

Задание: Обратный прокси NGINX для 4 Hello World приложений с использованием Docker и Docker Compose

Цель

Развернуть 4 минимальных Hello World-приложения на разных языках (Python, Go, Java, Node.js) в контейнерах Docker и настроить NGINX (также в контейнере) как обратный прокси. NGINX должен направлять запросы на соответствующие приложения в зависимости от пути (/python, /go, /java, /node) и работать по HTTPS с самоподписанным сертификатом [\[1\]](#) [\[2\]](#) [\[3\]](#).

Шаг 1: Структура проекта

Создай структуру директорий:

```
project-root/
├── python_app/
│   ├── app.py
│   └── Dockerfile
├── node_app/
│   ├── index.js
│   └── Dockerfile
├── go_app/
│   ├── main.go
│   └── Dockerfile
├── java_app/
│   ├── DemoApplication.java
│   └── Dockerfile
├── nginx/
│   ├── nginx.conf
│   ├── ssl/
│   │   ├── selfsigned.crt
│   │   └── selfsigned.key
│   └── Dockerfile
└── docker-compose.yml
```

Шаг 2: Приложения и Dockerfile

Python (Flask) — порт 8001

python_app/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)
```

python_app/Dockerfile

```
FROM python:3.9-slim
WORKDIR /app
COPY app.py .
RUN pip install flask
CMD ["python", "app.py"]
```

Node.js (Express) — порт 8002

node_app/index.js

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

node_app/Dockerfile

```
FROM node:18-alpine
WORKDIR /app
COPY index.js .
RUN npm init -y && npm install express
CMD ["node", "index.js"]
```

Go — порт 8003

go_app/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)
}

```

go_app/Dockerfile

```

FROM golang:1.21-alpine
WORKDIR /app
COPY main.go .
RUN go build -o goapp main.go
CMD ["/goapp"]

```

Java (Spring Boot) — порт 8004

java_app/DemoApplication.java

Для простоты рекомендуется использовать готовый jar-файл Spring Boot Hello World или сгенерировать с помощью Spring Initializr (<https://start.spring.io/>). Далее пример Dockerfile для jar:

java_app/Dockerfile

```

FROM openjdk:17-alpine
WORKDIR /app
COPY demo-0.0.1-SNAPSHOT.jar app.jar
EXPOSE 8004
ENTRYPOINT ["java", "-jar", "app.jar", "--server.port=8004"]

```

Шаг 3: Генерация самоподписанного SSL-сертификата

Перейди в папку `nginx/ssl` и сгенерируй сертификат:

```

openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
    -keyout selfsigned.key \
    -out selfsigned.crt \
    -subj "/CN=helloworld.alinadobs"

```

Шаг 4: Конфигурация NGINX

nginx/nginx.conf

```
events {}

http {
    server {
        listen 443 ssl;
        server_name helloworld.alinadobs;

        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;

        location /python/ {
            proxy_pass http://python_app:8001/;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }
        location /node/ {
            proxy_pass http://node_app:8002/;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }
        location /go/ {
            proxy_pass http://go_app:8003/;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }
        location /java/ {
            proxy_pass http://java_app:8004/;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }
    }
}
```

nginx/Dockerfile

```
FROM nginx:alpine
COPY nginx.conf /etc/nginx/nginx.conf
COPY ssl/ /etc/nginx/ssl/
```

Шаг 5: docker-compose.yml

docker-compose.yml

```
version: '3.8'

services:
  python_app:
    build: ./python_app
    container_name: python_app
    expose:
      - "8001"

  node_app:
    build: ./node_app
    container_name: node_app
    expose:
      - "8002"

  go_app:
    build: ./go_app
    container_name: go_app
    expose:
      - "8003"

  java_app:
    build: ./java_app
    container_name: java_app
    expose:
      - "8004"

  nginx:
    build: ./nginx
    container_name: nginx_proxy
    ports:
      - "443:443"
    depends_on:
      - python_app
      - node_app
      - go_app
      - java_app
```

Шаг 6: Пропиши DNS (hosts)

Добавь строку в /etc/hosts:

```
127.0.0.1 helloworld.alinadobs
```

Шаг 7: Запуск

Выполни:

```
docker-compose up --build
```

✓ Чек-лист

Этап	Проверка
☐	Все контейнеры успешно собираются и запускаются
☐	Все 4 приложения работают на своих портах внутри сети Docker
☐	NGINX проксирует запросы с /python, /node, /go, /java
☐	HTTPS работает через <code>https://helloworld.alinadobs/...</code>

Теперь ты умеешь

- Поднимать многосервисные приложения через Docker Compose
- Настраивать обратный прокси для нескольких сервисов в Docker
- Работать с SSL в NGINX внутри контейнера
- Писать и запускать минимальные приложения на 4 языках в контейнерах ^{[1] [2] [3]}

Примечание: Для Java-приложения рекомендуется использовать уже собранный jar-файл Spring Boot, чтобы не усложнять Dockerfile и сборку ^[2].

Полезные ссылки для самостоятельного изучения:

- Пример репозитория с подобным заданием: <https://github.com/mohamedfazrin/helloworld> ^[2]
- Пример генерации self-signed сертификата для NGINX в Docker: <https://blog.devops.dev/nginx-with-self-signed-certificate-on-docker-fef9c6ead6fc> ^[3]
- Пример reverse proxy с Docker Compose: <https://blog.devops.dev/devops-setting-up-a-docker-reverse-proxy-nginx-multiple-local-apps-21b6f03eefa0> ^[1]

✱

1. <https://blog.devops.dev/devops-setting-up-a-docker-reverse-proxy-nginx-multiple-local-apps-21b6f03eefa0>
2. <https://github.com/mohamedfazrin/helloworld>
3. <https://blog.devops.dev/nginx-with-self-signed-certificate-on-docker-fef9c6ead6fc>