**Практическое занятие №4**

**Настройка Nginx как Reverse Proxy для Go-приложения с элементами безопасности**

**Цель:**  
Настроить Nginx как Reverse Proxy для Go-приложения, реализовать балансировку нагрузки и применить базовые меры защиты.

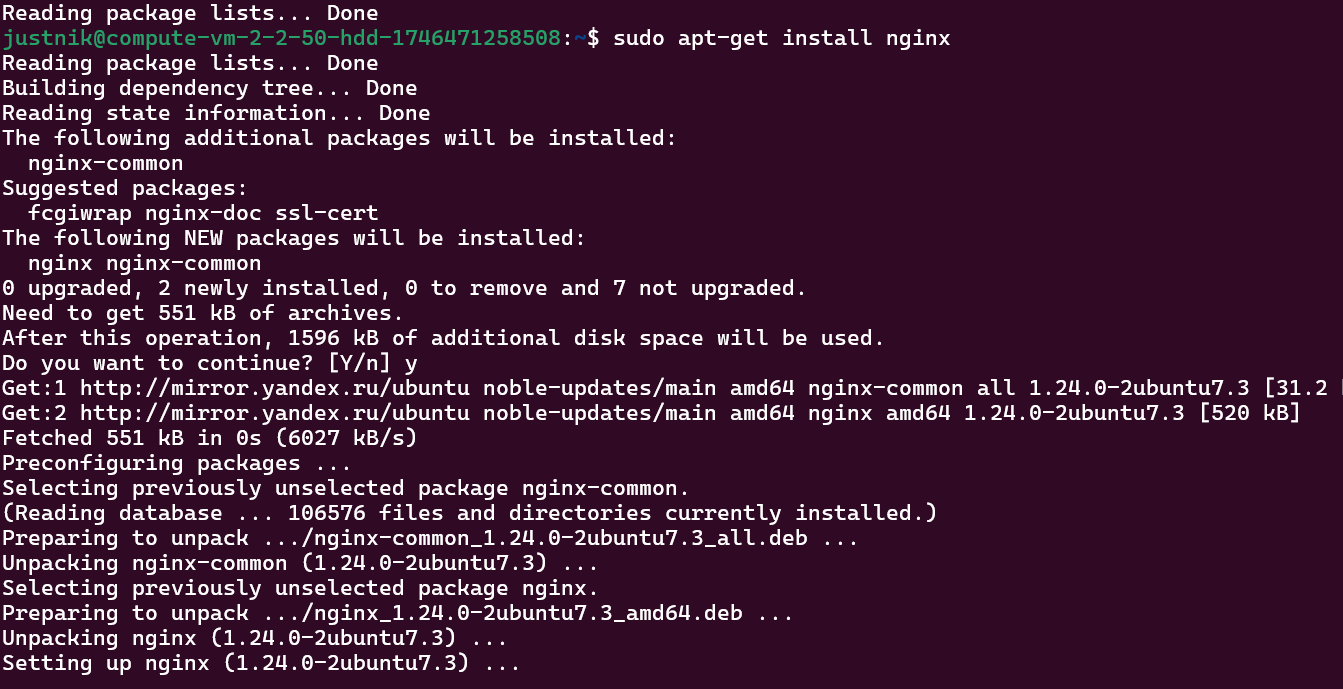
**Задание:**

1. **Установка и настройка Nginx**
2. Балансировка нагрузки
3. Настройка HTTPS и безопасного проксирования
4. Меры безопасности
5. Логирование и мониторинг
6. Подготовка отчета

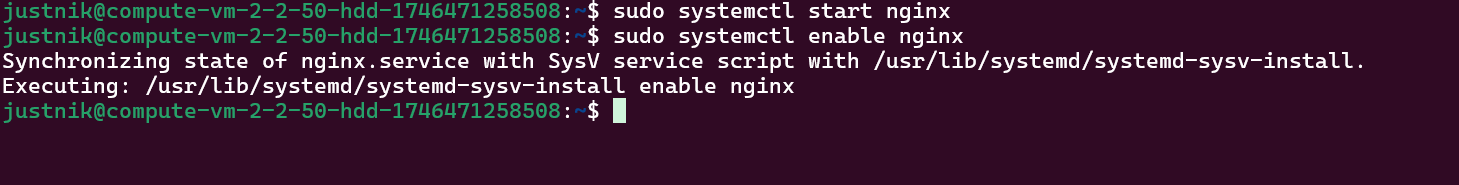
Выполнение работы:

**1. Установка и базовая настройка Nginx**

1. Установить Nginx (apt install nginx или yum install nginx).



1. Проверить запуск: systemctl start nginx и systemctl enable nginx.



1. Настроить базовую конфигурацию Nginx для проксирования Go-приложения (работающего на localhost:8080).

Пример конфигурации /etc/nginx/sites-available/app:

server {

listen 80;

server\_name example.com;

location / {

proxy\_pass http://localhost:8080;

proxy\_set\_header Host $host;

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

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

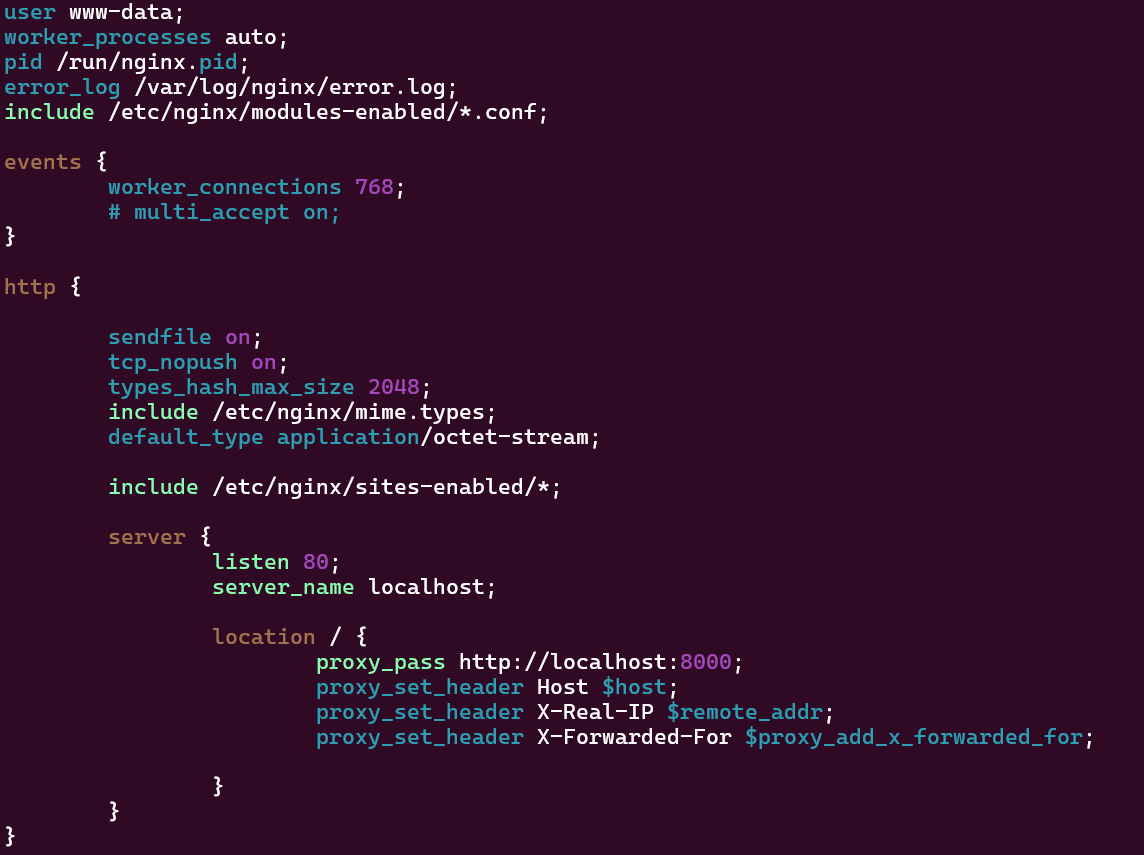
}

}

Применить конфигурацию:

ln -s /etc/nginx/sites-available/app /etc/nginx/sites-enabled/

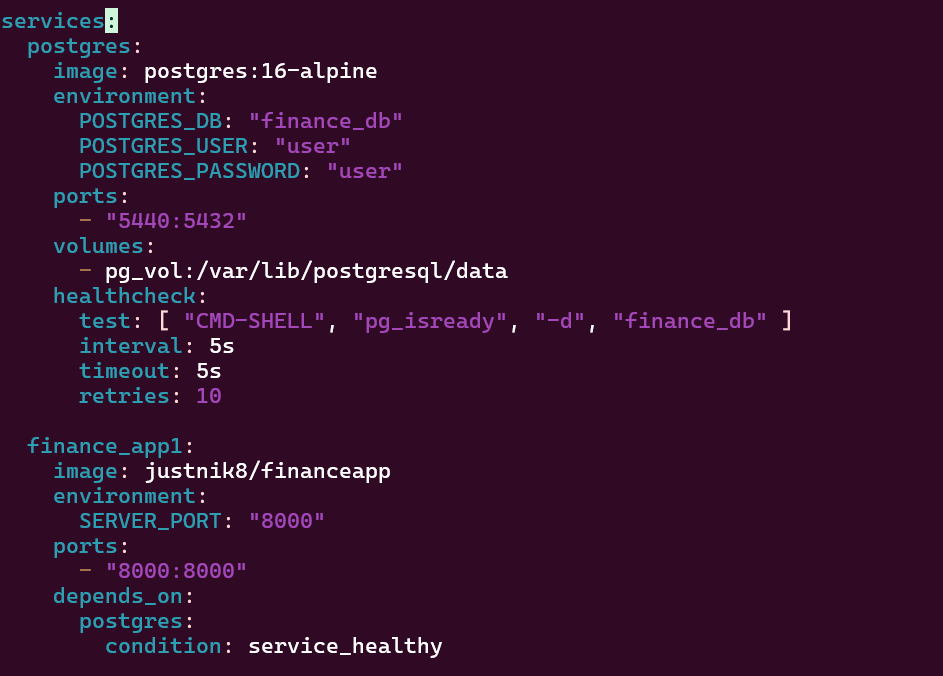
systemctl reload nginx



**2. Балансировка нагрузки**

1. Запустить несколько экземпляров Go-приложения (например, на портах 8080, 8081, 8082).

Docker compose для 3 экземплязров:





1. Добавить балансировку в Nginx:

Пример конфигурации с балансировкой:

upstream backend {

least\_conn; # Использование сервера с наименьшим числом соединений

server 127.0.0.1:8080;

server 127.0.0.1:8081;

server 127.0.0.1:8082;

}

server {

listen 80;

server\_name example.com;

location / {

proxy\_pass http://backend;

proxy\_set\_header Host $host;

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

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

}

}

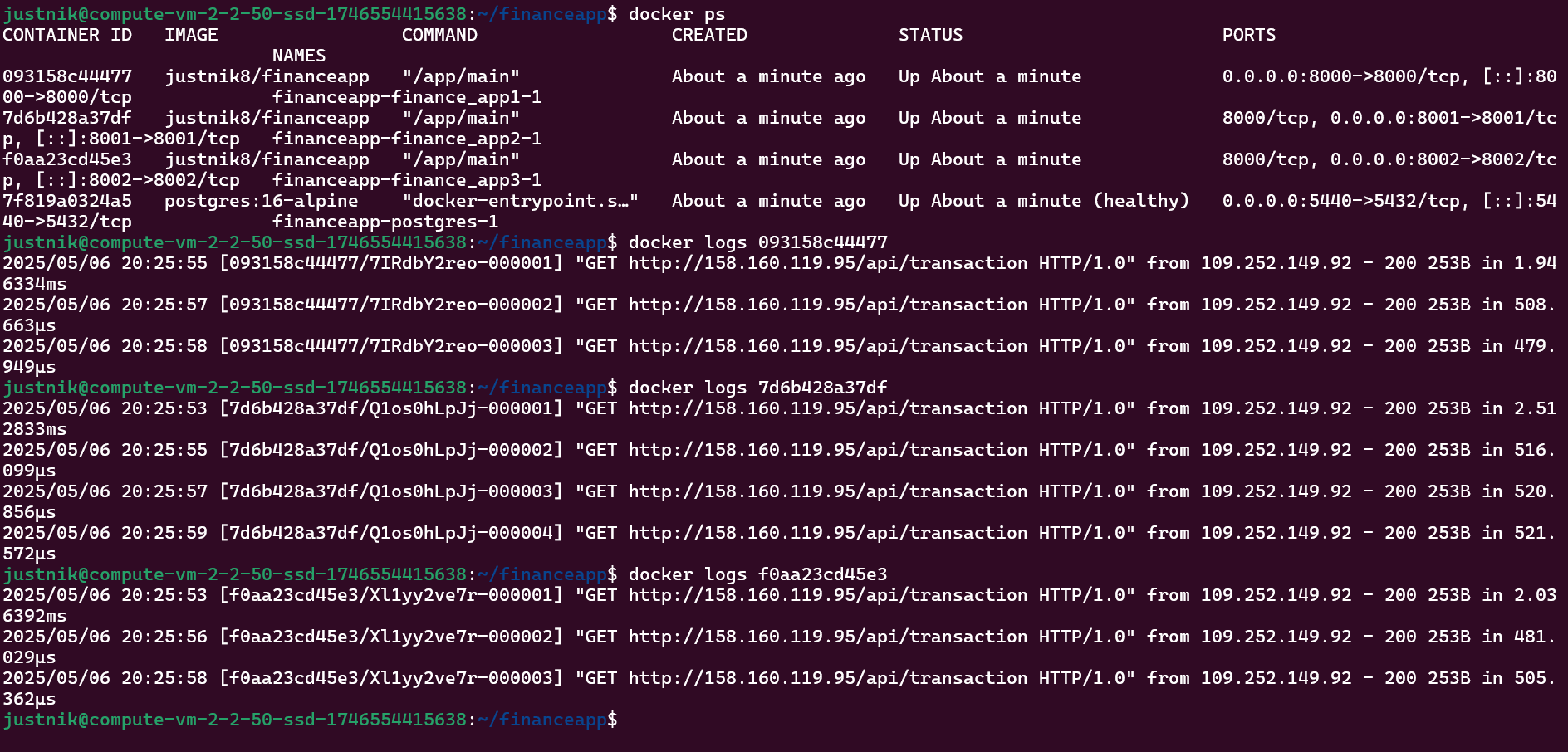
Перезапустить Nginx:

systemctl reload nginx

nginx конфигурация:



Проверка работы балансера (Отправлено 10 запросов):



**3. Настройка HTTPS и безопасного проксирования**

1. Установить Certbot и получить SSL-сертификат (если работает на сервере):

apt install certbot python3-certbot-nginx

certbot --nginx -d example.com

1. Обновить конфигурацию Nginx для HTTPS:

server {

listen 80;

server\_name example.com;

return 301 https://$host$request\_uri;

}

server {

listen 443 ssl;

server\_name example.com;

ssl\_certificate /etc/letsencrypt/live/example.com/fullchain.pem;

ssl\_certificate\_key /etc/letsencrypt/live/example.com/privkey.pem;

ssl\_protocols TLSv1.2 TLSv1.3;

ssl\_ciphers HIGH:!aNULL:!MD5;

location / {

proxy\_pass http://backend;

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 https;

}

}

Проверить сертификаты и автоматическое обновление:

certbot renew --dry-run

**4. Меры безопасности**

**Ограничение числа подключений** (защита от DDoS):  
Добавить в /etc/nginx/nginx.conf:

limit\_req\_zone $binary\_remote\_addr zone=one:10m rate=10r/s;

Затем в конфигурацию сервера:

location / {

limit\_req zone=one burst=20 nodelay;

proxy\_pass http://backend;

}

**Защита от брутфорса (напр. для /login)**:

location /login {

limit\_req zone=one burst=5 nodelay;

proxy\_pass http://backend;

}

**Защита от крупных запросов** (ограничение размера тела запроса):

client\_max\_body\_size 2M;

**Ограничение методов запроса** (например, запрет DELETE):

if ($request\_method !~ ^(GET|POST|PUT|HEAD)$) {

return 403;

}

**Ограничение доступа по IP** (например, запретить доступ к /admin с других IP):

location /admin {

allow 192.168.1.100;

deny all;

}

**Скрытие версий серверного ПО**: В /etc/nginx/nginx.conf:

server\_tokens off;

**5. Логирование и мониторинг**

Включить подробные логи запросов:

access\_log /var/log/nginx/access.log;

error\_log /var/log/nginx/error.log warn;

Установить и настроить fail2ban для защиты от атак:

apt install fail2ban

Создать конфигурацию /etc/fail2ban/jail.local:

[nginx-http-auth]

enabled = true

filter = nginx-http-auth

action = iptables-multiport[name=HTTP, port="http,https", protocol=tcp]

logpath = /var/log/nginx/error.log

bantime = 600

maxretry = 5

Итоговая конфигурация nginx:

|  |
| --- |
| user www-data;  worker\_processes auto;  pid /run/nginx.pid;  error\_log /var/log/nginx/error.log;  include /etc/nginx/modules-enabled/\*.conf;  events {  worker\_connections 768;  # multi\_accept on;  }  http {  server\_tokens off;  access\_log /var/log/nginx/access.log;  #sendfile on;  #tcp\_nopush on;  #types\_hash\_max\_size 2048;  #include /etc/nginx/mime.types;  #default\_type application/octet-stream;  #include /etc/nginx/sites-enabled/\*;  upstream backend {  server 127.0.0.1:8000;  server 127.0.0.1:8001;  server 127.0.0.1:8002;  }  server {  listen 80;  server\_name localhost;  location / {  proxy\_pass http://backend;  proxy\_set\_header Host $host;  proxy\_set\_header X-Real-IP $remote\_addr;  proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;  client\_max\_body\_size 2M;  if ($request\_method !~ ^(GET|POST|PUT|HEAD)$) {  return 403;  }  server\_tokens off;  }  }  } |

**Результат:**

* Nginx работает как Reverse Proxy с балансировкой нагрузки.
* HTTPS настроен с Let’s Encrypt.
* Реализованы базовые меры безопасности.
* Включено логирование и защита от атак.

**Подготовка отчета:**

**1. Размещение кода в Git**

1. **Создать репозиторий на GitHub/GitLab/Gitea** (если еще не создан).
2. Добавить код проекта и убедиться, что он структурирован:

git init

git add .

git commit -m "Initial commit"

git branch -M main

git remote add origin <URL репозитория>

git push -u origin main

**2. Подготовка README.md**

1. Создать файл README.md в корне проекта.
2. Добавить в README.md основные разделы:

# Проект: Reverse Proxy на Nginx для Go-приложения

## 📌 Описание

В этом проекте реализована настройка Nginx как Reverse Proxy для Go-приложения.

Включены балансировка нагрузки, безопасность и настройка HTTPS.

## 🛠 Технологии

- Go

- Nginx

- Docker

- Let's Encrypt (SSL)

- Fail2Ban (защита от атак)

## 📂 Структура проекта

├── nginx/ │ ├── default.conf # Основная конфигурация Nginx ├── app/ │ ├── main.go # Простейший сервер Go ├── README.md ├── docker-compose.yaml ├── .gitignore

**🚀 Запуск проекта**

* 1. Установить Docker и Docker Compose.
  2. Склонировать репозиторий:

git clone <URL репозитория>

cd project-name

* 1. Запустить контейнеры:

docker-compose up -d

* 1. Проверить работу по адресу: http://localhost.

**🔐 Безопасность**

* 1. Ограничено число запросов (DDoS-защита)
  2. Настроены ограничения по IP-адресам
  3. Включен HTTPS через Let's Encrypt

**🧪 Тестирование**

* 1. Проверить доступность сервиса:

curl -I http://localhost

* 1. Проверить балансировку нагрузки:

for i in {1..10}; do curl -s http://localhost | grep "Server ID"; done

* 1. Проверить HTTPS:

curl -I https://example.com

**📜 Логи**

* 1. **Nginx access log:** /var/log/nginx/access.log
  2. **Nginx error log:** /var/log/nginx/error.log

**🏁 Выводы**

В ходе работы были изучены и применены:

* 1. Настройка Reverse Proxy на Nginx
  2. Балансировка нагрузки между экземплярами Go-приложения
  3. Настройка HTTPS с автоматическим обновлением сертификатов
  4. Реализация базовых мер безопасности

**3. Добавление отчетных файлов в репозиторий**

1. **Загрузить файл отчета** (report.pdf или report.md).
2. **Добавить ER-диаграмму, схему архитектуры проекта** (если были созданы).

git add README.md report.pdf diagram.png

git commit -m "Добавлен отчет и документация"

git push

**4. Итоговая проверка**

1. Проверить, что проект корректно клонируется и запускается.
2. Проверить работоспособность Nginx и балансировки.
3. Убедиться, что вся информация есть в README.md.