**Лабораторная работа: Развёртывание отказоустойчивого веб-приложения с разделением компонентов во фреймворке Docker Swarm (время выполнения 30-35 минут)**

**Цель лабораторной работы**

В рамках лабораторной работы слушатели научатся развёртывать микросервисное веб-приложение, состоящее из трёх компонентов: **frontend**, **backend** и **база данных**, в кластере **Docker Swarm**. Приложение демонстрирует работу с веб-интерфейсом, API-сервисом и централизованным хранилищем данных.

Основной упор делается на реализацию отказоустойчивой архитектуры за счёт:

* масштабирования сервисов frontend и backend;
* использования **базы данных MySQL**, развернутой как сервис внутри Swarm;
* хранения данных на volume для обеспечения их устойчивости к сбоям контейнеров;
* тестирования доступности системы при отключении одного или нескольких узлов кластера.

**Результат выполнения**

После выполнения лабораторной работы студент должен уметь:

* Настраивать кластер Docker Swarm и объединять узлы;
* Разворачивать многокомпонентные приложения с помощью docker stack deploy;
* Использовать overlay-сети для взаимодействия сервисов;
* Конфигурировать работу Flask-приложения с внешней базой MySQL;
* Обеспечивать сохранность данных с помощью Docker volumes;
* Проверять отказоустойчивость путём симуляции падения узлов и масштабирования сервисов;
* Диагностировать состояние кластера и отдельных компонентов с помощью команд Docker.

## 2. Требования к окружению

Для прохождения лабораторной работы необходимо подготовить следующее окружение:

### 2.1. Аппаратные требования

| **Компонент** | **Характеристики** |
| --- | --- |
| Количество узлов | **3 виртуальные машины** (1 менеджер, 2 воркера) |
| CPU | от 2 ядер на узел |
| ОЗУ | от 4 ГБ на узел |
| Диск | от 40 ГБ на узел |

**Примечание:** Развёртывание возможно как в локальной среде (например, VirtualBox), так и в облаке на базе Cloud.ru.

### 2.2. Программное обеспечение

На каждой виртуальной машине должно быть установлено:

* Операционная система: **Linux (Ubuntu 22.04+, Debian 12+, CentOS 8+ и т.п.)**
* **Docker Engine** версии 20.10 и выше

Установите Docker и Docker Compose на виртуальных машинах:

С инструкцией установки вы можете ознакомиться на сайте - <https://docs.docker.com/engine/install/>, где представлены инструкции для установки Docker на различные операционные системы, в моём случае я буду использовать Ubuntu (<https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository> ).

Для начала мы установим требуемое ПО и добавим официальный репозиторий Docker в пакетный менеджер:

sudo apt-get update

sudo apt-get install ca-certificates curl -y

sudo install -m 0755 -d /etc/apt/keyrings

sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc

sudo chmod a+r /etc/apt/keyrings/docker.asc

После этого мы добавим ключ репозитория (вся конструкция — это одна команда):

echo \

"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \

$(. /etc/os-release && echo "${UBUNTU\_CODENAME:-$VERSION\_CODENAME}") stable" | \

sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update

Теперь мы готовы приступить к установке Docker, Compose и сопутствующего ПО:

sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin -y

Далее мы добавим нашего пользователя в группу docker, чтобы он мог использовать docker без повышения привилегий:

sudo usermod -aG $USER

после этого перезагрузите систему и проверьте работоспособность docker командой:

docker run hello-world

полученное сообщение подтвердит, что вы всё правильно установили и настроили:



Эти действия необходимо выполнить на всех трёх машинах.

### 2.3. Сетевые требования

* Узлы должны быть доступны друг другу по внутреннему сетевому интерфейсу (для создания Swarm overlay-сети).
* Порты, которые должны быть открыты:
  + TCP **2377** — для управления кластером Swarm;
  + TCP и UDP **7946** — для связи между нодами (распространение информации о сервисах);
  + UDP **4789** — для оверлейных сетей;
  + **8080/tcp** — доступ к frontend сервису;
  + **5000/tcp** — доступ к backend сервису (внутреннее взаимодействие).
* В учебных целях можно просто отключить файрволл на машинах.

### 2.4. Права доступа

На всех узлах должна быть возможность выполнять команды от имени суперпользователя (sudo).

### 2.5. Дополнительные настройки

* На всех узлах должны быть выставлены корректные **hostname** и прописаны друг другу в /etc/hosts или через приватную DNS, чтобы избежать проблем с именами в кластере.
* На время выполнения работы рекомендуется отключить фаерволлы или разрешить указанные выше порты.

2.6 Учётная запись на <https://hub.docker.com/>

Учётная запись будет необходима для того чтобы загрузить наши образы Docker в публичный реестр образов, так же вы можете вместо Docker Hub использовать собственный локальный реестр.

## 3. Требования к знаниям и навыкам

Для успешного выполнения лабораторной работы вы должны обладать следующими базовыми знаниями и навыками:

### 3.1. Базовые знания

* Понимание принципов работы контейнеризации и технологии Docker;
* Знание архитектуры микросервисных приложений (разделение на frontend, backend, БД);
* Основы работы с сетями в Docker (bridge, overlay, порты и т.д.);
* Понимание базовых принципов отказоустойчивости и масштабирования приложений;
* Знание основ клиент-серверной архитектуры веб-приложений.

### 3.2. Практические навыки

* Установка и настройка Docker Engine на Linux-системах;
* Работа с командами Docker CLI (docker run, docker ps, docker exec, docker-compose, docker swarm);
* Базовая работа с Linux (навигация по файловой системе, редактирование конфигурационных файлов, работа в терминале);
* Навыки работы с текстовыми редакторами в Linux (например, nano, vim, gedit);
* Работа с системными службами (systemctl, проверка статуса сервисов).

## 5. Начало работы над проектом: создание кластера Swarm и структуры приложения

### 5.1 Создание Docker Swarm

Сначала вам необходимо выяснить ip-адреса ваших машин, это можно сделать с помощью команды ip addr либо в графическом интерфейсе настроек

В моём случае адреса у узлов такие:

Основной сервер 10.0.2.10

Первый узел 10.0.2.11

Второй узел 10.0.2.12

На основной машине нам необходимо создать кластер Swarm, документация по Swarm доступна по адресу <https://docs.docker.com/engine/swarm/swarm-tutorial/create-swarm/>

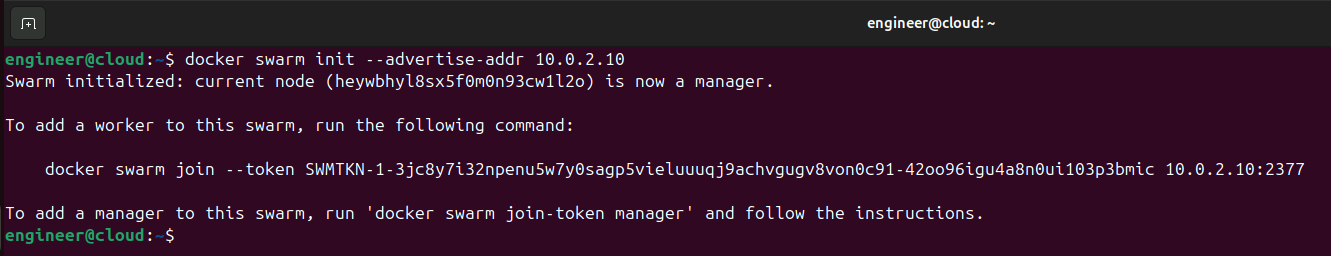
В терминале мы выполним команду по созданию кластера:

docker swarm init --advertise-addr <MANAGER-IP>

где вместо <MANAGER-IP> мы подставим IP-адрес машины, в моём случае 10.0.2.10

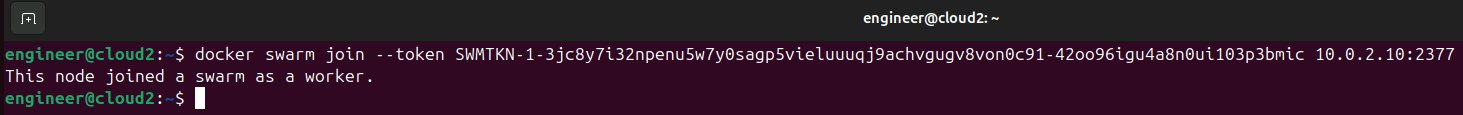
docker swarm init --advertise-addr 10.0.2.10

В ответ мы получим:



В сообщении указано, что текущая машина является менеджером кластера, а также представлена команда по добавлению узлов в кластер, которую необходимо выполнить на этих узлах, команду нужно скопировать удобным для вас способом на другие машины, например, для этого можно воспользоваться SSH, либо просто копированием/вставкой, если виртуальная машина это позволяет.

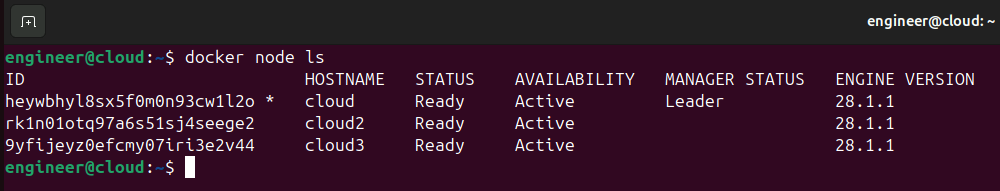
Когда вы выполните команду на других машинах, то получите сообщение:



После того, как вы добавите необходимые узлы, вы можете проверить ваш кластер выполнив команду на управляющем узле:

docker node ls

в ответ вы получите список узлов:



Таким образом вы создали кластер Swarm из трёх узлов, один из которых является его лидером.

### 5.2. Создание структуры каталогов и файлов

На управляющем узле создайте новую директорию для проекта:

mkdir swarm-app

cd swarm-app

Создаём базовую структуру папок для всех компонентов приложения:

mkdir backend frontend

Создаём директорию для хранения дополнительных конфигураций и файлов для базы данных:

mkdir mysql\_data

mkdir mysql-init

Структура проекта должна выглядеть так:

swarm-app/

├── backend/

├── frontend/

├── mysql\_data/

├── mysql-init

### 5.3. Создание файлов приложения

**1. Переходим в папку backend/ и создаём приложение на Flask:**

cd backend

touch app.py requirements.txt Dockerfile

cd ..

 app.py — исходный код сервера backend;

 requirements.txt — список зависимостей Python;

 Dockerfile — инструкции для сборки образа backend-приложения.

2. Переходим в папку frontend/ и создаём фронтенд-приложение:

cd frontend

touch default.conf Dockerfile

cd ..

 default.conf — конфигурационный файл Nginx

 Dockerfile — инструкции для сборки образа frontend-приложения.

cd mysql-init

touch init.sql

cd ..

3. В корне проекта создаём главный файл docker-swarm.yml, который будет описывать стек приложения:

touch docker-swarm.yml

5.3. Итоговая структура проекта на этом этапе

swarm-app/

├── backend/

│ ├── app.py

│ ├── requirements.txt

│ └── Dockerfile

├── frontend/

│ ├── default.conf

│ └── Dockerfile

├── mysql\_data/ # директория для volume MySQL

├── mysql-init/

│ └── init.sql

└── docker-swarm.yml

На этом этапе мы **подготовили каркас проекта**.  
Следующим шагом будет **заполнение файлов содержимым**:

* Код backend (Flask-приложение с подключением к MySQL);
* Конфигурация Nginx frontend (проксирование в приложение);
* Описание Dockerfile для каждого сервиса;
* Сборка docker-compose.yml для Swarm-стека.

## 5.4. Создание кода backend-приложения

### 5.4.1. Файл backend/requirements.txt

Сначала определим зависимости нашего приложения:

flask

flask-mysqldb

### 5.4.2. Файл backend/app.py

Создаём минимальный рабочий код Flask-приложения с подключением к MySQL.

from flask import Flask, request, redirect, url\_for, render\_template\_string

from flask\_mysqldb import MySQL

app = Flask(\_\_name\_\_)

# Настройки подключения к MySQL

app.config['MYSQL\_HOST'] = 'db'

app.config['MYSQL\_USER'] = 'user'

app.config['MYSQL\_PASSWORD'] = 'password'

app.config['MYSQL\_DB'] = 'appdb'

# Инициализация MySQL

mysql = MySQL(app)

# HTML-шаблон с Bootstrap

HTML\_TEMPLATE = '''

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Заметки в Docker Swarm</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css" rel="stylesheet">

</head>

<body class="bg-light">

<div class="container py-5">

<h1 class="mb-4 text-center">📝 Заметки в Swarm</h1>

<form method="post" action="/" class="mb-4">

<div class="input-group">

<input type="text" name="note" class="form-control" placeholder="Введите новую заметку" required>

<button class="btn btn-primary" type="submit">Добавить</button>

</div>

</form>

<div class="card shadow">

<div class="card-body">

{% if notes %}

<ul class="list-group">

{% for id, content in notes %}

<li class="list-group-item d-flex justify-content-between align-items-center">

<span>{{ content }}</span>

<span class="badge bg-secondary rounded-pill">#{{ id }}</span>

</li>

{% endfor %}

</ul>

{% else %}

<p class="text-muted">Пока нет ни одной заметки...</p>

{% endif %}

</div>

</div>

</div>

</body>

</html>

'''

@app.route('/', methods=['GET', 'POST'])

def index():

conn = mysql.connection

cursor = conn.cursor()

if request.method == 'POST':

# читаем поле note из формы

note = request.form.get('note')

if note:

cursor.execute("INSERT INTO entries (name) VALUES (%s)", (note,))

conn.commit()

return redirect(url\_for('index'))

# GET: читаем все записи

cursor.execute("SELECT id, name FROM entries ORDER BY id")

notes = cursor.fetchall()

cursor.close()

return render\_template\_string(HTML\_TEMPLATE, notes=notes)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(host='0.0.0.0', port=5000)

5.5. Создание Dockerfile для backend-приложения

Перейдите в директорию backend/ и создайте или откройте файл Dockerfile:

cd backend

nano Dockerfile

Вставьте следующий код:

FROM python:3.9-slim

# Установка зависимостей для компиляции mysqlclient

RUN apt-get update && apt-get install -y \

gcc \

default-libmysqlclient-dev \

pkg-config \

&& rm -rf /var/lib/apt/lists/\*

WORKDIR /app

COPY requirements.txt requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["python", "app.py"]

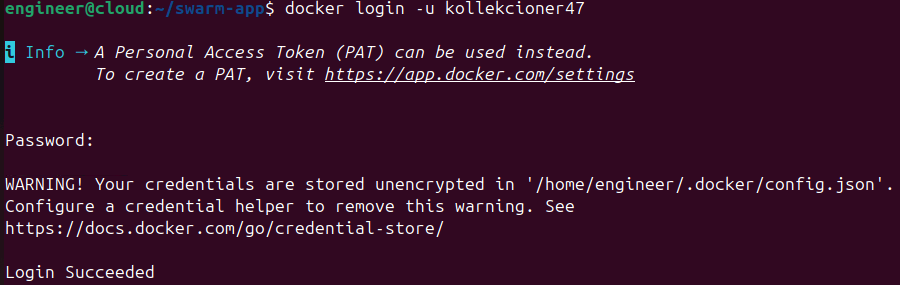
### Загрузка в реестр Docker Hub

После нам необходимо собрать образ и загрузить его в реестр, в качестве примера я буду использовать публичный реестр Docker Hub

Сначала вам нужно осуществить аутентификацию, для этого введите команду и свой логин и пароль:

docker login -u ваша\_учётная\_запись

после ввода логина и пароля вы увидите сообщение об успешном входе в учётную запись



После этого вы можете собрать ваш образ и загрузить его в реестр:

Docker build . -t ваша\_учётная\_запись/backend

Docker push ваша\_учётная\_запись/backend

5.6. Создание frontend-приложения

5.6.1. Файл frontend/default.conf

Перейдите в папку frontend и создайте файл default.conf:

cd ../frontend

nano default.conf

Вставьте следующий код:

server {

listen 80;

# Прокси всех запросов (GET, POST и т.д.) на backend

location / {

proxy\_pass http://backend:5000;

proxy\_http\_version 1.1;

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;

}

}

5.6.2. Файл frontend/Dockerfile

Создаём Dockerfile для frontend:

nano Dockerfile

И вставляем код:

# Используем nginx как веб-сервер

FROM nginx:1.27.5-alpine

# Копируем HTML-файл в стандартную директорию nginx

COPY default.conf /etc/nginx/conf.d/default.conf

После этого вы можете собрать ваш образ и загрузить его в реестр:

docker build . -t ваша\_учётная\_запись/frontend

docker push ваша\_учётная\_запись/ frontend

cd ..

5.7 Создание файла mysql-init/init.sql

cd mysql-init

nano init.sql

Вставьте код, который создаёт целевую таблицу:

CREATE DATABASE IF NOT EXISTS appdb;

USE appdb;

CREATE TABLE IF NOT EXISTS entries (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255) NOT NULL

);

После сохраните и вернитесь на уровень вверх:

cd ..

## 6. Создание docker-swarm.yml для запуска в Docker Swarm

Мы создадим файл, который:

* запускает три сервиса:
  + frontend — перенаправленеи к приложению
  + backend — Flask API
  + db —MySQL
* подключает их в одну **оверлейную сеть**
* сохраняет данные MySQL в **volume**
* указывает количество реплик (frontend и backend — по 2)

Создайте файл в корне проекта:

nano docker-swarm.yml

Вставьте:

services:

db:

image: mysql:8.0

environment:

MYSQL\_ROOT\_PASSWORD: rootpass

MYSQL\_DATABASE: appdb

MYSQL\_USER: user

MYSQL\_PASSWORD: password

volumes:

- db-data:/var/lib/mysql

- ./mysql-init:/docker-entrypoint-initdb.d

networks:

- appnet

deploy:

placement:

constraints: [node.role == manager]

backend:

image: kollekcioner47/backend

depends\_on:

- db

environment:

MYSQL\_DATABASE\_HOST: db

MYSQL\_DATABASE\_USER: user

MYSQL\_DATABASE\_PASSWORD: password

MYSQL\_DATABASE\_NAME: appdb

networks:

- appnet

deploy:

replicas: 2

restart\_policy:

condition: on-failure

frontend:

image: kollekcioner47/frontend

ports:

- "8080:80"

networks:

- appnet

depends\_on:

- backend

deploy:

replicas: 2

restart\_policy:

condition: on-failure

volumes:

db-data:

networks:

appnet:

driver: overlay

### Объяснение ключевых моментов

* volumes: db-data — сохраняет базу между перезапусками.
* deploy.replicas — создаёт отказоустойчивость фронта и бэка.
* networks.overlay — даёт сервисам доступ друг к другу по имени (например, http://backend:5000).
* placement.constraints — закрепляем базу только за менеджером (где volume будет доступен локально). Это простое решение — более сложное потребует Galera или Vitess.

### Развёртывание в Swarm

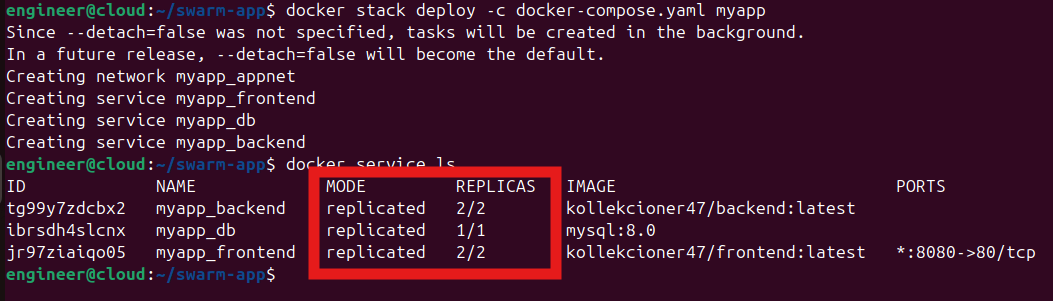
Если кластер уже инициализирован (docker swarm init), то для развертывания:

docker stack deploy -c docker-compose.yml myapp

Далее нужно подождать несколько минут, пока скачаются образы и запустится приложение, проверить статус можно с помощью команды:

docker service ls

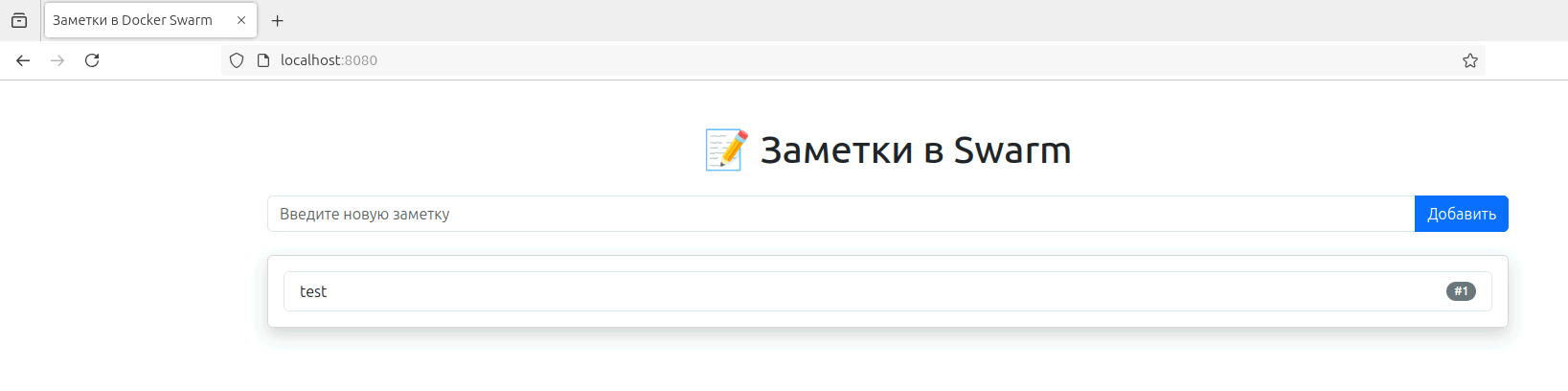
Все контейнеры должны быть в статусе replicated и полное количество реплик:



## ****6. Тестирование отказоустойчивости и работоспособности приложения****

### **6.1. Проверка доступности веб-интерфейса**

1. Открой браузер и перейди по IP-адресу любого узла к порту, который проброшен (например, http://<IP-адрес-любого-ноды>:8080 или просто http://localhost:8080).
2. Убедись, что:
   * Загружается веб-интерфейс.
   * Отображаются записи (если добавлялись).
   * Можно добавить новую запись — она сохраняется и отображается после обновления.



### **6.2. Проверка отказоустойчивости backend**

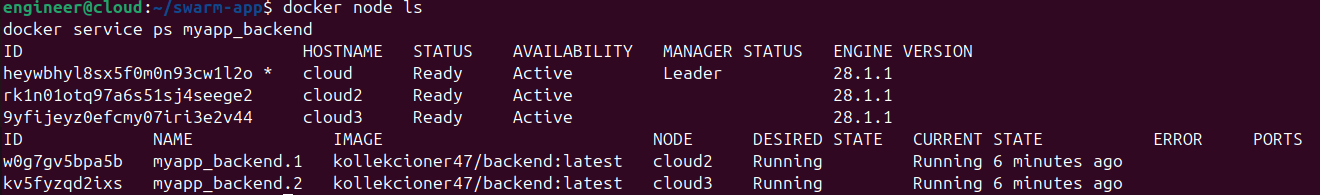
**Тест 1: "Что будет, если упадёт один рабочий сервер?"**

1. Открываем терминал и смотрим, кто где работает:

docker node ls

docker service ps myapp\_backend

Убедись, что myapp\_backend запущен на двух разных воркерах.



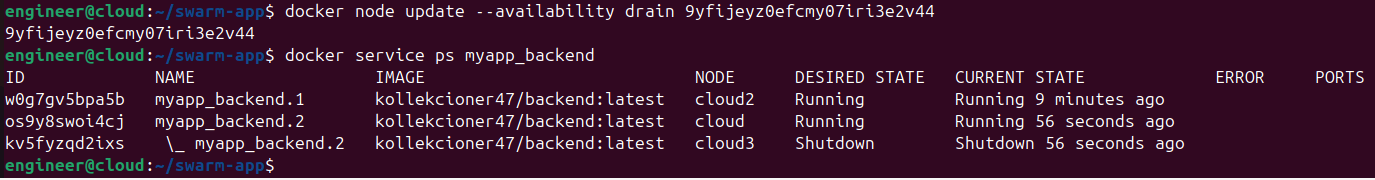
1. Теперь симулируем отказ: говорим одному воркеру, что он временно "устал":

docker node update --availability drain <ID\_ноды>

Всё! Swarm сразу же поймёт: «О, этот сервер недоступен, надо срочно перенести задачу!»

1. Через несколько секунд:

docker service ps myapp\_backend



Ты увидишь, как реплика, которая была на ушедшем воркере, перезапускается на другом — всё **продолжает работать**!

1. Проверь браузером: заметки отображаются? Можно добавить новую? Всё ок?

**Тест 2: "Что будет, если упадёт менеджер?"**

У нас база данных прикреплена к **менеджеру**. А значит:

* Пока работает менеджер — всё супер.
* Упадёт менеджер — база недоступна.

Что делаем:

1. Идём на виртуалку с менеджером и просто выключаем Docker (или саму машину):

sudo systemctl stop docker

1. Теперь попробуй открыть сайт и добавить новую заметку.

Что произойдёт? Ошибка — и это **нормально**! Бэкенд не может подключиться к MySQL.

Включаем менеджер обратно:

sudo systemctl start docker

Ждём 5–10 секунд. Swarm восстанавливает myapp\_db, и всё снова работает!

## Тест 3: "А если нам нужен больше мощности?"

Хочешь обработать больше пользователей? Легко!

docker service scale myapp\_backend=5

docker service scale myapp\_frontend=4

Swarm сам раскидает новые копии по всем свободным серверам.

Ты даже не думаешь, куда и как — всё делает кластер!

**Итого**

* Один сервер падает — приложение продолжает работать.
* Менеджер падает — база временно недоступна, но восстанавливается автоматически.
* Масштабирование — это просто одна команда.

Docker Swarm — это простой и мощный инструмент для создания отказоустойчивых распределённых приложений. Даже если часть инфраструктуры выходит из строя, Swarm сам всё восстанавливает, перераспределяет и масштабирует — без участия человека. И главное — вы только что сами это увидели и протестировали вживую.