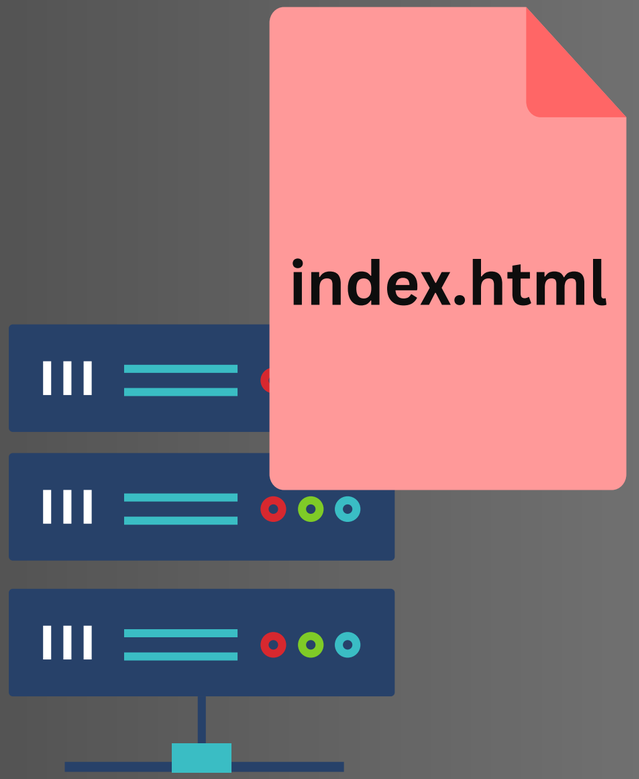
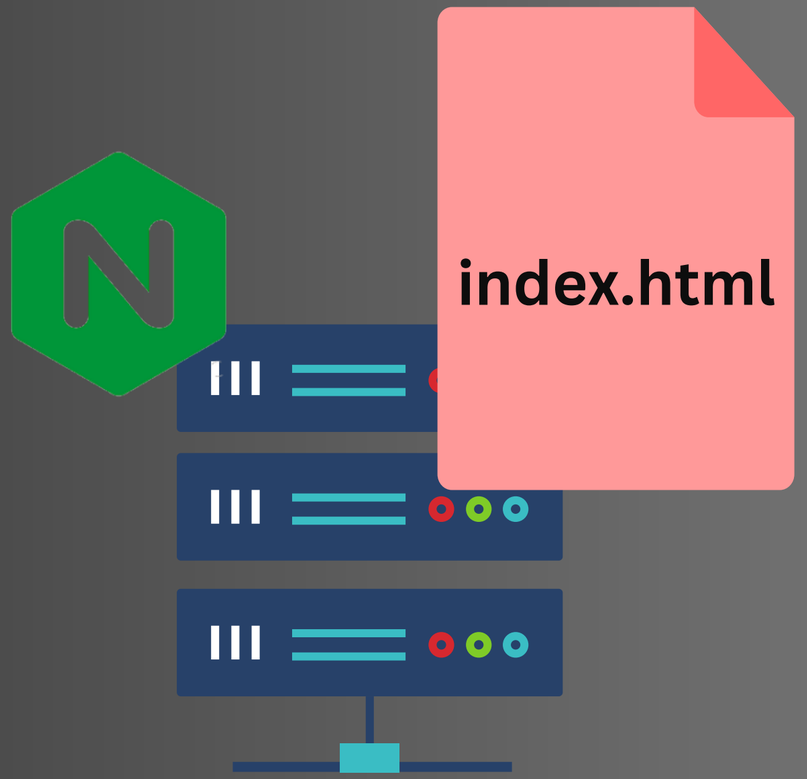
ALRIGHT!

What is Nginx?

ginx is a high-performance web server that delivers websites, handles requests, and can also work as a reverse proxy, load balancer, and caching tool to manage and scale web traffic efficiently.

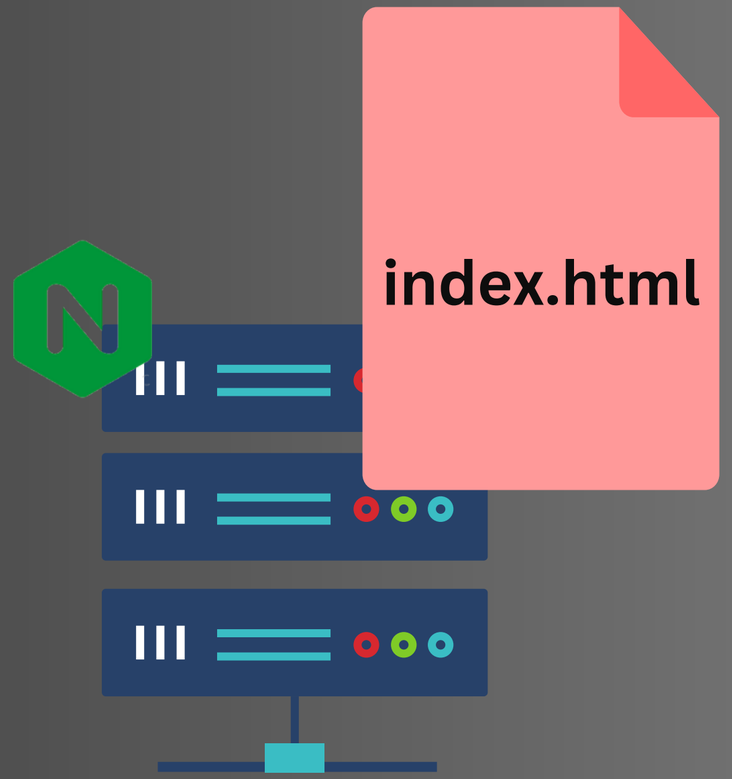


server

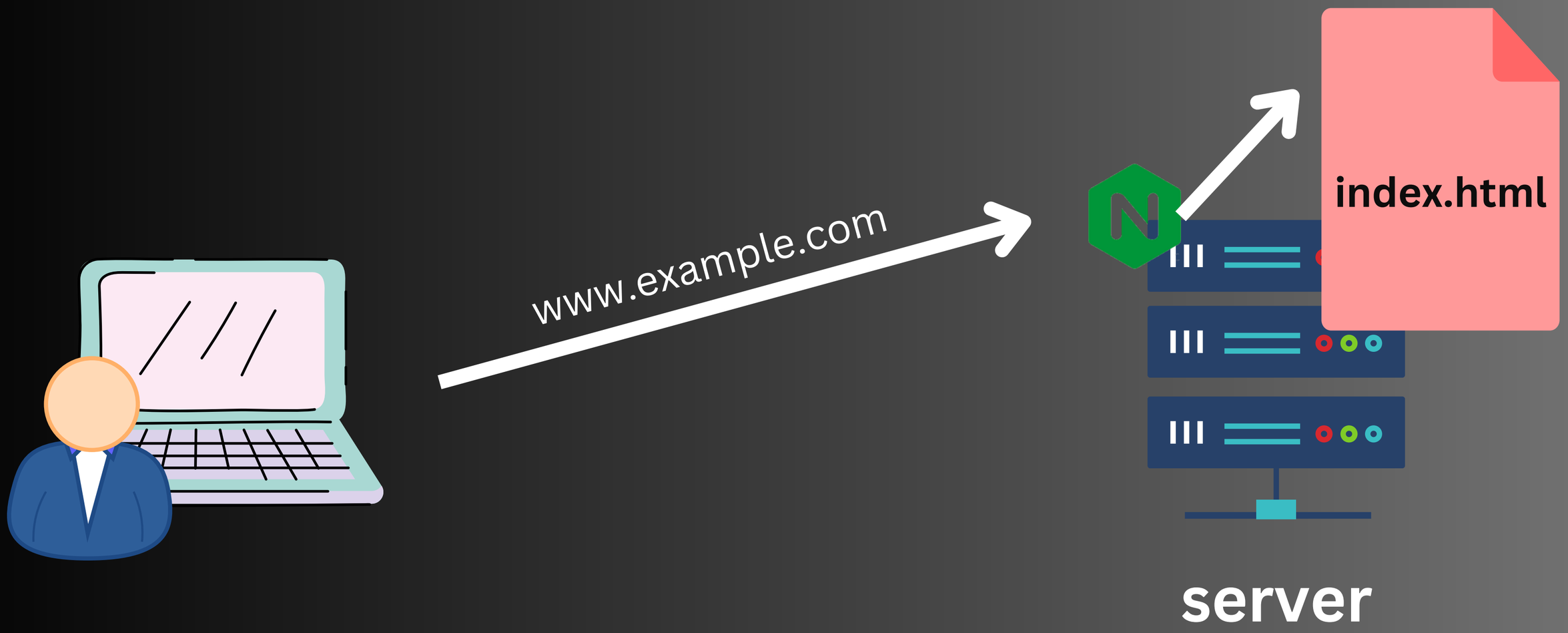


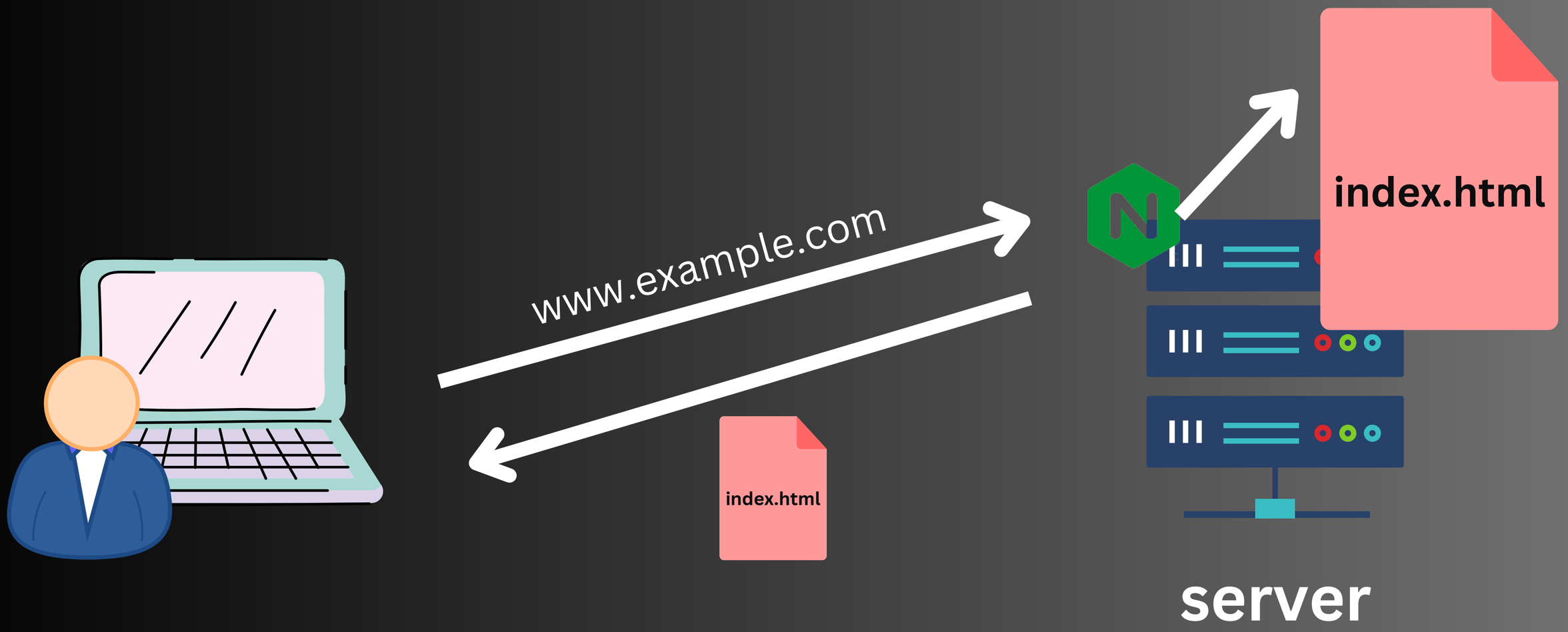
server

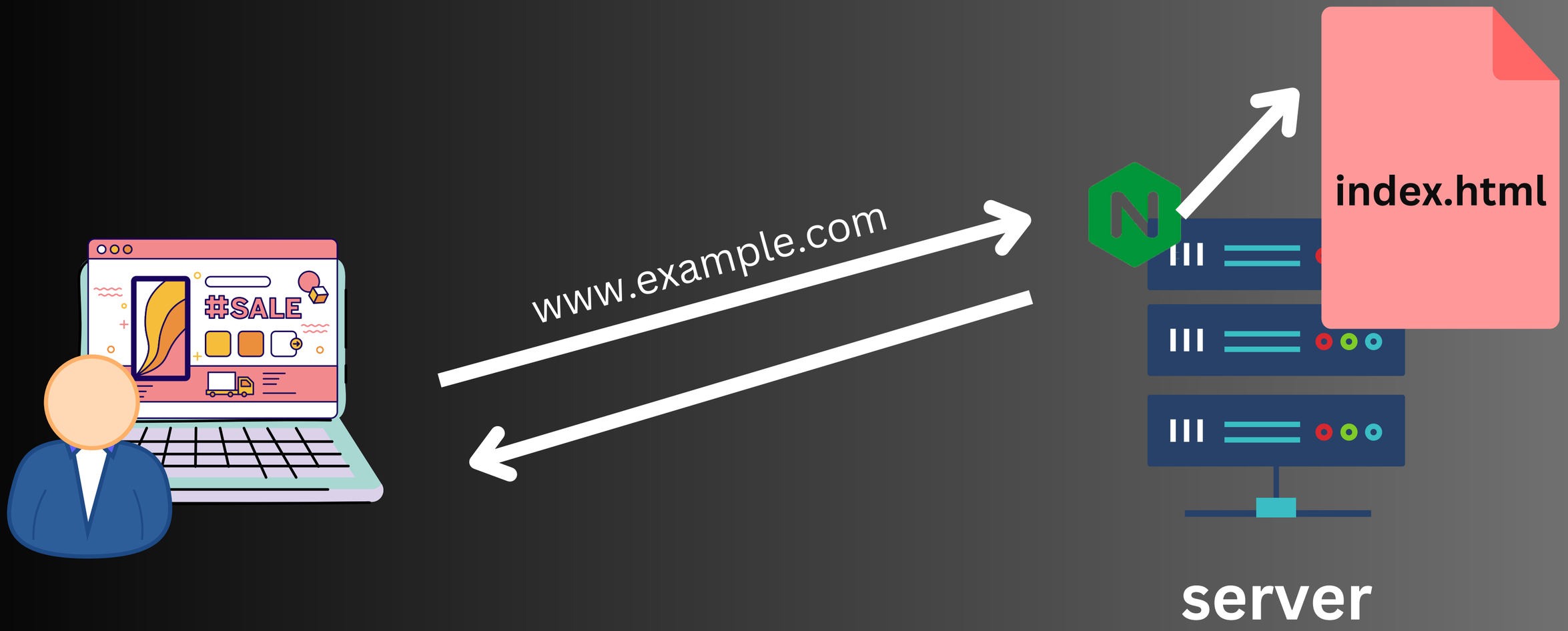
www.example.com



server


www.example.com
index.html
server


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


#SALE
www.example.com
index.html
server

Practical

Prerequisites

- AWS Account
- We will test this on EC2 instance

- Installing Nginx
  - yum install nginx
  - apt install nginx
- Start Nginx service
  - systemctl start/stop/reload nginx
- Enable HTTP service in firewalld
  - sudo firewall-cmd --permanent --add-service=http
- Access Default website from browser
- Nginx config
  - /etc/nginx/nginx.conf
- Default webpage
  - /usr/share/nginx/html

- Create our own static website
- Configure nginx config

Nginx Config
/etc/nginx/nginx.conf

```
user;

events {
  }

http {
    server {
        location { }
    }
}
```

High-Level Structure:

1. **Global Block:**
   - Applies settings globally (e.g., user, processes, logging).
2. **Events Block:**
   - Manages connection-level configurations (e.g., worker connections).
3. **HTTP Block:**
   - Contains configurations for handling HTTP requests:
     - MIME types, logging, and general settings.
     - Includes all server blocks.
4. **Server Block(s):**
   - Defines settings for handling requests for specific domains or IPs.
   - Can contain multiple location blocks.
5. **Location Block(s):**
   - Handles specific request URIs or patterns (e.g., /api/, *.php).

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

Nginx Logging

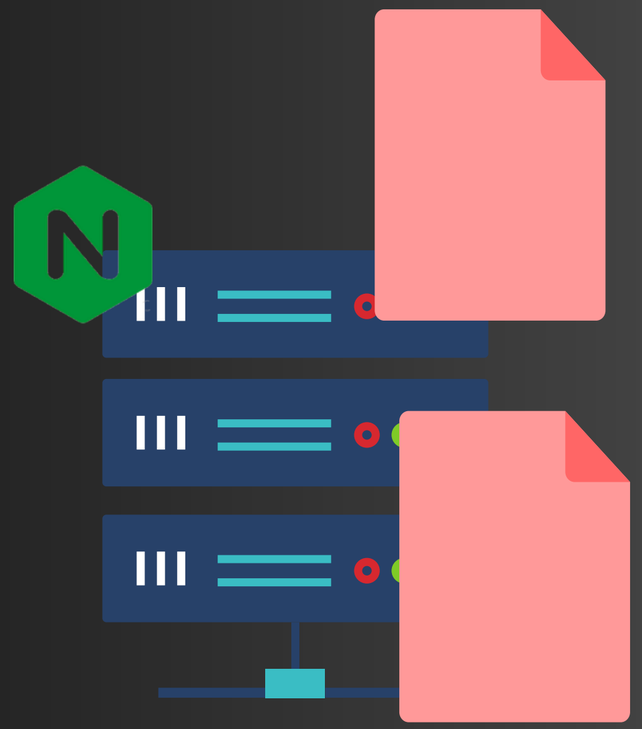
/var/log/nginx/

By default, worker_connections is often set to 1024.

- worker_processes 4 (for a 4-core CPU)
- worker_connections 10,000
- Total max connections = 4 × 10,000 = 40,000 connections

Note: If worker_processes = 4 and worker_connections = 10,000, the nginx user must have a file descriptor limit of at least 40,000.

Hosting Multiple Websites



website1

website2

server

```
http {

    server {
        location { }
    }
    server {
        location { }
    }

}
```

Custom Domain

- Register a domain name.
- In DNS settings, create a A record and point to public IP of our server (EC2)
- In nginx.conf, server block, use
    - server_name www.example.com;

```
server {
    listen 80;
    server_name yourdomain.com www.yourdomain.com;

    root /usr/share/nginx/html/;
    index index.html index.htm;

    location / {
        try_files $uri $uri/ =404;
    }

    error_page 404 /404.html;
    location = /404.html {
        root /usr/share/nginx/html/;
    }

    access_log /var/log/nginx/yourdomain_access.log;
    error_log /var/log/nginx/yourdomain_error.log;
}
```

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.

- 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-nginx -y
  - Ubuntu
    - sudo apt install certbot python3-certbot-nginx -y

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

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

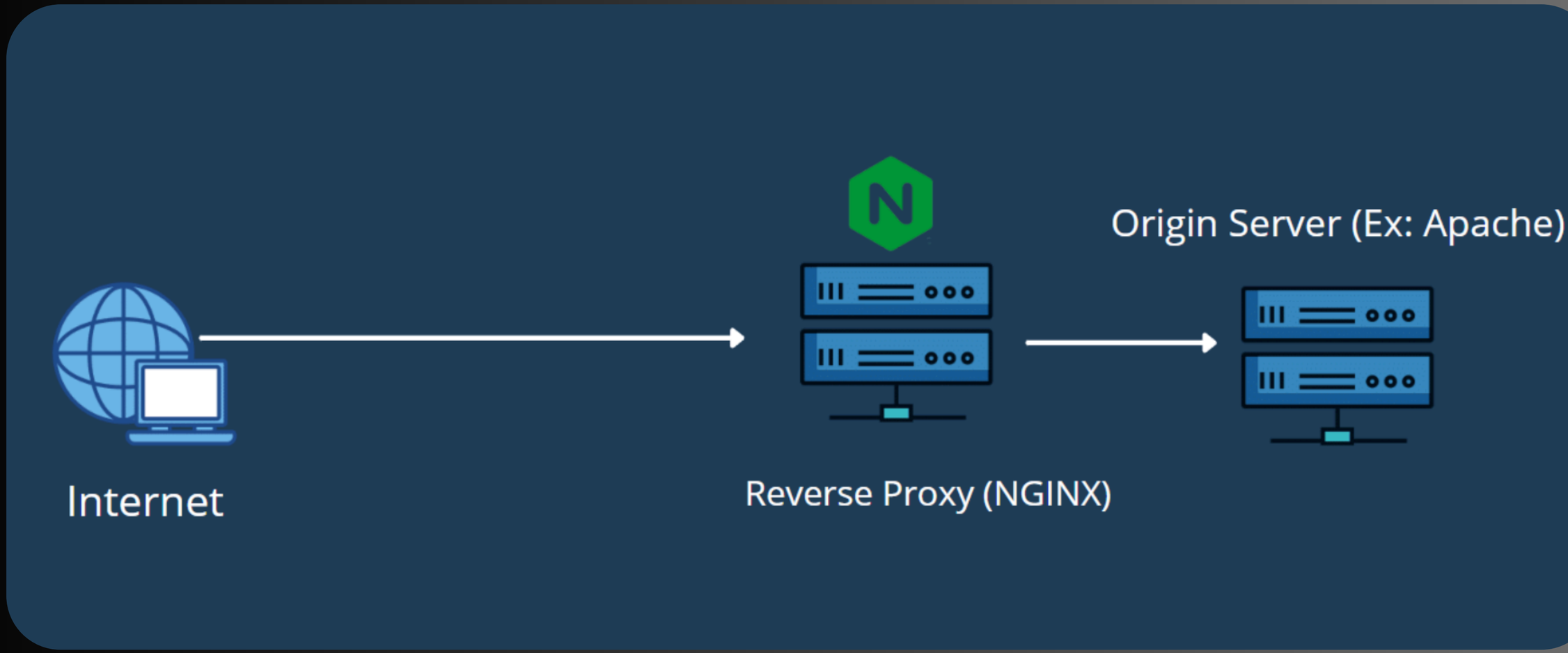
- Files will be generated in
  - /etc/letsencrypt/live/yourdomain.com/

```
listen       443 ssl http2;

ssl_certificate /etc/letsencrypt/live/yourdomain.com/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/yourdomain.com/privkey.pem;

ssl_protocols TLSv1.2 TLSv1.3;
```

Reverse Proxy

An Nginx 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 (Ex: Apache)
Internet
Reverse Proxy (NGINX)

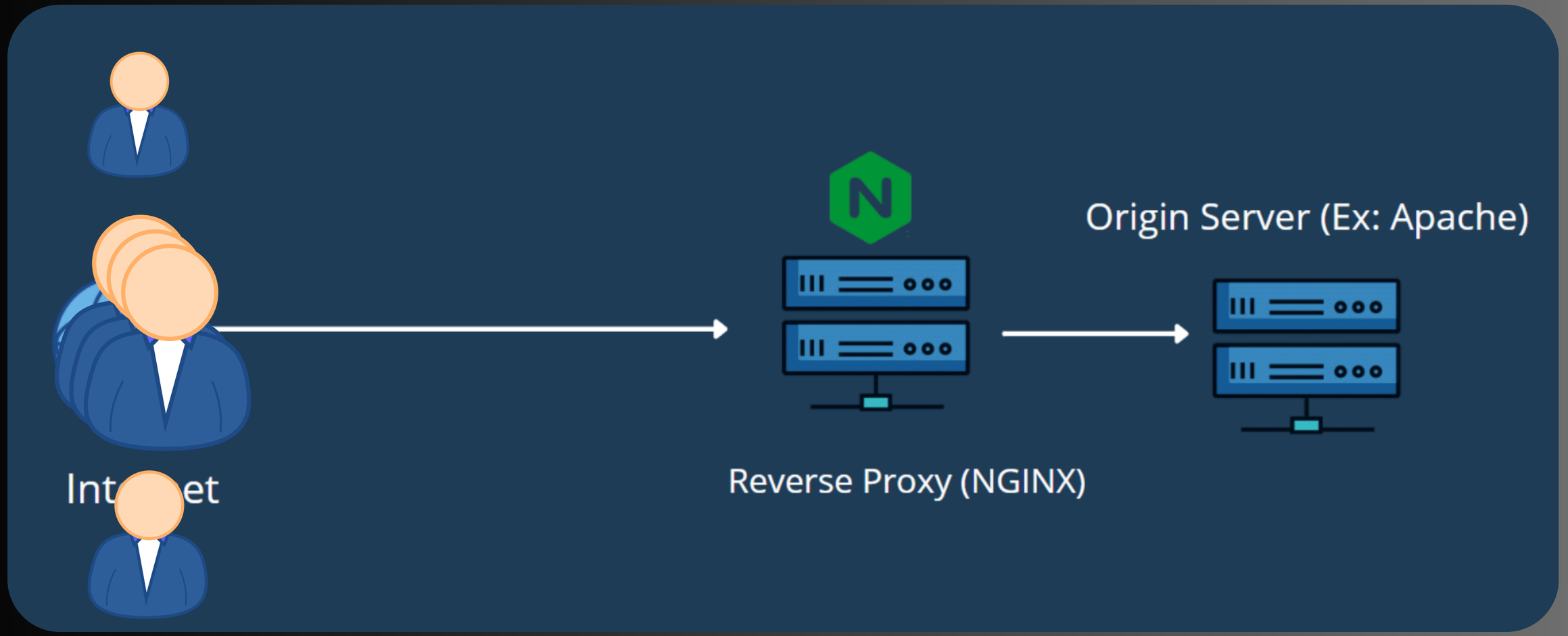
```
server {
    listen 80;
    server_name example.com;

    location / {
        proxy_pass http://127.0.0.1:3000;
     }
}
```

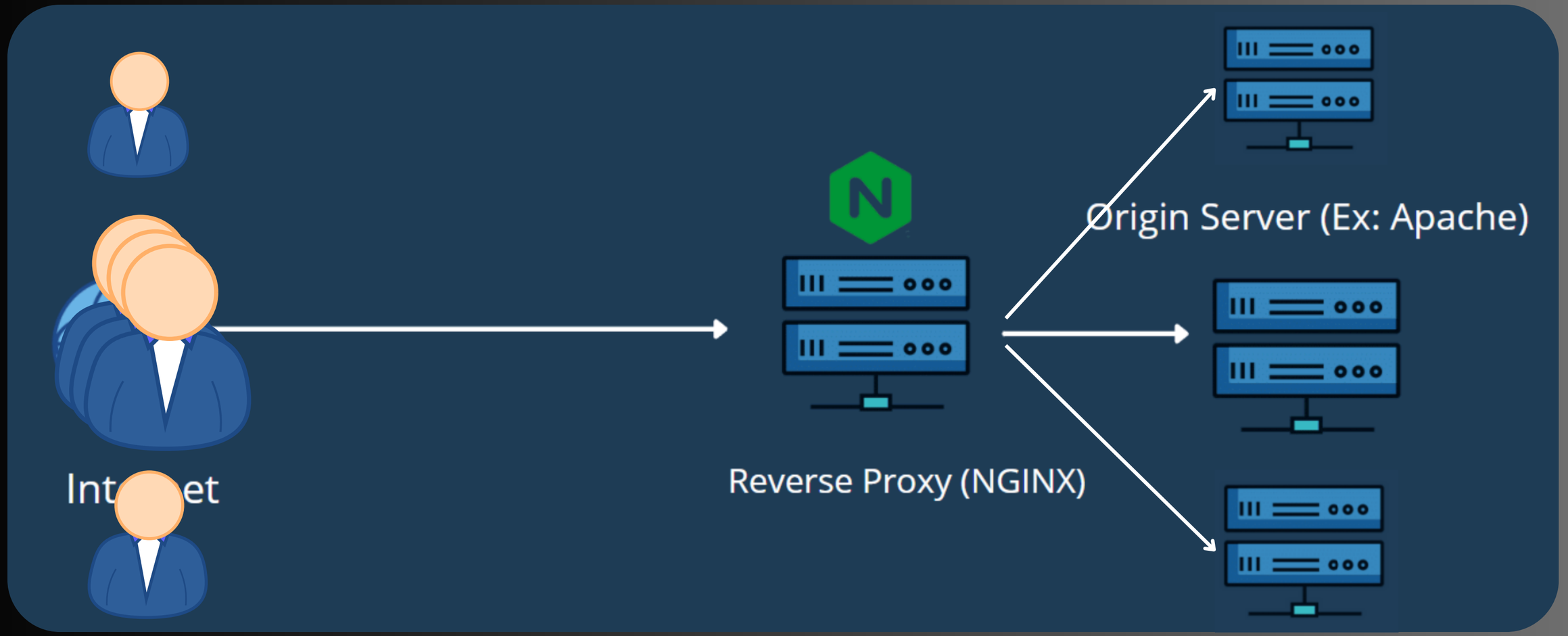
Userdata script to install and run Apache Webserver

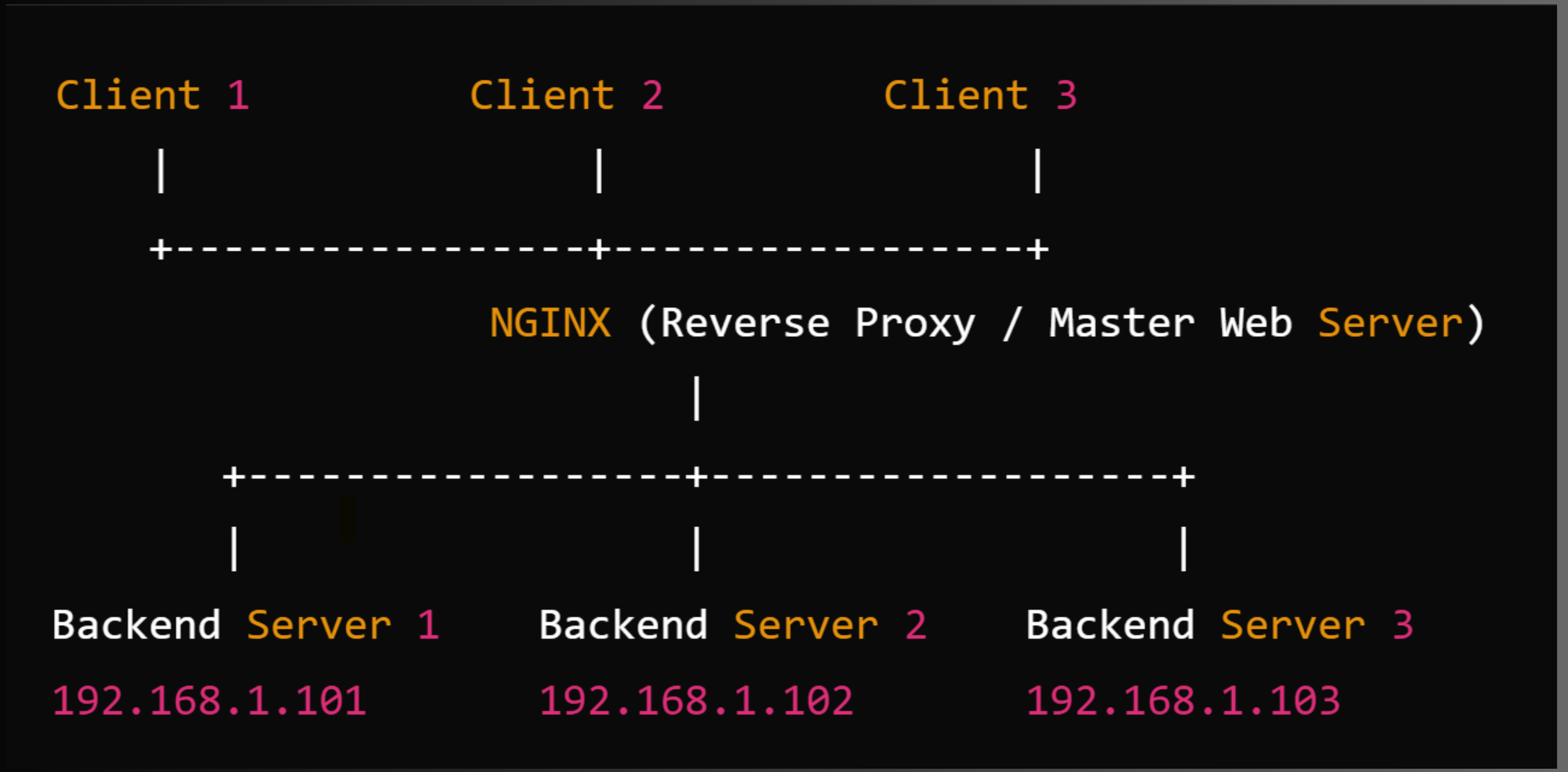
```
#!/bin/bash
sudo yum update -y

# Install Apache web server (httpd)
sudo yum install -y httpd
sudo systemctl start httpd
sudo systemctl enable httpd

# Create a simple HTML file to verify the web server is running
echo "<html><h1>This is Website 1</h1></html>" > /var/www/html/index.html
```

Load Balancing

Nginx load balancing is a feature where Nginx distributes incoming traffic across multiple backend servers to ensure no single server gets overloaded, improving performance, reliability, and scalability of your application.


Origin Server (Ex: Apache)
Reverse Proxy (NGINX)


Origin Server (Ex: Apache)
Reverse Proxy (NGINX)


Client 1
Client 2
Client 3
NGINX (Reverse Proxy / Master Web Server)
Backend Server 1
Backend Server 2
Backend Server 3
192.168.1.101
192.168.1.102
192.168.1.103

```
http {
    upstream backend_servers {
        server 127.0.0.1:3000;
        server 127.0.0.1:3001;
        server 127.0.0.1:3002;
    }

    server {
        listen 80;
        server_name example.com;

        location / {
            proxy_pass http://backend_servers;
        }
    }
}
```

Different Modes of LB

- **Round-Robin (Default)**: Evenly distributes requests across servers; no configuration needed.
- **Least Connections**: Sends requests to the server with the fewest active connections (least_conn).
- **IP Hash**: Routes requests based on the client’s IP to ensure session stickiness (ip_hash).
- **Weighted Round-Robin**: Assigns weights to servers to handle traffic proportionally (server 127.0.0.1 weight=3;).

High Availability

Backup Server will only server in case of Primary Fail

```
upstream backend {
    server 192.168.1.1:3000;
    server 192.168.1.2:3000;
    server 192.168.1.3:3000 backup;
}
```

Nginx Timeouts

```
# Frontend (Client-Side) Timeouts
client_header_timeout 10s;
client_body_timeout 10s;
send_timeout 15s;
keepalive_timeout 20s;

# Backend (Upstream Server) Timeouts
proxy_connect_timeout 5s;
proxy_read_timeout 30s;
proxy_send_timeout 30s;
```

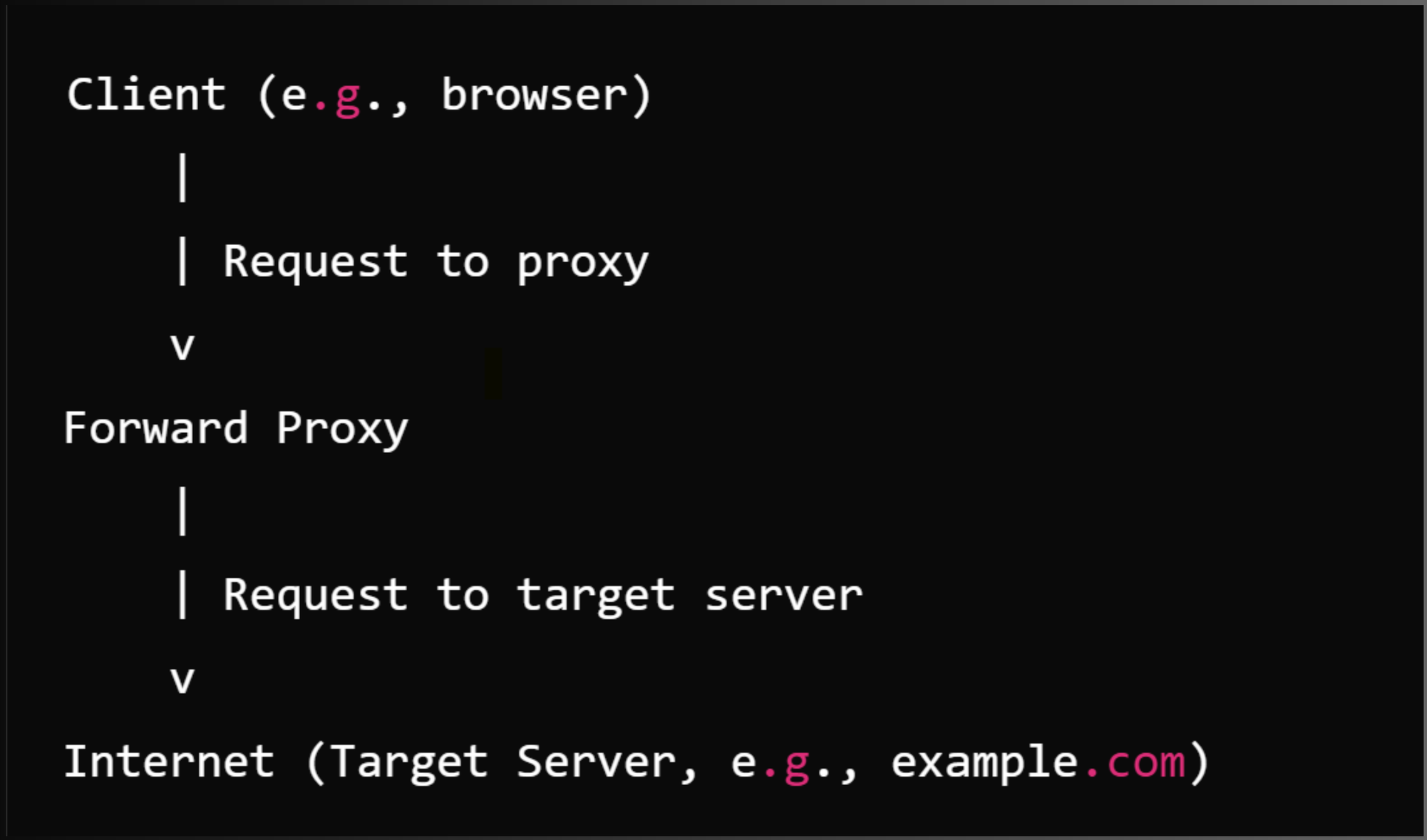
Nginx Catching

Nginx caching is a process where Nginx stores copies of responses (like HTML, images, or API data) to serve them directly to users, reducing backend load and improving response times.

```
http {
    # Define the cache path
    proxy_cache_path /var/cache/nginx levels=1:2 keys_zone=my_cache:10m inactive=60m
max_size=1g;

server {
        listen 80;
        server_name example.com;

        location / {
            proxy_cache my_cache;               # Enable caching using the defined cache
            proxy_cache_valid 200 60m;          # Cache 200 OK responses for 60 minutes
            proxy_cache_key "$scheme$request_uri";  # Define the cache key
            proxy_pass http://backend_server;      # Forward requests to the backend server
            add_header X-Cache-Status $upstream_cache_status; # Add cache status header for debugging
        }
```

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

Basic Questions

1. What is Nginx, and how does it differ from Apache?
2. Explain the difference between Nginx as a web server and as a reverse proxy.
3. What is the default configuration file for Nginx?
4. How do you enable HTTP/2 in Nginx?
5. How do you troubleshoot Nginx issues?

Intermediate Questions

- What is the role of proxy_pass in Nginx?
- How do you configure Nginx as a load balancer?
- How does Nginx handle caching?
- How do you set up SSL/TLS with Nginx?
- How do you configure a custom 404 page?
- How do you block specific IP addresses in Nginx?

Feature	NGINX 🚀	Apache 🔥
Architecture	Event-driven (asynchronous, non-blocking)	Process/thread-based (blocking)
Performance	Handles high concurrency efficiently	Struggles with high concurrency
Static Content	Faster (directly serves files)	Slower (spawns processes/threads)
Dynamic Content (PHP, etc.)	Requires external processors (FastCGI, PHP-FPM)	Built-in processing (mod_php)
Configuration	Uses `location` blocks (simpler for routing)	Uses `.htaccess` (flexible but slower)
Memory Usage	Low (efficient resource usage)	High (spawns multiple processes)
Best Use Case	High-traffic sites, load balancing, reverse proxy	Smaller, dynamic content-heavy apps

- NGINX-Based Scalable Architecture on AWS
- Set up an NGINX web server on AWS EC2 to serve static and dynamic content.
- Configured NGINX as a reverse proxy to forward requests to backend services.
- Implemented load balancing across multiple EC2 instances to handle high traffic efficiently.
- Secured the application with SSL/TLS certificates (Let's Encrypt, Certbot).
- Managed DNS settings for a custom domain and ensured high availability.