Docker: Основные команды

Работу подготовили студентки группы 5040102/50101

Ежова Елена и Кац Софья

Что такое Docker?

- **Docker** — это платформа для разработки, доставки и запуска приложений в Контейнерах.
- **Контейнер** — это стандартизированная единица программного обеспечения, которая инкапсулирует приложение и все его зависимости.
- Пример: «У меня на машине работает, а на сервере — нет». Объяснение: различия в версиях ОС, библиотек, зависимостей между средами (разработка, тестирование, продакшн).

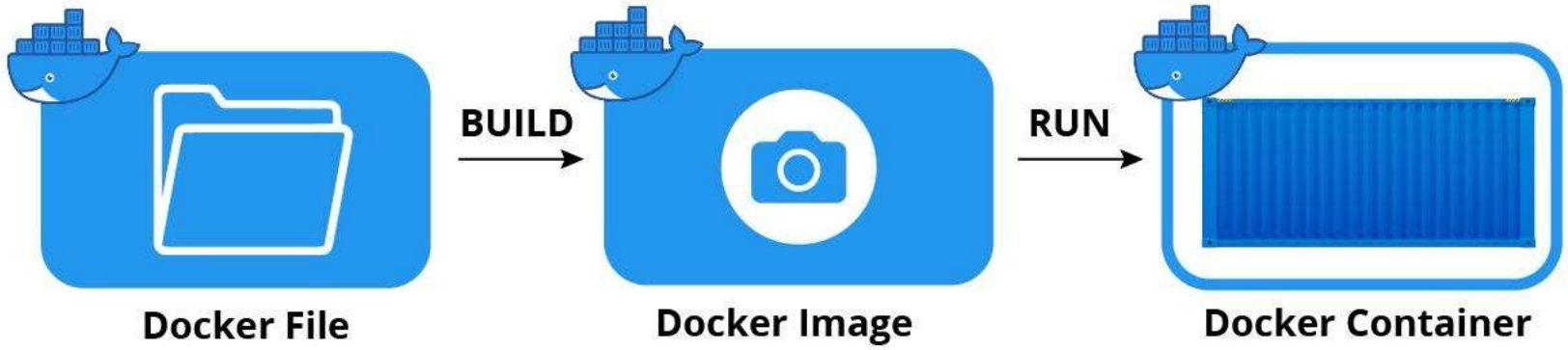
Ключевые концепции: Image и Container

Docker Image:

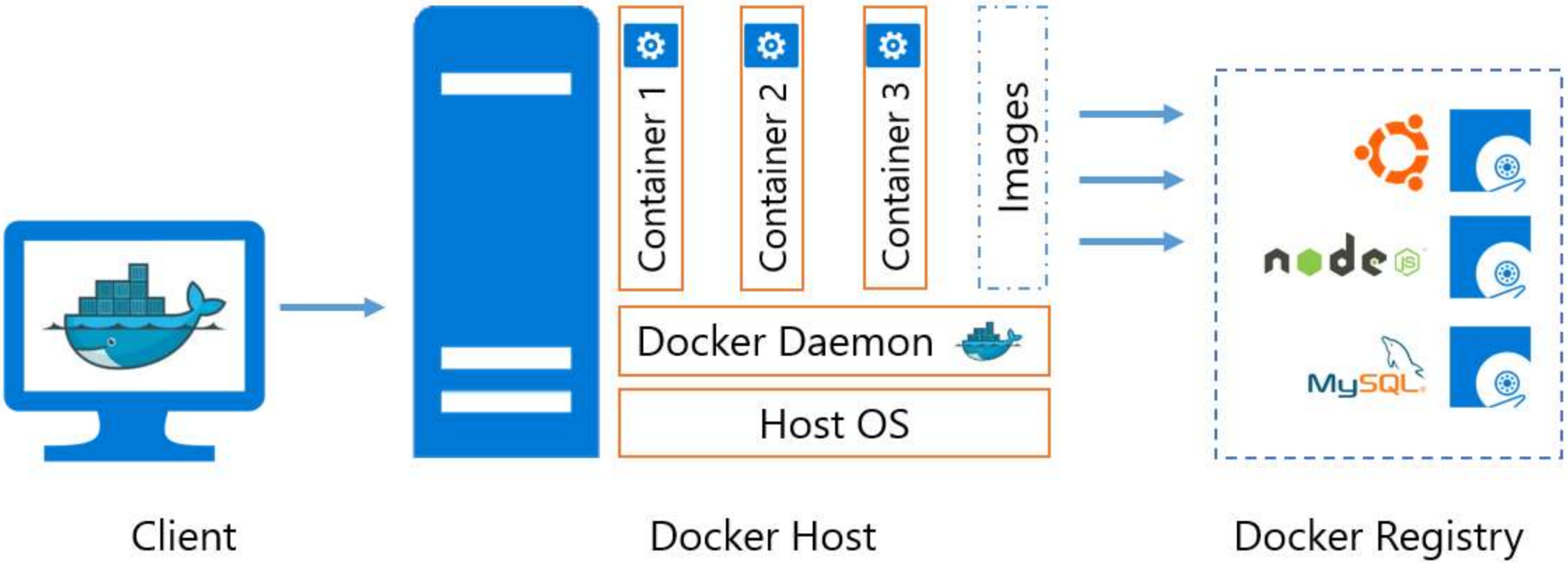
- Неизменяемый шаблон, читаемый пакет с приложением и средой
- Создается на основе DockerFile
- Хранится в реестрах

Docker Container:

- Запущенный, работающий экземпляр Образа
- Имеет собственное файловое пространство, сеть и изолированные процессы
- Жизненный цикл: создание, запуск, остановка, удаление



Архитектура Docker



Работа с Образами

- *docker pull* – загрузка Образа из реестра
  - Пример: *docker pull nginx:latest*
  - Что делает: Подгружает Образ Nginx с тегом latest с Docker Hub
- *docker images* – просмотр списка скачанных Образов на локальной машине
  - Пример: *docker images* или *docker image ls*
  - Вывод: Покажет REPOSITORY, TAG, IMAGE ID, SIZE
- *docker search* – поиск образов в реестре
  - Пример: *docker search nginx*
  - Вывод: Покажет NAME, DESCRIPTION, STARS, OFFICIAL, AUTOMATED
- *docker build* – создание Образа из DockerFile
  - Пример: *docker build -t my-app:1.0 .*
  - Что делает: дает имя и тег Образу
- *docker rmi* – удаление образа
  - Пример: *docker rmi <image_id>* или *docker rmi nginx:latest*
  - Примечание: Образ не будет удален при запущенном Контейнере

Создание DockerFile

```
FROM python:3.11-slim  # Используем официальный Python-образ как родительский
WORKDIR /app   # Устанавливаем рабочую директорию в контейнере
COPY requirements.txt   # Копируем файл с зависимостями
RUN pip install -r requirements.txt  # Устанавливаем зависимости
COPY . . # Копируем весь исходный код
ENV FLASK_APP=app.py   # Определяем переменную окружения
EXPOSE 5000 # Сообщаем, на каком порту будет работать контейнер
CMD ["flask", "run", "--host=0.0.0.0"]  # Команда для запуска приложения
```

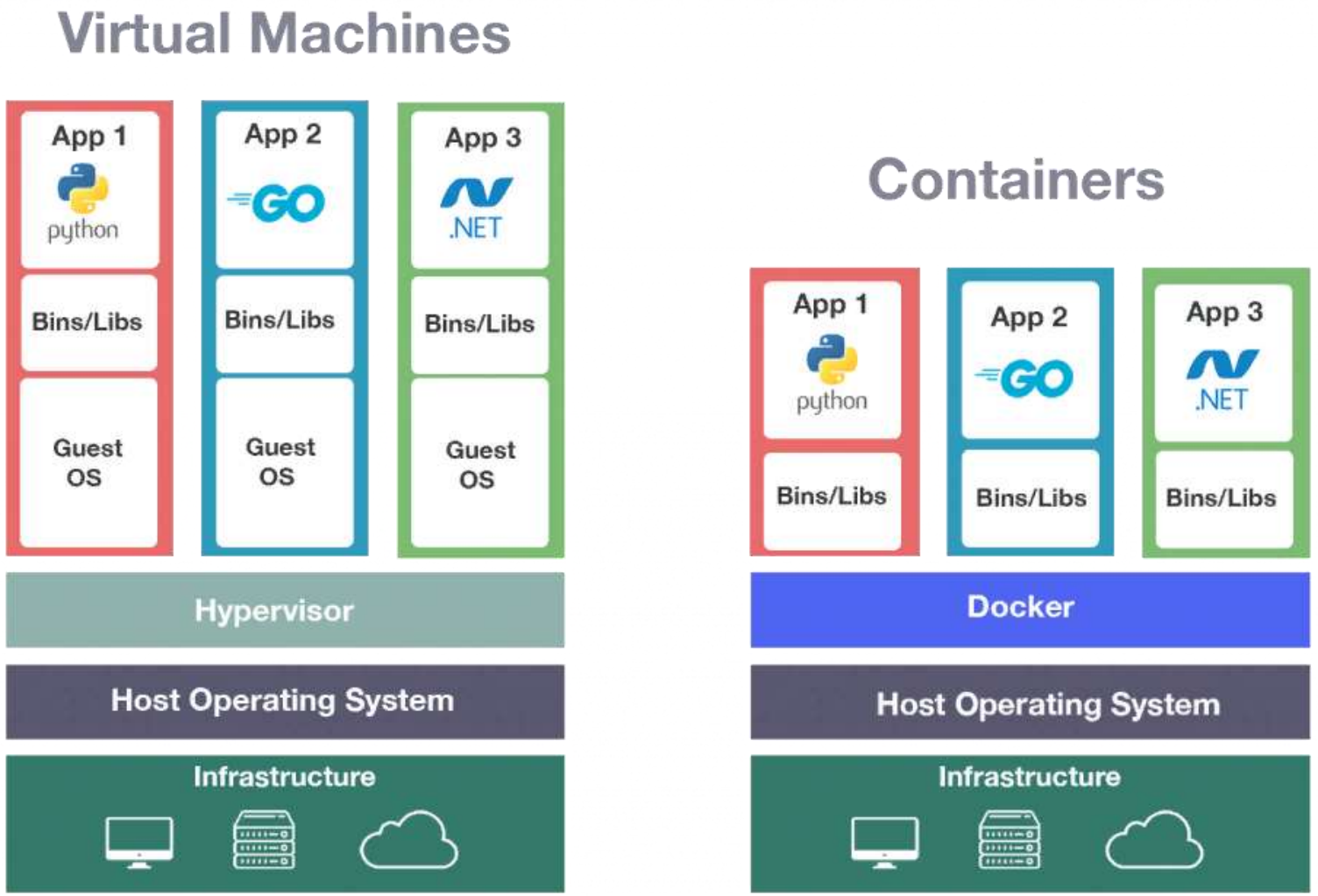
Основные команды для Контейнеров

- *docker run* – создает и запускает новый Контейнер из Образа
  - Базовый пример: *docker run nginx*
  - Пример с ключами: *docker run -d --name my-nginx -p 8080:80 nginx*
- *docker ps* – список запущенных Контейнеров
  - *docker ps -a* – список всех Контейнеров (включая остановленные)
- *docker stop* – остановка Контейнера
- *docker start* – запуск остановленного Контейнера
- *docker rm* – удаление остановленного Контейнера
  - *docker rm -f* – принудительное удаление запущенного Контейнера

Основные команды для Контейнеров

- *docker stats* – live-статистика по использованию ресурсов (CPU, память, сеть)
- *docker restart* – перезапуск
- *docker pause/unpause* – приостановка/возобновление процессов
- *docker kill* – немедленная остановка (отправляет SIGKILL)

Контейнеры vs. Виртуальные машины



Практический пример: Запускаем веб-сервер

Цель: Запустить контейнер с Nginx и получить доступ к нему из браузера

Шаги:

docker pull nginx:alpine

docker run -d --name my-web-server -p 8080:80 nginx:alpine

Открываем браузер и переходим по адресу http://locahost:8080

Видим страницу "Welcome to nginx!"

docker stop my-web-server

docker rm my-web-server

Команды для диагностики и отладки

- *docker logs* – просмотр логов Контейнера
  - Базовый пример: *docker logs my-nginx*
  - *docker logs -f my-nginx* – “подписываться” на вывод логов в реальном времени
- *docker exec* – выполнение команды внутри запущенного Контейнера
  - *docker exec -it my-nginx /bin/bash* – попадаете в командную строку (bash) внутри контейнера
- *docker inspect* – детальная информация о контейнере в JSON
  - *docker inspect my-db | grep “IPAddress”* – попадаете в командную строку (bash) внутри контейнера

Работа с данными: Тома и Монтирование

Проблема: данные в Контейнере непостоянны; удаление Контейнера влечет удаление данных

- Решение 1: Тома (Volumes)
  - Управляемый Docker'ом механизм хранения данных.
  - Создание: *docker volume create my_volume*
  - Использование: *docker run -v my_volume:/var/lib/mysql mysql*
  - Просмотр: *docker volume ls*
- Решение 2: Монтирование (Blind Mounts)
  - Монтирование конкретной директории с хоста в контейнер.
  - Использование: *docker run -v /home/user/app:/app my-app*
  - Идеально для разработки (изменения кода на хосте сразу видны в контейнере).

Сети в Docker

Как Контейнеры общаются друг с другом и с внешним миром?

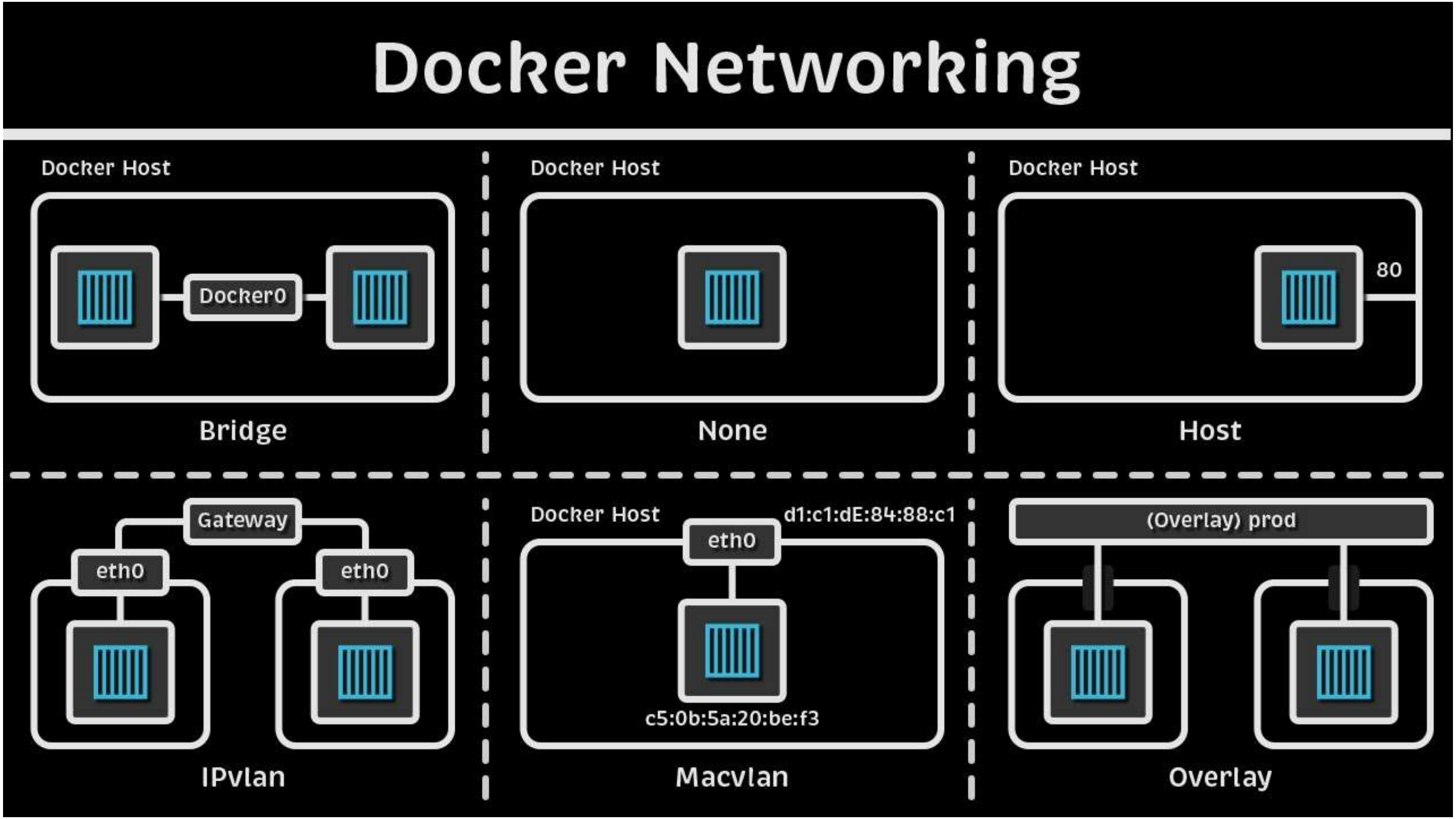
Сетевые драйверы:

- *bridge* (по умолчанию): Создаётся виртуальная внутренняя сеть. Контейнеры видят друг друга по имени
- *host*: Контейнер использует сетевой стек хоста (высокая производительность, нет изоляции)
- *none*: Отсутствие сети

Основные команды:

- *docker network ls* – вывести список сетей
- *docker network create my_network* – создать свою сеть
- *docker run --network=my_network --name app1 my-app* – запустить Контейнер в конкретной сети
- Теперь другой контейнер в этой сети может подключиться к *app1* по имени

Docker Networking



«Уборка» и оптимизация в Docker

Команды для очистки:

- *docker container prune* – удалить все остановленные Контейнеры
- *docker image prune* – удалить "висячие" Образы
- *docker image prune -a* – удалить все Образы, не используемые запущенными Контейнерами
- *docker volume prune* – удалить неиспользуемые Тома
- *docker system prune -a* – полная очистка

Мониторинг ресурсов:

- *docker system df* – аналог df для Docker