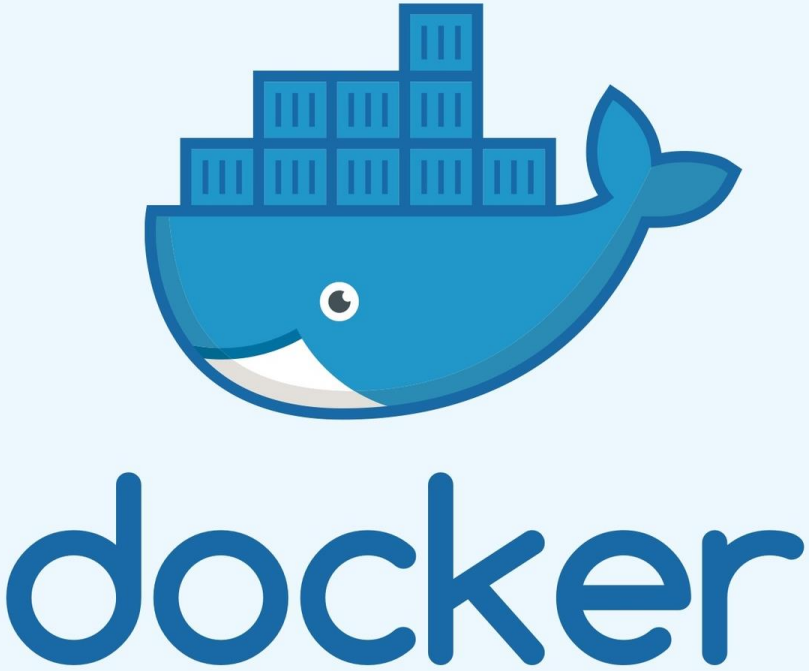
# Тема: Введение в Docker.



# План занятия:

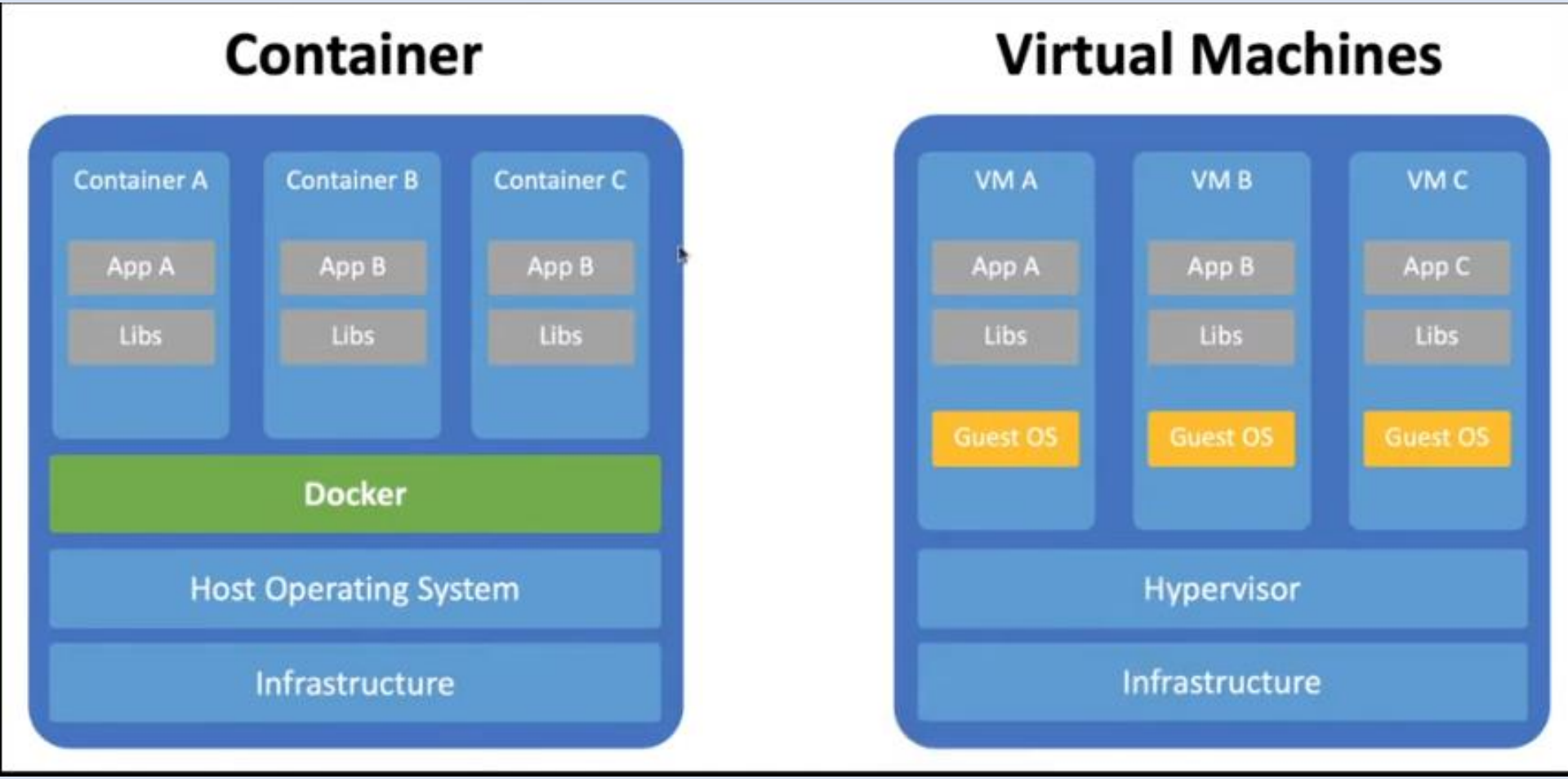
1. Введение.
2. Установка docker
3. Обзор docker
4. Управление образами и контейнерами
5. Управление сетями в docker
6. Обзор docker-compose

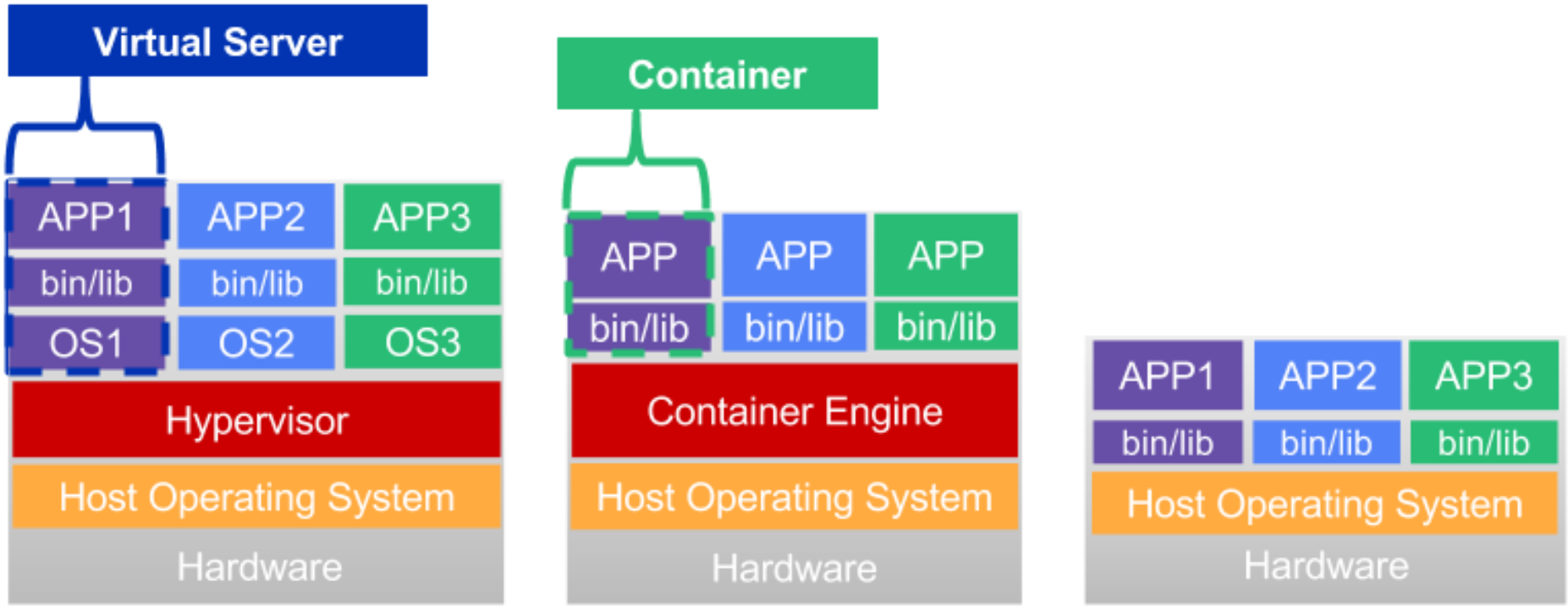
# 1. Введение

Контейнеризация - это метод виртуализации, который позволяет упаковать и запустить приложение и его зависимости в контейнере.

Контейнер представляет собой легковесный, автономный пакет, который содержит все необходимое для запуска приложения: код, среду выполнения, библиотеки, зависимости и настройки.

Другими словами, контейнеризация позволяет "упаковать" приложение и его окружение так, чтобы оно могло работать на любой платформе, включая разные операционные системы и хосты, без необходимости повторного настройки или установки зависимостей.


Container
Container A
App A
Libs
Container B
App B
Libs
Container C
App B
Libs
Docker
Host Operating System
Infrastructure
Virtual Machines
VM A
App A
Libs
Guest OS
VM B
App B
Libs
Guest OS
VM C
App C
Libs
Guest OS
Hypervisor
Infrastructure


Virtual Server
APP1
APP2
APP3
bin/lib
bin/lib
bin/lib
OS1
OS2
OS3
Hypervisor
Host Operating System
Hardware
Container
APP
APP
APP
bin/lib
bin/lib
bin/lib
Container Engine
Host Operating System
Hardware
APP1
APP2
APP3
bin/lib
bin/lib
bin/lib
Host Operating System
Hardware

```
16325 ?        S      0:22 /usr/lib/virtualbox/VBoxXPCOMIPCD
16331 ?        Sl     1:37 /usr/lib/virtualbox/VBoxSVC --auto-shutdown
16346 ?        Sl   378:10  \_ /usr/lib/virtualbox/VBoxHeadless --comment UbuntuWPT --startvm b80620b
27952 ?        Ssl   20:20 /usr/bin/containerd
28695 ?        Sl     6:07  \_ containerd-shim -namespace moby -workdir /var/lib/containerd/io.contai
28714 ?        Ss     9:13      \_ /usr/bin/python2 /usr/bin/supervisord -n -c /etc/supervisord.conf
28779 ?        S      0:05          \_ /usr/pgsql-11/bin/postgres -D /srv/postgres -c shared_preload_
28911 ?        Ss     0:00          |   \_ postgres: logger
28966 ?        Ss     0:00          |   \_ postgres: checkpointer
28967 ?        Ss     0:02          |   \_ postgres: background writer
28968 ?        Ss     0:03          |   \_ postgres: walwriter
28969 ?        Ss     0:05          |   \_ postgres: autovacuum launcher
28970 ?        Ss     0:45          |   \_ postgres: stats collector
28971 ?        Ss     0:00          |   \_ postgres: logical replication launcher
28989 ?        Ss     0:07          |   \_ postgres: pmm-managed pmm-managed 127.0.0.1(34750) idle
29131 ?        Ss     0:07          |   \_ postgres: pmm-managed pmm-managed 127.0.0.1(34876) idle
29133 ?        Ss     0:07          |   \_ postgres: pmm-managed pmm-managed 127.0.0.1(34886) idle
10933 ?        Ss     0:07          |   \_ postgres: pmm-managed pmm-managed 127.0.0.1(44392) idle
10943 ?        Ss     0:07          |   \_ postgres: pmm-managed postgres 127.0.0.1(44398) idle
10954 ?        Ss     9:55          |   \_ postgres: pmm-managed postgres 127.0.0.1(44402) idle
10963 ?        Ss     0:07          |   \_ postgres: pmm-managed pmm-managed 127.0.0.1(44406) idle
28780 ?        Sl    16:25          \_ /usr/bin/clickhouse-server --config-file=/etc/clickhouse-server
28781 ?        SLl   44:35          \_ /usr/sbin/grafana-server --homepath=/usr/share/grafana --confi
28931 ?        SLl    0:00          |   \_ /var/lib/grafana/plugins/vertamedia-clickhouse-datasource/
28784 ?        S      0:00          \_ /usr/sbin/crond -n
28785 ?        SLl  246:25          \_ /usr/sbin/victoriametrics --promscrape.config=/etc/victoriamet
```

# Преимущества контейнеризации

Изоляция: Контейнеры изолируют приложения друг от друга и от базовой системы.

Переносимость: Контейнеры могут запускаться в любой среде, где установлен Docker.

Экономия ресурсов: Контейнеры более эффективны, чем виртуальные машины, поскольку они не требуют отдельной операционной системы.

Упрощенное развертывание: Контейнеры можно легко развертывать и обновлять с помощью инструментов автоматизации.

## Область применения Docker

Разработка и тестирование: Docker позволяет разработчикам создавать изолированные среды для разработки и тестирования, что упрощает отладку и устранение неполадок.

Развертывание приложений: Docker упрощает развертывание приложений, поскольку позволяет упаковывать приложение и его зависимости в единый исполняемый пакет, который можно легко развернуть на любом сервере с установленным Docker.

Микросервисы: Docker идеально подходит для создания и управления микросервисами, поскольку позволяет легко разрабатывать, развертывать и масштабировать отдельные компоненты приложения.

Контейнеризация унаследованных приложений: Docker можно использовать для контейнеризации унаследованных приложений, что позволяет запускать их в современных средах без необходимости переписывать код.

CI/CD: Docker используется в конвейерах CI/CD для автоматизации сборки, тестирования и развертывания приложений.

Облачные вычисления: Docker широко используется в облачных средах, таких как AWS, Azure и GCP, для развертывания и управления контейнеризованными приложениями.

Образование: Docker используется в образовательных целях для обучения студентов основам контейнеризации и разработки облачных приложений.

Примеры приложений, которые можно контейнеризовать с помощью Docker:

Веб-приложения: WordPress, Drupal, Joomla, Magento, Node.js, Python Django, Ruby on Rails

Базы данных: MySQL, PostgreSQL, MongoDB, Redis, Cassandra

Микросервисы: Любое приложение, разработанное с использованием микросервисной архитектуры

Инструменты DevOps: Jenkins, GitLab CI/CD, Kubernetes

Утилиты: Nginx, Apache, HAProxy, Logstash, Kibana

Машинное обучение и искусственный интеллект: TensorFlow, PyTorch, Jupyter Notebook

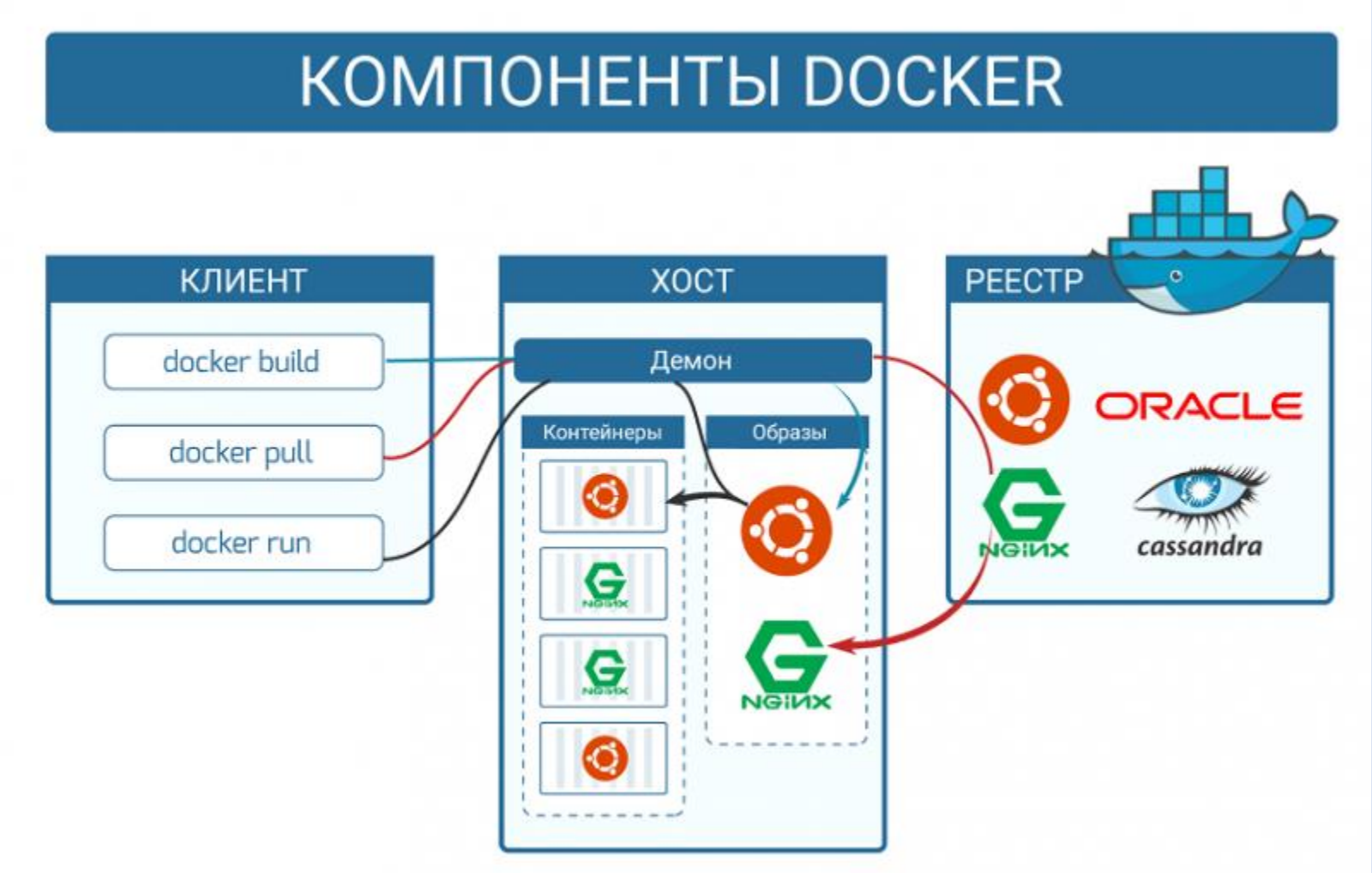
Игры: Minecraft, Terraria, Stardew Valley

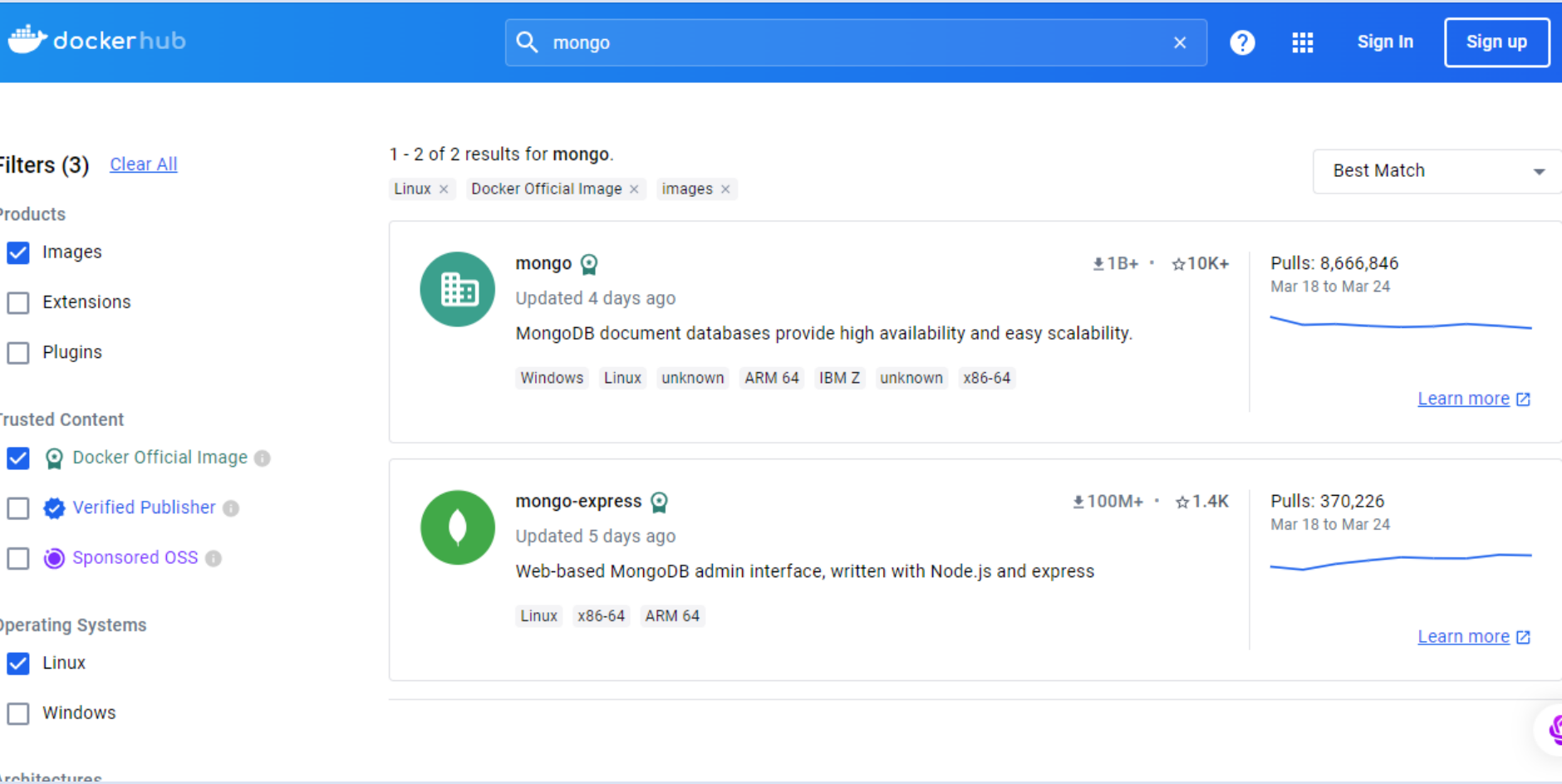
Офисные приложения: LibreOffice, OpenOffice, GIMP

Мультимедийные приложения: VLC, Plex, Kodi

Системные утилиты: Cron, SSH, Fail2ban

Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации




dockerhub
mongo
Sign In
Sign up
Filters (3)
Clear All
Products
Images
Extensions
Plugins
Trusted Content
Docker Official Image
Verified Publisher
Sponsored OSS
Operating Systems
Linux
Windows
1 - 2 of 2 results for mongo.
Linux ×
Docker Official Image ×
images ×
Best Match
mongo
1B+ · ☆10K+
Updated 4 days ago
MongoDB document databases provide high availability and easy scalability.
Windows
Linux
unknown
ARM 64
IBM Z
unknown
x86-64
Pulls: 8,666,846
Mar 18 to Mar 24
Learn more
mongo-express
100M+ · ☆1.4K
Updated 5 days ago
Web-based MongoDB admin interface, written with Node.js and express
Linux
x86-64
ARM 64
Pulls: 370,226
Mar 18 to Mar 24
Learn more

## Основные понятия Docker

- Docker образы (Images) — это неизменяемые файлы, содержащие исходный код, библиотеки, зависимости, инструменты и другие файлы, необходимые для выполнения приложения. Образы используются для создания контейнеров.
- Docker контейнеры (Containers) — это запущенные экземпляры образов Docker. Контейнеры можно запускать, останавливать, перемещать и удалять.
- Dockerfile — это текстовый файл, содержащий последовательность инструкций и команд для автоматической сборки образа Docker.
- Docker Hub (реестр)— это облачный репозиторий, который позволяет хранить, обмениваться и управлять образами Docker. Он поддерживает как публичные, так и частные хранилища образов.

**Основные команды Docker включают:**

docker build — создает образ из Dockerfile.

docker pull — загружает образ из репозитория.

docker run — запускает контейнер из образа.

docker push — загружает образ в репозиторий.

docker images — показывает список локальных образов.

docker ps — показывает список запущенных контейнеров.

docker exec - запустить интерактивную сессию терминала внутри контейнера.

# 2. Установка Docker

**Ubuntu 22.04. Способ №1.**

**Обновляем индексы пакетов apt**

sudo apt update

**Устанавливаем дополнительные пакеты**

sudo apt install curl software-properties-common ca-certificates apt-transport-https -y

**Импортируем GPG-ключ (**для верификации подписей ПО**)**

wget -O- https://download.docker.com/linux/ubuntu/gpg | gpg --dearmor | sudo tee /etc/apt/keyrings/docker.gpg > /dev/null

**Добавляем репозиторий докера**

echo "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu jammy stable"| sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

**Устанавливаем докер**

sudo apt install docker -y

**Ubuntu 22.04. Способ №2.**

**1. Установите Snapd:**

sudo apt update

sudo apt install snapd

**2. Установите Docker:**

sudo snap install docker

**3. Запустите Docker:**

sudo systemctl start docker

**4. Добавьте пользователя в группу Docker:**

sudo usermod -aG docker $USER

**5. Проверка установки:**

docker run hello-world

Установка Docker в Centos 7:

**1. Установите необходимые пакеты:**

sudo yum install yum-utils device-mapper-persistent-data lvm2

**2. Добавьте репозиторий Docker:**

sudo yum-config-manager --add-repo
https://download.docker.com/linux/centos/docker-ce.repo

**3. Установите Docker CE:**

sudo yum install docker-ce

**4. Запустите службу Docker:**

sudo systemctl start docker

**5. Добавьте пользователя в группу Docker:**

sudo usermod -aG docker $USER

**6. Проверка установки:**

docker run hello-world

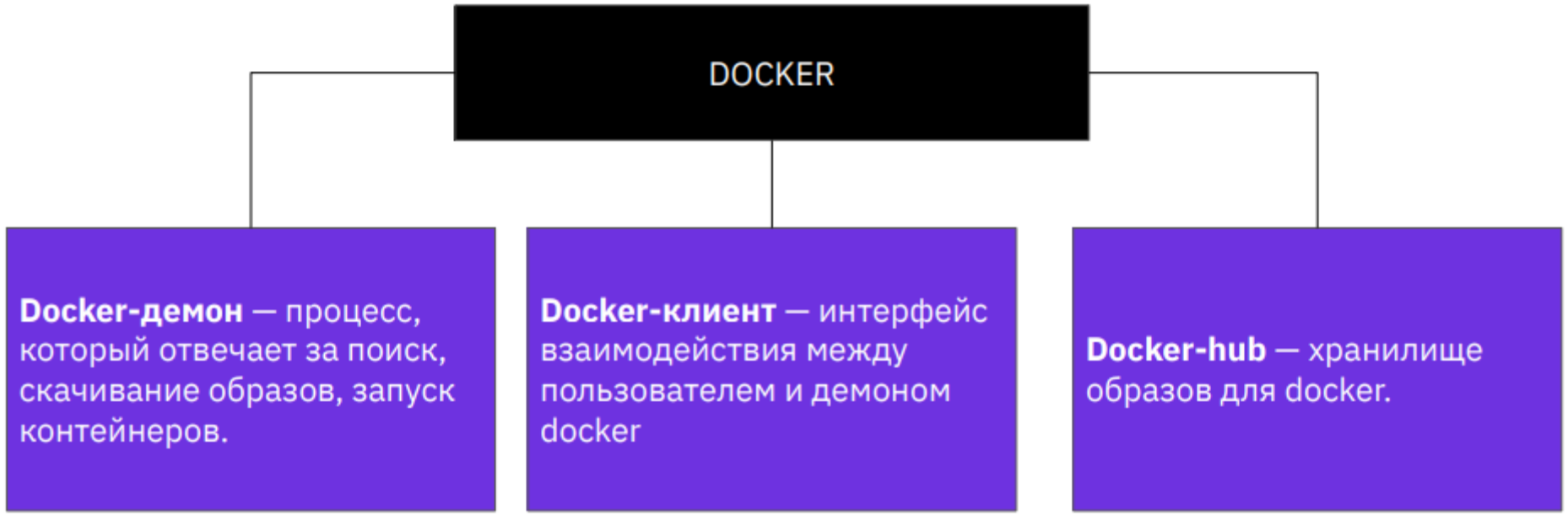
```
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

[root@localhost ~]#
```

Официальная страница Docker:

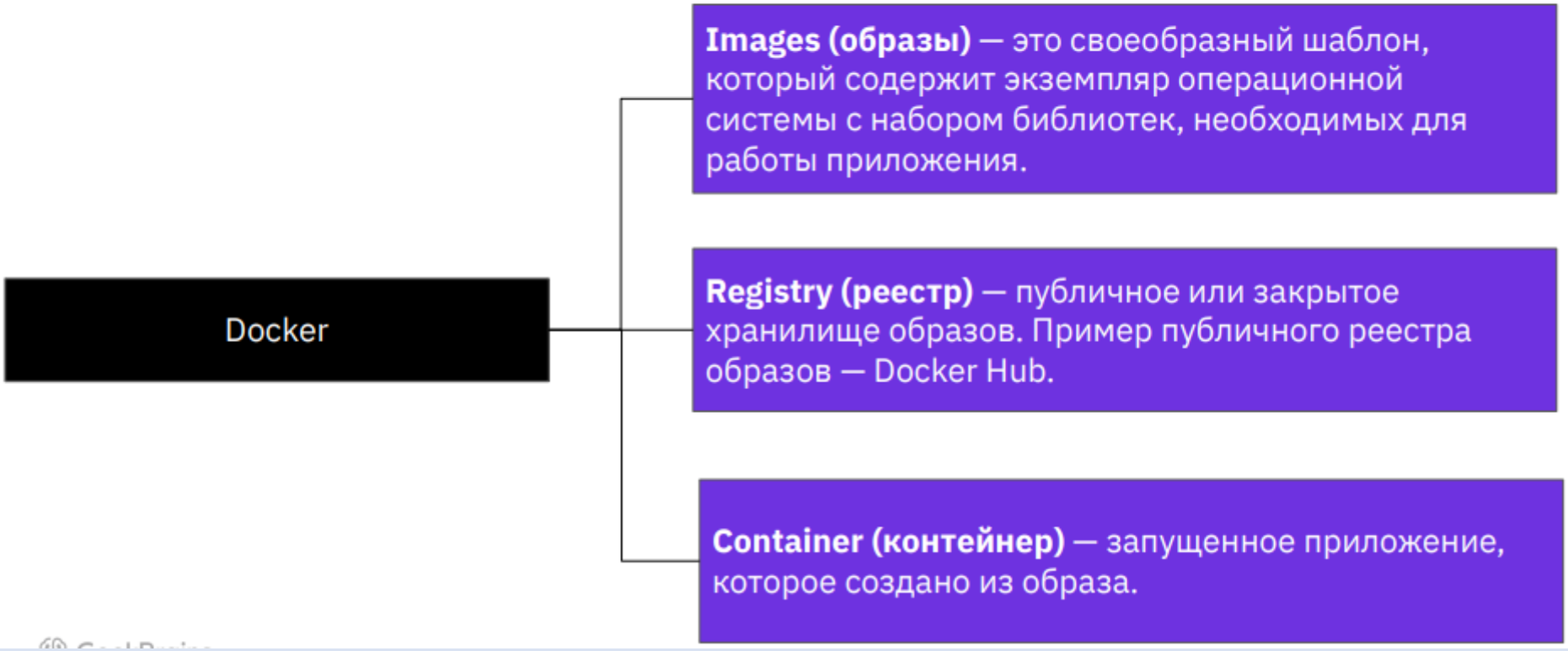
https://docs.docker.com/engine/install/centos/#next-steps

# 3. Обзор Docker

## Docker с точки зрения ОС



# Docker с точки зрения архитектуры



# 4. Управление образами и контейнерами

Управление образами:

Список образов: Просмотреть список доступных образов:

```
docker images
```

Создание образа: Создать образ из Dockerfile:

```
docker build -t <название_образа> <путь_к_Dockerfile>
```

Загрузка образа: Загрузить образ из Docker Hub:

```
docker pull <название_образа>
```

Удаление образа: Удалить образ:

```
docker rmi <название_образа>
```

Управление контейнерами:

Создание контейнера: Создать контейнер из образа:

```
docker run --name <имя_контейнера> -d <название_образа>
```

Список контейнеров: Просмотреть список запущенных контейнеров:

```
docker run --name <имя_контейнера> -d <название_образа>
```

Список всех контейнеров: Просмотреть список всех контейнеров (включая остановленные):

```
docker ps -a
```

Запуск контейнера: Запустить остановленный контейнер:

```
docker start <имя_контейнера>
```

Остановка контейнера: Остановить работающий контейнер:

```
docker stop <имя_контейнера>
```

Перезапуск контейнера: Перезапустить контейнер:

```
docker restart <имя_контейнера>
```

Удаление контейнера: Удалить контейнер:

```
docker rm <имя_контейнера>
```

Логи контейнера: Просмотреть логи контейнера:

```
docker logs <имя_контейнера>
```

Вход в контейнер: Войти в интерактивный режим внутрь контейнера:

```
docker exec -it <имя_контейнера> /bin/bash
```

# Управление образами и контейнерами

**docker search image_name — поиск образа в реестре. Например, docker search nginx найдёт все образы, которые содержат веб-сервис nginx.**

**docker pull image_name скачает диск из реестра, например, docker pull nginx.**

**docker run --name container_name image_name запустить контейнер из скачанного образа**

**docker rm container_name удалит контейнер**

Docker-образ можно получить двумя основными способами: скачать готовый образ из репозитория Docker Hub или другого репозитория образов, либо создать свой собственный образ с помощью Dockerfile.

**Dockerfile** - это текстовый файл, который содержит инструкции для сборки Docker-образа. Dockerfile определяет, как должен быть создан образ, какие зависимости и настройки должны быть включены, какие файлы должны быть скопированы в образ и другие действия, необходимые для создания окружения, в котором будет работать ваше приложение.

# Структура Dockerfile:

**FROM** — определит базовый образ, из которого будет собираться контейнер.

**MAINTAINER** — сообщит контейнеру имя автора создаваемого образа.

**RUN** — запустит команду внутри образа.

**ADD** — берёт файлы с хоста и кладёт внутрь образа.

**VOLUME** — директория, которая будет подключена в контейнер.

**EXPOSE** — задаст порт, через который контейнер будет общаться с внешним миром.

**CMD** — команда, которая будет запущена при старте контейнера из образа.

# Структура Dockerfile:

docker build -t image_name . - сборка образа

docker run --name container_name image_name - запуск контейнера из созданного образа

```
# Указываем базовый образ
FROM ubuntu:latest

# Устанавливаем необходимые пакеты
RUN apt-get update && \
    apt-get install -y nginx

# Копируем файл конфигурации nginx
COPY nginx.conf /etc/nginx/nginx.conf

# Определяем рабочую директорию
WORKDIR /usr/share/nginx/html

# Копируем файлы приложения
COPY index.html .

# Определяем порт, который будет доступен извне контейнера
EXPOSE 80

# Запускаем команду при запуске контейнера
CMD ["nginx", "-g", "daemon off;"]
```

# 4. Управление сетями в docker.

Docker предоставляет множество инструментов для управления сетями в вашем контейнерном окружении. Вот некоторые основные возможности:

Сетевые драйверы: Docker поддерживает различные сетевые драйверы, которые определяют, как контейнеры взаимодействуют и обмениваются данными между собой и с внешними сетями. Некоторые из них включают в себя:

bridge: Стандартный драйвер сети для создания сети между контейнерами на одном хосте.

host: Использует сетевое пространство хоста для связи контейнеров.

Macvlan: Даёт контейнерам прямой доступ к интерфейсу и субинтерфейсу (VLAN) хоста.

overlay: Позволяет создавать сети, которые могут взаимодействовать между хостами.

# Управление сетями в docker

Сеть Docker

**Bridge** — сети по умолчанию, аналог типа подключения NAT в VirtualBox. Связь устанавливается через Bridge-интерфейс, который поднимается в операционной системе при установке Docker и носит название docker0.

**Host** — с помощью этого драйвера контейнер получает доступ к собственному интерфейсу хоста. Аналог подключения «Мост» в VirtualBox.

**Macvlan** - даёт контейнерам прямой доступ к интерфейсу и суб-интерфейсу (VLAN) хоста.

**Overlay -** позволяет строить сети на нескольких хостах с Docker.

# Управление сетями в docker

1. docker network ls - просмотреть доступные сети
2. docker network inspect network_name - просмотреть участников сети
3. Доступ к приложениям, запущенным в контейнере, осуществляется через iptables.

Создание сети в Docker с помощью команды docker network create:

```
docker network create mynetwork
```

Подключение контейнеров к сети: Контейнеры могут быть присоединены к существующим сетям или созданным пользовательским сетям при их запуске. Например:

```
docker run --name mycontainer --network mynetwork myimage
```

Просмотр сетей: Для просмотра существующих сетей в Docker вы можете использовать команду docker network ls. Например:

```
docker network ls
```

Информация о сети:

```
docker network inspect mynetwork
```

# 5. Обзор docker-compose

Docker Compose - это инструмент для определения и запуска многоконтейнерных приложений. Он позволяет определить все компоненты вашего приложения в файле docker-compose.yml и запустить их одной командой.

В файле docker-compose.yml вы определяете все сервисы, необходимые для вашего приложения, а также настройки для каждого сервиса, такие как образ Docker, параметры сети, переменные среды, привязка портов и т. д.

# Файл docker-compose.yml:

```yaml
version: '3'
services:
  nginx:
    image: nginx:latest
    ports:
      - 80:80
    volumes:
      - /var/www/html
```

**Отступы важны!**
**(2 пробела)**

Пример простого файла docker-compose.yml, который определяет два сервиса - веб-сервер (nginx) и базу данных (MySQL):

```yaml
version: '3'

services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
    volumes:
      - ./html:/usr/share/nginx/html

  db:
    image: mysql:latest
    environment:
      MYSQL_ROOT_PASSWORD: example
      MYSQL_DATABASE: mydatabase
      MYSQL_USER: myuser
      MYSQL_PASSWORD: mypassword
```

Этот файл docker-compose.yml определяет два сервиса: web и db.

Сервис web:

image: nginx:latest: Задает образ Docker для этого сервиса. В данном случае используется образ nginx:latest, который будет загружен из Docker Hub.

ports: - "80:80": Определяет проброс портов. Это означает, что порт 80 контейнера будет доступен с хоста на порту 80.

volumes: - ./html:/usr/share/nginx/html: Указывает монтирование тома. Директория ./html на хосте будет доступна внутри контейнера по пути /usr/share/nginx/html.

Сервис db:

image: mysql:latest: Задает образ Docker для этого сервиса. В данном случае используется образ mysql:latest.

environment: Определяет переменные среды, передаваемые в контейнер. В данном случае заданы параметры для настройки MySQL: MYSQL_ROOT_PASSWORD, MYSQL_DATABASE, MYSQL_USER, MYSQL_PASSWORD.

Чтобы запустить приложение с использованием Docker Compose, вы переходите в каталог, содержащий файл docker-compose.yml, и выполняете команду:

```
docker-compose up -d
```

Эта команда прочитает файл docker-compose.yml, создаст и запустит контейнеры для каждого определенного сервиса, а также создаст и настроит сети и тома, указанные в файле.

Пример запуска приложения Nginx в контейнере Docker на CentOS 7:

Домашнее задание:

1. Изучить дополнительные материалы.