ChatGPT

Задание: Настройка NGINX как обратного прокси и развёртывание Hello World приложений

Часть 1: Настройка NGINX как обратного прокси

Цель:

- Проксировать запросы на разные бэкенды в зависимости от поддомена и пути.
- Использовать блоки `upstream`.
- Разделить логи.
- Обслуживать статику по пути `/static/`.
- Поддерживать SSL, Gzip-сжатие и кэширование.

Необходимые действия:

1. Установи nginx и openssl:

```
sudo apt update && sudo apt install nginx openssl -y
```

2. Запусти два простых бэкенда на Python:

```
# Терминал 1
python3 -m http.server 8001 --bind 127.0.0.1

# Терминал 2
python3 -m http.server 8002 --bind 127.0.0.1
```

3. Настрой NGINX:

4. Добавь в `/etc/nginx/nginx.conf` или в отдельный файл в `sites-available`:

```
upstream backend_app {
    server 127.0.0.1:8001;
}

upstream backend_api {
    server 127.0.0.1:8002;
}
```

1. Сгенерируй самоподписанный сертификат:

```
sudo mkdir -p /etc/nginx/ssl
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout /etc/nginx/ssl/selfsigned.key \
  -out /etc/nginx/ssl/selfsigned.crt \
  -subj "/CN=localhost"
```

2. Добавь конфигурации:

`/etc/nginx/sites-available/app.localhost`

```
server {
    listen 443 ssl;
    server_name app.localhost;

    ssl_certificate     /etc/nginx/ssl/selfsigned.crt;
    ssl_certificate_key /etc/nginx/ssl/selfsigned.key;

    gzip on;
    gzip_types text/plain application/json application/javascript text/css;

    access_log /var/log/nginx/app_access.log;
    error_log  /var/log/nginx/app_error.log;

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

    location /static/ {
        root /var/www/;
        expires 1h;
        add_header Cache-Control "public";
    }
}
```

`/etc/nginx/sites-available/api.localhost`

```
server {
    listen 443 ssl;
    server_name api.localhost;

    ssl_certificate     /etc/nginx/ssl/selfsigned.crt;
    ssl_certificate_key /etc/nginx/ssl/selfsigned.key;

    gzip on;
    gzip_types text/plain application/json application/javascript text/css;
```

```
    access_log /var/log/nginx/api_access.log;
    error_log  /var/log/nginx/api_error.log;

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

1. Активируй конфигурации:

```
sudo ln -s /etc/nginx/sites-available/app.localhost /etc/nginx/sites-enabled/
sudo ln -s /etc/nginx/sites-available/api.localhost /etc/nginx/sites-enabled/
sudo mkdir -p /var/www/static
echo "Hello from static!" | sudo tee /var/www/static/index.html
sudo nginx -t && sudo systemctl reload nginx
```

2. Добавь DNS-сопоставления:

Для Linux / macOS:

```
sudo nano /etc/hosts
```

Добавь строки:

```
127.0.0.1 app.localhost
127.0.0.1 api.localhost
```

Для Windows:

1. Открой блокнот от имени администратора
2. Открой файл `C:\Windows\System32\drivers\etc\hosts`
3. Добавь:

```
127.0.0.1 app.localhost
127.0.0.1 api.localhost
```

Часть 2: Развёртывание Hello World-бэкендов (если ты не программировал)

Цель:

Показать, как собрать и запустить минимальный сервер (бэкенд), который отвечает "Hello World". Это полезно, если ты никогда не писал код.

Python (Flask)

1. Установи:

```
sudo apt install python3-pip -y
pip3 install flask
```

2. Создай файл `app.py`:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello from Python Flask!"

app.run(host='0.0.0.0', port=8001)
```

3. Запусти:

```
python3 app.py
```

Node.js (Express)

1. Установи:

```
sudo apt install nodejs npm -y
```

2. Создай проект:

```
mkdir nodeapp && cd nodeapp
npm init -y
npm install express
```

3. Создай файл `index.js`:

```
const express = require('express');
const app = express();
app.get('/', (req, res) => res.send('Hello from Node.js Express!'));
app.listen(8002);
```

4. Запусти:

```
node index.js
```

Go

1. Установи:

```
sudo apt install golang -y
```

2. Создай файл `main.go`:

```
package main
import (
    "fmt"
    "net/http"
)
func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello from Go!")
}
func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8003", nil)
}
```

3. Запусти:

```
go run main.go
```

Java (Spring Boot CLI, упрощённо)

1. Установи SDKMAN и Spring Boot CLI:

```
curl -s "https://get.sdkman.io" | bash
source "$HOME/.sdkman/bin/sdkman-init.sh"
sdk install springboot
```

2. Создай:

```
spring init --dependencies=web demo
cd demo
```

3. Замени `src/main/java/com/example/demo/DemoApplication.java`:

```
package com.example.demo;
import org.springframework.boot.*;
import org.springframework.boot.autoconfigure.*;
import org.springframework.web.bind.annotation.*;

@RestController
@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

    @GetMapping("/")
    public String home() {
        return "Hello from Java Spring Boot!";
    }
}
```

4. Запусти:

```
./mvnw spring-boot:run
```

Проверочный чек-лист:

Этап	Проверка
	Установлен NGINX и SSL-сертификат
	Прокси работает через `app.localhost` и `api.localhost`
	Gzip и кэширование включены
	Приложения на Python, Node.js, Go, Java работают и отвечают Hello World

---

Теперь ты умеешь: - Настраивать NGINX как прокси - Поднимать HTTPS и обслуживать статику - Запускать простейшие серверные приложения на Python, Node.js, Go и Java