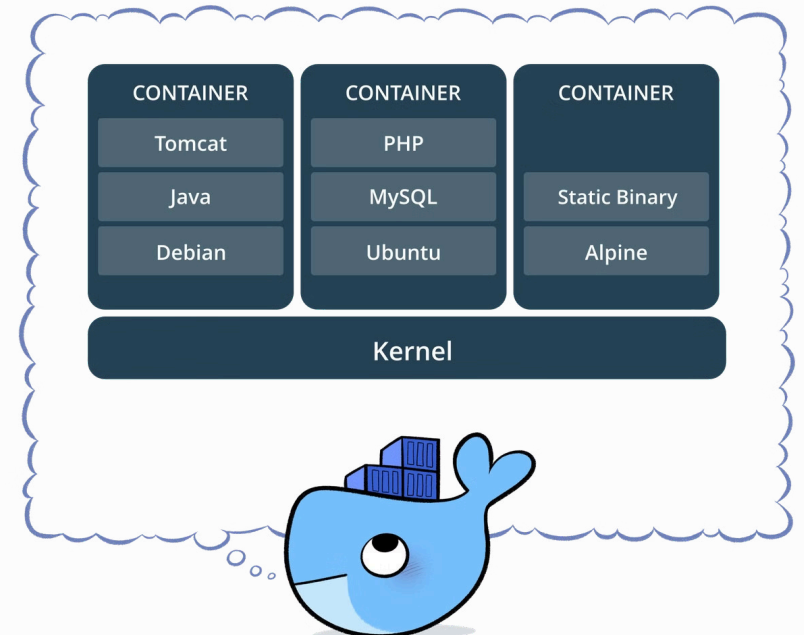


# Введение в Docker

Docker - это программная платформа, которая позволяет создавать, развертывать и управлять изолированными контейнерами программ. Она дает возможность упаковывать приложения вместе с их средой выполнения, обеспечивая их переносимость и масштабируемость между различными средами.

 **by** Илья Исаев



# Виртуализация и контейнеризация

Аппаратная виртуализация - технология, которая позволяет создавать виртуальные машины на физическом оборудовании с эффективным распределением ресурсов.

- изолированное ядро ОС
- изоляция на уровне процессора
- изолированное пользовательское окружение
- разные ОС на одном хосте
- управляется гипервизором
- большая изоляция VM

Контейнеризация - технология абстракции, которая позволяет упаковывать и исполнять приложения вместе со всеми их зависимостями в изолированных средах, называемых контейнерами.

- общее ядро с хостом
- изоляция на уровне ядра
- изолированное пользовательское окружение
- ОС, которые поддерживают ядро хоста
- управление движком контейнеризации
- меньше размер, быстрее запуск

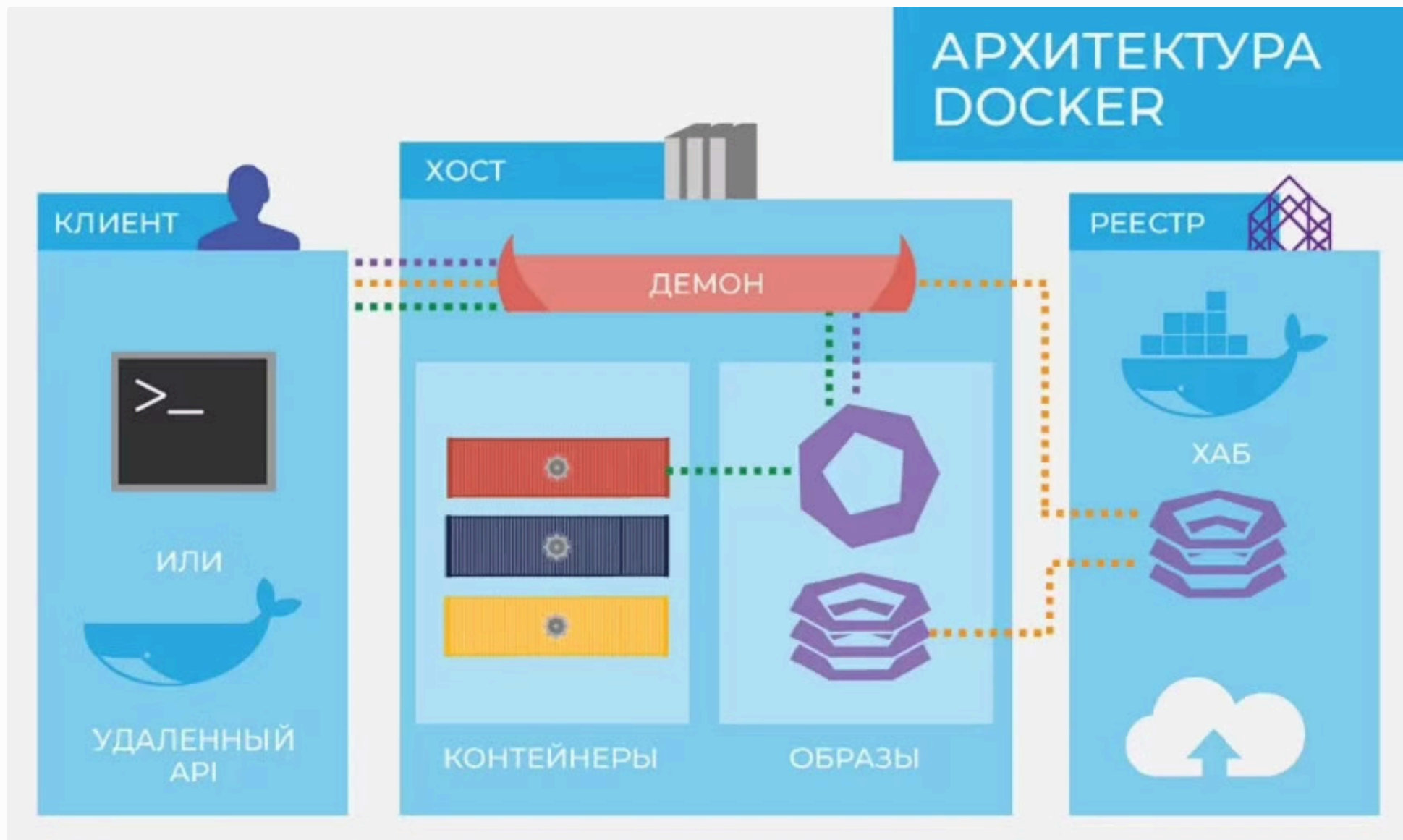
# Применение контейнеров для ПО



# History

- Docker - самая популярная реализация контейнеров
- Передано в Cloud Native Computing Foundation
- Есть несколько Container Runtime: containerd, cri-o, rkt
- Докеризовать тоже самое, что контейнеризовать
- Dockerfile тоже самое, что Containerfile (Red Hat)
- Image тоже самое, что Docker image и Container image (Red Hat), но не Container
- Containers - виртуальные среды, которые включают себя образ или образы в запущенном виде.
- Registry - репозиторий, в котором хранятся image.
- Volume — том, механизм для хранения и управления данными, которые используются внутри контейнеров.

# Архитектура



Хост Docker – это ОС, на которой работает докер

Демон Docker — это служба, которая управляет всеми элементами докера: сетями, хранилищами, образами и контейнерами.

# Сеть в Docker

- *none*: отключение всех сетевых ресурсов.
- *bridge*: сетевой драйвер по умолчанию. По сути, это мост между контейнером и хостовой машиной. Мостовые сети обычно используются, когда приложения выполняются в автономных контейнерах, которые должны взаимодействовать друг с другом.
- *host*: для автономных контейнеров устраняется сетевая изолированность между контейнером и хостом docker и напрямую используются сетевые ресурсы хоста.
- *overlay*: наложенные сети соединяют несколько демонов docker.
- *macvlan*: позволяют присваивать контейнеру MAC-адрес, благодаря чему он выглядит как физическое устройство в сети.

# Example Dockerfile

```
# Используем официальный образ Python в качестве базового
FROM python:3.9
```

```
# Устанавливаем рабочую директорию внутри контейнера
WORKDIR /app
```

```
# Копируем файлы приложения в рабочую директорию
COPY . /app
```

```
# Устанавливаем зависимости приложения
RUN pip install --no-cache-dir -r requirements.txt
```

```
# Указываем команду для запуска приложения
CMD ["python", "app.py"]
```

- UnionFS
- Каждая инструкция – отдельный layer
- В основном инструкции создают layers снизу вверх (кроме CMD и ENTRYPOINT)
- Всё, что попало в нижние layers не может быть оттуда удалено. Даже из верхних layers.

# Базовый Image - пользовательский

- Alpine
- Scratch (нулевой image)
- Red Hat UBI
- чем меньше зависимостей, тем лучше

**docker-library/  
python**


Docker Official Image packaging for Python


61  
Contributors

25  
Issues

3k  
Stars


1k  
Forks



 GitHub

**python/3.10/alpine3.21/Dockerfile at master · docker-library/pyt...**

Docker Official Image packaging for Python. Contribute to docker-library/python development by creating an account on GitHub.

 Made with Gamma



# Сборка и дистрибуция **image**

`docker build .` (сборка локального image)

`docker build --tag web:1.0.0 -t web:latest .` (указание тега)

`docker build --tag myname/web:1.0.0 .` (указание репозитория)

`docker run -ti --rm -p 8000:8000 --name web myname/web:1.0.0` (запуск container из image)

`docker push myname/web:1.0.0` (отправить image в docker registry)

# Docker Compose

Управление мультиконтейнерными приложениями

- Сборка, запуск, статус нескольких контейнеров
- Запуск проекта конфигурируется, например, с помощью YAML
- Запуск проекта на одном хосте
- Зависимости и ожидания между сервисами
- Обычно делается для упрощения разработки

# Работа с **Docker**-контейнерами



Работа с Docker-контейнерами является ключевым аспектом использования Docker. Вы можете создавать контейнеры на основе готовых образов, управлять их состоянием, просматривать информацию о них, анализировать их логи и непосредственно взаимодействовать с работающими контейнерами. Все эти операции помогут вам эффективно разворачивать и управлять приложениями в Docker.

# Хранение данных в Docker

## 1 Тома Docker

Тома Docker – это отдельные блоки для хранения данных, которые могут быть подключены к одному или нескольким контейнерам. Они обеспечивают постоянное, изолированное хранение, не зависящее от жизненного цикла контейнера.

## 2 Постоянные хранилища

Постоянные хранилища позволяют сохранять важные данные за пределами контейнера. Это удобно для баз данных, файлов конфигурации и медиа-контента, которые должны сохраняться при перезагрузке контейнера.

## 3 Использование облачных сервисов

Docker также поддерживает использование облачных хранилищ данных, как AWS S3 или Azure Blob Storage, для обеспечения масштабируемого и надежного хранения за пределами инфраструктуры.

# Best Practices



Docker Documentation



## Best practices

Hints, tips and guidelines for writing clean, reliable Dockerfiles

# Example Dockerfile

```
# Базовый image
FROM python:3.10-alpine

# Переменные, используемые для создания окружения, в котором запустится приложение
ARG USER=app
ARG UID=1001
ARG GID=1001

# Установка фреймворка
RUN pip install --no-cache-dir Flask==2.2.*

# Создание пользователя операционной системы и его домашнего каталога
RUN addgroup -g ${GID} -S ${USER} \
  && adduser -u ${UID} -S ${USER} -G ${USER} \
  && mkdir -p /app \
  && chown -R ${USER}:${USER} /app
USER ${USER}

# Переход в каталог /app
WORKDIR /app

# Переменные окружения, необходимые для запуска web-приложения
ENV FLASK_APP=server.py \
  FLASK_RUN_HOST="0.0.0.0" \
  FLASK_RUN_PORT="8000" \
  PYTHONUNBUFFERED=1

# Копирование кода приложения в домашний каталог
COPY --chown=${USER}:${USER} server.py /app

# Публикация порта, который прослушивается приложением
EXPOSE 8000

# Команда запуска приложения
CMD ["flask", "run"]
```

```
docker build -t leomag/server:0.0.0 --network host -t leomag/server:latest server
```

# Example Dockerfile

```
# Показать текущие образы  
docker image ls
```

```
# Запуск образа в контейнере  
docker run --rm -p 8000:8000 --name server --network host leomag/server:0.0.0
```

```
# Показать текущие контейнеры  
docker container list
```

# Example Docker Hub

Получим Access Token в <https://hub.docker.com/settings/security>

```
docker login -u leomag
```

```
docker push leomag/server:0.0.0
```

```
docker pull leomag/server:0.0.0
```

Лимиты: For authenticated users, there will be a 40 pull/hour rate limit per user; for unauthenticated usage, there will be a 10 pull/hour rate limit per IP address.



# Example Docker Compose

```
# Версия API Docker compose
version: "3"

# Раздел, в котором описываются приложения (сервисы).
services:

  # Раздел для описания приложения 'server'.
  server:

    # Имя image tag
    image: leomag/server:0.0.0

    # Параметры сборки Docker image.
    build:
      # Путь к Dockerfile,
      context: server/
      # Использовать host-сеть при сборке,
      network: host

    # Перенаправление портов из Docker container на host-машину.
    ports:
      - 8000:8000

    # Имя user, используемого в image,
    user: "1001"

    # Используемый тип сети при запуске container.
    network_mode: host

    # Проверка готовности приложения к работе. Параметр "--spider" означает: не загружать url,
    # а только проверить его наличие.
    healthcheck:
      test: wget --no-verbose --tries=1 --spider http://localhost:8000 || exit 1
      interval: 5s
      timeout: 5s
      retries: 5

  # Раздел для описания приложения 'client'.
  client:

    image: leomag/client:0.0.0

    build:
      context: client/

    user: "1001"

    network_mode: host

    # Команда запуска приложения внутри container,
    command: "python ./client.py"

    # Зависимость от других сервисов,
    depends_on:
      # Сервис 'client' зависит от сервиса 'server'.
      # Прежде чем запустить 'client' необходимо дождаться запуска 'server'.
      # Условием запуска сервиса 'server' является его healthcheck.
      server:
        condition: service_healthy
```

# Example Docker Compose

```
# Собрать  
docker compose build
```

```
# Запустить  
docker compose up
```

```
docker logs name_image
```