

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное
образовательное учреждение высшего образования
«Самарский национальный исследовательский университет
имени академика С.П. Королева»
(Самарский университет)

Институт информатики, математики и электроники
Факультет информатики
Кафедра суперкомпьютеров и общей информатики

Отчет по лабораторной работе №3

Дисциплина: «Развертывание и жизненный цикл программного обеспечения»

Тема: «**Docker**»

Выполнил: Мелешенко И.С.

Группа: 6133-010402D

Самара 2021

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.

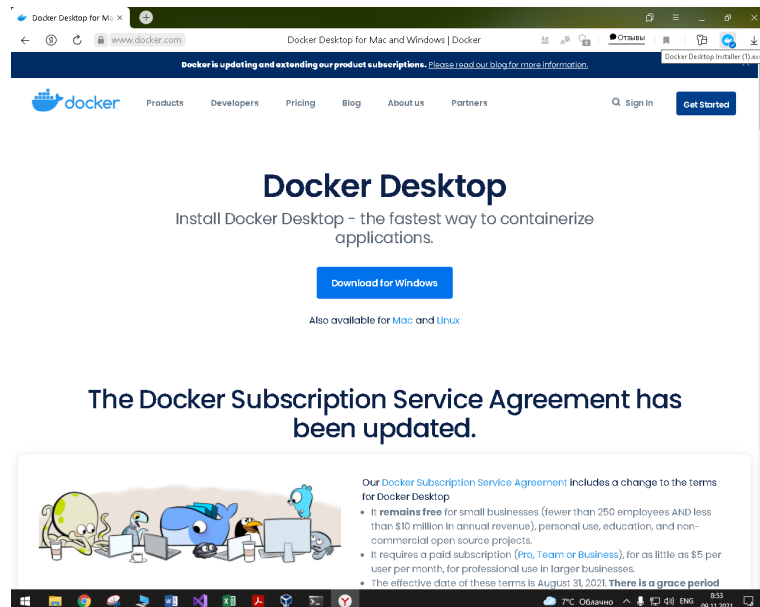
2.6. (docker-swarm-6) Создать и запустить Docker-Swarm конфигурацию.

ХОД ВЫПОЛНЕНИЯ РАБОТЫ

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>

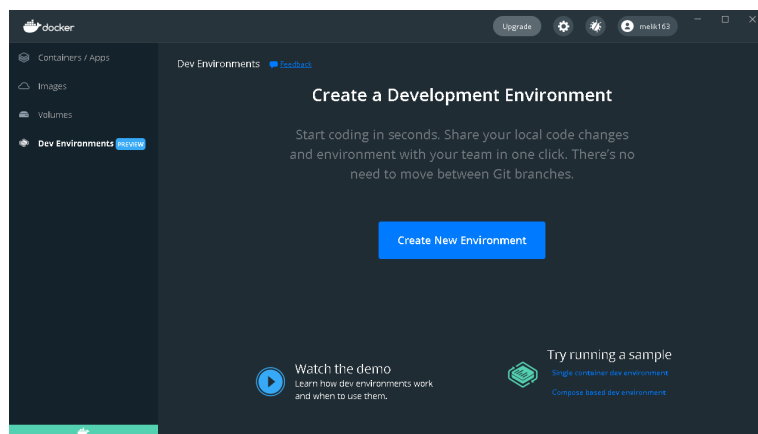


1.2 На данном ПК потребуется произвести настройку для запуска программы Docker Desktop, более подробно об этом можно почитать на сайте по ссылке <https://docs.microsoft.com/en-us/windows/wsl/install-manual>.

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

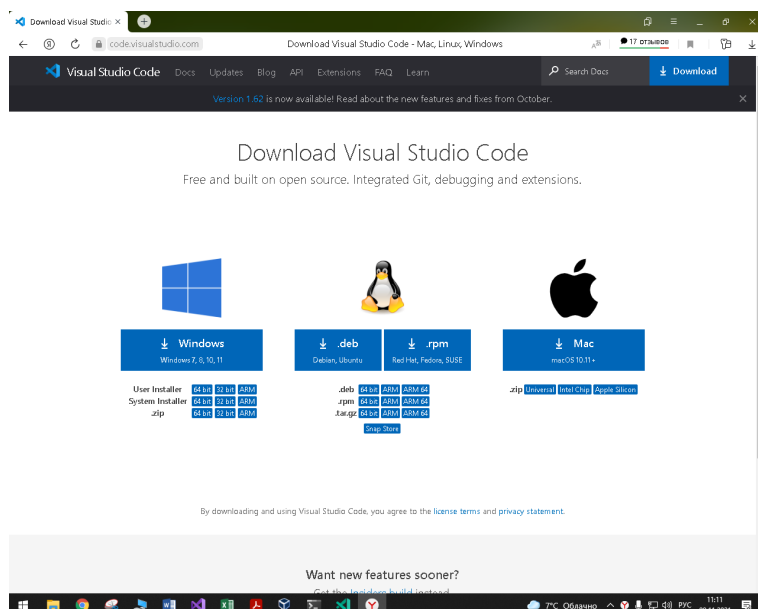
```
$ dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

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>



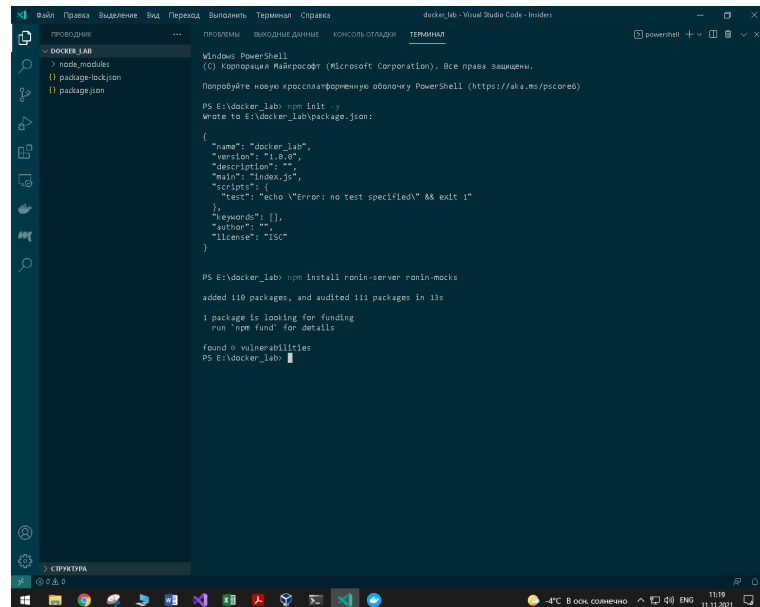
2.2. Перед началом работы проведем проверку работоспособности Docker-контейнер, для этого воспользуемся приложением с сайта по ссылке: <https://docs.docker.com/language/nodejs/build-images/#sample-application>.

В терминале VS Code выполним следующие команды:

```
$ cd E://Lab-3-Docker
```

```
$ npm init -y
```

```
$ npm install ronin-server ronin-mocks
```



```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Получайте новые красочные формы PowerShell (https://aka.ms/powershell)

PS E:\docker_lab> npm init -y
Wrote to E:\docker_lab\package.json:

{
  "name": "docker_lab",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

PS E:\docker_lab> npm install ronin-server ronin-mocks
added 110 packages, and audited 111 packages in 13s

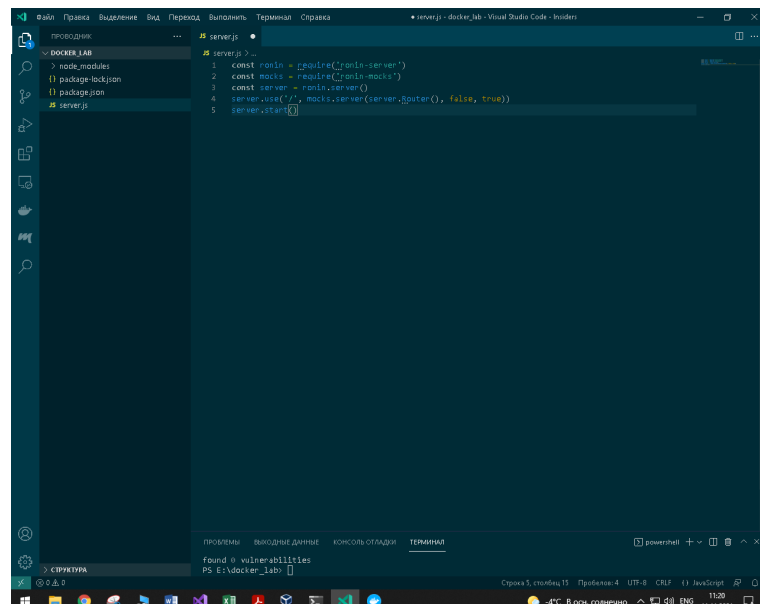
1 package is looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS E:\docker_lab>
```

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()
```

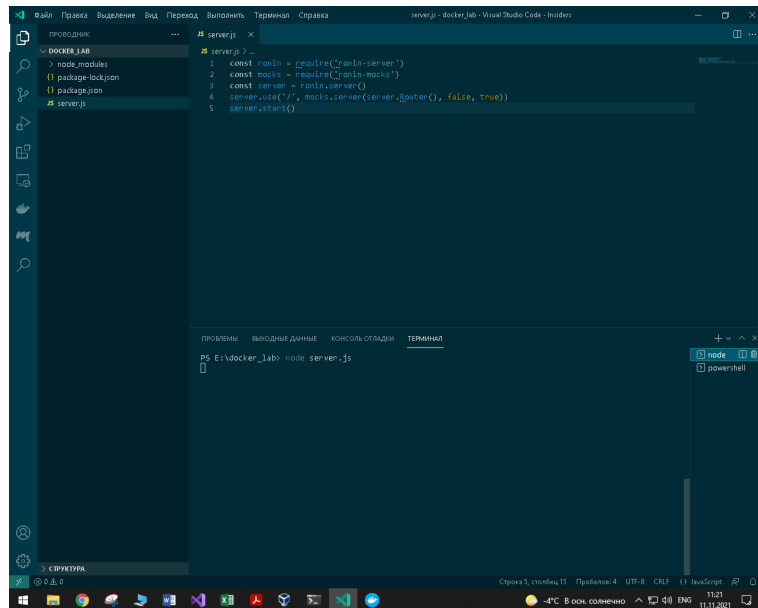


```
server.js
1 const ronin = require('ronin-server')
2 const mocks = require('ronin-mocks')
3 const server = ronin.server()
4 server.use('/', mocks.server(server.Router(), false, true))
5 server.start()
```

```
found 0 vulnerabilities
PS E:\docker_lab>
```

После чего запустим сервер командой

`$ node server.js`

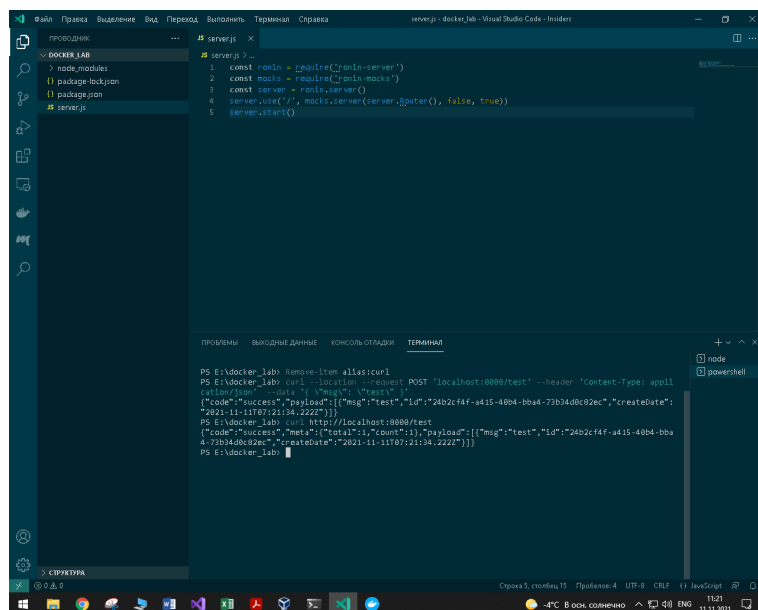


В соседнем терминале выполним следующие команды:

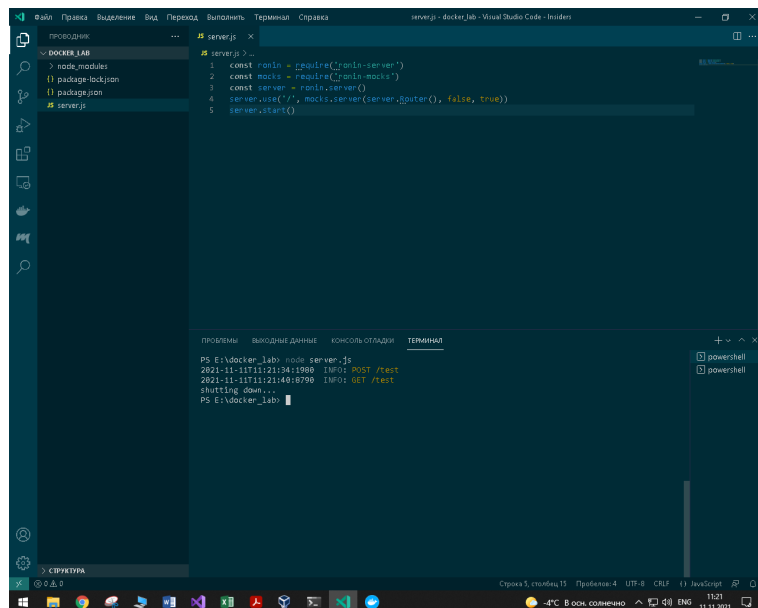
```
$ Remove-item alias:curl
```

```
$ 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

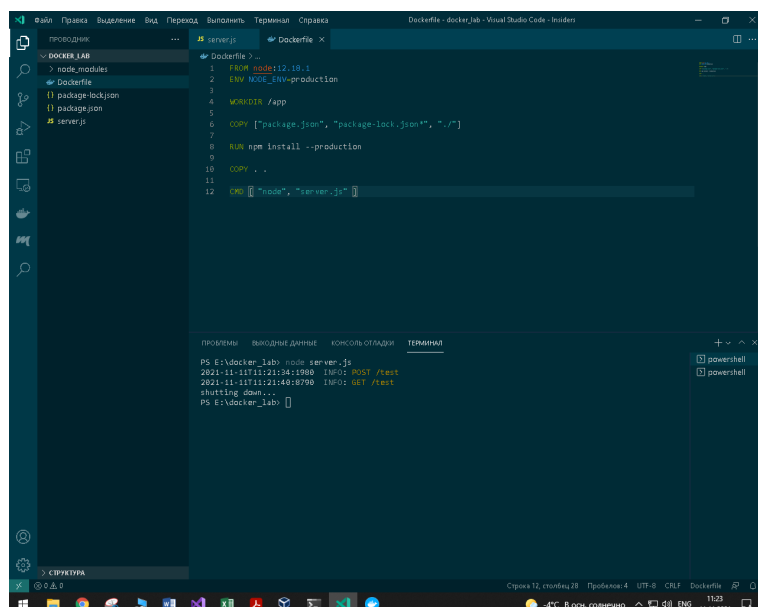
WORKDIR /app

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

RUN npm install --production

COPY . .

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

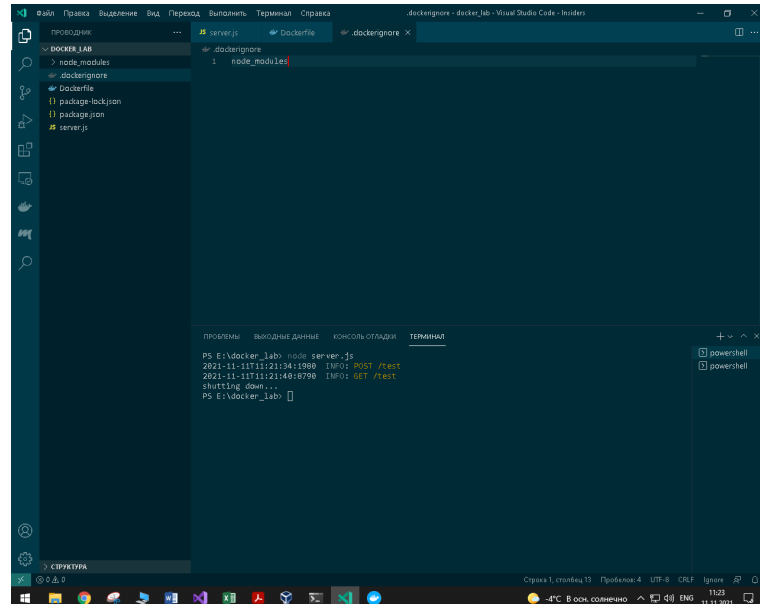


Сохраним данный файл с названием Dockerfile (без расширения), в необходимой папке.

2.4 Создание .dockerignore

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

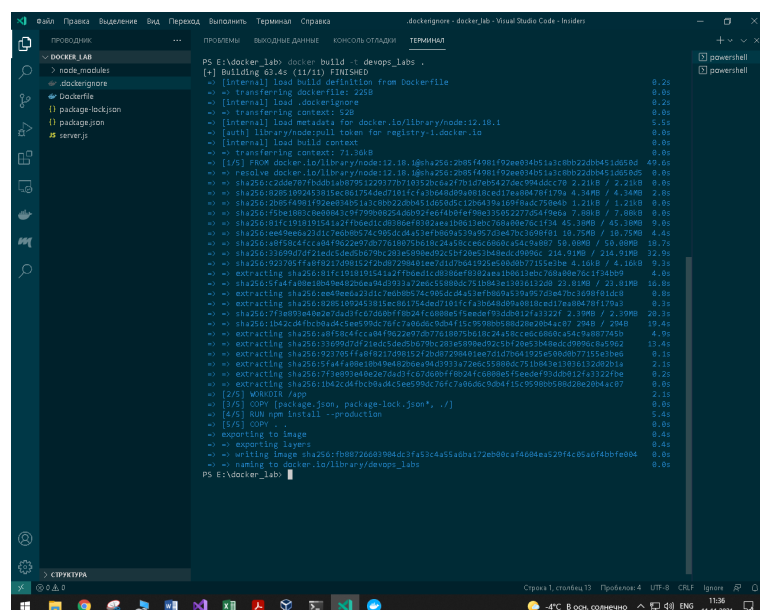
```
node_modules
```



Шаг 3. Создание Docker-образа

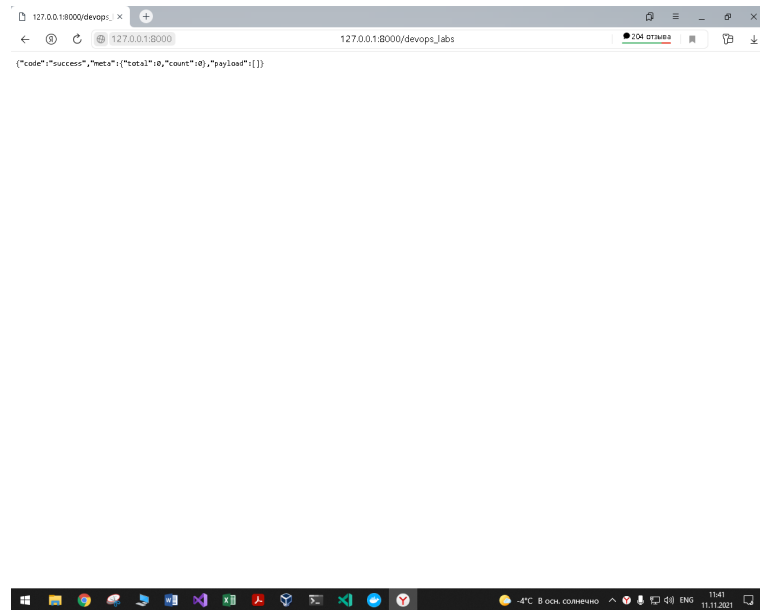
Откроем терминал и перейдем в папку, где был сохранен Dockerfile, после чего создадим Docker –образ, при помощи команды:

```
$ docker build -t devops_labs .
```



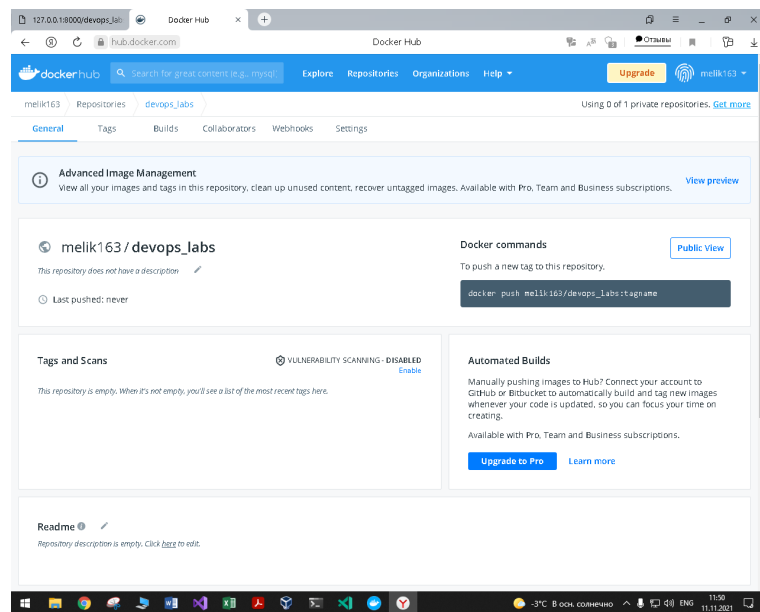
Просмотрим созданный образ Docker, при помощи команды:

```
$ docker image ls
```

Шаг 5. Отправка Docker-образа в Docker Hub

5.1 Создание репозитория в Docker Hub.



5.2 Загрузка на Docker Hub

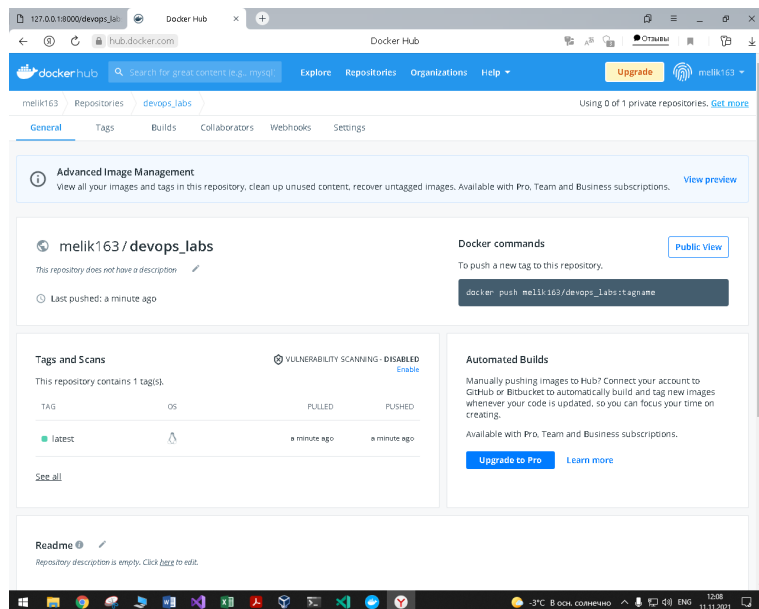
```
$ docker login
```

```
$ docker tag devops_labs melik163/devops_labs
```

```
$ docker push melik163/devops_labs
```

```
PS E:\docker_lab> docker image ls
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
devops_labs   latest    fb872668390   About a minute ago   933MB
PS E:\docker_lab> docker run --rm -it 127.0.0.1:8080 devops_labs
7a1a02f16f772a231580963f294922f7b2c59d0e2f6f6a921e4e805578
PS E:\docker_lab> docker login
Authentication with existing credentials...
Login Succeeded
PS E:\docker_lab> docker tag devops_labs melik163/devops_labs
PS E:\docker_lab> docker push melik163/devops_labs
Using default tag: latest
The push refers to repository [docker.io/melik163/devops_labs]
e904291523a: Pushed
2b0566873c: Pushed
781fc2af07a: Pushed
767ef708aa7: Pushed
d448e44f3c: Mounted from library/node
78b2c950aac: Mounted from library/node
dacaab5344d: Mounted from library/node
bc17c485699: Mounted from library/node
ea95a607f0d: Mounted from library/node
748ffead5c3: Mounted from library/node
6a4c9e92024: Mounted from library/node
230ca356262f: Mounted from library/node
8354d5985537: Mounted from library/node
latest: digest: sha256:d8556c4f218f1574e66720e4d7c264286f57a6fcd89e6ac8c47af7af0b size: 3851
PS E:\docker_lab>
```

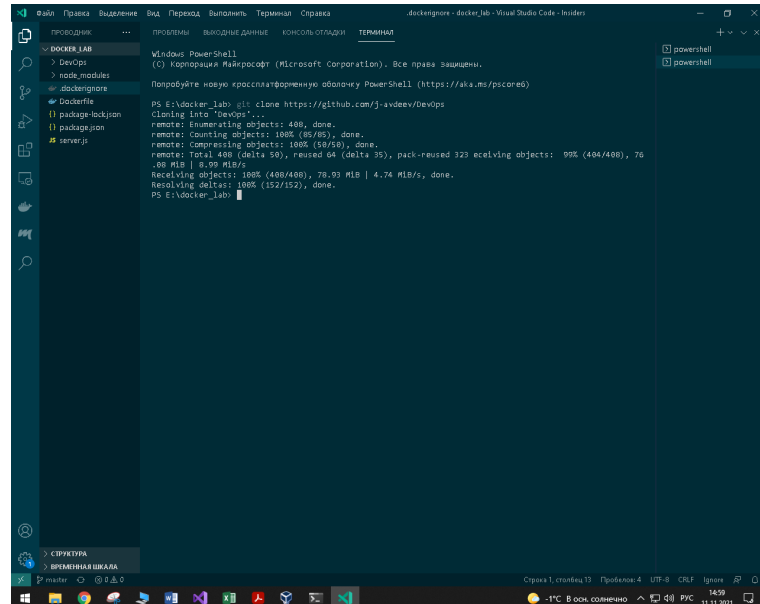
Перейдем в браузер и проверим выгрузку Docker-контейнера на Docker Hub.



ЧАСТЬ 2

Шаг 1. Подготовка данных.

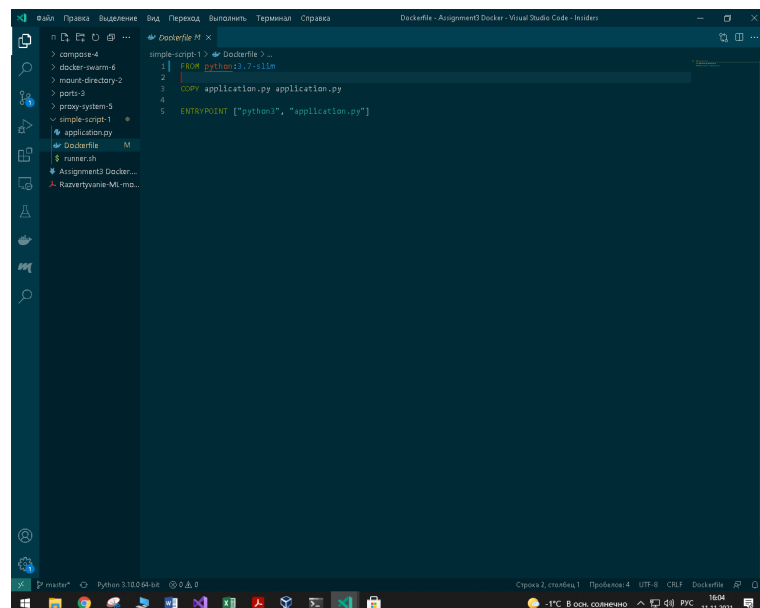
Клонируем репозиторий с сайта GitHub по ссылке <https://github.com/j-avdeev/DevOps>.



Шаг 2. Создать и запустить Docker-контейнер с python-приложением. (Directory simple-script-1).

2.1 Корректировка Dockerfile.

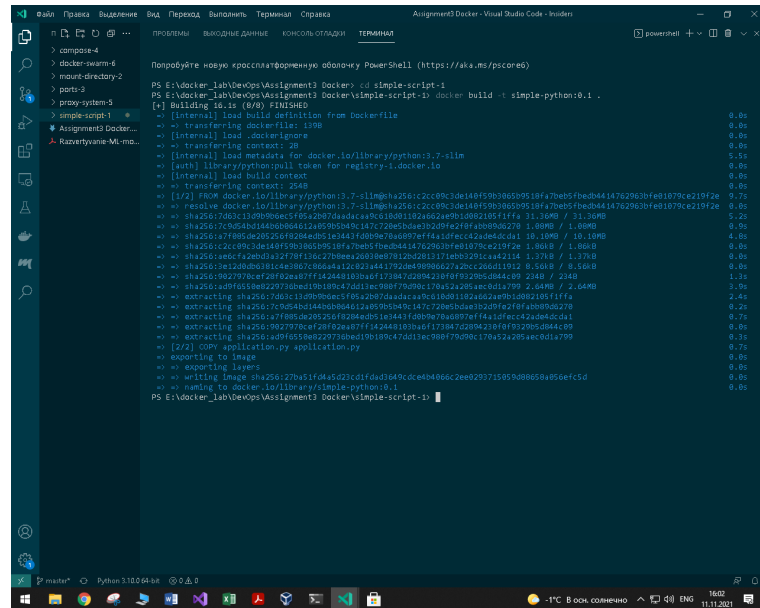
В Dockerfile изменим тег с FROM python:3.7 на FROM python:3.7-slim, поскольку это существенно снизит время загрузки и построения образа.



2.2 Построение Docker-образа.

Для построения Docker-образа воспользуемся командой:

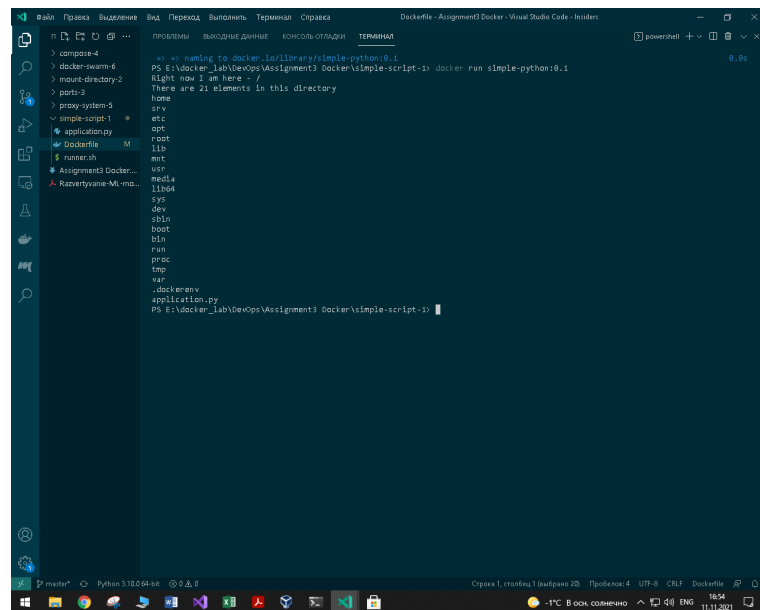
\$ docker build -t simple-python:0.1 .



2.3 Запуск построенного образа.

Для запуска построенного Docker-образа воспользуемся командой:

\$ docker run simple-python:0.1

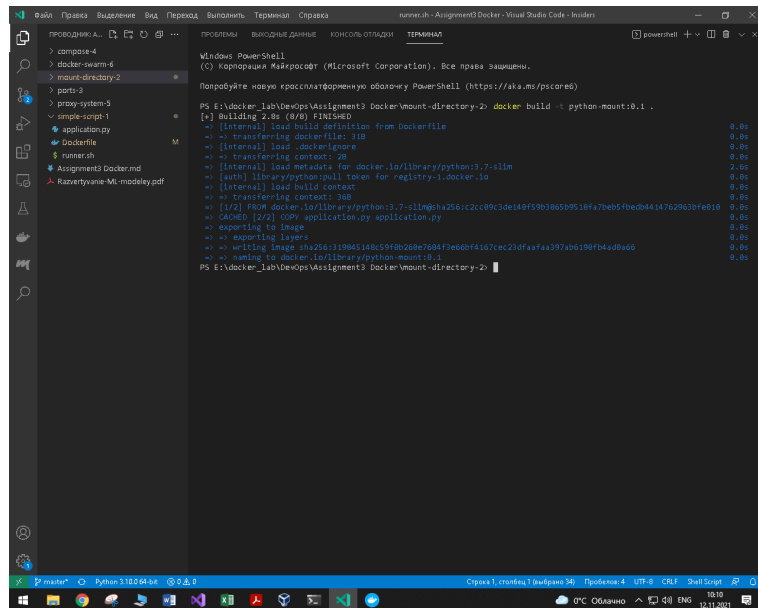


Шаг 3. Создание и запуск Docker- контейнера с установкой каталога.

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 .



```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попытка запуска новой кроссплатформенной оболочки PowerShell (https://aka.ms/powershell)

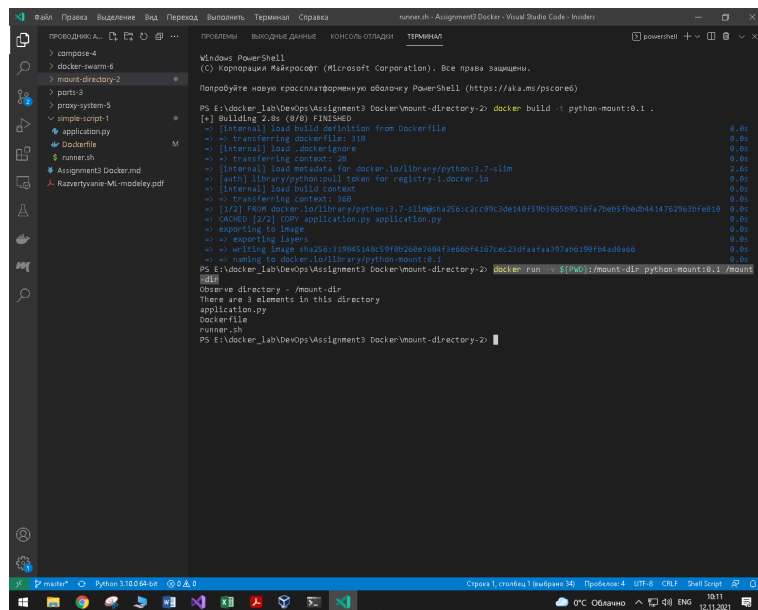
PS E:\docker_lab\DevOps\Assignment3 Docker\mount-directory-2> docker build -t python-mount:0.1 .
[*] Building 2.8s (8/8) FINISHED
-> [internal] load build definition from Dockerfile
=> transferring dockerfile: 318
=> [internal] load dockerignore
=> transferring context: 28
=> [internal] load metadata for docker.io/library/python:3.7-slim
=> [auth] library/python:pull token for registry-1.docker.io
=> [internal] load build context
=> transferring context: 380
=> [1/2] FROM docker.io/library/python:3.7-slim@sha256:c2cc9c3de140f59b38659518fa79eb5fbed4414762963bfe10
=> CACHED [2/2] COPY application.py application.py
=> exporting to image
=> exporting layers
=> writing image sha256:319845148c59f826de768f2e6b0f4167cc23d4fa397ab619f64ad8a66
=> naming to docker.io/library/python-mount:0.1
PS E:\docker_lab\DevOps\Assignment3 Docker\mount-directory-2>
```

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`



```
PS E:\docker_lab\DevOps\Assignment3 Docker\mount-directory-2> docker run -v ${PWD}:/mount-dir python-mount:0.1 /mount-dir
Observe directory - /mount-dir
There are 3 elements in this directory
application.py
dockerfile
runner.sh
PS E:\docker_lab\DevOps\Assignment3 Docker\mount-directory-2>
```

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 .`

```
PS E:\docker_lab\DevOps\Assignment3 Docker\ports-3> docker build -t python-server:0.1 .
[*] Building 3.3s (8/8) FINISHED
-> [internal] load build definition from Dockerfile
-> => transferring dockerfile: 185B
-> [internal] load dockerignore
-> => transferring context: 2B
-> [internal] load metadata for docker.io/library/python:3.7-slim
-> [auth] library/pythonpull token for registry-1.docker.io
-> CACHED [1/3] FROM docker.io/library/python:3.7-slimsha256:c2cc89c3de149f9703805b9518f479eb5fbed441476296
-> [2/3] RUN mkdir /sync-folder
-> [2/3] WORKDIR /sync-folder
-> exporting to image
-> exporting layers
-> writing image sha256:b750b6ef9608291ec56578768d24cf841918c479945626958c49f6edc4b29
-> naming to docker.io/library/python-server:0.1
PS E:\docker_lab\DevOps\Assignment3 Docker\ports-3>
```

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`

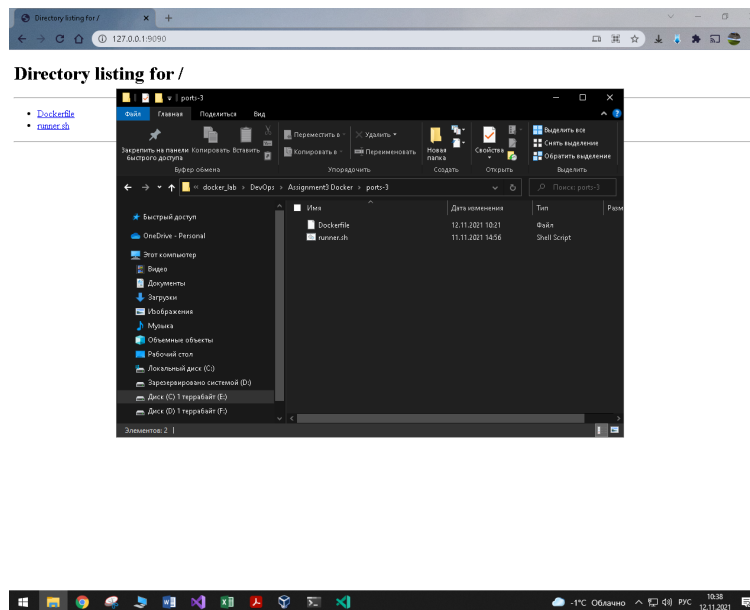
```
PS E:\docker_lab\DevOps\Assignment3 Docker\ports-3> docker build -t python-server:0.1 .
[*] Building 3.3s (8/8) FINISHED
-> [internal] load build definition from Dockerfile
-> => transferring dockerfile: 185B
-> [internal] load dockerignore
-> => transferring context: 2B
-> [internal] load metadata for docker.io/library/python:3.7-slim
-> [auth] library/pythonpull token for registry-1.docker.io
-> CACHED [1/3] FROM docker.io/library/python:3.7-slimsha256:c2cc89c3de149f9703805b9518f479eb5fbed441476296
-> [2/3] RUN mkdir /sync-folder
-> [2/3] WORKDIR /sync-folder
-> exporting to image
-> exporting layers
-> writing image sha256:b750b6ef9608291ec56578768d24cf841918c479945626958c49f6edc4b29
-> naming to docker.io/library/python-server:0.1
PS E:\docker_lab\DevOps\Assignment3 Docker\ports-3> 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/>

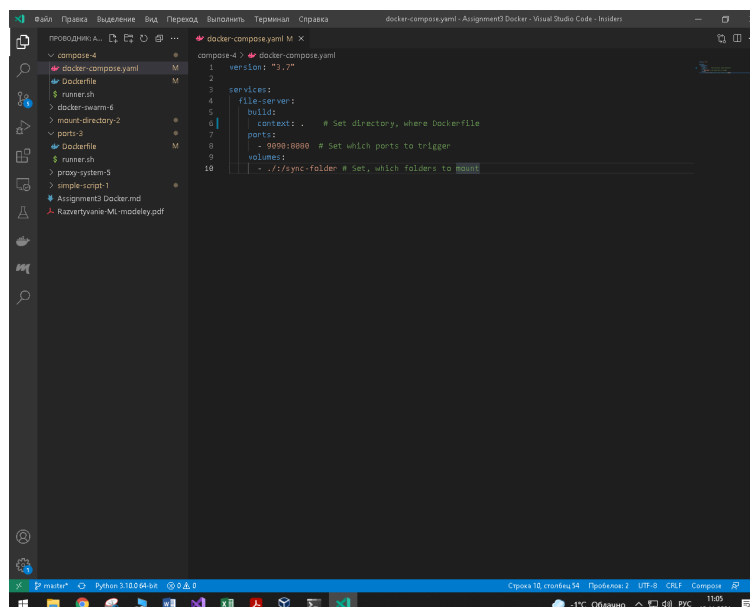
А заодно проверим расположение в директории на хосте.



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

5.1 Корректировка yaml файла.

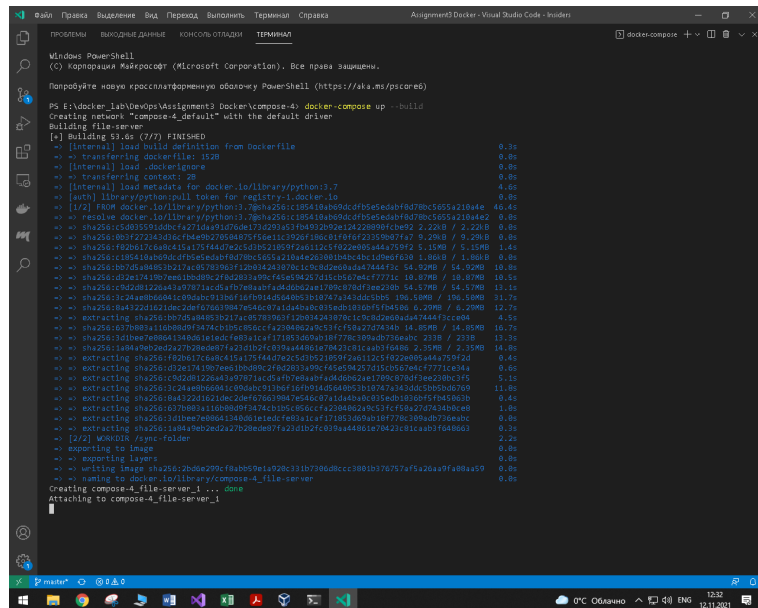
Перед началом выполнения данного задания необходимо исправить неверный docker-compose.yaml файл. А именно исправить значение ключа context: context: .



5.2 Сборка Docker-образа и запуск Docker-контейнера

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

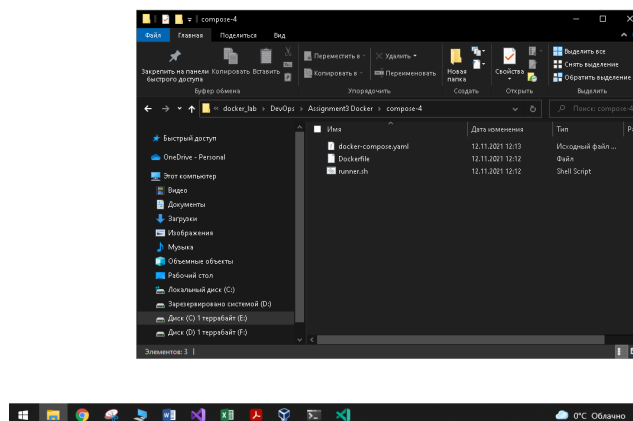
\$ docker-compose up --build



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/>

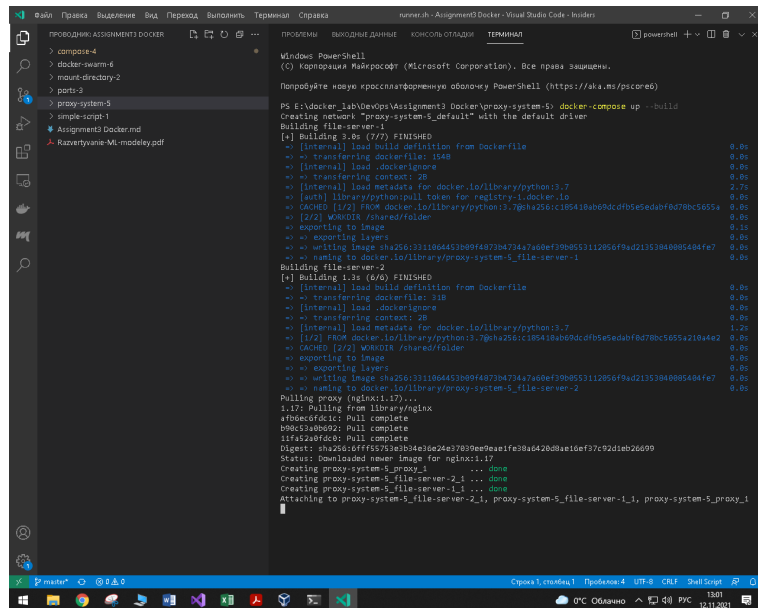


Шаг 6. Создание и запуск Docker-контейнера с распределением нагрузки.

6.1 Создание и запуск Docker-контейнера.

Для одновременного создания и запуска Docker-контейнера воспользуемся командой:

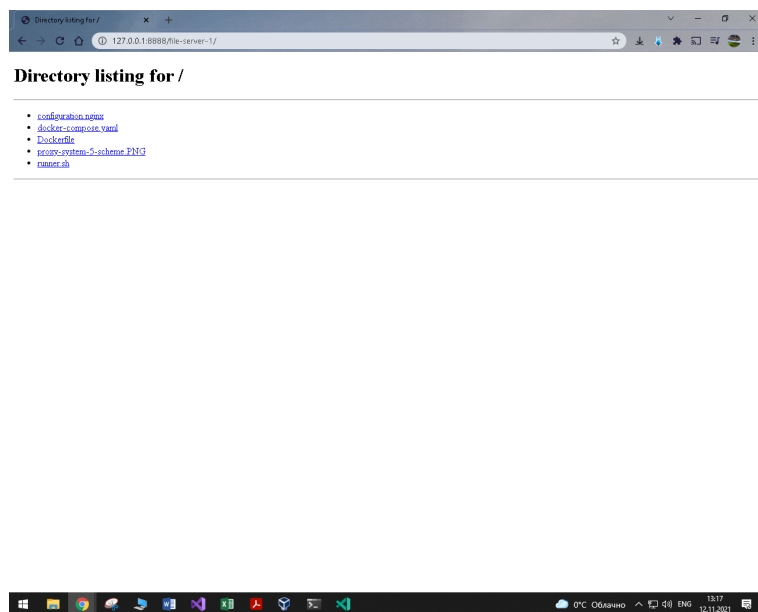
`$ docker-compose up --build`



6.2 Проверка работоспособности.

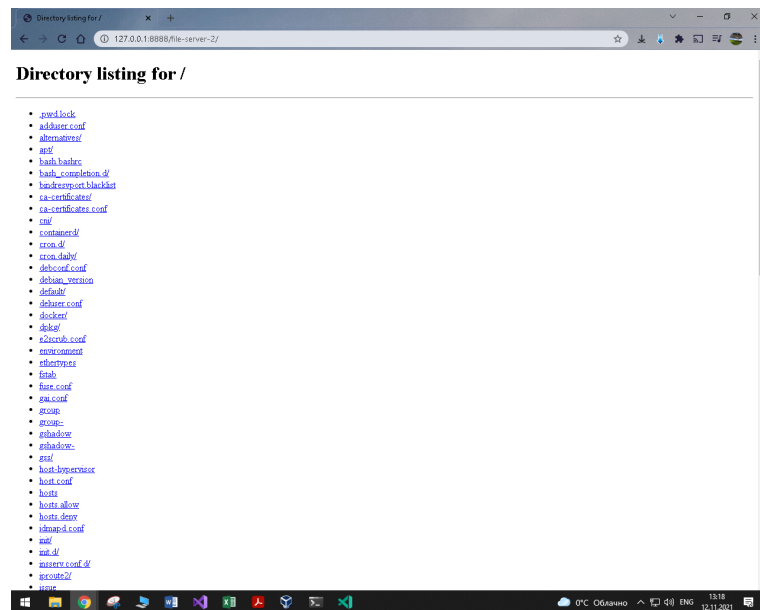
Для проверки первого сервера перейдем в браузер и в адресной строке введем следующий адрес:

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



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

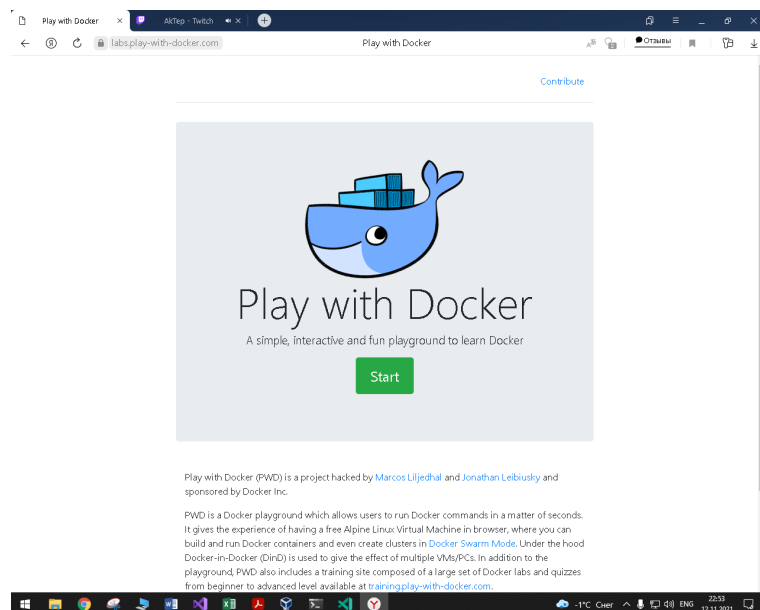
<http://127.0.0.1:8888/file-server-2/>



Шаг 7. Создать и запустить Docker-Swarm конфигурацию.

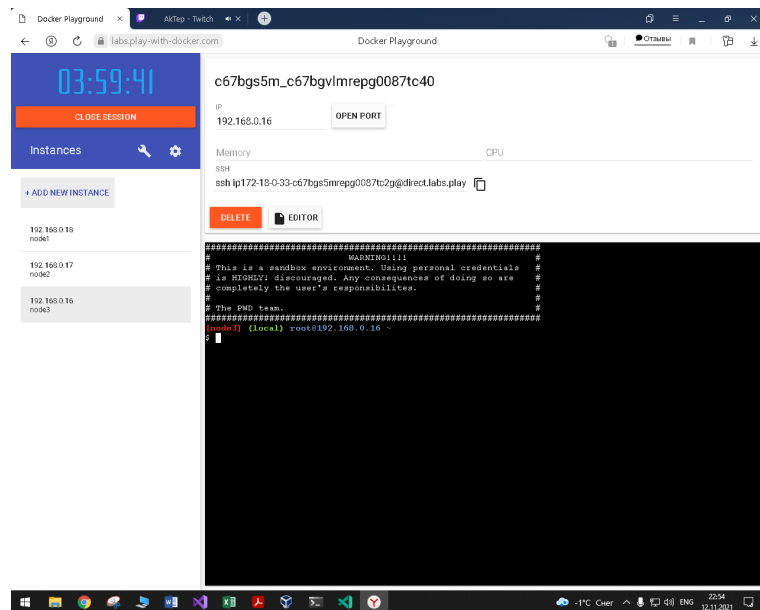
7.1 Play-with-docker.

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



7.2 Создание node.

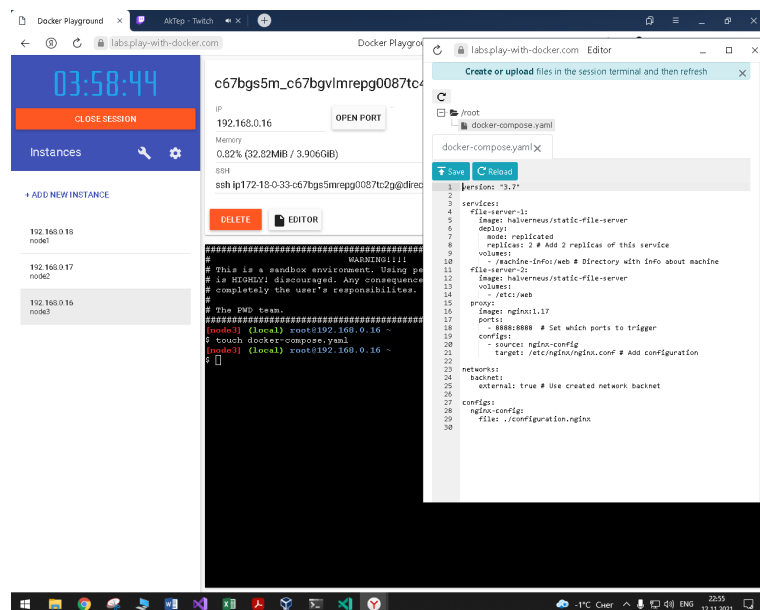
Создадим node в количестве трех штук. Для этого достаточно нажать три раза на «Add new instance».



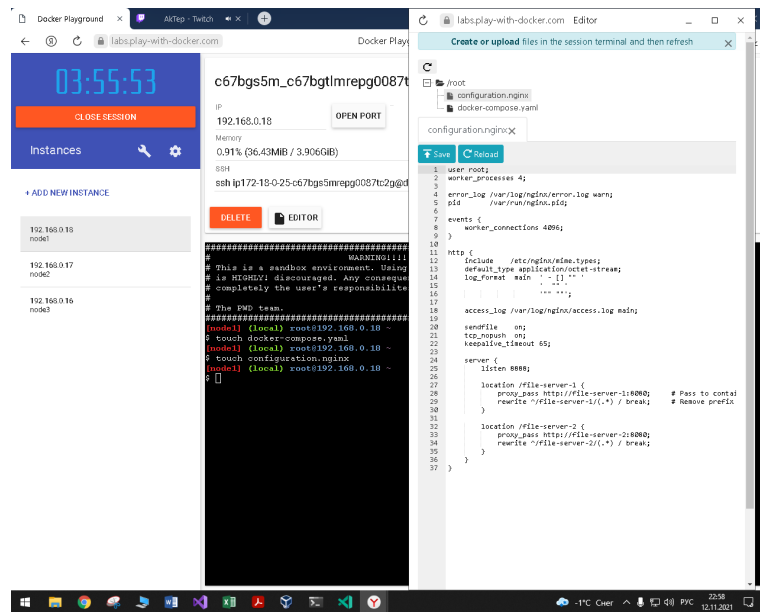
7.1 Добавление файлов и назначение manager mode.

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

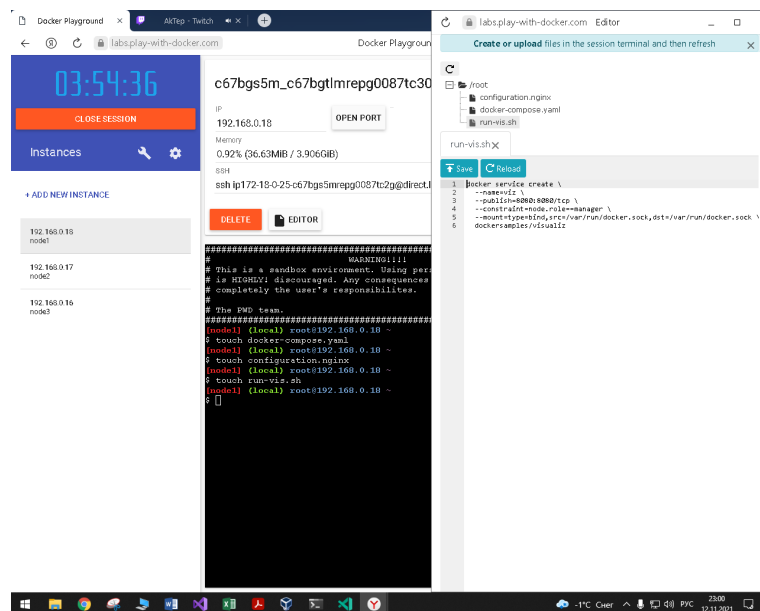
\$ touch docker-compose.yaml



\$ touch configuration.nginx

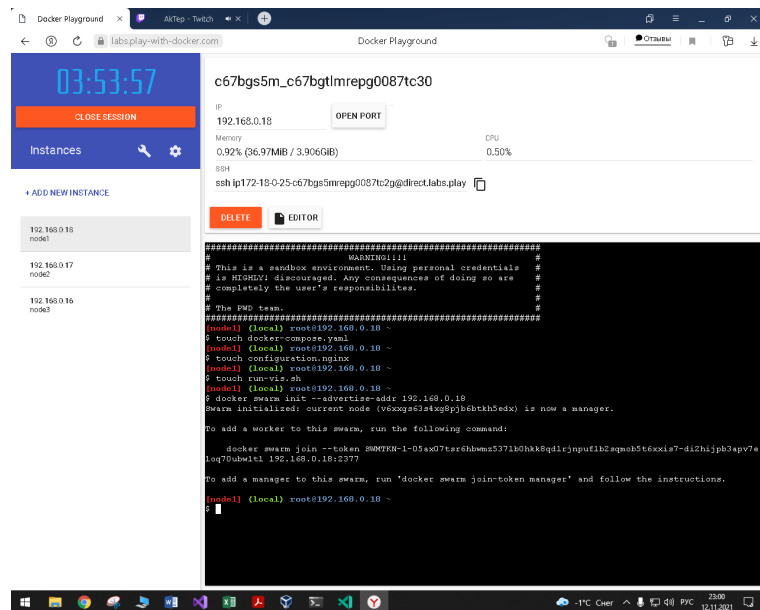


\$ touch run-vis.sh



Node1 присвоим manager mode

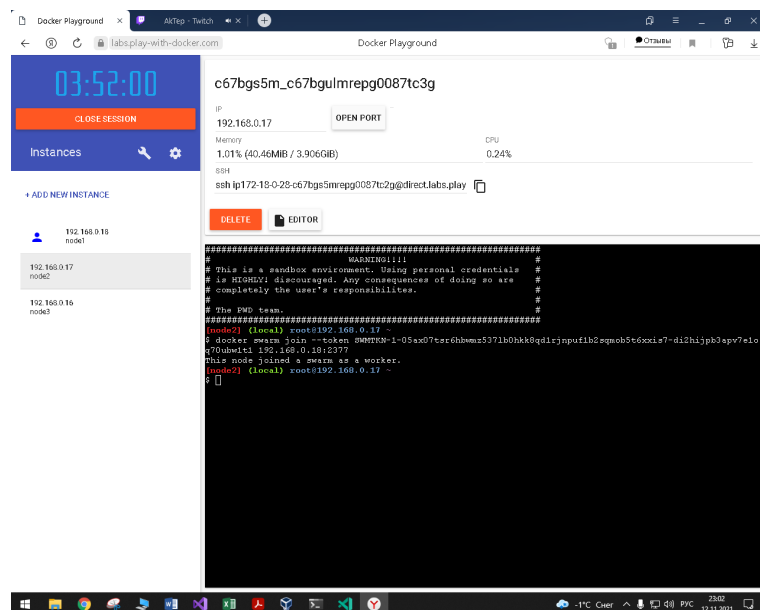
\$ docker swarm init --advertise-addr 192.168.0.18



7.2 Присоедините оставшихся node к главной.

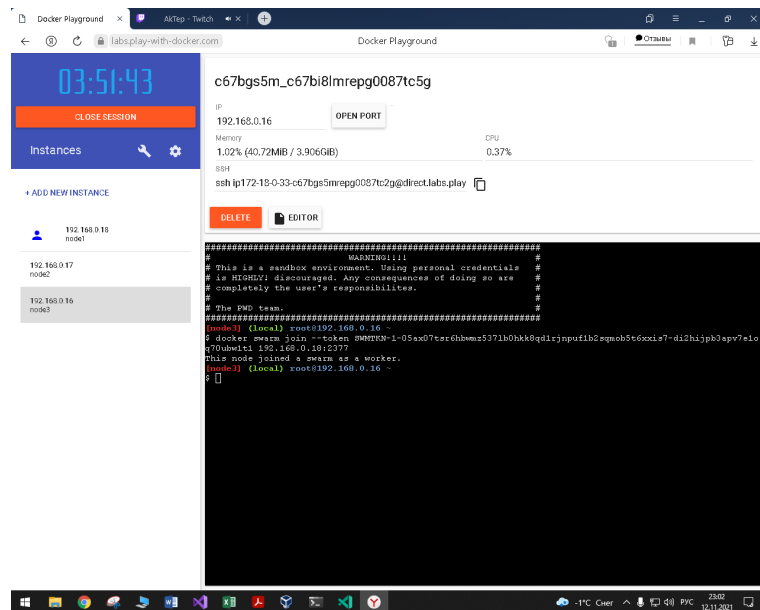
7.2.1 Присоединение node2:

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



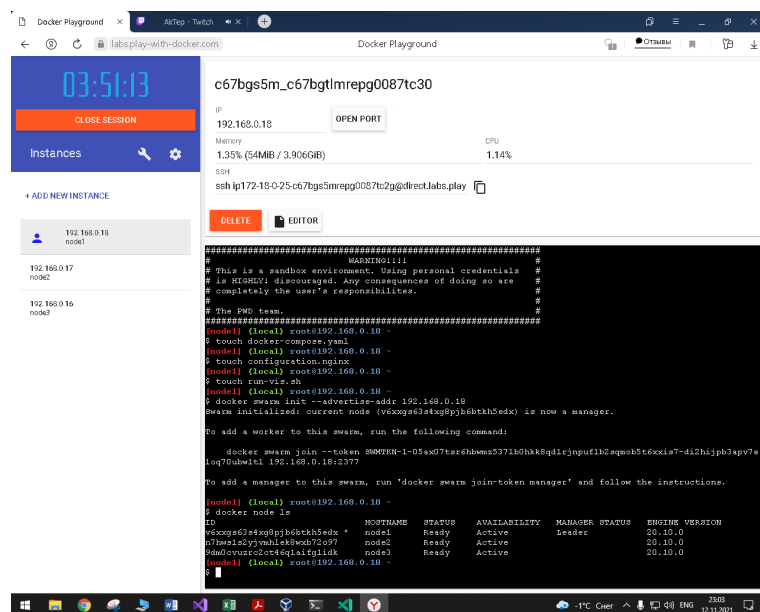
7.2.2 Присоединение node3:

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



7.3 Запуск Docker Swarm Visualizer на главной node.

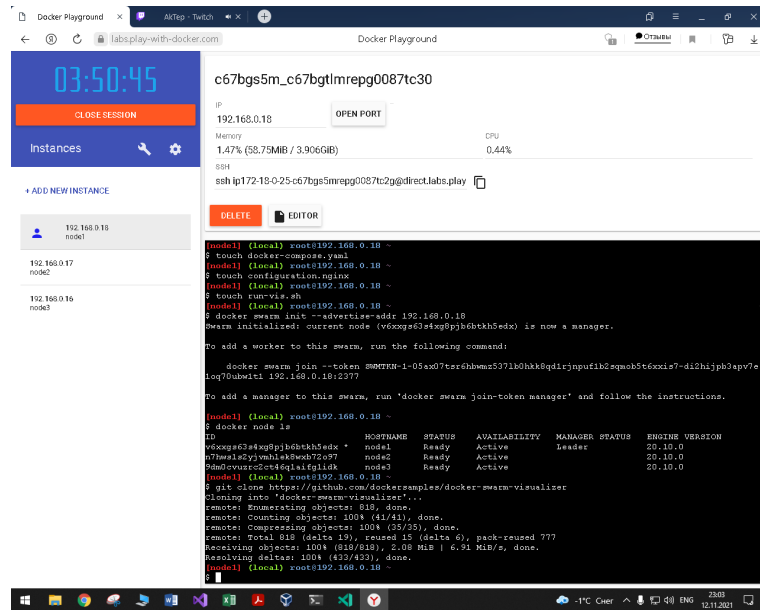
Проверим подключение всех node.



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

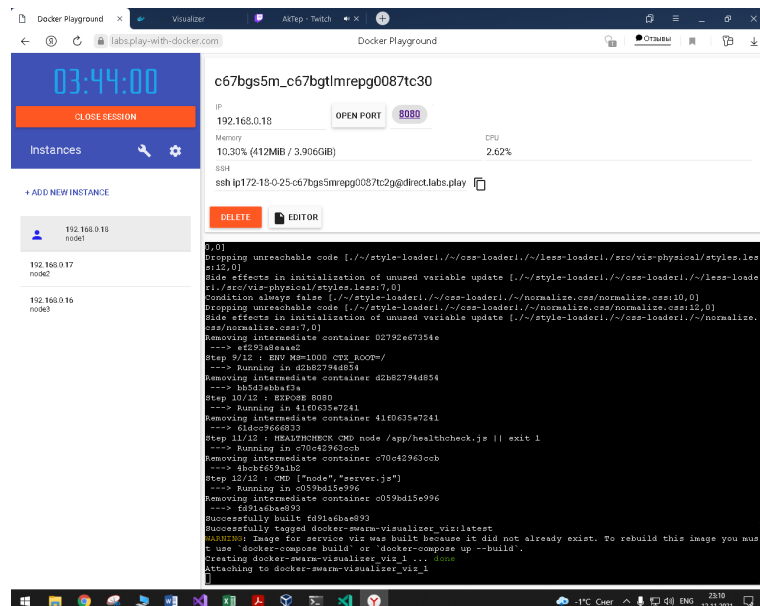
Перейдем в папку в которой расположен docker-swarm-visualizer:

```
$ cd docker-swarm-visualizer
```

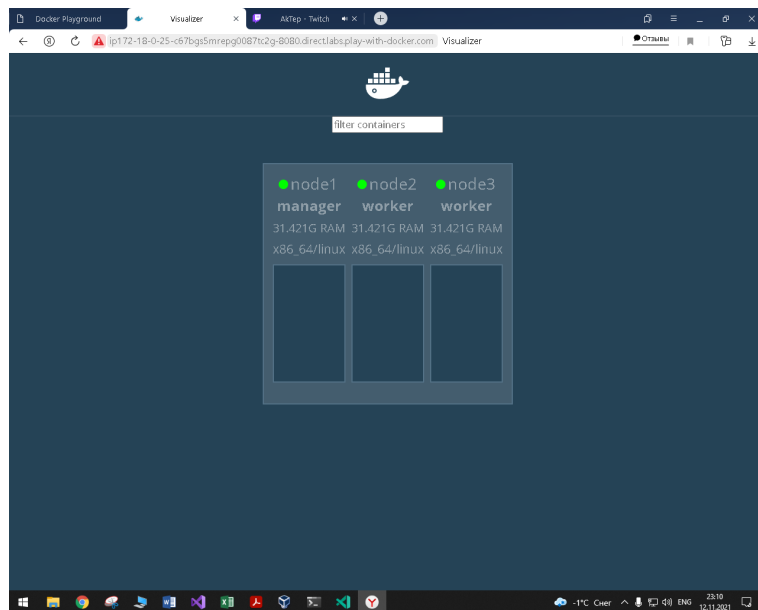
И поднимем его следующей командой

```
$ 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.

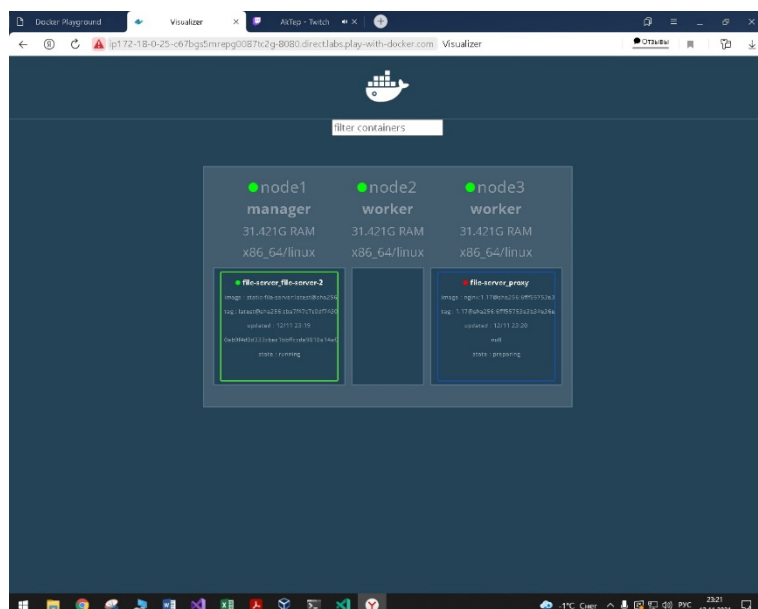
Для развертывания приложения необходимо выполнить следующий набор команд:

```
$ docker network create --driver overlay backnet
```

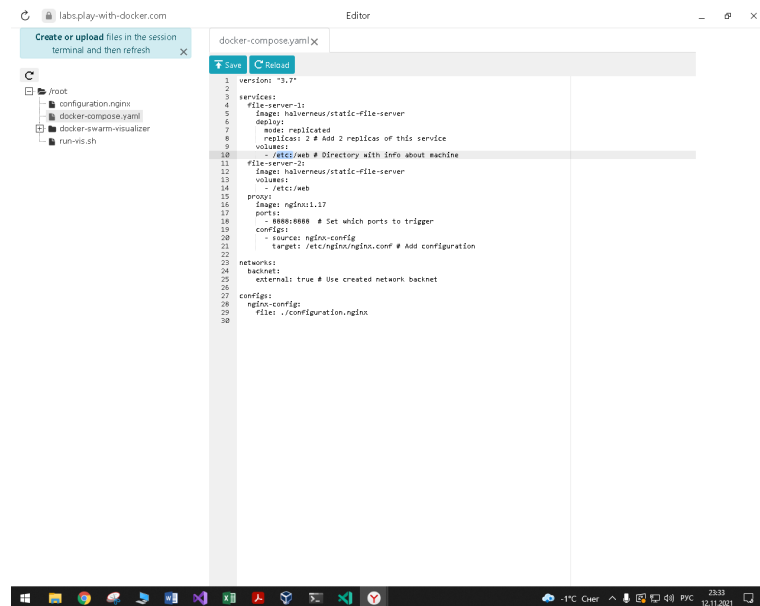
```
$ cd ~
```

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

После чего проверим работоспособность.

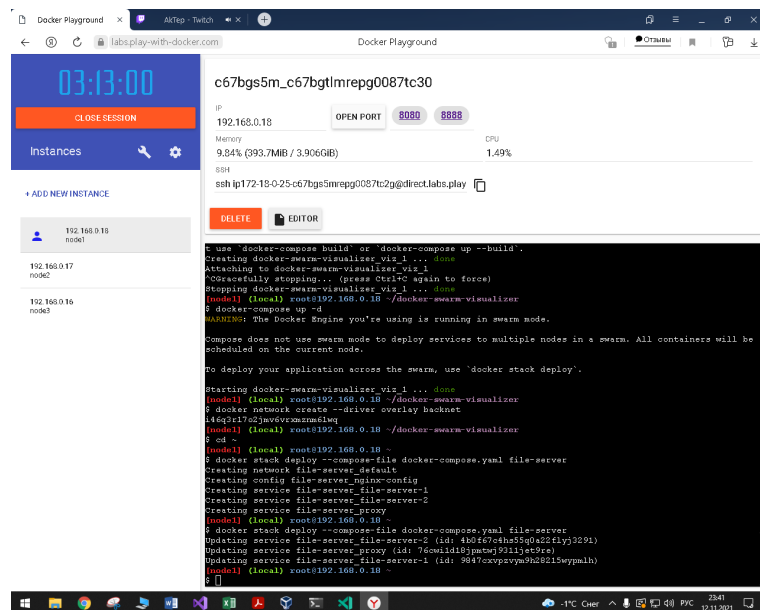


Вроде работает, но не совсем ерно. Зайдем в docker-compose.yaml файл и изменим /machine-info на /etc, поскольку директории /machine-info у нас нет.

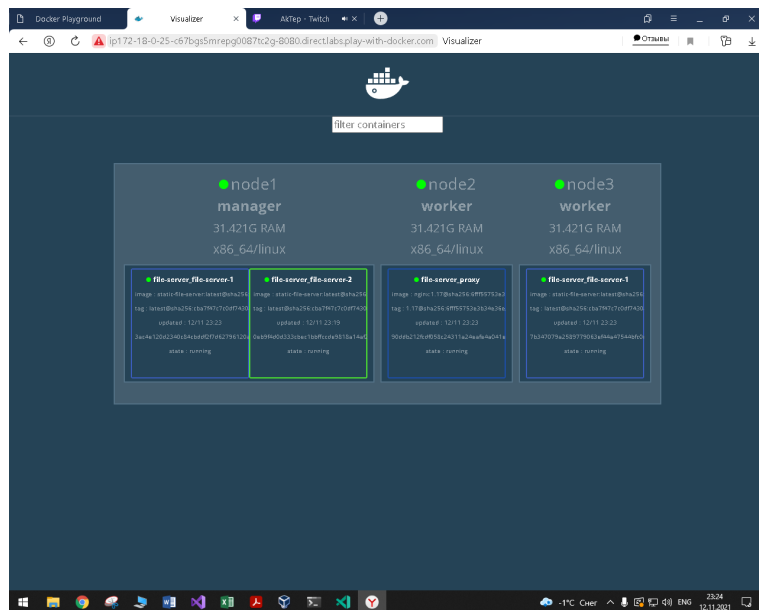


После сохранения повторим команду:

`$ docker stack deploy --compose-file docker-compose.yml file-server`



И проверим работоспособность.



Теперь работает.

ЗАКЛЮЧЕНИЕ

В заключение лабораторной работы были изучены основные утилиты Docker для работы с контейнерными приложениями, а также Docker Compose и Docker Swarm. Было создано и протестировано Node.js приложение. Был создан и загружен в Docker Hub Docker-образ на основе приложения. Приложение python также было упаковано в контейнер с подключением томов, перенаправлением портов и прокси-серверами. Несколько контейнеров Docker Swarm были настроены и развернуты в Play with Docker.