Федеральное государственное автономное образовательное
учреждение высшего образования
«Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по дисциплине

«Облачные и туманные вычисления»

Выполнил студент группы P3415
Барсуков Максим Андреевич

Преподаватель:
Перл Ольга Вячеславовна

Санкт-Петербург
2025 г.

Содержание

Задание

Цель: Изучить разницу между простым запуском контейнера и оркестрацией сервиса. Эта работа закрывает пробел между «работает на моем ноутбуке» и «работает в кластере».

Сценарий: Вы выступаете в роли DevOps-инженеров простого интернет-магазина, которому нужно справиться с наплывом трафика (например, в «Черную пятницу»).

Приложение:

- **Frontend:** Веб-страница на Python/Flask, отображающая счётчик посещений.
- **Backend:** База данных Redis для хранения количества хитов.

Ход работы:

1. **Контейнеризация:** Написать Dockerfile для Python-приложения.
2. **Docker Compose:** Написать docker-compose.yml для локального запуска связки App + Redis. Убедиться, что они «видят» друг друга.
3. **Инициализация Swarm:** Перевести Docker в режим Swarm (docker swarm init).
4. **Развертывание стека (Deploy Stack):** Конвертировать файл compose в деплой стека (Stack Deploy).
5. **Тест на отказ (Self-healing):** Вы вручную «убиваете» (docker kill) активный контейнер и наблюдают, как Swarm автоматически пересоздаёт его.
6. **Масштабирование:** Выполнить команду docker service scale frontend=5.

Ключевые знания: Обнаружение сервисов (Service Discovery) — как приложение находит Redis по имени), реплики и самовосстанавливающиеся системы.

Приложение

Frontend

```
import os
from flask import Flask, jsonify
import redis

redis_client = redis.Redis(
    host=os.getenv("REDIS_HOST", "redis"),
    port=int(os.getenv("REDIS_PORT", "6379")),
    db=int(os.getenv("REDIS_DB", "0")),
    password=os.getenv("REDIS_PASSWORD"),
    decode_responses=True,
)

app = Flask(__name__)

@app.route("/")
def index():
    count = redis_client.incr("hits")
    return jsonify(visit_count=count), 200

@app.route("/health")
def health():
    try:
        redis_client.ping()
        return jsonify(status="ok"), 200
    except redis.RedisError as exc:
        return jsonify(status="failed", detail=str(exc)), 500

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

Backend

Хранилище представлено Redis. Для локального стенда пароль берётся из файла `.env`, порт Redis не открывается наружу. Данные в именованном `volume redis-data`.

Ход работы

Контейнеризация

Для Python-сервиса собран образ на базе `python:3.11-slim`. Переменные окружения задают адрес Redis и параметры запуска Flask. Зависимости ставятся отдельно.

```
FROM python:3.11-slim


ENV PYTHONDONTWRITEBYTECODE=1 \
    PYTHONUNBUFFERED=1 \
    FLASK_APP=app.py \
    FLASK_RUN_HOST=0.0.0.0 \
    FLASK_RUN_PORT=5000


WORKDIR /app
COPY app/requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt


COPY app/ ./
EXPOSE 5000
CMD ["flask", "run", "--host=0.0.0.0", "--port=5000"]
```

Список зависимостей приложения в `requirements.txt`:

```
Flask==2.3.3
redis==5.0.1
```

Docker-Compose

Файл `docker-compose.yml` описывает два сервиса: `frontend` и `redis`. Пароль Redis и остальные переменные читаются из `.env`. Порт Redis наружу не открывается, фронтэнд доступен на `localhost:5002`.

```
services:
    frontend:
        build: .
        image: lab1-frontend:latest
        ports:
            - "5002:5000"
        env_file: .env
        environment:
            - REDIS_HOST=${REDIS_HOST:-redis}
            - REDIS_PORT=${REDIS_PORT:-6379}
            - REDIS_DB=${REDIS_DB:-0}
            - REDIS_PASSWORD=${REDIS_PASSWORD}
        depends_on:
            - redis
        deploy:
            replicas: 1
            restart_policy:
                condition: any
    redis:
        image: redis:7-alpine
        env_file: .env
        command: ["redis-server", "--appendonly", "yes", "--requirepass",
"${REDIS_PASSWORD}"]
        volumes:
            - redis-data:/data
        deploy:
            replicas: 1
            restart_policy:
                condition: any


volumes:
    redis-data:
```

Пример конфигурации окружения (в репозитории хранится `.env.example`):

```
REDIS_PASSWORD=password
REDIS_HOST=redis
REDIS_PORT=6379
REDIS_DB=0
```

Локальный запуск и проверка (до перехода в Swarm), как показано на рисунках 1, 2:

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

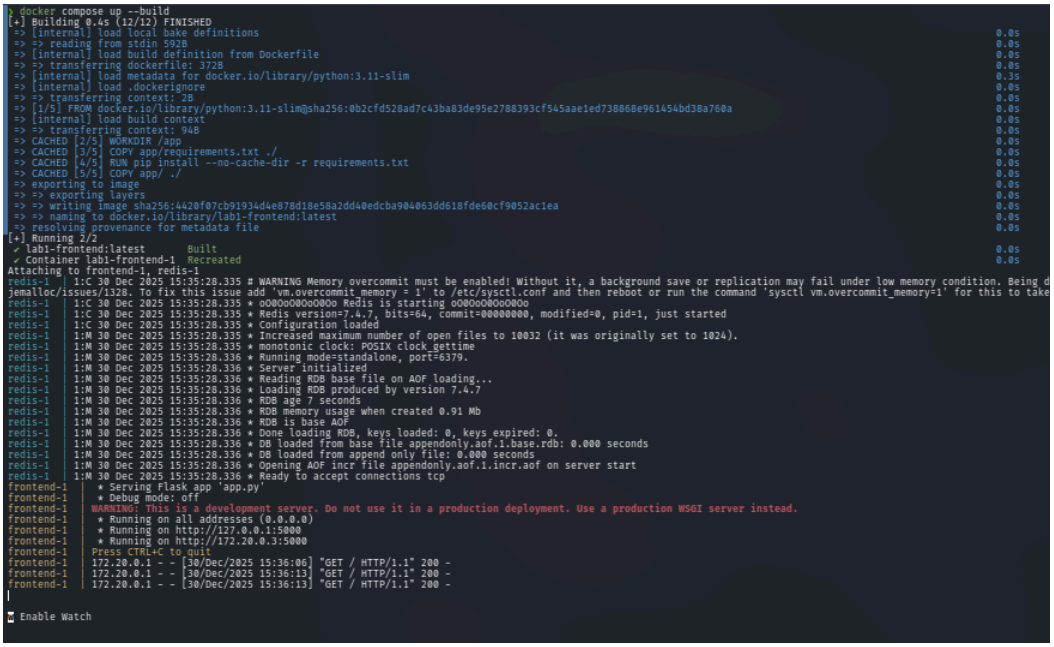

Рисунок 1 – Запуск команды docker compose up --build

```
curl -s http://localhost:5002/
```

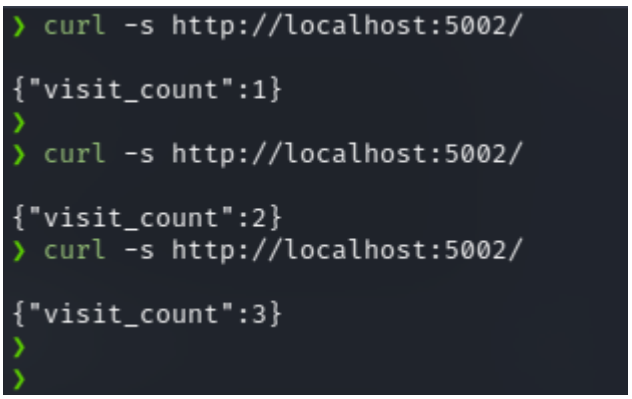

Рисунок 2 – Проверка работы сервиса

Инициализация Swarm

Перевод Docker в режим оркестрации. Команда выводит токены для присоединения узлов, как показано на рисунке 3:

```
docker swarm init
```

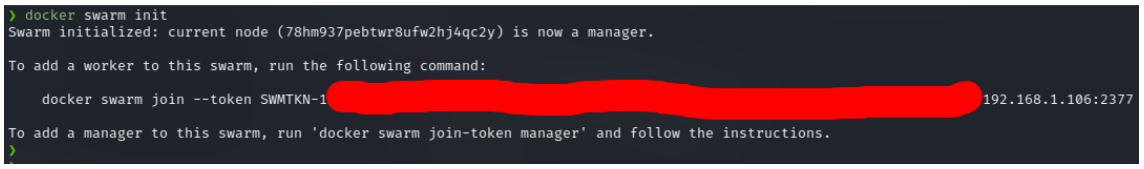

Рисунок 3 – Запуск docker swarm

Развертывание стека

Перед деплоем образ фронтенда собирается и тегируется, затем запускается стек app. Список сервисов показывает доступный порт фронта, как показано на рисунке 4.

```
docker compose build frontend
docker stack deploy -c docker-compose.yml app
docker stack services app
```

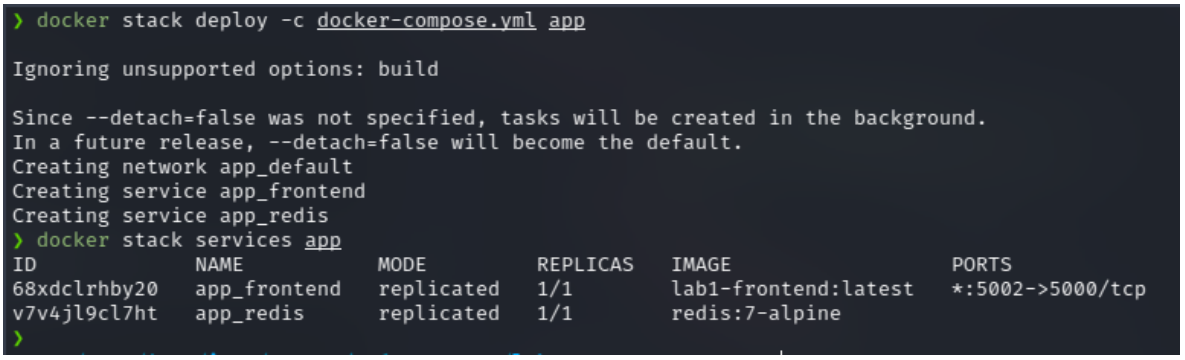

Рисунок 4 – Развертывание стека

Тест на отказ

Одна из реплик фронтенда намеренно завершается, после чего планировщик автоматически поднимает новую задачу, что видно по списку задач сервиса на рисунке 5.

```
docker kill app_frontend.1.ow8qi7jca4i2mqlksutx6gfsf
docker service ps app_frontend
```

```
❯ docker kill app_frontend.1.ow8qi7jca4i2mqlksutx6gfsf
app_frontend.1.ow8qi7jca4i2mqlksutx6gfsf
❯
❯
❯ docker service ps app_frontend
ID             NAME                  IMAGE                  NODE      DESIRED STATE   CURRENT STATE           ERROR                         PORTS
6l0eecthr7pc   app_frontend.1        lab1-frontend:latest   laptop    Ready           Ready 1 second ago
ow8qi7jca4i2    \_ app_frontend.1    lab1-frontend:latest   laptop    Shutdown        Failed 4 seconds ago    "task: non-zero exit (137)"
❯ docker service ps app_frontend
ID             NAME                  IMAGE                  NODE      DESIRED STATE   CURRENT STATE            ERROR                         PORTS
6l0eecthr7pc   app_frontend.1        lab1-frontend:latest   laptop    Running         Running 13 seconds ago
ow8qi7jca4i2    \_ app_frontend.1    lab1-frontend:latest   laptop    Shutdown        Failed 18 seconds ago    "task: non-zero exit (137)"
```

Рисунок 5 – Тест на отказ

Масштабирование

Масштабирование выполняется командой `docker service scale`. После этого у сервиса стало 5 реплик.

```
docker service scale app_frontend=5
```

```
❯ docker service scale app_frontend=5
app_frontend scaled to 5
overall progress: 5 out of 5 tasks
1/5: running   [==================================================>]
2/5: running   [==================================================>]
3/5: running   [==================================================>]
4/5: running   [==================================================>]
5/5: running   [==================================================>]
verify: Service app_frontend converged
```

Рисунок 6 – Масштабирование сервиса

Заключение

В ходе выполнения лабораторной работы было разработано и развернуто контейнеризованное веб-приложение на Flask с БД в Redis. Проект был первоначально настроен для локальной разработки с использованием Docker Compose, после чего та же конфигурация была успешно применена для развертывания в кластере Docker Swarm. В ходе работы были продемонстрированы ключевые возможности оркестратора: отказоустойчивость (автоматическое восстановление сервиса после сбоя) и горизонтальное масштабирование (увеличение количества экземпляров сервиса Flask до пяти). Конфигурационные параметры централизованы в файле .env, а доступ к Redis ограничен внутренней сетью. В результате получен опыт переноса приложения из среды одиночных контейнеров в полноценную оркестрируемую инфраструктуру.