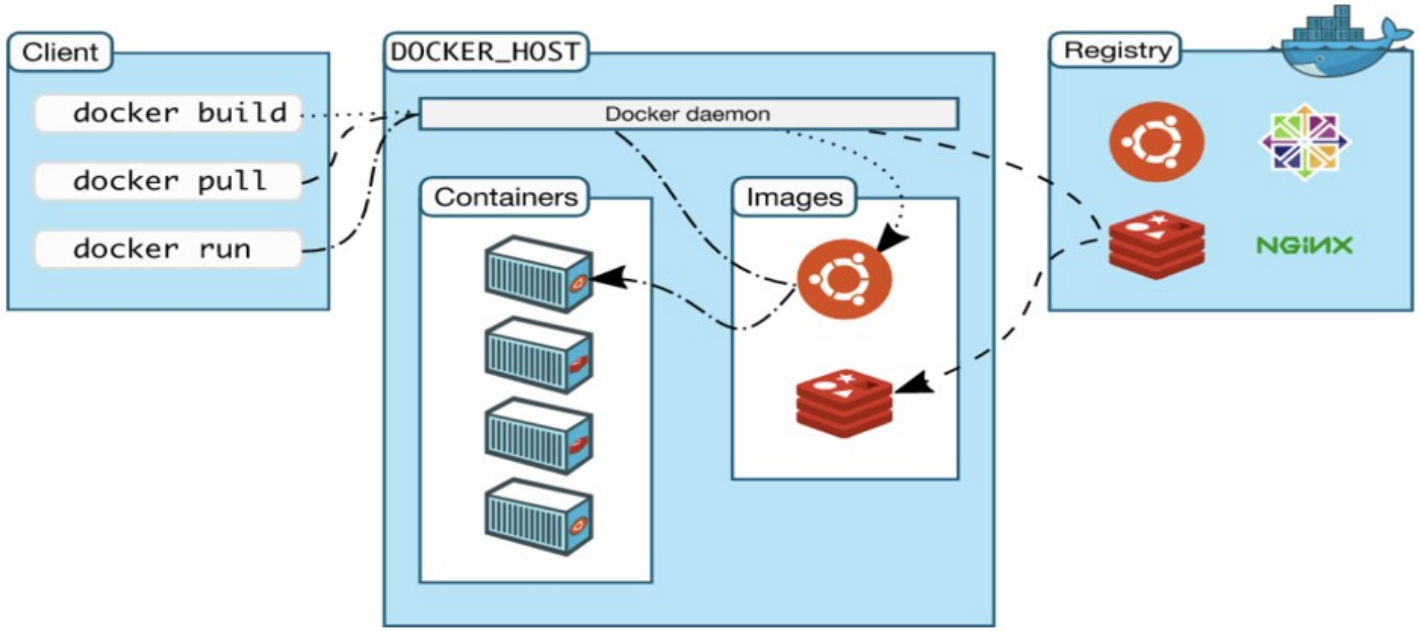
Разработка инфраструктуры программного обеспечения

Практические применения контейнеров

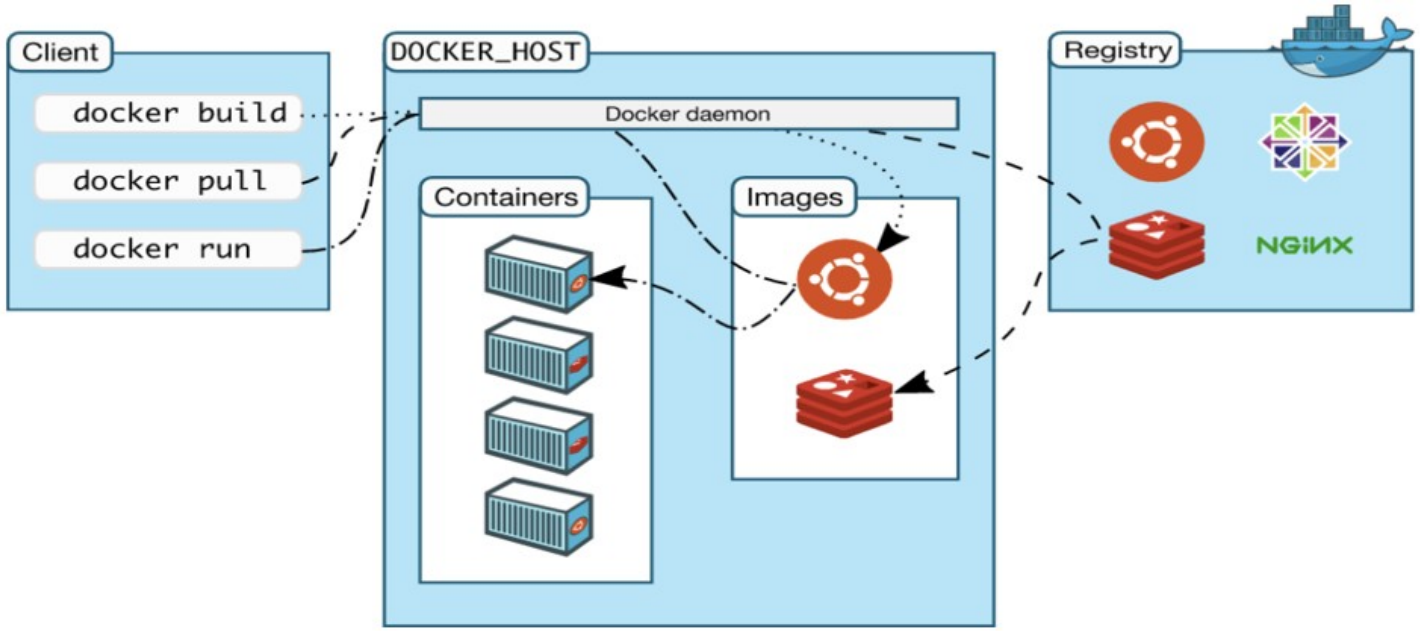
Лаборатория RISC-V технологий,
2025 г.

Терминология и общее устройство докер контейнеров



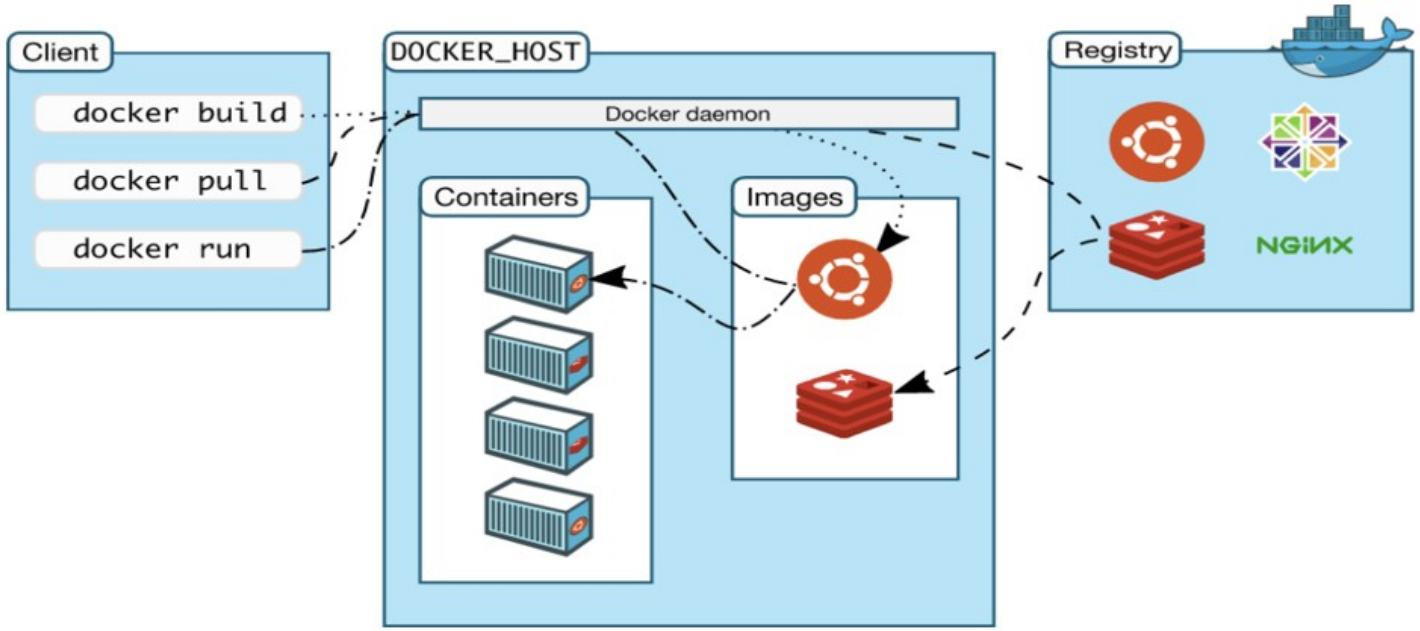
- Клиент — фронтенд docker приложения. Включает богатый CLI

Терминология и общее устройство докер контейнеров



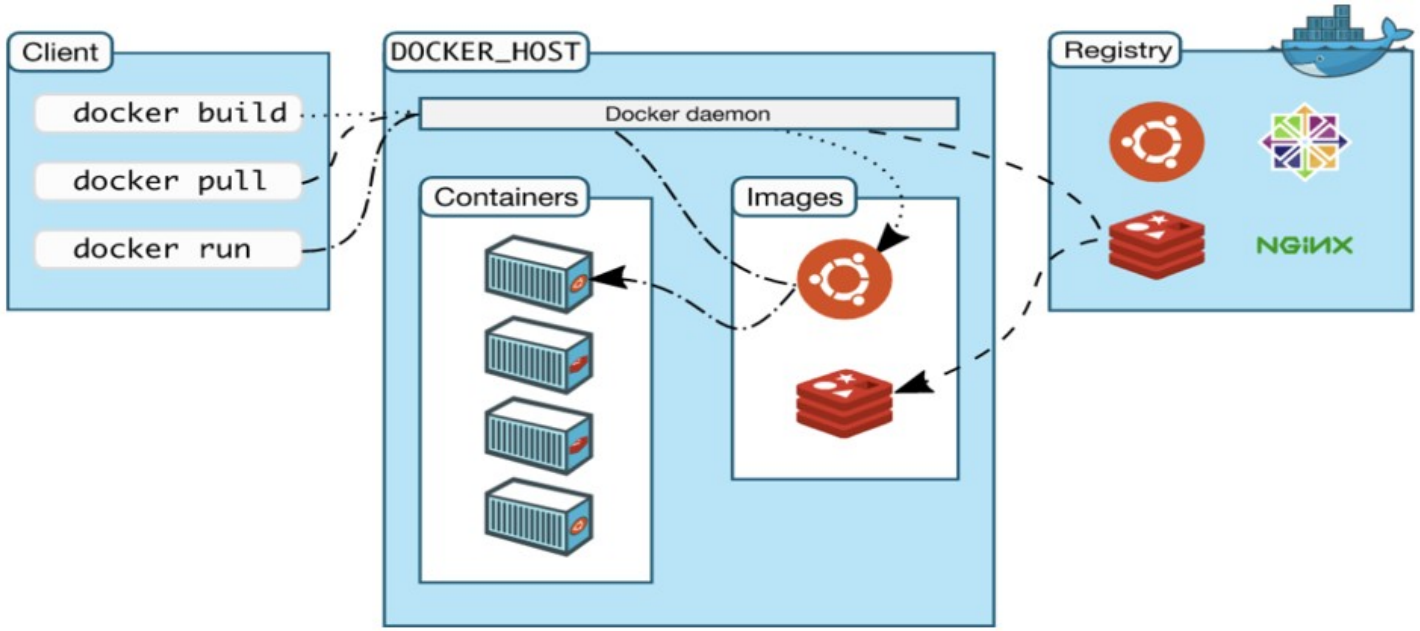
- Клиент — фронтенд docker приложения. Включает богатый CLI
- DOCKER_HOST — машина, где запущен бекенд docker. Включает в себя docker daemon, занимающийся мониторингом контейнеров. Вся связь с контейнерами происходит через демон.

Терминология и общее устройство докер контейнеров



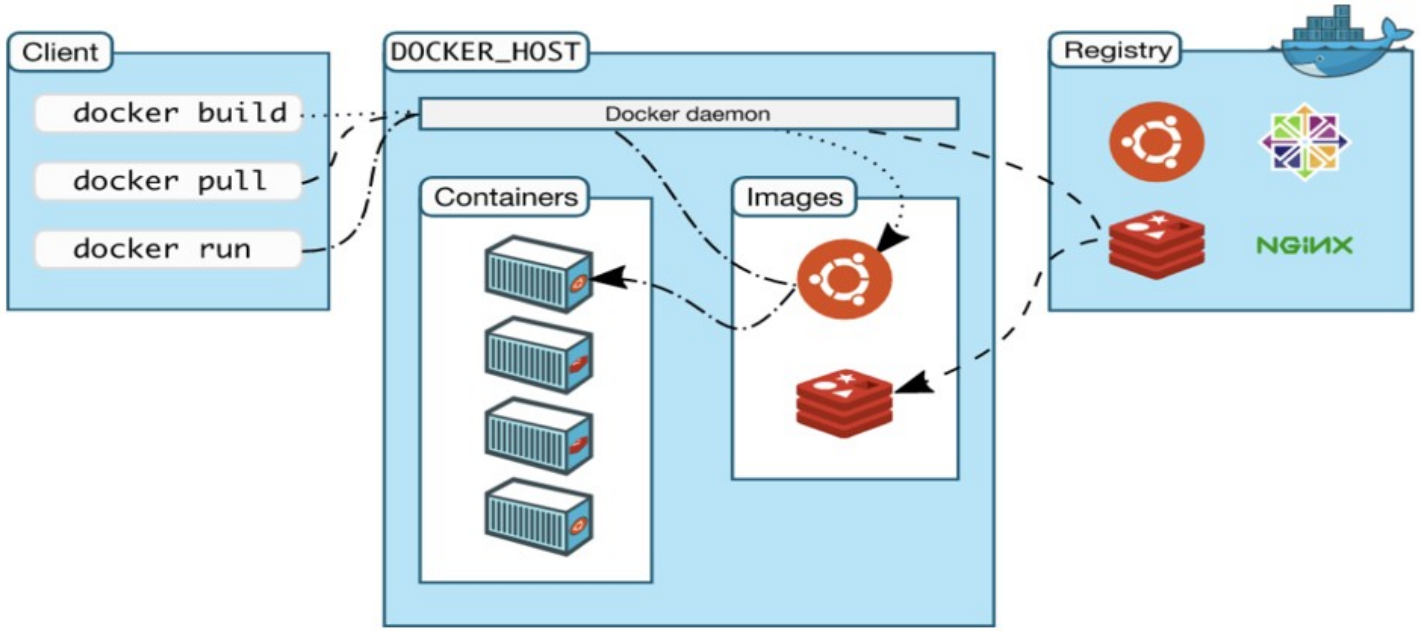
- Клиент — фронтенд docker приложения. Включает богатый CLI
- DOCKER_HOST — машина, где запущен бекенд docker. Включает в себя docker daemon, занимающийся мониторингом контейнеров. Вся связь с контейнерами происходит через демон.
- Images — образы, собранный из докерфайлов. Используются для создание контейнеров

Терминология и общее устройство докер контейнеров



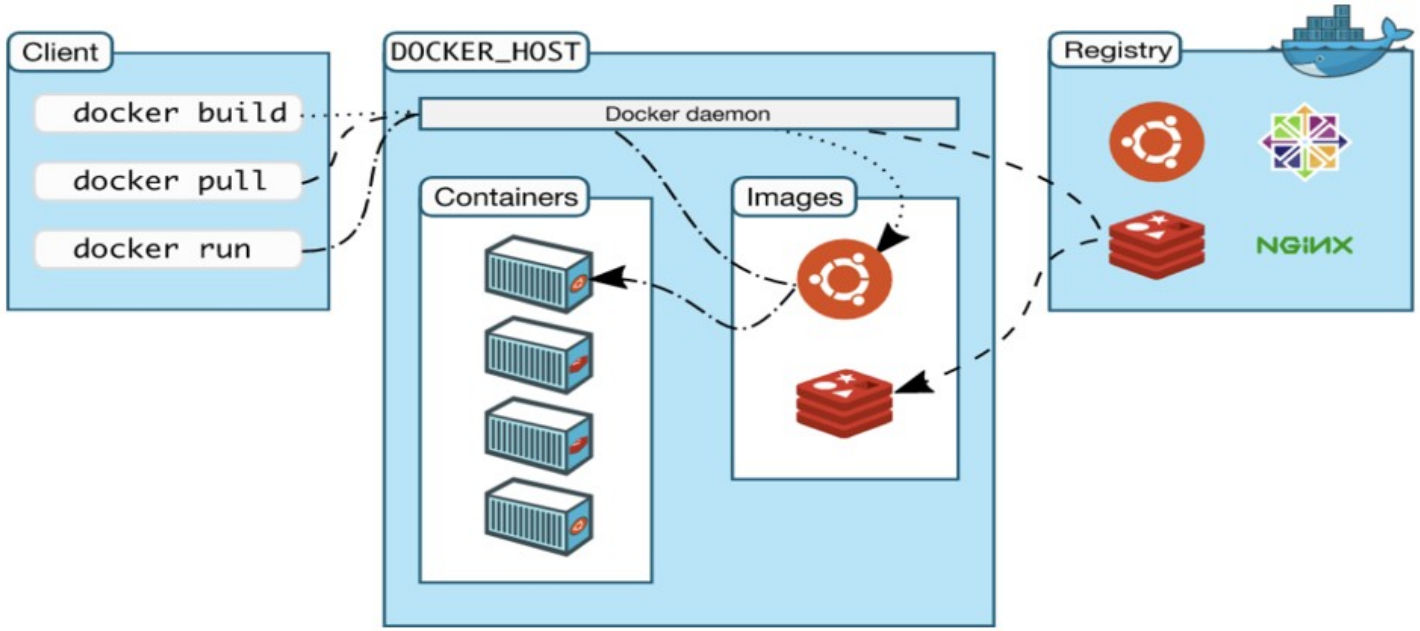
- Клиент — фронтенд docker приложения. Включает богатый CLI
- DOCKER_HOST — машина, где запущен бекенд docker. Включает в себя docker daemon, занимающийся мониторингом контейнеров. Вся связь с контейнерами происходит через демон.
- Images — образы, собранный из докерфайлов. Используются для создание контейнеров
- Контейнеры — некоторая файловая структура, представляющая собой изолированную систему

Терминология и общее устройство докер контейнеров



- **Client** — фронтенд docker приложения. Включает богатый CLI
- **DOCKER_HOST** — машина, где запущен бекенд docker. Включает в себя docker daemon, занимающийся мониторингом контейнеров. Вся связь с контейнерами происходит через демон.
- **Images** — образы, собранный из докерфайлов. Используются для создание контейнеров
- **Containers** — некоторая файловая структура, представляющая собой изолированную систему
- **Registry** — облачное хранилище образов

Терминология и общее устройство докер контейнеров



- **Client** — фронтенд docker приложения. Включает богатый CLI
- **DOCKER_HOST** — машина, где запущен бекенд docker. Включает в себя docker daemon, занимающийся мониторингом контейнеров. Вся связь с контейнерами происходит через демон.
- **Images** — образы, собранный из докерфайлов. Используются для создание контейнеров
- **Containers** — некоторая файловая структура, представляющая собой изолированную систему
- **Registry** — облачное хранилище образов

Структура докерфайлов

- Dockerfile — последовательность директив, на основе которых собирается образ

```
# Base image for our future container
FROM ubuntu:24.04

# Let's build app here for now
WORKDIR /root/app/

# Copy files from host to container
COPY ./ ./

# If dependencies need to be installed using some package manager, you first need to run update,
# since future container will be a freshly created system isolated from host, and no update from remote
# apt repositories exist
RUN apt-get update

# Install debs
RUN apt-get install -y libgtest-dev

# Enter container and run command
CMD ["./build.sh"]
```

Структура докерфайлов

- Dockerfile — последовательность директив, на основе которых собирается образ
- **FROM —** выбор образа, лежащего в основе. Без него докер образ будет абсолютно пустым и поставить туда что-то будет также проблематично

```
# Base image for our future container
FROM ubuntu:24.04

# Let's build app here for now
WORKDIR /root/app/

# Copy files from host to container
COPY ./ ./

# If dependencies need to be installed using some package manager, you first need to run update,
# since future container will be a freshly created system isolated from host, and no update from remote
# apt repositories exist
RUN apt-get update

# Install debs
RUN apt-get install -y libgtest-dev

# Enter container and run command
CMD ["./build.sh"]
```

Структура докерфайлов

- Dockerfile — последовательность директив, на основе которых собирается образ

- **FROM —** выбор образа, лежащего в основе. Без него докер образ будет абсолютно пустым и поставить туда что-то будет также проблематично
- **WORKDIR —** указание рабочей директории

```
# Base image for our future container
FROM ubuntu:24.04

# Let's build app here for now
WORKDIR /root/app/

# Copy files from host to container
COPY ./ ./

# If dependencies need to be installed using some package manager, you first need to run update,
# since future container will be a freshly created system isolated from host, and no update from remote
# apt repositories exist
RUN apt-get update

# Install debs
RUN apt-get install -y libgtest-dev

# Enter container and run command
CMD ["./build.sh"]
```

Структура докерфайлов

- Dockerfile — последовательность директив, на основе которых собирается образ

- **FROM —** выбор образа, лежащего в основе. Без него докер образ будет абсолютно пустым и поставить туда что-то будет также проблематично
- **WORKDIR —** указание рабочей директории
- **COPY —** копирование файлов с хоста в контейнер

```
# Base image for our future container
FROM ubuntu:24.04

# Let's build app here for now
WORKDIR /root/app/

# Copy files from host to container
COPY ./ ./

# If dependencies need to be installed using some package manager, you first need to run update,
# since future container will be a freshly created system isolated from host, and no update from remote
# apt repositories exist
RUN apt-get update

# Install debs
RUN apt-get install -y libgtest-dev

# Enter container and run command
CMD ["./build.sh"]
```

Структура докерфайлов

- Dockerfile — последовательность директив, на основе которых собирается образ

- **FROM —** выбор образа, лежащего в основе. Без него докер образ будет абсолютно пустым и поставить туда что-то будет также проблематично
- **WORKDIR —** указание рабочей директории
- **COPY —** копирование файлов с хоста в контейнер
- **RUN —** запуск команд **внутри предварительного образа**

```
# Base image for our future container
FROM ubuntu:24.04

# Let's build app here for now
WORKDIR /root/app/

# Copy files from host to container
COPY ./ ./

# If dependencies need to be installed using some package manager, you first need to run update,
# since future container will be a freshly created system isolated from host, and no update from remote
# apt repositories exist
RUN apt-get update

# Install debs
RUN apt-get install -y libgtest-dev

# Enter container and run command
CMD ["./build.sh"]
```

Структура докерфайлов

- Dockerfile — последовательность директив, на основе которых собирается образ

- **FROM —** выбор образа, лежащего в основе. Без него докер образ будет абсолютно пустым и поставить туда что-то будет также проблематично
- **WORKDIR —** указание рабочей директории
- **COPY —** копирование файлов с хоста в контейнер
- **RUN —** запуск команд **внутри предварительного образа**
- **CMD —** команда, которая запускается при **запуске контейнера**

```
# Base image for our future container
FROM ubuntu:24.04

# Let's build app here for now
WORKDIR /root/app/

# Copy files from host to container
COPY ./ ./

# If dependencies need to be installed using some package manager, you first need to run update,
# since future container will be a freshly created system isolated from host, and no update from remote
# apt repositories exist
RUN apt-get update

# Install debs
RUN apt-get install -y libgtest-dev

# Enter container and run command
CMD ["./build.sh"]
```

Структура докерфайлов

- Собрать образ: **podman build <build_context>**
- Путь указывается к **build context**. Это может быть как путь локально на машине к директории с докерфайлом, так и гит репозиторий или реестр
- Пример: **podman build .**
  podman build

  https://github.com/user/myrepo.git

  podman build http://server/context.tar.gz

```
# Base image for our future container
FROM ubuntu:24.04

# Let's build app here for now
WORKDIR /root/app/

# Copy files from host to container
COPY ./ ./

# If dependencies need to be installed using some package manager, you first need to run update,
# since future container will be a freshly created system isolated from host, and no update from remote
# apt repositories exist
RUN apt-get update

# Install debs
RUN apt-get install -y libgtest-dev

# Enter container and run command
CMD ["./build.sh"]
```

Теггирование и публикация образов

- Для запуска контейнера в сыром виде нужно указать его прямой хеш, который получается из команды **podman build**

Теггирование и публикация образов

- Для запуска контейнера в сыром виде нужно указать его прямой хеш, который получается из команды **podman build**
- Альтернативно каждому имеджу задается **tag**

podman build . -t ubuntu_24.04_basic:1.0

Теггирование и публикация образов

- Для запуска контейнера в сыром виде нужно указать его прямой хеш, который получается из команды **podman build**
- Альтернативно каждому имеджу задается **tag**
- Теги являются способом удобного версионирования ваших образов
- Полный формат тега: **[HOST[:PORT_NUMBER]/]PATH[:TAG]**
- **Пример :** docker.io:443/library/nginx:1.0

podman build . -t ubuntu_24.04_basic:1.0

podman build . -t my-username/ubuntu_24.04_basic:1.0

Теггирование и публикация образов

- Для запуска контейнера в сыром виде нужно указать его прямой хеш, который получается из команды **podman build**
- Альтернативно каждому имеджу задается **tag**
- Теги являются способом удобного версионирования ваших образов
- Полный формат тега: **[HOST[:PORT_NUMBER]/]PATH[:TAG]**
- **Пример :** docker.io:443/library/nginx:1.0
- Для публикации готового образа достаточно сделать **podman push**

podman build . -t ubuntu_24.04_basic:1.0

podman build . -t my-username/ubuntu_24.04_basic:1.0

Некоторые подходы к проектированию контейнеров

- Был рассмотрен вариант с копированием исходного кода в контейнер и последующим запуском команды в нем при старте

Некоторые подходы к проектированию контейнеров

- Был рассмотрен вариант с копированием исходного кода в контейнер и последующим запуском команды в нем при старте
- Чем данный подход хорош, а чем он плох?

Некоторые подходы к проектированию контейнеров

- Был рассмотрен вариант с копированием исходного кода в контейнер и последующим запуском команды в нем при старте
- Чем данный подход хорош, а чем он плох?

Плюсы:

- Сборка и тестирование проекта в один **podman run**
- Полная изоляция включая файловую систему

Некоторые подходы к проектированию контейнеров

- Был рассмотрен вариант с копированием исходного кода в контейнер и последующим запуском команды в нем при старте
- Чем данный подход хорош, а чем он плох?

Плюсы:

- Сборка и тестирование проекта в один **podman run**
- Полная изоляция включая файловую систему

Минусы:

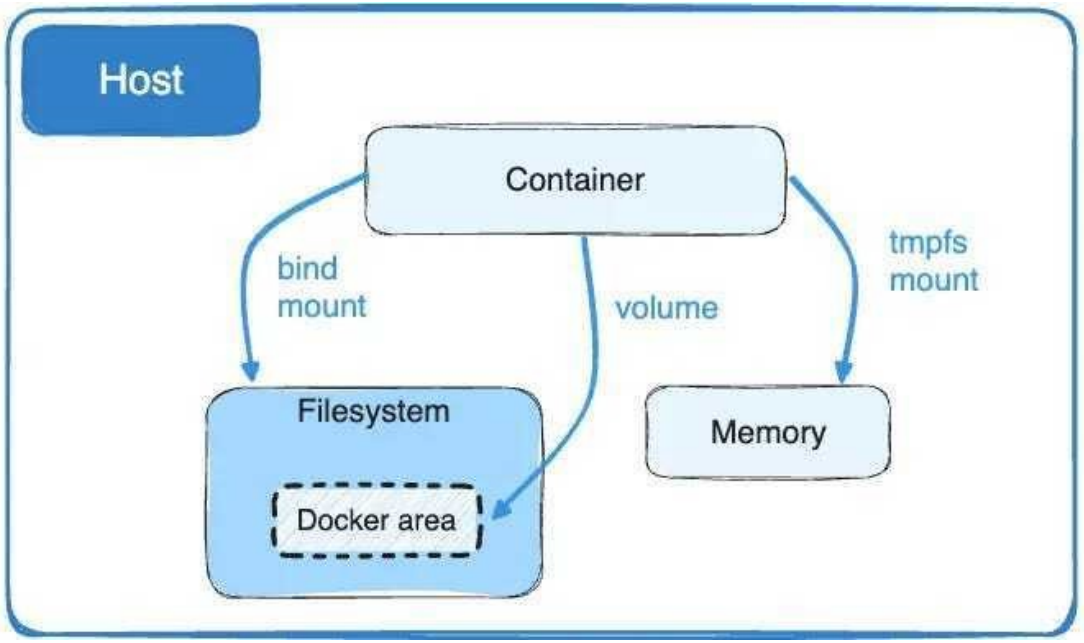
- Необходимо менять докерфайл при изменении команд для запуска (а также перестраивать образ)
- Все сорцы проекта будут дублироваться в контейнер

Некоторые подходы к проектированию контейнеров

- Альтернативный вариант — все, что относится к сорцам проекта, не копировать в контейнер, а **маунтить**

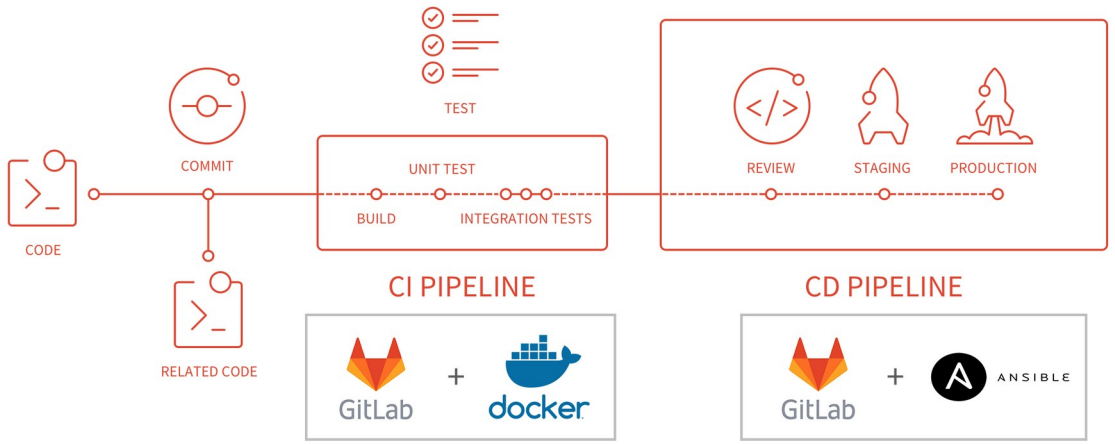
Некоторые подходы к проектированию контейнеров

- Альтернативный вариант — все, что относится к сорцам проекта, не копировать в контейнер, а **маунтить**
- **Bind mounts —** прямой маунт на хостовую машину. Любые изменения в замаученной директории отображаются в контейнере, или наоборот.
- **Volumes —** работают так же как **bind mounts** с точки зрения результата, но создают собственное пространство на файловой системе



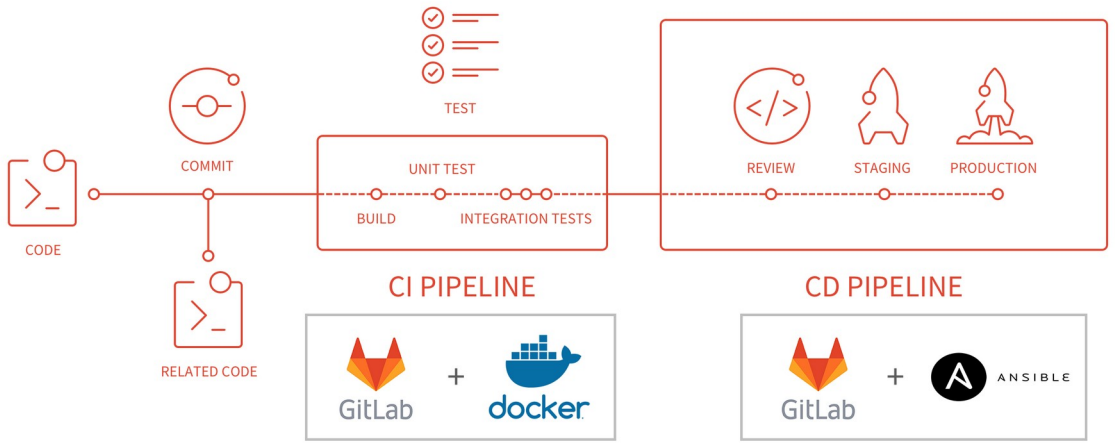
Использование контейнеров в CI

- Контейнеры, как мы уже обсуждали, решают воспроизводимости в пространстве



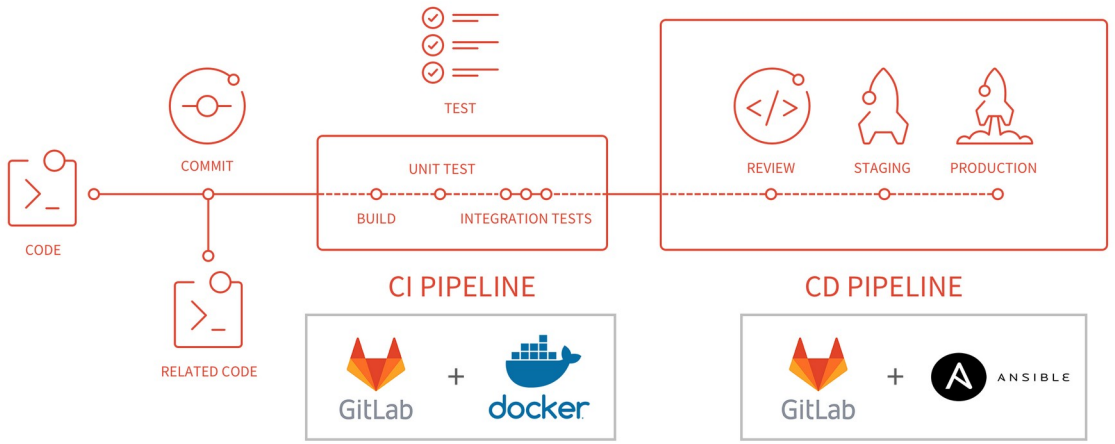
Использование контейнеров в CI

- Контейнеры, как мы уже обсуждали, решают воспроизводимости в пространстве
- Вопрос в том, где нам хранить докерфайлы и где строить образы?



Использование контейнеров в CI

- Контейнеры, как мы уже обсуждали, решают воспроизводимости в пространстве
- Вопрос в том, где нам хранить докерфайлы и где строить образы?
- В зависимости от подхода есть два варианта — либо хранить их в репозитории и в CI собирать образ, копируя в него исходники — для большей изоляции



Использование контейнеров в CI

- Контейнеры, как мы уже обсуждали, решают воспроизводимости в пространстве
- Вопрос в том, где нам хранить докерфайлы и где строить образы?
- В зависимости от подхода есть два варианта — либо хранить их в репозитории и в CI собирать образ, копируя в него исходники — для большей изоляции
- Либо же собирать их в отедельном репозитории и выкладывать в реестр