

Лабораторная работа №6

Docker

Цель

Исследовать основные возможности, предоставляемые приложениями, использующими и создающими контейнеры. Изучить основы контейнеризации с использованием Docker. Получить практические навыки по созданию, управлению и деплою контейнеров.

Краткие теоретические сведения о Docker

Docker — это платформа для разработки, доставки и запуска приложений в контейнерах. Контейнеры позволяют упаковать приложение и все его зависимости в единый единичный блок, который можно легко перемещать между различными средами и запускать на любом сервере, который поддерживает Docker.

Основные концепции Docker:

1. Контейнеризация:

— Контейнеризация — это технология, которая позволяет изолировать приложения и их зависимости в отдельных контейнерах. Это обеспечивает согласованное окружение для разработки, тестирования и запуска приложений.

2. Docker-образ (Image):

— Docker-образ — это неизменяемый шаблон, который используется для создания контейнеров. Образ включает все необходимые компоненты для запуска приложения: код, библиотеки, зависимости и конфигурации.

3. Docker-контейнер (Container):

– Контейнер — это запущенный экземпляр Docker-образа. Контейнеры являются легковесными и изолированными, что позволяет запускать несколько контейнеров на одном хосте без конфликта.

4. Dockerfile:

– Dockerfile — это текстовый файл, содержащий инструкции для сборки Docker-образа. Каждая инструкция описывает шаг сборки, например, копирование файлов, установка зависимостей или запуск команд.

5. DockerHub:

– DockerHub — это облачный реестр Docker-образов, где можно хранить и делиться своими образами. Это позволяет легко распространять и повторно использовать образы в различных проектах и командах.

6. Оркестрация контейнеров:

– Оркестрация — это процесс управления и координации нескольких контейнеров на кластере серверов. Инструменты оркестрации, такие как DockerCompose и Kubernetes, помогают автоматизировать развертывание, масштабирование и управление контейнерами.

Преимущества использования Docker:

1. Портативность:

– Контейнеры можно запускать на любой системе, поддерживающей Docker, что упрощает переносимость приложений между различными средами (разработка, тестирование, производство).

2. Изоляция:

- Контейнеры изолированы друг от друга и от хостовой системы, что обеспечивает безопасность и независимость приложений.

3. Упрощение DevOps процессов:

- Docker упрощает процессы разработки, тестирования и развертывания приложений, позволяя разработчикам и операционным командам эффективно взаимодействовать.

4. Быстрое масштабирование:

- Контейнеры позволяют быстро масштабировать приложения, добавляя новые экземпляры контейнеров по мере необходимости.

Использование Docker значительно упрощает управление инфраструктурой и обеспечивает гибкость при работе с приложениями.

Минусы Docker:

– Сложность управления контейнерами в больших масштабах:

В больших инфраструктурах управление множеством контейнеров может быть сложным и требовать использования дополнительных инструментов оркестрации, таких как Kubernetes, которые сами по себе могут быть сложными в настройке и управлении.

– Проблемы с безопасностью:

Контейнеры, работающие на общем ядре операционной системы, могут иметь уязвимости, которые позволят одному контейнеру нарушить безопасность других контейнеров или хостовой системы. Это требует дополнительных мер по обеспечению безопасности.

– Ограниченная производительность:

Несмотря на то, что контейнеры менее ресурсоемкие, чем виртуальные машины, они все равно имеют накладные расходы на изоляцию, что может

снизить производительность по сравнению с нативным выполнением приложений на хосте.

– **Совместимость и зависимости:**

Не все приложения могут быть легко контейнеризованы, особенно если они сильно зависят от определенной операционной системы или среды исполнения. Это может потребовать значительных усилий по адаптации.

– **Проблемы с сетевой конфигурацией:**

Сетевые настройки Docker могут быть сложными и иногда ограниченными. Особенно это касается настройки многослойных сетей и обеспечения безопасности в сетевых взаимодействиях между контейнерами.

– **Ограниченные ресурсы хоста:**

Контейнеры используют ресурсы хостовой системы, и при большом количестве контейнеров может возникнуть проблема исчерпания ресурсов, таких как память, процессорное время и дисковое пространство.

– **Отсутствие графического интерфейса:**

По умолчанию Docker не предоставляет графический интерфейс для управления контейнерами, что может усложнять управление для пользователей, предпочитающих графические инструменты.

– **Проблемы с управлением состоянием и данными:**

Управление состоянием и данными внутри контейнеров может быть сложным, особенно если контейнеры предполагается запускать на разных хостах или мигрировать между серверами.

Заключение:

Docker является мощным инструментом для контейнеризации и управления приложениями, но его использование требует учета некоторых недостатков и проблем. При правильном подходе и использовании дополнительных инструментов для управления и оркестрации контейнеров, многие из этих недостатков можно минимизировать или устранить.

Задание для лабораторной работы:

Установка DockerDesktop

Windows

Системные требования

1. В BIOS включена поддержка виртуализации Intel VT-x/AMD-V.
2. Установлена Windows редакции Pro или Enterprise.
3. Компьютер работает под управлением Windows 10, обновлен до версии 2004, сборки 18362 или более поздней версии.

Настройка WSL

1. В мастере "Включение и выключение компонентов Windows" включить компоненты

"Платформа виртуальной машины", "Hyper-V" и "Подсистема Windows для Linux" или

выполнить в Powershell следующие команды.
Перезагрузить компьютер.

Powershell

```
dism.exe /online /enable-feature /featurename:Microsoft-Hyper-V /all  
/norestart
```

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

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

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

2. Установить пакет обновлений ядра
(https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi).

После установки перезагрузить компьютер.

3. В Powershell выполнить обновление WSL командой `wsl --update`

4. Подтвердить правильность настройки установив, например, Debian через `wsl --install -d Debian`

Установка DockerDesktop

1. Скачать установщик с официального сайта

Docker (<https://www.docker.com/products/docker-desktop/>).

2. Запустить установщик.

3. Принять предлагаемую конфигурацию и нажать на "Ok" (рисунок 1.1).

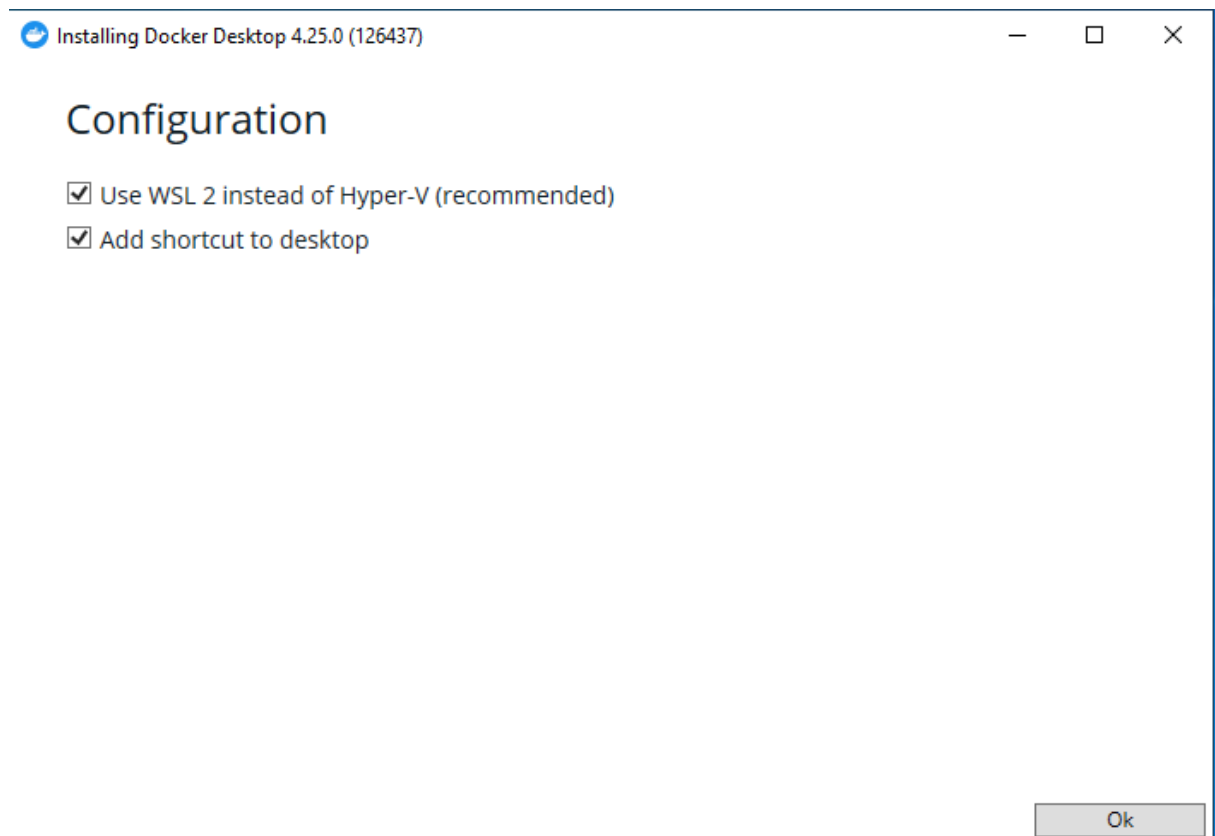


Рисунок 1.1 – Предлагаемая конфигурация

4. После установки согласиться с перезагрузкой компьютера (рисунок 1.2).

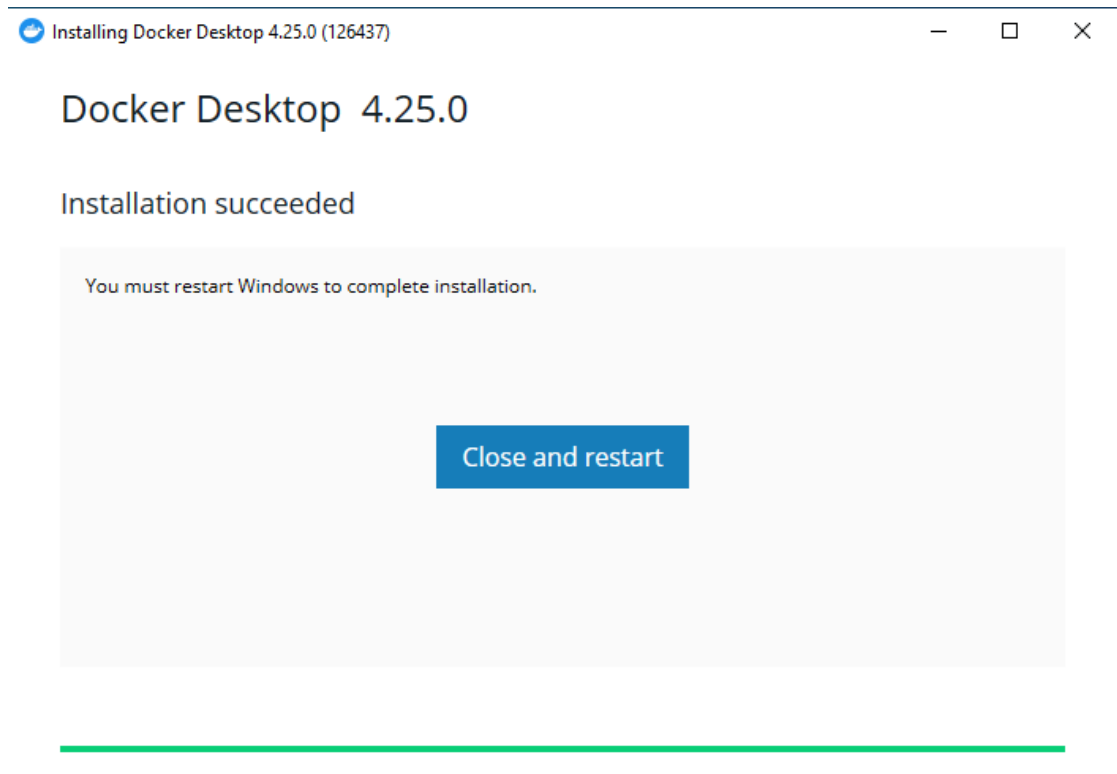


Рисунок 1.2 – Соглашение с перезагрузкой компьютера

5. Запустить приложение DockerDesktop.
7. При согласии принять лицензионное соглашение.
8. Завершить установки, применив рекомендуемые настройки (рисунок 1.3).

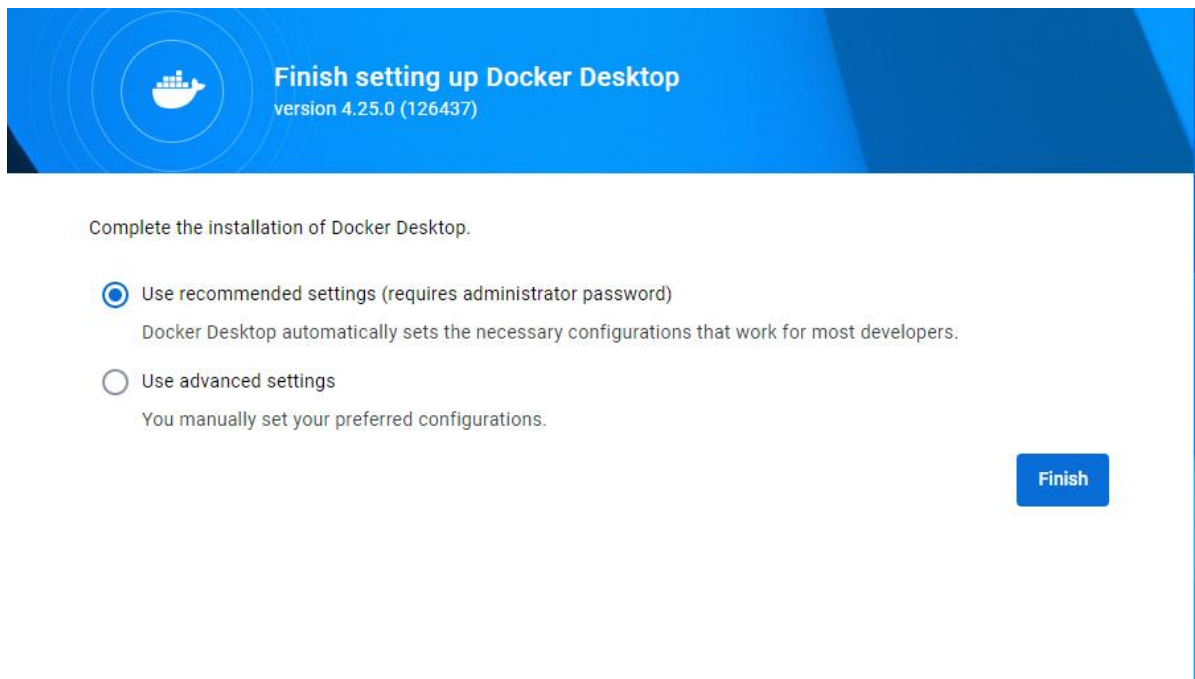


Рисунок 1.3 – Завершение установки

Linux

Системные требования

1. 64битный процессор, поддерживающий виртуализацию.

Установка DockerDesktop

1. Скачать установщик с официального сайта Docker (<https://www.docker.com/products/docker-desktop/>).

2. Установить пакет:

Shell

Debian, Ubuntu

```
sudo apt install docker-desktop-*.deb
```

Fedora

```
sudo dnf config-manager
```

--add-repo

```
https://download.docker.com/linux/fedora/docker-ce.repo
```

```
sudo dnf install -y docker-desktop-*.rpm
```


~~Недостающие пакеты будут автоматически установлены.~~

~~3. Запустить приложение DockerDesktop.~~

~~4. При согласии принять лицензионное соглашение.~~

Источники

1. Установка и настройка WSL:

<https://learn.microsoft.com/en-us/windows/wsl/install-manual>

2. Установка и настройка DockerDesktop для Windows:

<https://>

Работа Docker Desktop

Обзор:

На рисунке 1.4 представлено главное окно DockerDesktop. В левой части экрана:

- * Containers - управление контейнерами;
- * Images - управление образами;
- * Volumes - управление разделами;
- * DevEnvironments - создание и настройка среды разработки на основе контейнеризации;
- * DockerScout - анализ образов на наличие уязвимостей;
- * Extensions - управление расширениями для DockerDesktop.

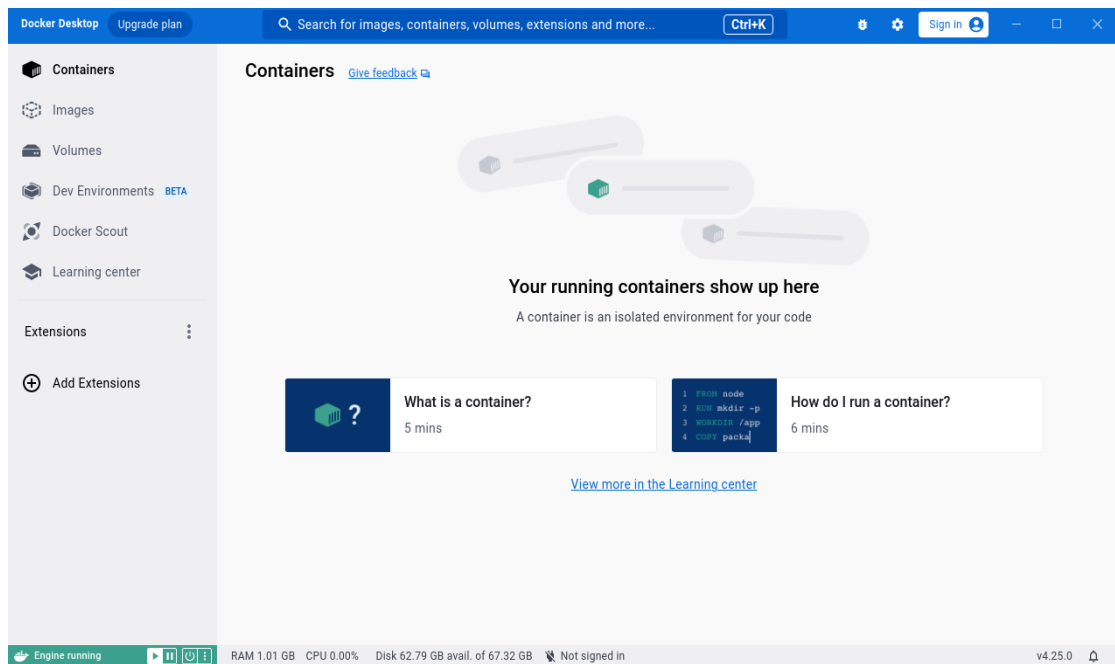


Рисунок 1.4 – Главное окно DockerDesktop

Раздел «Образы» (рисунок 1.5) содержит:

- * Local - управление локальными образами;
- * Hub - управление собственными образами в DockerHub.

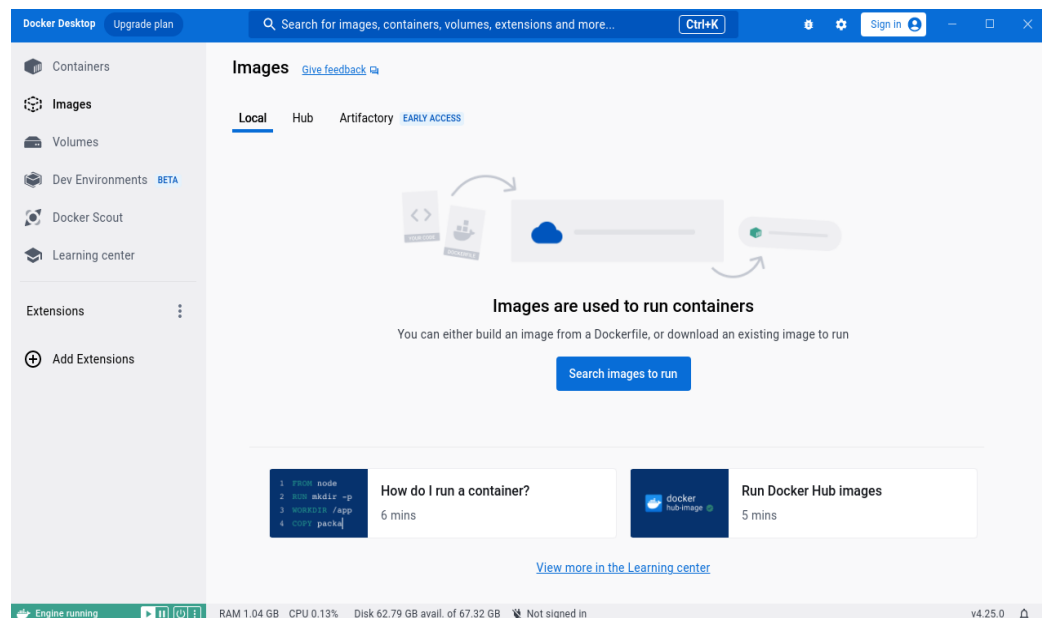


Рисунок 1.5 – Раздел «Образы»

Для поиска образов можно воспользоваться поиском в верхней части окна. Например, при вводе 'ubuntu' будет выведен список образов, среди которых будет образ ОС Ubuntu для контейнеризации (рисунок 1.6).

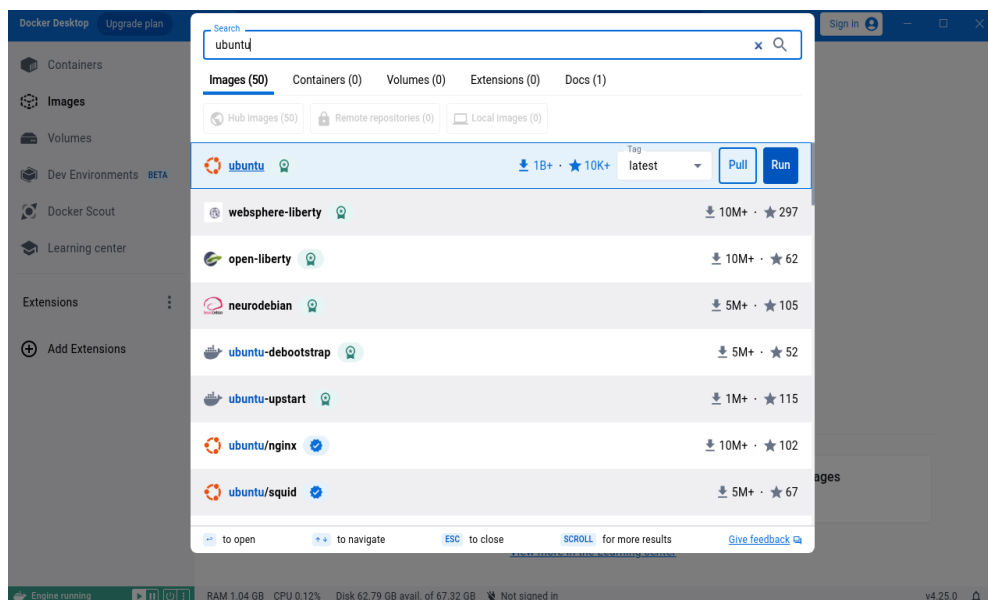


Рисунок 1.6 – Список образов при вводе 'ubuntu'

Запуск контейнеров:

1. Найти образ веб-сервера nginx, введя в поиске 'nginx' (рисунок 1.7). Выбрать релевантный образ.

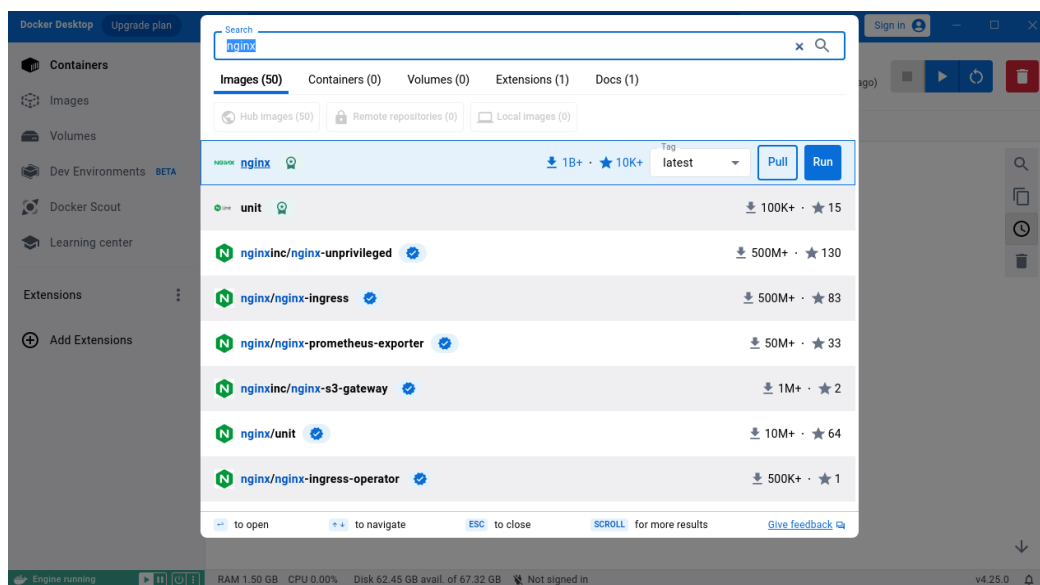


Рисунок 1.7 – Выбор релевантного образа

2. В окне настройки задать имя контейнера, пробрасываемый порт (например `nginx_demo` и `8080` соответственно) (рисунок 1.8). Нажать кнопку "Run".

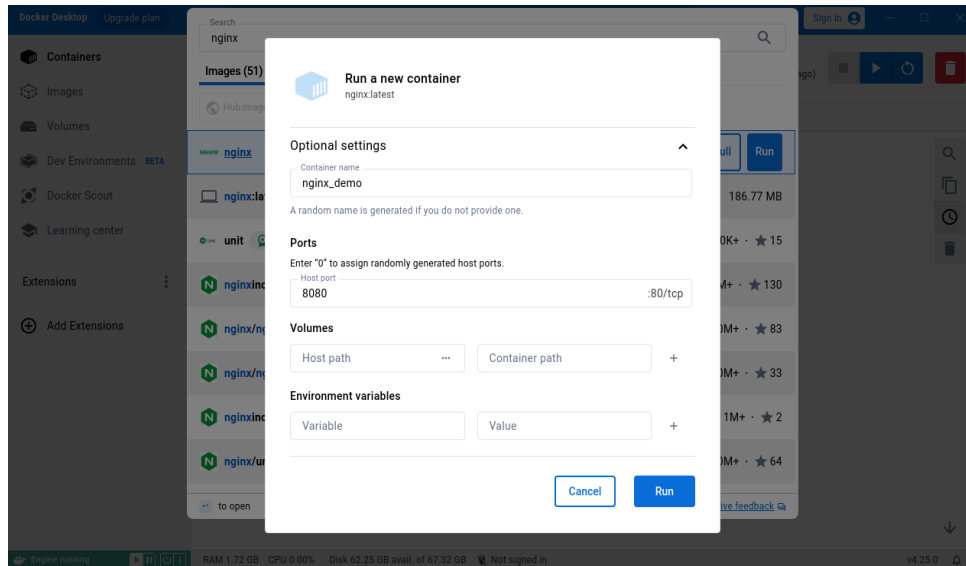


Рисунок 1.8 – Окно настроек

3. Проверить работу контейнера двумя способами. Первый - через логи (рисунок 1.9). Второй - в веб-браузере перейти `localhost:8080`, должна отобразиться приветственная страница nginx (рисунок 1.10).

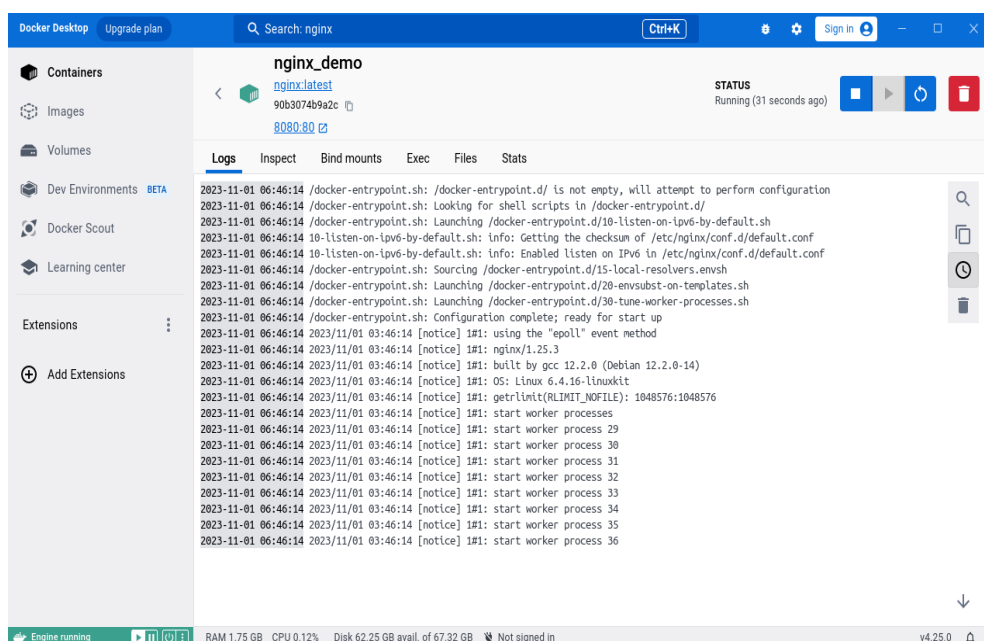


Рисунок 1.10 – Проверка работы контейнера через логи

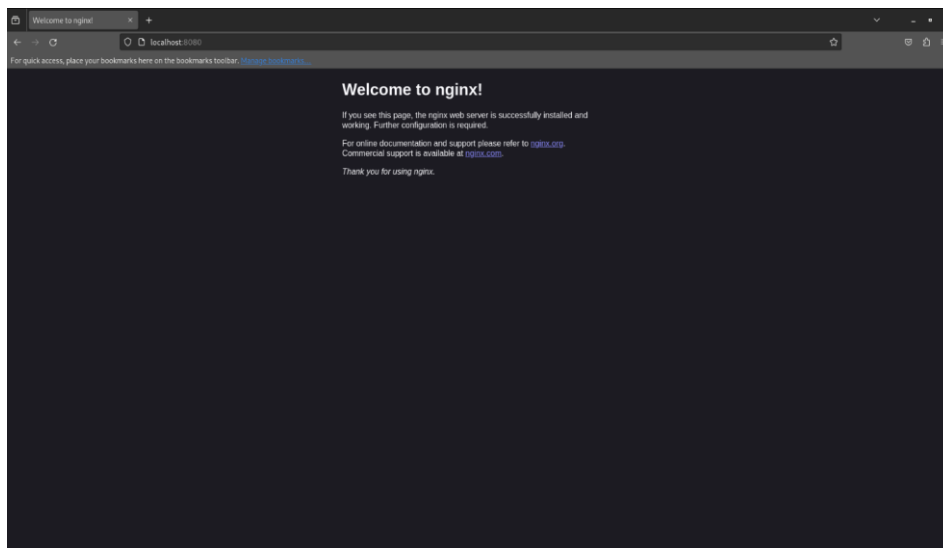


Рисунок 1.11 – Проверка работы контейнера в веб-браузере

Расширения:

DockerDesktop поддерживает создание собственных и установку сторонних расширений. На рисунке 1.12 показан магазин расширений. В дальнейшем будет использовано расширение "PortNavigator".

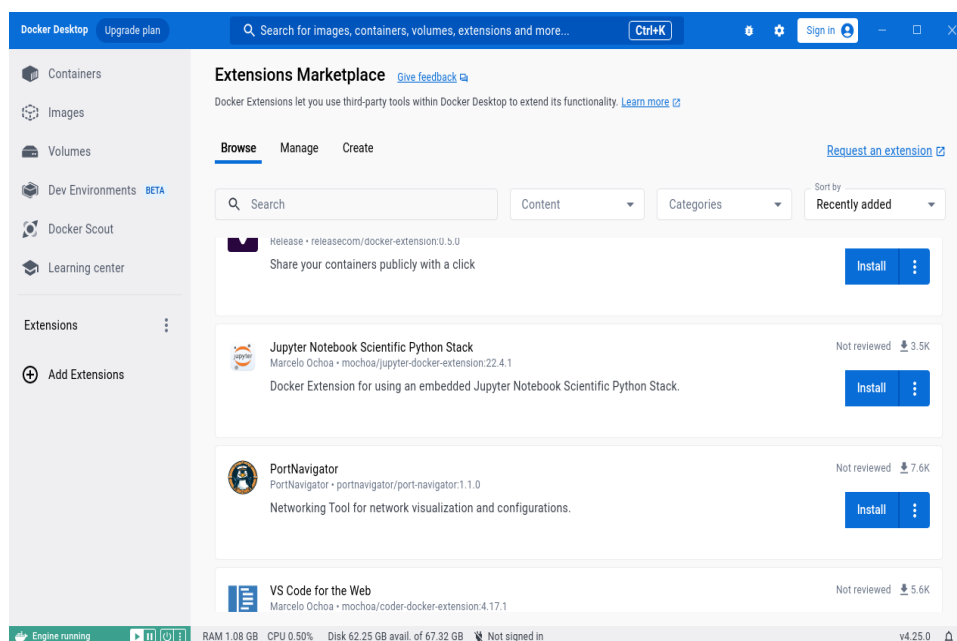


Рисунок 1.12 – Магазин расширений

Дополнительное задание:

Исследование возможностей DockerCompose на примере настройки Zabbix

Здесь описан пример создания и настройки системы мониторинга на основе Zabbix. Предполагается, что в качестве прокси будет использован Nginx, приложения баз данных - PostgreSQL, и соответственно будет установлен веб-сервер Zabbix с поддержкой PostgreSQL.

Установка DockerCompose:

Установка DockerCompose возможна только для Linux.

Для установки достаточно загрузить необходимый пакет через пакетный менеджер:

```
apt install -y docker-compose # Ubuntu, Debian
dnf install -y docker-compose # Fedora
```

Создание конфигурации DockerCompose

Установка глобальных параметров

Для удобства запуска нескольких контейнеров и объединения их единой сетью будет составлена конфигурация `docker-compose.yml`.

В пустой папке создать `docker-compose.yml`. Заполнить файл следующим содержимым:

```
yaml
networks:
  frontend:
  backend:
```

Таким образом с помощью свойства `networks` задан список docker-сетей. Пусть будут созданы следующие подсети: `frontend` - подсеть для

прокси Nginx и веб-приложения Zabbix, `backend` - подсеть для базы данных и веб-сервера Zabbix.

Установка параметров сервиса прокси Nginx

Добавить в файл описание сервиса Nginx:

```
yml
service:
proxy:
image: nginx
ports:
  - 80:80
  - 443:443
volumes:
  - /path/to/certs:/certs:ro
  - ./nginx.conf:/etc/nginx/nginx.conf:ro
  - /path/to/logs:/var/log/nginx
networks:
frontend:
```

`проху` - имя сервиса. Свойством `image` задаётся название образа.

Свойством `ports` задаётся список пробрасываемых портов. В данном случае были проброшены порты 80 (для http соединений) и 443 (для HTTPS соединений).

Свойством `volumes` задан список разделов, точек монтирования в контейнер. В данном случае были созданы следующие точки монтирования: каталог с сертификатами `/path/to/certs` монтируется внутри контейнера как `/certs` в режиме `ro` (readonly - только для чтения), каталог с логами `/path/to/logs` - как `/var/log/nginx` в режиме `rw` (чтение-запись) по умолчанию, файл-конфигурация для Nginx `./nginx.conf` - как `/etc/nginx/nginx.conf` в режиме `ro`. Конфигурация Nginx будет описана позже.

Свойством ``networks`` задан список сетей, к которым данный контейнер будет подключен.

Установка параметров сервиса баз данных

Добавить в файл описание сервиса PostgreSQL:

```
yml
database:
image: postgres:15.4-alpine
environment:
  - POSTGRES_PASSWORD=secretpass
volumes:
  - /path/to/postgres_data:/var/lib/postgresql/data
networks:
backend:
```

Таким образом ``docker-compose.yml`` должен принять примерно следующий вид:

```
yml
service:
proxy:
image: nginx
  # ...
database:
image: postgres:15.4-alpine
  # ...

networks:
# ...
```

Отметим, что в задании имени образа присутствует тег образа.

Описатель сервиса имеет дополнительные свойства.

Свойство ``environment`` задаёт список переменных окружения, доступных внутри контейнера. В данном случае переменная ``POSTGRES_PASSWORD`` задаёт пароль администратора баз данных.

Другим способом установки переменных окружения является задание свойства ``env_file``, в котором указан список файлов, откуда подгрузить переменные. Таким образом в ``docker-compose.yml`` будут записаны следующие строки:


```
yml
env_file:
- ./prod.env
```

а `prod.env` будет иметь следующее содержимое:

```
shell
POSTGRES_PASSWORD=secretpass
```

Также данный образ поддерживает так называемый "DockerSecrets" - способ задания в конфигурации источника секретных данных, таких как логин, пароль, различные ключи. Как правило - это те же переменные окружения, но оканчивающиеся на `_FILE`, а в качестве значения указывается путь внутри контейнера, откуда брать секретные данные. Если использовать данный способ, то файл `/path/to/postgres/password` будет содержать пароль администратора баз данных, а в конфигурации будут присутствовать следующие строки:

```
yml
database:
  # ...
environment:
  - POSTGRES_PASSWORD_FILE=/run/postgres-password.secret
volumes:
  - /path/to/postgres/password:/run/postgres-password.secret:ro
```

Настройка Zabbix

В соответствии с указанными на DockerHub настройками возможна следующая конфигурация сервисов Zabbix:

```
yml
zabbix-server:
image: zabbix/zabbix-server-pgsql:alpine-latest
environment:
  - DB_SERVER_HOST=database
  - DB_SERVER_PORT=5432
  - POSTGRES_DB=zabbix
```

```

    - POSTGRES_USER=zabbix
    - POSTGRES_PASSWORD=secretpass
ports:
    - 10051:10051
networks:
backend:

zabbix-web:
image: zabbix/zabbix-web-apache-pgsql:alpine-latest
environment:
    - DB_SERVER_HOST=database
    - DB_SERVER_PORT=5432
    - POSTGRES_DB=zabbix
    - POSTGRES_USER=zabbix
    - POSTGRES_PASSWORD=secretpass
    - ZBX_SERVER_HOST=zabbix-server
networks:
frontend:
backend:

```

Отметим, что для сервиса `zabbix-server` необязательно производить проброс порта 10051 в случае отсутствия настройки активной проверки.

Также стоит отметить, что в переменных `DB_SERVER_HOST` задан не конкретный IP-адрес контейнера в подсети, а доменное имя контейнера. Это возможно, потому что Docker имеет встроенный DNS-сервер. Аналогичен смысл значения переменной `ZBX_SERVER_HOST`.

Расстановка зависимостей

DockerCompose при запуске проекта одновременно запускает все сервисы. В результате может сложиться ситуация, когда сервис был проинициализирован раньше своей зависимости, и при попытке взаимодействия с ней получит ошибку. Пример представлен на рисунке 3.1.1, Nginx сообщает, что не удалось разрешить имя zabbix-web. Поэтому необходимо указывать зависимости для организации порядка запуска сервисов с помощью свойства `depends_on`.

```

proxy_1      | /docker-entrypoint.sh: Configuration complete; ready for start up
zabbix-web_1 | ** Using POSTGRES_PASSWORD variable from ENV
zabbix-web_1 | *****
zabbix-web_1 | * DB_SERVER_HOST: database
proxy_1      | nginx: [emerg] host not found in upstream "zabbix-web" in /etc/nginx/nginx.conf:24
zabbix-web_1 | * DB_SERVER_PORT: 5432
zabbix-web_1 | * DB_SERVER_DBNAME: zabbix
zabbix-web_1 | * DB_SERVER_SCHEMA: public

```

Рисунок 1.13 – Пример ошибки

В конфигурации `docker-compose.yml` следует указать зависимости.

Файл примет следующий вид:

```

yml
services:
  proxy:
    image: nginx
    depends_on:
      - zabbix-web
      # ...

  database:
    image: postgres:15.4-alpine
    # Нет зависимостей
    # ...

  zabbix-server:
    image: zabbix/zabbix-server-pgsql:alpine-latest
    depends_on:
      - database
      # ...

  zabbix-web:
    image: zabbix/zabbix-web-apache-pgsql:alpine-latest
    depends_on:
      - database
      - zabbix-server
      # ...

# ...

```

Конфигурирование Nginx

Далее представлено содержимое `nginx.conf`:

```

conf
worker_processes 1;

events {}

http {
    server {
        server_name zabbix.lnt;
        listen 80;

        return 301 https://$host$request_uri;
    }

    server {
        server_name zabbix.lnt;
listen 443 ssl;

        ssl_certificate /certs/lnt.zabbix.crt;
        ssl_certificate_key /certs/root-ca.key;

        location / {
            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;
            proxy_pass http://zabbix-web:8080;
        }
    }
}

```

Заметим, что при настройке Nginx также используются преимущества, предоставляемые Docker DNS сервером. Также предполагается, что при создании сертификата было задано доменное имя сервера `zabbix.lnt`, а веб-приложение Zabbix слушает порт по умолчанию 8080.

Запуск проекта

Находясь в каталоге с `docker-compose.yml`, запустить проект следующей командой:

```
shell
docker-composeup
```

Проект можно запустить в режиме демона, если задать опцию `-d``:

```
shell
docker-composeup -d
```

Также при запуске возможно указание имени проекта:

```
shell
docker-compose -p myzabbix up
```

Запущенный в фоне проект можно остановить с помощью следующей команды:

```
shell
docker-compose -p myzabbix down
```

Список выполняющихся контейнеров возможно получить, выполнив:

```
shell
docker-compose -p myzabbixps
```

Результат выполнения представлен на рисунке 1.14

```
$ docker-compose -p myzabbix ps
```

Name	Command	State	Ports
myzabbix_database_1	docker-entrypoint.sh postgres	Up	5432/tcp
myzabbix_proxy_1	/docker-entrypoint.sh nginx ...	Up	0.0.0.0:443->443/tcp, :::443->443/tcp, 0.0.0.0:80->80/tcp, :::80->80/t
myzabbix_zabbix-server_1	/sbin/tini -- /usr/bin/doc ...	Up	0.0.0.0:10051->10051/tcp, :::10051->10051/tcp
myzabbix_zabbix-web_1	docker-entrypoint.sh /usr/ ...	Up	8080/tcp, 8443/tcp

Рисунок 1.14 – Результат выполнения

Обзор запущенного проекта в DockerDesktop

Запущенный проект также можно просмотреть в DockerDesktop.

Во вкладке "Контейнеры" будет отображён список контейнеров, сгруппированных по проектам (рисунок 1.15).

	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
<input type="checkbox"/>	myzabbix		Running (4/4)	1.38%		12 minutes ago	
<input type="checkbox"/>	myzabbix_database_1 7d046991ad5	postgres:15.4-alpine	Running	0.75%		12 minutes ago	
<input type="checkbox"/>	myzabbix_zabbix-server_1 c912ca811712	zabbix/zabbix-server-pgsql:alpine-latest	Running	0.62%	10051:10051	12 minutes ago	
<input type="checkbox"/>	myzabbix_zabbix-web_1 d0487663b3ac	zabbix/zabbix-web-apache-pgsql:alpine-latest	Running	0.01%		12 minutes ago	
<input type="checkbox"/>	myzabbix_proxy_1 252e7093c945	nginx	Running	0%	443:443	12 minutes ago	

Рисунок 1.15 – Список контейнеров

При установке расширения "PortNavigator" можно посмотреть на существующие сети и принадлежность контейнеров к ним (рисунок 1.16).



Рисунок 1.16 – Существующие сети и принадлежность контейнеров к ним

Контрольные вопросы:

4. Что такое контейнеризация и чем она отличается от виртуализации?
5. Что такое Docker-образ и Docker-контейнер?
6. Что такое Dockerfile и какие инструкции в нем обычно используются?
7. Как создать Dockerfile для своего приложения?
8. Какие основные инструкции используются в Dockerfile и что они означают (например, FROM, RUN, COPY)?
9. Как Docker управляет сетевыми настройками контейнеров?
10. Какие меры по обеспечению безопасности контейнеров можно предпринять?
11. Какие основные преимущества использования Docker для разработки и развертывания приложений?
12. Какие недостатки и проблемы могут возникнуть при использовании Docker?