Министерство цифрового развития, связи и   
массовых коммуникаций Российской Федерации

Сибирский государственный университет телекоммуникаций и информатики

**Лабораторно-практическая работа №2**

по дисциплине: ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Настройка среды для создания программного приложения

**Выполнил**: Макаров М. С.

**Группа**: ДПО-43

**Вариант: 6**

**Проверил**: Полетайкин А. Н.

Новосибирск, 2025

**Оглавление**

[Тема 3](#_Toc202188877)

[Цель 3](#_Toc202188878)

[Задание 3](#_Toc202188879)

[Описание выполненных действий, результаты выполнения 5](#_Toc202188880)

[Разработка индивидуального задания на разработку ИС. Разработка функциональных требований и требований к обеспечению ИС 5](#_Toc202188881)

[Краткий ход разработки приложения 6](#_Toc202188882)

[Ссылки 12](#_Toc202188883)

[Приложения 13](#_Toc202188884)

[Приложение 1. Содержимое файла views.py 13](#_Toc202188885)

[Приложение 2. Код html-страницы шаблона 14](#_Toc202188886)

[Приложение 3. Содержимое Dockerfile 16](#_Toc202188887)

[Приложение 3. Содержимое docker-compose.yaml 17](#_Toc202188888)

# Тема

Настройка среды для создания программного приложения.

# Цель

Освоение навыков установки и настройки СУБД и средств разработки приложения на примере СУБД PostgreSQL и фрейморка Django. Освоение техники установки Docker Debian и работы с репозиториями Docker. Изучение средств создания образа для контейнера с приложением, освоение техники запуска контейнеров Docker с использованием Dockerfile.

# Задание

1. Согласно индивидуальному заданию на разработку ИС сформулировать её название, разработать функциональные требования и требования к обеспечению ИС.
2. Согласно предъявленным требованиям установить СУБД и фреймворк (или WEB-сервер с PHP). Виртуальную среду создавать с уникальным именем проекта.
3. Создать приложение, реализующее 1-2 базовых функциональных требований из числа заявленных ранее. Это может быть, например, разработка страницы для парсинга сайта или для пользовательского внесения справочных данных в БД. Главное требование: страницы должны быть динамические.
4. В гитлабе создать инициализирующий коммит. Разместить в репозитории созданный проект приложения.
5. Провести установку Docker Debian.
6. С помощью *Dockerfile* создать образ для реализации созданного приложения. Если в качестве средств разработки используется *Python Django*, то в качестве исходного образа можно использовать официальный образ Python версии 3.11 (FROM python:3.11). Можно использовать тег *latest* вместо 3.11, но он может быть не очень стабильным.
7. Создать WEB-приложение в контейнере *docker***.**
8. Загрузить образ на докер-хаб.
9. В ВМ с ОС Debian в каталоге */home/[имя пользователя]* при помощи команд утилиты *git* склонировать ранее созданный удаленный репозиторий, в отчет предоставить скриншот папки клонированного репозитория.
10. В папке с репозиторием создать папку LAB2, в которую выложить отчет, docker-compose.yaml со всеми параметрами использования вашего образа на докер-хаб и запуска контейнера.
11. Выполнить коммит изменений в локальном репозитории, выполнить push в удаленный репозиторий.
12. Запустить контейнер: docker-compose up –d. Проверить работоспособность.
13. После выполнения всех работ и получения положительного результата рекомендуется удалить все неработающие контейнеры и образы, с ними связанные командой docker system prune --all

# Описание выполненных действий, результаты выполнения

## Разработка индивидуального задания на разработку ИС. Разработка функциональных требований и требований к обеспечению ИС

В ходе анализа поставленной задачи было решено разработать простое веб-приложение для управления персональным списком задач. Разработка такого приложения позволит в полной мере освоить базовые навыки установки и настройки СУБД и средств разработки приложения на примере СУБД PostgreSQL и фрейморка Django, освоить технику установки Docker Debian и работы с репозиториями Docker, изучить средства создания образа для контейнера с приложением и освоить технику запуска контейнеров Docker с использованием Dockerfile.

**1. Спецификация требований к ИС**

**1.1. Название ИС:** TaskMaster

**1.2. Цель ИС:** Разработка простого и интуитивно понятного веб-приложения для управления персональным списком задач.

**1.3. Функциональные требования:**

**FR-1 (Просмотр задач):** Пользователь должен иметь возможность просматривать полный список всех добавленных задач на главной странице приложения.

**FR-2 (Добавление задачи):** Пользователь должен иметь возможность добавить новую задачу в список через форму на главной странице. Новая задача по умолчанию имеет статус "не выполнена".

**FR-3 (Завершение задачи):** Пользователь должен иметь возможность отметить любую задачу как "выполненную". Визуально выполненная задача должна отличаться от невыполненной (например, перечеркнутым текстом).

**FR-4 (Удаление задачи):** Пользователь должен иметь возможность удалить любую задачу из списка.

**FR-5 (Сортировка задач):** Задачи в списке должны быть отсортированы по дате создания в обратном порядке (самые новые вверху).

**1.4. Требования к обеспечению (нефункциональные):**

**NFR-1 (Операционная система):** Debian.

**NFR-2 (Серверная часть):** Язык программирования Python 3, фреймворк Django.

**NFR-3 (Система управления базами данных):** PostgreSQL.

**NFR-4 (Среда развертывания):** Приложение должно быть контейнеризировано с использованием Docker и запускаться с помощью Docker Compose.

**NFR-5 (Система контроля версий):** Исходный код проекта должен храниться в удаленном Git-репозитории.

## Краткий ход разработки приложения

Разработка информационной системы TaskMaster проходила в несколько ключевых этапов, охватывающих подготовку среды, написание кода приложения, его контейнеризацию и развертывание.

**Этап 1: Подготовка среды разработки (ВМ №1)**

Первым шагом стала настройка виртуальной машины на базе ОС Debian. Была установлена СУБД PostgreSQL. Для взаимодействия с ней был создан специальный пользователь userdb с паролем и база данных study. Для обеспечения безопасного подключения из нашего будущего приложения был отредактирован конфигурационный файл pg\_hba.conf, разрешающий локальные подключения для пользователя userdb к базе study по паролю (метод md5).

Далее была подготовлена среда для разработки на Python. С помощью менеджера пакетов apt был установлен python3-pip и python3-venv. В специально созданной директории /django-apps была развернута и активирована виртуальная среда taskmaster\_env. Это позволило изолировать зависимости проекта от системных пакетов. Внутри активированной среды с помощью pip были установлены основные зависимости: фреймворк django и драйвер для работы с PostgreSQL — psycopg2-binary.

**Этап 2: Разработка Django-приложения**

Внутри виртуальной среды был создан проект Django с именем todo\_project и приложение tasks. В файле settings.py проекта были прописаны учетные данные для подключения к ранее настроенной базе данных PostgreSQL и добавлен адрес \* в ALLOWED\_HOSTS для удобства разработки.

Разработка самого приложения велась в соответствии с архитектурой MVT (Model-View-Template), принятой в Django:

1. **Model:** В файле tasks/models.py была описана модель Task, представляющая одну задачу. Модель содержит поля: title (название задачи), completed (статус выполнения, булево значение) и created\_at (дата и время создания). Также в Meta-классе была задана сортировка по умолчанию, чтобы новые задачи отображались первыми.
2. **View:** В файле tasks/views.py были реализованы три функции-представления. task\_list отвечает за отображение списка всех задач и обработку POST-запроса от формы для добавления новой задачи. complete\_task и delete\_task принимают id задачи в качестве параметра, находят соответствующий объект в базе данных, изменяют его статус или удаляют, после чего перенаправляют пользователя обратно на главную страницу. Содержимое файла представлено на рисунке 1. Код в текстовом формате находится в приложении 1.

.

Рисунок 1 – содержимое views.py

1. **Template:** В директории tasks/templates/tasks/ был создан HTML-шаблон task\_list.html. С помощью синтаксиса шаблонов Django в нем был организован цикл {% for %} для вывода всех задач, условные блоки {% if %} для отображения кнопок "Выполнить" или "Удалить" в зависимости от статуса задачи, а также форма для добавления новой задачи. Содержимое файла представлено на рисунке 2, текстовый формат в приложении 2.



Рисунок 2 – содержимое файла task\_list.html

1. **URLs:** Были настроены маршруты. В файле tasks/urls.py созданы пути для каждого представления, а затем этот файл был подключен к основному файлу todo\_project/urls.py.

После создания моделей были выполнены миграции (makemigrations и migrate), которые создали соответствующую таблицу в базе данных study. Приложение было протестировано локально с помощью встроенного веб-сервера Django.

**Этап 3: Контейнеризация и развертывание (ВМ №2)**

Для развертывания приложения была использована вторая, "чистая" ВМ, на которую были установлены Docker и Docker Compose.

Команды, использованные для проверки корректности установки Docker и Docker Compose приведены на рисунке 3.

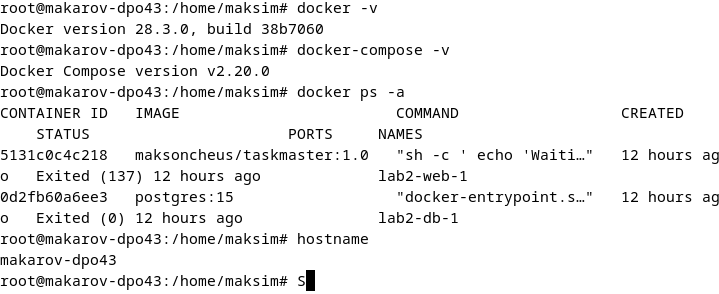


Рисунок 3 – проверка установки Docker и Docker Compose

Весь код проекта был помещен в Git-репозиторий. Для подготовки к контейнеризации в корне проекта был создан файл requirements.txt командой pip freeze, содержащий все Python-зависимости. Итоговое содержимое репозитория представлено на рисунке 4.

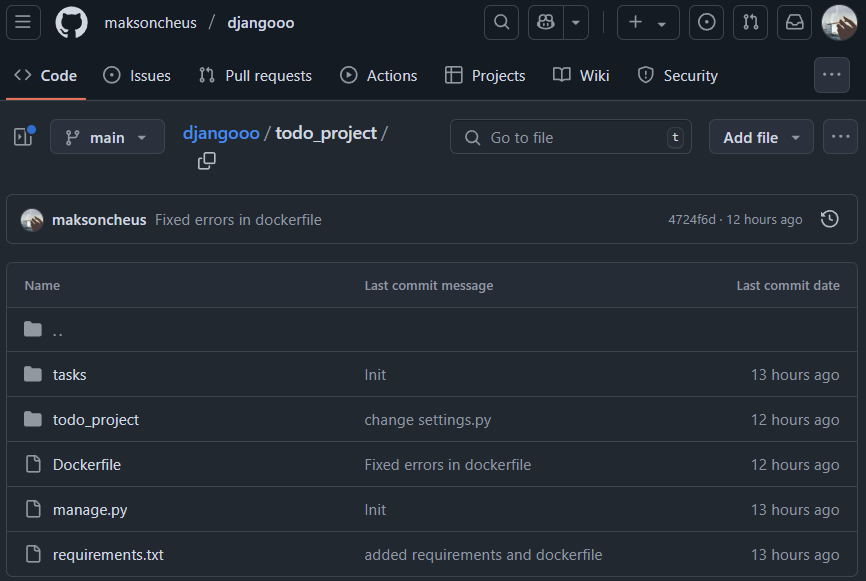


Рисунок 4 – содержимое репозитория с приложением

Был написан Dockerfile, описывающий процесс сборки образа для нашего веб-приложения. Он использует официальный образ python:3.11 в качестве основы, копирует requirements.txt, устанавливает зависимости, а затем копирует весь код проекта внутрь образа. Команда CMD была определена для запуска миграций и веб-сервера при старте контейнера. Графически содержимое файла Dockerfile представлено на рисунке 5. Текстовый вариант находится в приложении 3.

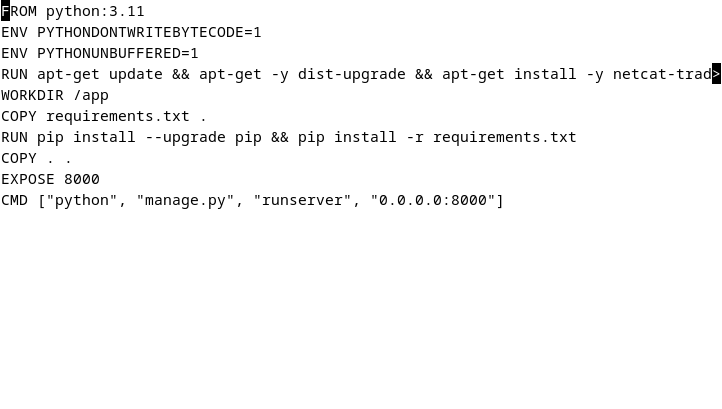


Рисунок 5 – содержимое Dockerfile

Собранный образ был помечен тегом maksoncheus/taskmaster:1.0 и загружен в публичный репозиторий на Docker Hub. Подтверждение тому можно найти на рисунке 6.

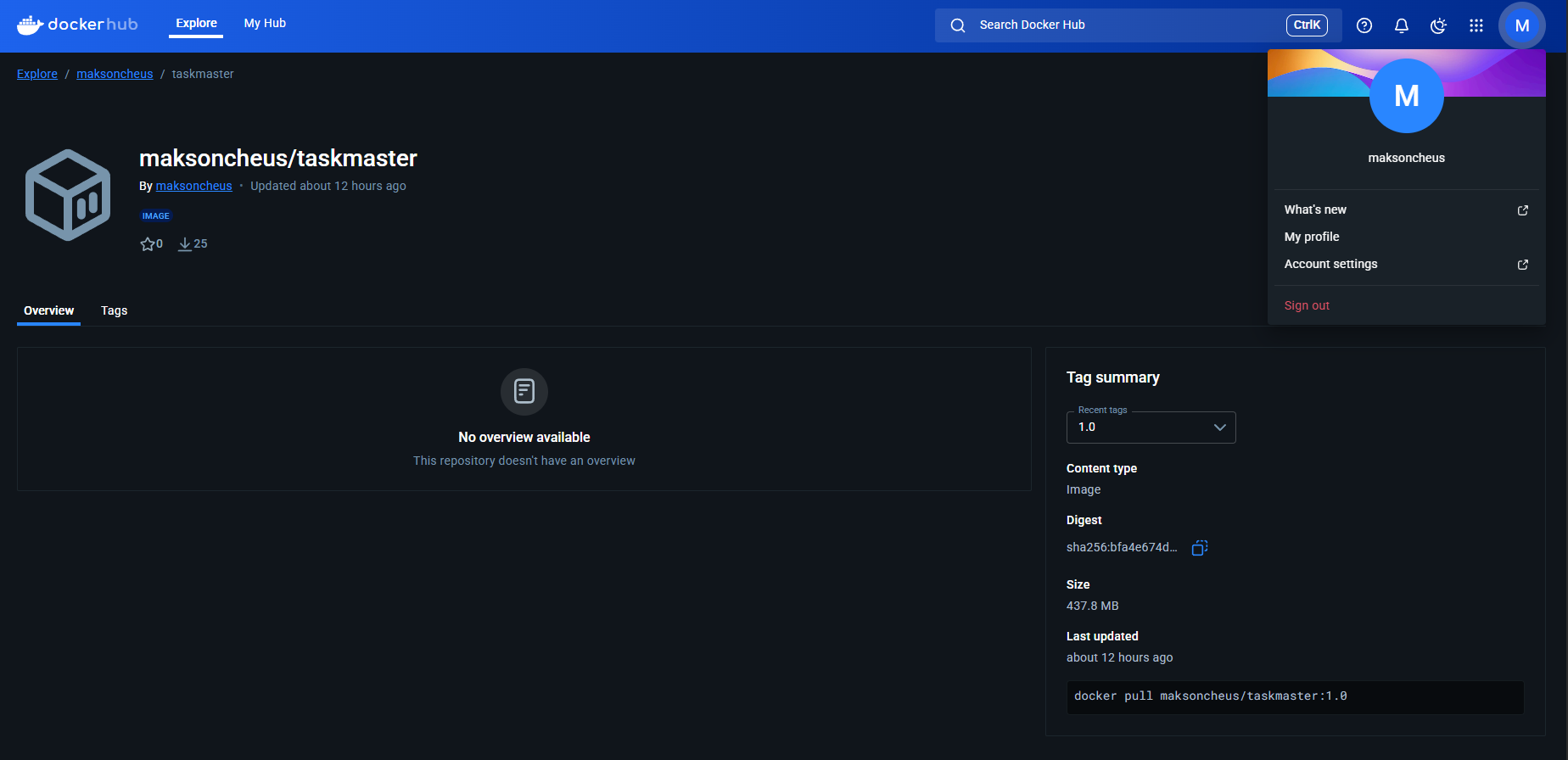


Рисунок 6 – образ на Docker Hub

Финальным шагом стала оркестрация приложения и базы данных с помощью docker-compose.yaml. В этом файле были описаны два сервиса:

* db: запускает контейнер с PostgreSQL из официального образа, пробрасывает переменные окружения для создания нужной базы и пользователя и использует volume для персистентного хранения данных.
* web: запускает контейнер из нашего загруженного на Docker Hub образа. С помощью depends\_on задана зависимость от сервиса db, гарантирующая, что веб-приложение запустится только после старта базы данных. Переменные окружения используются для передачи в приложение учетных данных для подключения к БД (имя хоста db стало доступно благодаря внутренней сети Docker). Порт 8000 контейнера был проброшен на порт 8000 хост-машины.

Содержимое файла docker-compose.yaml представлено графически на рисунке 7, текстовый формат в приложении 4.



Рисунок 7 – содержимое файла docker-compose.yaml

После запуска командой docker-compose up -d оба сервиса успешно стартовали, и приложение TaskMaster стало доступно в сети, полностью работая в изолированной контейнерной среде.

## Ссылки

Ссылка на репозиторий с разработанным приложением и docker-compose:

<https://github.com/maksoncheus/djangooo.git>

Ссылка на образ в Docker Hub:

<https://hub.docker.com/r/maksoncheus/taskmaster>

# Приложения

## Приложение 1. Содержимое файла views.py

from django.shortcuts import render, redirect

from .models import Task

def task\_list(request):

if request.method == 'POST':

title = request.POST.get('title')

if title:

Task.objects.create(title=title)

return redirect('task\_list')

tasks = Task.objects.all()

return render(request, 'tasks/task\_list.html', {'tasks': tasks})

def complete\_task(request, task\_id):

task = Task.objects.get(id=task\_id)

task.completed = True

task.save()

return redirect('task\_list')

def delete\_task(request, task\_id):

task = Task.objects.get(id=task\_id)

task.delete()

return redirect('task\_list')

## Приложение 2. Код html-страницы шаблона

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<title>TaskMaster - Список задач</title>

<style>

body { font-family: sans-serif; max-width: 800px; margin: auto; padding: 20px; background-color: #f4f4f4; }

h1 { color: #333; }

.task-container { background: #fff; padding: 15px; border-radius: 5px; margin-top: 20px; }

.task { display: flex; align-items: center; justify-content: space-between; padding: 10px; border-bottom: 1px solid #eee; }

.task:last-child { border-bottom: none; }

.task.completed { text-decoration: line-through; color: #aaa; }

.task-actions a { margin-left: 10px; text-decoration: none; padding: 5px 10px; border-radius: 3px; }

.action-complete { background-color: #28a745; color: white; }

.action-delete { background-color: #dc3545; color: white; }

form { margin-top: 20px; display: flex; }

form input[type="text"] { flex-grow: 1; padding: 10px; border: 1px solid #ddd; border-radius: 3px; }

form button { padding: 10px 15px; border: none; background-color: #007bff; color: white; border-radius: 3px; margin-left: 10px; cursor: pointer; }

</style>

</head>

<body>

<h1>Список задач</h1>

<form action="{% url 'task\_list' %}" method="post">

{% csrf\_token %}

<input type="text" name="title" placeholder="Введите новую задачу..." required>

<button type="submit">Добавить</button>

</form>

<div class="task-container">

{% for task in tasks %}

<div class="task {% if task.completed %}completed{% endif %}">

<span>{{ task.title }}</span>

<div class="task-actions">

{% if not task.completed %}

<a href="{% url 'complete\_task' task.id %}" class="action-complete">✅ Выполнить</a>

{% endif %}

<a href="{% url 'delete\_task' task.id %}" class="action-delete">❌ Удалить</a>

</div>

</div>

{% empty %}

<p>Задач пока нет. Добавьте первую!</p>

{% endfor %}

</div>

</body>

</html>

## Приложение 3. Содержимое Dockerfile

FROM python:3.11

ENV PYTHONDONTWRITEBYTECODE=1

ENV PYTHONUNBUFFERED=1

RUN apt-get update && apt-get -y dist-upgrade && apt-get install -y netcat-trad>

WORKDIR /app

COPY requirements.txt .

RUN pip install --upgrade pip && pip install -r requirements.txt

COPY . .

EXPOSE 8000

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]

## Приложение 3. Содержимое docker-compose.yaml

ersion: '3.8'

services:

db:

image: postgres:15

volumes:

- postgres\_data:/var/lib/postgresql/data/

environment:

- POSTGRES\_DB=study

- POSTGRES\_USER=userdb

- POSTGRES\_PASSWORD=qwerty123

web:

image: maksoncheus/taskmaster:1.0

command: >

sh -c "

echo 'Waiting for postgres...' &&

while ! nc -z db 5432; do

sleep 0.1;

done;

echo 'PostgreSQL started';

python manage.py migrate &&

python manage.py runserver 0.0.0.0:8000"

volumes:

- ../todo\_project/:/app/ # Монтируем код для удобства отладки

ports:

- "8000:8000"

environment:

- POSTGRES\_DB=study

- POSTGRES\_USER=userdb

- POSTGRES\_PASSWORD=qwerty123

- POSTGRES\_HOST=db

- POSTGRES\_PORT=5432

depends\_on:

- db

volumes:

postgres\_data: