

Департамент образования города Москвы

Государственное автономное образовательное учреждение  
высшего образования города Москвы  
«Московский городской педагогический университет»

Институт цифрового образования  
Департамент информатики, управления и технологий

**Лабораторная работа 2.1**  
**по дисциплине «Интеграция и развертывание программного**  
**обеспечения с помощью контейнеров»**

**Тема: «Создание Dockerfile и сборка образа»**

Направление подготовки 38.03.05 – бизнес-информатика  
Профиль подготовки «Аналитика данных и эффективное управление»  
(очная форма обучения)

Выполнила:  
Студентка группы АДЭУ-211  
St\_88

Москва  
2025

## **ВВЕДЕНИЕ**

Цель работы: научиться создавать Dockerfile и собирать образы Docker для приложений.

Задачи:

1. Создать Dockerfile для указанного приложения.
2. Собрать образ Docker с использованием созданного Dockerfile.
3. Запустить контейнер из собранного образа и проверить его работоспособность.
4. Выполнить индивидуальное задание.

**Вариант 1.** Создайте Dockerfile для приложения на Node.js, которое выводит "Hello, Node.js!" при доступе к корневому URL.

## ХОД РАБОТЫ

### 1. Выполнение примера: создание Dockerfile для приложения на Flask

#### 1.1. Создание нового каталога проекта, перемещение в него:

```
dev@dev-vm: ~/flask-app
dev@dev-vm:~$ mkdir flask-app
dev@dev-vm:~$ cd flask-app
dev@dev-vm:~/flask-app$
```

#### 1.2. Создание файла app.py:

```
dev@dev-vm:~/flask-app$ touch app.py
dev@dev-vm:~/flask-app$ ls
app.py
dev@dev-vm:~/flask-app$
```

Содержимое файла:

```
app.py
1 from flask import flask
2
3 app = Flask(__name__)
4
5 @app.route('/')
6 def hello():
7     return "Hello, World!"
8
9 if __name__ == '__main__':
10     app.run(host='0.0.0.0', port=5000)
```

#### 1.3. Создание файла requirements.txt:

```
dev@dev-vm:~/flask-app$ touch requirements.txt
dev@dev-vm:~/flask-app$ ls
app.py requirements.txt
dev@dev-vm:~/flask-app$
```

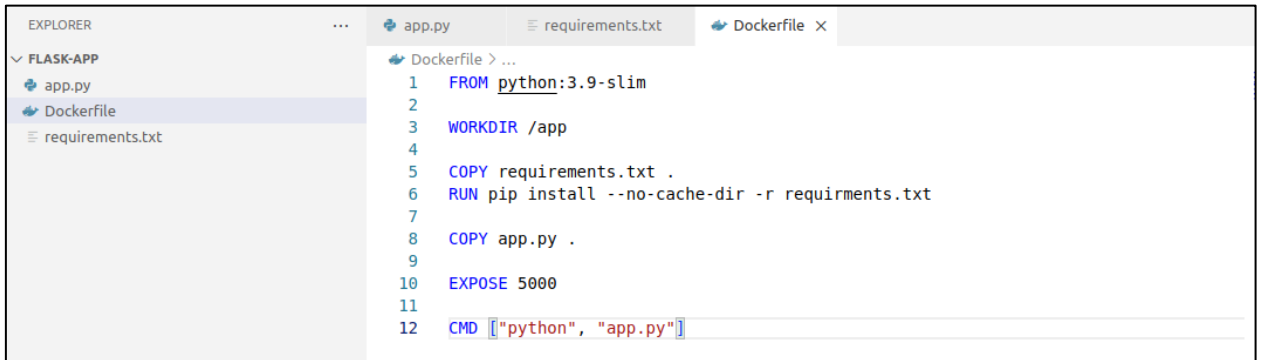
Содержимое файла:

```
requirements.txt
1 Flask
```

#### 1.4. Создание файла Dockerfile:

```
dev@dev-vm:~/flask-app$ touch Dockerfile
dev@dev-vm:~/flask-app$ ls
app.py  Dockerfile  requirements.txt
dev@dev-vm:~/flask-app$
```

#### Содержимое файла:

A screenshot of a code editor window. On the left, the 'EXPLORER' sidebar shows a file tree for 'FLASK-APP' containing 'app.py', 'Dockerfile', and 'requirements.txt'. The 'Dockerfile' is selected. The main editor area shows the content of the Dockerfile with line numbers 1 through 12. The content is as follows:

```
1 FROM python:3.9-slim
2
3 WORKDIR /app
4
5 COPY requirements.txt .
6 RUN pip install --no-cache-dir -r requirements.txt
7
8 COPY app.py .
9
10 EXPOSE 5000
11
12 CMD ["python", "app.py"]
```

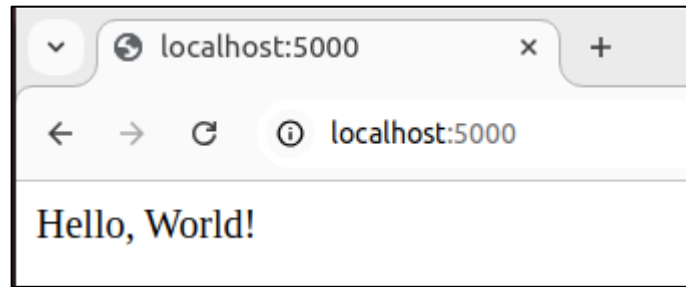
#### 1.5. Сборка образа Docker: выполнение команды для сборки образа в каталоге проекта:

```
dev@dev-vm:~/flask-app$ docker build -t my-flask-app .
2025/03/06 18:50:04 in: []string{}
2025/03/06 18:50:04 Parsed entitlements: []
[+] Building 3.9s (10/10) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.1s
=> => transferring dockerfile: 203B                                0.0s
=> [internal] load metadata for docker.io/library/python:3.9-slim 2.9s
=> [internal] load .dockerignore                                   0.0s
=> => transferring context: 2B                                       0.0s
=> [1/5] FROM docker.io/library/python:3.9-slim@sha256:d1fd807555208707ec95b284 0.0s
=> [internal] load build context                                   0.1s
=> => transferring context: 240B                                       0.0s
=> CACHED [2/5] WORKDIR /app                                         0.0s
=> CACHED [3/5] COPY requirements.txt .                             0.0s
=> CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt 0.0s
=> [5/5] COPY app.py .                                              0.2s
=> exporting to image                                               0.2s
=> => exporting layers                                              0.1s
=> => writing image sha256:706e8ed25e57d741cdb6862a9a6c3130ee56649c0a4748fcec4 0.0s
=> => naming to docker.io/library/my-flask-app                      0.0s
dev@dev-vm:~/flask-app$
```

#### 1.6. Запуск контейнера из собранного образа:

```
dev@dev-vm:~/flask-app$ docker run -d --name my-flask-app -p 5000:5000 my-flask-app
aea85d65ac8072484fe11b739bf35f9b95f305562dc2ab4736937abfae40e4f5
dev@dev-vm:~/flask-app$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
NAME
aea85d65ac80   my-flask-app   "python app.py"         5 seconds ago Up 4 seconds  0.0.0.0:5000->5000/tcp, [::]:5000->5000/tcp
my-flask-app
dev@dev-vm:~/flask-app$
```

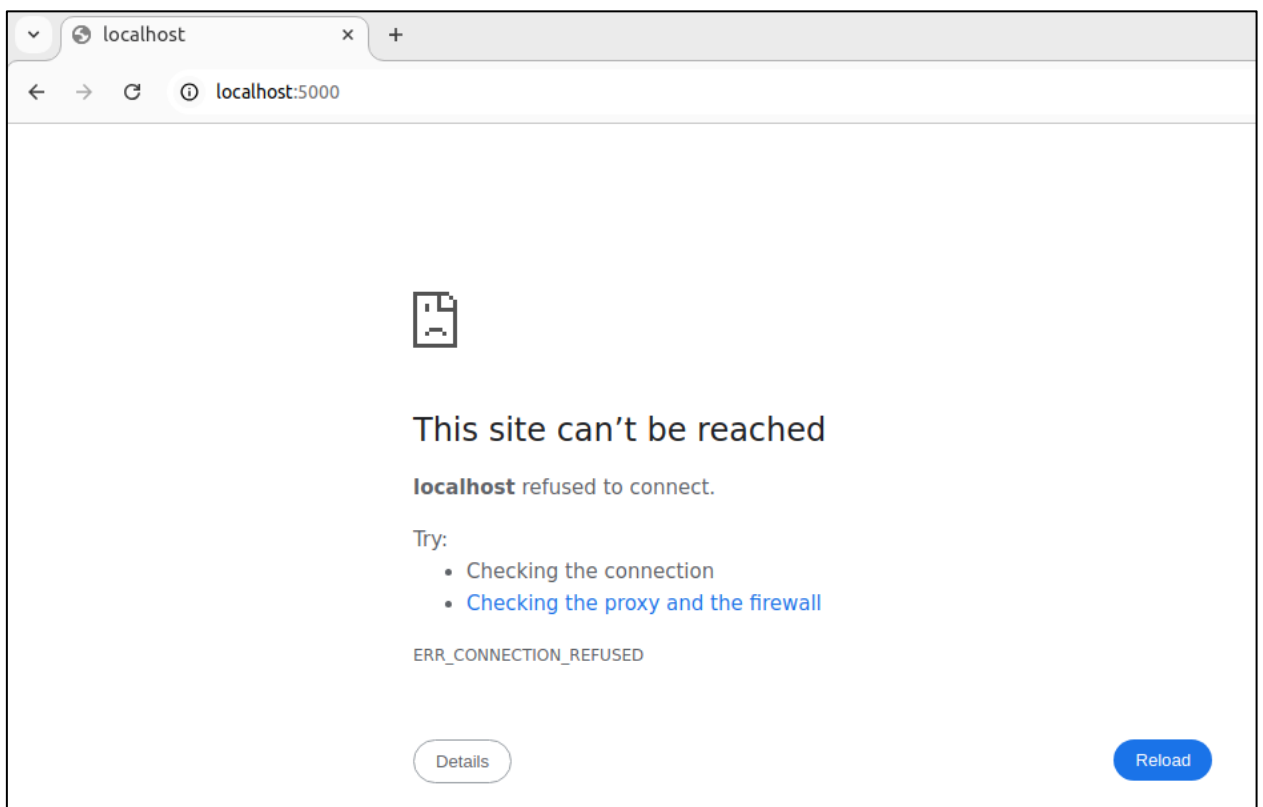
Проверка успешности запуска – успешно:



1.7. Остановка и удаление контейнера:

```
dev@dev-vm:~/flask-app$ docker stop my-flask-app
my-flask-app
dev@dev-vm:~/flask-app$ docker rm my-flask-app
my-flask-app
dev@dev-vm:~/flask-app$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
```

Проверка успешности остановки и удаления контейнера:



2. Выполнение индивидуального задания: **создание Dockerfile для приложения на Node.js, которое выводит "Hello, Node.js!" при доступе к корневому URL.**

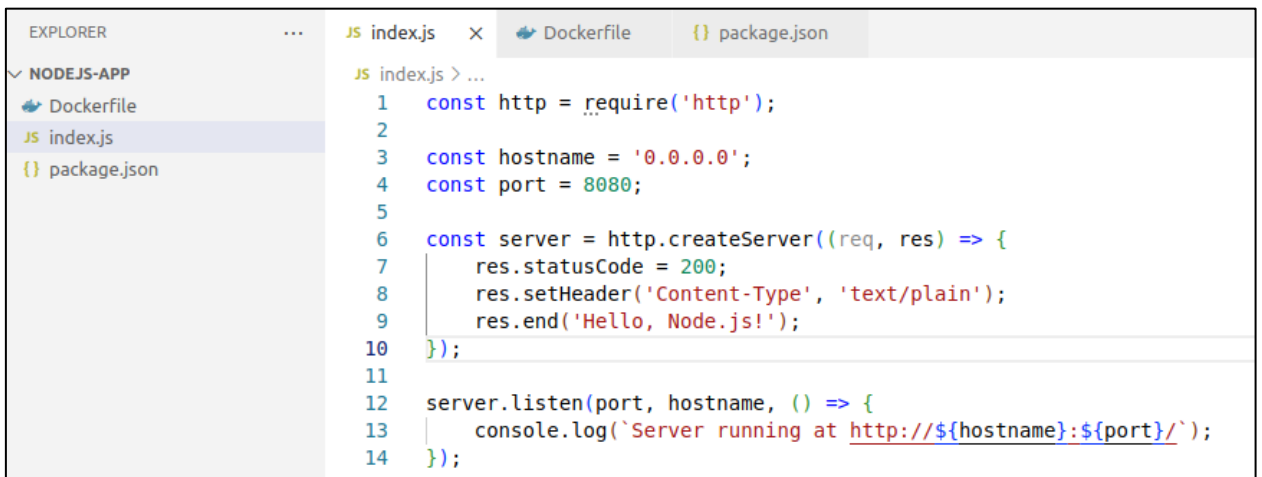
2.1. Создание нового каталога для проекта, переход в него:

```
dev@dev-vm:~$ mkdir nodejs-app
dev@dev-vm:~$ cd nodejs-app
dev@dev-vm:~/nodejs-app$
```

2.2. Создание файла index.js, который будет содержать код Node.js:

```
dev@dev-vm:~/nodejs-app$ touch index.js
dev@dev-vm:~/nodejs-app$ ls
index.js
dev@dev-vm:~/nodejs-app$
```

Содержимое файла:

A screenshot of the Visual Studio Code editor interface. The Explorer sidebar on the left shows a project named 'NODEJS-APP' with files 'Dockerfile', 'index.js', and 'package.json'. The 'index.js' file is selected and its content is displayed in the main editor area. The code in index.js is as follows:

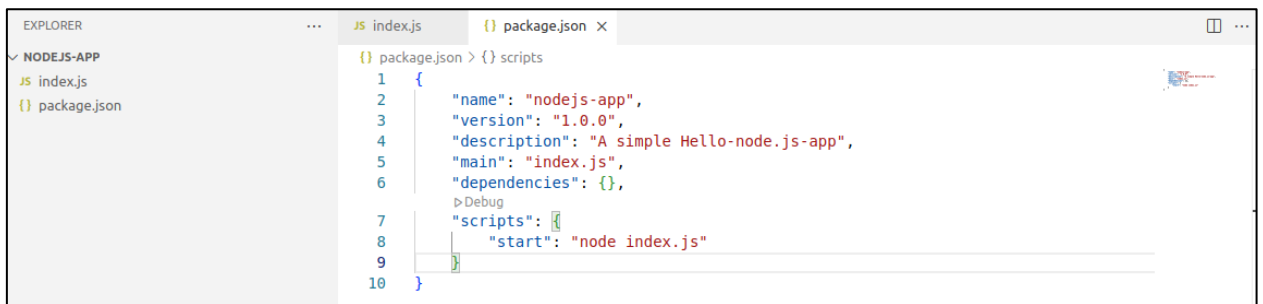
```
1  const http = require('http');
2
3  const hostname = '0.0.0.0';
4  const port = 8080;
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader('Content-Type', 'text/plain');
9    res.end('Hello, Node.js!');
10 });
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at http://${hostname}:${port}/`);
14 });
```

В данном файле: указано, что сервер будет доступен по протоколу http, на локальном хосте, порту 8080; далее идет код для создания нового http-сервера, который будет отсылать запрос «успешно ли запущен сервер?», и при ответе об успешности будет выводить сообщение "Hello, Node.js!".

2.3. Создание файла package.json, который содержит основную необходимую информацию о приложении:

```
dev@dev-vm:~/nodejs-app$ touch package.json
dev@dev-vm:~/nodejs-app$ ls
index.js  package.json
dev@dev-vm:~/nodejs-app$
```

Содержимое файла:



```
EXPLORER
... JS index.js {} package.json x
NODEJS-APP
JS index.js
{} package.json

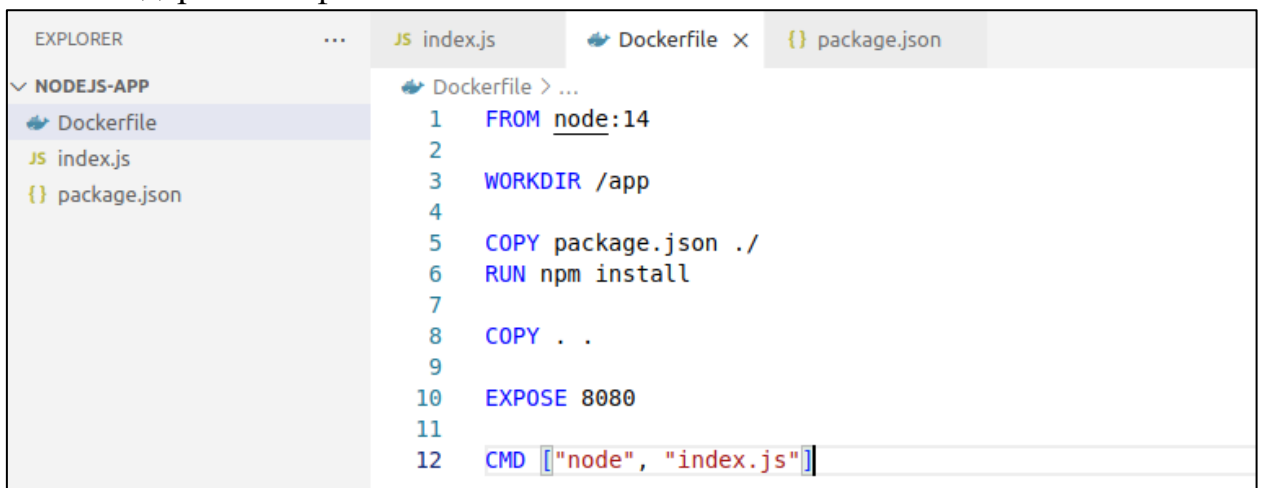
{} package.json > {} scripts
1 {
2   "name": "nodejs-app",
3   "version": "1.0.0",
4   "description": "A simple Hello-node.js-app",
5   "main": "index.js",
6   "dependencies": {},
7   "scripts": {
8     "start": "node index.js"
9   }
10 }
```

В данном файле: указано наименование приложения (должно совпадать с текущей директорией проекта); версия приложения; краткое описание; указание файла js, который будет исполняться.

## 2.4. Создание Dockerfile:

```
dev@dev-vm:~/nodejs-app$ touch Dockerfile
dev@dev-vm:~/nodejs-app$ ls
Dockerfile index.js package.json
dev@dev-vm:~/nodejs-app$
```

Содержимое файла:



```
EXPLORER
... JS index.js Dockerfile x {} package.json
NODEJS-APP
Dockerfile
JS index.js
{} package.json

Dockerfile > ...
1 FROM node:14
2
3 WORKDIR /app
4
5 COPY package.json ./
6 RUN npm install
7
8 COPY . .
9
10 EXPOSE 8080
11
12 CMD ["node", "index.js"]
```

В данном файле: указана необходимая версия Node.js; указано, что /app будет назначена рабочая директория; копирование файла с основной информацией о приложении; установка менеджера пакетов; указание доступного порта 8080.

2.5. Также для того, чтобы приложение запустилось, необходимо предварительно установить Node.js и менеджер пакетов npm:

```
dev@dev-vm:~/nodejs-app$ sudo apt install nodejs
[sudo] password for dev:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  javascript-common libc-ares2 libjs-highlight.js libnode72 nodejs-doc
Suggested packages:
  apache2 | lighttpd | httpd npm
The following NEW packages will be installed:
  javascript-common libc-ares2 libjs-highlight.js libnode72 nodejs nodejs-doc
0 to upgrade, 6 to newly install, 0 to remove and 17 not to upgrade.
```

```
dev@dev-vm:~/nodejs-app$ sudo apt install npm
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
```

Проверка успешности установки:

```
dev@dev-vm:~/nodejs-app$ nodejs -v
v12.22.9
dev@dev-vm:~/nodejs-app$ npm -v
8.5.1
dev@dev-vm:~/nodejs-app$
```

2.6. Итак, все файлы рабочего каталога готовы, все необходимые инструменты установлены. Можно переходить к сборке Docker образа:

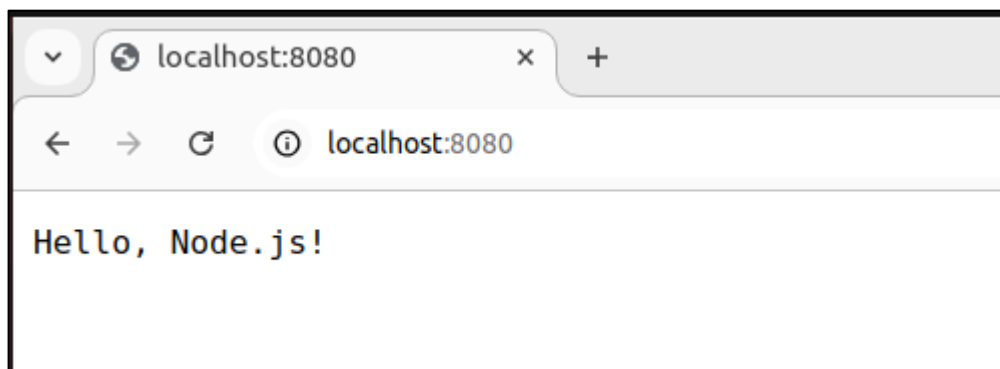
```
dev@dev-vm:~/nodejs-app$ docker build -t nodejs-app .
2025/03/06 20:33:52 in: []string{}
2025/03/06 20:33:52 Parsed entitlements: []
[+] Building 75.2s (10/10) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                0.0s
=> => transferring dockerfile: 150B                                0.0s
=> [internal] load metadata for docker.io/library/node:14         1.7s
=> [internal] load .dockerignore                                    0.0s
=> => transferring context: 2B                                       0.0s
=> [1/5] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a 69.8s
=> => resolve docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a8 0.1s
=> => sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2 776B / 776B 0.0s
=> => sha256:1d12470fa662a2a5cb50378dc8ea228c1735747db410bbef 7.51kB / 7.51kB 0.0s
=> => sha256:2ff1d7c41c74a25258bfa6f0b8adb0a727f84518f55f65ca 50.45MB / 50.45MB 3.3s
=> => sha256:b253aeafeaa7e0671bb60008df01de101a38a045ff7bc656e3 7.86MB / 7.86MB 1.3s
=> => sha256:2cafa3fbb0b6529ee4726b4f599ec27ee557ea3dea70191823 2.21kB / 2.21kB 0.0s
=> => sha256:3d2201bd995cccf12851a50820de03d34a17011dcbb9ac9f 10.00MB / 10.00MB 3.5s
```

2.7. Запуск контейнера из только что собранного образа nodejs-app:

```
dev@dev-vm:~/nodejs-app$ docker run -d --name my-nodejs-app -p 8080:8080 nodejs-app
2b4f808b9ad8c53cf3f43389ed474782bcbaa5a7abca2e3c9b6681c4c668bbb2
dev@dev-vm:~/nodejs-app$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
TS
2b4f808b9ad8   nodejs-app    "docker-entrypoint.s..." 8 seconds ago  Up 7 seconds  0.0
.0.0:8080->8080/tcp, [::]:8080->8080/tcp
my-nodejs-app
dev@dev-vm:~/nodejs-app$
```



2.8. Проверка доступности сервера по адресу `http://localhost:8080`:



2.9. Остановка и удаление контейнера:

```
dev@dev-vm:~/nodejs-app$ docker stop 2b4f808b9ad8
2b4f808b9ad8
dev@dev-vm:~/nodejs-app$ docker rm 2b4f808b9ad8
2b4f808b9ad8
dev@dev-vm:~/nodejs-app$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS   NAMES
dev@dev-vm:~/nodejs-app$
```

## **ВЫВОДЫ**

В ходе выполнения данной лабораторной работы были выполнены все поставленные задачи:

1. по варианту из примера был создан рабочий каталог приложения на Flask, содержащий в себе Dockerfile, далее на его основе был создан образ Docker и успешно запущен соответствующий контейнер Docker;
2. по индивидуальному заданию также был создан проект приложения на Node.js, которое выводит "Hello, Node.js!" при доступе к корневому URL, на основе проекта был собран образ, а далее успешно запущен контейнер.

Таким образом была достигнута цель работы - научиться создавать Dockerfile и собирать образы Docker для приложений, а также реализовано индивидуальное задание.

## ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

### 1. Что такое Dockerfile и для чего он используется?

Dockerfile — это текстовый файл, содержащий ряд инструкций по созданию образа Docker. Он определяет базовый образ, код приложения, зависимости и любые другие конфигурации, необходимые для создания образа.

### 2. Какие основные инструкции используются в Dockerfile?

- FROM: указывает базовый образ для использования.
- WORKDIR: устанавливает рабочую директорию внутри контейнера.
- COPY: копирует файлы из локальной файловой системы в образ.
- RUN: выполняет команды в образе во время процесса сборки.
- EXPOSE: указывает порты, которые контейнер будет слушать.
- CMD: указывает команду для запуска при запуске контейнера из образа.

### 3. Как выполняется сборка образа Docker с использованием Dockerfile?

Сборка образа выполняется после того, как готовы Dockerfile и код приложения. Команда для сборки запускается в рабочем каталоге:

```
docker build -t my-python-app .
```

Флаг `-t` помечает образ именем (`my-python-app`);

`.` в конце указывает контекст сборки (текущий каталог).

Для проверки образа можно воспользоваться командой, выводящей весь перечень доступных образов:

```
docker images
```

### 4. Как запустить контейнер из собранного образа?

Для запуска контейнера из собранного образа необходимо выполнить команду:

```
docker run -p 5000:5000 my-python-app
```

Флаг `-p` сопоставляет порт 5000 на вашем хост-компьютере с портом 5000 в контейнере.

## **5. Каковы преимущества использования Dockerfile для создания образов Docker?**

- Автоматизация — исключение ручных настроек.
- Воспроизводимость — идентичное окружение на всех этапах.
- Версионность — изменения в Dockerfile можно отслеживать через Git.
- Легковесность — образы основаны на слоях, что экономит ресурсы.
- Изоляция — приложение работает в независимом окружении.