**The Summery of NGINX**

**Introduction**

Nginx (pronounced "engine x" /ˌɛndʒɪnˈɛks/ EN-jin-EKS, stylized as NGINX or nginx) is a web server that can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache. The software was created by Russian developer Igor Sysoev and publicly released in 2004.Nginx is free and open-source software, released under the terms of the 2-clause BSD license. A large fraction of web servers uses Nginx, often as a load balancer.

**What is NGINX?**

NGINX (pronounced "engine-x") is a high-performance web server and reverse proxy server that also works as:

Load balancer

HTTP cache

Mail proxy

Originally designed to handle high concurrency with low memory usage, it’s widely used for serving websites, APIs, and apps due to its speed and efficiency.

**Why Use NGINX?**

Lightweight and Fast

Handles 10,000+ concurrent connections

Open-source and actively maintained

Used by top sites (Netflix, Dropbox, GitHub)

Nginx may be configured to serve static web content or to act as a proxy server.

Nginx can be deployed to also serve dynamic content on the network using FastCGI, SCGI handlers for scripts, WSGI application servers or Phusion Passenger modules, and can serve as a software load balancer.

Nginx uses an asynchronous event-driven approach, rather than threads, to handle requests. Nginx's modular event-driven architecture can provide predictable performance under high loads.

**What is Caching in NGINX?**

Caching in NGINX refers to storing a copy of a response (like an HTML page or API result) so that the next time someone asks for the same resource, NGINX can deliver it faster without contacting the backend server again.

**Why Use Caching?**

Improves speed and user experience

Reduces load on backend servers

Saves bandwidth and system resources

**Types of Caching in NGINX**

Static File Caching (by browser):

You set headers to tell the browser to cache static files (images, CSS, JS).

Proxy Caching (by NGINX):

NGINX caches content from an upstream server (like a Node.js/PHP API).

**NGINX configuration**

Install NGINX

sudo apt update

sudo apt install nginx

nginx is a powerful web server used to serve websites, static content, or proxy traffic.

Check NGINX Status

sudo systemctl status nginx

Shows whether NGINX is active, inactive, or failed.

Useful for confirming that the service is running correctly.

Start / Stop / Restart NGINX

sudo systemctl start nginx Starts NGINX if not running

sudo systemctl stop nginx Stops NGINX

sudo systemctl restart nginx Restarts (apply changes)

NGINX Configuration Directory

NGINX reads its settings from config files in /etc/nginx/.

Important subdirectories:

* /etc/nginx/nginx.conf: **Main global configuration**
* /etc/nginx/sites-available/: Place to **create custom site configs**
* /etc/nginx/sites-enabled/: Stores **enabled sites** (via symbolic links)

Create a New Site Configuration

E.g.: /etc/nginx/sites-available/sample.lk

server {

listen 80;

server\_name sample.lk;

location / {

root /var/www/sample.lk;

index index.html;

}

}

Link Site to Enable It

sudo ln -s /etc/nginx/sites-available/sample.lk /etc/nginx/sites-enabled/

This command creates a symbolic link, making the config file active.

NGINX reads only files in sites-enabled/, so this step is required.

Test Configuration & Reload NGINX

sudo nginx -t

sudo systemctl reload nginx

nginx -t: Tests for syntax errors in config

reload: Applies changes **without stopping the server**

Access from Browser or curl

Open browser: http://localhost/ or <http://sample.lk/>

curl <http://localhost/>

**Different Services Under the Same Domain**

The task involved configuring NGINX on a Kali Linux system to serve multiple independent services under a single domain by assigning each service a unique path (e.g., /service1, /service2, /service3). This approach allows different applications or static web content to be hosted on the same domain, simplifying access and improving organization. Each service is separated logically by URL paths, making it easier to manage and scale without the need for multiple domains or subdomains.

By using NGINX as a reverse proxy and web server, requests to specific URL paths are directed to corresponding directories or backend services. This setup enhances flexibility, especially in development and testing environments, and is a common technique in microservice architecture and containerized applications. The configuration helps reduce server resource usage, improves maintainability, and enables centralized control over routing and traffic management.

Serve different services under the same domain using URL paths like:

* http://sample.lk/service1
* http://sample.lk/service2
* <http://sample.lk/service3>

**Created Document Root**

sudo mkdir -p /var/www/sample1.lk/service1

sudo mkdir -p /var/www/sample1.lk/service2

sudo mkdir -p /var/www/sample1.lk/service3

**Added Sample HTML Files**

echo "<h1>Service 1 Page</h1>" | sudo tee /var/www/sample1.lk/service1/index.html

These commands added basic HTML files to each service directory to test them.

**Created NGINX Configuration File**

created /etc/nginx/sites-available/sample1.lk and added the proper configuration

**Enabled Site**

sudo ln -s /etc/nginx/sites-available/sample1.lk /etc/nginx/sites-enabled/

**Started and Enabled NGINX**

sudo systemctl start nginx

sudo systemctl enable nginx

**Verified it Works**

curl http://sample1.lk/service1/

curl http://sample1.lk/service2/

curl <http://sample1.lk/service3/>

All services returned the correct HTML content – success

**Reverse Proxy Configuration**

A reverse proxy is a server that sits between clients and backend servers, forwarding client requests to the appropriate backend service. It acts as an intermediary, hiding the complexity and details of the backend infrastructure from the user. This setup is commonly used to distribute traffic, improve security by hiding internal servers, enable load balancing, and allow multiple services to be accessible via a single public IP or domain. By using a reverse proxy, organizations can efficiently manage requests, cache content, and enforce security policies without exposing backend servers directly to the internet.

Configuring a reverse proxy with NGINX involves setting up server blocks and defining location directives that map incoming requests to backend services running on different ports or machines. The configuration typically includes specifying the proxy pass directive to forward requests to the target backend URL, along with headers such as Host and X-Real-IP to preserve client information. After writing the configuration file, it is essential to test the setup using nginx -t to check for syntax errors and then reload NGINX to apply the changes. This configuration allows multiple web applications or APIs to be served through a single domain, simplifying deployment and improving resource management.

**NGINX reverse proxy configuration that forwards requests from different paths to different backend services**

server {

listen 80;

server\_name sample.lk;

location /service1/ {

proxy\_pass http://localhost:3001/;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

}

location /service2/ {

proxy\_pass http://localhost:3002/;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

}

}

Testing the configuration

**sudo nginx -t**

Result

nginx: the configuration file /etc/nginx/nginx.conf syntax is ok

nginx: configuration file /etc/nginx/nginx.conf test is successful

Test by accessing in your browser or using curl

curl http://sample.lk/service1/

curl <http://sample.lk/service2/>

I get responses from your backend services running on ports 3001 and 3002 respectively.