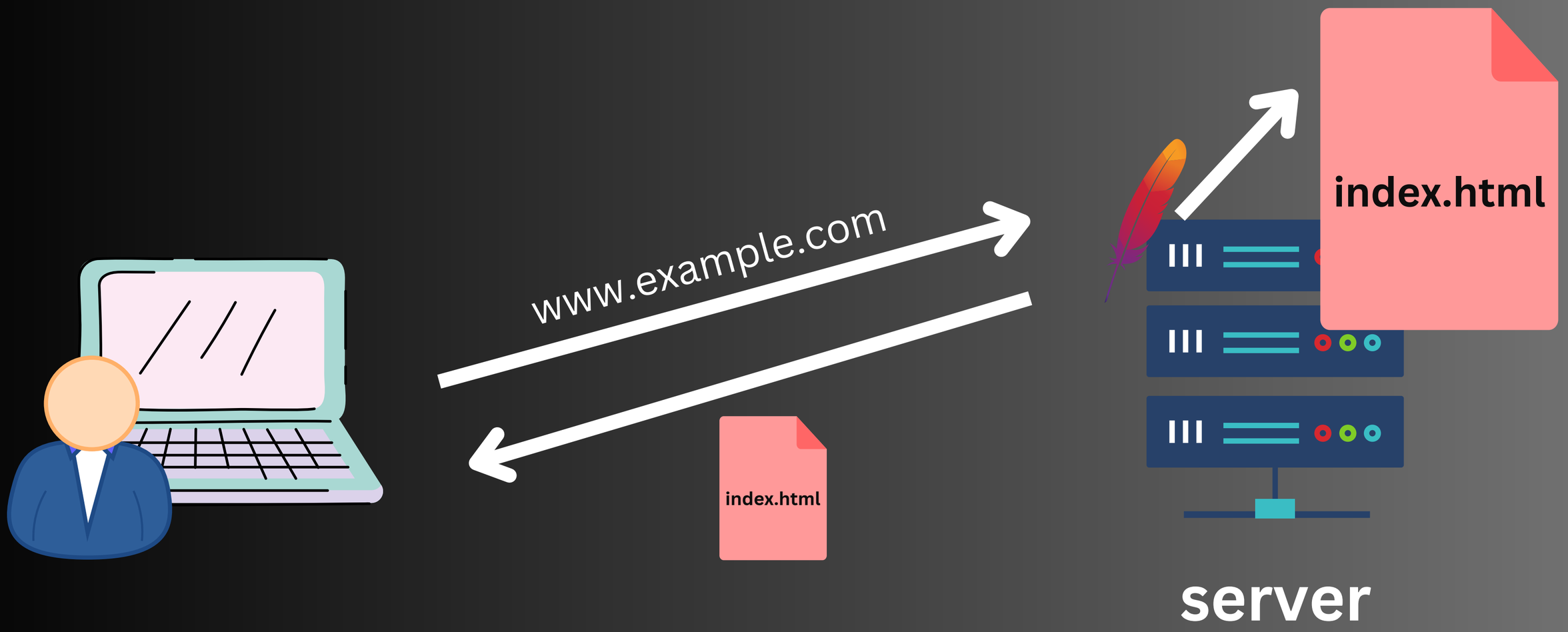


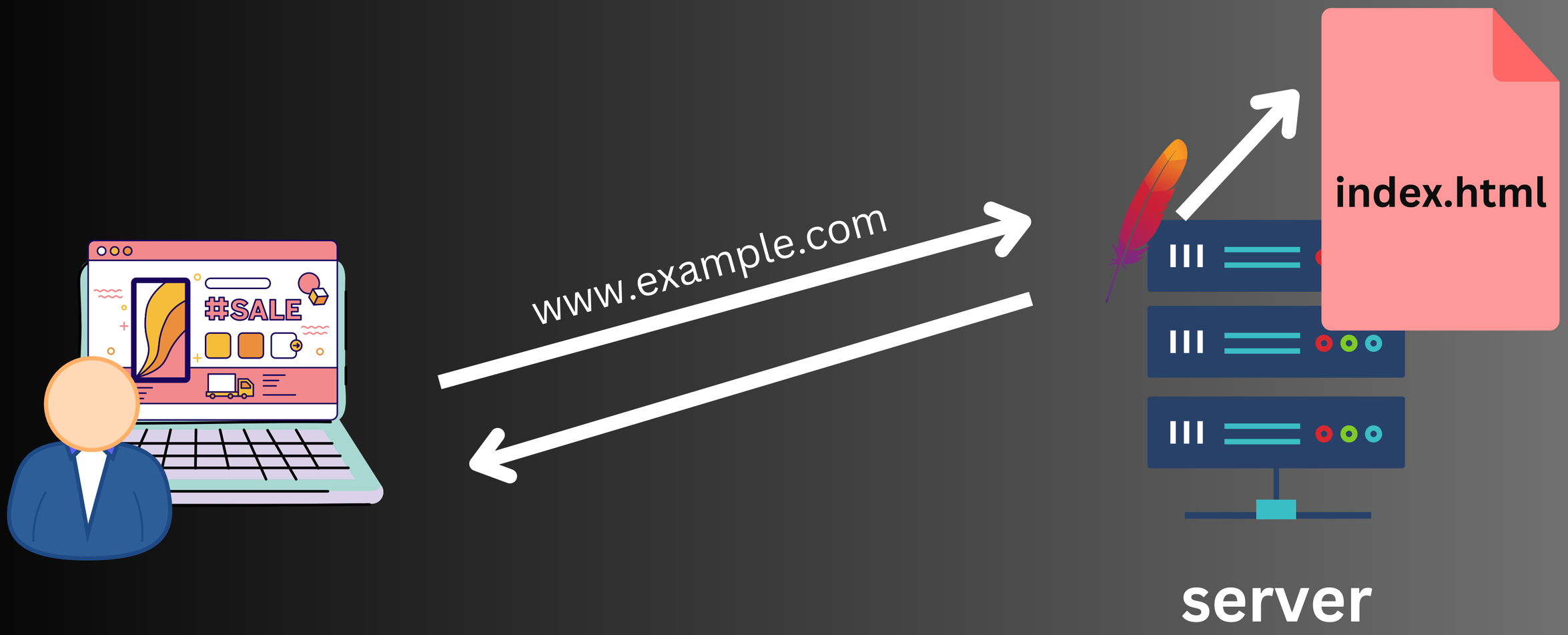
server

www.example.com

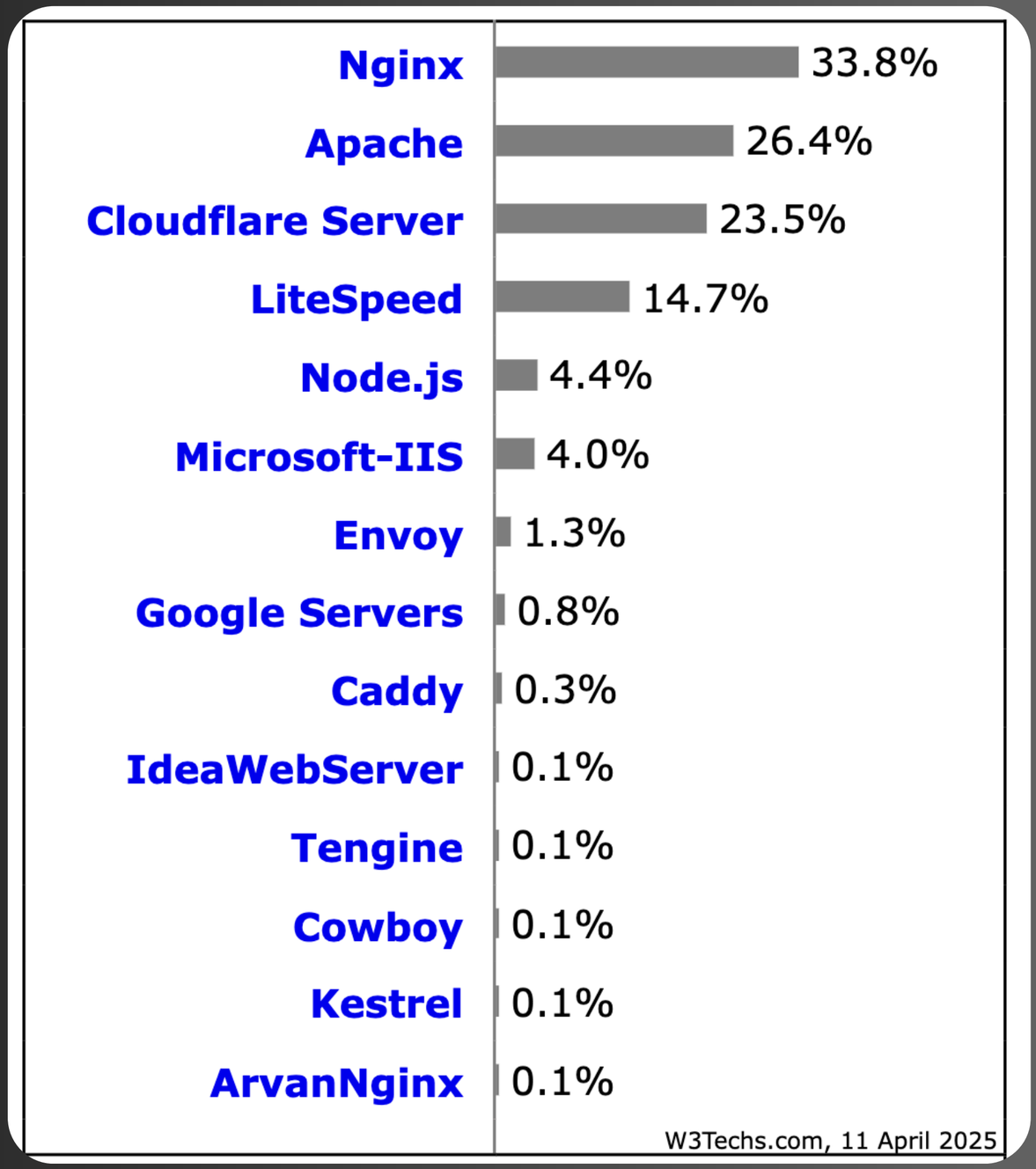


server


www.example.com
index.html
index.html
server


#SALE
www.example.com
index.html
server

Popularity


Nginx 33.8%
Apache 26.4%
Cloudflare Server 23.5%
LiteSpeed 14.7%
Node.js 4.4%
Microsoft-IIS 4.0%
Envoy 1.3%
Google Servers 0.8%
Caddy 0.3%
IdeaWebServer 0.1%
Tengine 0.1%
Cowboy 0.1%
Kestrel 0.1%
ArvanNginx 0.1%
W3Techs.com, 11 April 2025

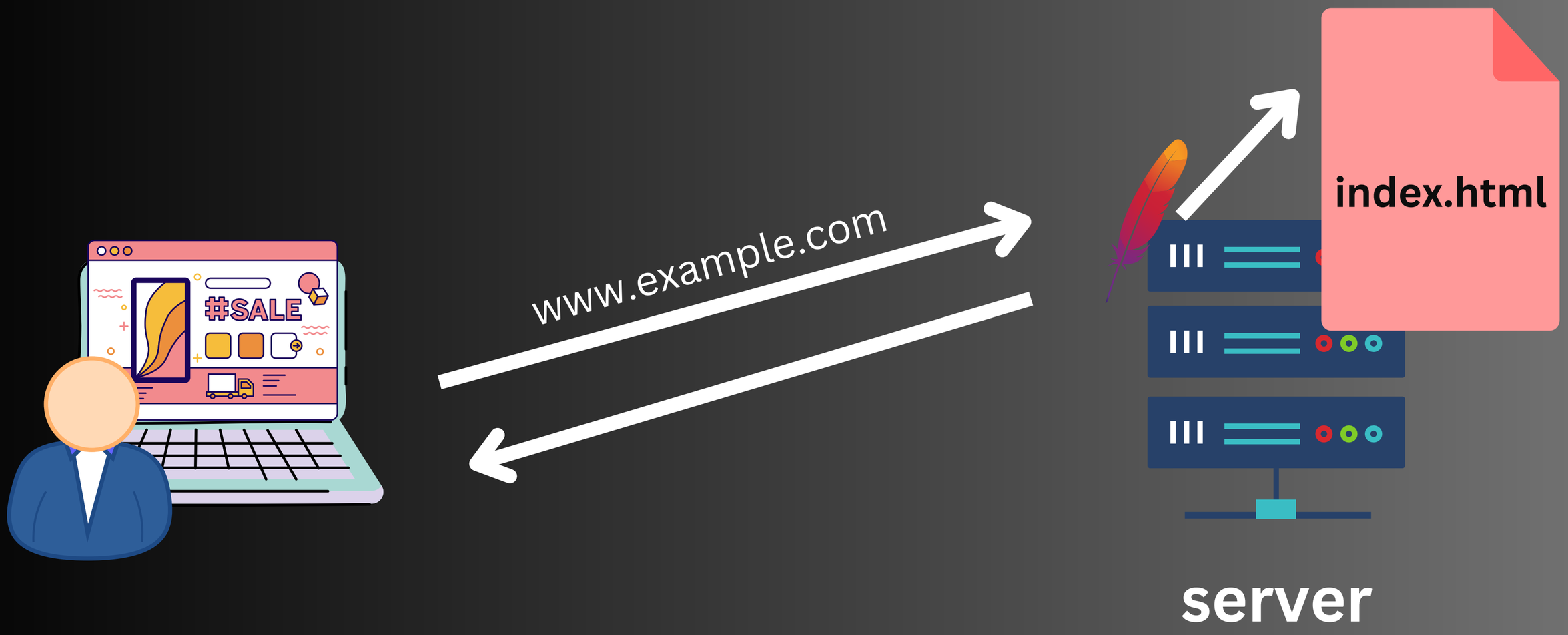
✅ Common Use Cases in Web Hosting

Use Case	How Apache Helps
🌐 **Hosting static websites**	Serves HTML, CSS, JS files directly — simple and fast
🦫 **Dynamic content with PHP**	Integrates with PHP via `mod_php` or `php-fpm`
🔁 **Reverse proxy & Load balancing**	Distributes traffic to backend apps via `mod_proxy`
🧰 **Multi-site hosting (shared hosting)**	Uses virtual hosts to serve multiple domains from one server
🔒 **Secure web hosting (HTTPS)**	Full support for TLS/SSL certificates with Let's Encrypt
📁 **Serving apps behind proxies**	Works with backend app servers (Node.js, Django, etc.)
🔐 **Access control/auth**	Built-in basic & digest auth, IP whitelisting, etc.

🔄 Apache vs Others (Nginx, LiteSpeed, etc.)

Feature	Apache	Nginx
Config flexibility	✅ Very high	🔸 Medium
Dynamic content	✅ Native support	❌ Needs FastCGI proxy
Static file speed	🔸 Good	✅ Faster
Modules	✅ Plug-in model	🔸 Limited/custom build
Popular in shared hosting	✅ Most popular	🔸 Growing

Practical


#SALE
www.example.com
index.html
server

Prerequisites

- AWS Account
- We will test this on EC2 instance

- Installing Apache Webserver
  - yum install httpd
  - apt install apache2
- Start HTTPD service
  - systemctl start/stop/reload httpd
- Enable HTTP service in firewalld
  - sudo firewall-cmd --permanent --add-service=http

- Access Default website from browser
- Config
  - /etc/httpd/conf/httpd.conf (centos)
  - /etc/apache2/apache2.conf (ubuntu)

- Create our own static website
- Configure httpd config

HTTPD Config Min Working

```
ServerRoot "/etc/httpd"
Listen 80

Include conf.modules.d/*.conf
DocumentRoot "/var/www/html"

DirectoryIndex index.html
ErrorLog "logs/error_log"

<IfModule mime_module>
    TypesConfig /etc/mime.types
</IfModule>
```

To check the syntax in config file

httpd -t

HTTP Status

HTTP Status	Meaning	Why Cover It?
404	Not Found	Prevents generic Nginx error messages
403	Forbidden	Blocks unauthorized access with a friendly message
500	Internal Server Error	Hides technical errors from users
502	Bad Gateway	Helps when backend crashes or restarts
503	Service Unavailable	Useful for maintenance pages
504	Gateway Timeout	Indicates slow or unresponsive backend

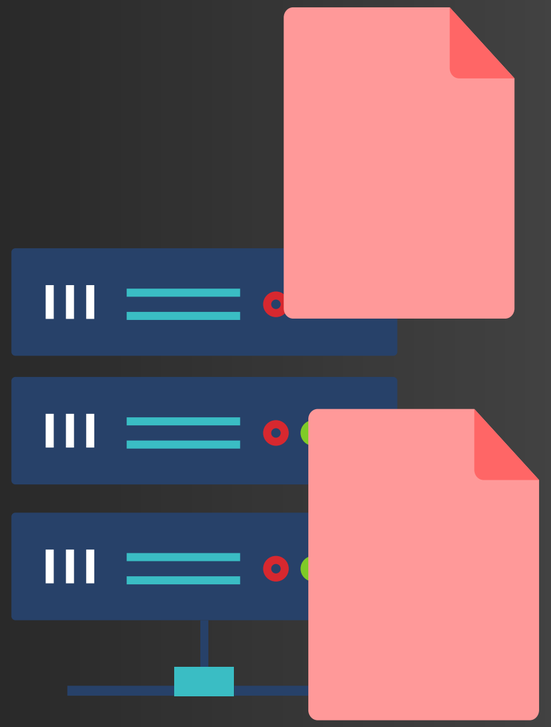
HTTPD Logging

/var/log/httpd/

🔍 LogFormat Breakdown:

Code	Meaning
`%h`	Remote **IP address** (we covered this earlier)
`%l`	Remote **logname** (from identd) – usually `-` because it's rarely used
`%u`	Remote **user** (if HTTP auth is used), else `-`
`%t`	**Time** the request was received
`\"%r\"`	The full **HTTP request line** (e.g., `"GET /index.html HTTP/1.1"`)
`%>s`	Final **HTTP status code** sent to the client (e.g., `200`, `404`)
`%b`	Size of the response in **bytes** (excluding headers), `-` if zero

Hosting Multiple Websites



website1

website2

server

- Changes in httpd.conf
  - Add: IncludeOptional conf.d/*.conf

- For different websites maintain different directories
  - /var/www/site1/public_html
  - /var/www/site2/public_html

- In /etc/httpd/conf.d/ create conf for each websites
  - site1.conf
  - site2.conf
  - Add VirtualHost block

```
<VirtualHost *:80>
    ServerName site1.com
    ServerAlias www.site1.com
    DocumentRoot /var/www/site1/public_html

    ErrorLog /var/log/httpd/site1_error.log
    CustomLog /var/log/httpd/site1_access.log combined
</VirtualHost>
```

- If we want to host multiple websites on different ports then in httpd.conf just use multiple Listen
  - Listen 80
  - Listen 8080

- If we are using domain name then make changes in /etc/hosts

Custom Domain

- Register a domain name.
- In DNS settings, create a A record and point to public IP of our server (EC2)

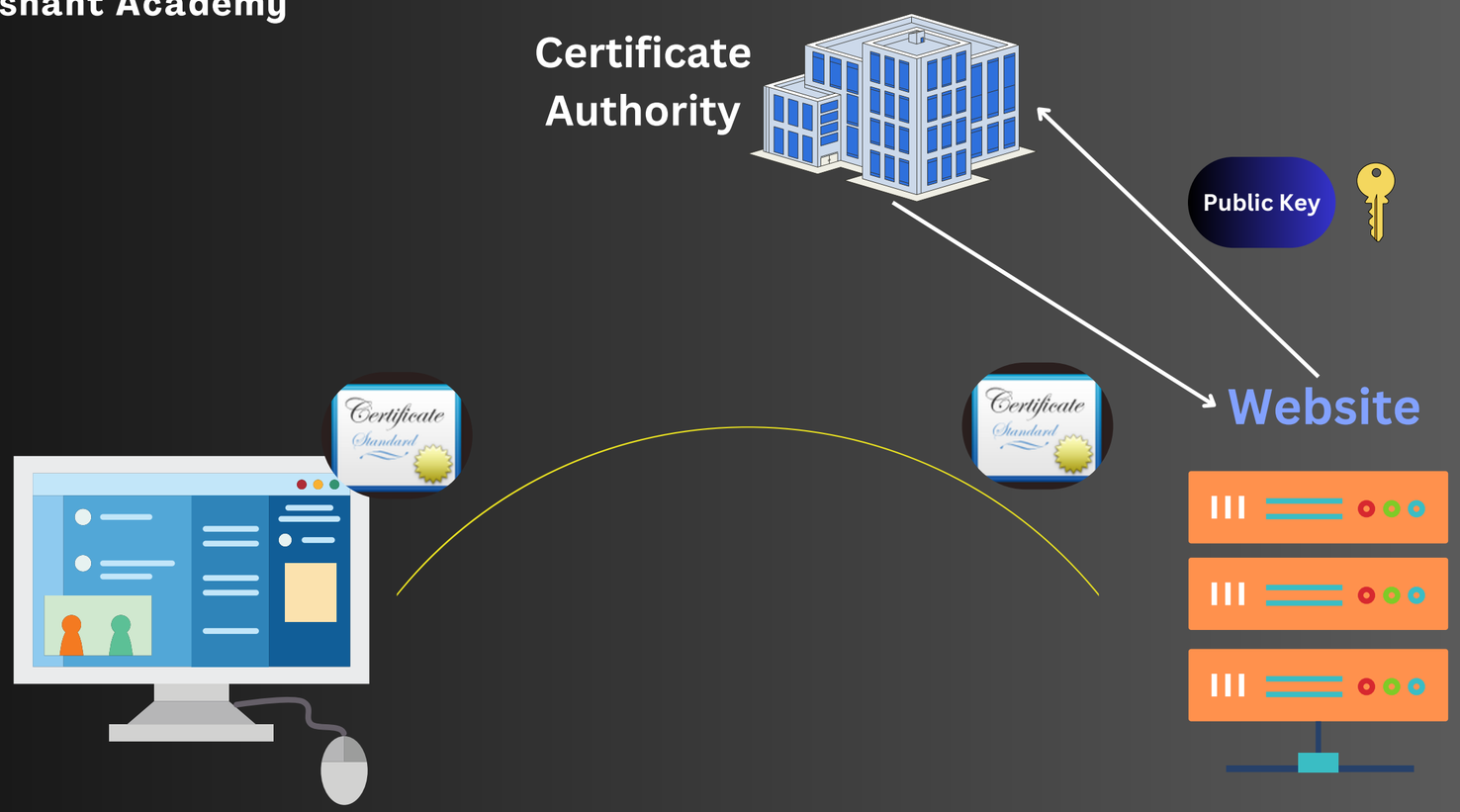
- In conf, use
    - ServerName www.example.com;

HTTPS Setup

HTTPS is a secure version of HTTP that encrypts data between your browser and a website, making it safe from hackers.

It uses SSL/TLS to protect sensitive information like passwords and credit card details.

MPrashant Academy



- For Mac
  - brew install certbot
- For Windows
  - choco install certbot -y
- For Linux
  - Centos
    - sudo yum install epel-release -y
    - sudo yum install certbot python3-certbot-apache -y
  - Ubuntu
    - sudo apt install certbot python3-certbot-apache -y

- First stop the webserver (if listening on port 80) as below command will use local port 80.

- To generate certificates
  - sudo certbot certonly --standalone -d yourdomain.com

- Files will be generated in
  - /etc/letsencrypt/live/yourdomain.com/

Changes in Config

SSLEngine on
SSLCertificateFile /etc/pki/tls/certs/apache-selfsigned.crt
SSLCertificateKeyFile /etc/pki/tls/private/apache-selfsigned.key

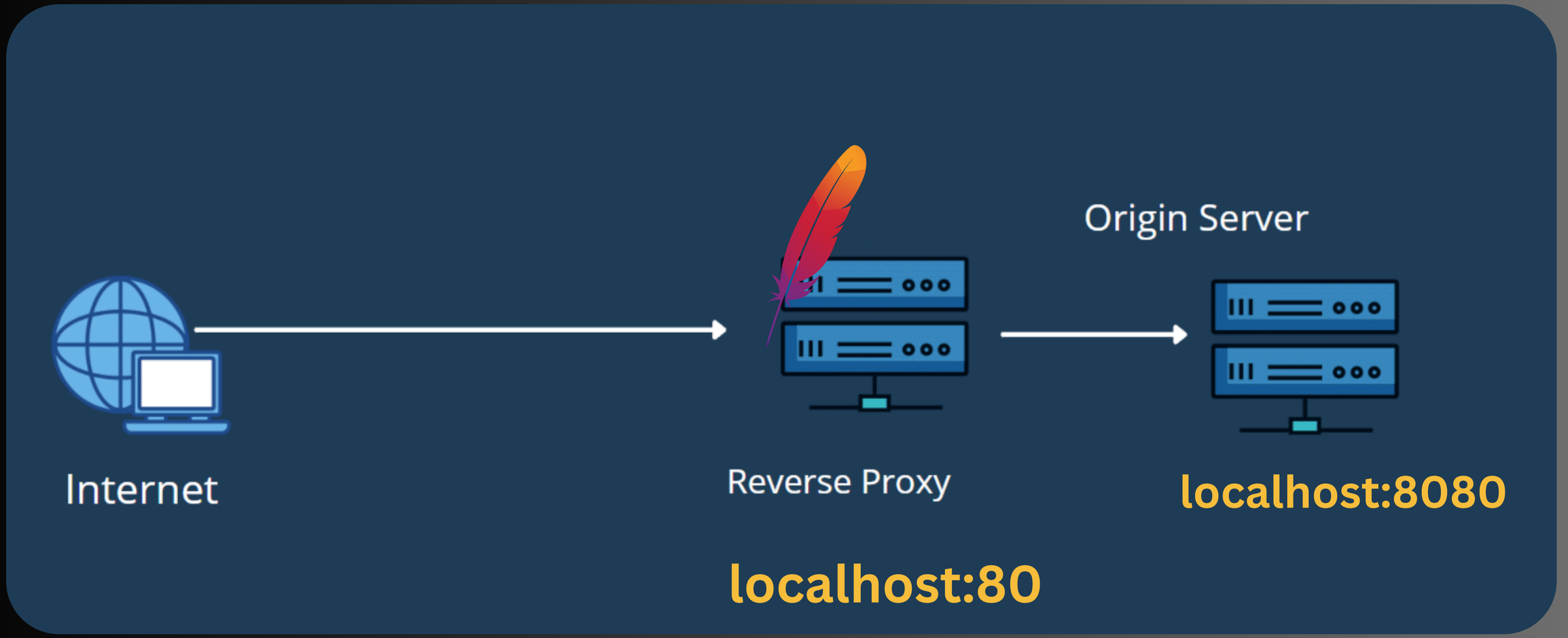
- `SSLEngine on` → turn on SSL support
- `SSLCertificateFile` → path to the SSL certificate (public)
- `SSLCertificateKeyFile` → path to the private key

Reverse Proxy

Reverse proxy acts as an intermediary between clients and backend servers.

It forwards client requests to the appropriate server, handles responses, and provides benefits like load balancing, caching, and security.


Origin Server
Internet
localhost:8080


Origin Server
Internet
Reverse Proxy
localhost:8080
localhost:80

Setup

- Install/start React App
- Make changes in httpd.conf file to pass the request to & access React App.

Steps to setup and run react app

- sudo yum install -y nodejs
- npx create-react-app my-react-app
- cd my-react-app
- npm start

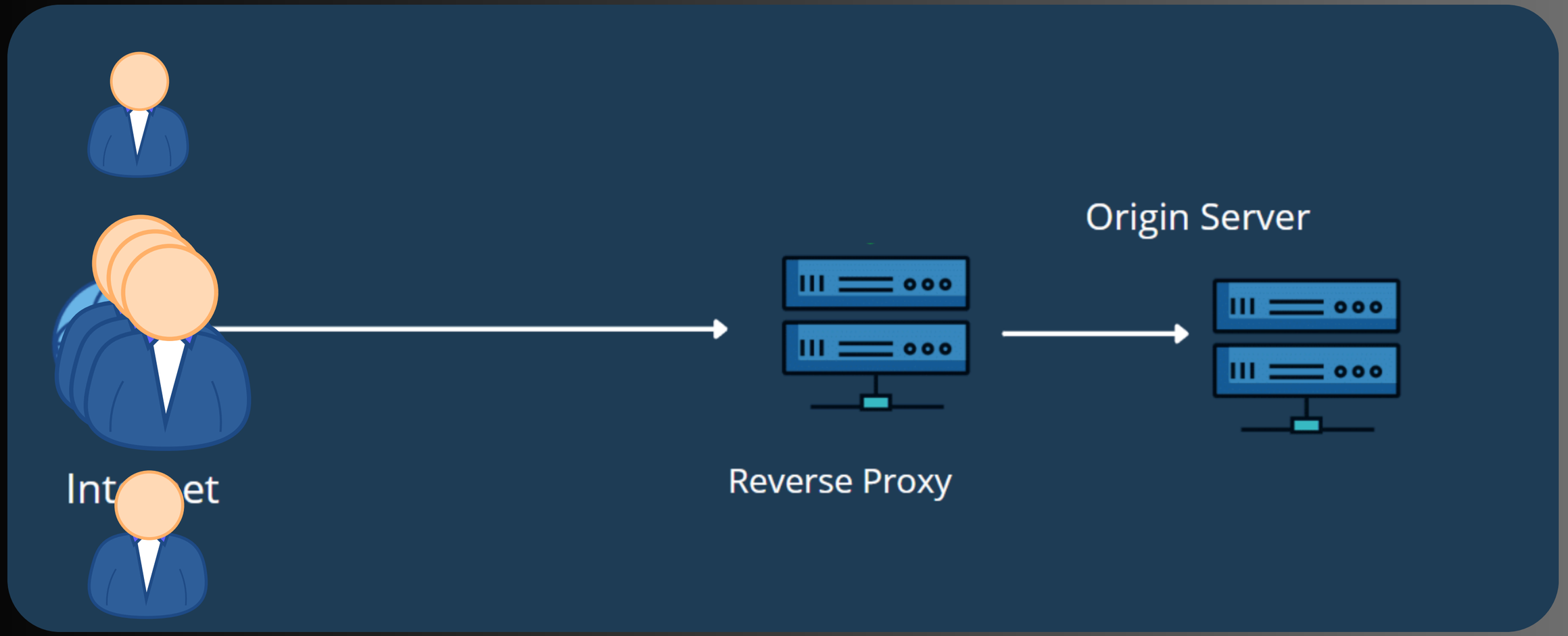
```
<VirtualHost *:80>
    ServerName mysite.local

    ProxyPreserveHost On
    ProxyPass / http://127.0.0.1:8080/
    ProxyPassReverse / http://127.0.0.1:8080/

</VirtualHost>
```

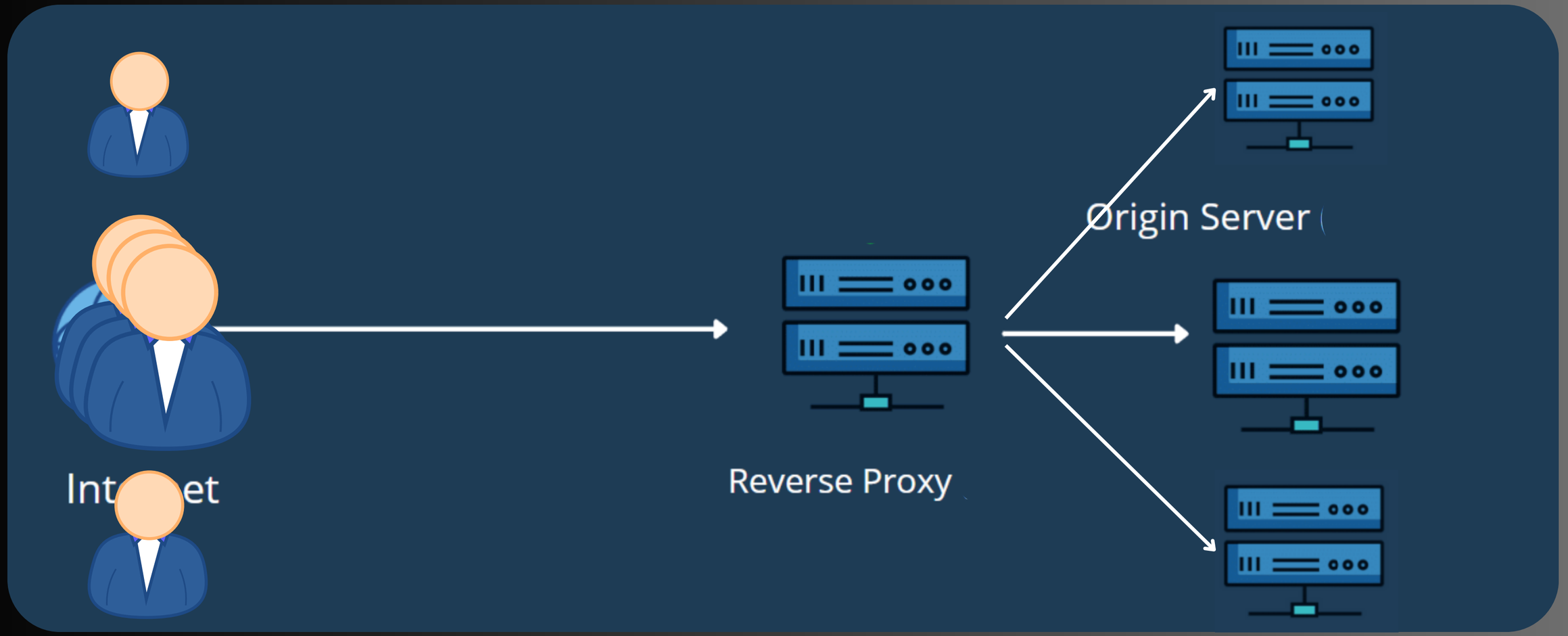
To use custom port for HTTP, you may need to enable it in SeLinux, use below command

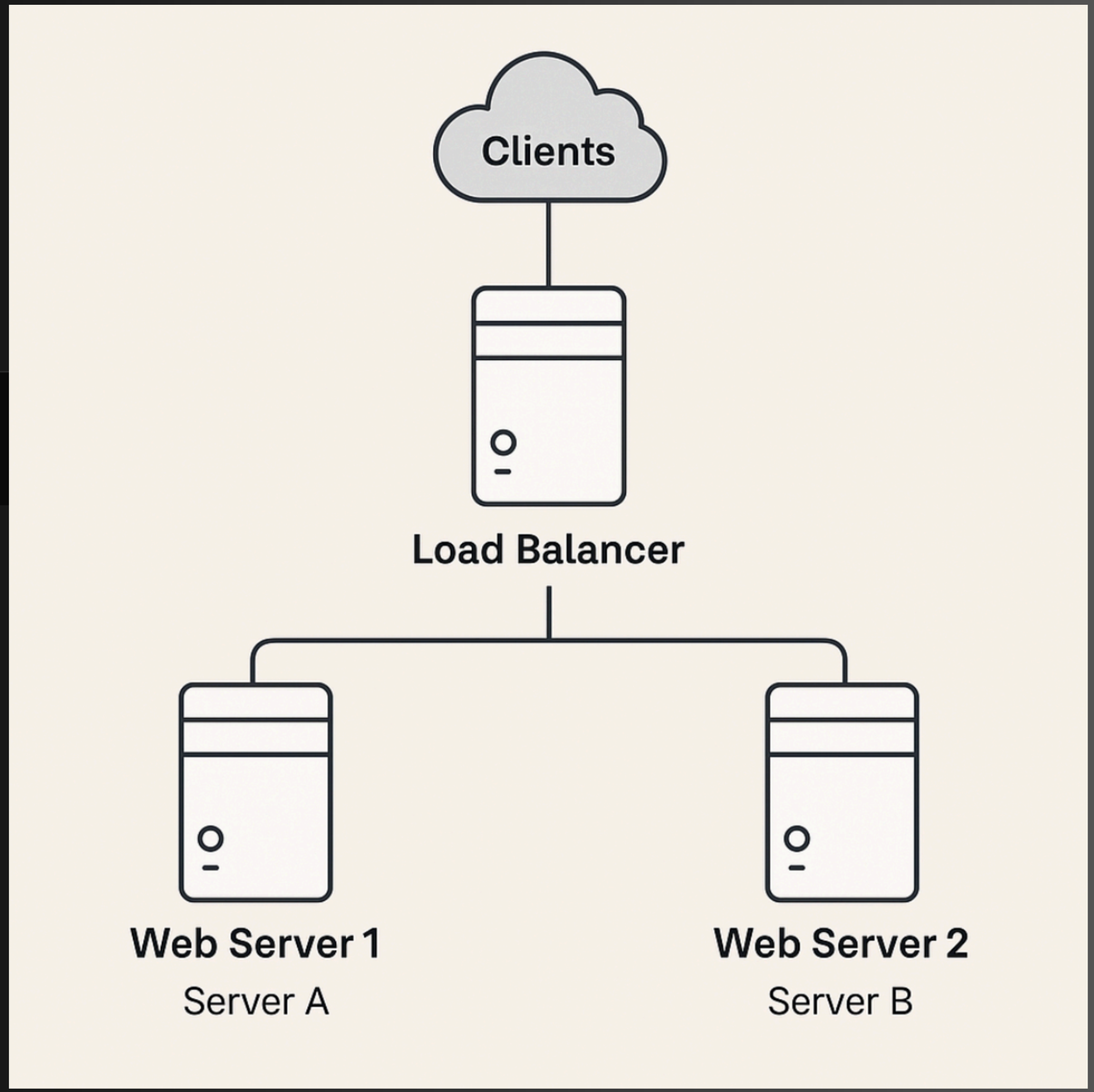
- sudo semanage port -a -t http_port_t -p tcp 8081
- sudo setsebool -P httpd_can_network_connect 1

Load Balancing

Apache web servers, load balancing refers to distributing incoming network traffic across multiple servers to prevent overload, improve performance, and ensure high availability by using a load balancer that acts as a proxy.


Origin Server
Reverse Proxy


Origin Server
Reverse Proxy


Clients
Load Balancer
Web Server 1
Server A
Web Server 2
Server B

```
<VirtualHost *:80>
    ServerName loadbalancer.local

    <Proxy "balancer://mycluster">
        BalancerMember http://192.168.122.101:80
        BalancerMember http://192.168.122.102:80
        ProxySet lbmethod=byrequests
    </Proxy>

    ProxyPass "/" "balancer://mycluster/"
    ProxyPassReverse "/" "balancer://mycluster/"
</VirtualHost>
```

ProxySet lbmethod=byrequests

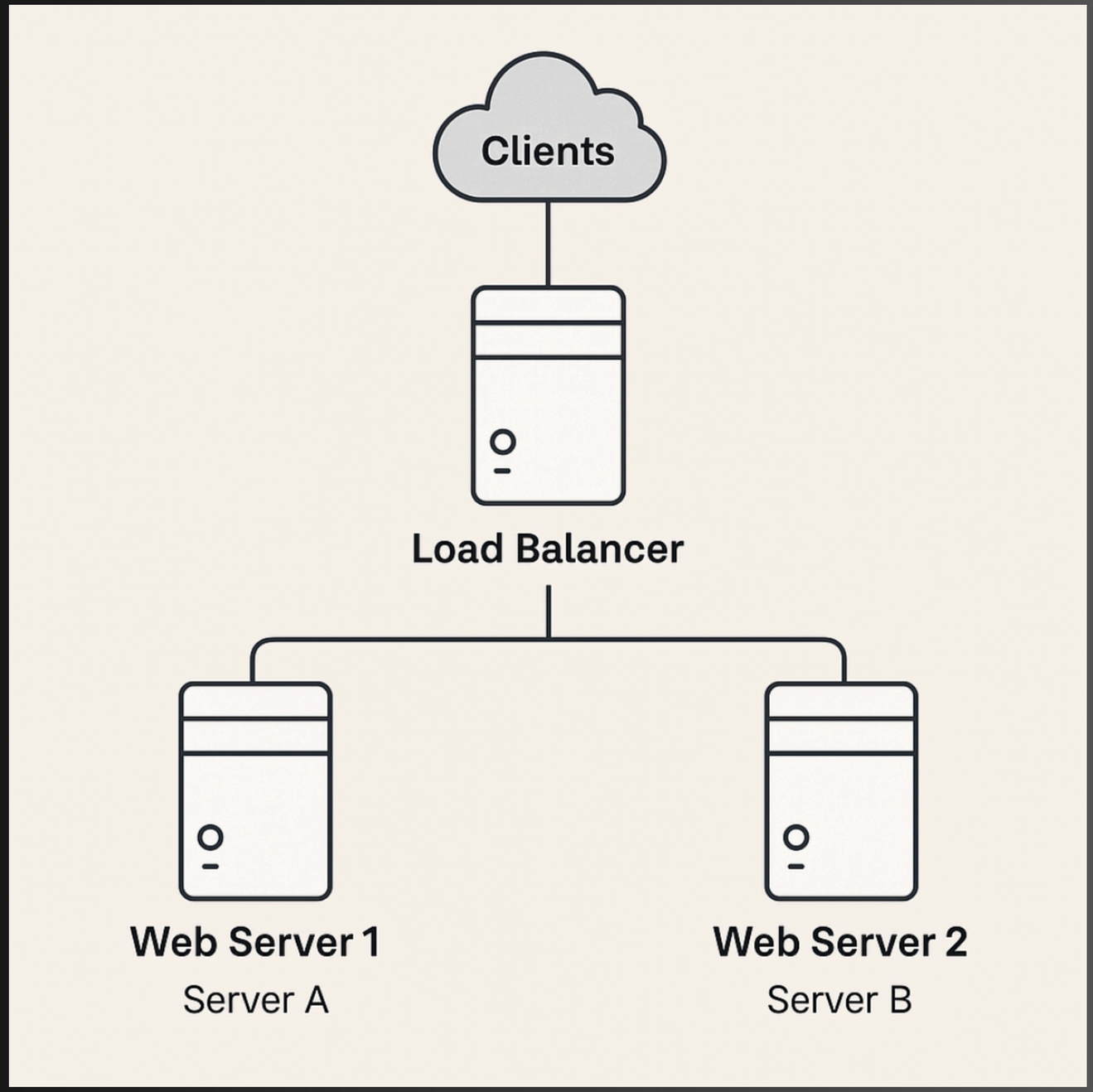
byrequests = basic round-robin method.
It rotates traffic across backends request-by-request, but doesn’t care if a backend is actually alive — unless you configure health checks.

Different Modes of LB

Method	Use Case	Pros	Cons
byrequests	Even traffic, general purpose	Simple and reliable	No traffic awareness
bytraffic	Mixed content sizes (e.g., media)	Traffic-sensitive	May not help with CPU
bybusyness	High concurrency or CPU-heavy apps	Load-sensitive	Less predictable pattern
heartbeat	High-availability, automatic failover	Smart failover detection	Requires extra setup

High Availability

High Availability (HA) in Apache Web Server ensure your website or app stays accessible even if one or more backend servers fail, by using load balancing, failover, and optionally health checks.


Clients
Load Balancer
Web Server 1
Server A
Web Server 2
Server B

Backup Server will only server in case of Primary Fail

```
<Proxy "balancer://mycluster">
   BalancerMember http://10.211.55.11:8084 hcheck=on
   BalancerMember http://localhost:8083 hcheck=on

   ProxySet lbmethod=byrequests \
         health=on \
         hcurl=/heartbeat \
         hcexpr=ok200
</Proxy>

ProxyPass "/" "balancer://mycluster/"
ProxyPassReverse "/" "balancer://mycluster/"

# Define the health check expression
ProxyHCExpr ok200 {%{REQUEST_STATUS} =~ /^2../}
```

In Each Backend server

```
<Location /heartbeat>
   SetHandler heartbeat
</Location>
```

🔐 `.htaccess` and `.htpasswd` in Apache Web Server

✅ Purpose:

They are used together to implement **Basic Authentication** — a simple way to **restrict access to certain parts of your website** (like admin pages or internal tools).

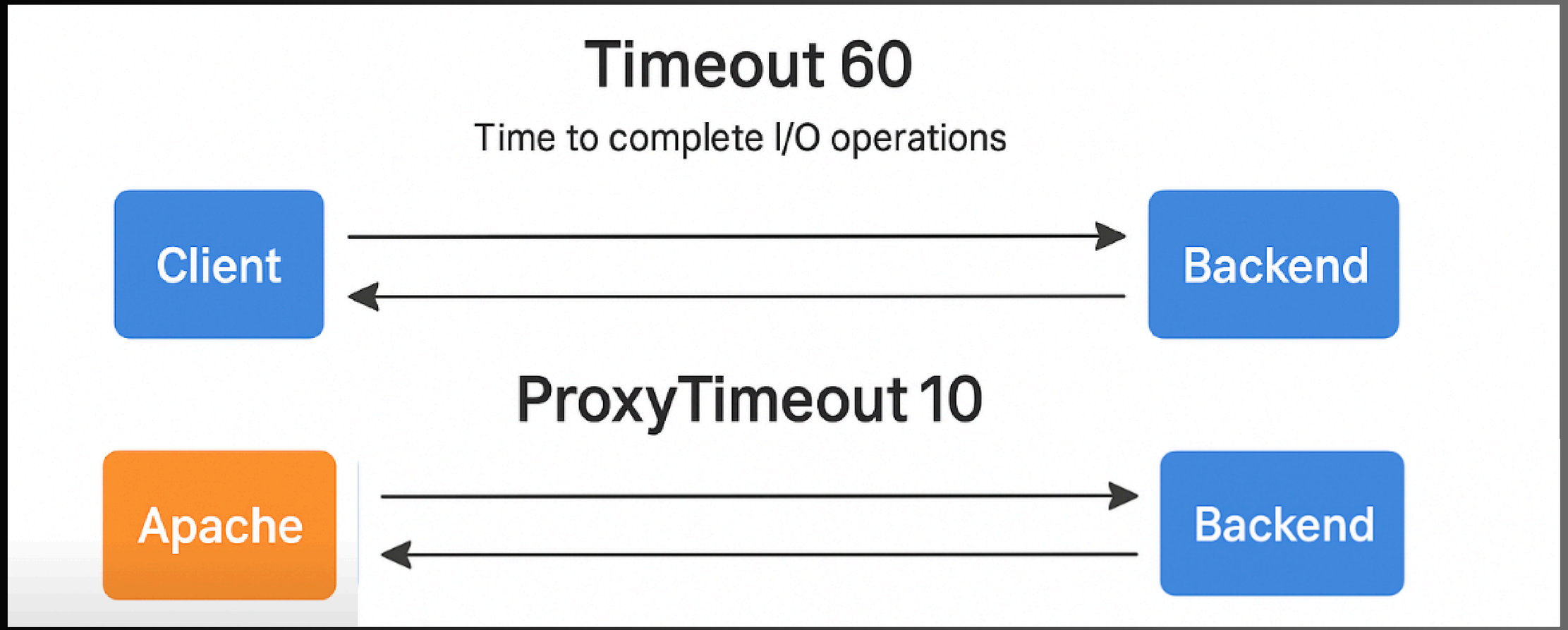
Timeouts

Timeout in a web server is the maximum amount of time the server will wait for a specific event (like

- receiving a request,
- a backend response,
- or completing a connection)

before it gives up and returns an error or closes the connection.

```
Timeout 60
ProxyTimeout 10

<Proxy "balancer://mycluster">
    BalancerMember http://10.0.0.1:8080 timeout=5 retry=15
    BalancerMember http://10.0.0.2:8080 timeout=5 retry=15
    ProxySet lbmethod=byrequests
</Proxy>
```


Timeout 60
Time to complete I/O operations
Client
Backend
ProxyTimeout 10
Apache
Backend

Catching

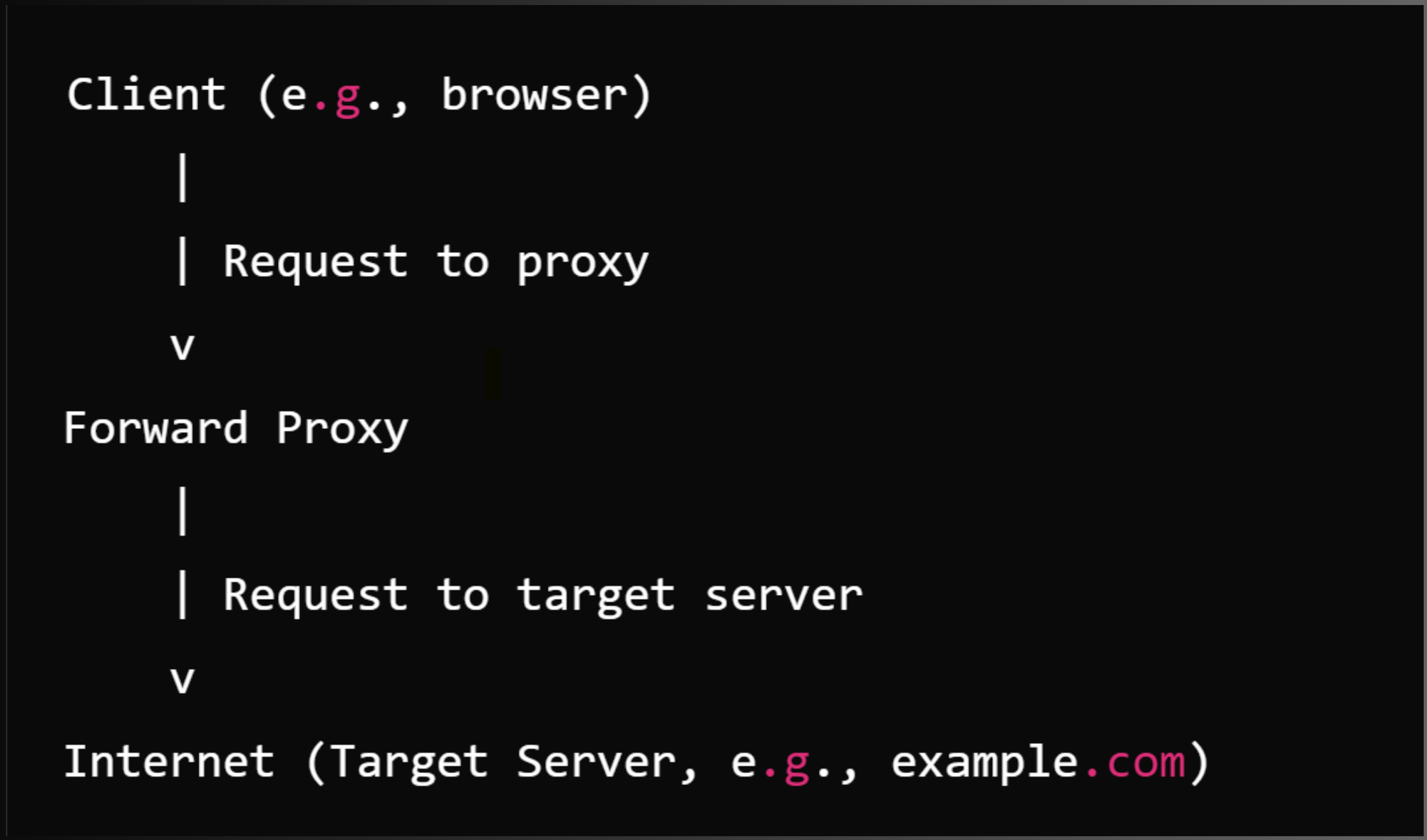
Caching in Apache means storing copies of responses (like HTML, images, CSS, etc.) either in memory or on disk — so Apache can serve them faster next time without regenerating or re-fetching them.

Types of Catching

Type	Modules Used	Where Cached	Best For
Static file cache	`mod_cache_disk`	Disk	HTML, CSS, JS
Memory cache	`mod_cache_socache`	RAM	Fast lookup
Reverse proxy	`mod_cache` + `mod_proxy`	Disk/memory	Backend content

Protocol	Description
HTTP/HTTPS	Used for serving web content. Supports HTTP/1.0, HTTP/1.1, HTTP/2, and HTTP/3 (with QUIC).
WebSocket	For real-time, full-duplex communication between clients and servers. Works over HTTP/HTTPS.
TCP/UDP	For generic stream-based or datagram-based traffic like database traffic or load balancing.
FastCGI	Commonly used for serving dynamic content, such as PHP applications.
gRPC	Handles RPC (Remote Procedure Call) communication, often with HTTP/2.
Mail (SMTP, IMAP, POP3)	Acts as a mail proxy server, supporting load balancing or securing mail servers.
Reverse Proxying (HTTPS/TLS Termination)	Works as a reverse proxy for web servers or services. Can terminate HTTPS/TLS connections.
QUIC and HTTP/3	Supports QUIC (UDP-based) for fast and efficient HTTP/3 connections.
SCGI and uWSGI	Used for interfacing with applications running on SCGI or uWSGI.
Streaming Protocols	Handles media streaming like HLS and RTMP (with additional modules).

Protocols


Client (e.g., browser)
|
| Request to proxy
v
Forward Proxy
|
| Request to target server
v
Internet (Target Server, e.g., example.com)

Interview Questions