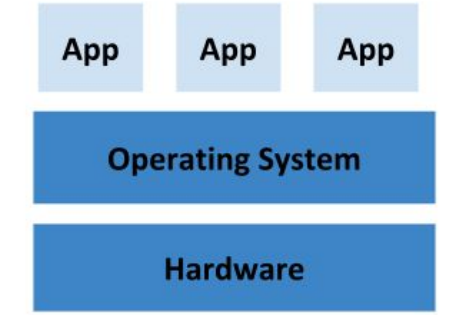
# Containerization and Docker

# Эпоха традиционного развертывания

На ранних этапах организации запускали приложения на физических серверах.

Не было способа определить границы ресурсов для приложений на физическом сервере, и это вызывало проблемы с распределением ресурсов. Например, если на физическом сервере запущено несколько приложений, могут быть случаи, когда одно приложение будет занимать большую часть ресурсов, и в результате другие приложения будут работать хуже.

Решением для этого было бы запускать каждое приложение на другом физическом сервере. Но это дорого.




Traditional Deployment

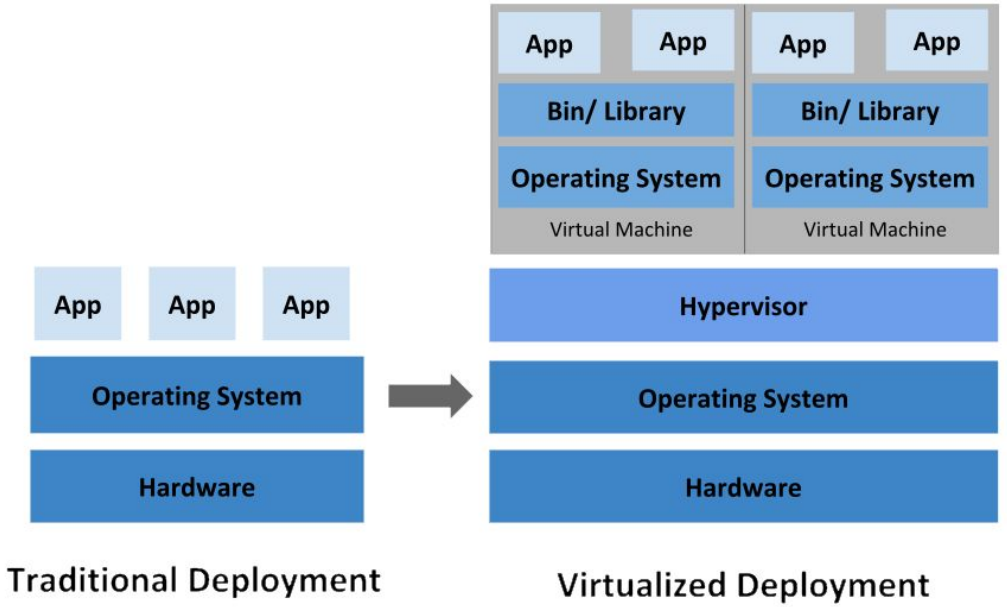
# Виртуализация

**Виртуализация** — это технология, которая позволяет одному физическому серверу выполнять функции нескольких виртуальных машин (VM).

Каждая такая **VM** имеет свою операционную систему и установленные приложения (Bin/Library).

**Гипервизор** позволяет распределить ресурсы физического сервера между VM.

**Приложение (App)**, выполняемое на этих VM, отделено от базовых аппаратных ресурсов.



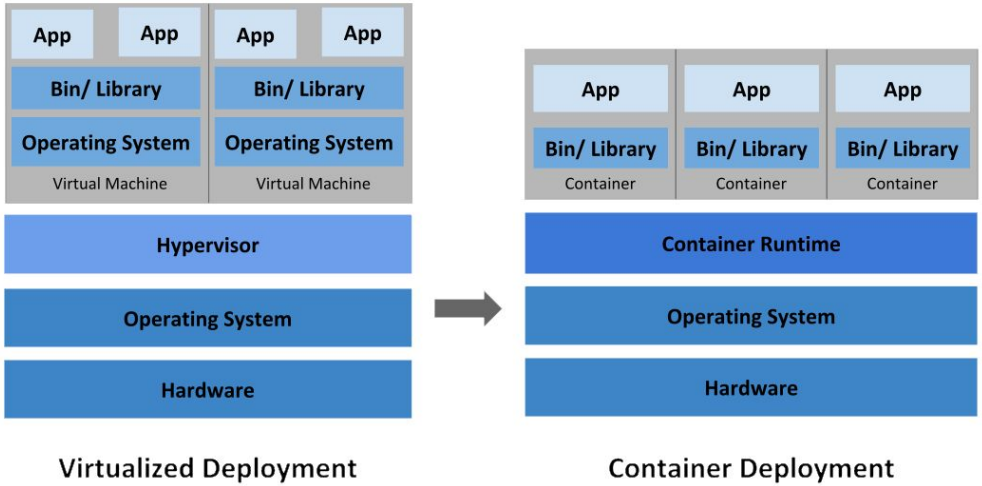

** Hypervisor - ПО для создания виртуальный машин*

# Основные принципы

- **Изоляция**. Каждая виртуальная машина работает независимо от других. Это обеспечивает стабильность и безопасность.
- **Консолидация**. Виртуализация позволяет сократить количество физических серверов, объединяя их функции в одном устройстве. Это помогает экономить ресурсы и уменьшить затраты на оборудование и обслуживание.
- **Гибкость**. Виртуальные машины можно легко создавать, изменять и удалять по мере необходимости. Это помогает быстро адаптироваться к изменяющимся потребностям бизнеса.
- **Управляемость**. Виртуальными средами можно управлять централизованно. Это упрощает администрирование и повышает эффективность использования ресурсов.

# Контейнеризация

**Контейнеры** похожи на VM, но их изоляция немного слабее: они используют общую операционную систему. Как и у VM, у контейнера есть своя файловая система, общий доступ к процессору, памяти и так далее.



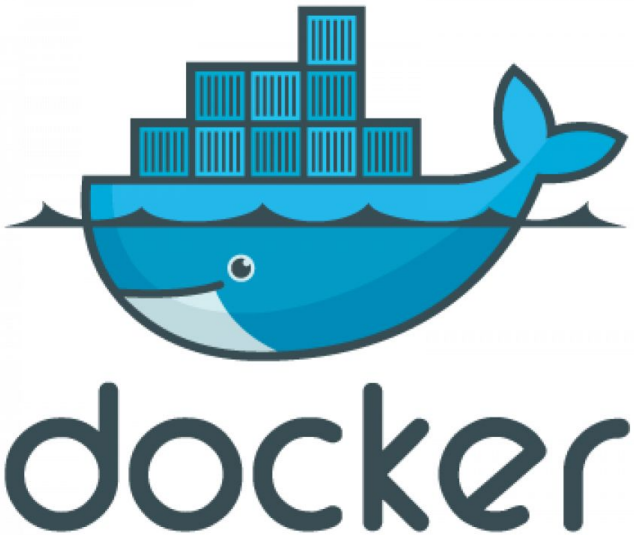
# Преимущества контейнеров

- **Легкость**: контейнеры мало весят.
- **Гибкость**: повышенная простота и эффективность создания образа контейнера.
- **Изоляция**: процессы одного контейнера полностью изолированы от общей инфраструктуры и других контейнеров.
- **Согласованность** среды разработки, тестирования и производства.
- **Наблюдаемость**: отображаются не только информация и показатели на уровне операционной системы, но и работоспособность приложений и другие сигналы.
- **Кроссплатформенность**: работает в Ubuntu, RHEL, CoreOS, локально, в крупных общедоступных облаках и где угодно еще.
- **Удобно в MSA**: приложения разбиты на более мелкие, независимые части и могут развертываться и управляться динамически - это не монолитный стек, работающий на одной большой машине одного назначения.

# Docker

Самый популярный сервис контейнеризации.

- Docker **Image** - шаблон контейнера, доступный только для чтения
- Docker **Registry** - хранилище различных образов контейнеров (например, Docker Hub)
- Docker **Container** - экземпляр образа



# Docker Image

**Docker Image** — это read-only шаблон. Представляет собой исполняемый пакет, содержащий все необходимое для запуска приложения: код, среду выполнения, библиотеки, переменные окружения и файлы конфигурации.

Например, образ может содержать операционку Ubuntu с Apache и приложением на ней.

Docker позволяет легко создавать новые образы, обновлять существующие, или вы можете скачать образы созданные другими людьми.

# Dockerfile

**Dockerfile** - это текстовый файл, содержащий набор инструкций, которые используются docker build для создания пользовательского образа docker.

**Инструкция** - директива с аргументами. Например, FROM (установить базовый образ), RUN (выполнить команду).

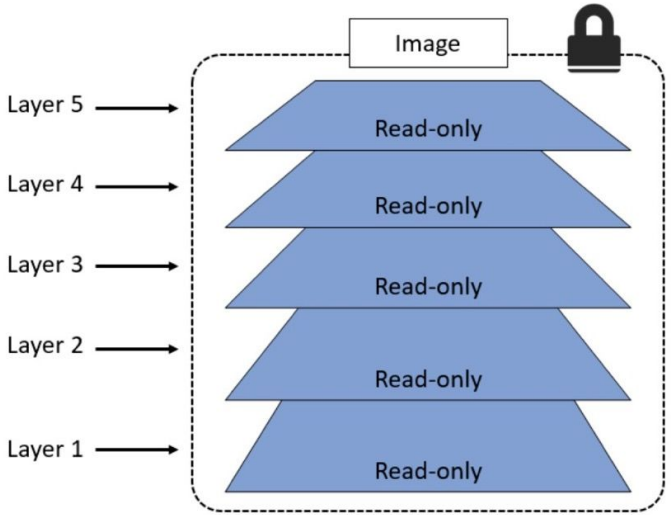
**Контекст сборки** - это главная папка, из которой копируются файлы с помощью директив ADD и COPY.

# Слои образа

Docker-образ состоит из **слоев** (как правило один слой – одна инструкция).

При загрузке или скачивании Docker-образа, операции производятся только с теми слоями, которые были изменены.

Слои исходного Docker-образа являются общими между всеми его версиями и не дублируются.




```
docker image history helloworld
IMAGE               CREATED             CREATED BY
cb84eb33ca20        58 seconds ago      /bin/sh -c #(nop)  ENTRYPOINT ["node" "hello…
7d652a817a9f        58 seconds ago      /bin/sh -c #(nop)  EXPOSE 8888
334575e947c9        59 seconds ago      /bin/sh -c #(nop) ADD file:b9606ef53b832e66e…
```

# Docker Registry

**Registry Server** (реестр, хранилище) – это репозиторий, в котором хранятся образы. Примеры: Ubuntu, Postgres, NGINX.

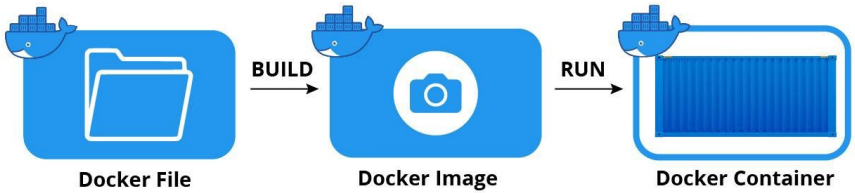
После создания образа на локальном компьютере его можно отправить (**push**) в хранилище, а затем извлечь (**pull**) на другом компьютере и запустить его там. Образы версионируются с помощью **tag** (тегов). Если тег не указан, то по умолчанию используется latest.

Существуют общедоступные и закрытые реестры образов. Примеры: Docker Hub (репозитории docker.io), RedHat Quay.io (репозитории quay.io).

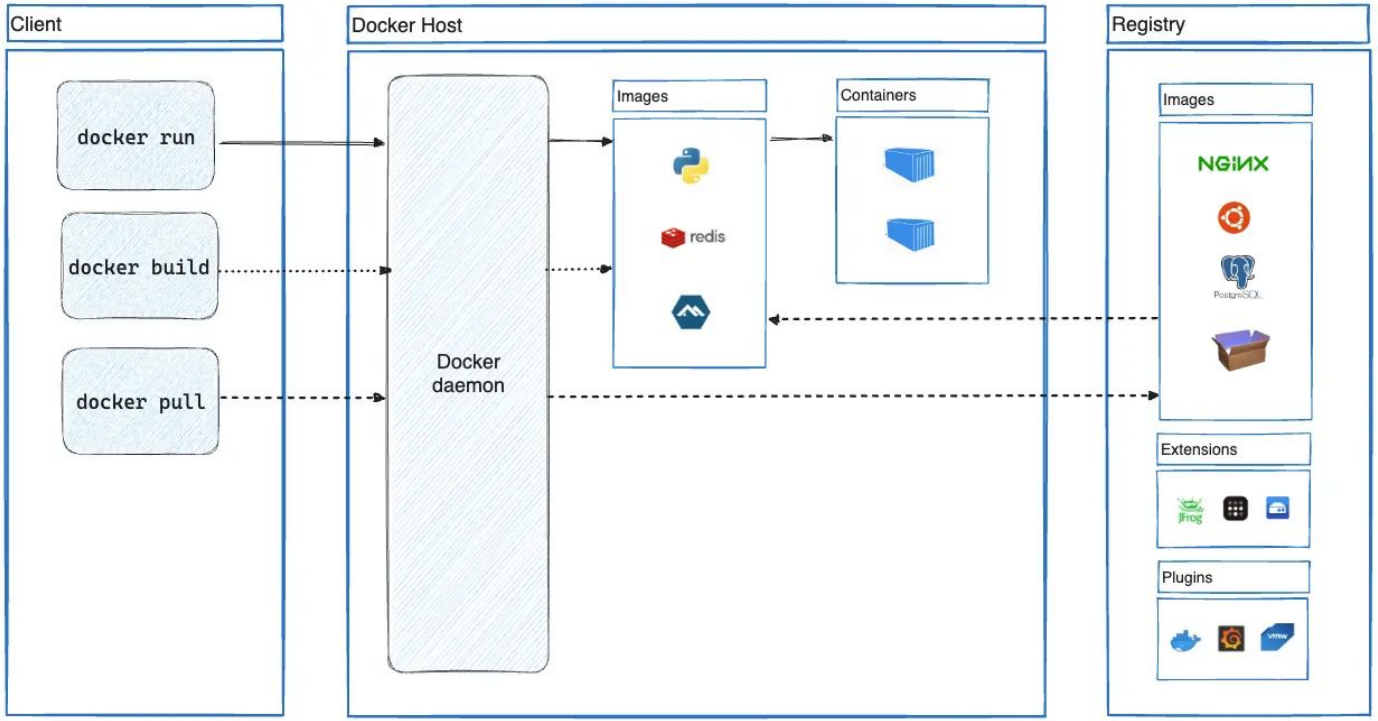
# Docker Container

**Container** (контейнер) – это экземпляр образа контейнера.

**Выполняемый контейнер** – это запущенный процесс, изолированный от других процессов на сервере и ограниченный выделенным объемом ресурсов (ЦПУ, ОЗУ, диска и др.). Выполняемый контейнер сохраняет все слои образа с доступом на чтение и формирует сверху свой исполняемый слой с доступом на запись.



# Архитектура Docker

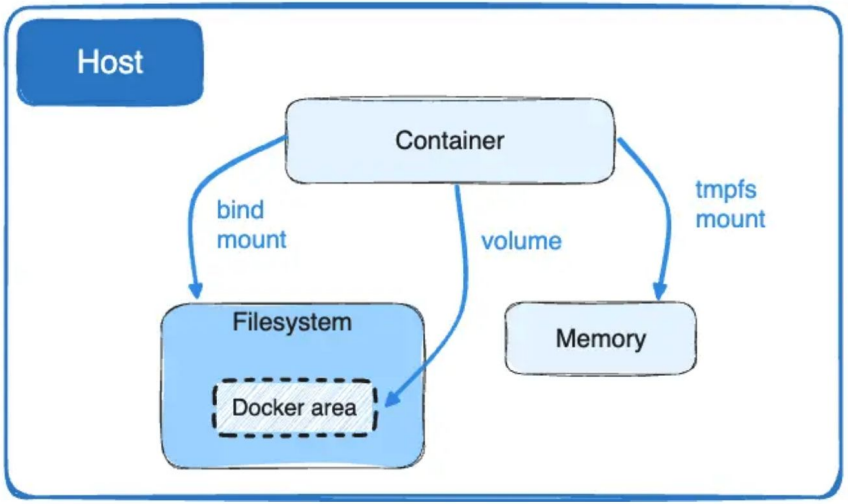


# Демонстрация Docker

# Монтирование директории / Volume

**Mount**: предоставление доступа контейнеру на чтение содержимого вашей папки из основной операционной системы.

**Volume**: это постоянные хранилища данных для контейнеров, созданные и управляемые Docker.



# Docker Compose

Контейнеры могут быть легко настроены с помощью **docker-compose**.

**version**: версия docker-compose (версия 3 - последняя на данный момент).
**services**: контейнеры, которые мы хотим запустить.
**<app>**: название сервиса
**build**: шаги, описывающие процесс сборки.
**context**: где находится файл Dockerfile.
**ports**: сопоставление портов основной операционной системы с контейнером.

```yaml
version: '3'

services:
  app:
    build:
      context: .
    ports:
      - 8080:80
```

# Демонстрация Docker Compose

# Основы GIT

Git - это популярная система контроля версий.
Он используется для:

- Отслеживания изменений
- Отслеживания, кто внес изменения
- Совместной работы

https://www.w3schools.com/git/default.asp

**Работа с Git**

- Инициализируйте Git в папке, сделав ее **репозиторием**
- Теперь Git создает скрытую папку для отслеживания изменений в этой папке
- Когда файл изменяется, добавляется или удаляется, он считается **modified**
- Вы выбираете измененные файлы, которые хотите поместить в **Stage**
- Staged файлы фиксируются (**Committed)**
- Git позволяет просматривать полную **историю** каждой коммита.
- Вы можете вернуться (**revert**) к любой предыдущей фиксации.

# Задание: Основы контейнеризации

**Оценка**: 1

**Описание**: Настройте сеть из 2-х Docker контейнеров:

- В одном из них размещается база данных (SQLite или другая)
- Другое - это ваша точка входа (entry point)

Вы должны продемонстрировать способность запрашивать данные из БД через точку входа. Данные в базе данных не должны исчезать при перезапуске контейнеров!

**DoD**: Репозиторий GitLab/GitHub с Dockerfile-ми для обоих контейнеров, docker-compose для всей системы, файл README со списком команд для запуска и тестирования. *Ссылка на Docker Hub.