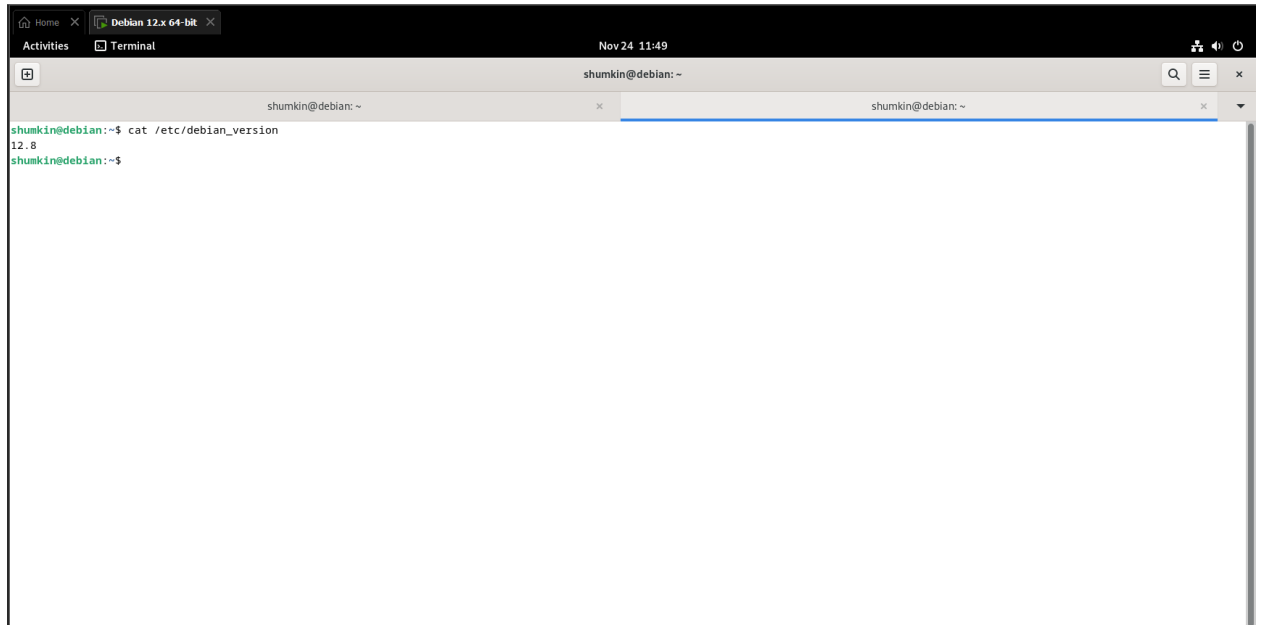


Общая часть дз

Обязательная часть дз

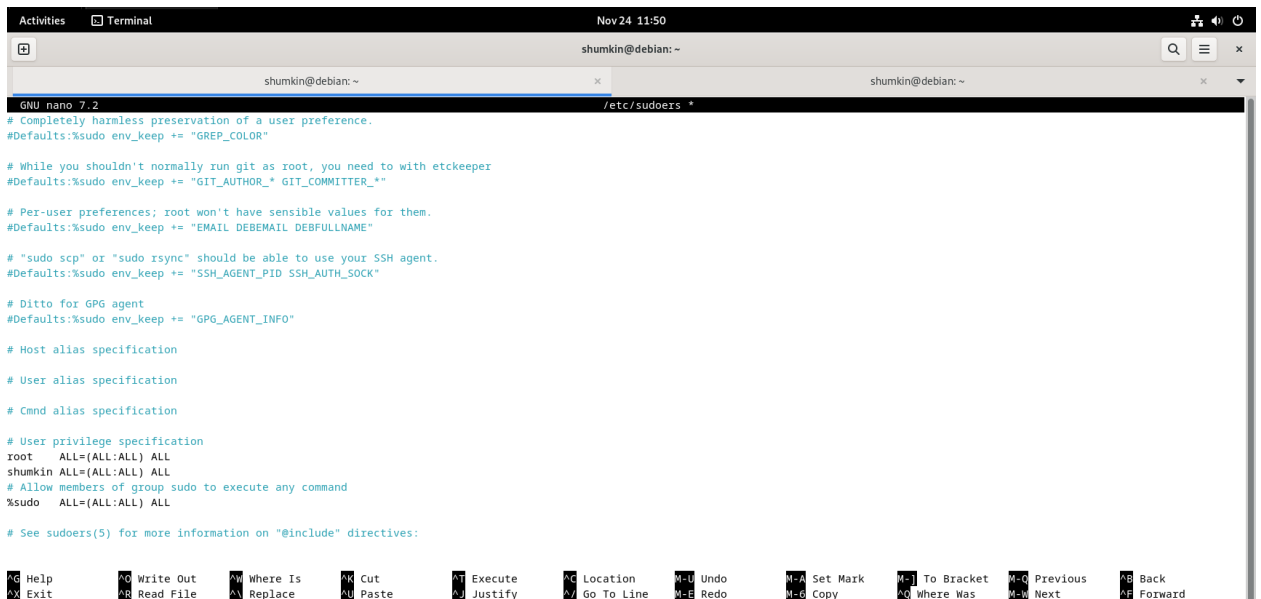
1.1. Подготовка окружения (установка Debian 12, Docker, запуск нужных сервисов с помощью Docker)

Скачиваем Debian 12.8 с официального сайта и устанавливаем на виртуальную машину.



```
shumkin@debian: ~$ cat /etc/debian_version
12.8
shumkin@debian: ~$
```

Переходим в пользователя root и даем непривилегированному пользователю права sudo.



```
GNU nano 7.2 /etc/sudoers
# Completely harmless preservation of a user preference.
#Defaults:sudo env_keep += "GREP_COLOR"

# While you shouldn't normally run git as root, you need to with etckeeper
#Defaults:sudo env_keep += "GIT_AUTHOR_ GIT_COMMITTER_"

# Per-user preferences; root won't have sensible values for them.
#Defaults:sudo env_keep += "EMAIL DEBEMAIL DEBFULLNAME"

# "sudo scp" or "sudo rsync" should be able to use your SSH agent.
#Defaults:sudo env_keep += "SSH_AGENT_PID SSH_AUTH_SOCK"

# Ditto for GPG agent
#Defaults:sudo env_keep += "GPG_AGENT_INFO"

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL
shumkin  ALL=(ALL:ALL) ALL
# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "@include" directives:
```

Обновляем пакеты

```
Activities Terminal Nov 24 11:53
shumkin@debian: ~
shumkin@debian:~$ sudo apt update && sudo apt upgrade
Hit:1 http://deb.debian.org/debian bookworm InRelease
Hit:2 http://deb.debian.org/debian bookworm-updates InRelease
Hit:3 http://security.debian.org/debian-security bookworm-security InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
shumkin@debian:~$
```

Настраиваем репозиторий и устанавливаем Docker

```
Activities Terminal Nov 24 13:26
shumkin@debian: ~
shumkin@debian:~$ sudo apt-get install ca-certificates curl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20230311).
The following NEW packages will be installed:
  curl
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 315 kB of archives.
After this operation, 500 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://deb.debian.org/debian bookworm/main amd64 curl amd64 7.88.1-10+deb12u8 [315 kB]
Fetched 315 kB in 2s (150 kB/s)
Selecting previously unselected package curl.
(Reading database ... 151395 files and directories currently installed.)
Preparing to unpack .../curl_7.88.1-10+deb12u8_amd64.deb ...
Unpacking curl (7.88.1-10+deb12u8) ...
Setting up curl (7.88.1-10+deb12u8) ...
Processing triggers for man-db (2.11.2-2) ...
shumkin@debian:~$ sudo install -m 0755 -d /etc/apt/keyrings
shumkin@debian:~$ sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.asc
shumkin@debian:~$ sudo chmod a+r /etc/apt/keyrings/docker.asc
shumkin@debian:~$ echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/debian \
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
shumkin@debian:~$ sudo apt-get update
Hit:1 http://security.debian.org/debian-security bookworm-security InRelease
Hit:2 http://deb.debian.org/debian bookworm InRelease
Hit:3 http://deb.debian.org/debian bookworm-updates InRelease
Get:4 https://download.docker.com/linux/debian bookworm InRelease [43.3 kB]
Get:5 https://download.docker.com/linux/debian bookworm/stable amd64 Packages [31.6 kB]
Fetched 74.9 kB in 8s (9,137 B/s)
Reading package lists... Done
shumkin@debian:~$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
```

```
Activities Terminal Nov 24 13:27
shumkin@debian: ~
update-alternatives: using /usr/sbin/ip6tables-nft to provide /usr/sbin/ip6tables (ip6tables) in auto mode
update-alternatives: using /usr/sbin/arptables-nft to provide /usr/sbin/arptables (arptables) in auto mode
update-alternatives: using /usr/sbin/ebtables-nft to provide /usr/sbin/ebtables (ebtables) in auto mode
Setting up docker-ce (5:27.3.1-1~debian.12~bookworm) ...
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /lib/systemd/system/docker.service.
Created symlink /etc/systemd/system/sockets.target.wants/docker.socket → /lib/systemd/system/docker.socket.
Setting up git (1:2.39.5-0+deb12u1) ...
Processing triggers for man-db (2.11.2-2) ...
Processing triggers for libc-bin (2.36-9+deb12u9) ...
shumkin@debian:~$ sudo docker version
Client: Docker Engine - Community
Version: 27.3.1
API version: 1.47
Go version: go1.22.7
Git commit: ce12230
Built: Fri Sep 20 11:41:11 2024
OS/Arch: linux/amd64
Context: default

Server: Docker Engine - Community
Engine:
Version: 27.3.1
API version: 1.47 (minimum version 1.24)
Go version: go1.22.7
Git commit: 41ca978
Built: Fri Sep 20 11:41:11 2024
OS/Arch: linux/amd64
Experimental: false
containerd:
Version: 1.7.23
GitCommit: 57f17b0a6295a39009d861b89e3b3b87b005ca27
runc:
Version: 1.1.14
GitCommit: v1.1.14-0-g2c9f560
docker-init:
Version: 0.19.0
GitCommit: de40ad0
shumkin@debian:~$
```

Создаем docker-compose.yml. В нем описываем nginx, php, postgres. У nginx внешний порт 80 перенаправляется на внутренний порт 80 контейнера. У postgres внешний порт 5432 перенаправляется на внутренний порт 5432 контейнера. Задаются переменные окружения. POSTGRES_USER: имя пользователя базы данных. POSTGRES_PASSWORD: пароль для указанного пользователя. POSTGRES_DB: имя базы данных, которая создаётся автоматически при запуске (в данном случае, main).

```
Activities Terminal Nov 24 13:52
shumkin@debian: ~/app
GNU nano 7.2 docker-compose.yml *
services:
  nginx:
    image: nginx:1.22.1
    container_name: nginx
    ports:
      - "80:80"

  php:
    image: php:8.2.24-fpm
    container_name: php

  postgres:
    image: postgres:17.1
    container_name: postgres
    environment:
      POSTGRES_USER: shumkin
      POSTGRES_PASSWORD: 123
      POSTGRES_DB: main
    ports:
      - "5432:5432"
```

Запускаем docker, и проверяем, что все работает.

The top screenshot shows a terminal window with the following commands and output:

```
shumkin@debian:~/app$ sudo docker compose up -d
[sudo] password for shumkin:
[+] Running 4/4
 ✓ Network app_default Created
 ✓ Container postgres Started
 ✓ Container nginx Started
 ✓ Container php Started
shumkin@debian:~/app$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
316cddf3ff08	php:8.2.24-fpm	"docker-php-entrypoi..."	10 seconds ago	Up 10 seconds	9000/tcp	php
8ca7236a383b	postgres:17.1	"docker-entrypoint.s..."	10 seconds ago	Up 10 seconds	0.0.0.0:5432->5432/tcp, :::5432->5432/tcp	postgres
7acdff594e05	nginx:1.22.1	"/docker-entrypoint..."	10 seconds ago	Up 10 seconds	0.0.0.0:80->80/tcp, :::80->80/tcp	nginx

The bottom screenshot shows a Firefox ESR browser window displaying the "Welcome to nginx!" page. The page content includes:

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

1.2. Работа с postgresql (Создание нового пользователя, создание таблиц)

Создаем файл `init.sql`, в котором описываем создание пользователя, выдача всех прав на базу данных и таблицы пользователю, создание таблиц и вставку данных в таблицу.

The screenshot shows a terminal window with the GNU nano 7.2 editor open, editing the file `postgres/init.sql`. The content of the file is as follows:

```
CREATE USER connection_user WITH PASSWORD '123';

CREATE TABLE users (
    username VARCHAR(20) PRIMARY KEY,
    password VARCHAR(20) NOT NULL
);

INSERT INTO users (username, password) VALUES
('user1', 'user1_password'),
('user2', 'user2_password');

CREATE TABLE information (
    column1 VARCHAR(100),
    column2 VARCHAR(100),
    column3 VARCHAR(100)
);

INSERT INTO information (column1, column2, column3) VALUES
('data1', 'data2', 'data3'),
('info1', 'info2', 'info3');

GRANT ALL PRIVILEGES ON DATABASE main TO connection_user;
GRANT ALL PRIVILEGES ON TABLE users TO connection_user;
GRANT ALL PRIVILEGES ON TABLE information TO connection_user;
```

Добавляем том для postgres в docker-compose.yml. Эта строка означает, что файл init.sql с хоста будет перенесен в директорию /docker-entrpoint-initdb.d/init.sql в контейнере. Эта директория используется для выполнения SQL-скриптов при первом запуске контейнера.

```
Activities Terminal Nov 24 14:10
shumkin@debian: ~/app
GNU nano 7.2 docker-compose.yml *
services:
  nginx:
    image: nginx:1.22.1
    container_name: nginx
    ports:
      - "80:80"

  php:
    image: php:8.2.24-fpm
    container_name: php

  postgres:
    image: postgres:17.1
    container_name: postgres
    volumes:
      - ./postgres/init.sql:/docker-entrpoint-initdb.d/init.sql
    environment:
      POSTGRES_USER: shumkin
      POSTGRES_PASSWORD: 123
      POSTGRES_DB: main
    ports:
      - "5432:5432"
```

Перезапускаем контейнеры и подключаемся к контейнеру postgres для проверки. Все успешно

```
Activities Terminal Nov 25 05:14
shumkin@debian: ~/app
shumkin@debian: ~/app
shumkin@debian: ~/app$ sudo docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
86e0baa28b5b   postgres:17.1 "docker-entrpoint.s..." 9 seconds ago Up 9 seconds  0.0.0.0:5432->5432/tcp, :::5432->5432/tcp   postgres
0f933ca79017   php:8.2.24-fpm "docker-php-entrpoi..." 9 seconds ago Up 9 seconds  9000/tcp                                   php
152c30e35e6a   nginx:1.22.1   "/docker-entrpoint..." 9 seconds ago Up 9 seconds  0.0.0.0:80->80/tcp, :::80->80/tcp          nginx
shumkin@debian: ~/app$ sudo docker exec -it postgres psql -U connection_user -d main
psql (17.1 (Debian 17.1-1.pgdg120+1))
Type "help" for help.

main=> \l

          List of databases
  Name | Owner | Encoding | Locale Provider | Collate | Ctype | Locale | ICU Rules | Access privileges
-----+-----+-----+-----+-----+-----+-----+-----+-----
main  | shumkin | UTF8 | libc | en_US.utf8 | en_US.utf8 | | | =Tc/shumkin
      |          |      |      |             |             | | | shumkin=CTc/shumkin
      |          |      |      |             |             | | | connection_user=CTc/shumkin
postgres
template0 | shumkin | UTF8 | libc | en_US.utf8 | en_US.utf8 | | | =c/shumkin
      |          |      |      |             |             | | | shumkin=CTc/shumkin
template1 | shumkin | UTF8 | libc | en_US.utf8 | en_US.utf8 | | | =c/shumkin
      |          |      |      |             |             | | | shumkin=CTc/shumkin
(4 rows)

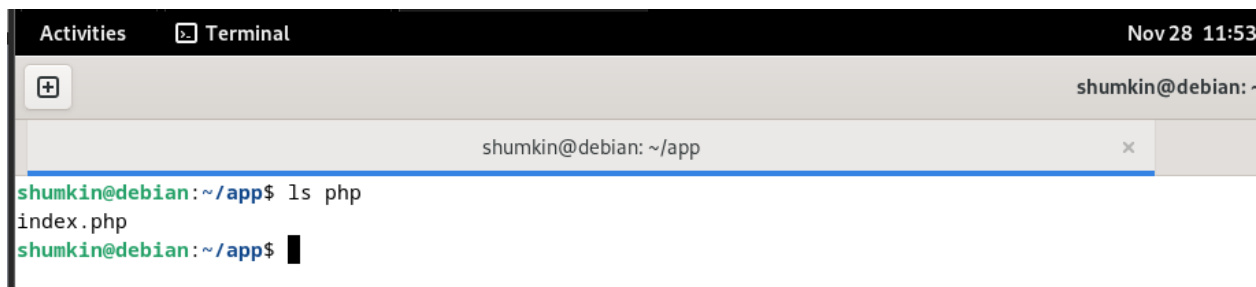
main=> \d

          List of relations
 Schema | Name      | Type  | Owner
-----+-----+-----+-----
public | information | table | shumkin
public | users       | table | shumkin
(2 rows)

main=> select * from users;
username | password
-----
user1    | user1_password
user2    | user2_password
```

1.3. Развертывание веб-приложения

Создаем директорию nginx и php. В директорию php скачиваем файл index.php



```
shumkin@debian: ~/app
shumkin@debian:~/app$ ls php
index.php
shumkin@debian:~/app$
```

Редактируем его для подключения к бд. В host пишем “postgres”, а не ip адрес, так как мы используем контейнеры.



```
GNU nano 7.2 php/index.php
<?php
function checkLogin($pdo, $username, $password) {
    $query = "SELECT * FROM users WHERE username = '$username' AND password = '$password'";
    try {
        $stmt = $pdo->query($query);
        return $stmt->fetch(PDO::FETCH_ASSOC);
    } catch (PDOException $e) {
        die("Ошибка при выполнении запроса: " . htmlspecialchars($e->getMessage()));
    }
}

$host = 'postgres';
$port = '5432';
$dbname = 'main';
$dbuser = 'connection_user';
$dbpassword = '123';
```

В директории nginx создаем файл default.conf. Указываем, что сервер будет слушать 80 порт, указываем имя сервера (localhost означает, что сервер будет доступен по ip адресу 127.0.0.1). Устанавливаем корневую директорию для веб-сервера (/var/www/html). Указываем, что nginx будет искать файл index.php как индексная страница. Если файл не будет найден, то будет ошибка 404. Последний блок конфиг файла конфигурирует обработку запросов, заканчивающихся на .php. Включаем стандартный набор параметров для работы с FastCGI для корректной передачи параметров серверу. Указываем, что запросы к PHP будут передаваться контейнеру с именем php на порт 9000. И указываем путь к файлу PHP, который должен быть выполнен.

```
Activities  Terminal  Nov 28 11:57
shumkin@debian: ~/app
shumkin@debian: ~/app
GNU nano 7.2  nginx/default.conf
server {
    listen 80;

    server_name localhost;

    root /var/www/html;
    index index.php;

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

    location ~ \.php$ {
        include fastcgi_params;
        fastcgi_pass php:9000;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    }
}
```

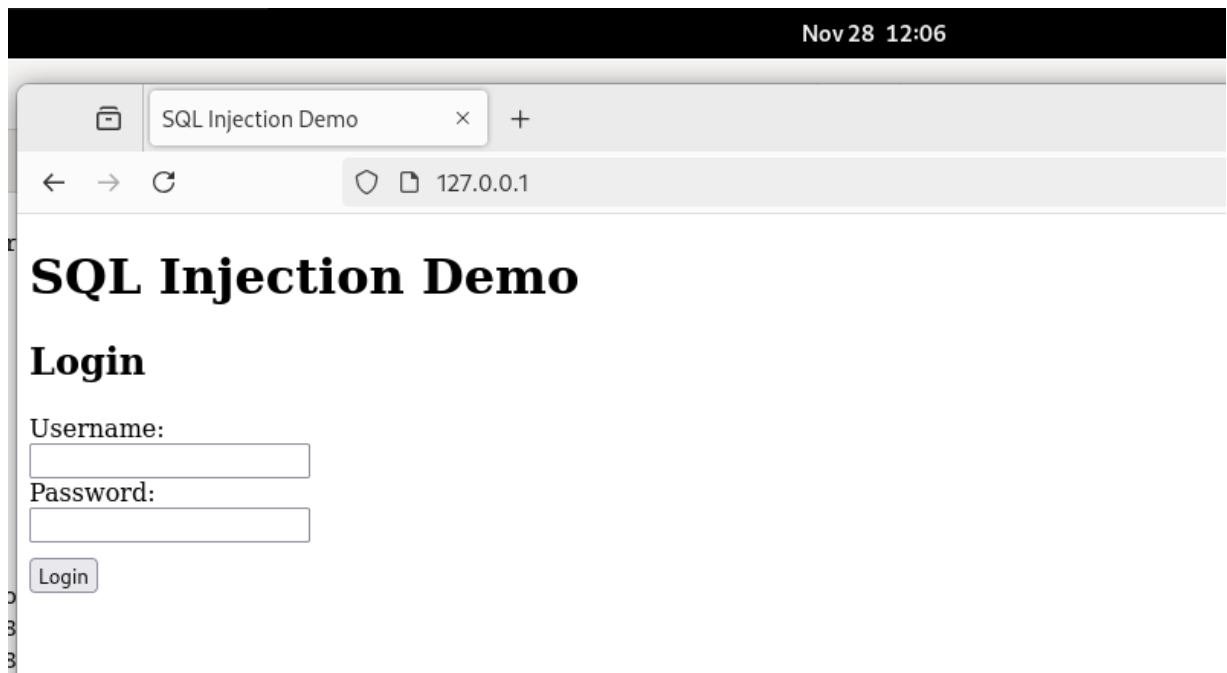
Редактируем docker-compose.yml. Монтируем локальную папку ./php в директорию /var/www/html контейнера. Это место, где nginx будет искать веб-приложение. Монтируем локальный конфигурационный файл nginx default.conf в контейнер. Монтируем локальную директорию ./php в контейнер php. Это директория, в которой PHP будет искать файлы для обработки.

```
Activities  Terminal  Nov 28 12:03
shumkin@debian: ~/app
shumkin@debian: ~/app
GNU nano 7.2  docker-compose.yml
services:
  nginx:
    image: nginx:1.22.1
    container_name: nginx
    volumes:
      - ./php:/var/www/html
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
    ports:
      - "80:80"

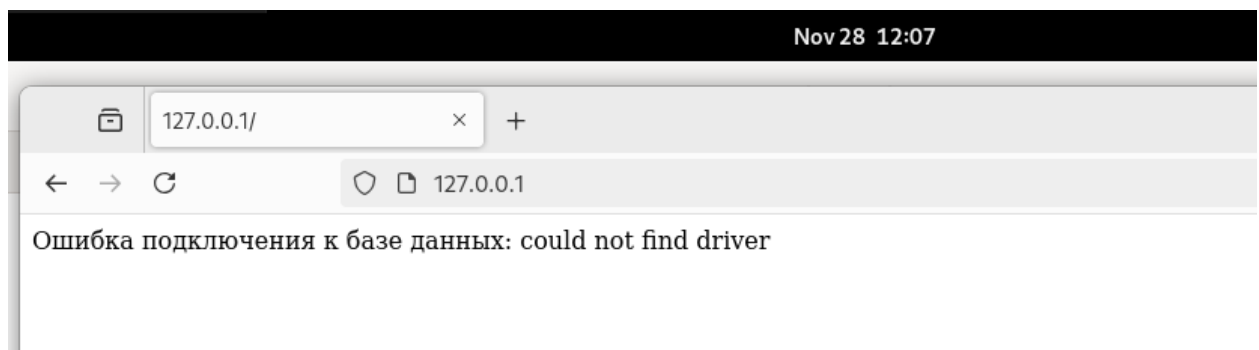
  php:
    image: php:8.2.24-fpm
    container_name: php
    volumes:
      - ./php:/var/www/html

  postgres:
    image: postgres:17.1
    container_name: postgres
    volumes:
      - ./postgres/init.sql:/docker-entrypoint-initdb.d/init.sql
    environment:
      POSTGRES_USER: shumkin
      POSTGRES_PASSWORD: 123
      POSTGRES_DB: main
    ports:
      - "5432:5432"
```

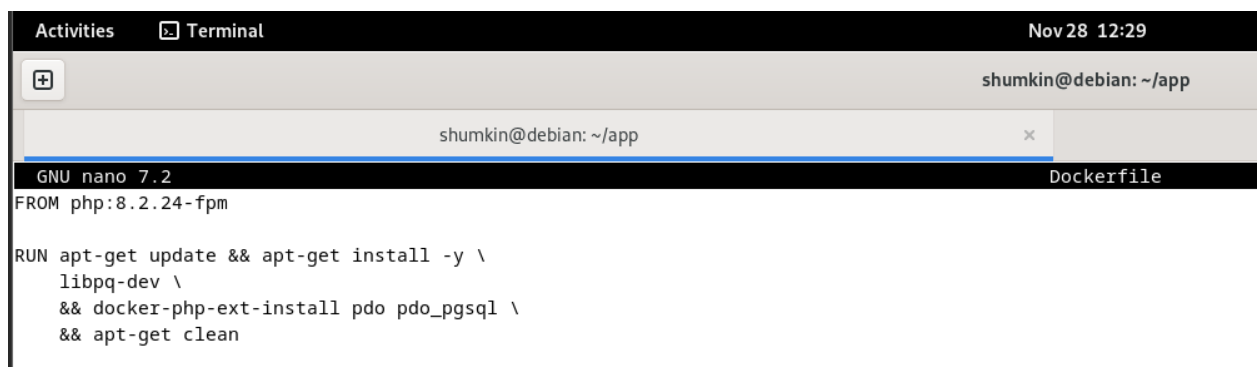
Перезапускаем контейнер, проверяем работу. Видим нашу индексную страницу.



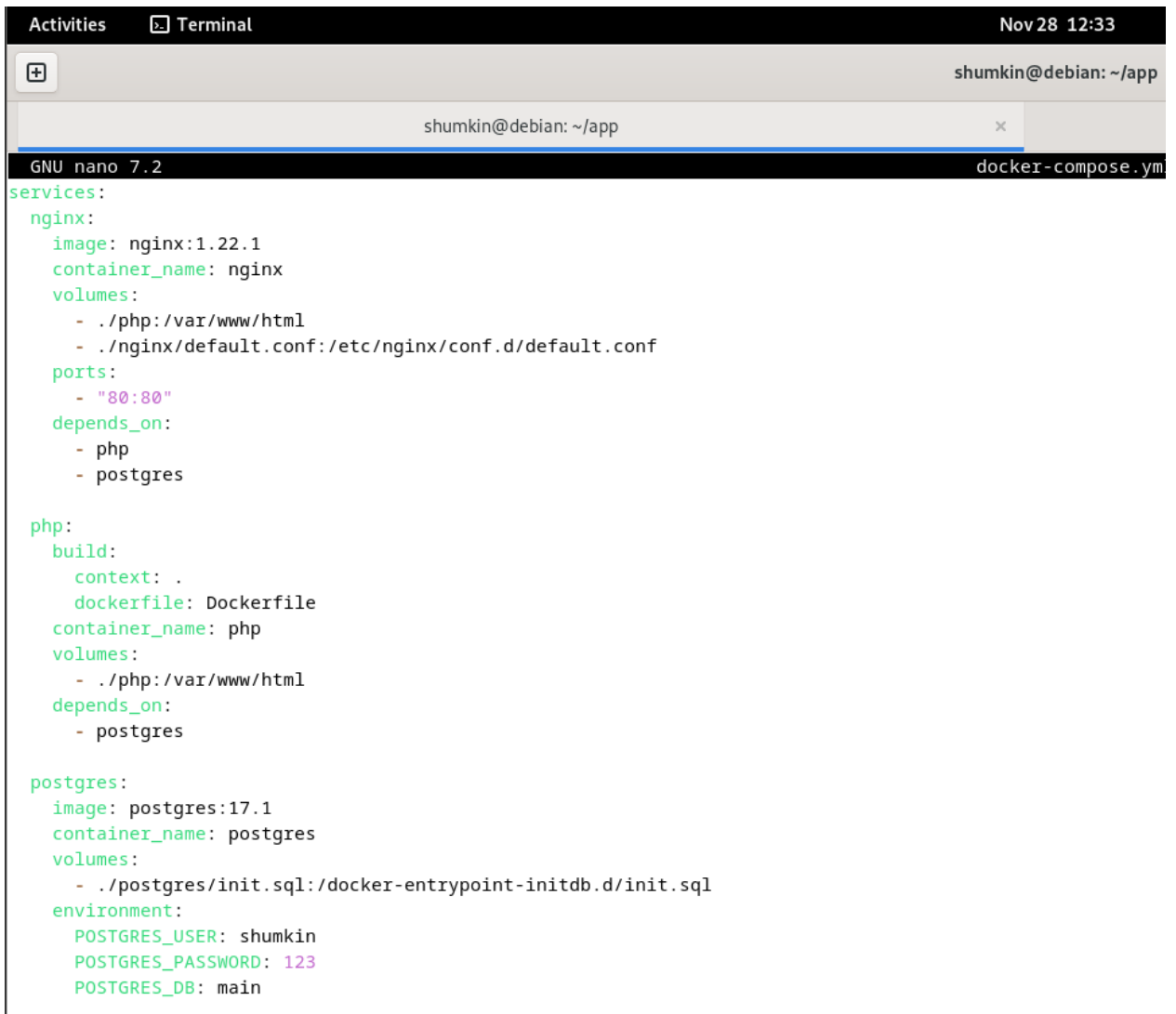
Проверяем работу формы авторизации. Ошибка, не установлен драйвер для работы с PostgreSQL



Создаем файл Dockerfile, в котором опишем образ php. Указываем, какой образ нам нужен, и указываем команды для установки драйвера и необходимые пакеты.



В docker-compose.yml немного меняем блок php. Теперь этот контейнер собирается на основе Dockerfile. Также пишем зависимости одного контейнера от другого. Nginx запускается после запуска postgres и php. Php запускается после postgres.

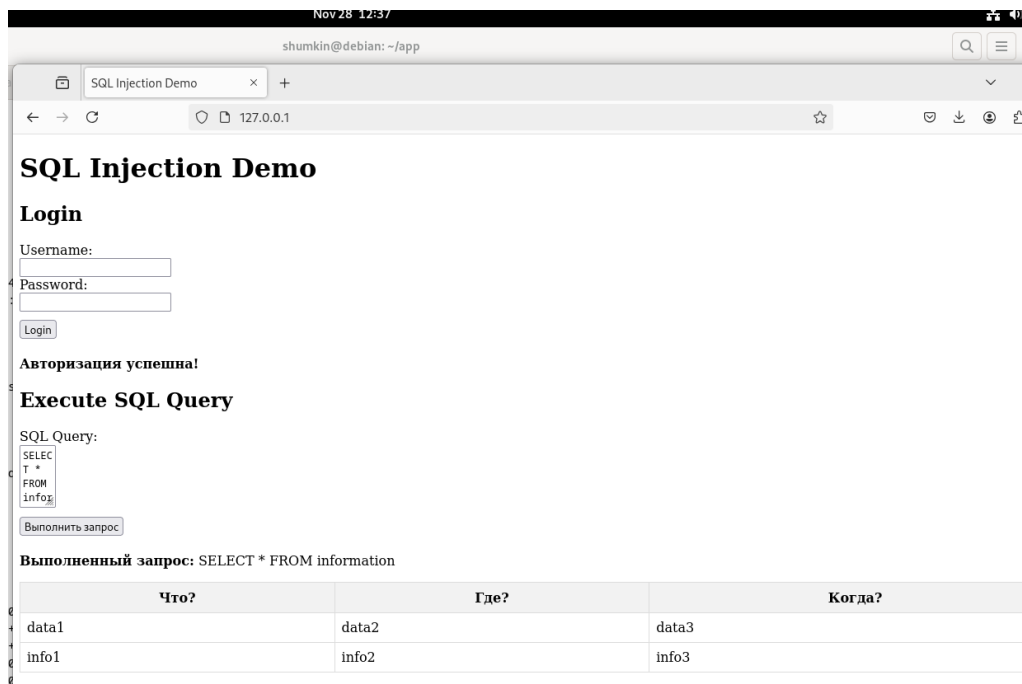


```
Activities  Terminal  Nov 28 12:33
shumkin@debian: ~/app
shumkin@debian: ~/app
GNU nano 7.2  docker-compose.yml
services:
  nginx:
    image: nginx:1.22.1
    container_name: nginx
    volumes:
      - ./php:/var/www/html
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
    ports:
      - "80:80"
    depends_on:
      - php
      - postgres

  php:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: php
    volumes:
      - ./php:/var/www/html
    depends_on:
      - postgres

  postgres:
    image: postgres:17.1
    container_name: postgres
    volumes:
      - ./postgres/init.sql:/docker-entrypoint-initdb.d/init.sql
    environment:
      POSTGRES_USER: shumkin
      POSTGRES_PASSWORD: 123
      POSTGRES_DB: main
```

Перезапускаем контейнер. Теперь все работает.



1.4. Развертывание обратного прокси-сервера (настройка HAProxy)

Создаем директорию `haproxy`, и в ней создаем `haproxy.cfg`. В блоке `defaults` указываем тайм-ауты. Устанавливаем тайм-ауты для установления соединения с `nginx` (5 секунд), для ожидания данных от клиента (50 секунд), для ожидания ответа от `nginx` (50 секунд). Далее указываем, что HAProxy будет слушать входящие подключения на всех сетевых интерфейсах (0.0.0.0) на порту 80, и все запросы будут перенаправлены на backend-сервер (`nginx`). В последнем блоке описываем backend-сервер с именем `nginx`, который работает на порту 80 в контейнере `nginx`, и включаем автоматическую проверку доступности сервера (`check`).

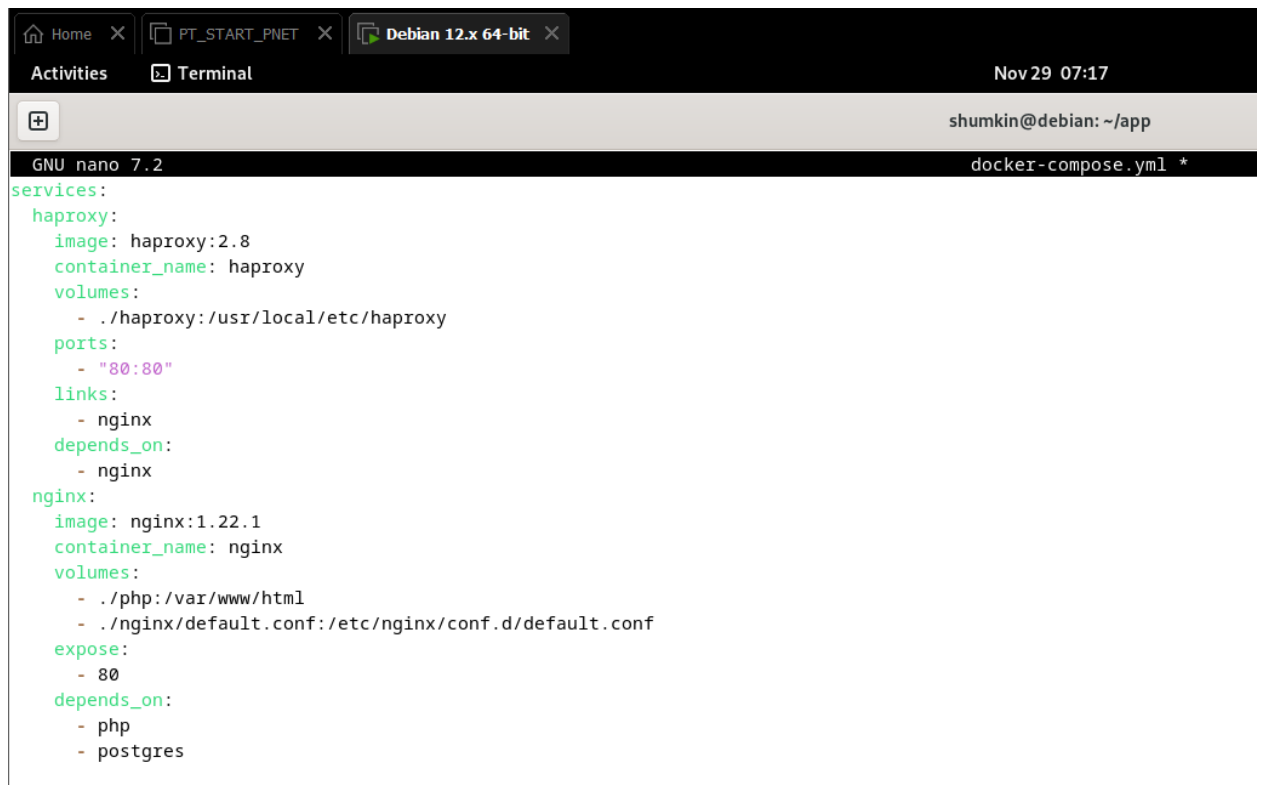
```
Activities Terminal Nov 29 07:07
shumkin@debian: ~/app
haproxy/haproxy.cfg
GNU nano 7.2
defaults
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms

frontend balancer
    bind 0.0.0.0:80
    default_backend web_backends

backend web_backends
    server nginx nginx:80 check
```

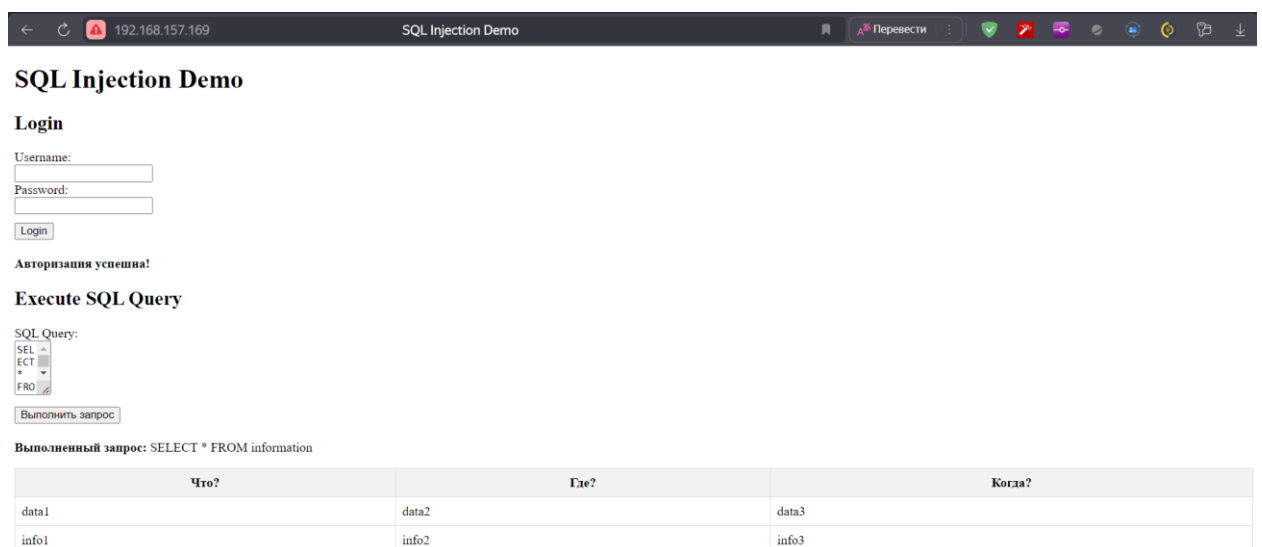
Редактируем `docker-compose.yml`. Добавляем контейнер `haproxy`. Конфиг, которые мы писали, будет перенесен в `/usr/local/etc/haproxy` в контейнере. Он будет работать на 80 порту. Устанавливаем связь с контейнером `nginx`. `Нарпоху` запустится после `nginx`.

У `nginx` заменяем `ports` на `expose`. Так как пробрасывать порт на хост нам не надо.



```
GNU nano 7.2 docker-compose.yml *
services:
  haproxy:
    image: haproxy:2.8
    container_name: haproxy
    volumes:
      - ./haproxy:/usr/local/etc/haproxy
    ports:
      - "80:80"
    links:
      - nginx
    depends_on:
      - nginx
  nginx:
    image: nginx:1.22.1
    container_name: nginx
    volumes:
      - ./php:/var/www/html
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
    expose:
      - 80
    depends_on:
      - php
      - postgres
```

Перезапускаем контейнеры. Все работает.



SQL Injection Demo

Login

Username:

Password:

Авторизация успешна!

Execute SQL Query

SQL Query:

Выполненный запрос: SELECT * FROM information

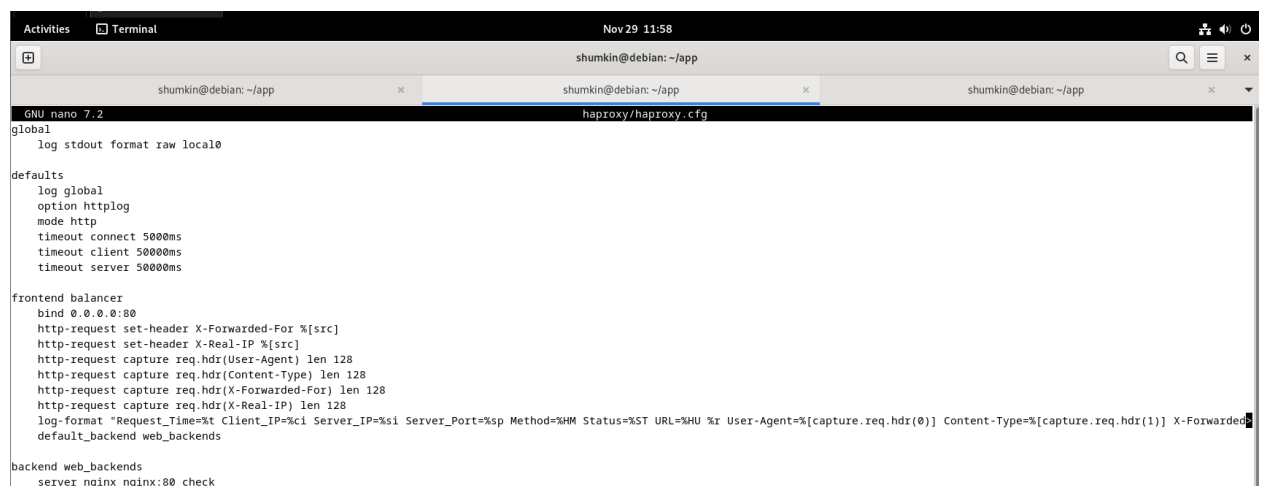
Что?	Где?	Когда?
data1	data2	data3
info1	info2	info3

1.5 Настройка логирования на обратном прокси-сервере (регистрации запроса, IP-адрес клиента, IP-адрес и порт веб-сервера, метод запроса,

код статуса ответа, URL и строка запроса, заголовки запроса и их значения (User-Agent, Content-Type, Cookie, X-Forwarded-For и X-Real-IP), размер запроса и ответа, время обработки запроса и ожидания ответа.)

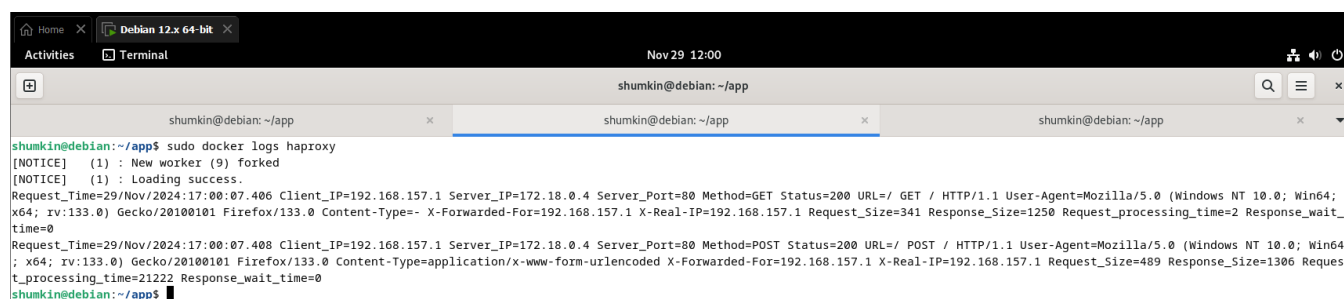
Редактируем haproxy.cfg. Добавляем глобальный раздел, в котором указываем, что логи записываются в стандартный вывод (stdout). В defaults добавляем директиву log global для включения глобального правила логирования во всех последующих разделах. Добавляем заголовки X-Forwarded-For и X-Real-IP. Так как по умолчанию haproxy не добавляет эти заголовки. Перехватываем заголовки User-Agent, Content-Type, X-Forwarded-For и X-Real-IP. Используем log format для вывода кастомизированных логов.

```
log-format "Request_Time=%t Client_IP=%ci Server_IP=%si Server_Port=%sp  
Method=%HM Status=%ST URL=%HU %r User-Agent=%[capture.req.hdr(0)]  
Content-Type=%[capture.req.hdr(1)] X-Forwarded>
```



```
Activities Terminal Nov 29 11:58
shumkin@debian: ~/app
shumkin@debian: ~/app
shumkin@debian: ~/app
shumkin@debian: ~/app
haproxy/haproxy.cfg
global
    log stdout format raw local0
defaults
    log global
    option httplog
    mode http
    timeout connect 5000ms
    timeout client 5000ms
    timeout server 5000ms
frontend balancer
    bind 0.0.0.0:80
    http-request set-header X-Forwarded-For %[src]
    http-request set-header X-Real-IP %[src]
    http-request capture req.hdr(User-Agent) len 128
    http-request capture req.hdr(Content-Type) len 128
    http-request capture req.hdr(X-Forwarded-For) len 128
    http-request capture req.hdr(X-Real-IP) len 128
    log-format "Request_Time=%t Client_IP=%ci Server_IP=%si Server_Port=%sp Method=%HM Status=%ST URL=%HU %r User-Agent=%[capture.req.hdr(0)] Content-Type=%[capture.req.hdr(1)] X-Forwarded-For=%[capture.req.hdr(2)]"
    default_backend web_backends
backend web_backends
    server nginx nginx:80 check
```

Проверяем. Делаем get и post запросы на наш сервер.



```
Home Debian 12.x 64-bit
Activities Terminal Nov 29 12:00
shumkin@debian: ~/app
shumkin@debian: ~/app
shumkin@debian: ~/app
shumkin@debian: ~/app$ sudo docker logs haproxy
[NOTICE] (1) : New worker (9) forked
[NOTICE] (1) : Loading success.
Request_Time=29/Nov/2024:17:00:07.406 Client_IP=192.168.157.1 Server_IP=172.18.0.4 Server_Port=80 Method=GET Status=200 URL=/ GET / HTTP/1.1 User-Agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:133.0) Gecko/20100101 Firefox/133.0 Content-Type=- X-Forwarded-For=192.168.157.1 X-Real-IP=192.168.157.1 Request_Size=341 Response_Size=1250 Request_processing_time=2 Response_wait_time=0
Request_Time=29/Nov/2024:17:00:07.408 Client_IP=192.168.157.1 Server_IP=172.18.0.4 Server_Port=80 Method=POST Status=200 URL=/ POST / HTTP/1.1 User-Agent=Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:133.0) Gecko/20100101 Firefox/133.0 Content-Type=application/x-www-form-urlencoded X-Forwarded-For=192.168.157.1 X-Real-IP=192.168.157.1 Request_Size=489 Response_Size=1306 Request_processing_time=21222 Response_wait_time=0
shumkin@debian: ~/app$
```

Дополнительное задание

1.7. SQLi

В поле логина вводим базовый payload ' or 1=1 -- -

Пароль пишем любой.

SQL Injection Demo

Login

Username:

Password:

Наш запрос прошел, и мы авторизовались.

SQL Injection Demo

Login

Username:

Password:

Авторизация успешна!

Execute SQL Query

SQL Query:

Выполненный запрос: SELECT * FROM information

Что?	Где?	Когда?
data1	data2	data3
info1	info2	info3

1.8 Ограничить доступ к развернутому веб-приложению для клиентов на основе значения заголовка запроса "User-Agent" (доступ должен предоставляться только тем клиентам, у которых значение заголовка равняется "X-Agent". Ограничение доступа настраивается на стороне прокси-сервера.)

В haproxy.cfg добавляем две строки.

acl valid_user_agent hdr(user-agent) -i X-Agent - проверка заголовка User-Agent на значение "X-Agent"

http-request deny if !valid_user_agent - Если заголовок User-Agent не равен "X-Agent", отклоняем запрос

```
Activities  Terminal  Nov 29 12:22
shumkin@debian: ~/app
shumkin@debian: ~/app
shumkin@debian: ~/app
GNU nano 7.2  haproxy/haproxy.cfg
global
    log stdout format raw local0

defaults
    log global
    option httplog
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms

frontend balancer
    bind 0.0.0.0:80
    acl valid_user hdr_sub(user-agent) -i X-Agent
    http-request deny if !valid_user
    http-request set-header X-Forwarded-For %[src]
    http-request set-header X-Real-IP %[src]
    http-request capture req.hdr(User-Agent) len 128
    http-request capture req.hdr(Content-Type) len 128
    http-request capture req.hdr(X-Forwarded-For) len 128
    http-request capture req.hdr(X-Real-IP) len 128
    log-format "Request_Time=%t Client_IP=%ci Server_IP=%si Server_Port=%sp Method=%HM Status=%ST URL=%HU %r User-Agent=%[capture
    default_backend web_backends

backend web_backends
    server nginx nginx:80 check
```

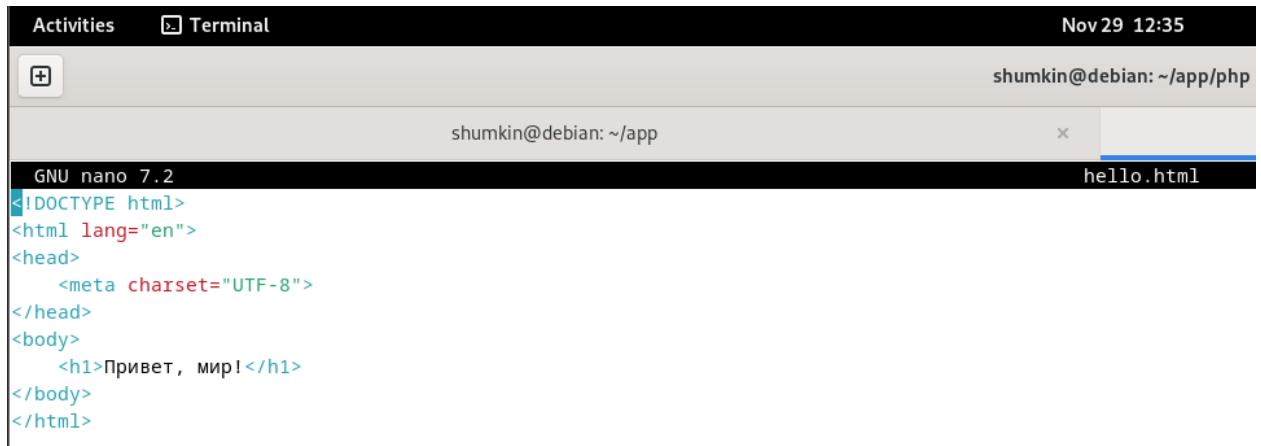
Проверка. Видим, что мы получили доступ к сайту, только когда добавили заголовок user-agent с нужным нам значением.

```
Activities  Terminal  Nov 29 12:23
shumkin@debian: ~/app
shumkin@debian: ~/app
shumkin@debian: ~/app
shumkin@debian:~/app$ curl 127.0.0.1
<html><body><h1>403 Forbidden</h1>
Request forbidden by administrative rules.
</body></html>
shumkin@debian:~/app$ curl -H "User-Agent: X-Agent" 127.0.0.1

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>SQL Injection Demo</title>
  <style>
    table {
      width: 100%;
      border-collapse: collapse;
    }
    th, td {
      border: 1px solid #ddd;
      padding: 8px;
    }
    th {
      background-color: #f2f2f2;
    }
    button {
      margin-top: 10px;
    }
    form {
      margin-bottom: 20px;
    }
  </style>
```

1.9. Создание новой HTML-страницы. Запрет доступа к ней на основе параметров запроса. (Редактирование default.conf у nginx для создания отдельного пути. Запрет доступа к HTML-странице на основе IP-адреса и User-Agent. Настройка ограничения доступа допускается на стороне прокси-сервера)

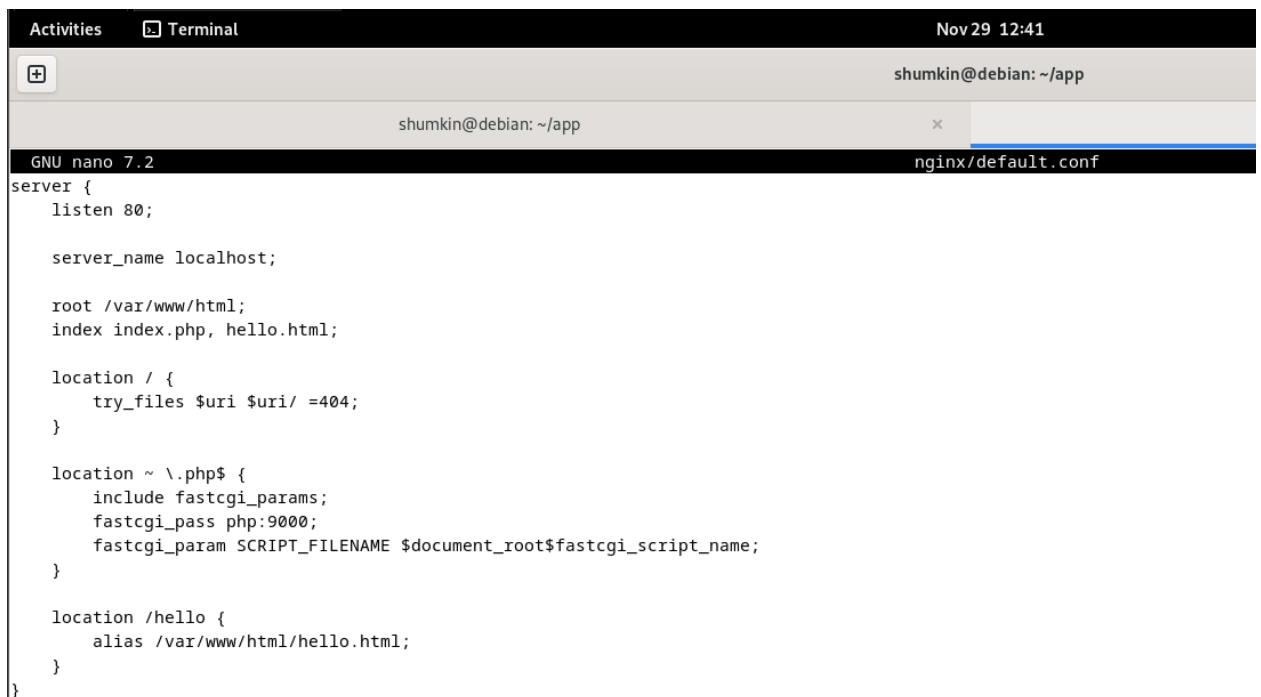
Создаем просто hello.html.



The screenshot shows a terminal window with the title bar 'Activities Terminal' and the date 'Nov 29 12:35'. The user is 'shumkin@debian' and the current directory is '~/app/php'. A new terminal tab is open at '~/app' with the file 'hello.html' selected. The nano 7.2 editor is open, showing the following HTML code:

```
GNU nano 7.2 hello.html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
</head>
<body>
  <h1>Привет, мир!</h1>
</body>
</html>
```

Добавляем новый location в default.conf у nginx. Используем директиву alias, чтобы указать точный путь к файлу для /hello.



The screenshot shows a terminal window with the title bar 'Activities Terminal' and the date 'Nov 29 12:41'. The user is 'shumkin@debian' and the current directory is '~/app'. A new terminal tab is open at '~/app' with the file 'nginx/default.conf' selected. The nano 7.2 editor is open, showing the following nginx configuration:

```
GNU nano 7.2 nginx/default.conf
server {
    listen 80;

    server_name localhost;

    root /var/www/html;
    index index.php, hello.html;

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

    location ~ \.php$ {
        include fastcgi_params;
        fastcgi_pass php:9000;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
    }

    location /hello {
        alias /var/www/html/hello.html;
    }
}
```

Проверяем.



```
Activities Terminal
shumkin@debian: ~/app
shumkin@debian:~/app$ curl -H "User-Agent: X-Agent" http://127.0.0.1/hello
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
</head>
<body>
  <h1>Привет, мир!</h1>
</body>
</html>
shumkin@debian:~/app$
```

Теперь настраиваем haproxy для запрета доступа к данной HTML-странице на основе заголовков запроса X-Real-IP и User-Agent. Добавляем следующие строки:

acl forbidden_ip hdr(X-Real-IP) -i 192.168.1.1 - ACL, которое проверяет, соответствует ли IP-адрес клиента, переданный в заголовке X-Real-IP, значению 192.168.1.1.

acl forbidden_user_agent hdr(User-Agent) -i "Edge" - ACL, которое проверяет, соответствует ли User-Agent клиента значению "Edge".

Запрещаем доступ, если выполняются условия ACL

http-request deny if forbidden_ip

http-request deny if forbidden_user_agent


```
Activities  Terminal  Nov 29 12:50
shumkin@debian: ~/app
shumkin@debian: ~/app
GNU nano 7.2  haproxy/haproxy.cfg
global
    log stdout format raw local0

defaults
    log global
    option httplog
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms

frontend balancer
    bind 0.0.0.0:80
    acl valid_user hdr_sub(user-agent) -i X-Agent
    http-request deny if !valid_user

    acl forbidden_ip hdr(X-Real-IP) -i 192.168.1.1
    acl forbidden_user_agent hdr(User-Agent) -i "Edge"
    http-request deny if forbidden_ip
    http-request deny if forbidden_user_agent

    http-request set-header X-Forwarded-For %[src]
    http-request set-header X-Real-IP %[src]
```

Проверяем. Все работает.

```
shumkin@debian:~/app$ curl -H "User-Agent: Edge" http://127.0.0.1/hello
<html><body><h1>403 Forbidden</h1>
Request forbidden by administrative rules.
</body></html>
shumkin@debian:~/app$
```

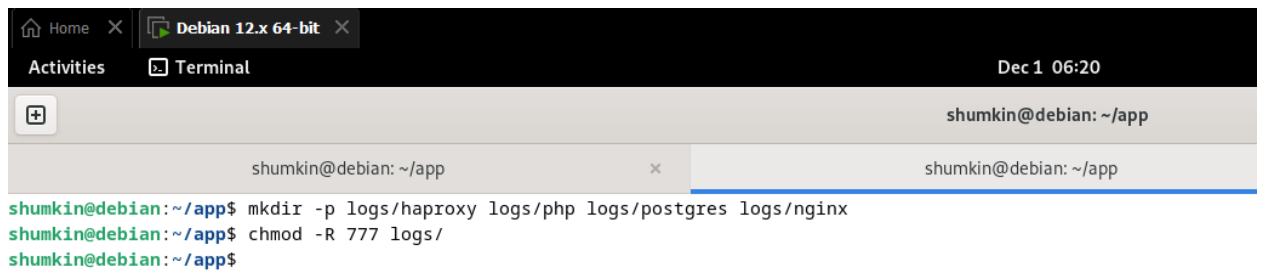
ДЗ для Отдела эксплуатации систем кибербезопасности (EXP)

Обязательная часть дз

3.1 Настройка ротации логов веб-сервера, прокси-сервера, БД, и системных логов. (Настраиваем ротацию логов с контейнеров на хост)

В рабочей директории создаем директорию logs. А в ней директории haproxy, nginx, postgresql, php. В данных директориях будут храниться логи с

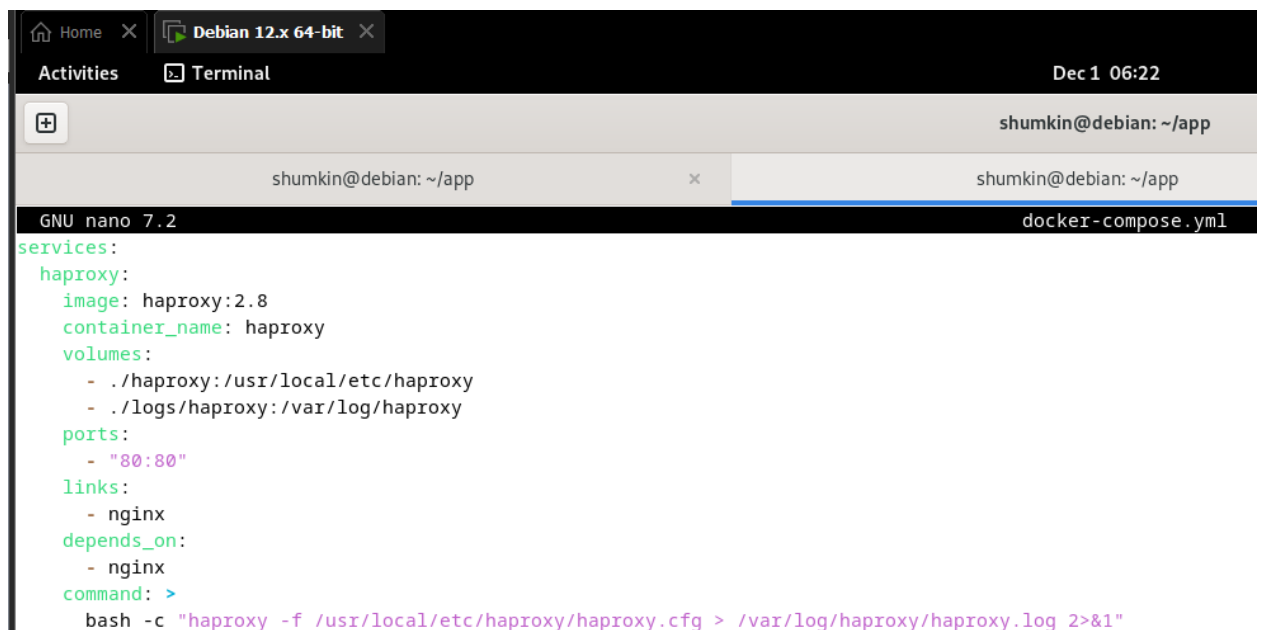
контейнеров. И даем все права этим директориям, чтобы мы могли записывать логи с контейнера в эти директории



```
shumkin@debian: ~/app
shumkin@debian: ~/app
shumkin@debian: ~/app$ mkdir -p logs/haproxy logs/php logs/postgres logs/nginx
shumkin@debian: ~/app$ chmod -R 777 logs/
shumkin@debian: ~/app$
```

В `docker-compose.yml` монтируем директорию `./logs/haproxy` на хосте в `/var/log/haproxy` внутри контейнера. Это позволит нам смотреть логи с хоста.

Добавляем перенаправление стандартного вывода (`stdout`) и стандартной ошибки (`stderr`) в файл лога `/var/log/haproxy/haproxy.log`. Это значит, что все логи `haproxy` будут записываться в этот файл, который будет доступен на хосте через монтированный том.



```
GNU nano 7.2 docker-compose.yml
services:
  haproxy:
    image: haproxy:2.8
    container_name: haproxy
    volumes:
      - ./haproxy:/usr/local/etc/haproxy
      - ./logs/haproxy:/var/log/haproxy
    ports:
      - "80:80"
    links:
      - nginx
    depends_on:
      - nginx
    command: >
      bash -c "haproxy -f /usr/local/etc/haproxy/haproxy.cfg > /var/log/haproxy/haproxy.log 2>&1"
```

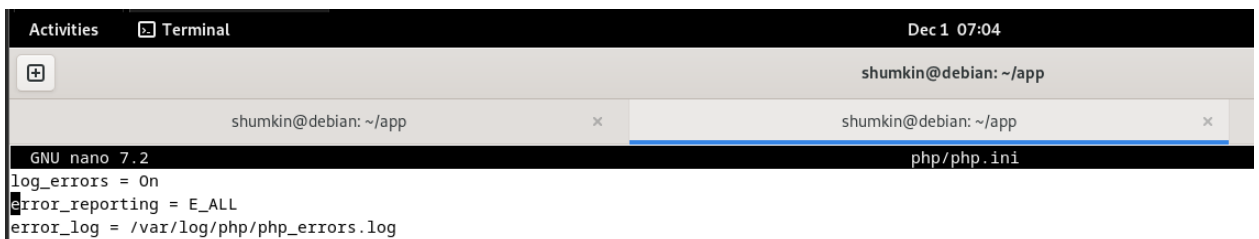
В контейнер `nginx` монтируем директорию `./logs/nginx` на хосте в `/var/log/nginx` внутри контейнера. По умолчанию логи `nginx` хранятся в `/var/log/nginx`. Логи буду сохраняться в `./logs/nginx` на хосте.

```

nginx:
  image: nginx:1.22.1
  container_name: nginx
  volumes:
    - ./php:/var/www/html
    - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
    - ./logs/nginx:/var/log/nginx
  expose:
    - 80
  depends_on:
    - php
    - postgres

```

В директории php создаем php.ini, в котором включаем логи. Включаем логирование ошибок. Указываем, какие типы ошибок должны быть зарегистрированы. E_ALL означает, что будут записаны все ошибки, предупреждения и уведомлений. Указываем путь к файлу, в котором будут записываться логи PHP (/var/log/php/php_errors.log)



The screenshot shows a terminal window with the title 'Activities Terminal' and a timestamp 'Dec 1 07:04'. The terminal displays the command prompt 'shumkin@debian: ~/app' and the file 'php/php.ini' being edited with 'GNU nano 7.2'. The content of the file is as follows:

```

log_errors = On
error_reporting = E_ALL
error_log = /var/log/php/php_errors.log

```

Монтируем php.ini в /usr/local/etc/php на контейнере. Монтируем ./logs/php в /var/log/php, чтобы смотреть логи с хоста в директории ./logs/php

```

php:
  build:
    context: ./php
    dockerfile: Dockerfile
  container_name: php
  volumes:
    - ./php:/var/www/html
    - ./logs/php:/var/log/php
    - ./php/php.ini:/usr/local/etc/php/php.ini
  expose:
    - 9000
  depends_on:
    - postgres

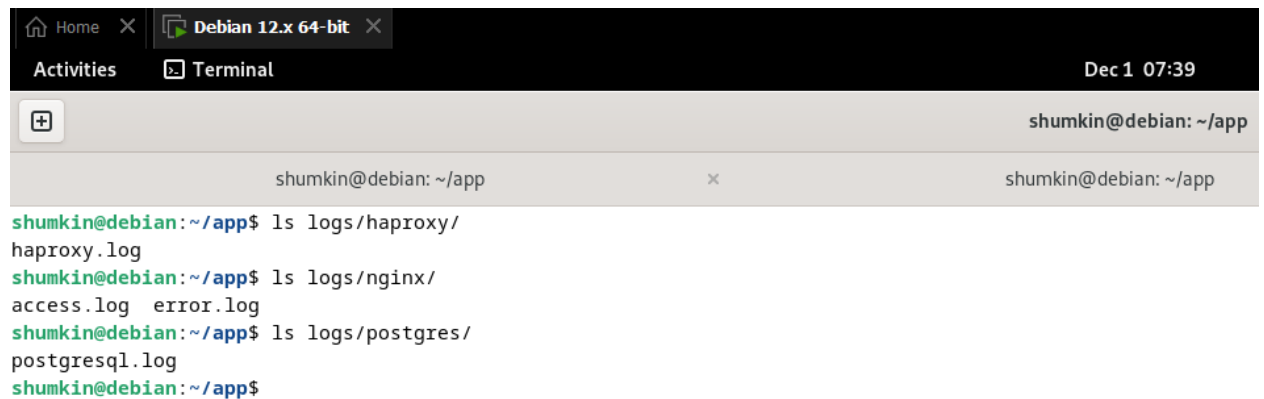
```

Монтируем директорию `./logs/postgres` в папку контейнера `/var/log/postgresql`. Это нужно для хранения логов PostgreSQL на хосте.

Указываем команду, которая будет выполнена при запуске контейнера: включаем сбор логов, задаёт каталог для хранения логов (`/var/log/postgresql`), задаём имя лог-файла, включаем логирование всех SQL-запросов

```
postgres:
  image: postgres:17.1
  container_name: postgres
  volumes:
    - ./postgres/init.sql:/docker-entrypoint-initdb.d/init.sql
    - ./logs/postgres:/var/log/postgresql
  environment:
    POSTGRES_USER: shumkin
    POSTGRES_PASSWORD: 123
    POSTGRES_DB: main
  ports:
    - "5432:5432"
  command: ["postgres", "-c", "logging_collector=on", "-c", "log_directory=/var/log/postgresql", "-c", "log_filename=postgresql.log", "-c", "log_statement=all"]
```

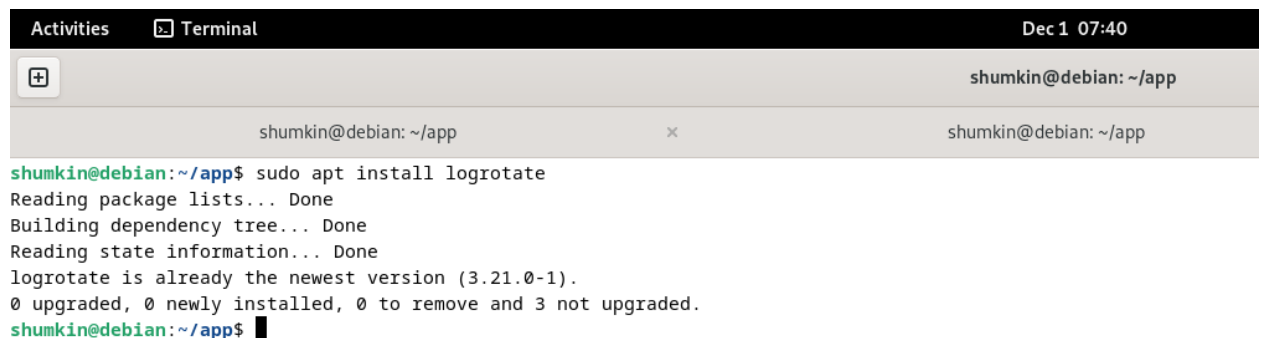
Проверяем, что логи записываются на хост



The screenshot shows a terminal window titled "Debian 12.x 64-bit" with the user "shumkin@debian" in the directory "~/app". The terminal output shows the following commands and results:

```
shumkin@debian:~/app$ ls logs/haproxy/
haproxy.log
shumkin@debian:~/app$ ls logs/nginx/
access.log  error.log
shumkin@debian:~/app$ ls logs/postgres/
postgresql.log
shumkin@debian:~/app$
```

Устанавливаем logrotate для ротации логов



The screenshot shows a terminal window titled "Debian 12.x 64-bit" with the user "shumkin@debian" in the directory "~/app". The terminal output shows the following commands and results:

```
shumkin@debian:~/app$ sudo apt install logrotate
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
logrotate is already the newest version (3.21.0-1).
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
shumkin@debian:~/app$
```

Создаем файл `/etc/logrotate.d/postgresql`. В нем указываем путь к нашим логам.

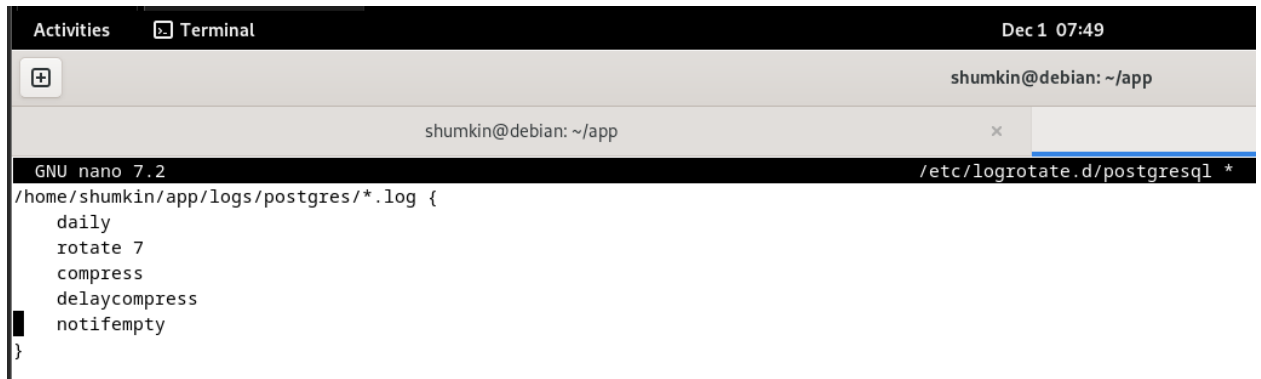
`daily` - указывает, что ротация будет происходить каждый день.

`rotate 7` - указывает, что следует хранить 7 архивированных файлов.

`compress` - указывает, что старые логи будут сжаты.

`delaycompress` – отложить сжатие логов до следующего цикла ротации

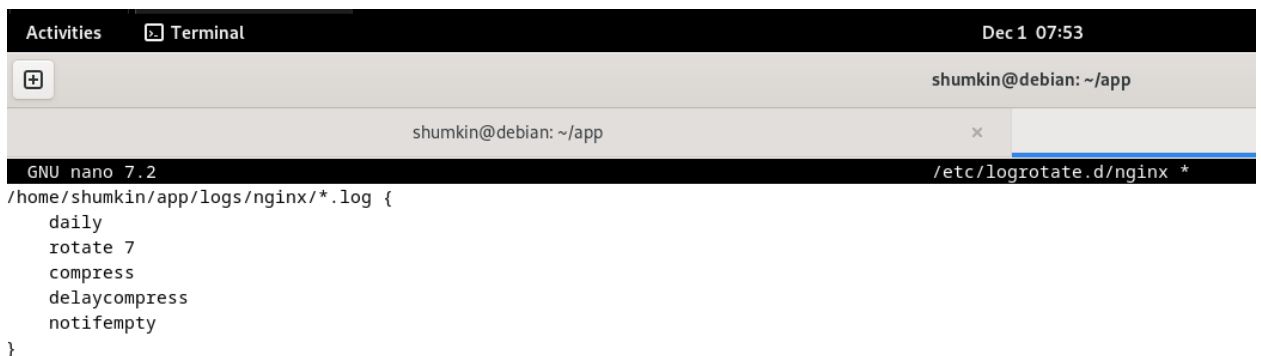
notifempty – указывает, что не нужно ротировать пустой файл лога



A terminal window titled 'Terminal' with a timestamp of 'Dec 1 07:49'. The user 'shumkin@debian' is in the directory '~/app'. A nano editor window is open, editing the file '/etc/logrotate.d/postgresql *'. The content of the file is a logrotate configuration for postgresql logs located at '/home/shumkin/app/logs/postgres/*.log'. The configuration includes: daily rotation, rotate 7 times, compression, delaycompress, and the 'notifempty' directive.

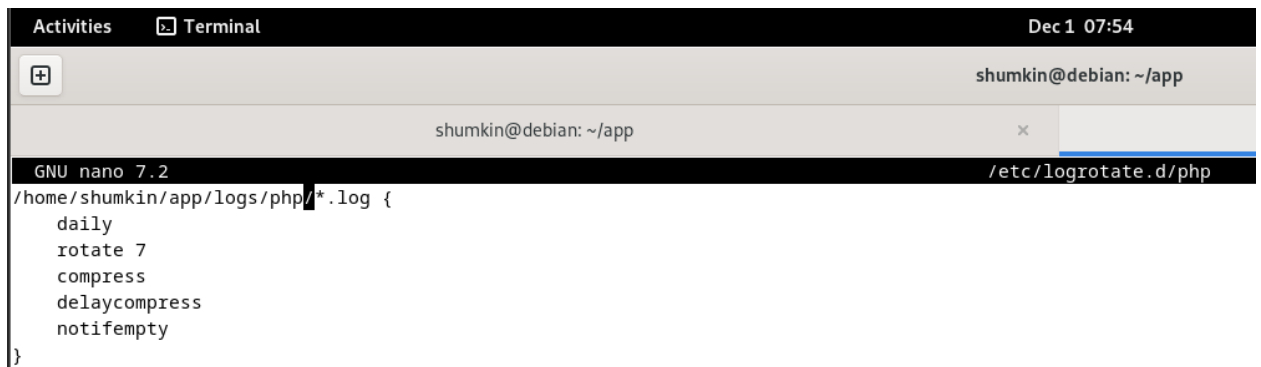
```
GNU nano 7.2 /etc/logrotate.d/postgresql *
/home/shumkin/app/logs/postgres/*.log {
    daily
    rotate 7
    compress
    delaycompress
    notifempty
}
```

То же самое делаем с логами остальных сервисов.



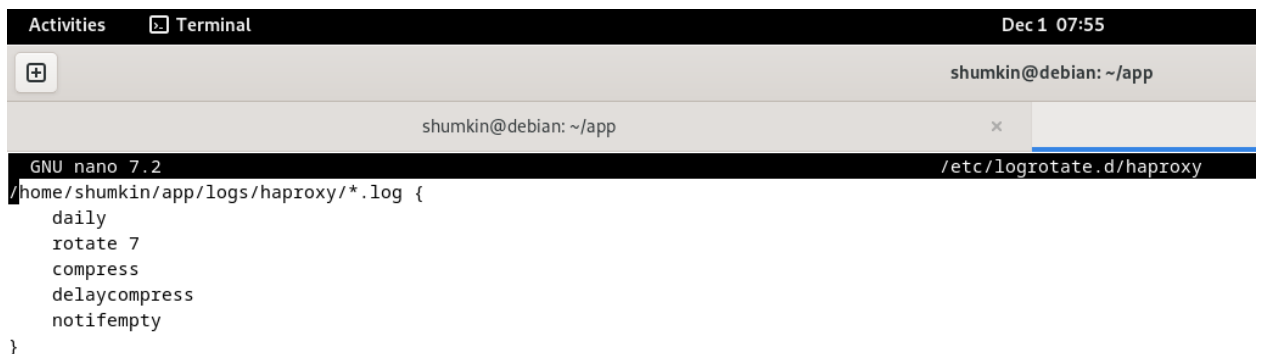
A terminal window titled 'Terminal' with a timestamp of 'Dec 1 07:53'. The user 'shumkin@debian' is in the directory '~/app'. A nano editor window is open, editing the file '/etc/logrotate.d/nginx *'. The content of the file is a logrotate configuration for nginx logs located at '/home/shumkin/app/logs/nginx/*.log'. The configuration includes: daily rotation, rotate 7 times, compression, delaycompress, and the 'notifempty' directive.

```
GNU nano 7.2 /etc/logrotate.d/nginx *
/home/shumkin/app/logs/nginx/*.log {
    daily
    rotate 7
    compress
    delaycompress
    notifempty
}
```



A terminal window titled 'Terminal' with a timestamp of 'Dec 1 07:54'. The user 'shumkin@debian' is in the directory '~/app'. A nano editor window is open, editing the file '/etc/logrotate.d/php'. The content of the file is a logrotate configuration for php logs located at '/home/shumkin/app/logs/php/*.log'. The configuration includes: daily rotation, rotate 7 times, compression, delaycompress, and the 'notifempty' directive.

```
GNU nano 7.2 /etc/logrotate.d/php
/home/shumkin/app/logs/php/*.log {
    daily
    rotate 7
    compress
    delaycompress
    notifempty
}
```

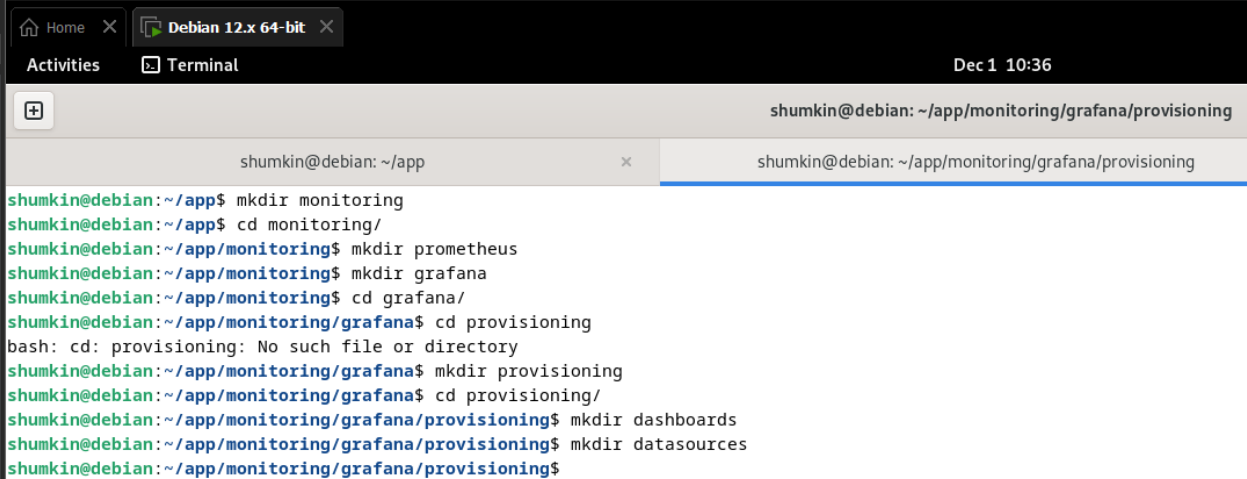


A terminal window titled 'Terminal' with a timestamp of 'Dec 1 07:55'. The user 'shumkin@debian' is in the directory '~/app'. A nano editor window is open, editing the file '/etc/logrotate.d/haproxy'. The content of the file is a logrotate configuration for haproxy logs located at '/home/shumkin/app/logs/haproxy/*.log'. The configuration includes: daily rotation, rotate 7 times, compression, delaycompress, and the 'notifempty' directive.

```
GNU nano 7.2 /etc/logrotate.d/haproxy
/home/shumkin/app/logs/haproxy/*.log {
    daily
    rotate 7
    compress
    delaycompress
    notifempty
}
```

3.2 Мониторинг веб-сервера (Grafana + Prometheus). (Настройка Grafana и Prometheus, описание их в docker-compose.yml, выбор метрик в prometheus)

Создаем директорию monitoring, а в ней директории grafana и prometheus. В Grafana создаем provisioning, а в ней dashboards и datasources. В данных директориях будут храниться файлы конфигураций этих сервисов.



```
shumkin@debian: ~/app$ mkdir monitoring
shumkin@debian: ~/app$ cd monitoring/
shumkin@debian: ~/app/monitoring$ mkdir prometheus
shumkin@debian: ~/app/monitoring$ mkdir grafana
shumkin@debian: ~/app/monitoring$ cd grafana/
shumkin@debian: ~/app/monitoring/grafana$ cd provisioning
bash: cd: provisioning: No such file or directory
shumkin@debian: ~/app/monitoring/grafana$ mkdir provisioning
shumkin@debian: ~/app/monitoring/grafana$ cd provisioning/
shumkin@debian: ~/app/monitoring/grafana/provisioning$ mkdir dashboards
shumkin@debian: ~/app/monitoring/grafana/provisioning$ mkdir datasources
shumkin@debian: ~/app/monitoring/grafana/provisioning$
```

Создаем новые контейнеры Prometheus, Grafana, node-exporter.

Монтируем локальную директорию ./monitoring/prometheus в контейнер в директорию /etc/prometheus. В этой папке будут храниться конфигурационные файлы Prometheus.

--config.file=/etc/prometheus/prometheus.yml - указываем путь к конфигурационному файлу Prometheus, который находится по адресу /etc/prometheus/prometheus.yml.

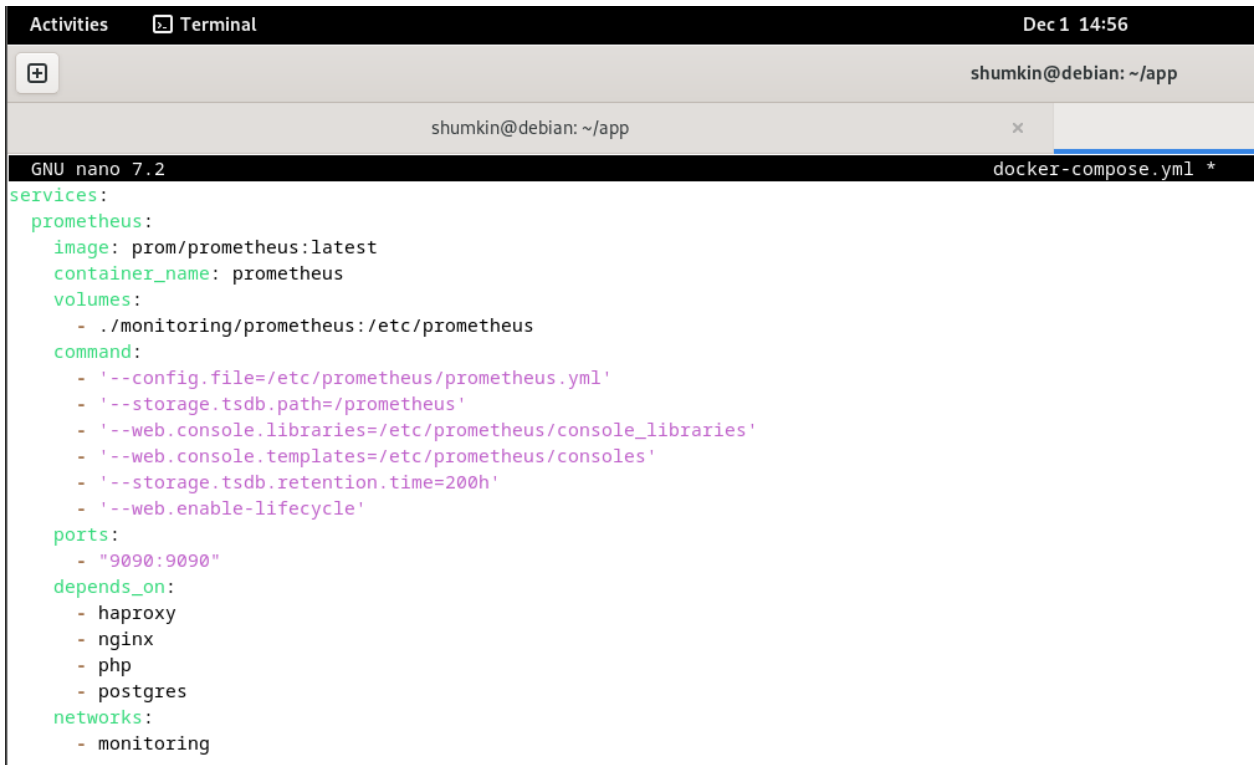
--storage.tsdb.path=/prometheus - определяем путь, где Prometheus будет хранить данные временных рядов внутри контейнера.

--web.console.libraries=/etc/prometheus/console_libraries - указываем путь к библиотекам консоли Prometheus, которые используются для веб-интерфейса.

--web.console.templates=/etc/prometheus/consoles - указываем путь к шаблонам консоли Prometheus, которые используются для визуализации в веб-интерфейсе.

--storage.tsdb.retention.time=200h - устанавливаем время хранения данных в Prometheus — в данном случае, данные будут храниться в течение 200 часов.

Prometheus будет запускаться после того, как сервисы haproxy, nginx, php, postgres будут запущены.



The screenshot shows a terminal window with the title 'Activities Terminal' and the date 'Dec 1 14:56'. The user is 'shumkin@debian' and the current directory is '~/app'. The terminal displays the content of a file named 'docker-compose.yml' using the 'nano' editor. The configuration defines a 'prometheus' service that depends on 'haproxy', 'nginx', 'php', and 'postgres'. It maps the host's monitoring directory to the container's etc directory and sets various configuration options like config file, storage path, and retention time. The service is mapped to port 9090.

```
services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    volumes:
      - ./monitoring/prometheus:/etc/prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
      - '--web.console.libraries=/etc/prometheus/console_libraries'
      - '--web.console.templates=/etc/prometheus/consoles'
      - '--storage.tsdb.retention.time=200h'
      - '--web.enable-lifecycle'
    ports:
      - "9090:9090"
    depends_on:
      - haproxy
      - nginx
      - php
      - postgres
    networks:
      - monitoring
```

У Grafana все просто. Монтируем нашу директорию с хоста в контейнер, где будут храниться конфигурационные файлы

```
grafana:
  image: grafana/grafana:latest
  container_name: grafana
  ports:
    - "3000:3000"
  volumes:
    - ./monitoring/grafana/provisioning:/etc/grafana/provisioning
  depends_on:
    - prometheus
  networks:
    - monitoring
```

Node exporter нужен нам, чтобы получать системные метрики. Монтируем директории /proc, /sys и корневую директорию с хоста в контейнер чтобы собирать информацию о системе хоста.

--path.procfs=/host/proc - указываем путь к каталогу /proc на хосте, который будет использоваться для сбора данных о процессах.

--path.rootfs=/rootfs - указываем путь к корневой файловой системе, монтированной на хосте.

--path.sysfs=/host/sys - указываем путь к каталогу /sys на хосте

--collector.filesystem.ignored-mount-points=^/(sys|proc|dev|host|etc)(\$\$|/) - эта настройка говорит node-exporter, чтобы он игнорировал определенные точки монтирования файловой системы, такие как /sys, /proc, /dev, /host, /etc, которые не являются полезными для мониторинга.

restart: unless-stopped - контейнер будет автоматически перезапущен, если он завершит свою работу.

```
nodeexporter:
  image: prom/node-exporter:latest
  container_name: node-exporter
  volumes:
    - /proc:/host/proc:ro
    - /sys:/host/sys:ro
    - /:/rootfs:ro
  command:
    - '--path.procfs=/host/proc'
    - '--path.rootfs=/rootfs'
    - '--path.sysfs=/host/sys'
    - '--collector.filesystem.ignored-mount-points=^/(sys|proc|dev|host|etc)($$|/)'
  restart: unless-stopped
  ports:
    - "9100:9100"
  networks:
    - monitoring
```

Создаем Prometheus.yml в /monitoring/prometheus. Задаем интервал времени 15 секунд, с которым Prometheus будет опрашивать источники метрик. Далее описываем, откуда Prometheus будет брать метрики (с node-exporter и haproxy)



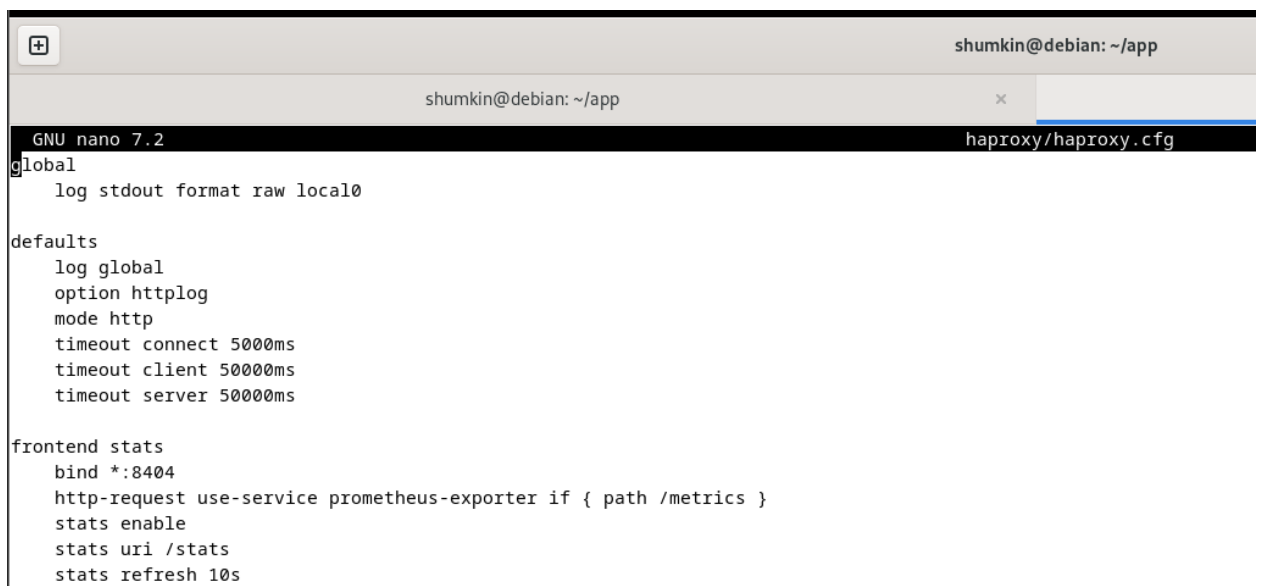
The screenshot shows a terminal window with the title bar 'Activities Terminal' and the date 'Dec 1 15:05'. The terminal prompt is 'shumkin@debian: ~/app'. The nano text editor is open, editing the file 'monitoring/prometheus/prometheus.yml'. The content of the file is as follows:

```
GNU nano 7.2 monitoring/prometheus/prometheus.yml
global:
  scrape_interval: 15s
  evaluation_interval: 15s

scrape_configs:
  - job_name: 'node'
    static_configs:
      - targets: ['node-exporter:9100']
    metrics_path: '/metrics'
    relabel_configs:
      - source_labels: [__param_target]
        target_label: instance
      - target_label: job
        replacement: 'node'

  - job_name: 'haproxy'
    static_configs:
      - targets: ['haproxy:8404']
```

Редактируем haproxy.cfg. Добавляем блок frontend stats. Это нужно, чтобы haproxy давал нам метрики без установки дополнительных экспортеров Prometheus. Порт 8404 будет использоваться для доступа к метрикам haproxy. Если запрос приходит с путем /metrics, то haproxy будет использовать сервис prometheus-exporter для обработки этого запроса. Автообновление веб-интерфейса метрик будет происходить каждые 10 секунд



The screenshot shows a terminal window with the title bar 'shumkin@debian: ~/app'. The terminal prompt is 'shumkin@debian: ~/app'. The nano text editor is open, editing the file 'haproxy/haproxy.cfg'. The content of the file is as follows:

```
GNU nano 7.2 haproxy/haproxy.cfg
global
  log stdout format raw local0

defaults
  log global
  option httplog
  mode http
  timeout connect 5000ms
  timeout client 50000ms
  timeout server 50000ms

frontend stats
  bind *:8404
  http-request use-service prometheus-exporter if { path /metrics }
  stats enable
  stats uri /stats
  stats refresh 10s
```

Теперь настройки Grafana. Создаем файл datasource.yml. В нем указывается информация о подключении Grafana к серверу Prometheus для получения метрик. Источник данных Prometheus будет использоваться по умолчанию в Grafana для создания дашбордов



```
Activities Terminal Dec 1 15:13
shumkin@debian: ~/app
shumkin@debian: ~/app
GNU nano 7.2 monitoring/grafana/provisioning/datasources/datasource.yml
apiVersion: 1
datasources:
- name: Prometheus
  type: prometheus
  access: proxy
  url: http://prometheus:9090
  isDefault: true
  jsonData:
    httpMethod: GET
  secureJsonData: {}
```

Создаем dashboard.yml. В нем определяется, как Grafana будет загружать и управлять дашбордами. Используем провайдер Prometheus.



```
Activities Terminal Dec 1 15:15
shumkin@debian: ~/app
shumkin@debian: ~/app
GNU nano 7.2 monitoring/grafana/provisioning/dashboards/dashboard.yml
apiVersion: 1
providers:
- name: 'Prometheus'
  orgId: 1
  folder: ''
  type: file
  disableDeletion: false
  editable: true
  allowUiUpdates: true
  options:
    path: /etc/grafana/provisioning/dashboards
```

Создаем node_exporter.json. В нем описываем, какие метрики мы будем визуализировать.

CPU Idle - $\text{avg}(\text{rate}(\text{node_cpu_seconds_total}\{\text{mode}=\text{'idle'}\}[5\text{m}]))$ - эта метрика отслеживает процент времени, когда процессор находится в состоянии простоя (не занят выполнением задач). Она помогает понять, сколько времени процессор "бездействует". Высокий процент "Idle" может указывать на то, что сервер недогружен, а низкий — на возможную перегрузку процессора.

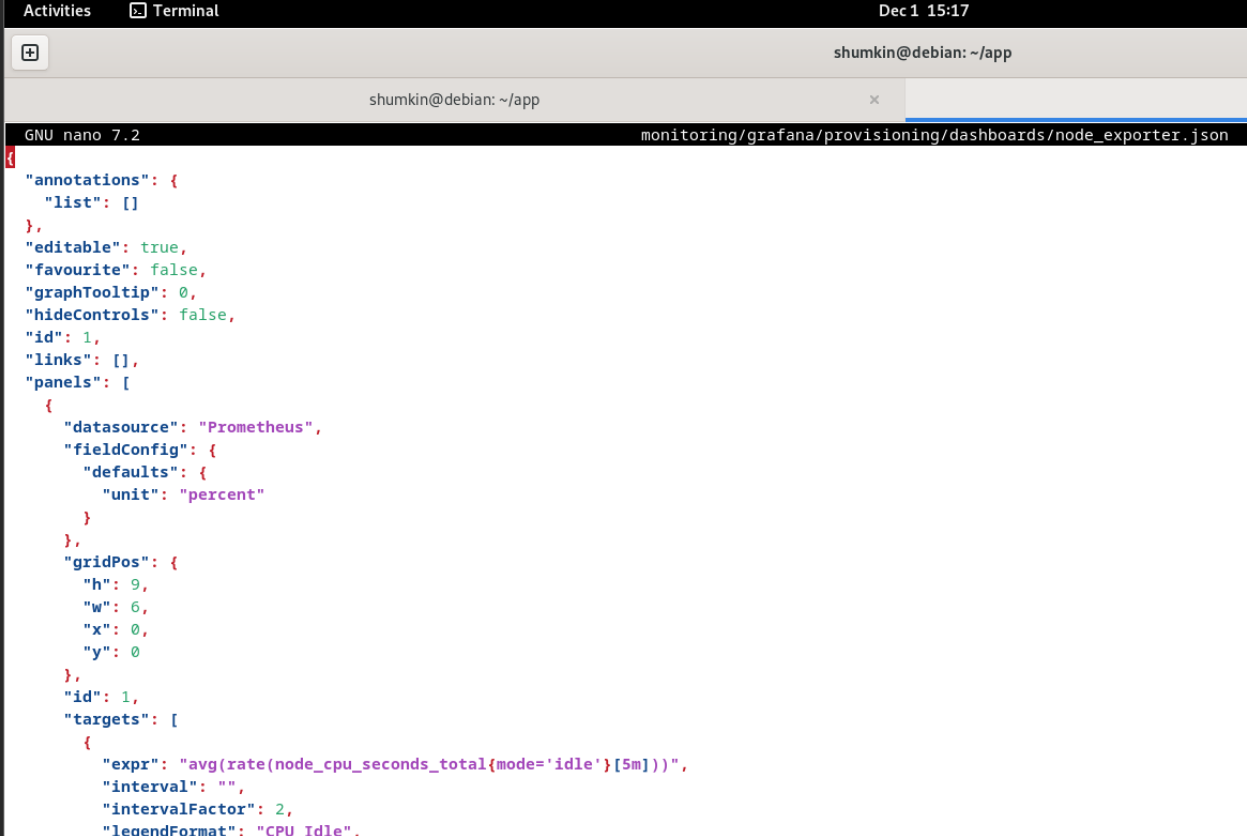
CPU User - $\text{avg}(\text{rate}(\text{node_cpu_seconds_total}\{\text{mode}=\text{'user'}\}[5\text{m}]))$ - Эта метрика отслеживает процент времени, когда процессор занят выполнением пользовательских процессов (не системных). Помогает анализировать, насколько активно используется процессор для выполнения приложений и задач пользователя. Высокие значения могут сигнализировать о большой нагрузке на сервер

Memory Used - $\text{avg}(((\text{node_memory_MemTotal_bytes} - \text{node_memory_MemFree_bytes}) / \text{node_memory_MemTotal_bytes}) * 100)$ - эта метрика показывает процент использования оперативной памяти. Он рассчитывается как отношение используемой памяти к общей доступной памяти. Полезен для мониторинга состояния памяти на сервере.

Disk Free - $\text{avg}(\text{node_filesystem_free_bytes}\{\text{fstype}=\sim\text{'ext4|xfs'}\} / 1024 / 1024 / 1024) / \text{avg}(\text{node_filesystem_free_bytes}\{\text{fstype}=\sim\text{'ext4|xfs'}\} / \text{node_filesystem_size_bytes}\{\text{fstype}=\sim\text{'ext4|xfs'}\} * 100)$ - метрика отображает количество свободного места на диске в гигабайтах и процентах. Помогает следить за свободным местом на жестких дисках сервера. Недостаток места может привести к сбоям в работе системы или приложений.

Network Receive - $\text{avg}(\text{rate}(\text{node_network_receive_bytes_total}[5\text{m}]))$ - Эта метрика отслеживает скорость приема данных по сети. Она измеряет количество байтов, полученных сервером за последние 5 минут. Полезно для мониторинга входящего трафика. Если скорость передачи данных слишком велика или наоборот, слишком мала, это может указывать на проблемы с сетью или с приложениями.

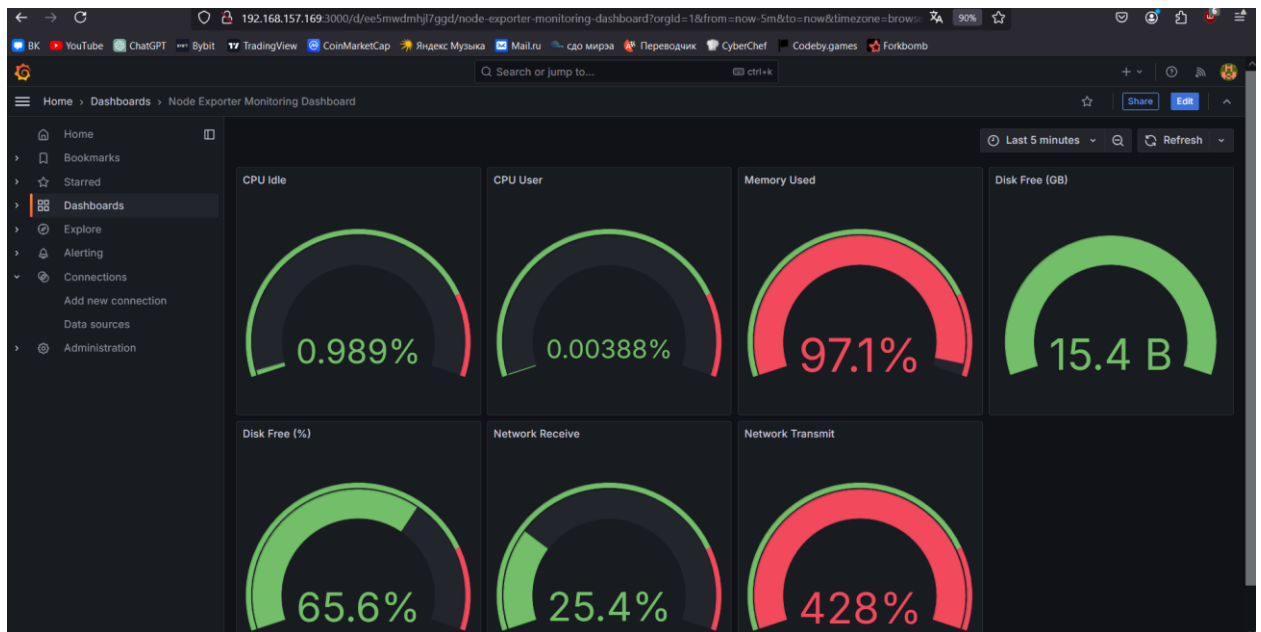
Network Transmit - $\text{avg}(\text{rate}(\text{node_network_transmit_bytes_total}[5\text{m}]))$ - метрика отслеживает скорость передачи данных с сервера. Это важно для мониторинга исходящего трафика. Высокая скорость передачи может свидетельствовать о большом количестве данных, отправляемых сервером, что может указывать на активную работу приложений или проблемы с сетью.



The screenshot shows a terminal window with the title bar "Activities Terminal" and the date/time "Dec 1 15:17". The terminal prompt is "shumkin@debian: ~/app". The user is editing a file named "monitoring/grafana/provisioning/dashboards/node_exporter.json" using the "GNU nano 7.2" editor. The file content is a JSON configuration for a Grafana dashboard panel. The configuration includes a "list" of annotations, a "fieldConfig" with a "unit" of "percent", a "gridPos" of {h: 9, w: 6, x: 0, y: 0}, and a "targets" array with a single target. The target's "expr" is "avg(rate(node_cpu_seconds_total{mode='idle'}[5m]))", the "interval" is "", the "intervalFactor" is 2, and the "legendFormat" is "CPU Idle".

```
GNU nano 7.2 monitoring/grafana/provisioning/dashboards/node_exporter.json
{
  "annotations": {
    "list": []
  },
  "editable": true,
  "favourite": false,
  "graphTooltip": 0,
  "hideControls": false,
  "id": 1,
  "links": [],
  "panels": [
    {
      "datasource": "Prometheus",
      "fieldConfig": {
        "defaults": {
          "unit": "percent"
        }
      },
      "gridPos": {
        "h": 9,
        "w": 6,
        "x": 0,
        "y": 0
      },
      "id": 1,
      "targets": [
        {
          "expr": "avg(rate(node_cpu_seconds_total{mode='idle'}[5m]))",
          "interval": "",
          "intervalFactor": 2,
          "legendFormat": "CPU Idle",

```



Теперь создаем файл `haproxy.json`.

HTTP Responses - `irate(haproxy_backend_http_responses_total)` - Позволяет отслеживать количество ответов сервера с различными HTTP-кодами (2xx, 4xx, 5xx). Это важно для понимания успешности обработки запросов, наличия ошибок и анализа производительности веб-сервера.

Response Errors - `irate(haproxy_backend_response_errors_total)` - Метрика ошибок ответов отображает, сколько запросов не удалось обработать корректно. Помогает диагностировать проблемы на уровне бэкенда.

Bytes In/Out - `irate(haproxy_backend_bytes_in_total) / irate(haproxy_backend_bytes_out_total)` - Отражает объем входящего и исходящего трафика, который поступает на серверы. Используется для мониторинга загрузки сети.

```
Activities Terminal Dec 1 15:26
shumkin@debian: ~/app
shumkin@debian: ~/app
GNU nano 7.2 monitoring/grafana/provisioning/dashboards/haproxy.json
{
  "annotations": {
    "list": [
      {
        "builtIn": 1,
        "datasource": "-- Grafana --",
        "enable": true,
        "hide": true,
        "iconColor": "rgb(0, 211, 255, 1)",
        "name": "Annotations & Alerts",
        "type": "dashboard"
      }
    ]
  },
  "description": "HAproxy",
  "editable": true,
  "gnetId": 364,
  "graphTooltip": 1,
  "id": 7,
  "iteration": 1660031327645,
  "links": [],
  "panels": [
    {
      "collapsed": false,
      "datasource": null,
      "gridPos": {
        "h": 1,
        "w": 24
      }
    }
  ]
}
```

