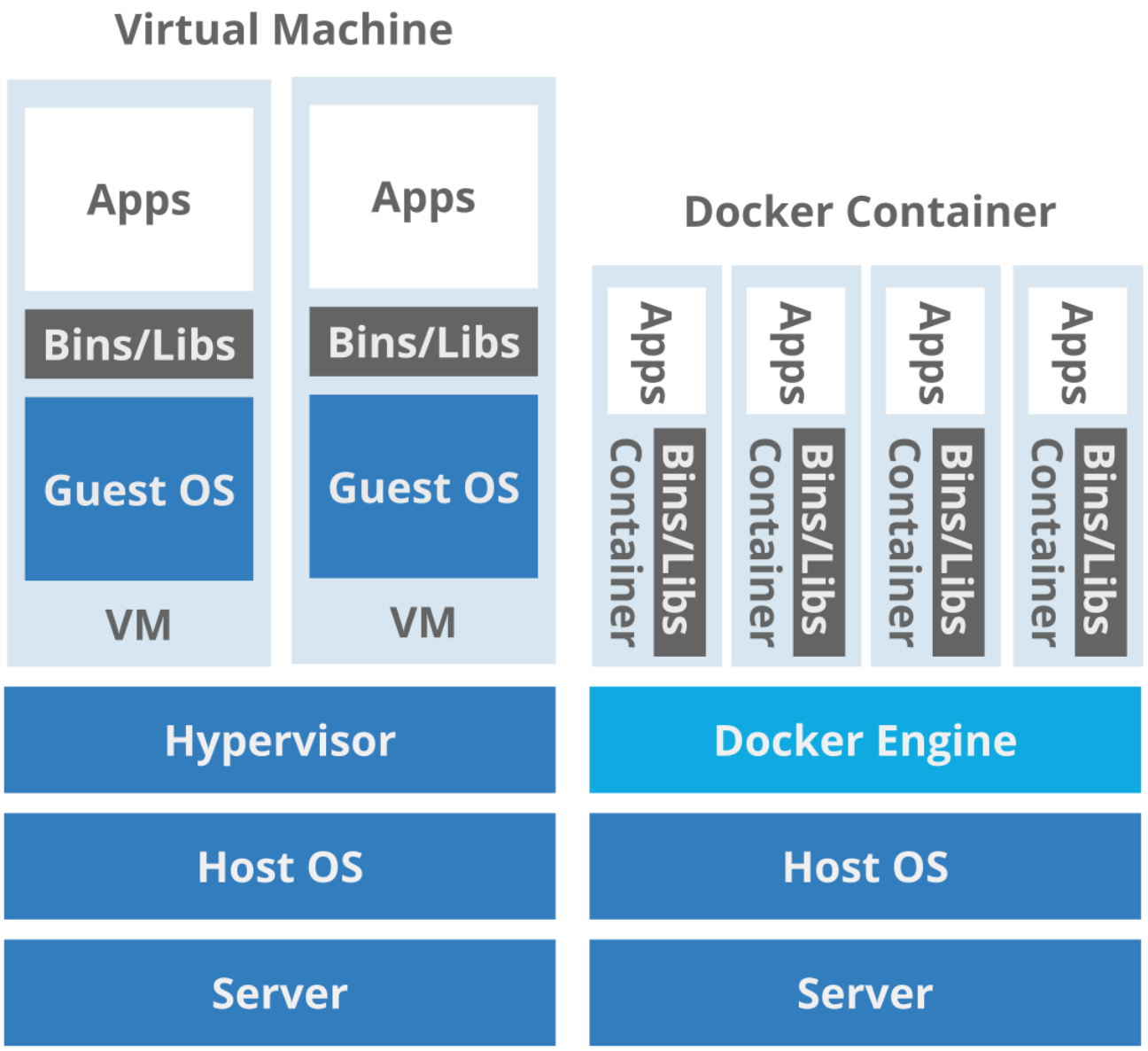
Ссылки

- Дока https://docs.docker.com
- Лабы по docker https://labs.play-with-docker.com
- Ресурс по обучению от создателей этих лаб https://training.play-with-docker.com

Об архитектуре



Плюсы - компактность, эффективность, версионируемость.
Минусы - меньшая изоляция, "приявязка" к архитектуре Hosl OS.

Контейнеризация основана на cgroups, network name sapces, Union файловая система (например, OverlayFS).

Основные понятия

Основные понятия Docker включают ключевые элементы и термины, которые описывают архитектуру и работу с этой платформой контейнеризации:

- **Docker-платформа** — программа для упаковки и запуска приложений в контейнерах с их зависимостями, обеспечивающая переносимость и воспроизводимость окружения.
- **Docker Engine** — клиент-серверное приложение, состоящее из Docker Daemon (демона), Docker Client (клиента) и REST API, которое управляет контейнерами, образами, сетями и томами.
- **Docker Client** — основной интерфейс командной строки (CLI), через который пользователь взаимодействует с Docker Engine, отправляя команды Docker Daemon.
- **Docker Daemon** — сервер, обрабатывающий запросы API, который создает, запускает и управляет контейнерами и образами.
- **Docker Image** (образ) — неизменяемый образ файловой системы с приложением и его зависимостями, из которого создаются Docker Container.
- **Docker Container** (контейнер) — запущенный экземпляр образа, изолированное пространство для выполнения приложения, использующий ресурсы хоста и изолирующее процессы между собой.
- **Dockerfile** — текстовый файл с инструкциями для создания Docker Image, описывающий шаги конфигурации контейнера.
- **Volume (том)** — механизм для постоянного хранения данных, чтобы сохранить данные вне контейнера, обеспечивая их сохранность при перезапуске или удалении контейнера.
- **Docker Registry (реестр)** — удаленное хранилище Docker Image, куда можно загрузить и откуда можно скачать образы; самый известный — Docker Hub (https://hub.docker.com).
- **Docker Repository** (репозиторий) — коллекция образов с одинаковым именем, но разными тегами (версиями), например, Python с тегами разных версий образов.

Дополнительно:

- Tag (тег) — идентификатор версии образа в репозитории.
- Network (сеть) — механизм конфигурирования сетевых подключений между контейнерами и внешним миром.
- Docker Compose — инструмент для описания и запуска многоконтейнерных приложений с помощью файла конфигурации YAML.
- Layer (слой) — отдельный уровень файловой системы в Docker Image, обеспечивающий эффективное хранение и передачу образов.

- Sandbox (песочница) — изолированная среда контейнера для безопасного запуска приложений.

Сценарий lets start

01. Установка

https://docs.docker.com/engine/install/debian/

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
sudo tee /etc/apt/sources.list.d/docker.sources <<EOF
Types: deb
URIs: https://download.docker.com/linux/debian
Suites: $(. /etc/os-release && echo "$VERSION_CODENAME")
Components: stable
Signed-By: /etc/apt/keyrings/docker.asc
EOF

sudo apt-get update
```

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin
```

- **docker-ce (Docker Community Edition)** — основной пакет Docker, включает в себя сам Docker Engine (демон), который управляет контейнерами и образами. Это базовая платформа для создания, запуска и управления контейнерами.
- **docker-ce-cli** — клиентская часть Docker, предоставляющая командный интерфейс (Docker CLI) для взаимодействия с Docker Engine. Позволяет выполнять команды, управлять контейнерами и образами из терминала.
- **containerd.io** — низкоуровневая среда выполнения контейнеров, которая фактически занимается управлением жизненным циклом контейнера (создание, запуск, остановка). Docker Engine использует containerd для работы с контейнерами на уровне ядра.
- **docker-buildx-plugin** — расширенный плагин для сборки образов Docker. Поддерживает кросс-сборку, многоархитектурные (multi-platform) образы и улучшенные возможности сборки, выходящие за рамки стандартного docker build.

- **docker-compose-plugin** — официальный плагин для запуска мультиконтейнерных приложений с помощью команды `docker compose` (замена устаревшей команды `docker-compose`). Позволяет описывать и запускать несколько контейнеров, их сети и тома из одного YAML-файла.

02. Проверка

```
systemctl status docker
docker run hello-world
```

03. Запускаем первый контейнер

```
#что как?
docker system info

#качаем имадж
docker pull nginx
docker pull ubuntu
docker images

#создаем контейнер
docker create nginx
docker ps
docker ps -a

# запуск - остановка созданного контейнера
docker start <container_ID>
docker stop <container_ID>

# создание и запуск контейнера RUN
# docker run [опции] <имя_образа> [аргументы]
docker run -d nginx # в фоне
docker run -it ubuntu bash # интеркативно
docker run --rm ubuntu echo "Hello, Docker!"
docker run --name my_nginx nginx
docker run --memory=512m --cpus=1 nginx
```

04. Меняем контейнер и делаем свой образ

```
docker exec -it <container_ID> bash && apt install -y nano exit
docker commit <container_ID> my_custom_image
docker run -it my_custom_image bash
```

05. Работаем внутри контейнера снаружи

```
docker exec -it <container_ID> bash
docker exec <container_ID> ls /

# для интерактиванолго уонтейнера
docker attach <container_ID>
# Ctrl+C остановить контейнер, Ctrl+P и  Ctrl+Q просто выйти
```

06. Файлы для контейнера

Монтирование :

```
docker run -d -v ~/realdir:/app ubuntu
```

Нормально, через volume (/var/lib/docker/volumes/):

```
docker volume create my_data
docker run -d -v my_data:/data ubuntu

docker volume ls
docker volume rm my_data

# Скопировать в корень контейнера file
docker cp file <containerID>:/

# Скопировать file из корня контейнера в текущую директорию командной строки
docker cp <containerID>:/file .
```

07. Основы сети в контейнере

Один контейнер

```
docker run -d -p 8080:80 nginx
```

В браузере http://localhost:8080 или явно

```
docker run -d -p 192.168.1.100:8080:80 nginx
```

Сеть для контейнеров

```
# docker network create my_network
# docker run -d...--network=my_network my_app_image
```

```
docker run -dit --name u1 ubuntu
docker run -dit --name u2 ubuntu
docker network create my_network
docker network ls

docker network inspect my_network
docker network connect my_network <container_ID>
# docker network disconnect my_network <container_ID>
```

08. Dockerfile

Зачем?

Если стандартные образы из реестра не подходят для ваших задач, можно создать свой собственный образ с помощью Dockerfile. Этот файл описывает инструкции для создания образа, включая базовый образ, команды для установки зависимостей и копирования файлов проекта. Можно запушить в Docker Hub с помощью команды docker push.

Особенности структуры:

- Инструкции обрабатываются сверху вниз.
- Каждая инструкция создаёт новый слой в образе.
- Если сборка прерывается, при повторном запуске Docker использует кешированные слои до места изменения.

Команды

Некоторые инструкции, которые используются в Dockerfile:

- **FROM** — указывает базовый образ, с которого следует начать сборку.
- **RUN** — выполняет команды в процессе создания образа, например, установку зависимостей или запуск сценариев.
- **WORKDIR** — задаёт рабочую директорию внутри контейнера, где будут выполняться остальные команды.
- **COPY** — копирует файлы и директории с хоста в образ.
- **ENV** — устанавливает переменные окружения внутри контейнера.
- **EXPOSE** — указывает, какие порты будет использовать контейнер, но не пробрасывает их автоматически.
- **CMD** — команда для запуска приложения, выполняется при старте контейнера.

Пример простого Dockerfile

```

# Используем официальный базовый образ Ubuntu
```

```
FROM ubuntu:22.04
# Обновляем индекс пакетов и устанавливаем нужные утилиты
RUN apt-get update && apt-get install -y iproute2 iputils-ping iptraf nmap &&
apt-get clean
# Указываем команду по умолчанию (можно заменить по нужде)
CMD ["/bin/bash"]
```

Создание образа:

```
docker build -f /path/to/file -t my-ubuntu-app .
```

по умолчанию файл называется Dockerfile.

Полезные команды

https://docs.docker.com/reference/cli/docker/

Управление контейнерами (Container Management)

- **docker ps -a** - Посмотреть все контейнеры
- **docker start container-name** - Запустить контейнер
- **docker kill/stop container-name** - Убить (SIGKILL) /Остановить (SIGTERM) контейнер
- **docker logs --tail 100 container-name** - Вывести логи контейнера, последние 100 строк
- **docker inspect container-name** - Вся инфа о контейнере + IP
- **docker rm container-name**- Удалить контейнер (поле каждой сборки Dockerfile)
- **docker rm -f $(docker ps -aq)** - Удалить все запущенные и остановленные контейнеры
- **docker events container-name** - Получения событий с контейнера в реальном времени.
- **docker port container-name** - Показать публичный порт контейнера
- **docker top container-name** - Отобразить процессы в контейнере
- **docker stats container-name** - Статистика использования ресурсов в контейнере
- **docker diff container-name** - Изменения в ФС контейнера

Запуск docker (Run)

`--rm` удалит после закрытия контейнера,
`--restart unless-stopped` добавит автозапуск контейнера)
-Запуск контейнера интерактивно или как демона/detached (`-d`),
Порты: слева хостовая система, справа в контейнере,

- **docker run -d -p 80:80 -p 22:22 debian:11.1-slim sleep infinity**
- **docker update --restart unless-stopped redis** - добавит к контейнеру правило перезапускаться при закрытии, за исключением команды стоп, это простой автозапуск
- **docker exec -it container-name /bin/bash** - интерактивно подключиться к контейнеру для управления, exit чтобы выйти
- **docker attach container-name** - подключиться к контейнеру чтоб мониторить ошибки логи в реалтайме

Управление образами (Images Management)

- **docker build -t my_app .** - Билд контейнера в текущей папке, Скачивает все слои для запуска образа
- **docker images / docker image ls** - Показать все образы в системе
- **docker image rm / docker rmi image** - Удалить image
- **docker insert URL** - вставляет файл из URL в контейнер
- **docker save -o backup.tar** - Сохранить образ в backup.tar в STDOUT с тегами, версиями, слоями
- **docker load -i backup.tar** - Загрузить образ в .tar в STDIN с тегами, версиями, слоями
- **docker import** - Создать образ из .tar
- **docker image history --no-trunc** - Посмотреть историю слоёв образа
- **docker system prune -f** - Удалит все, кроме используемого (лучше не использовать на проде, ещё кстати из-за старого кеша может собираться старая версия контейнера)