

Практическая работа №5

1.1. Добавление нового сервиса базы данных в Docker Compose

Изначально необходимо создать docker-compose.yml файл, который будет разворачивать наше приложение вместе с базой данных.

```
version: '3.7'

services:
  web:
    build: .
    ports:
      - "5000:5000"
    depends_on:
      - db

  db:
    image: postgres:13
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: secret
      POSTGRES_DB: flask_app
    ports:
      - "5432:5432"
```

1.2. Подключение Flask к базе данных через SQLAlchemy

Для подключения базы данных укажем адресс базы данных, он будет получаться из .env файла. А также создадим модель данных для пользователя с 3 полями.

```
import os
from dotenv import load_dotenv
from flask import Flask, request
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)

app.config['SQLALCHEMY_DATABASE_URI'] =
os.getenv("SQLALCHEMY_DATABASE_URI")
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] =
os.getenv("SQLALCHEMY_TRACK_MODIFICATIONS")

db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)

    def __repr__(self):
        return f'<User {self.username}>'
```

```

def to_dict(self):
    return {
        'id': self.id,
        'username': self.username,
        'email': self.email
    }

```

1.3. Создание таблицы и выполнение операций CRUD через API

Были созданы маршруты приложения для получения, создания, удаления и редактирования пользователей в бд.

```

@app.route('/', methods=['GET'])
def home():
    return "Hello, World!"

@app.route('/users', methods=['GET'])
def list_users():
    users = User.query.all()
    users_list = [user.to_dict() for user in users]
    return jsonify(users_list)

@app.route('/user/<int:user_id>', methods=['GET'])
def get_user_by_id(user_id):
    user = User.query.get(user_id)

    if not user:
        return {"error": "User not found"}, 404

    return jsonify(user.to_dict())

@app.route('/users', methods=['POST'])
def add_user():
    data = request.get_json()
    username = data.get('username')
    email = data.get('email')

    if is_user_exist(email):
        return {"error": "User already exists"}, 409

    try:
        new_user = User(username=username, email=email)
        db.session.add(new_user)
        db.session.commit()
    except Exception as err:
        return {"error": str(err)}, 500

    return jsonify({'message': 'User created successfully', 'user_id':
new_user.id}), 201

@app.route('/users/<int:user_id>', methods=['PUT'])
def modify_user(user_id):
    user = User.query.get(user_id)
    data = request.get_json()

    if not user:
        return {"error": "User not found"}, 404

```

```

try:
    if 'username' in data and data['username']:
        user.username = data['username']

    if 'email' in data and data['email']:
        user.email = data['email']

    db.session.commit()
except Exception as err:
    return {"error": str(err)}, 500

return jsonify({'message': 'User updated successfully'})

@app.route('/users/<int:user_id>', methods=['DELETE'])
def remove_user(user_id):
    user = User.query.get(user_id)

    if not user:
        return {"error": "User not found"}, 404

    try:
        db.session.delete(user)
        db.session.commit()
    except Exception as err:
        return {"error": str(err)}, 500

    return jsonify({'message': 'User deleted successfully'})

```

1.4. Проведение миграций базы данных с помощью Flask-Migrate

Были добавлены миграции базы данных с помощью flask-migrate, которые вызываются при инициализации приложения.

```

import os
from dotenv import load_dotenv
from flask import Flask, request
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate

app = Flask(__name__)
app.config.from_object(Config)

...

db = SQLAlchemy(app)
migrate = Migrate(app, db)

```

Проведенные вручную миграции:

```

docker-compose up --build
# flask db upgrade
INFO [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
#

```

▼ migrations

- versions
- alembic.ini
- env.py
- README
- script.py.mako

2. Настроить Flask на использование Redis для кеширования данных

Было добавлено кеширование при помощи redis. В приложение добавлены строки подключения к redis серверу.

```

app.config['CACHE_TYPE'] = 'redis'
app.config['CACHE_REDIS_HOST'] = os.getenv("REDIS_HOST")
app.config['CACHE_REDIS_PORT'] = os.getenv("REDIS_PORT")
app.config['CACHE_REDIS_DB'] = os.getenv("REDIS_DB")
app.config['CACHE_REDIS_URL'] =
f"redis://{app.config['CACHE_REDIS_HOST']}:{app.config['CACHE_REDIS_PORT']}
/{app.config['CACHE_REDIS_DB']}"

```

На эндпоинт с получением пользователя по id было добавлено кеширование ответа с использованием функции, которая создает ключ для redis. А также был добавлен эндпоинт очистки кеша по ключу.

```

def make_user_id_cache_key(user_id):
    return f"user_data::{user_id}"

@app.route('/user/<int:user_id>')
@cache.cached(timeout=300, make_cache_key=make_user_id_cache_key)
def get_user(user_id):
    user = User.query.get(user_id)

    if not user:
        return USER_NOT_FOUNDED, 404

    return user.to_dict(), 200

```

```
@app.get('/clear_cache/<int:user_id>')
def clear_user_cache(user_id):
    cache.delete(f'user_data::{user_id}')
    return {'message': f'Cache for user {user_id} cleared'}
```

Был добавлен redis в docker-compose.yml.

```
redis:
  image: redis:alpine
  ports:
    - "6379:6379"
  healthcheck:
    test: ["CMD", "redis-cli", "ping"]
    interval: 10s
    timeout: 5s
    retries: 5
```

Запуск приложения с базой данных и redis.

```
=> => naming to docker.io/library/cloudservices5-web:latest
=> => unpacking to docker.io/library/cloudservices5-web:latest
=> [web] resolving provenance for metadata file
[+] Running 3/3
  ✓ Container cloudservices5-redis-1 Created
  ✓ Container cloudservices5-db-1 Created
  ✓ Container cloudservices5-web-1 Recreated
Attaching to db-1, redis-1, web-1
redis-1 | 1:C 19 Nov 2024 18:44:32.562 * o000o000o000o Redis is starting o000o000o000o
redis-1 | 1:C 19 Nov 2024 18:44:32.562 * Redis version=7.4.1, bits=64, commit=00000000, modified=0, pid=1, just started
redis-1 | 1:C 19 Nov 2024 18:44:32.562 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server /path/to/redis.conf
redis-1 | 1:M 19 Nov 2024 18:44:32.563 * monotonic clock: POSIX clock_gettime
redis-1 | 1:M 19 Nov 2024 18:44:32.564 * Running mode=standalone, port=6379.
redis-1 | 1:M 19 Nov 2024 18:44:32.564 * Server initialized
redis-1 | 1:M 19 Nov 2024 18:44:32.564 * Loading RDB produced by version 7.4.1
redis-1 | 1:M 19 Nov 2024 18:44:32.564 * RDB age 10 seconds
redis-1 | 1:M 19 Nov 2024 18:44:32.564 * RDB memory usage when created 0.96 Mb
redis-1 | 1:M 19 Nov 2024 18:44:32.564 * Done loading RDB, keys loaded: 0, keys expired: 0.
redis-1 | 1:M 19 Nov 2024 18:44:32.564 * DB loaded from disk: 0.000 seconds
redis-1 | 1:M 19 Nov 2024 18:44:32.564 * Ready to accept connections tcp
db-1 |
db-1 | PostgreSQL Database directory appears to contain a database; Skipping initialization
```

Проверяем, что наш сервер работает.

Container CPU usage ⓘ

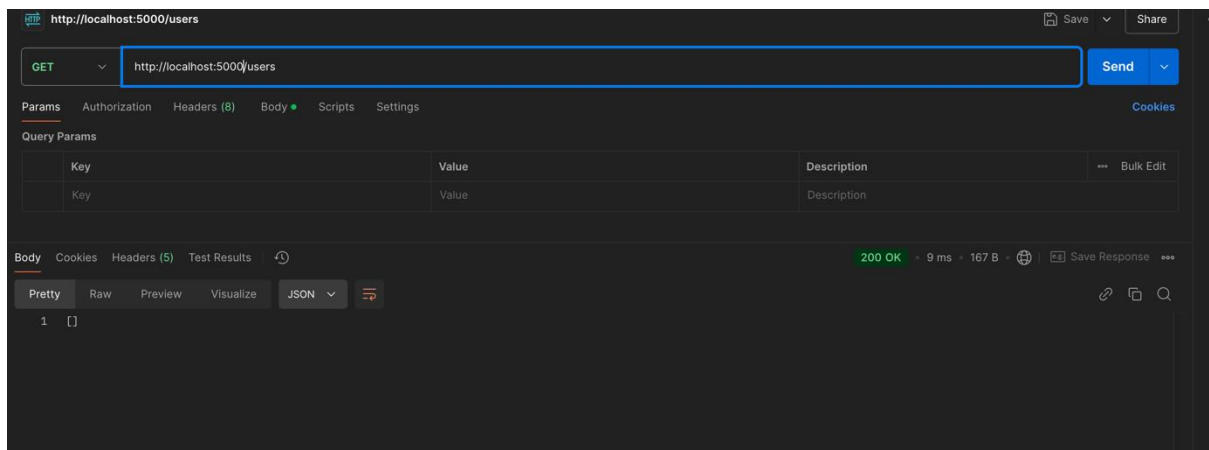
3.33% / 1000% (10 CPUs available)

Container memory usage ⓘ

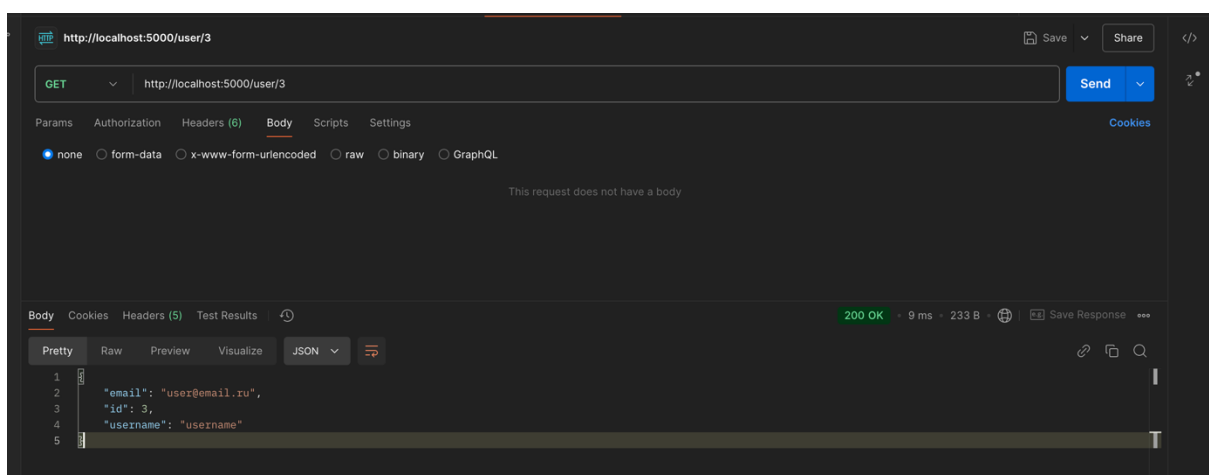
138.08MB / 7.48GB

☐ Only show running containers

| <input type="checkbox"/> | Name | Container ID | Image | Port(s) |
|--------------------------|-----------------------|--------------|---------------------------|-----------------------------|
| <input type="checkbox"/> | wonderful_curie | 1a0cc60cd146 | redis:alpine | 6379:6379 |
| <input type="checkbox"/> | postgres_1 | 674eba1f6355 | postgres:<none> | 5432:5432 |
| <input type="checkbox"/> | <u>cloudservices5</u> | - | - | - |
| <input type="checkbox"/> | redis-1 | b80e41ea0254 | redis:alpine | 6379:6379 ↗ |
| <input type="checkbox"/> | db-1 | 3766146dbcd7 | postgres:13 | 5432:5432 ↗ |
| <input type="checkbox"/> | web-1 | d849696738de | cloudservices5-web:<none> | 8081:8081 ↗ |



Проверим кеширование эндпоинта с получением пользователя по id.



3. Тестирование Flask-приложения

Были созданы тесты для приложения с использованием библиотеки для тестирования pytest. Добавлены тесты валидации статус кодов и ответов, а также тесты проверки кеширования.

```
import pytest
from main import app
from .schemas import UsersResponse
from .utils import create_user, mock_email, mock_username, delete_user,
validate_json, update_user

@pytest.fixture
def client():
    with app.test_client() as client:
        yield client

@pytest.fixture
def test_user(client):
    response = create_user(client, mock_email, mock_username)
    user_id = response.get_json()['user_id']
    yield user_id
    delete_user(client, user_id)
```

```

def test_get_user_by_id(client, test_user):
    response = client.get(f'/user/{test_user}')
    assert response.status_code == 200
    validate_json(UsersResponse, response.get_json())

def test_create_user(client):
    try:
        response = create_user(client, mock_email, mock_username)
        assert response.status_code == 200
        assert response.get_json()['message'] == 'User created successfully'
    finally:
        delete_user(client, response.get_json()['user_id'])

def test_create_existing_user(client, test_user):
    response = create_user(client, mock_email, mock_username)
    assert response.status_code == 404
    assert response.get_json()['message'] == 'User already exists'

def test_delete_user(client):
    user_id = create_user(client, mock_email,
mock_username).get_json()['user_id']
    response = delete_user(client, user_id)
    assert response.status_code == 200
    assert response.get_json()['message'] == 'User deleted successfully'

def test_update_user(client, test_user):
    new_username = "updated_username"
    response = update_user(client, test_user, new_username)
    assert response.status_code == 200
    assert response.get_json()['message'] == 'User updated successfully'

def test_cached_user_data(client, test_user):
    initial_username =
client.get(f'/user/{test_user}').get_json()['username']

    updated_username = "cached_username"
    update_user(client, test_user, updated_username)

    cached_username =
client.get(f'/user/{test_user}').get_json()['username']
    assert cached_username == initial_username

```

Был создан docker-compose.test.yml для запуска тестов в контейнере. Был выбран docker-compose так как было необходимо поднимать базу данных с редисом для запуска приложения.

```

version: '3'
services:

```

```

app:
  command:
    - pytest
  build:
    context: .
    dockerfile: Dockerfile
  ports:
    - "5000:5000"
  depends_on:
    - db
    - redis:

db:
  image: postgres:13
  environment:
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: secret
    POSTGRES_DB: flask_app
  ports:
    - "5432:5432"

redis:
  image: redis:alpine
  ports:
    - "6379:6379"

```

Был запущен `docker-compose.test` с тестированием приложения в контейнере.

```

2024-11-19 22:37:54 ===== test session starts =====
2024-11-19 22:37:54 platform linux -- Python 3.12.7, pytest-8.3.3, pluggy-1.5.0
2024-11-19 22:37:54 rootdir: /app
2024-11-19 22:37:54 configfile: pytest.ini
2024-11-19 22:37:54 collected 11 items
2024-11-19 22:37:54
2024-11-19 22:37:54 tests/test_app.py ..... [100%]
2024-11-19 22:37:54 ===== 11 passed in 0.50s =====

```

Тесты пройдены успешно.

4. Добавление nginx в качестве обратного прокси

Был добавлен `nginx`. Изначально добавлен конфигурационный файл `nginx`, который принимает подключения на 80 порт и перенаправляет запросы на 5000 порт, также были добавлено масштабирование (3 экземпляра приложения).

```

events {
    worker_connections 1024;
}

```



```

http {
    upstream flask_app {
        server web:5000;
        server web:5000;
        server web:5000;
    }

    server {
        listen 80;

        location / {
            proxy_pass http://flask_app;
            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;
        }
    }
}

```

Также было необходимо добавить nginx в docker-compose на 80 порт.

```

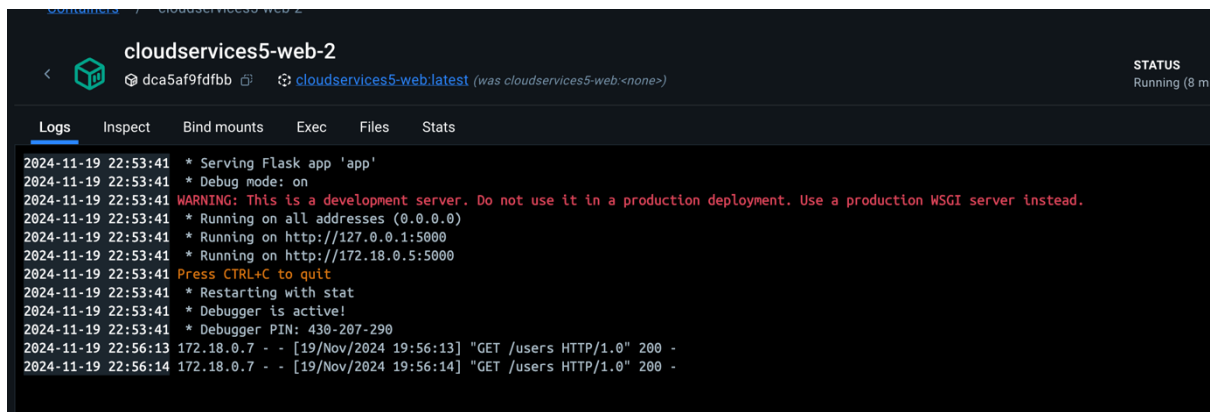
nginx:
  image: nginx:latest
  volumes:
    - ./nginx.conf:/etc/nginx/nginx.conf
  ports:
    - "80:80"
  depends_on:
    - web

```

Было запущено приложение с 3 экземплярами через docker-compose up --scale web=3.

| | | | | | | |
|--------------------------|---|---|----------------|----------------|---------------------------|-------------|
| <input type="checkbox"/> | ▼ | ● | cloudservices5 | - | - | - |
| <input type="checkbox"/> | | ● | redis-1 | 2a3fc28fc784 | redis:alpine | 6379:6379 ↗ |
| <input type="checkbox"/> | | ● | db-1 | ec8280170081 | postgres:13 | 5432:5432 ↗ |
| <input type="checkbox"/> | | ● | web-2 | dca5af9fdffb | cloudservices5-web:<none> | - |
| <input type="checkbox"/> | | ● | web-3 | faaeded251dd ↗ | cloudservices5-web:<none> | - |
| <input type="checkbox"/> | | ● | web-1 | f5ffe5501820 | cloudservices5-web:<none> | - |
| <input type="checkbox"/> | | ● | nginx-1 | 6fc4855674a1 | nginx:latest | 80:80 ↗ |

Было проверено, что запросы распределяются на 3 экземпляра приложения.



The screenshot shows the Docker Desktop interface for a container named 'cloudservices5-web-2'. The container is running on the 'dca5af9fdffb' host and is using the 'cloudservices5-web:latest' image. The status is 'Running (8 m)'. The 'Logs' tab is selected, showing the following output:

```
2024-11-19 22:53:41 * Serving Flask app 'app'
2024-11-19 22:53:41 * Debug mode: on
2024-11-19 22:53:41 WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
2024-11-19 22:53:41 * Running on all addresses (0.0.0.0)
2024-11-19 22:53:41 * Running on http://127.0.0.1:5000
2024-11-19 22:53:41 * Running on http://172.18.0.5:5000
2024-11-19 22:53:41 Press CTRL+C to quit
2024-11-19 22:53:41 * Restarting with stat
2024-11-19 22:53:41 * Debugger is active!
2024-11-19 22:53:41 * Debugger PIN: 430-207-290
2024-11-19 22:56:13 172.18.0.7 - - [19/Nov/2024 19:56:13] "GET /users HTTP/1.0" 200 -
2024-11-19 22:56:14 172.18.0.7 - - [19/Nov/2024 19:56:14] "GET /users HTTP/1.0" 200 -
```

5. Интеграция CI/CD с Docker

Для автоматизации CI/CD процесса был создан конфигурационный файл GitHub Actions, который автоматически собирает и отправляет Docker-образ нашего приложения в Docker Hub.

```
name: Build and Push Docker Image

on:
  push:
    branches:
      - main

jobs:
  build:
    runs-on: ubuntu-latest

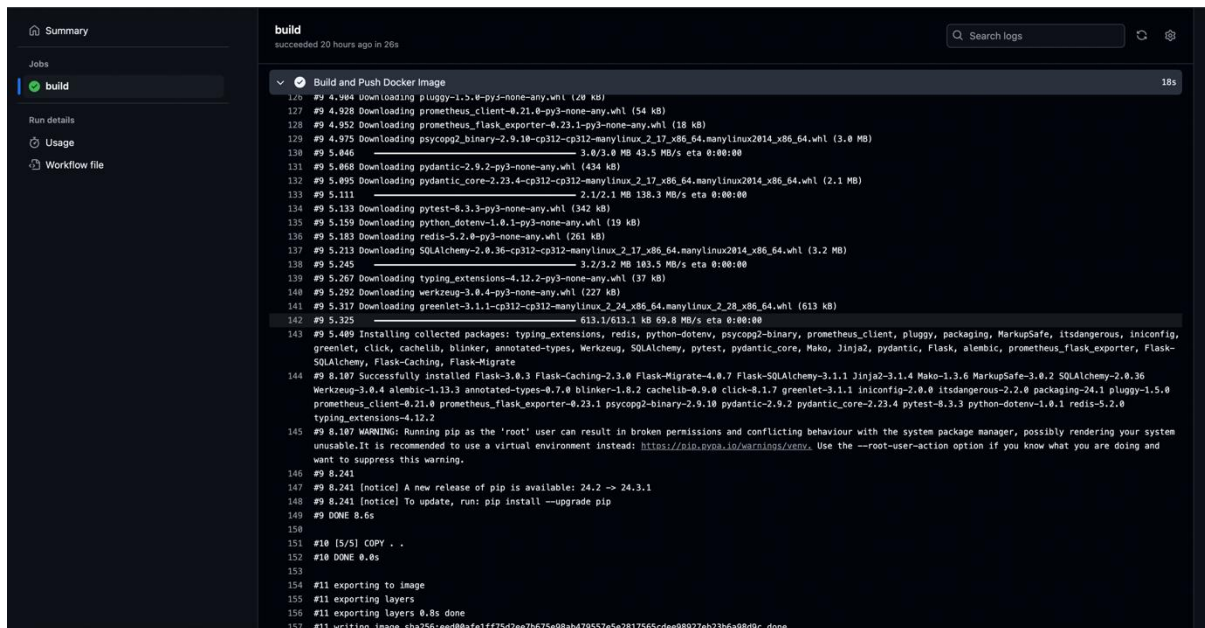
    steps:
      - name: Checkout repository
        uses: actions/checkout@v2

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v1

      - name: Log in to Docker Hub
        run: echo "${{ secrets.DOCKER_PASSWORD }}" | docker login --
username "${{ secrets.DOCKER_USERNAME }}" --password-stdin

      - name: Build and Push Docker Image
        run: |
          docker build -t mlestere/flask-app:latest .
          docker push mlestere /flask-app:latest
```

Он запускается при пуше в ветку main, выполняет проверку исходного кода, настраивает окружение для сборки Docker-образов, входит в Docker Hub с использованием секретов репозитория, собирает образ с тегом latest и загружает его в указанный репозиторий.



Был добавлен конфигурационный файл github actions, который запускает тесты и загружает и запускает приложение на удаленный сервер.

```
name: Tests and Deploy

on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v1

      - name: Build Docker Image
        run: |
          echo
          "SQLALCHEMY_DATABASE_URI=postgresql://postgres:secret@localhost:5432/flask_app" >> .env
          docker build -t myapp .

      - name: Start Services
        run: |
          docker run -d --name db -e POSTGRES_USER=postgres -e POSTGRES_PASSWORD=secret -e POSTGRES_DB=flask_app -p 5432:5432 postgres:13
          docker run -d --name redis -p 6379:6379 redis:alpine

      - name: Run Tests
        run: |
          docker run --network host myapp pytest
```

```

- name: Stop Services
  run: |
    docker stop db redis || true
    docker rm db redis || true

deploy:
  runs-on: ubuntu-latest
  needs: test

  steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Install sshpass
      run: sudo apt-get install -y sshpass

    - name: Deploy to Server
      run: |
        sshpass -p ${ secrets.SSH_PASSWORD } ssh -o
        StrictHostKeyChecking=no -p ${ secrets.SSH_PORT } ${
        secrets.SSH_USERNAME }@${ secrets.SSH_HOST } "cd
        /projects/CloudeService5 && git pull && docker-compose up -d --build &&
        docker-compose down"

```

Данный пайплайн будет запускать тесты, и при их успешном прохождении подключаться к удалённому серверу с подготовленным проектом, затем выполнять pull изменений из репозитория и перезапускать запущенное приложение.

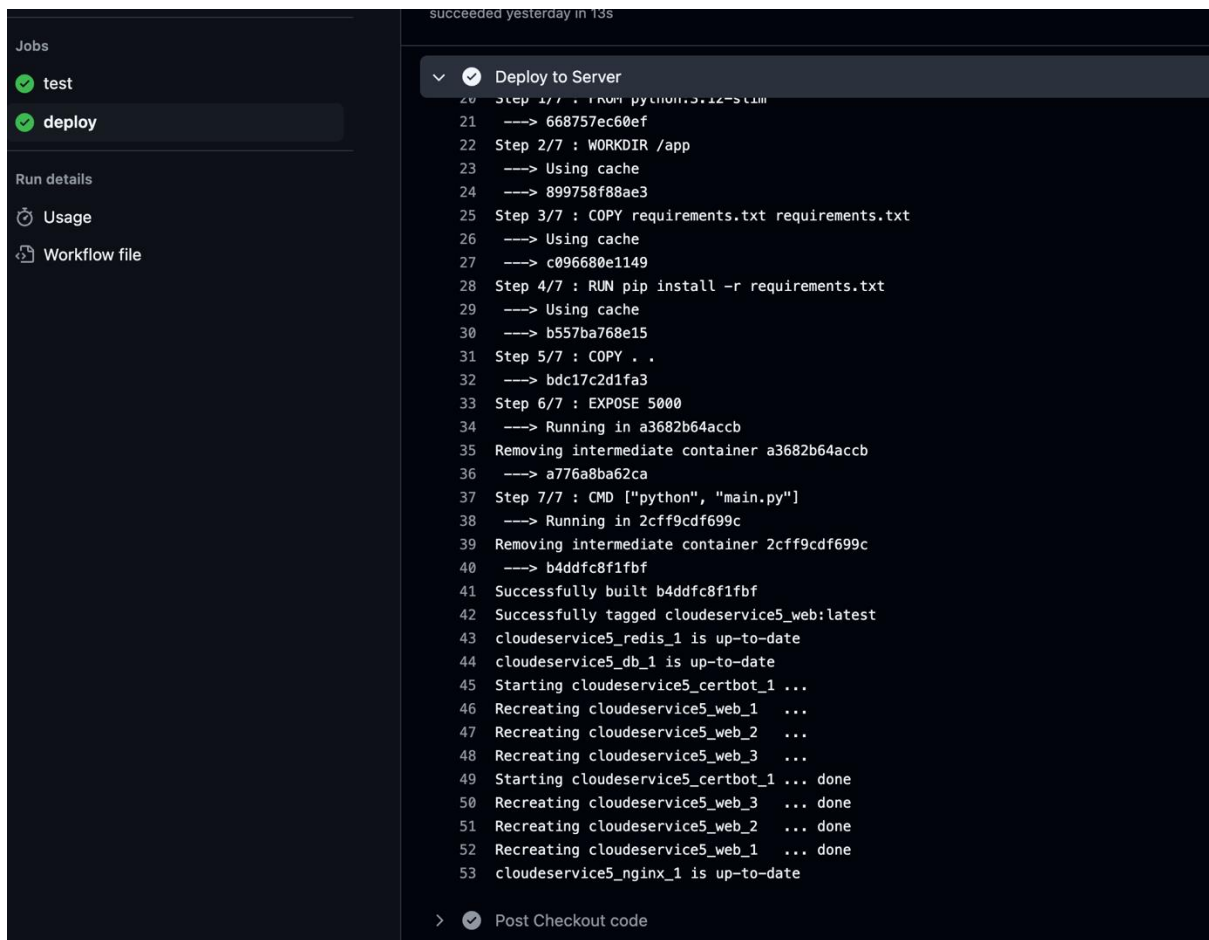
deploy.yml
on: push

test 53s → deploy 7s

Annotations
14 warnings

⚠ test
The following actions uses node12 which is deprecated and will be forced to run on node16: actions/checkout@v2, docker/setup-buildx-action@v1. For more info: <https://github.bl...>
[Show more](#)

⚠ test
The following actions use a deprecated Node.js version and will be forced to run on node20: actions/checkout@v2, docker/setup-buildx-action@v1. For more info: <https://github.b...>
[Show more](#)



6. Добавление SSL/TLS сертификатов

Для добавления SSL-сертификатов потребуется домен и сами сертификаты. Чтобы их создать, сначала запустим приложение, а затем с помощью Certbot сгенерируем ключи. Однако перед этим необходимо интегрировать Certbot в файл `docker-compose.yml`.

```
certbot:
  image: certbot/certbot
  volumes:
    - /etc/letsencrypt:/etc/letsencrypt
    - /var/www/certbot:/var/www/certbot
```

Далее, было необходимо изменить конфигурацию nginx сервера, так чтобы он принимал запросы и на 80 и на 443 порту, но запросы с 80 порта перенаправлял на 443, а также так, чтобы он использовал безопасное ssl соединение через домен.

```

events {
    worker_connections 1024;
}

http {
    upstream flask_app {
        server web:5000;
    }

    server {
        listen 80;
        server_name j-drift.ru www.j-drift.ru;

        location /.well-known/acme-challenge/ {
            root /var/www/certbot;
        }

        location / {
            return 301 https://$host$request_uri;
        }
    }

    server {
        listen 443 ssl;
        server_name j-drift.ru www.j-drift.ru;

        ssl_certificate /etc/letsencrypt/live/j-drift.ru/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/j-drift.ru/privkey.pem;

        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers HIGH:!aNULL:!MD5;
        ssl_prefer_server_ciphers on;

        location / {
            proxy_pass http://flask_app;
            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;
        }
    }
}

```

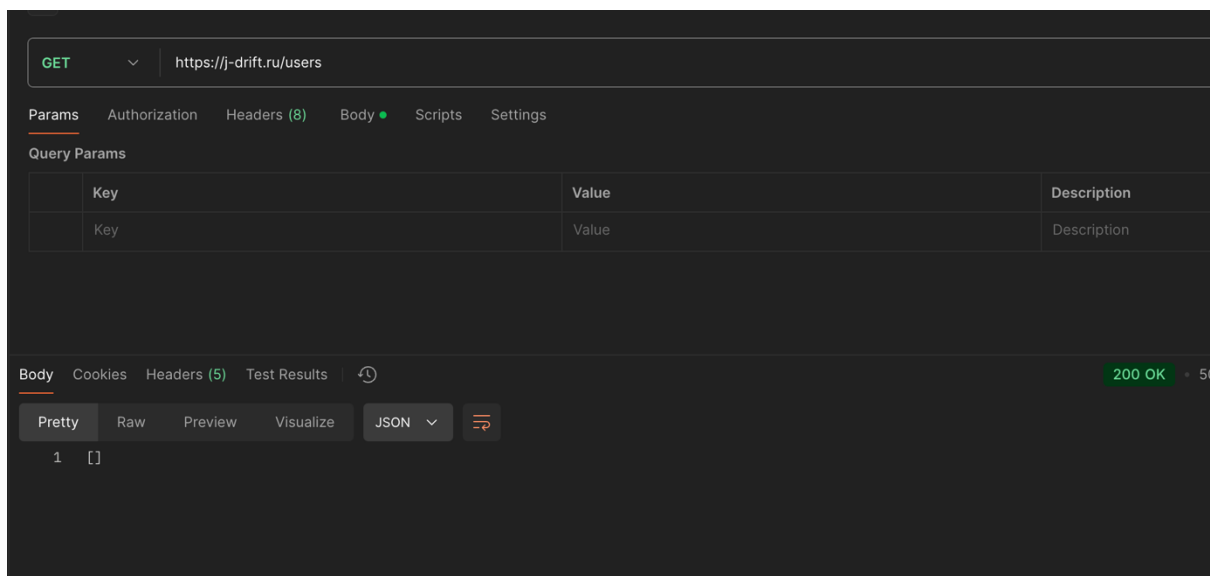
Также необходимо предоставить доступ к файлам с сертификатами для nginx через docker-compose.

```

nginx:
  image: nginx:latest
  volumes:
    - ./nginx.conf:/etc/nginx/nginx.conf
    - /etc/letsencrypt:/etc/letsencrypt
    - /var/www/certbot:/var/www/certbot
  ports:
    - "80:80"
    - "443:443"
  depends_on:
    - web

```

Проверка подключения по https:



7. Мониторинг и логгирование

Для установки Prometheus и Grafana, необходимо добавить их сервисы в docker-compose на портах 9090 и 3000.

```
prometheus:
  image: prom/prometheus
  volumes:
    - ./prometheus.yml:/etc/prometheus/prometheus.yml
  ports:
    - "9090:9090"
  depends_on:
    - web

grafana:
  image: grafana/grafana
  ports:
    - "3000:3000"
  depends_on:
    - prometheus
  environment:
    - GF_SECURITY_ADMIN_PASSWORD=secret
```

А также необходимо создать конфигурационный файл Prometheus.yml, предоставляющий метрики для анализа, в нашем случае – приложения, Prometheus, node-exporter.

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'flask_app'
    metrics_path: '/metrics'
    static_configs:
```

```

- targets: ['web:5000']

- job_name: 'node_exporter'
  static_configs:
    - targets: ['node-exporter:9100']

- job_name: 'prometheus'
  static_configs:
    - targets: ['localhost:9090']

```

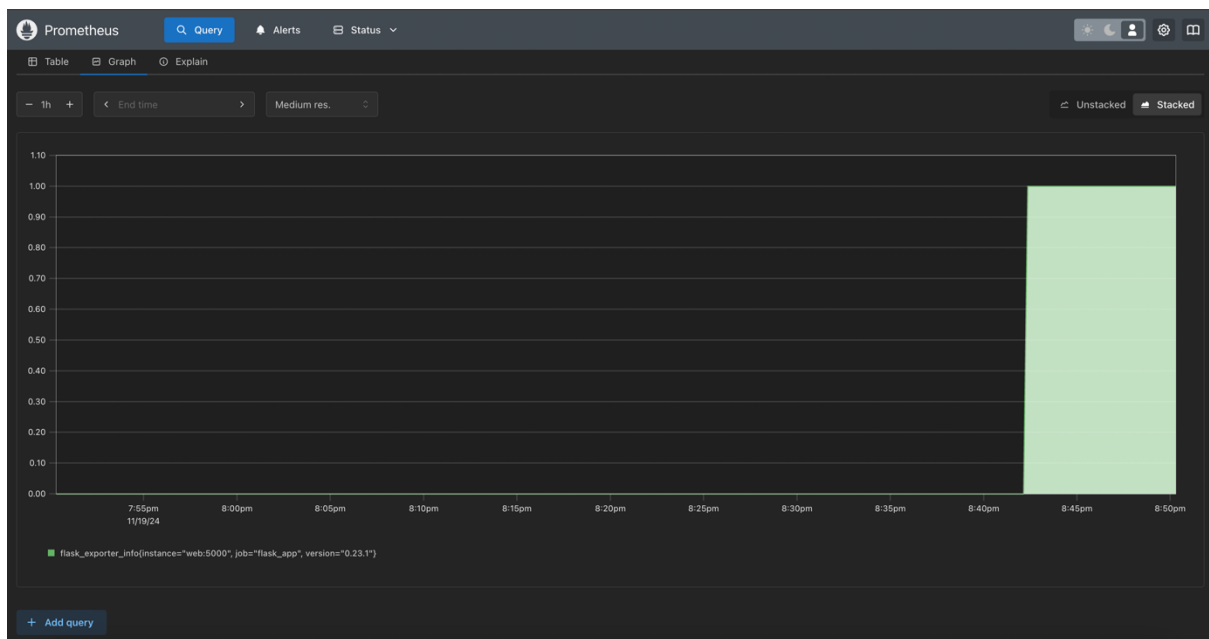
Для отображения логов приложения в Prometheus, в приложение был добавлен `prometheus_flask_exporter`, который создает эндпоинт `/metrics`.

```

metrics = PrometheusMetrics(app=app)
metrics.info('app_info', 'Application info', version='1.0.0')

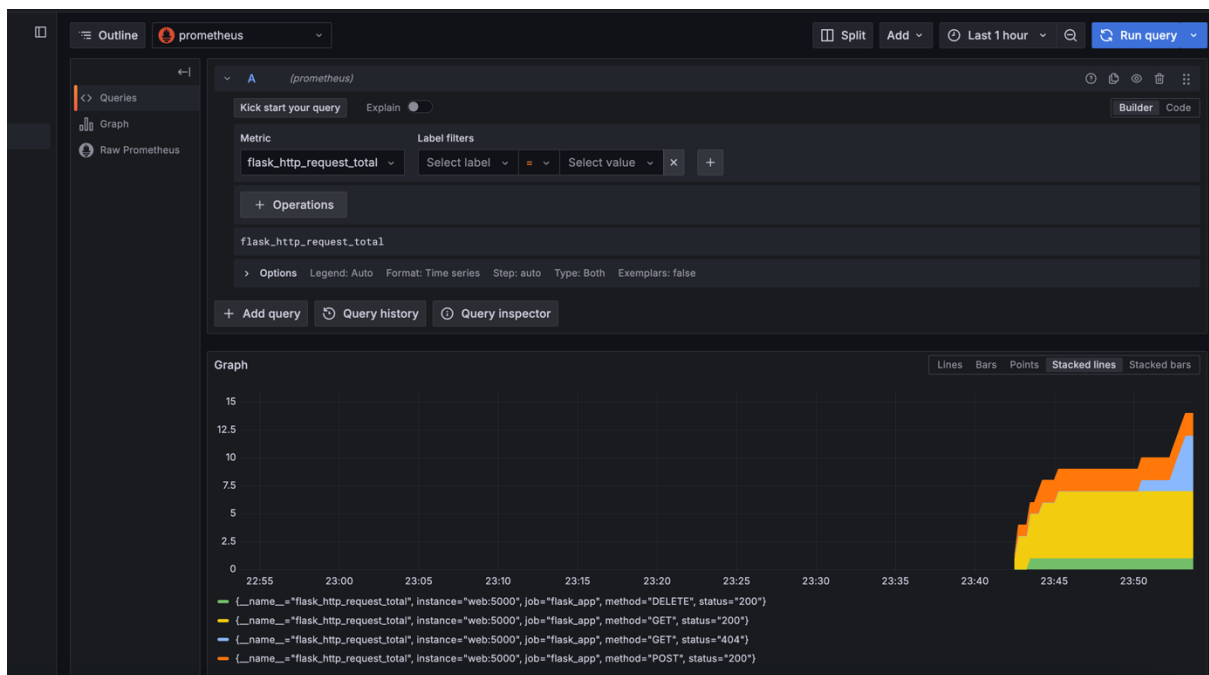
```

Был запущен Prometheus с Grafana. При переходе на порт 9090 попадаем на страницу Prometheus. Запрос:



При переходе на 3000 порт попадаем на Grafana. Чтобы Prometheus появился в Grafana необходимо добавить его в качестве сервера для данных.

Запрос в Grafana:



7.1. ELK Stack

Были добавлены 3 сервиса в docker-compose файл – elasticsearch, logstash, kibana, которые работают в тандеме, logstash берет логи из контейнера и направляет в elasticsearch, который индексирует данные и отправляет в kibana, которая отображает данные в приятном виде.

```
elasticsearch:
  image: elasticsearch:7.10.1
  environment:
    - discovery.type=single-node
  ports:
    - "9200:9200"
  volumes:
    - esdata:/usr/share/elasticsearch/data

logstash:
  hostname: logstash
  image: logstash:7.10.1
  volumes:
    - ./logstash.conf:/usr/share/logstash/pipeline/logstash.conf
  ports:
    - "5044:5044"
  depends_on:
    - elasticsearch

kibana:
  image: kibana:7.10.1
  ports:
    - "5601:5601"
  depends_on:
    - elasticsearch

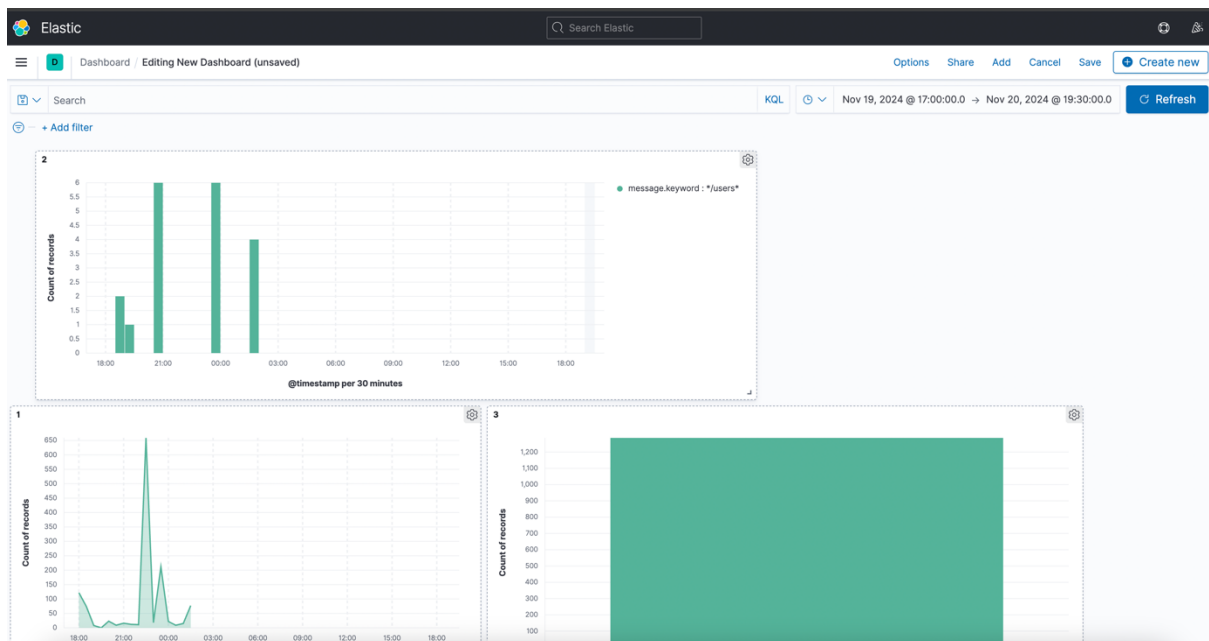
volumes:
  esdata:
```

Был добавлен конфигурационный файл logstash, который использует syslog для чтения и отправки логов контейнера в elasticsearch.

```
input {
  syslog {
    port => 5044
    type => "syslog"
    codec => json
  }
}

output {
  elasticsearch {
    hosts => ["elasticsearch:9200"]
    index => "docker-logs-%{+YYYY.MM.dd}"
  }
}
```

В Kibana по 5601 порту был создан дэшборд с логами.



8. Масштабирование с Docker Swarm

Был изменен docker-compose.yml файл. В данный файл была добавлена общая сеть для контейнеров, так как Swarm использует свой тип сети для создания кластера.

```
deploy:
  replicas: 3

networks:
  - app_network
```

```
networks:
  app_network:
    driver: overlay
```

Также был инициализирована нода swarm и создан кластер с приложением с помощью команды *docker stack deploy -c docker-compose.yml stack*.

The image shows two screenshots. The top one is a terminal window displaying the output of a `docker stack deploy` command. It shows a table of services with columns: ID, NAME, IMAGE, NODE, DESIRED STATE, CURRENT STATE, ERROR, and PORTS. The service `stack_web.1` is shown as `flask-app:latest` running on node `ample-copper.aeza.network` in a `Running` state.

| ID | NAME | IMAGE | NODE | DESIRED STATE | CURRENT STATE | ERROR | PORTS |
|------------|-------------|------------------|---------------------------|---------------|------------------------|-------|-------|
| ul632q8kme | stack_web.1 | flask-app:latest | ample-copper.aeza.network | Running | Running 19 minutes ago | | |

The bottom screenshot shows the Prometheus web interface. The query bar contains `>_ flask_http_request_total`. Below the query bar, there are tabs for `Table`, `Graph`, and `Explain`. The `Table` tab is selected, showing a table with the query results. The table has a header `Evaluation time` and three rows of data.

| Evaluation time |
|--|
| flask_http_request_total(instance="web:5000", job="flask_app", method="GET", status="404") |
| flask_http_request_total(instance="web:5000", job="flask_app", method="GET", status="200") |
| flask_http_request_total(instance="web:5000", job="flask_app", method="GET", status="500") |

At the bottom of the interface, there is a button labeled `+ Add query`.



https://j-drift.ru/users

POST



https://j-drift.ru/users

Params

Authorization

Headers (8)

Body

Scripts

Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON



```
1 {  
2   "email": "TESTUSER",  
3   "username": "usertest"  
4 }
```

Body

Cookies

Headers (5)

Test Results



Pretty

Raw

Preview

Visualize

JSON



```
1 {  
2   "message": "User created successfully",  
3   "user_id": 1  
4 }
```