Docker
внутреннее устройство

Презентацию подготовили:

Бабахина Софья
Липс Екатерина

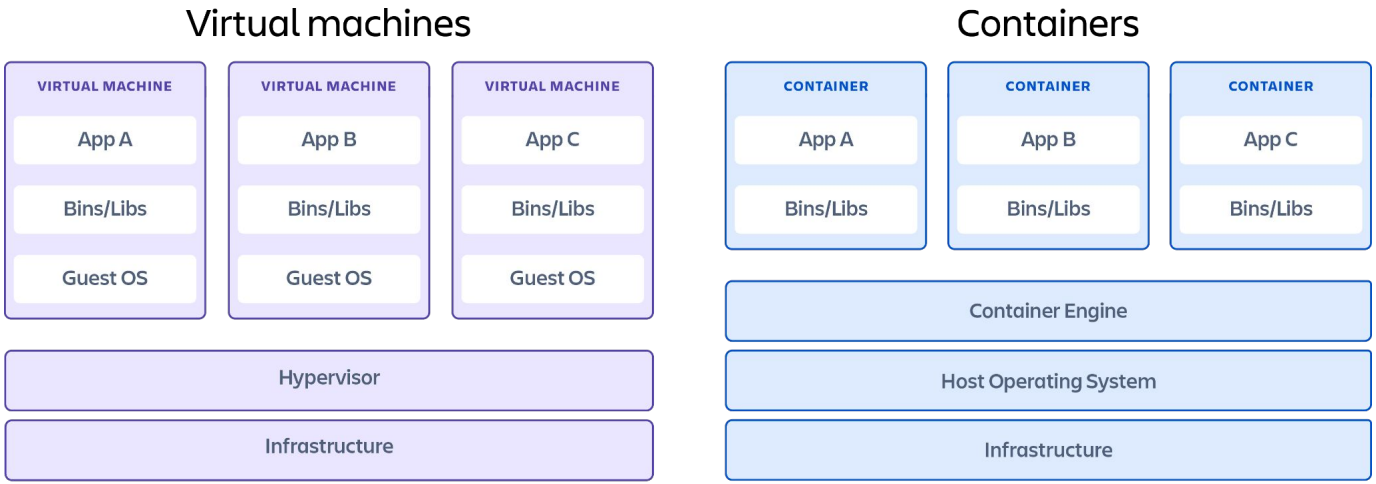
гр. 5040102/50201

Контейнеры vs Виртуальные Машины

Виртуализация — это процесс, при котором один системный ресурс, такой как оперативная память, ЦП, диск или сеть, может быть виртуализирован и представлен в виде множества ресурсов.

Основное различие контейнеров и виртуальных машин (ВМ):

ВМ виртуализируют весь компьютер вплоть до аппаратных уровней, а **контейнеры** — только программные уровни выше уровня операционной системы.



Контейнеры vs Виртуальные Машины

Контейнеры — это легкие программные пакеты, содержащие все **зависимости**, необходимые для запуска автономного программного приложения:

- системные библиотеки
- сторонние пакеты кода
- другие приложения уровня операционной системы

Плюсы:

- Скорость итерации
- Надежная экосистема

Минусы:

- Уязвимость общего узла

Контейнеры vs Виртуальные Машины

Виртуальные машины — это тяжелые программные пакеты, которые обеспечивают полную эмуляцию низкоуровневых аппаратных устройств, таких как ЦП, дисковые и сетевые устройства. Виртуальная машина также может включать дополнительный программный стек для запуска на эмулируемых аппаратных средствах. Такой пакеты аппаратных и программных средств представляет собой полнофункциональный снимок вычислительной системы.

Плюсы:

- Полная защита путем изоляции
- Интерактивная разработка

Минусы:

- Уязвимость общего узла

Архитектура Docker

Docker – это платформа, которая позволяет упаковать в контейнер приложение со всем окружением и зависимостями, а затем доставить и запустить его в целевой системе.

Компоненты Docker:

- **Docker host** – это операционная система, на которую устанавливают Docker и на которой он работает.
- **Docker daemon** – служба, которая управляет Docker-объектами: сетями, хранилищами, образами и контейнерами.
- **Docker client** – консольный клиент, при помощи которого пользователи взаимодействуют с Docker daemon и отправляют ему команды, создают контейнеры и управляют ими.
- **Docker image** – это неизменяемый образ (read-only шаблон), из которого разворачивается контейнер.
- **Docker container** – развёрнутое и запущенное с помощью Docker приложение.
- **Docker Registry** – репозиторий, в котором хранятся образы.
- **Dockerfile** – файл-инструкция для сборки образа.
- **Docker Compose** – инструмент для управления несколькими контейнерами. Он позволяет создавать контейнеры и задавать их конфигурацию.
- **Docker Desktop** – GUI-клиент, который распространяется по GPL.

Что происходит при docker run

```
docker run hello-world
```

Команда **`docker run`** указывает Docker запустить приложение в контейнере.

Что делает Docker для запуска приложения:

1. Скачивает образ hello-world;
2. Создает контейнер;
3. Инициализирует файловую систему и монтирует read-only уровень;
4. Инициализирует сеть/мост;
5. Устанавливает IP адреса;
6. Запускает указанный процесс;
7. Обрабатывает и выдает вывод вашего приложения.

Что происходит при docker run

Dockerfile

```
FROM busybox:latest
CMD echo Hello World!
```

Cmd

```
docker image build -t hello-world .
docker run hello-world
```

```
root@otus-virtual-machine:~/hello-world# docker build -t hello-world .
Sending build context to Docker daemon  2.048kB
Step 1/2 : FROM busybox:latest
 ---> 7cfbbec8963d
Step 2/2 : CMD echo Hello World!
 ---> Using cache
 ---> d7585ce8251b
Successfully built d7585ce8251b
Successfully tagged hello-world:latest
root@otus-virtual-machine:~/hello-world# docker run -d hello-world
4cc99ff54a783b576a64d087408b80092bd3e54bfe2f64635904e770c0937980
```

Изоляция: namespaces, cgroups

Docker использует технологию **namespaces** для организации изолированных рабочих пространств, которые мы называем контейнерами.

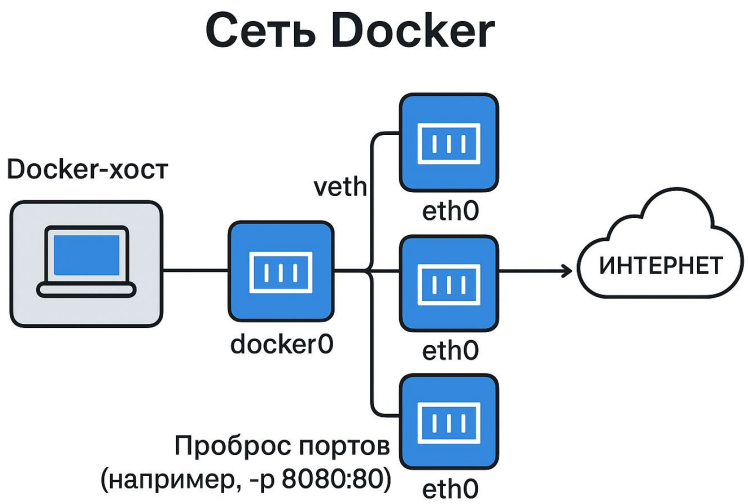
Список некоторых пространств имен, которые использует docker:

- **pid:** для изоляции процесса;
- **net:** для управления сетевыми интерфейсами;
- **ipc:** для управления IPC ресурсами (ICP: InterProccess Communication);
- **mnt:** для управления точками монтирования;
- **utc:** для изолирования ядра и контроля генерации версий(UTC: Unix timesharing system).

Docker также использует технологию **cgroups** или контрольные группы, которые позволяют разделять доступные ресурсы железа и если необходимо, устанавливать пределы и ограничения.

Сеть Docker

1. Docker создаёт виртуальную сеть для контейнеров
2. Каждый контейнер получает виртуальный сетевой интерфейс (eth0).
3. Все контейнеры подключаются к виртуальному мосту docker0 (как к роутеру).
4. Выход в интернет через NAT — весь трафик идёт через хост.
5. Порты пробрасываются через -p (например, -p 8080:80).



Типы сетей Docker

Тип сети	Что делает	Когда использовать
bridge	Создаёт общий мост docker0. Контейнеры видят друг друга и могут выйти в интернет.	Подходит для большинства случаев.
host	Контейнер использует сеть хоста напрямую, без изоляции.	Когда нужно максимум скорости и минимальные задержки (например, для игр или систем мониторинга).
none	У контейнера вообще нет сети.	Для тестов или высокой безопасности.
overlay	Объединяет контейнеры на разных серверах в одну виртуальную сеть.	Используется в кластерах (Docker Swarm, Kubernetes).

Три способа хранить данные в Docker

Тип	Где хранятся данные	Когда использовать	Примечание
Volume (Том)	В специальной директории, которой управляет Docker (/var/lib/docker/volumes/...)	Для постоянных данных, например база данных или файлы приложения	Docker сам управляет этим хранилищем
Bind mount	Прямая ссылка на папку локального компьютера (например, C:\project или /home/user/app)	Удобно при разработке, когда нужно, чтобы изменения в коде на хосте сразу появлялись в контейнере	Требует аккуратности с правами доступа
tmpfs	В оперативной памяти (RAM), ничего не пишется на диск	Для временных или чувствительных данных, которые не должны сохраняться (пароли, кеш, временные файлы)	Исчезает при остановке контейнера

Безопасность Docker

- **Два уровня безопасности**
  a. Build time (сборка образа) — важно, что попадает в образ.
  b. Runtime (запуск контейнера) — важно, с какими правами он работает.

- **Комбинация**
  - Минимальный образ
  - Отсутствие секретов
  - Ограничения прав
  - Подписи и сканирование

Этап	Что делать?	Зачем делать?
Сборка	Минимальные образы, multistage, без секретов	Меньше уязвимостей
Запуск	Не root, read-only, ограниченные права	Меньше рисков выхода за пределы
Проверка	Подписи, сканирование, SBOM	Контроль происхождения и безопасности

Производительность и отладка

Контейнеры делят ресурсы с хостом, поэтому узкие места обычно связаны с:

1. Файловой системой
2. Сетью
3. Логированием
4. Ограничениями ресурсов

Параметр	Что делает
--cpus	ограничивает число доступных CPU
--memory	ограничивает объём оперативной памяти
--pids-limit	ограничивает количество процессов
--blkio-weight	задаёт приоритет операций ввода/вывода

Инструменты диагностики Docker

Команда	Назначение
docker stats	мониторинг CPU, памяти, сети, диска
docker inspect	показывает конфигурацию контейнера
docker logs	вывод приложения (логи stdout/stderr)
docker events	события Docker (создание, перезапуск и т.д.)
docker system df	использование дискового пространства
iotop, iostat	нагрузка на диск
perf, strace	системные вызовы
tcpdump	сетевой трафик

Реестры и поставка

Тип	Примеры	Особенности
Публичные	**Docker Hub**	Общедоступные, просто использовать, но ограничены по приватности и скорости
Облачные	**ECR** (AWS), **GCR** (Google), **ACR** (Azure)	Удобная интеграция с облаками и CI/CD
Приватные	**Harbor**, **registry:2** (open-source Docker Registry)	Полный контроль, свои политики и безопасность

Реестры и поставка

- **CI/CD:**
  - Сканирование при push
  - Multi-stage builds
  - Кэширование слоев (BuildKit)

- **Подпись и политика:**
  - Подписи: Content Trust / cosign
  - RBAC и policy-gates
  - Автосканирование, неизменяемые теги

- **Оптимизация:**
  - Pull-through cache / registry mirror
  - Подпись + SBOM = проверяемая поставка