**Лабораторная работа №4. Docker**

**Задание**

Шаги

1 часть.

1.1. Установить Docker.

1.2. Создать Dockerfile > Image > Container для Node.js веб-приложения.

1.3. Загрузить Docker-образ в Docker Hub.

2 часть.

2.1. (simple-script-1) Создать и запустить Docker контейнер с python-приложением.

2.2. (mount-directory-2) Создать и запустить Docker контейнер с установкой каталога.

2.3. (ports-3) Создать и запустить Docker контейнер с перенаправлением портов.

2.4. (compose-4) Создать и запустить Docker контейнер с docker-compose.

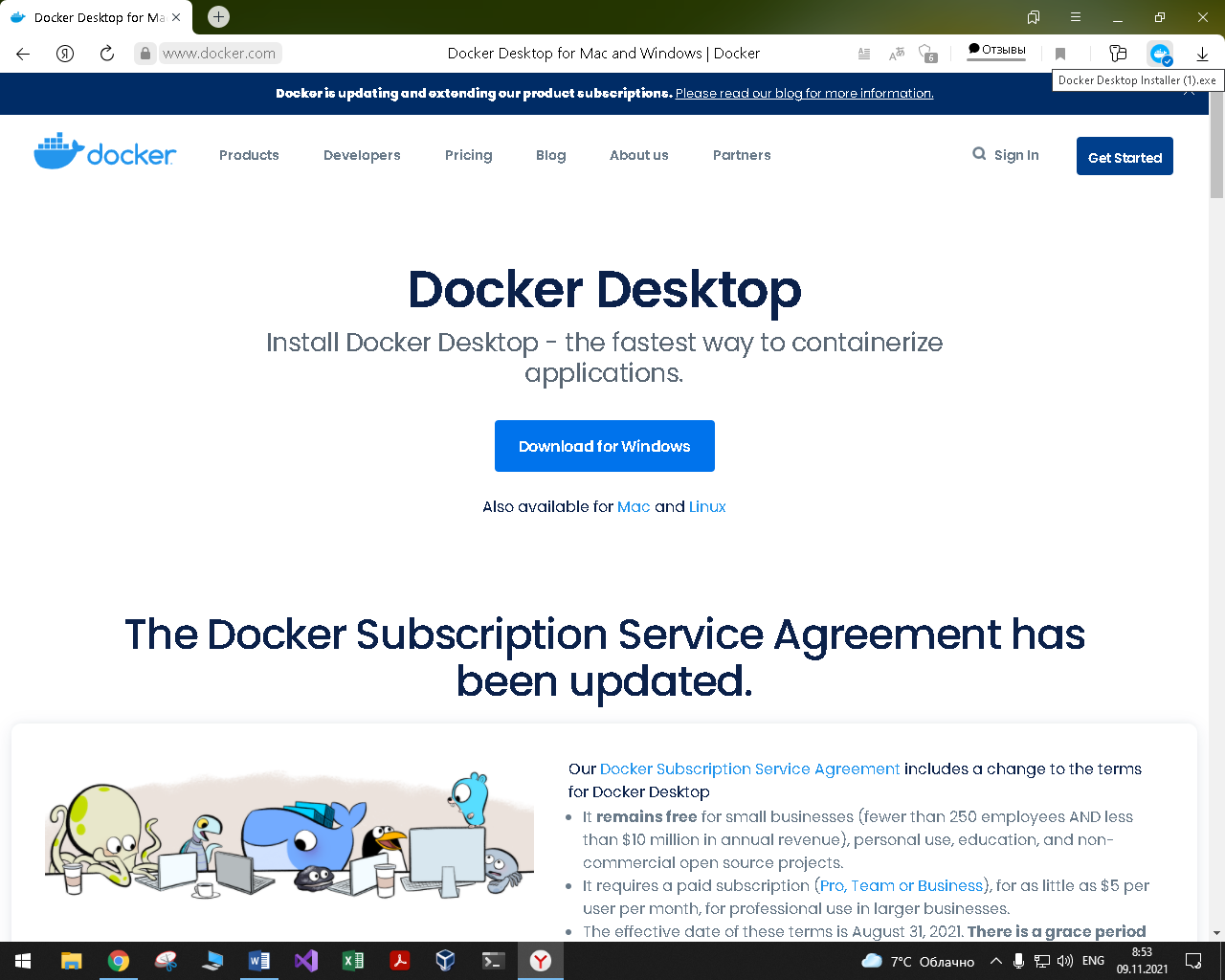
2.5. (proxy-system-5) Создать и запустить Docker контейнер с распределенной нагрузкой.

2.6. (docker-swarm-6) Создать и запустить Docker-Swarm конфигурацию.

ЧАСТЬ 1

Шаг 1. Скачать и установить Docker Desktop.

1.1 Клиент программы Docker Desktop доступен на сайте по ссылке https://www.docker.com/products/docker-desktop

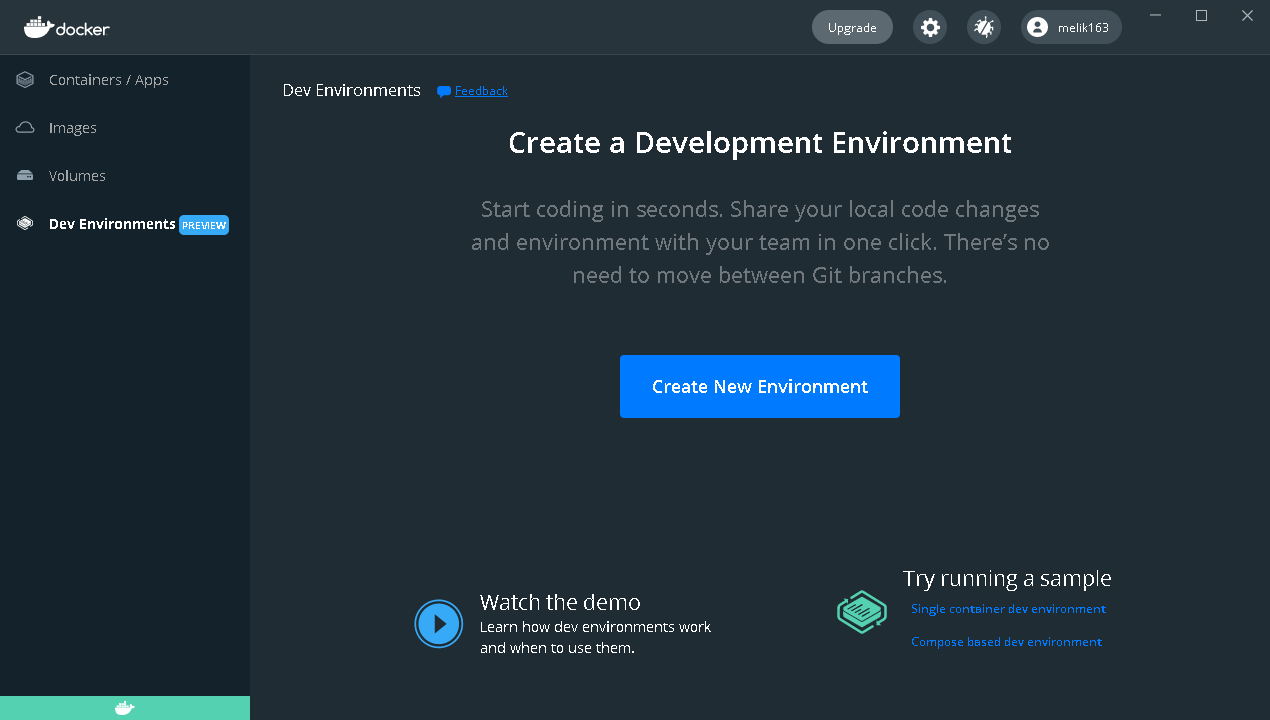


1.2 На данном ПК потребуется произвести настройку для запуска программы Docker Desktop, более подробно об этом можно почитать на сайте по ссылке https://docs.microsoft.com/en-us/windows/wsl/install-manual.

$ dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart

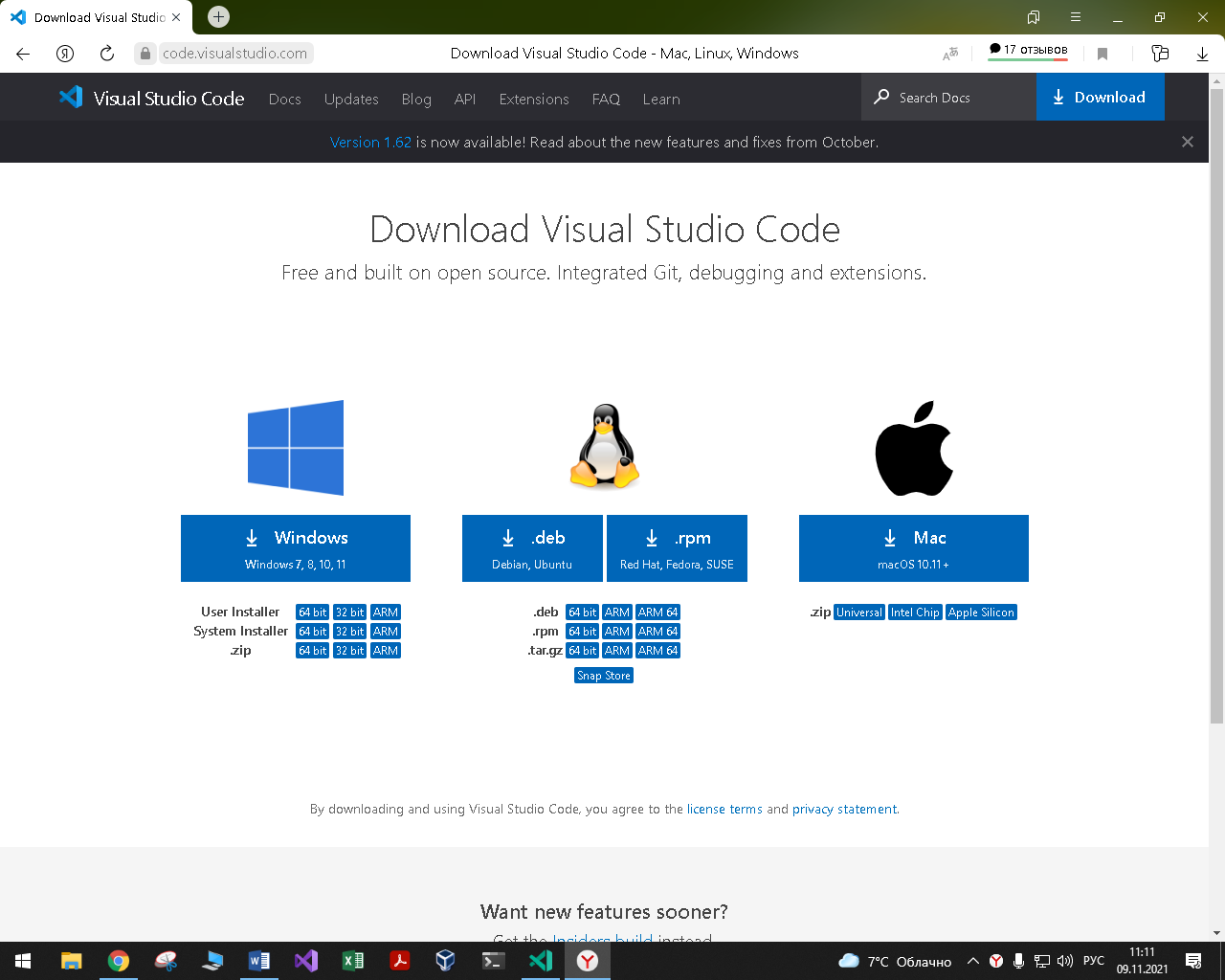
$ dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart

1.3 После чего потребуется перезапуск машины. Теперь Docker Desktop установлен и запущен.



Шаг 2. Создание Dockerfile.

2.1 Для работы с Docker воспользуемся средой разработки Visual Studio Code, скачать которую можно на сайте по ссылке https://code.visualstudio.com/download



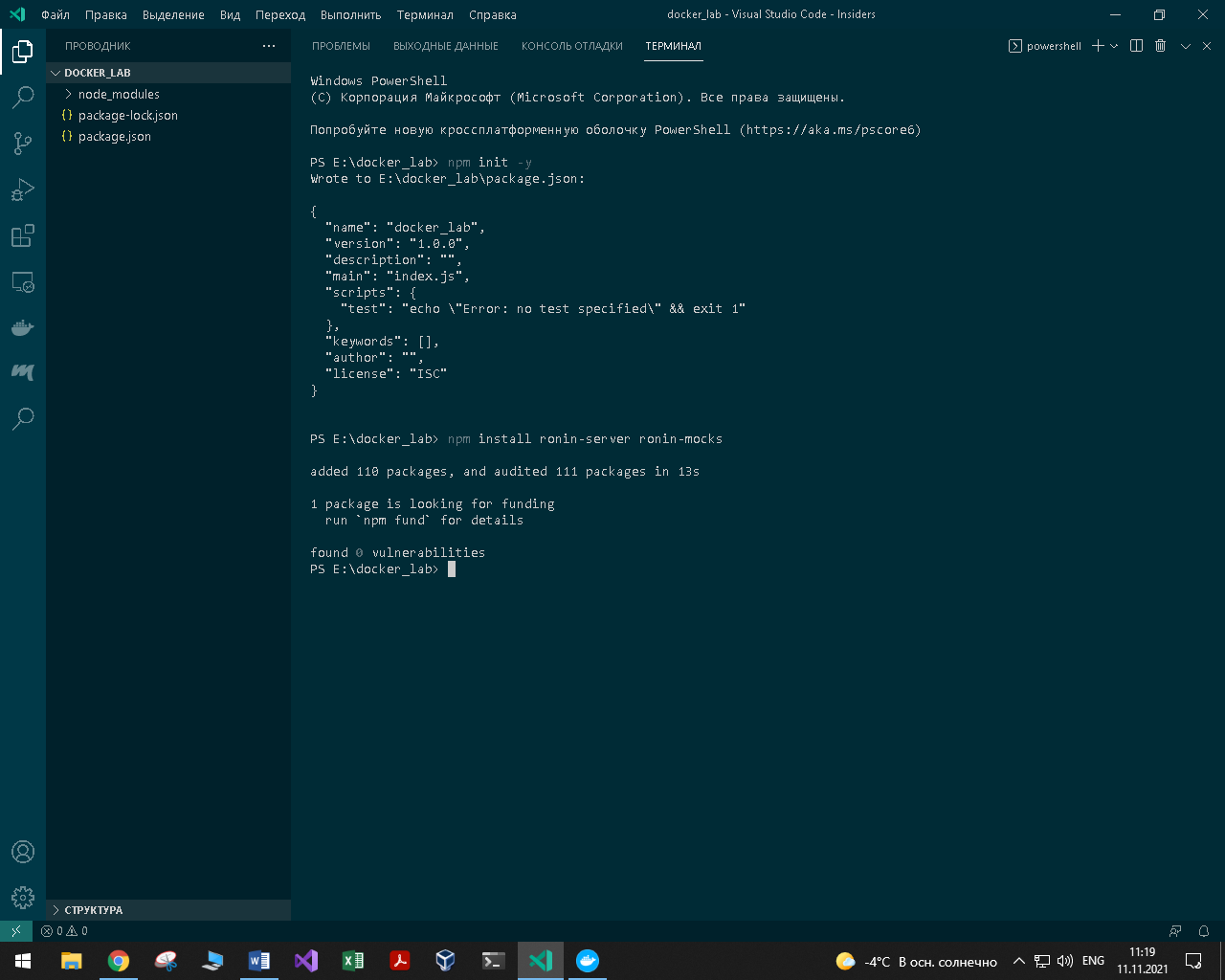
2.2. Перед началом работы проведем проверку работоспособности Docker-контейнер, для этого воспользуемся приложением с сайта по ссылке: <https://docs.docker.com/language/nodejs/build-images/#sample-application>.

В терминале VS Code выполним следующие команды:

$ cd E://Lab-3-Docker

$ npm init -y

$ npm install ronin-server ronin-mocks



Создадим файл server.js

В который поместим следующий код:

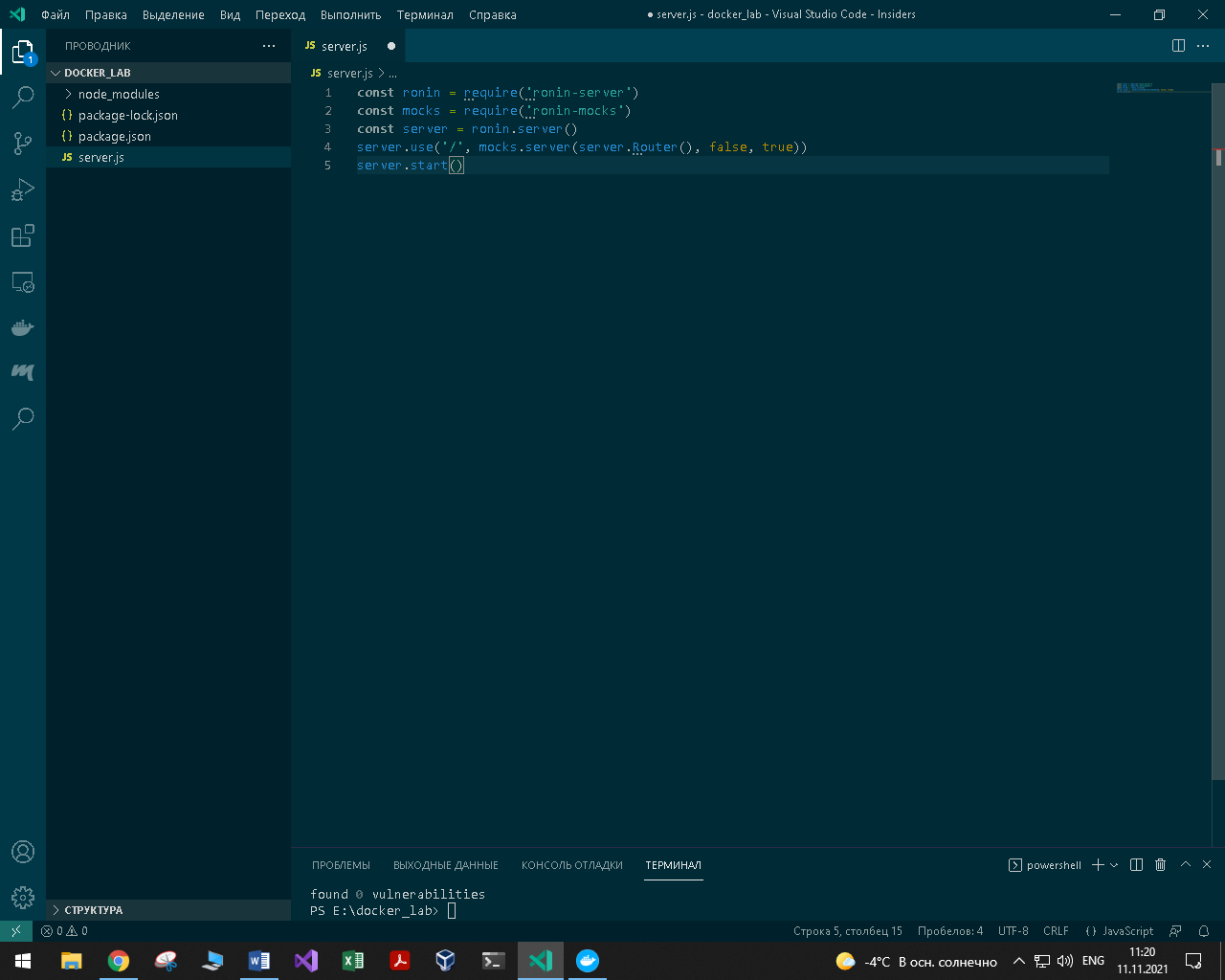
const ronin = require('ronin-server')

const mocks = require('ronin-mocks')

const server = ronin.server()

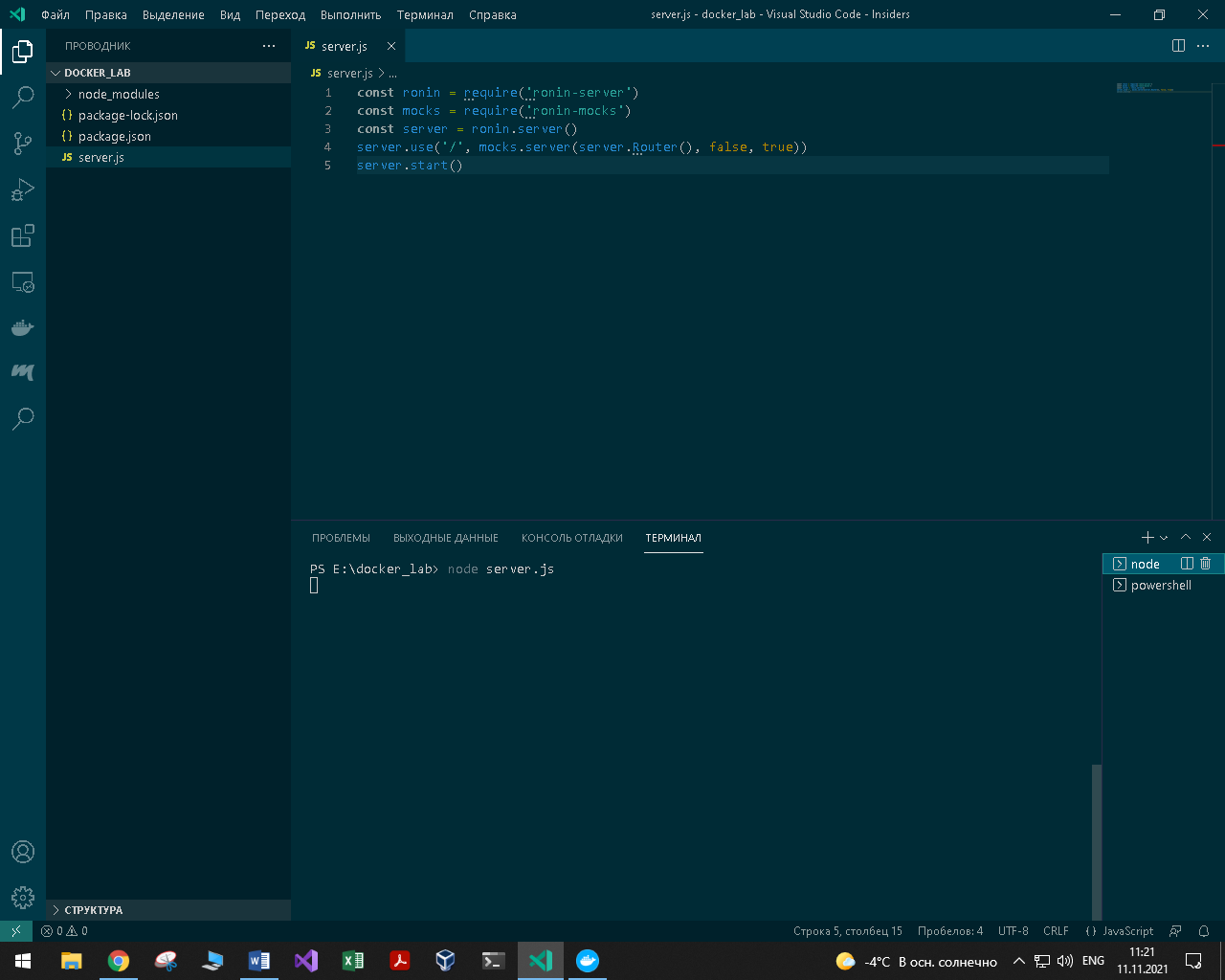
server.use('/', mocks.server(server.Router(), false, true))

server.start()



После чего запустим сервер командой

$ node server.js

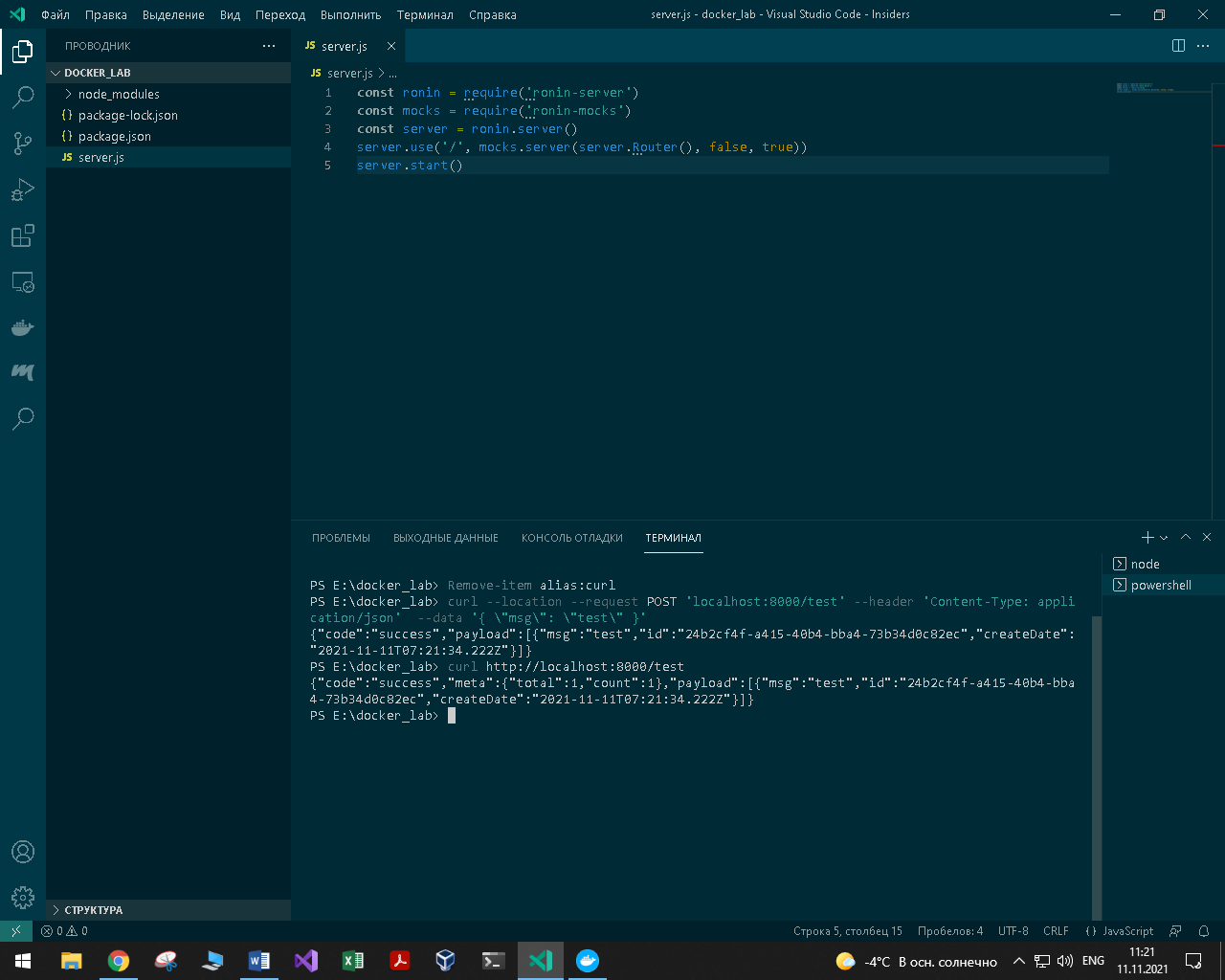


В соседнем терминале выполним следующие команды:

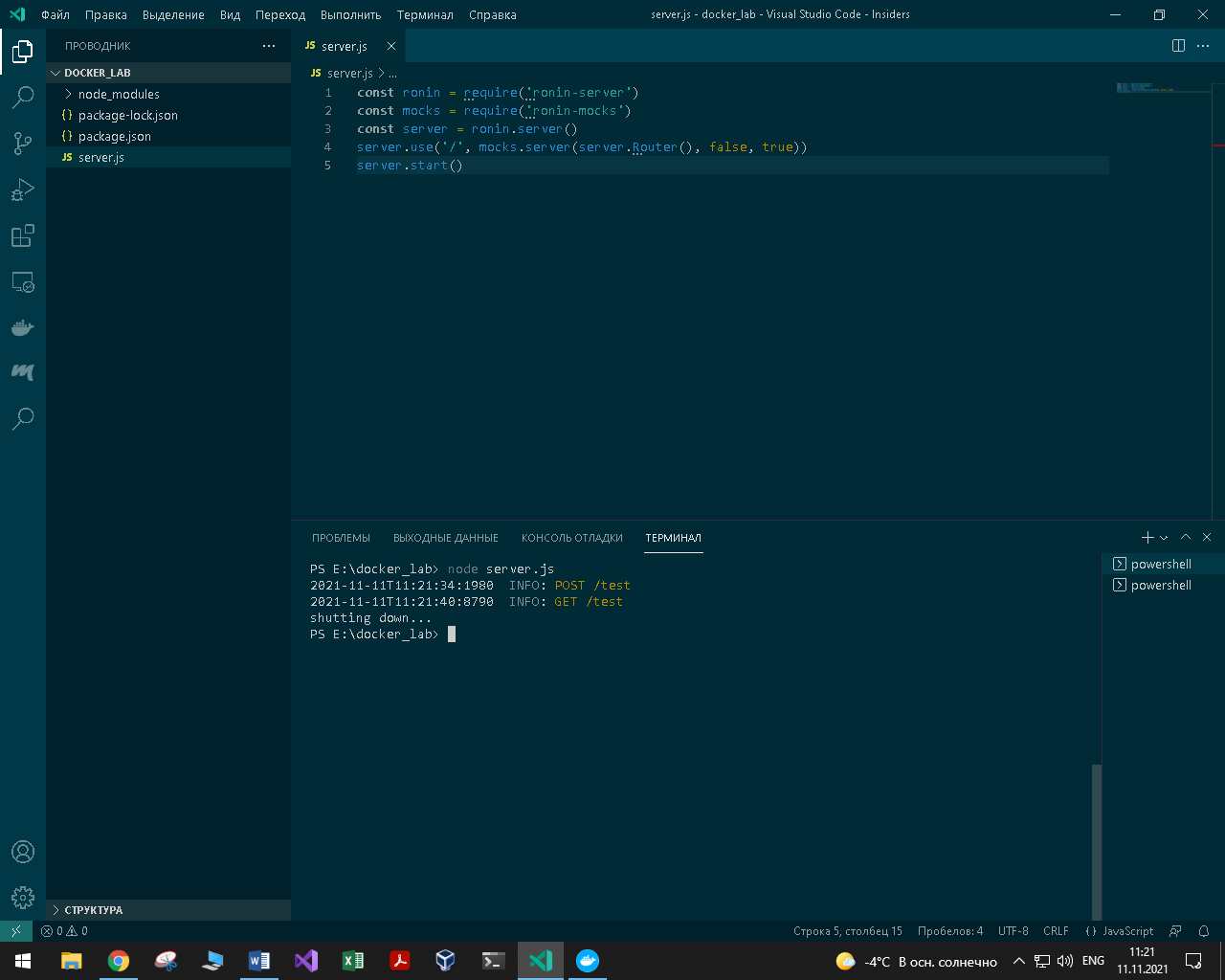
$ Remove-item alias:curl

$ curl --location --request POST 'localhost:8000/test' --header 'Content-Type: application/json' --data '{ \"msg\": \"test\" }'

$ curl http://localhost:8000/test



В результате сервер должен дать следующие ответы:



2.3 Создание Dockerfile

В VS Code создадим новый файл, в который поместим следующий код:

FROM node:12.18.1

ENV NODE\_ENV=production

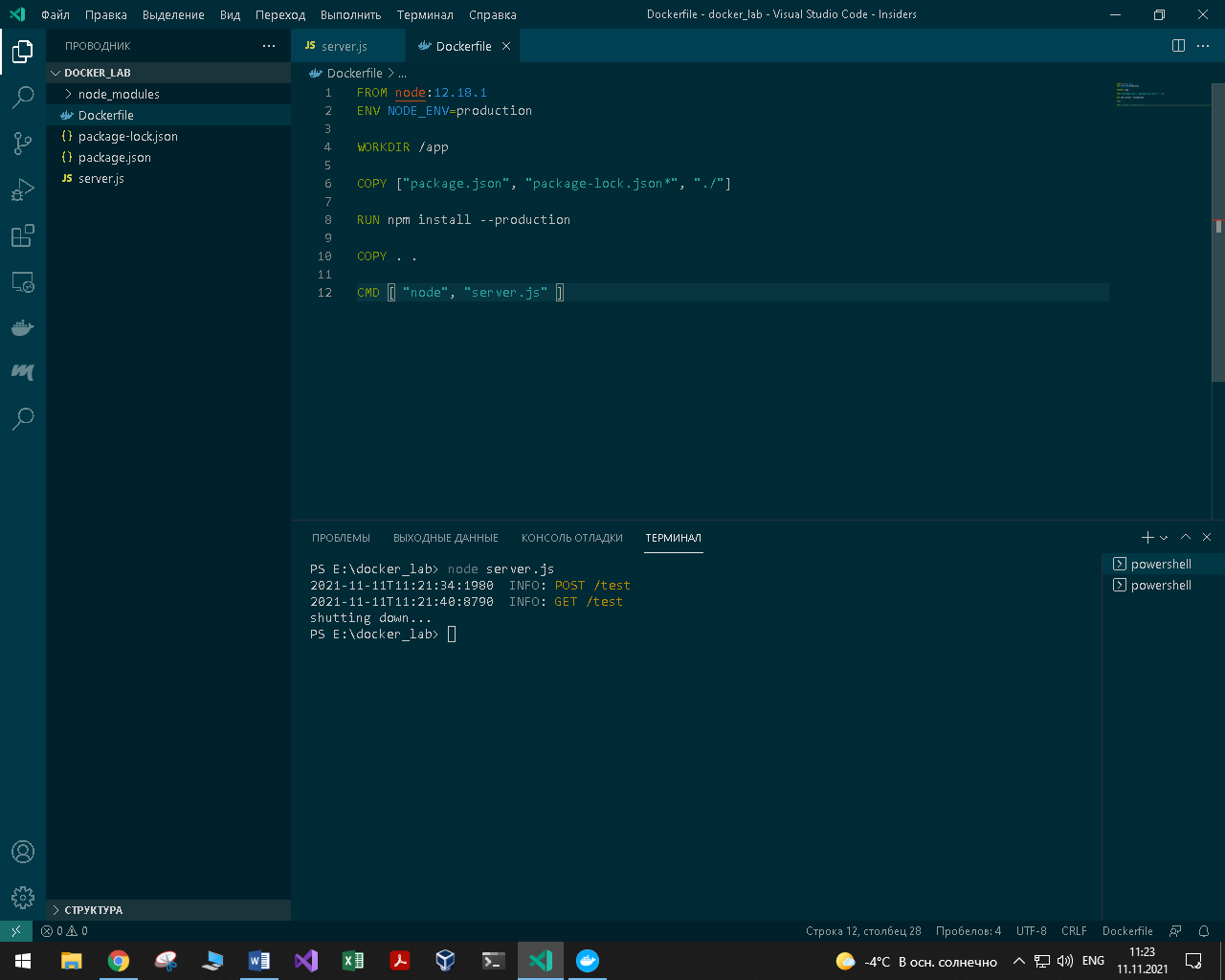
WORKDIR /app

COPY ["package.json", "package-lock.json\*", "./"]

RUN npm install --production

COPY . .

CMD [ "node", "server.js" ]

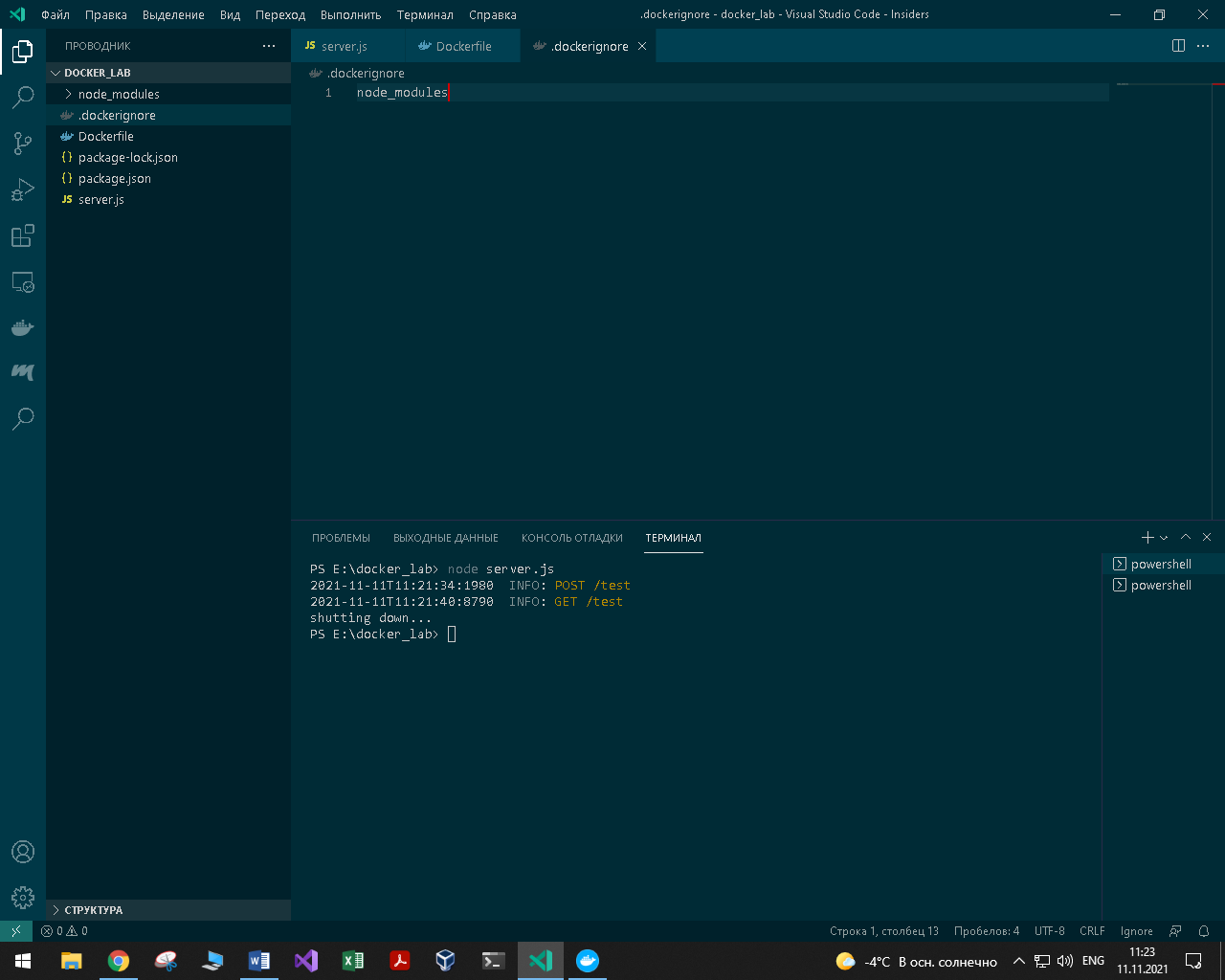


Сохраним данный файл с названием Dockerfile (без расширения), в необходимой папке.

2.4 Создание .dockerignore

Создадим новый файл с название .dockerignore, в который поместим следующий код:

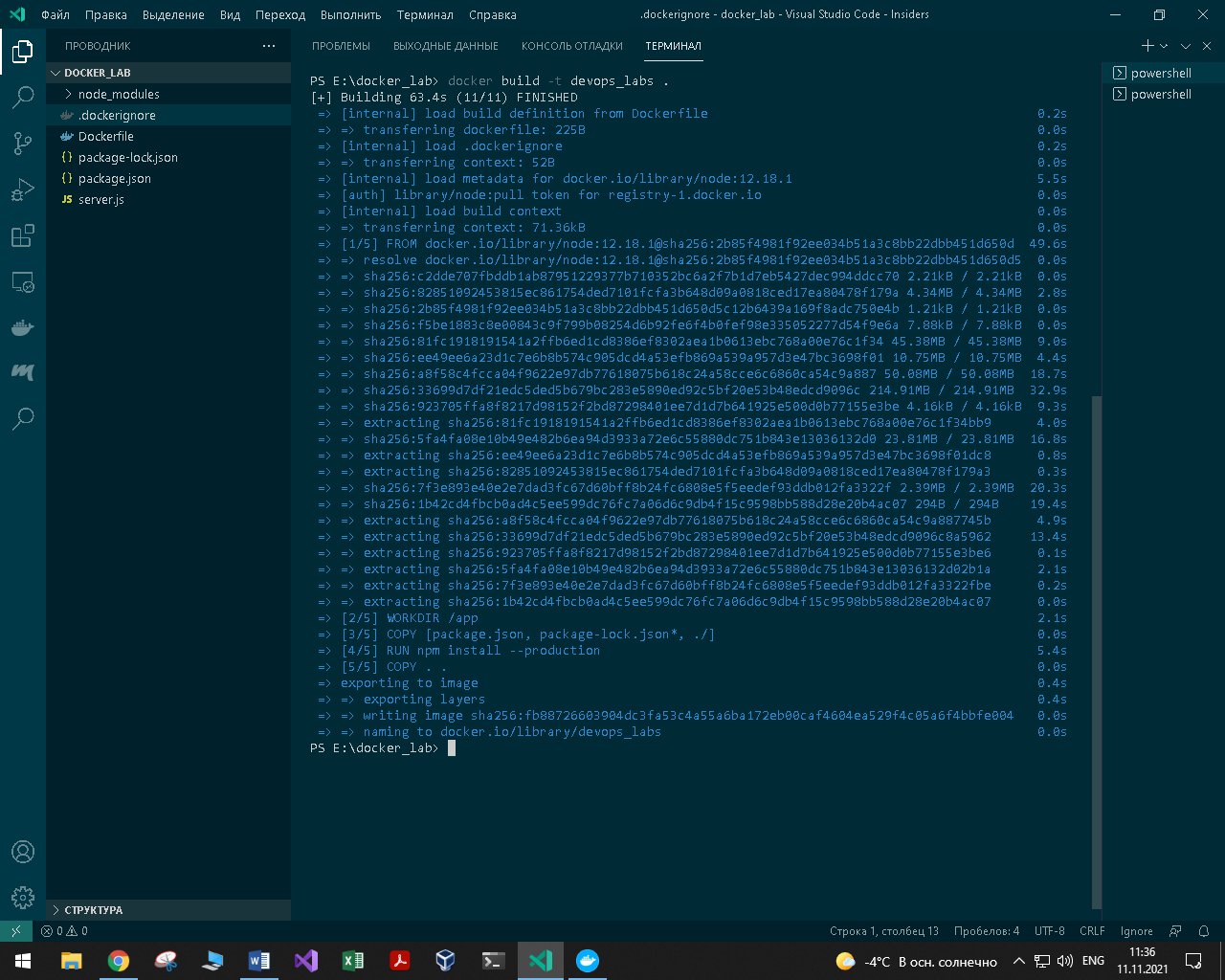
node\_modules



Шаг 3. Создание Docker-образа

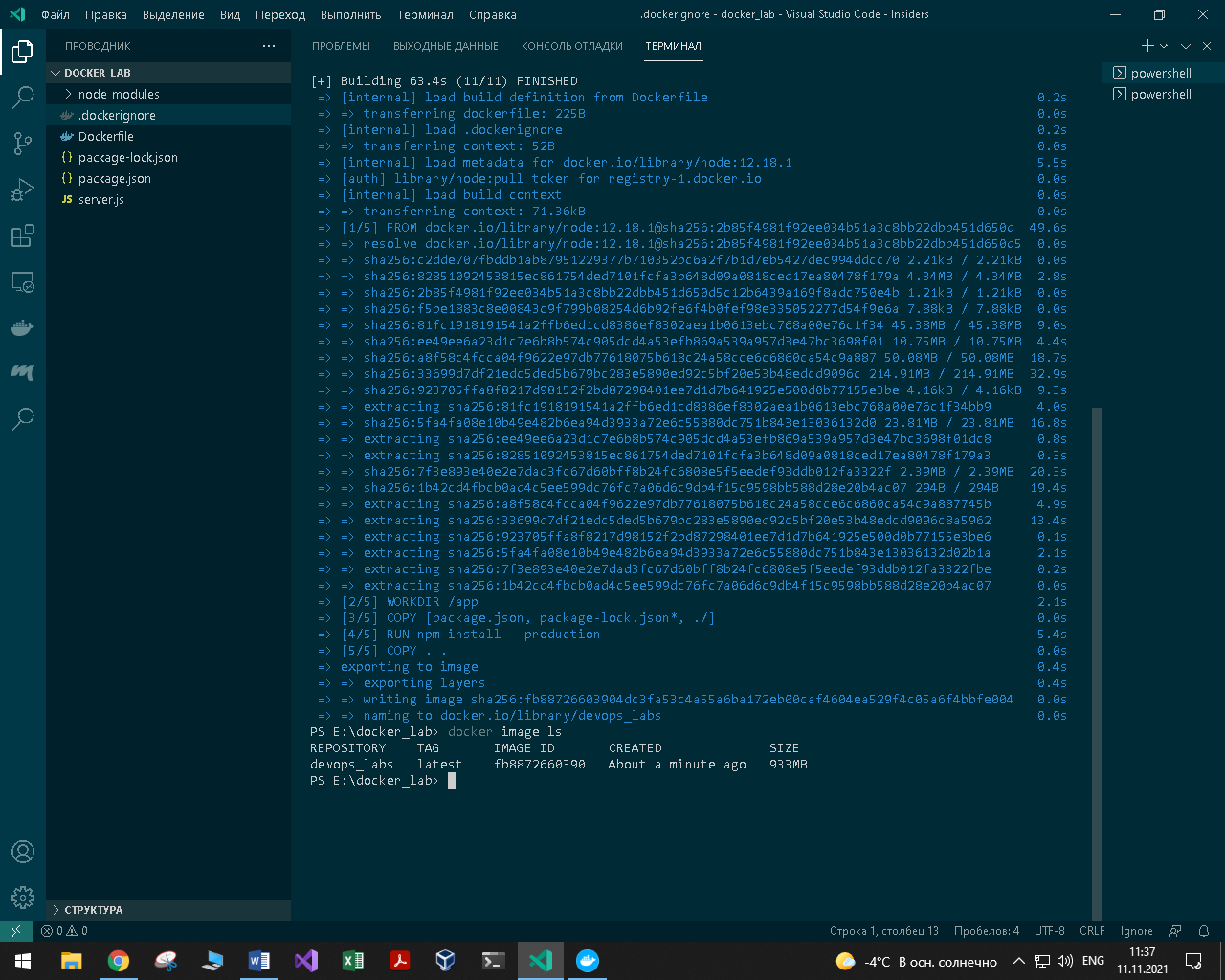
Откроем терминал и перейдем в папку, где был сохранен Dockerfile, после чего создадим Docker –образ, при помощи команды:

$ docker build -t devops\_labs .



Просмотрим созданный образ Docker, при помощи команды:

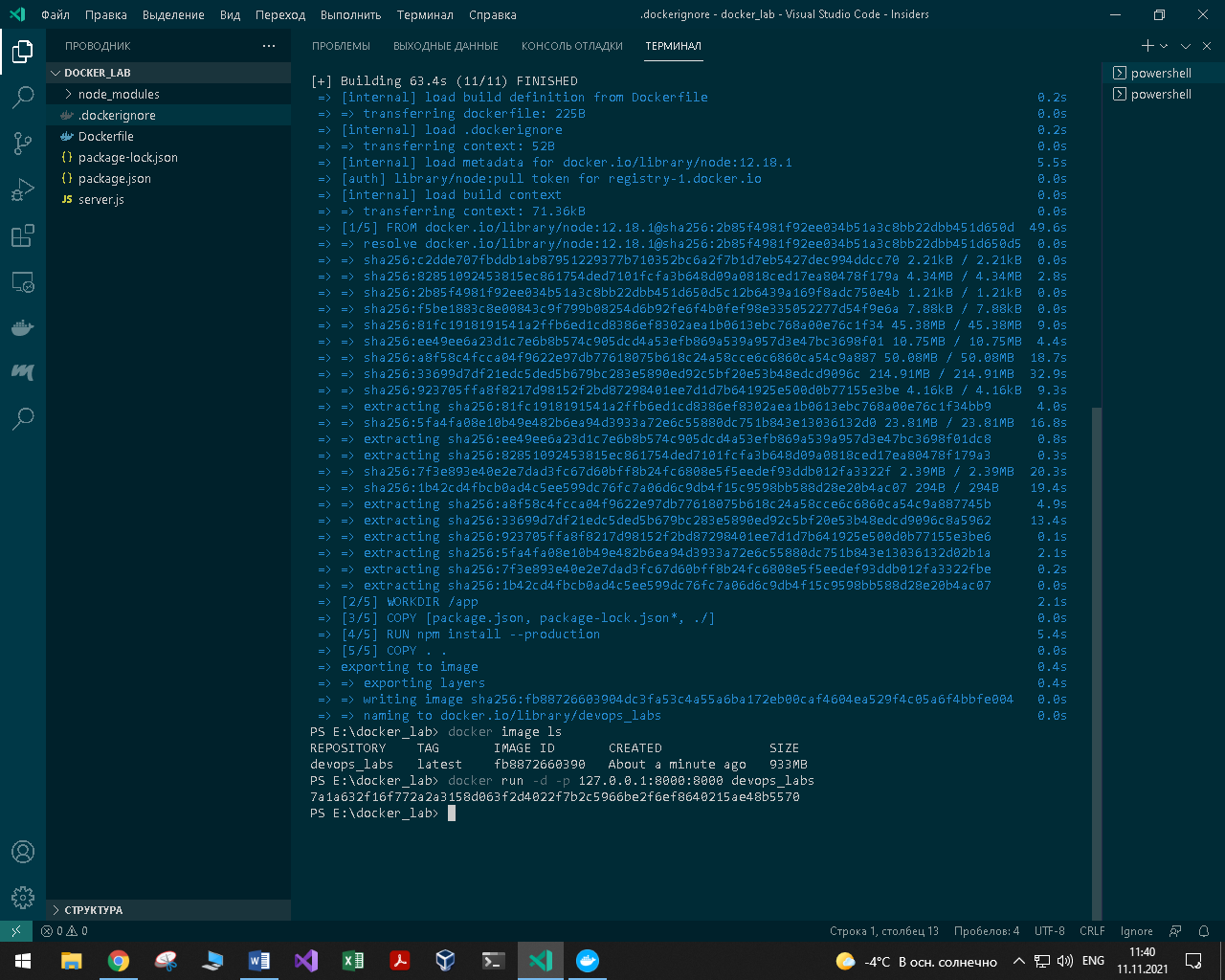
$ docker image ls



3.1 Запуск Docker-контейнера.

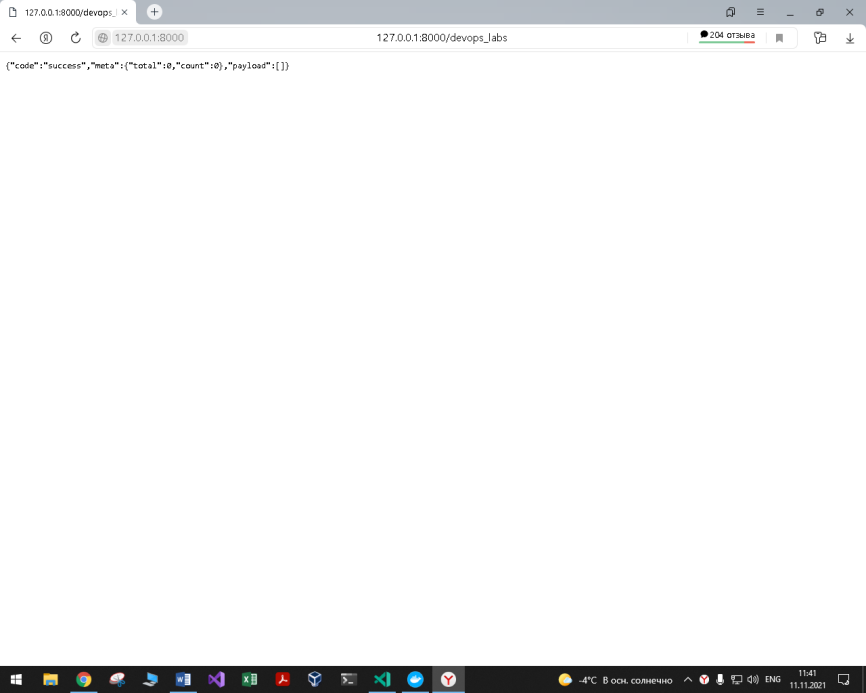
Для запуска Docker-контейнера необходимо выполнить следующую команду:

$ docker run -d -p 127.0.0.1:8000:8000 devops\_labs



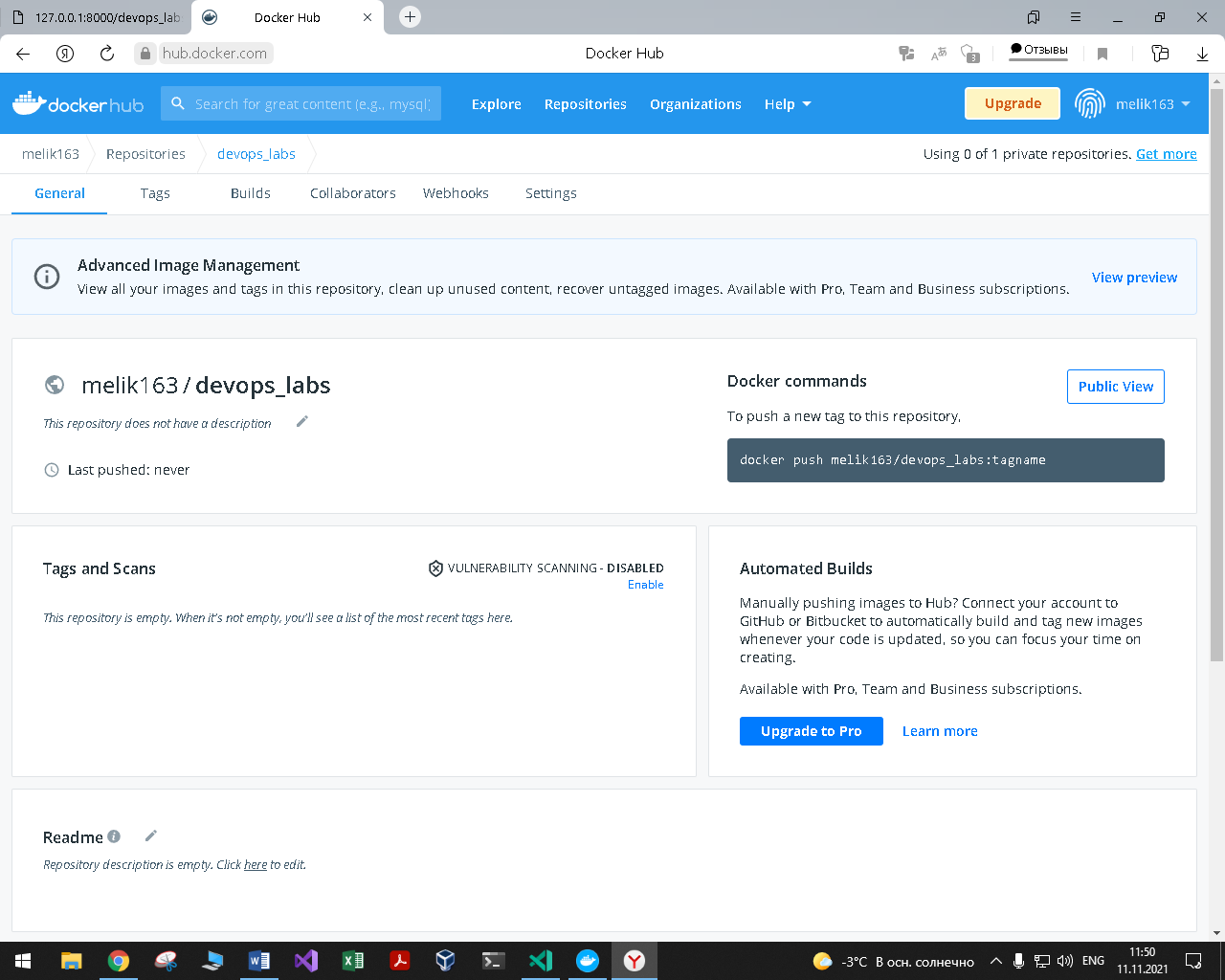
Проверим работоспособность построенного и запущенного Docker-контейнера. Для этого перейдем в браузер и укажем адрес, переданный в качестве параметра при выполнении команды построения Docker-образа, а также через слеш допишем название Docker-контейнера.

http://127.0.0.1:8000/devops\_labs



Шаг 5. Отправка Docker-образа в Docker Hub

5.1 Создание репозитория в Docker Hub.

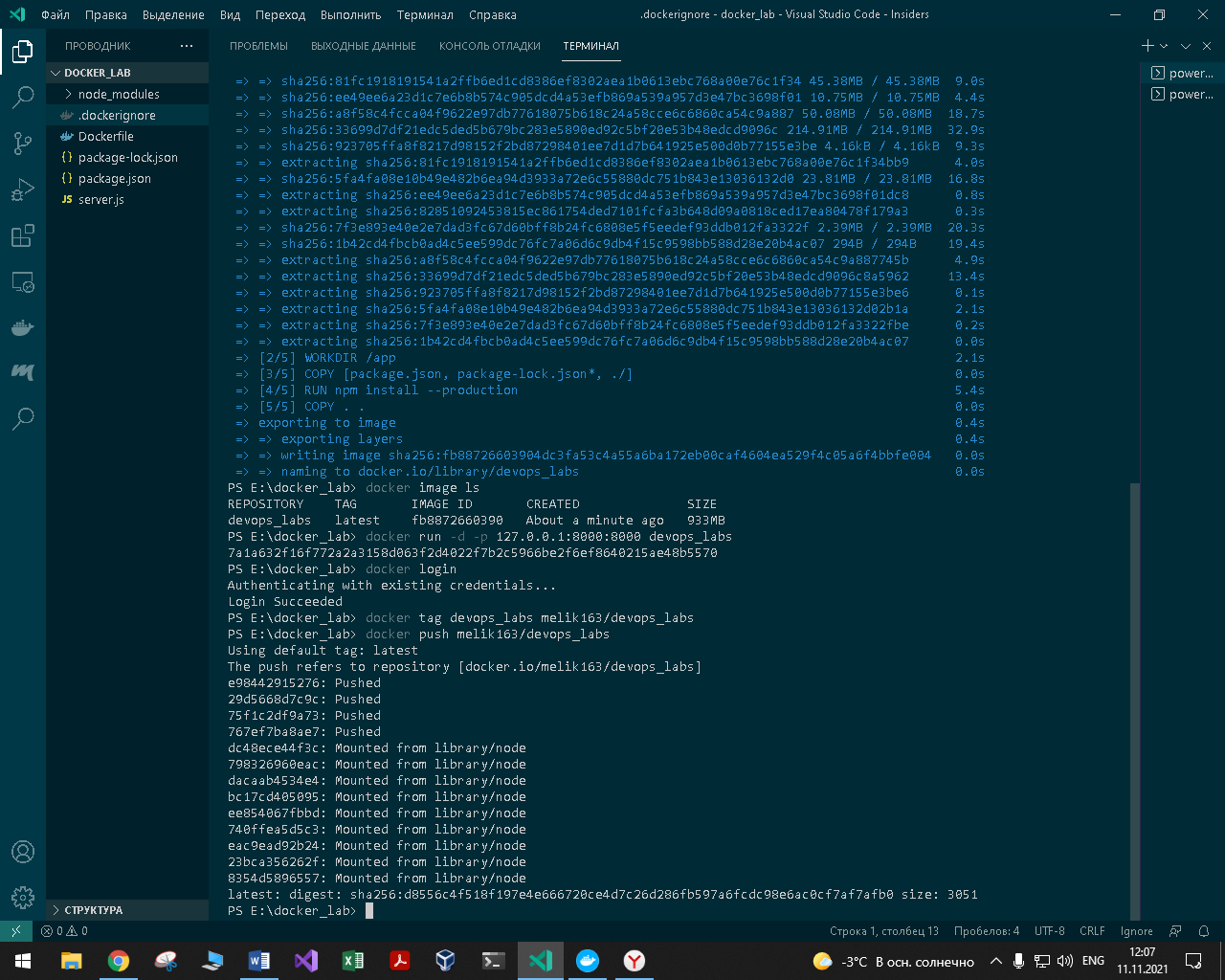


5.2 Загрузка на Docker Hub

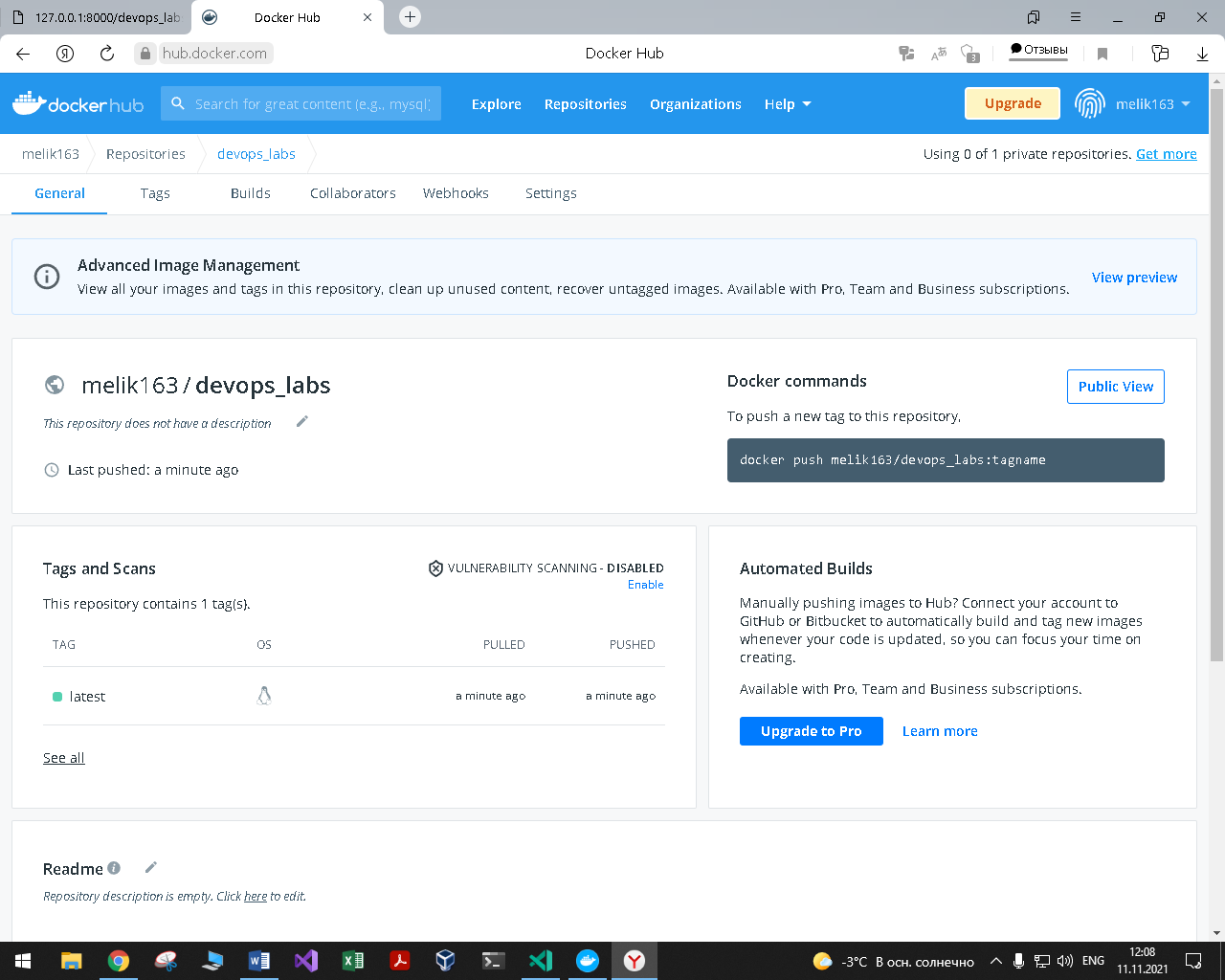
$ docker login

$ docker tag devops\_labs melik163/devops\_labs

$ docker push melik163/devops\_labs



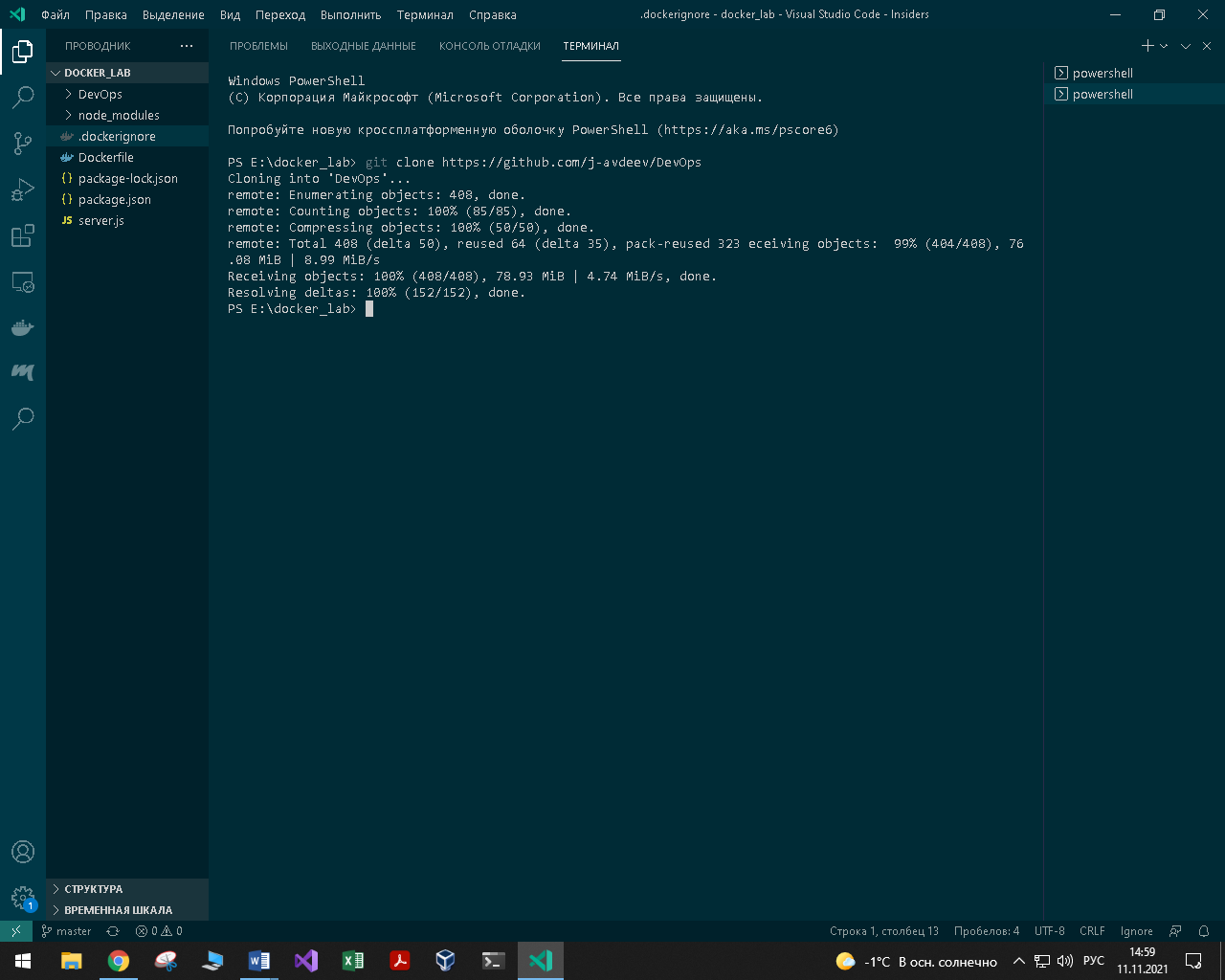
Перейдем в браузер и проверим выгрузку Docker-контейнера на Docker Hub.



ЧАСТЬ 2

Шаг 1. Подготовка данных.

Клонируем репозиторий с сайта GitHub по ссылке https://github.com/j-avdeev/DevOps.

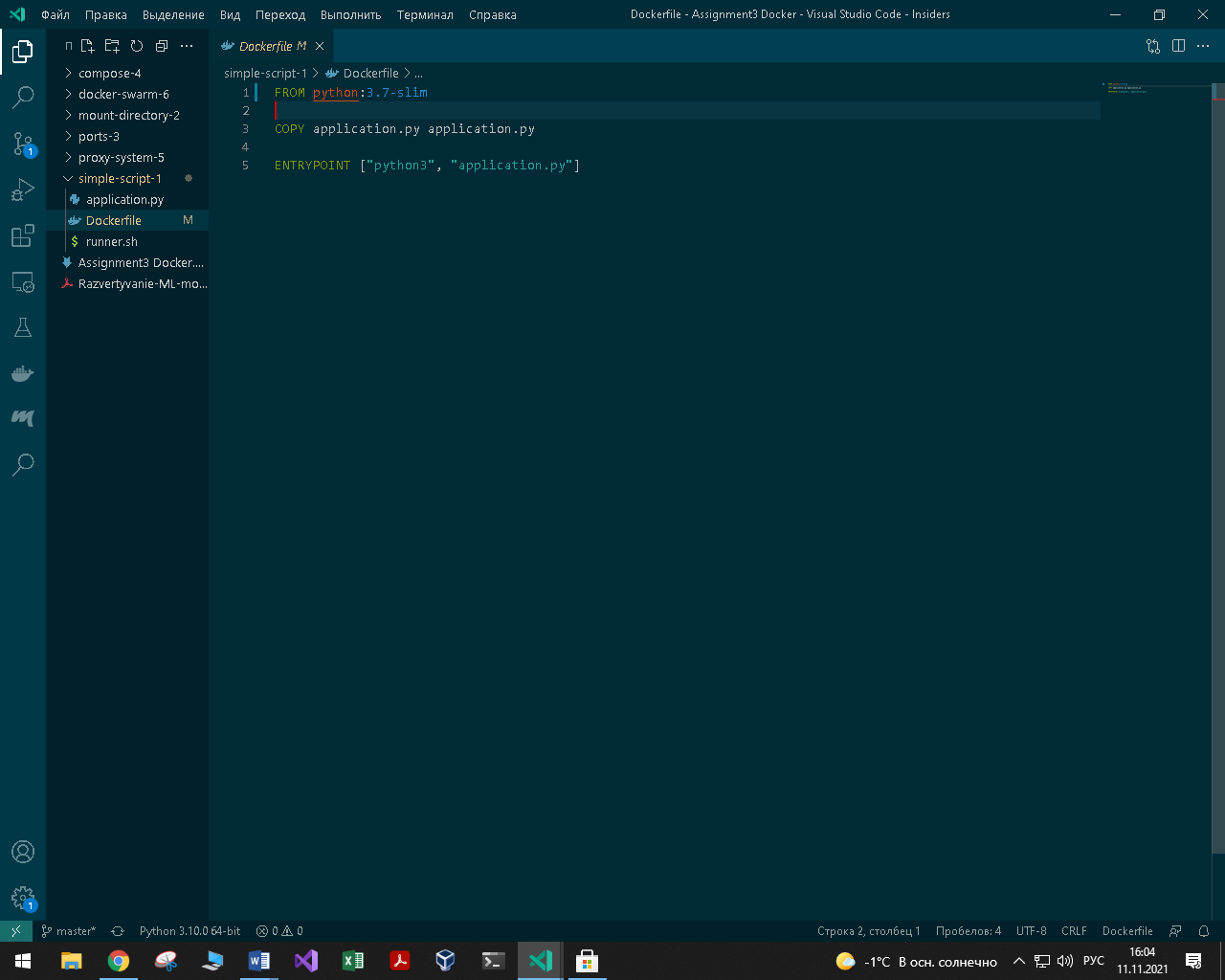


Шаг 2. Создать и запустить Docker-контейнер с python-приложением.

(Directory simple-script-1).

2.1 Корректировка Dockerfile.

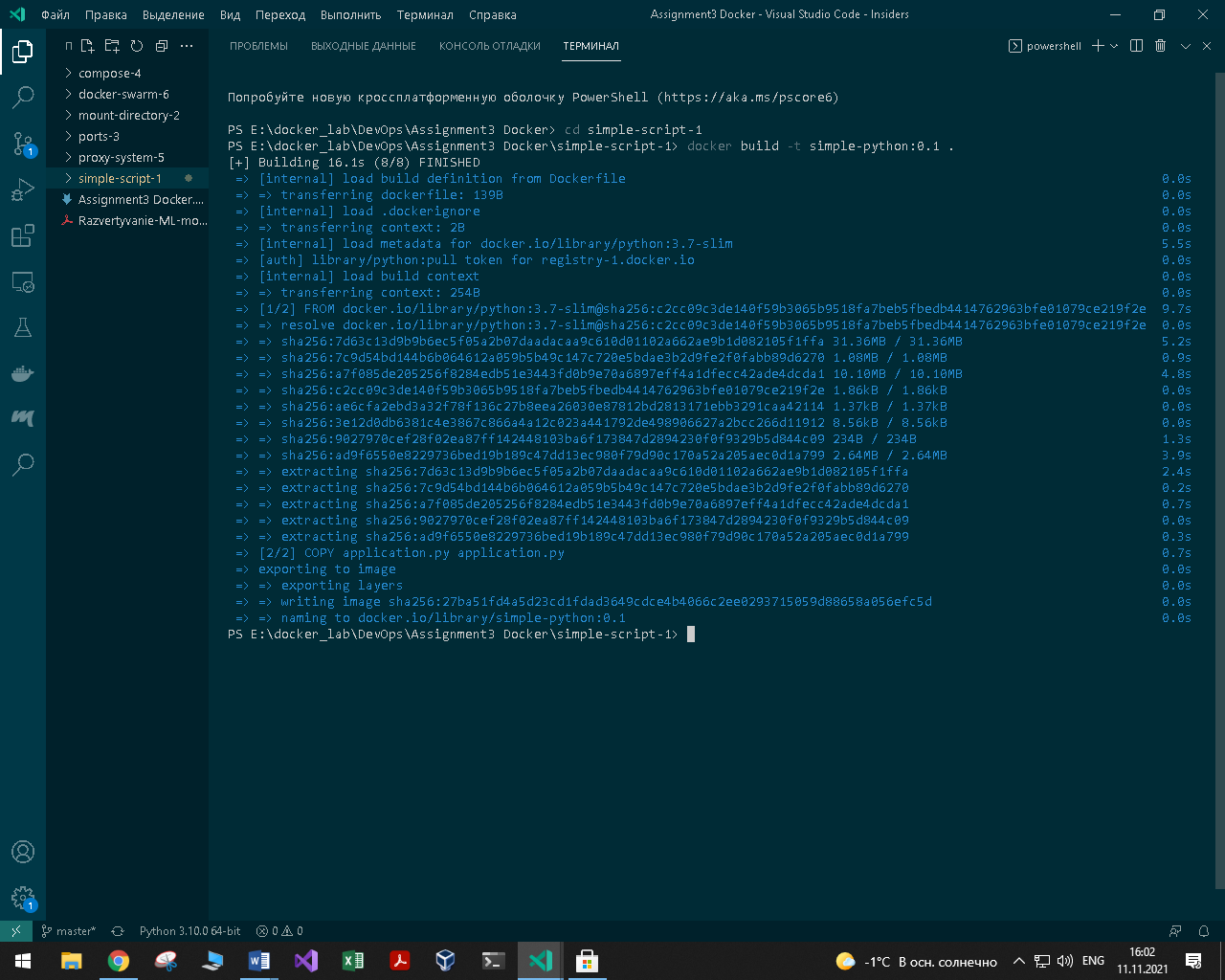
В Dockerfile изменим тег с FROM python:3.7 на FROM python:3.7-slim, поскольку это существенно снизит время загрузки и построения образа.



2.2 Построение Docker-образа.

Для построения Docker-образа воспользуемся командой:

$ docker build -t simple-python:0.1 .



2.3 Запуск построенного образа.

Для запуска построенного Docker-образа воспользуемся командой:

$ docker run simple-python:0.1

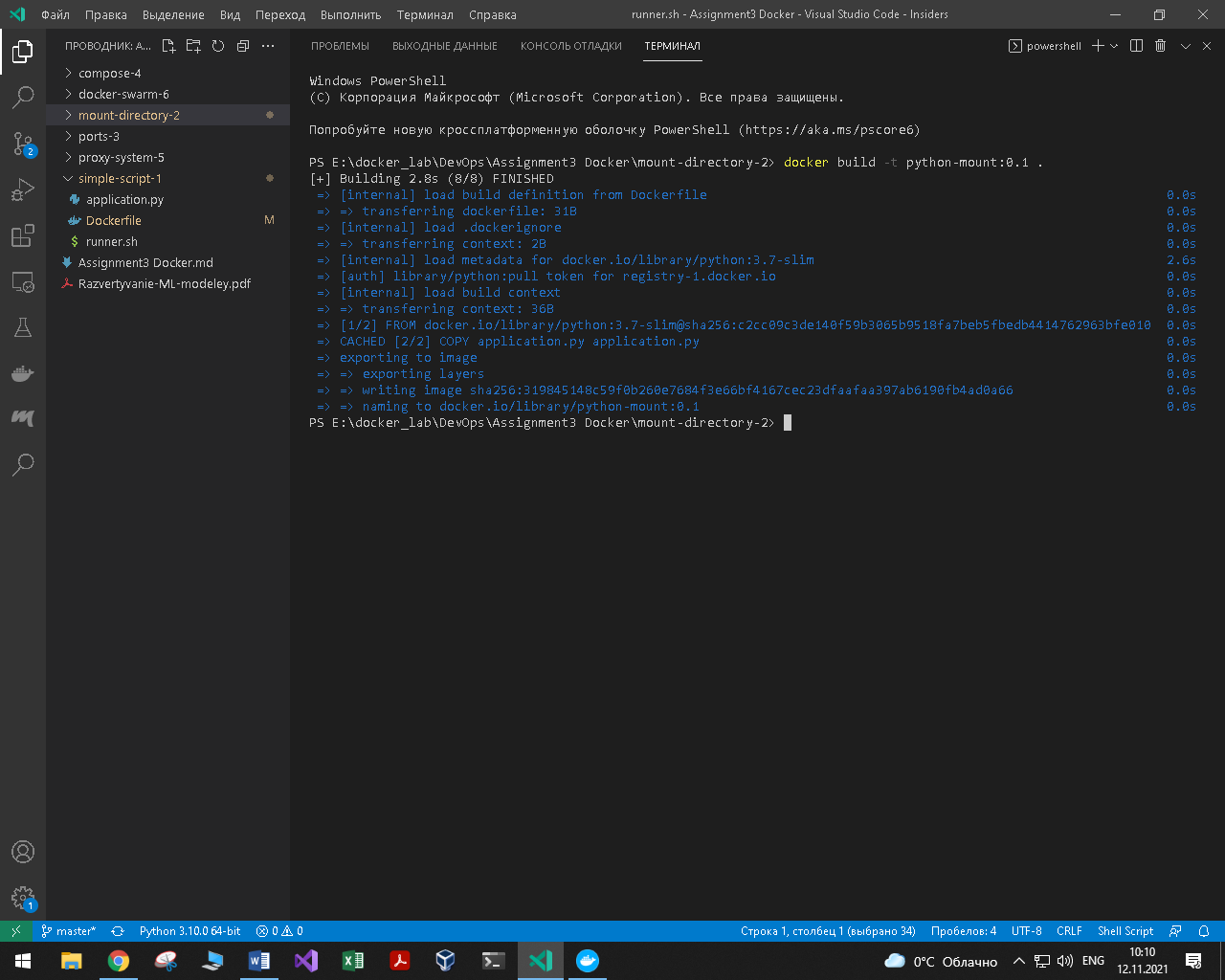


Шаг 3. Создание и запуск Docker- контейнера с установкой каталога.

3.1 Создадим Docker – образ.

Перед началом, создадим Docker-образ, на котором и будем запускать Docker-контейнер. Для создания Docker-образа воспользуемся командой:

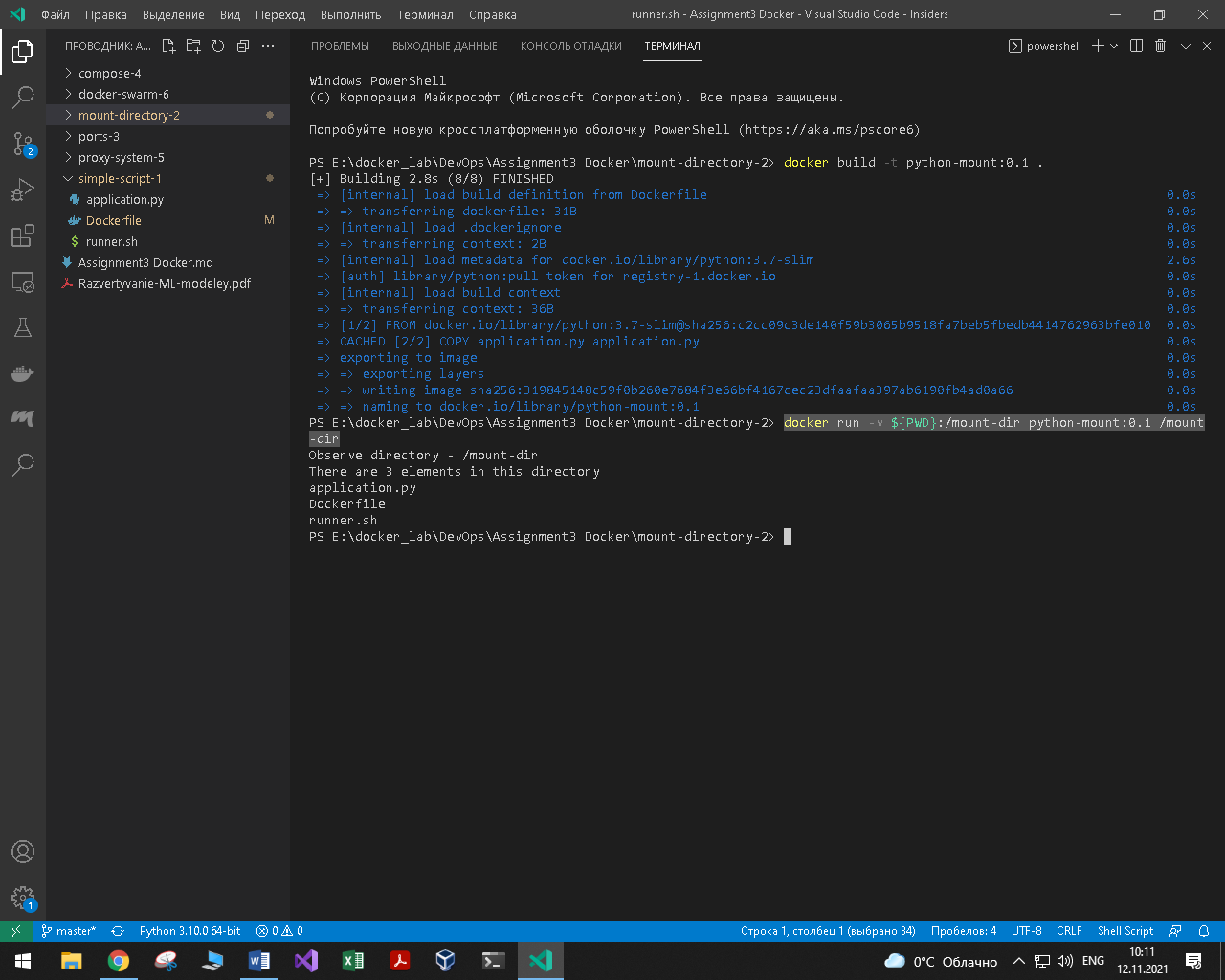
$ docker build -t python-mount:0.1 .



3.2 Запуск Docker-контейнера.

После того как Docker-образ был создан, запустим docker-контейнер. Для этого воспользуемся командой:

$ docker run -v ${PWD}:/mount-dir python-mount:0.1 /mount-dir

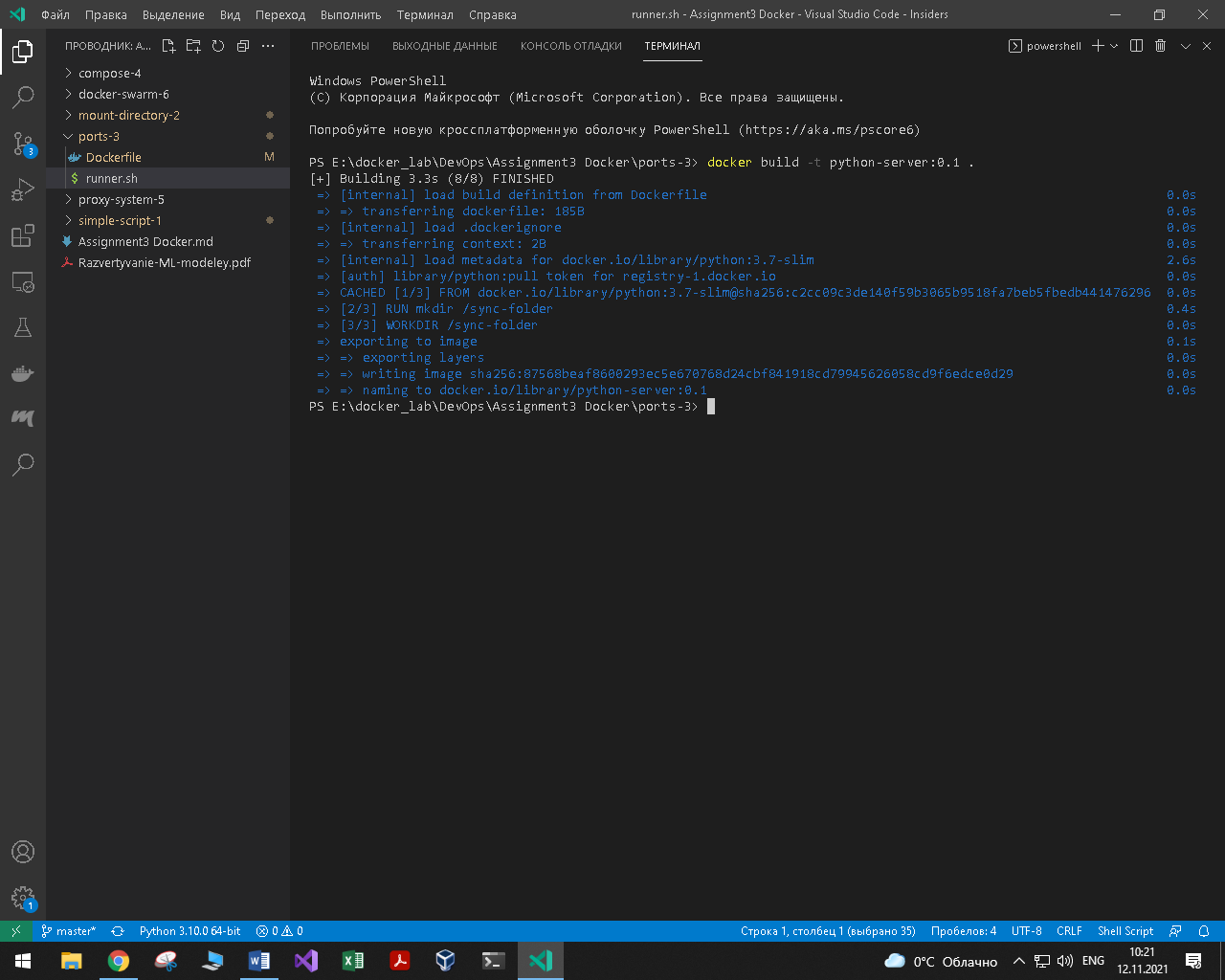


Шаг 4. Создание и запуск Docker-контейнера с пробросом портов.

4.1 Создание Docker –образа.

Создадим Docker –образ, из которого далее будем запускать Docker-контейнер, для построения Docker-образа воспользуемся следующей командой:

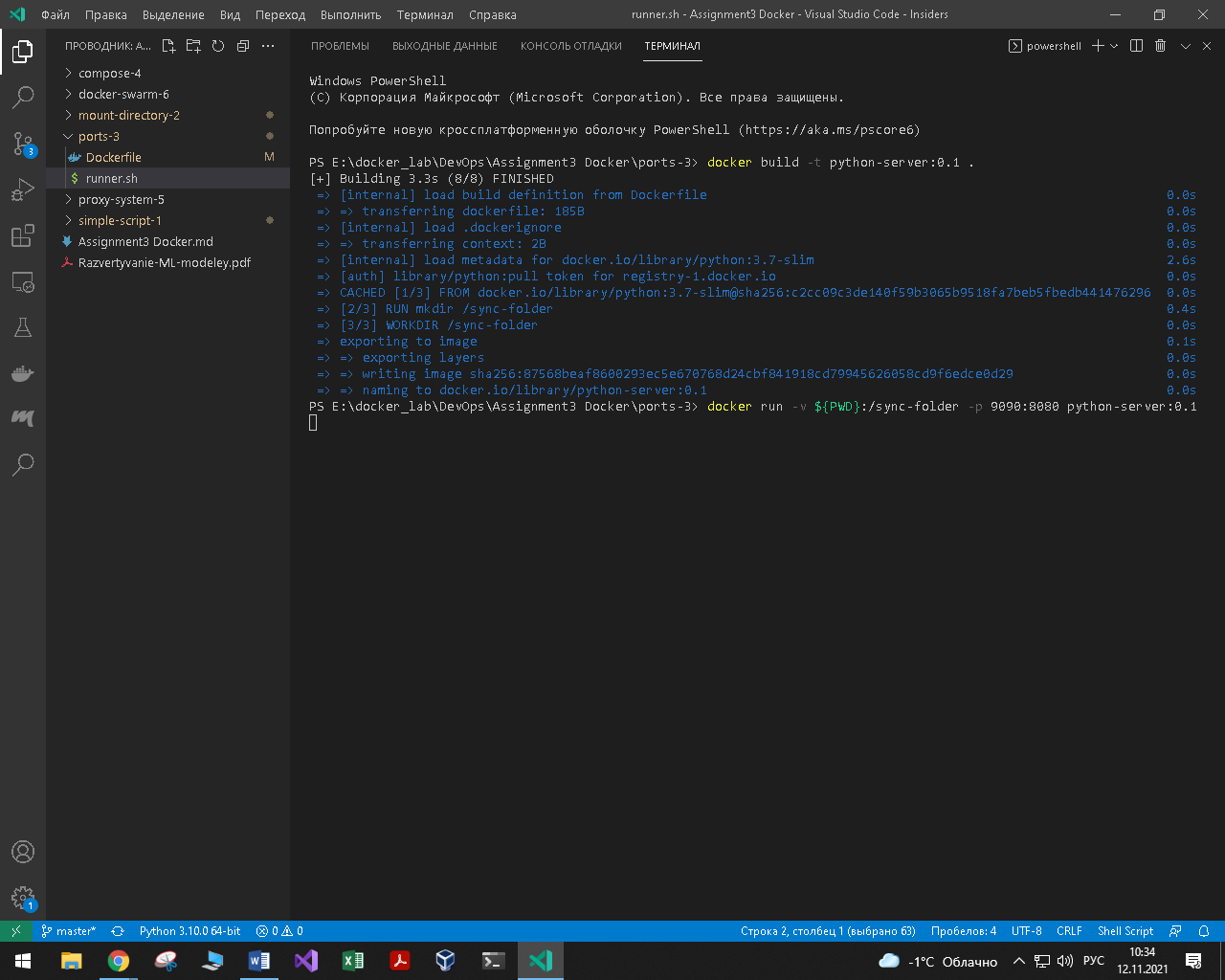
$ docker build -t python-server:0.1 .



4.2 Запуск Docker-контейнера.

Запустим построенный Docker-образ. Для этого воспользуемся следующей командой:

docker run -v ${PWD}:/sync-folder -p 9090:8080 python-server:0.1

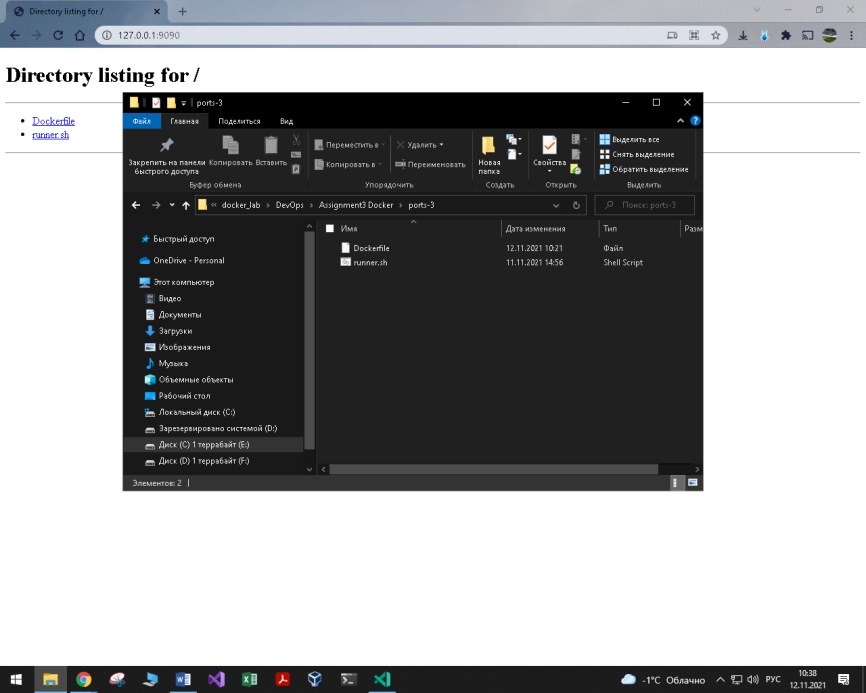


4.3 Проверка результатов.

Проверим результат запуска, в адресной строке браузера пропишем следующую команду:

<http://127.0.0.1:9090/>

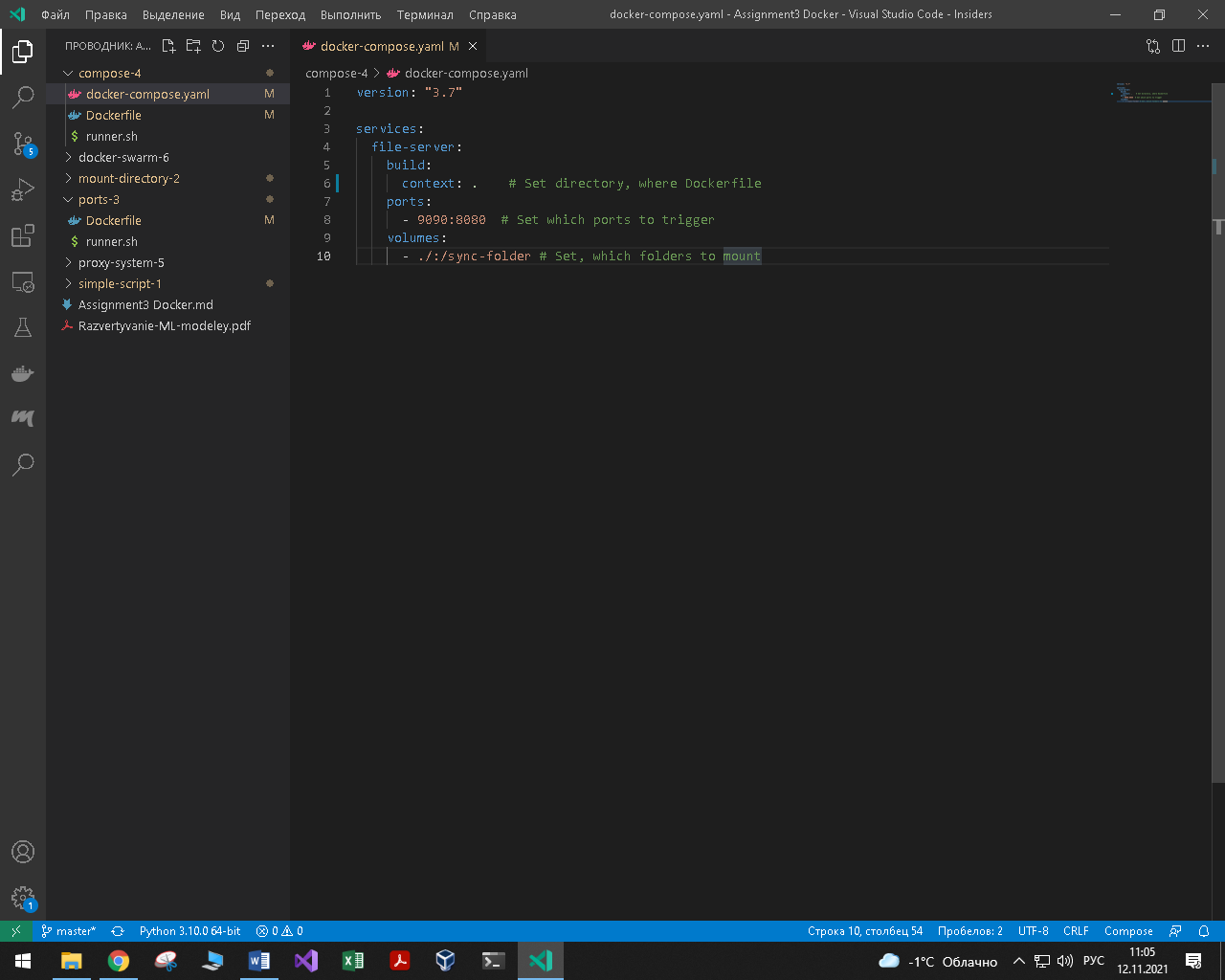
А заодно проверим расположение в директории на хосте.



Шаг 5. Создать и запустить Docker-контейнер при помощи docker-compose

5.1 Корректировка yml файла.

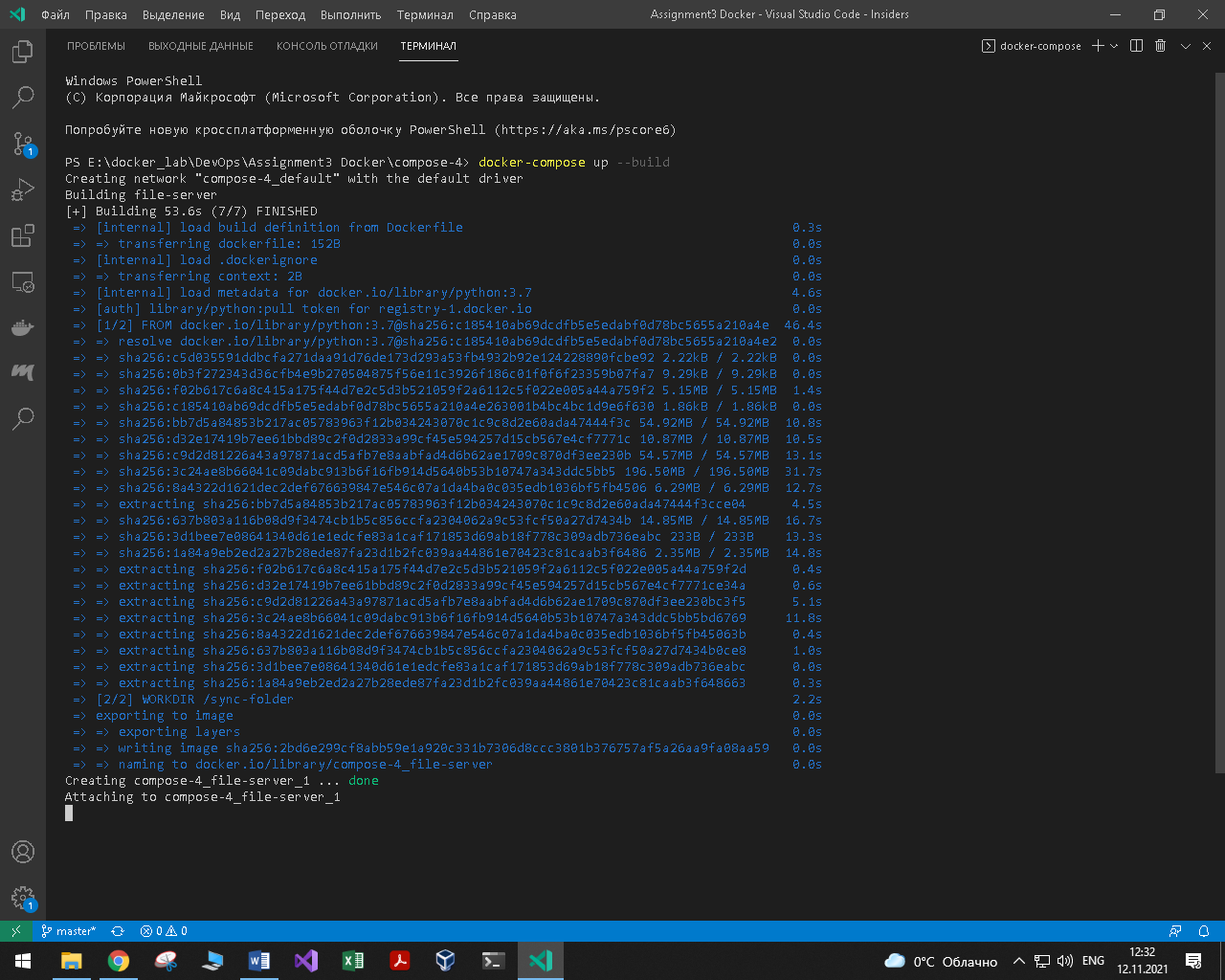
Перед началом выполнения данного задания необходимо исправить неверный docker-compose.yml файл. А именно исправить значение ключа context: context: .



5.2 Сборка Docker-образа и запуск Docker-контейнера

Для последовательной сборки Docker-образа и запуска Docker-контейнера воспользуемся командой:

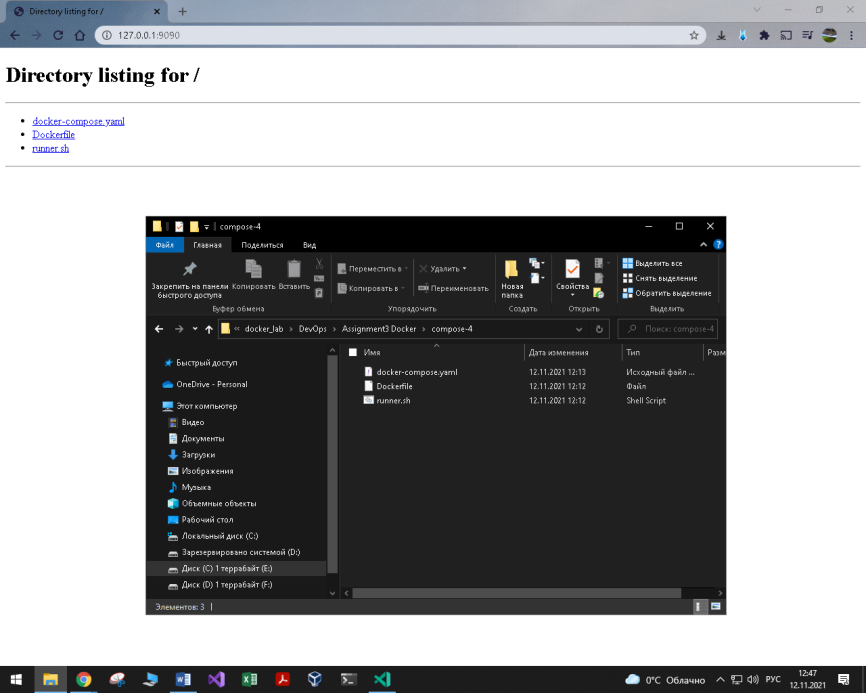
$ docker-compose up –build



5.3 Проверка выполнения.

После того как все было собрано и запущено проверим правильность выполнения, перейдем в браузер и в адресной строке наберем следующий адрес, а также проверим в расположенной директории:

http://127.0.0.1:9090/

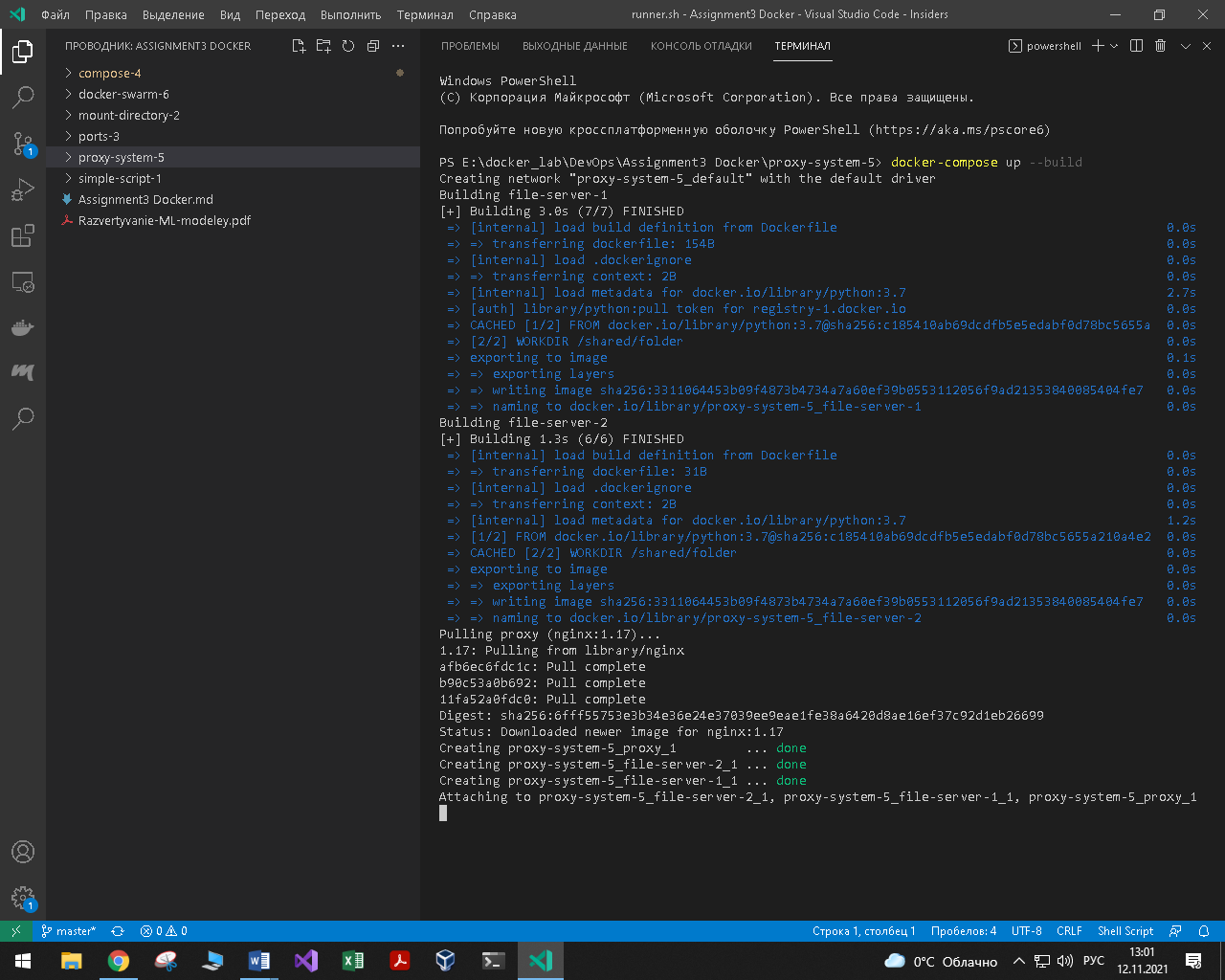


Шаг 6. Создание и запуск Docker-контейнера с распределением нагрузки.

6.1 Создание и запуск Docker-контейнера.

Для одновременного создания и запуска Docker-контейнера воспользуемся командой:

$ docker-compose up --build



6.2 Проверка работоспособности.

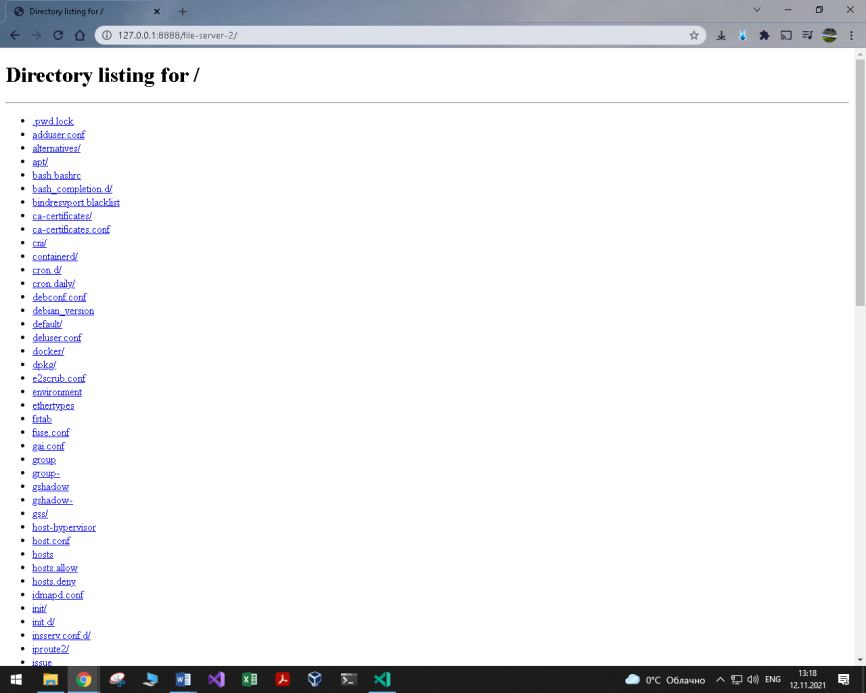
Для проверки первого сервера перейдем в браузер и в адресной строке введем следующий адрес:

http://127.0.0.1:8888/file-server-1/



Для проверки второго сервера перейдем в браузер и в адресной строке введем следующий адрес:

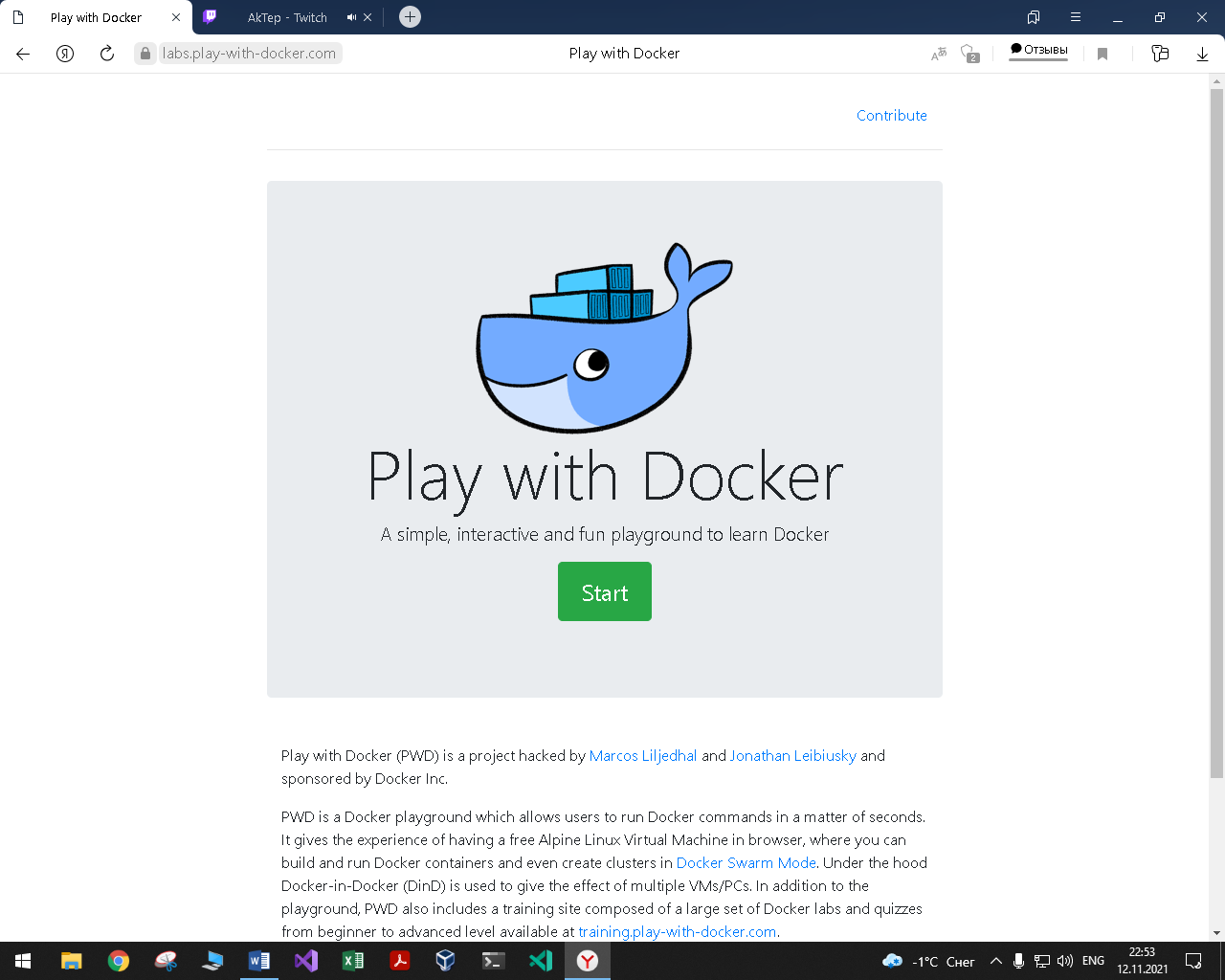
http://127.0.0.1:8888/file-server-2/



Шаг 7. Создать и запустить Docker-Swarm конфигурацию.

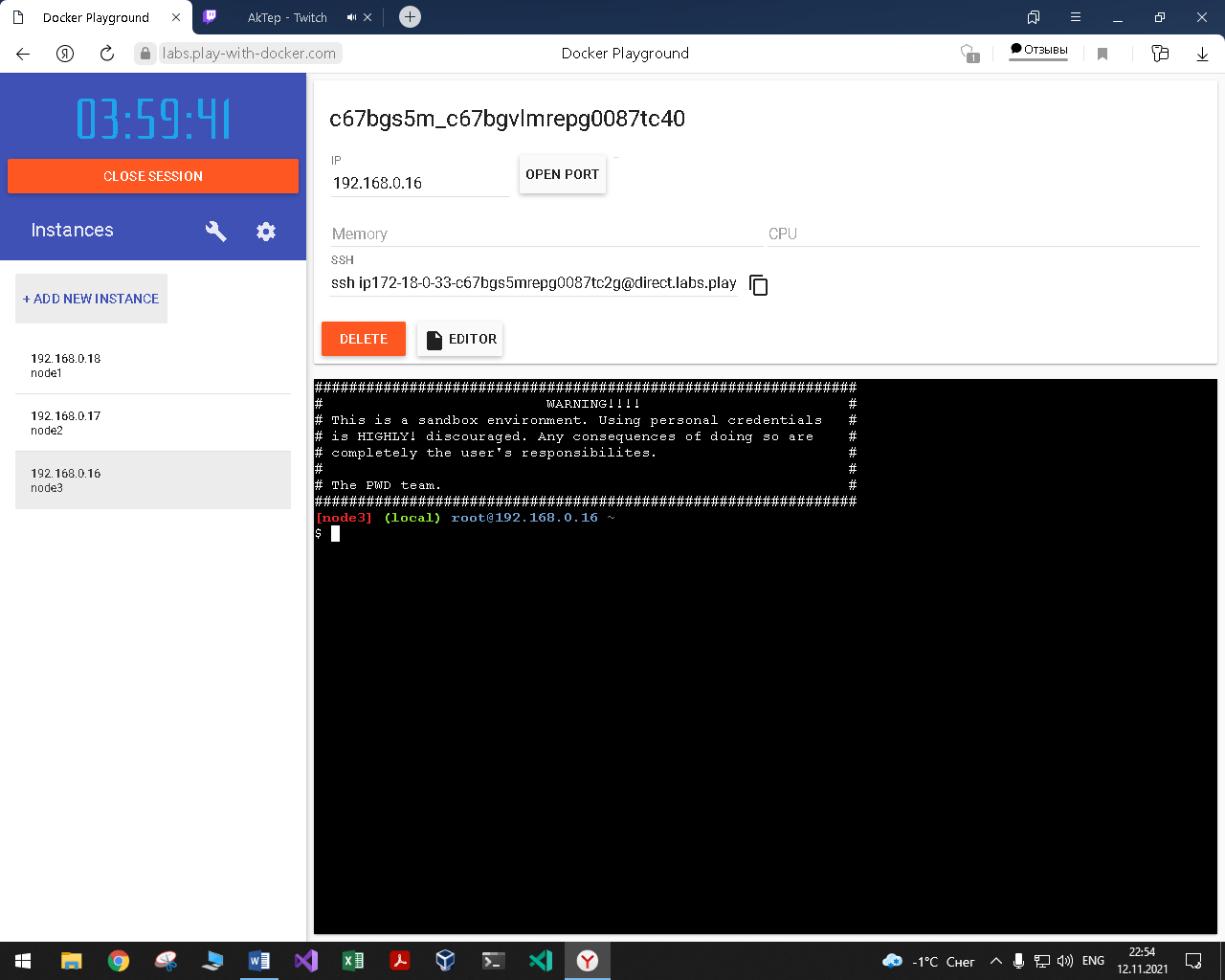
7.1 Play-with-docker.

Для выполнения данного шага лабораторной работы перейдем из Docker Desktop на сайт <https://labs.play-with-docker.com/>



7.2 Создание node.

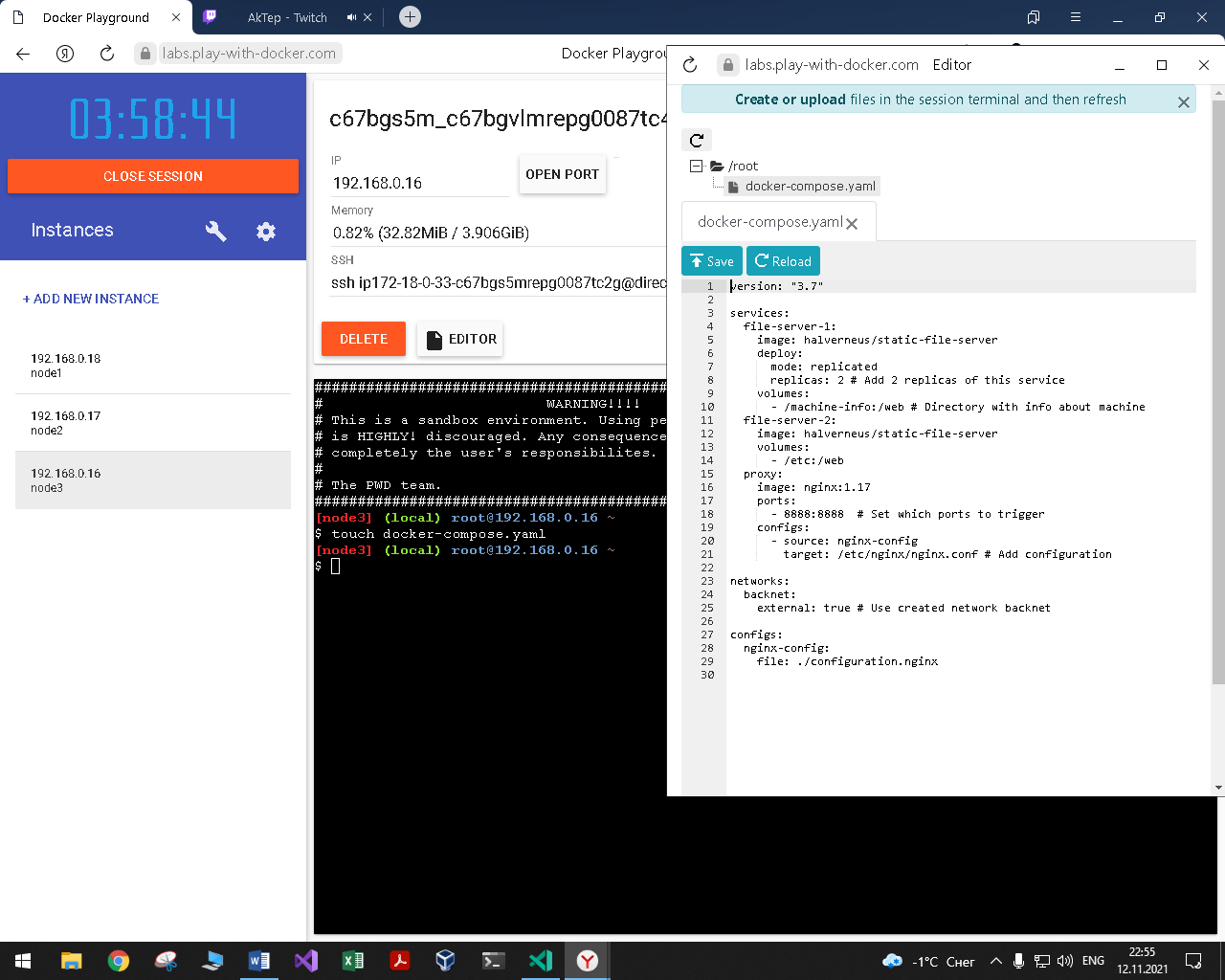
Создадим node в количестве трех штук. Для этого достаточно нажать три раза на «Add new instance».



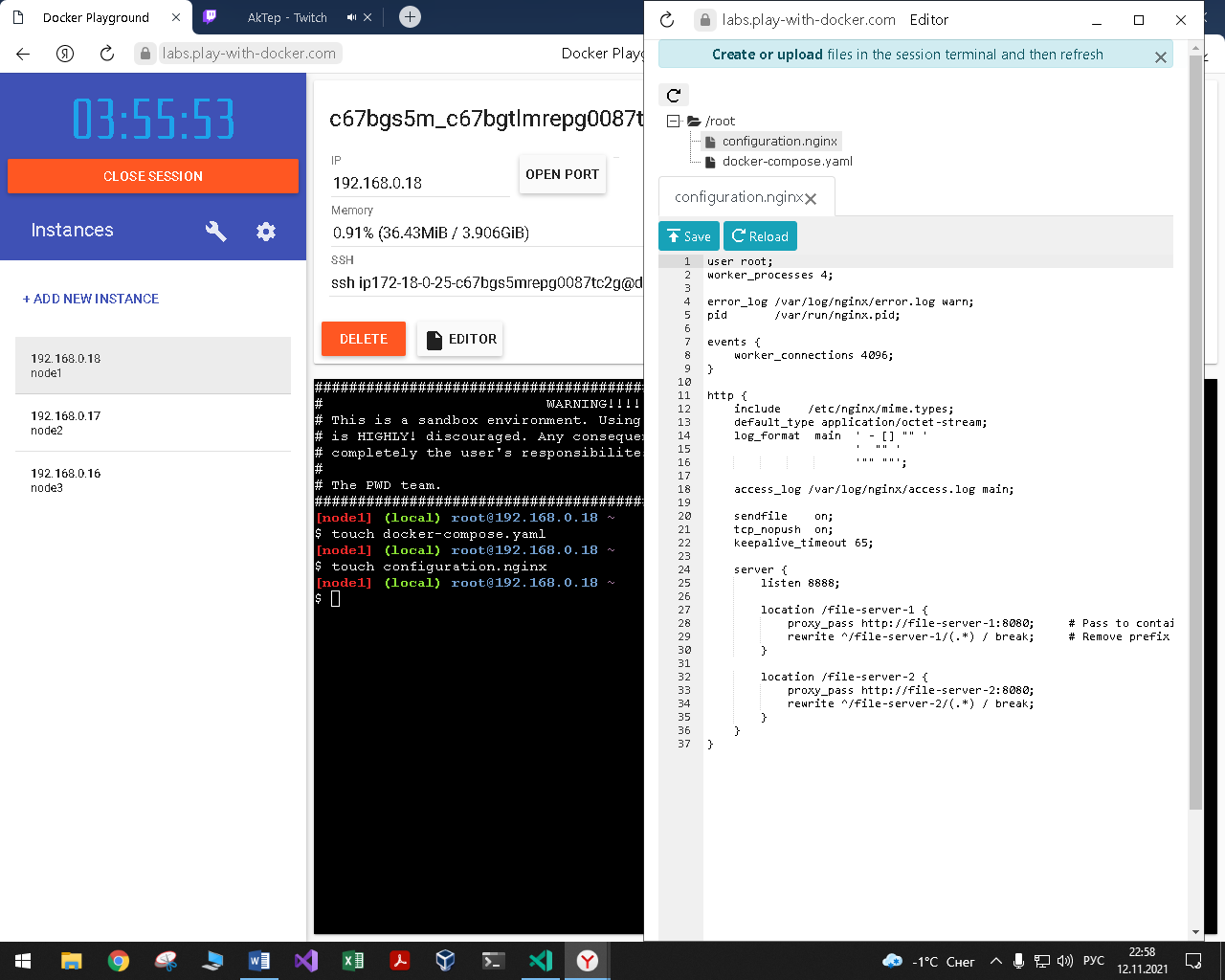
7.1 Добавление файлов и назначение manager mode.

Создадим файлы из папки docker-swarm-6 скачанной из репозитория с заданием, а так же скопируем их сожержание..

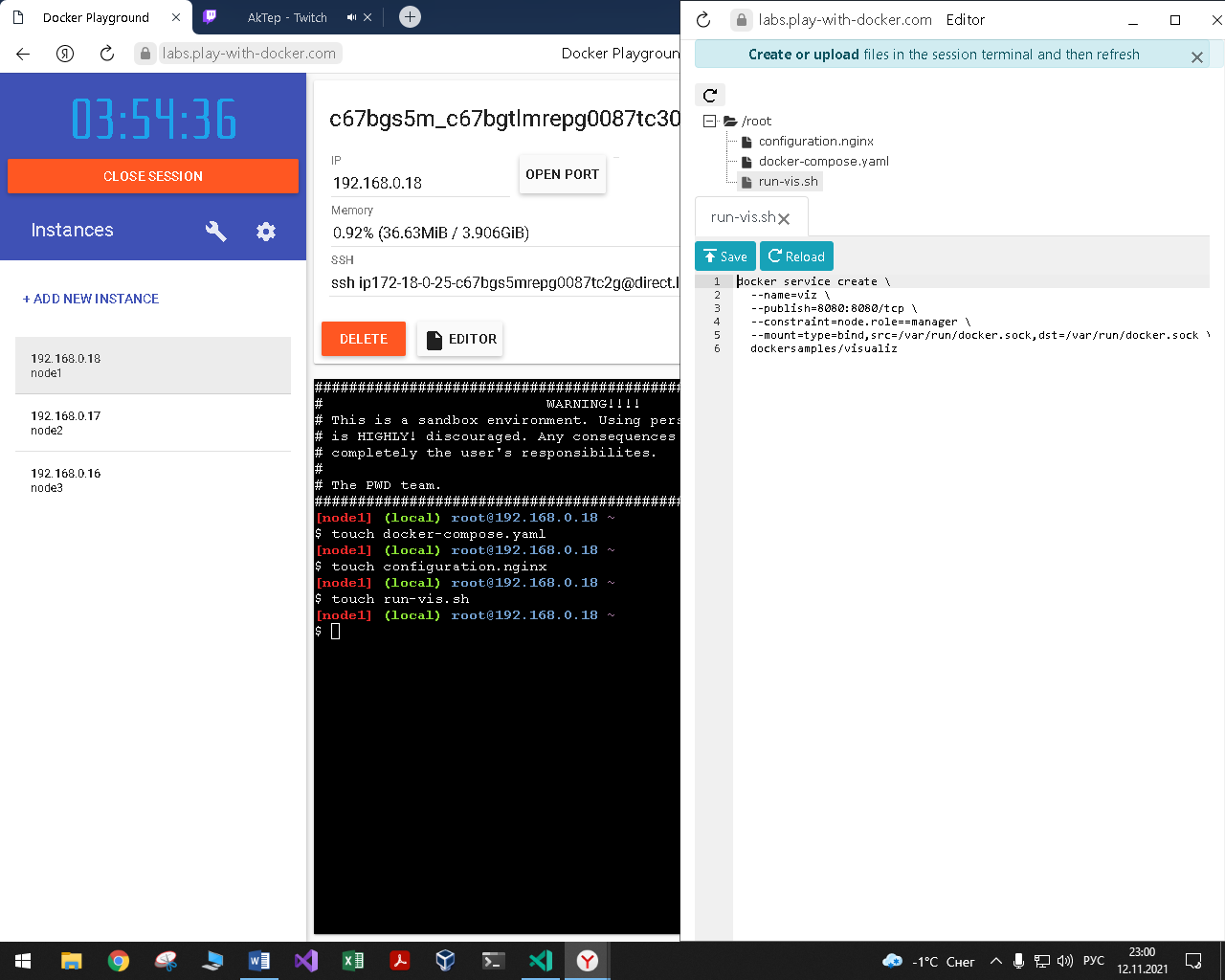
$ touch docker-compose.yaml



$ touch configuration.nginx

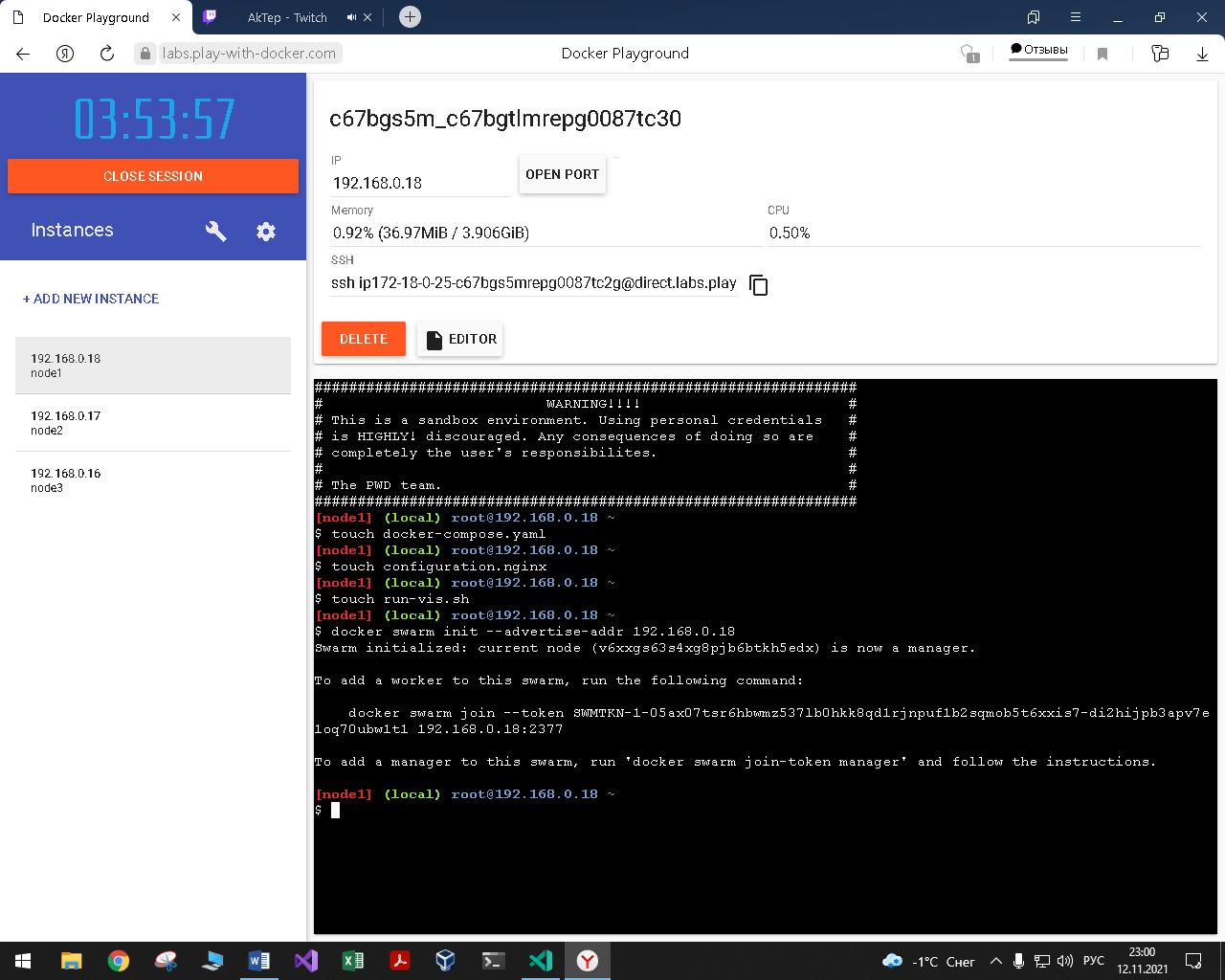


$ touch run-vis.sh



Node1 присвоим manager mode

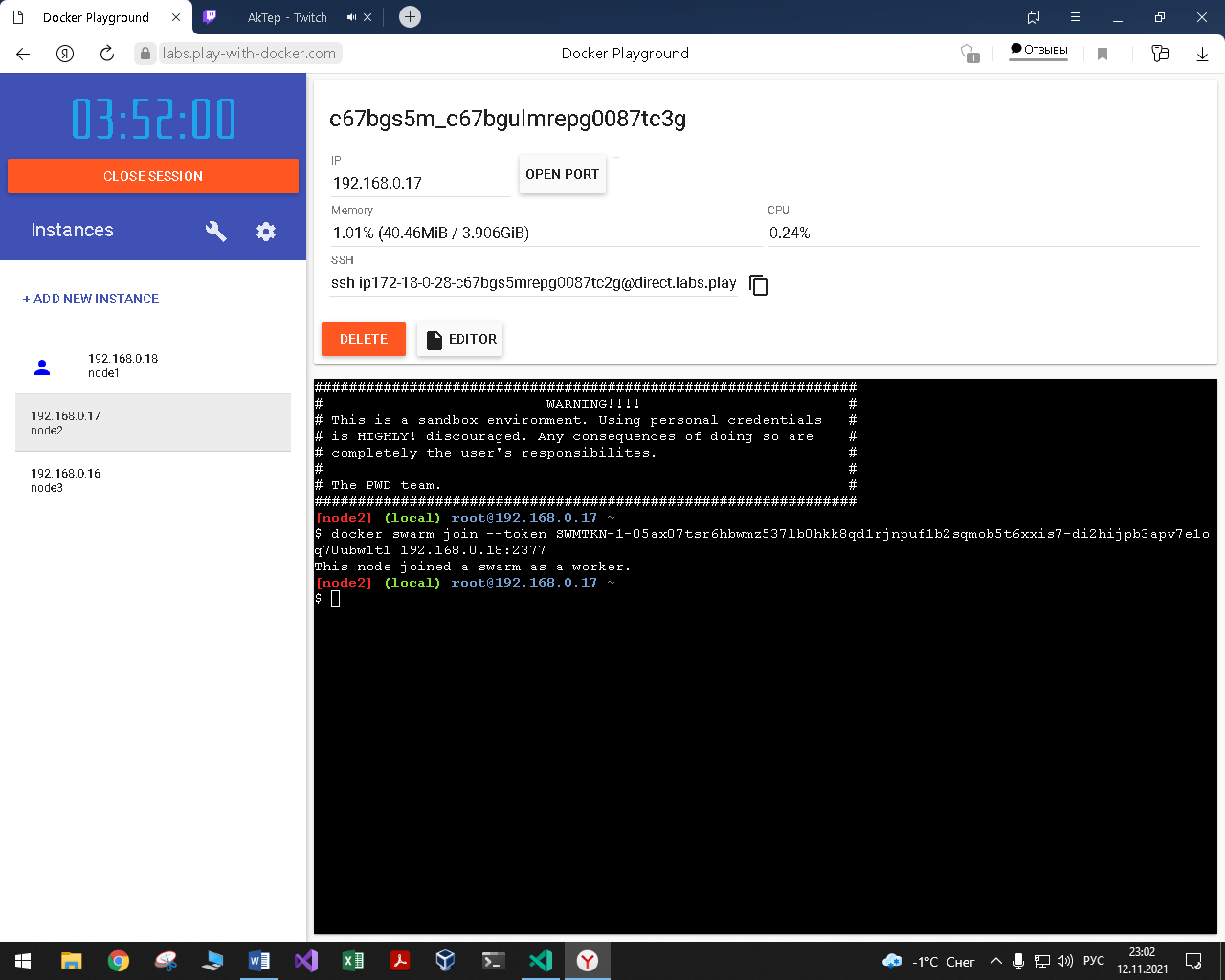
$ docker swarm init --advertise-addr 192.168.0.18



7.2 Присоедините оставшихся node к главной.

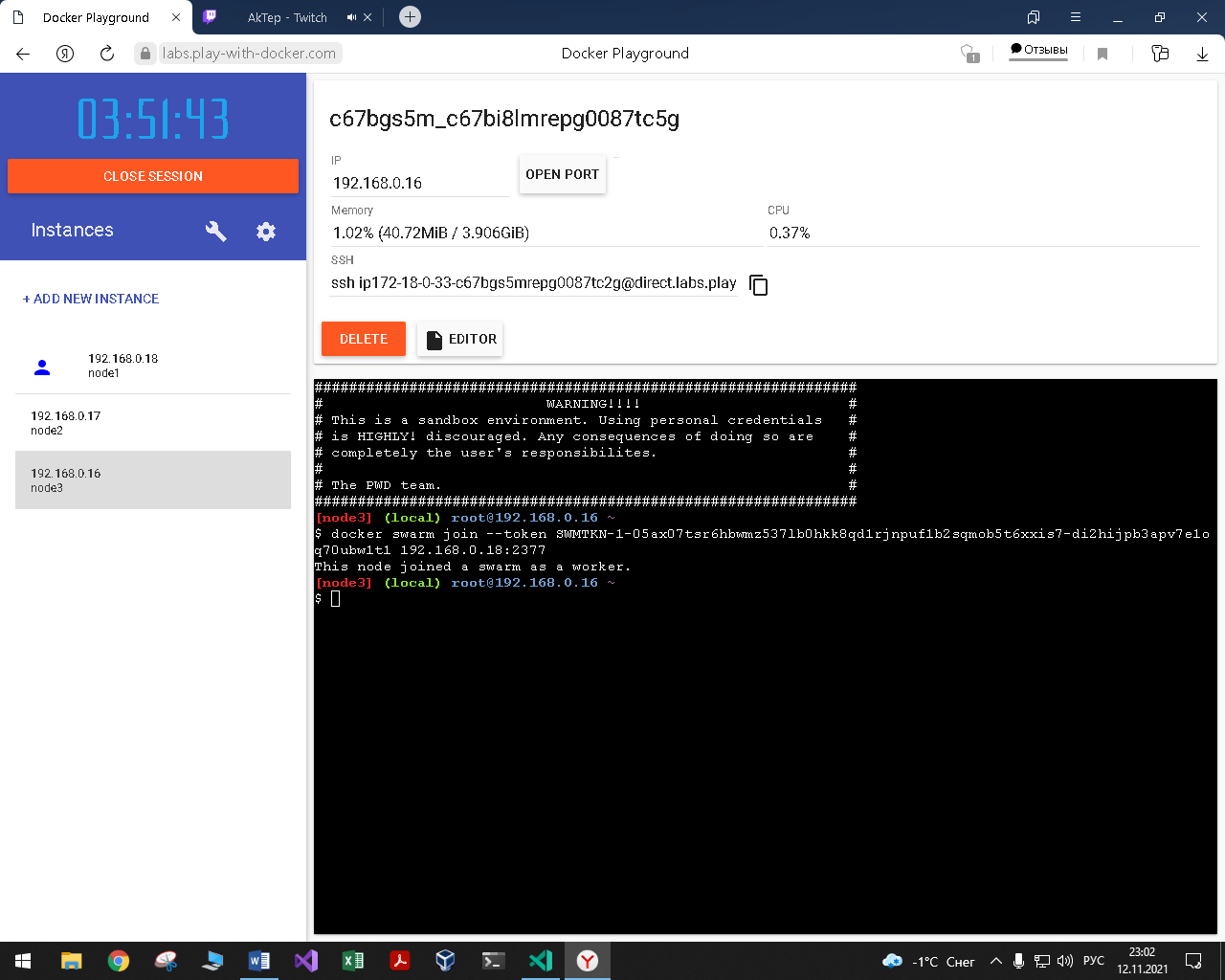
7.2.1 Присоединение node2:

$ docker swarm join --token SWMTKN-1-05ax07tsr6hbwmz537lb0hkk8qd1rjnpuf1b2sqmob5t6xxis7-di2hijpb3apv7e1oq70ubw1t1 192.168.0.18:2377



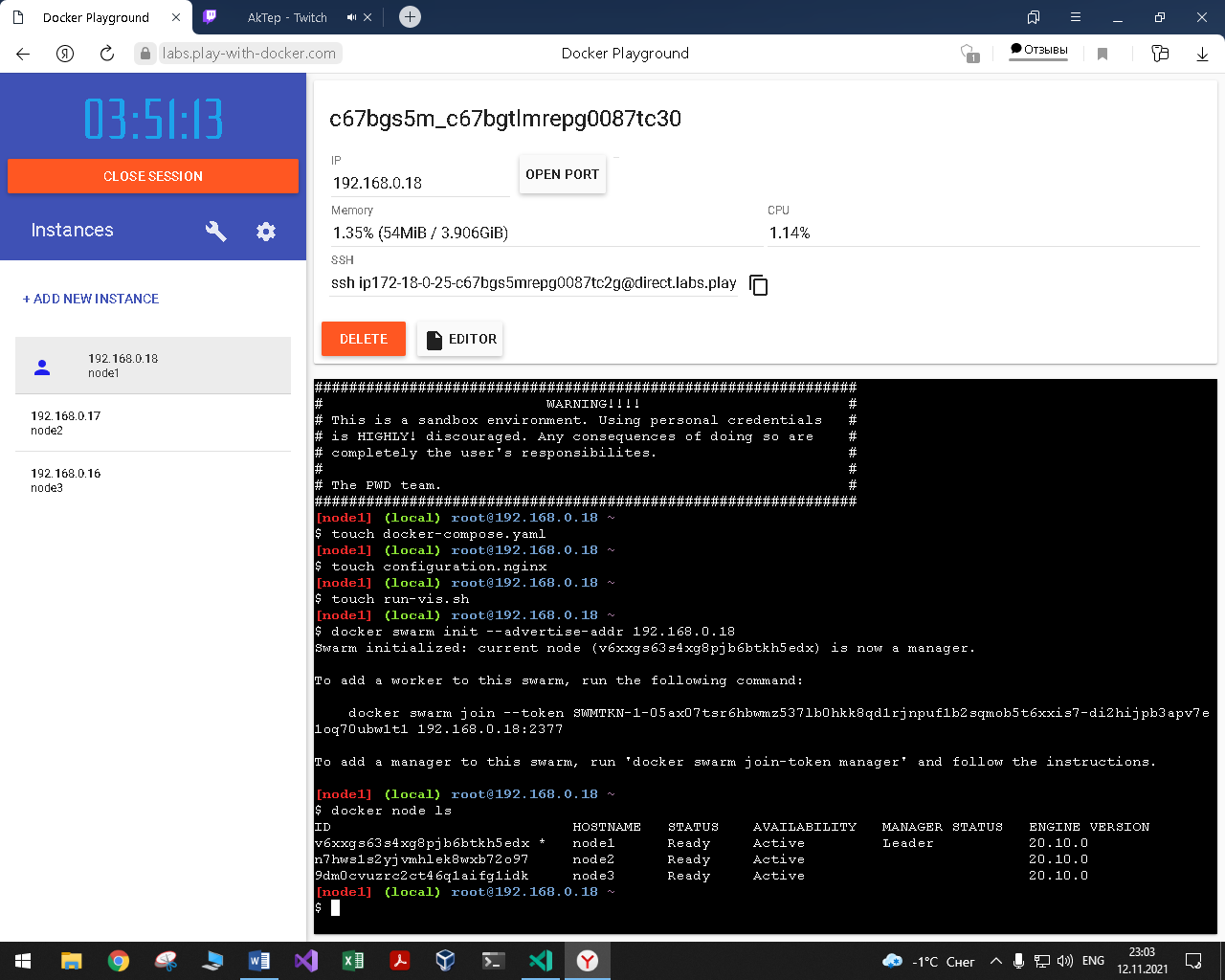
7.2.2 Присоединение node3:

$ docker swarm join --token SWMTKN-1-05ax07tsr6hbwmz537lb0hkk8qd1rjnpuf1b2sqmob5t6xxis7-di2hijpb3apv7e1oq70ubw1t1 192.168.0.18:2377



7.3 Запуск Docker Swarm Visualizer на главной node.

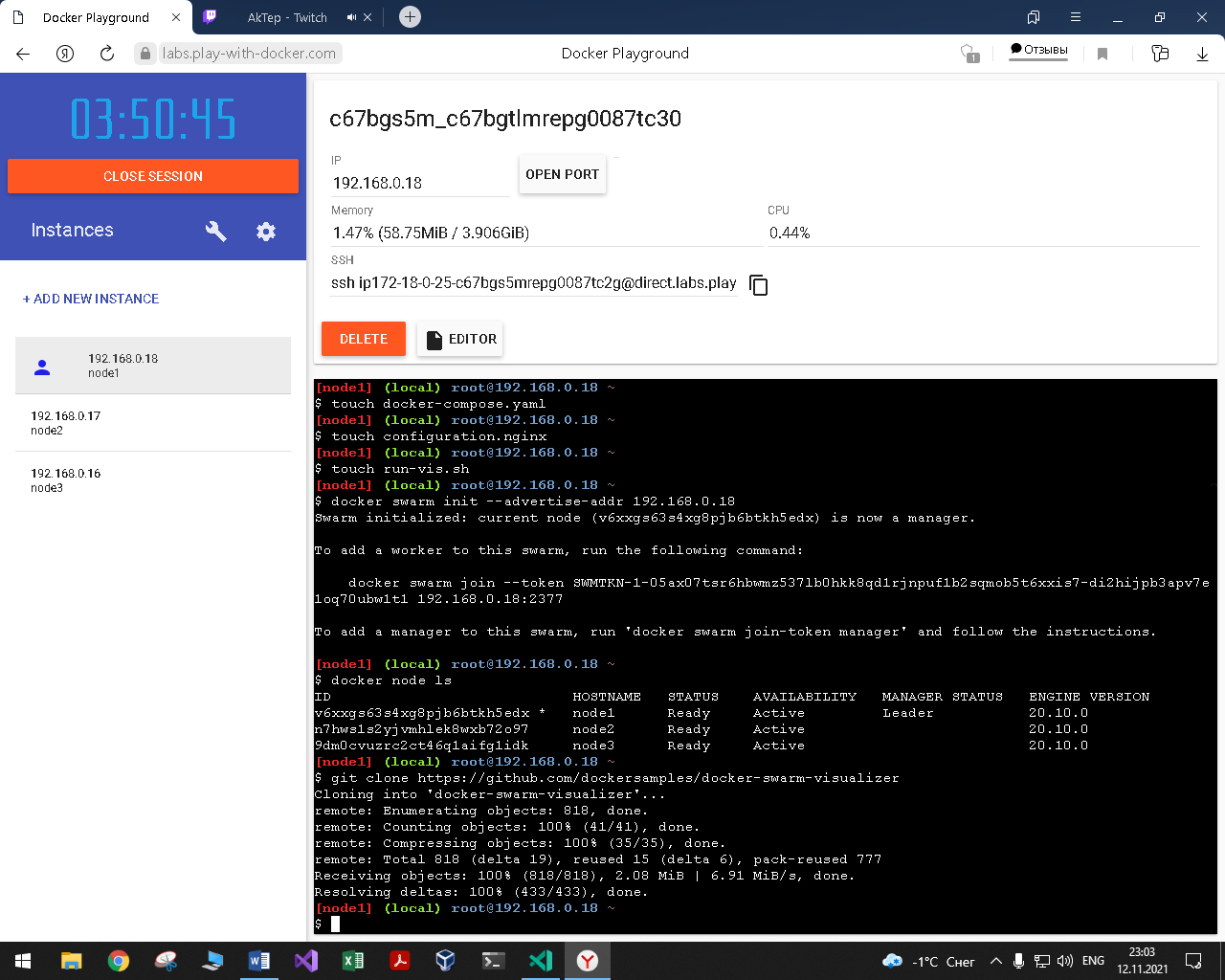
Проверим подключение всех node.



7.4 Копирование репозитория Docker Swarm Visualizer.

Для копирования репозитория Docker Swarm Visualizer воспользуемся командой:

$ git clone https://github.com/dockersamples/docker-swarm-visualizer



7.5 Запуск docker-swarm-visualizer

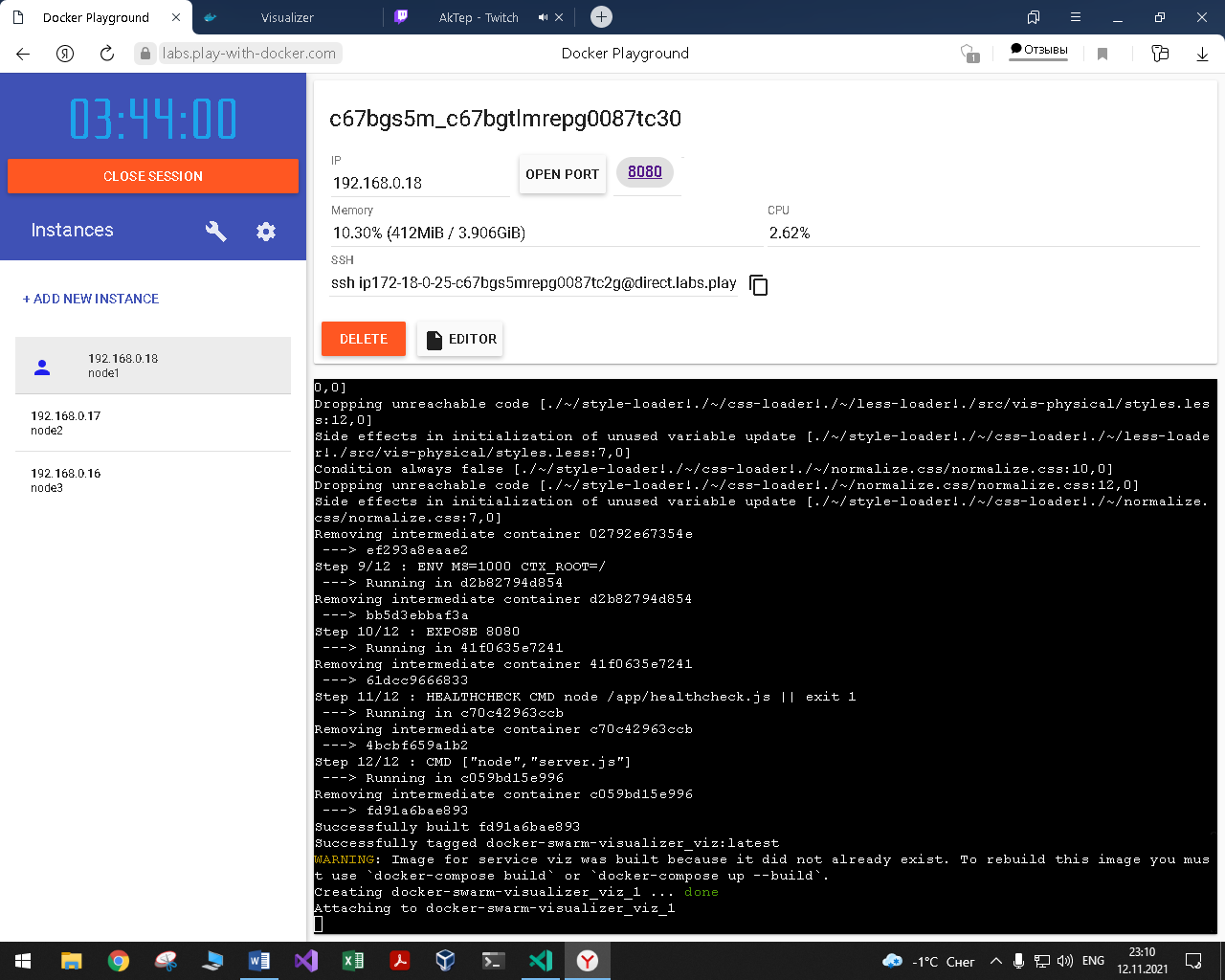
Перейдем в папку в которой расположен docker-swarm-visualizer:

$ cd docker-swarm-visualizer

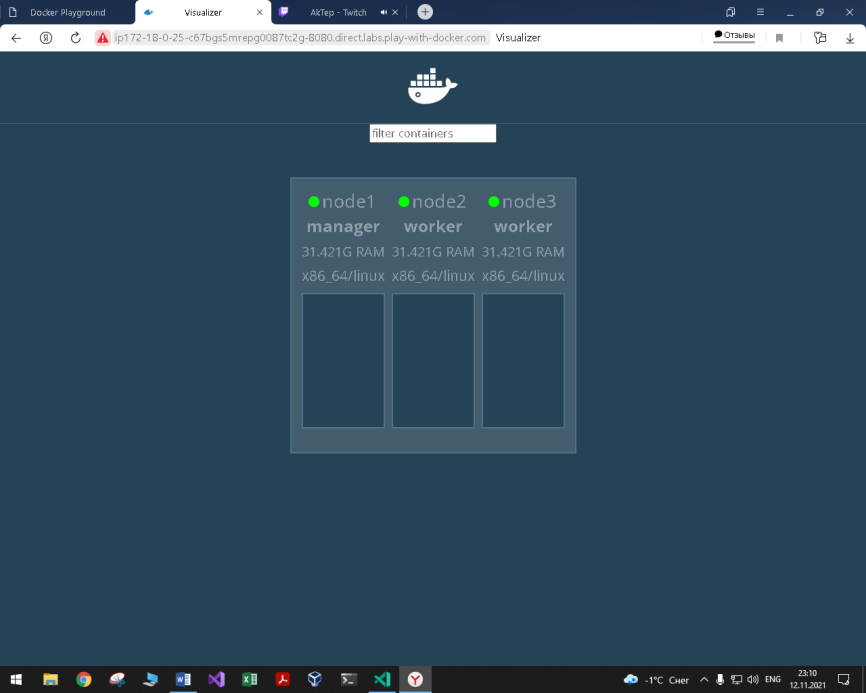
И поднимем его следующей командой

$ docker-compose up -d

После чего проверим работоспособность. Для этого нажмем на кнопку «8080».



После чего нас выведет на графический интерфейс.



7.6 Развертывание приложения.

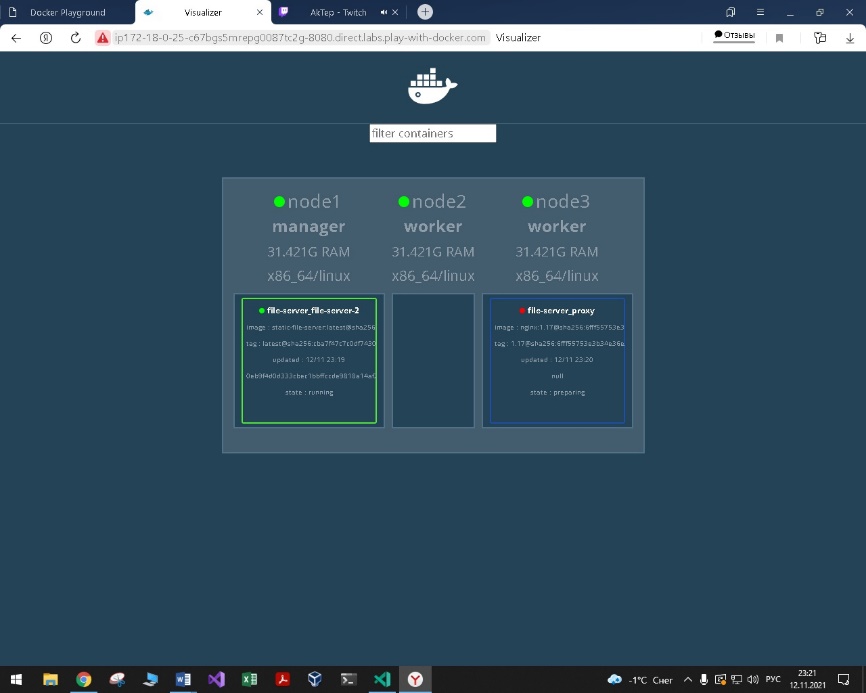
Для развертывания приложения необходимо выполнить следующий набор команд:

$ docker network create --driver overlay backnet

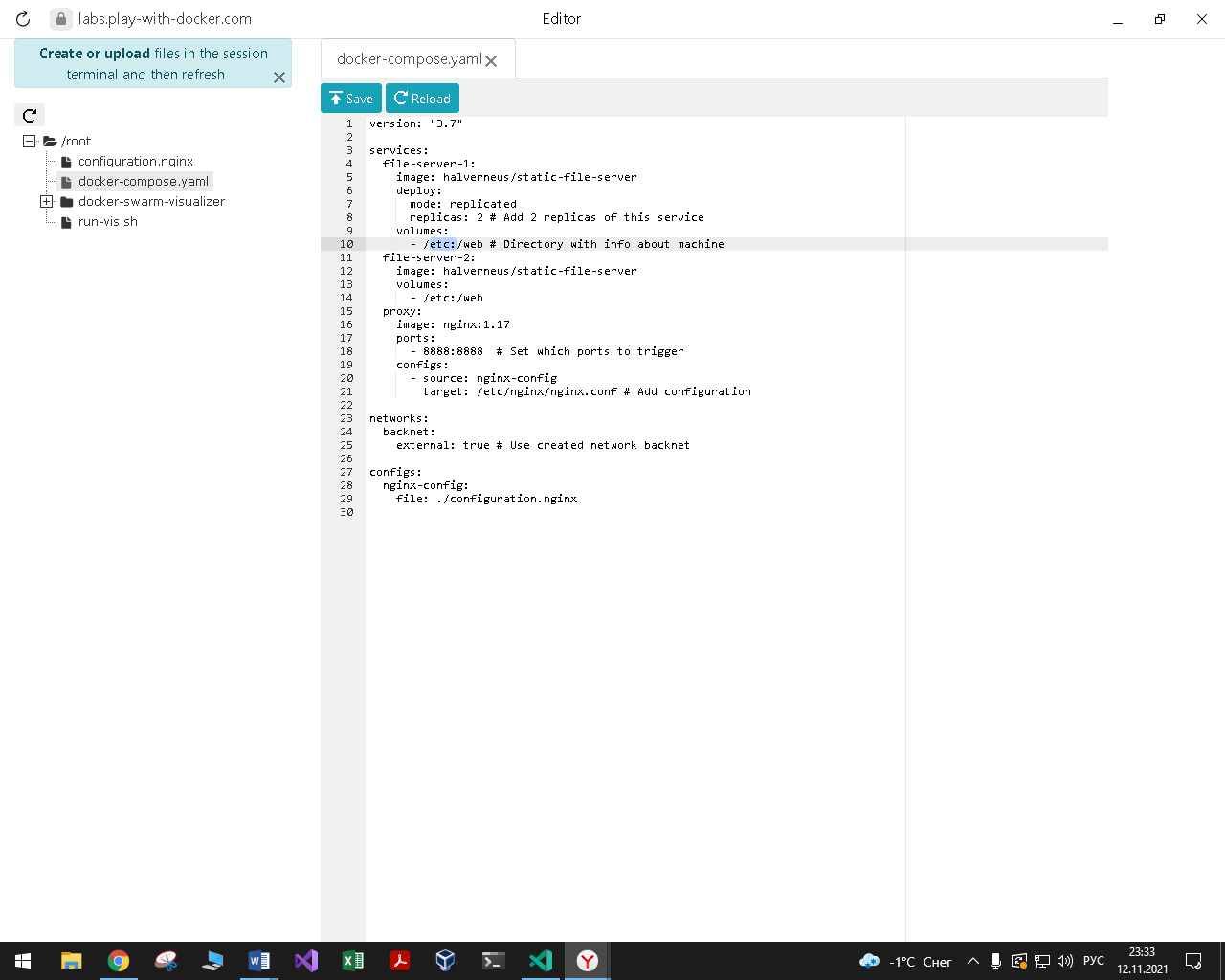
$ cd ~

$ docker stack deploy --compose-file docker-compose.yaml file-server

После чего проверим работоспособность.

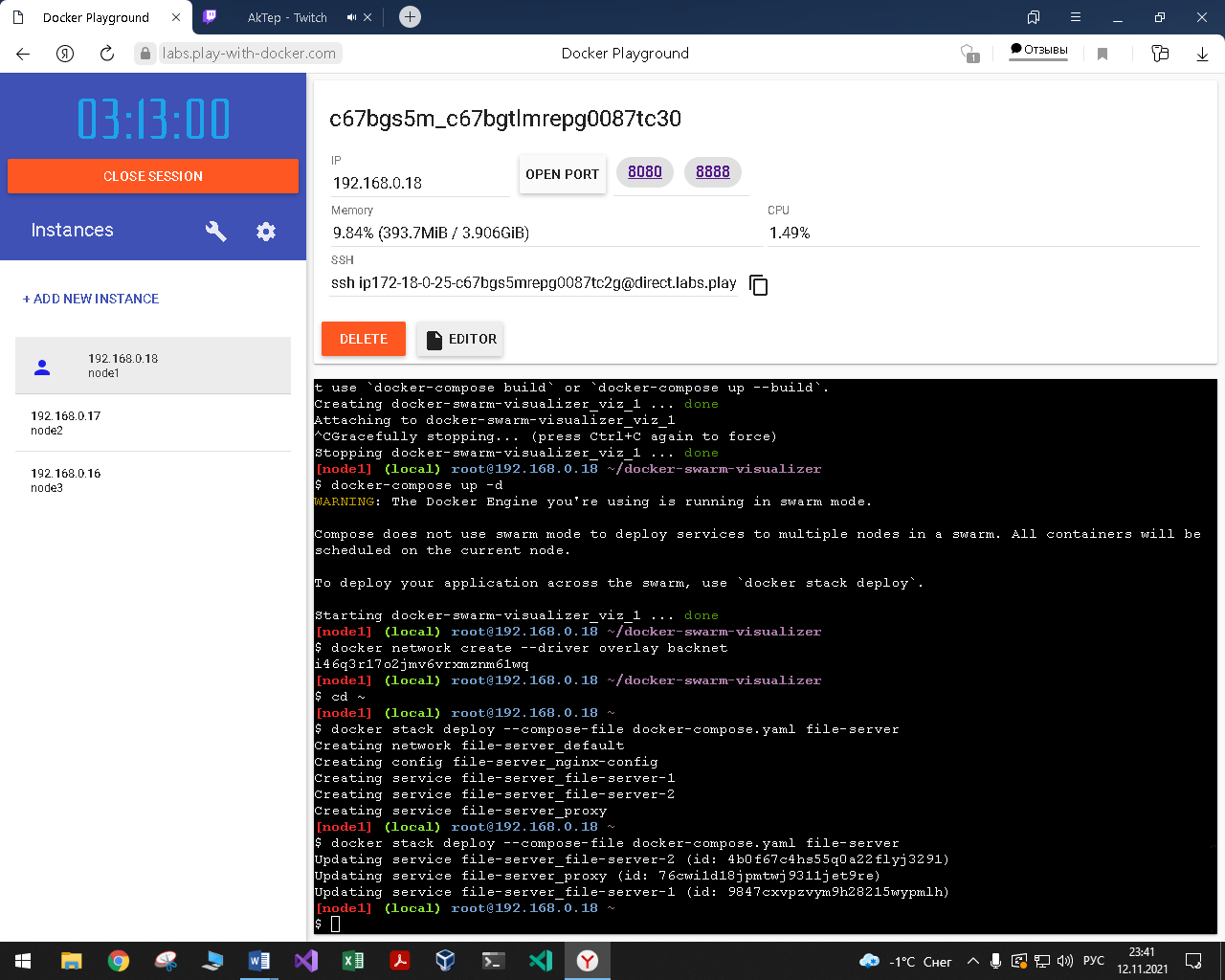


Вроде работает, но не совсем ерно. Зайдем в docker-compose.yaml файл и изменим /machine-info на /etc, поскольку директории /machine-info у нас нет.

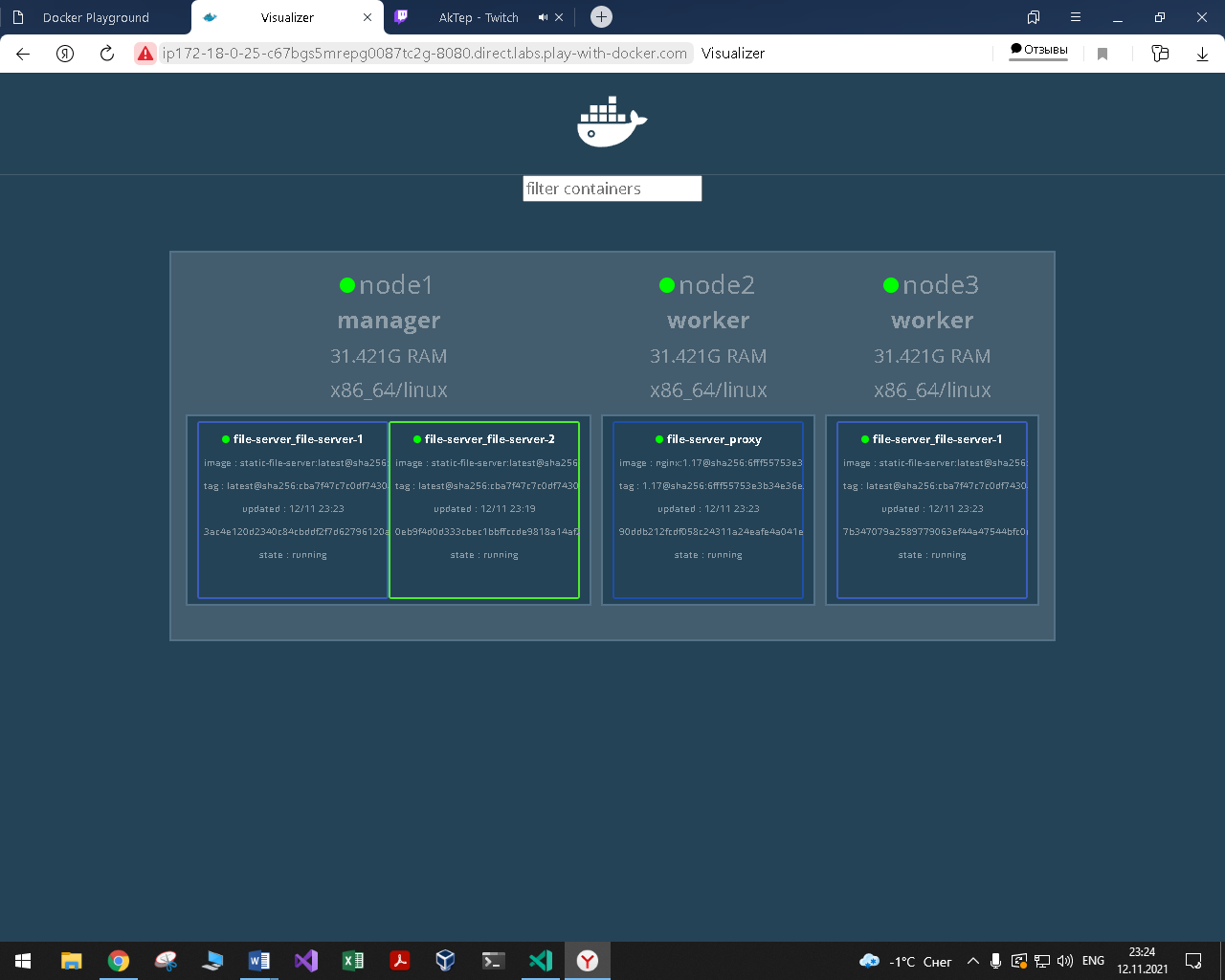


После сохранения повторим команду:

$ docker stack deploy --compose-file docker-compose.yaml file-server



И проверим работоспособность.



Теперь все работает

**Содержание отчета**

В состав отчета входит:

1. Титульный лист
2. Задание
3. Ход работы с подробным описанием каждой выполняемой командой со всеми собственными скриншотами, на которых должны быть видны дата и время выполнения работы
4. Форматирование текста: номера страниц, верхний колонтитул