



# Виртуализация и Контейнеризация

Лекторы:

Аспирант МФТИ, Шер Артём Владимирович Аспирант МФТИ, Зингеренко Михаил Владимирович 29 октября 2024

# Предпосылки

Есть много приложений - как запустить их одновременно?

- 1. Одно приложение на один сервер
- 2. Несколько приложений на один сервер

На сервере работают два приложения. Как они могут помешать друг другу?

- 1. Убить процесс
- 2. Занять всю RAM / весь CPU
- 3. Удалить файлы
- 4. Влезть в память
- 5. Зависеть от разных библиотек

Нужно изолировать процессы, их файлы и зависимости!

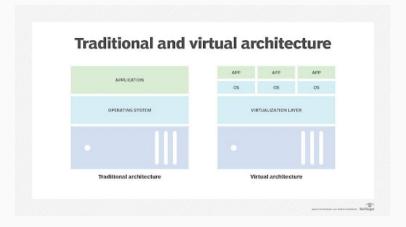
## Предпосылки

Имеем скомпилированное приложение. Что нам нужно сделать, чтобы запустить его на другом компьютере?

- Собрать пакет надеемся, что есть пакетный менеджер
- Указать зависимости не всё легко ставится под все ОС
- Проверить отсутствие конфликтов

Это не всегда легко, хотелось бы иметь упакованное самодостаточное приложение, для которого не нужно индивидуально что-то ставить.

## Виртуализация и Контейнеризация



#### Виртуализация

Виртуализация — это технология, позволяющая создавать виртуальные версии аппаратных компонентов, включая процессоры, оперативную память, жесткие диски и сеть. Она позволяет запускать несколько виртуальных машин (VM) на одном физическом сервере, используя его ресурсы более эффективно.

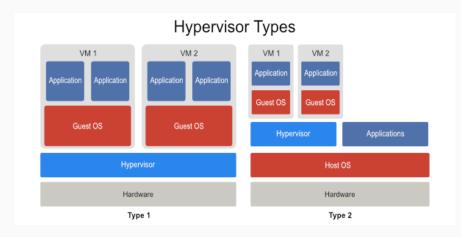
#### Типы виртуальных машин

- Аппаратные виртуальные машины (HVM): Эти виртуальные машины эмулируют полный набор аппаратного обеспечения, позволяя запускать полноценные операционные системы, такие как Windows или Linux.
  Например, такие гипервизоры, как VMware ESXi и Microsoft Hyper-V, создают аппаратные VM.
- Операционные виртуальные машины: Этот тип виртуальных машин предназначен для выполнения приложений, требующих конкретной среды выполнения, например, JVM (Java Virtual Machine) для Java. Они не эмулируют аппаратное обеспечение, а предоставляют платформу для выполнения программ на конкретных языках.

#### Гипервизор

Гипервизор — это программное обеспечение или аппаратный слой, который позволяет создавать, запускать и управлять виртуальными машинами (VM) на физическом сервере. Он создает виртуальную среду, которая эмулирует аппаратное обеспечение, предоставляя каждой виртуальной машине доступ к ресурсам физического сервера (ЦП, память, дисковое пространство и сетевые подключения), как если бы это были её собственные ресурсы.

#### Типы гипервизоров



# Гипервизор типа 1 (Bare-metal, нативный гипервизор)

Определение: Гипервизоры типа 1 работают непосредственно на аппаратном уровне (без хостовой операционной системы) и предоставляют платформу для запуска виртуальных машин напрямую на «чистой» аппаратуре.

Примеры: VMware ESXi, Microsoft Hyper-V (когда используется без хоста), KVM (Kernel-based Virtual Machine).

#### Особенности

- Производительность: Благодаря отсутствию хостовой ОС гипервизоры типа 1 работают почти с нативной производительностью, обеспечивая минимальную задержку и высокую эффективность.
- Безопасность: Высокий уровень изоляции между виртуальными машинами. Прямое управление оборудованием снижает риски, связанные с уязвимостями хостовой ОС.
- Применение: Чаще всего используется в серверных центрах, облачных платформах и крупных корпоративных средах, где требуется высокая масштабируемость и надежность.
- Недостатки: Более сложные в настройке и управлении по сравнению с гипервизорами типа 2; требуют специализированных навыков и инфраструктуры.

# Гипервизор типа 2 (Hosted, хостовый гипервизор)

Определение: Гипервизоры типа 2 устанавливаются поверх хостовой операционной системы, работая как обычное приложение, предоставляя возможность создавать и управлять виртуальными машинами через эту ОС.

Примеры: VMware Workstation, Oracle VirtualBox, Parallels Desktop для Mac.

#### Особенности

- Производительность: Из-за наличия дополнительного уровня в виде хостовой ОС гипервизоры типа 2 могут работать медленнее, чем гипервизоры типа 1. Это приводит к увеличению накладных расходов, особенно при использовании на слабом оборудовании.
- Безопасность: Гипервизоры типа 2 зависят от безопасности хостовой операционной системы, поэтому уязвимости хостовой ОС могут повлиять на безопасность виртуальных машин.
- Применение: Чаще используется для разработки, тестирования, локальных экспериментов и домашних лабораторий, где не требуются строгие требования к производительности и безопасности.
- Простота использования: Обычно проще в настройке и эксплуатации, так как не требует специфических знаний в

# Сравнение

Критерий	Тип 1 (Bare-metal)	Тип 2 (Hoste
Уровень запуска	Аппаратный уровень	Уровень хост-
Примеры	ESXi, Hyper-V, KVM	Workstation, Virt
Скорость	Высокая, близкая к нативной	Ниже, накладные
Безопасность	Высокая изоляция	Зависит от хос
Применение	Серверы, корпорации	Локальная разр
Удобство	Требует опыта	Простая настр

## Виртуализация vs Контейнеризация

Давайте использовать виртуальные машины!

- Создадим по виртуальной машине для каждого приложения.
- Процессы точно изолированы и не мешают друг другу

Все еще используем один физический сервер Проблемы виртуальных машин:

- Тяжелые приложение + ОС
- Используют больше ресурсов каждая ОС работает независимо

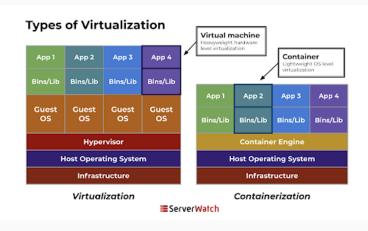
Что можем сделать? Использовать одну ОС (одно ядро)!

## Виртуализация vs Контейнеризация

Контейнеризация - способ запуска приложений, изолированных от ОС и друг друга, но при этом используя ядро и инструменты ОС.

- контейнеры легче
- все еще изолированы
- требуют меньше ресурсов

#### Виртуализация vs Контейнеризация



#### Docker

**Docker** Самая популярная система контейнеризации. Создает легковесные образы (MBs vs GBs) Основано на внутренних абстракциях ОС

Концепция: 1 процесс на 1 контейнер!



## Основные термины

**Docker Container** - запущенный контейнер с вашим приложением. Содержит в себе все необходимые файлы и один запущенный процесс - процесс приложения.

Docker Image - образ контейнера, "шаблон" контейнера.

Хранит в себе информацию о содержимом контейнеров, которые создаются на его основе.

**Dockerfile** - текстовый файл, содержащий в себе описание по созданию образа. Состоит из разных команд, которые готовят необходимое окружение для работы вашего приложения.

- FROM базовый образ
- RUN запустить команду
- COPY / ADD- скопировать файлы в образ
- USER изменить пользователя
- ENTRYPOINT стартовая команда для запуска контейнера

## Пайплайн работы с Docker



#### Механизмы изоляции в Docker

Docker использует два ключевых механизма Linux для изоляции контейнеров:

- Namespaces (пространства имен)
- Control Groups (cgroups) (группы управления)

Цель изоляции: обеспечить контейнерам отдельное окружение и контроль над доступными ресурсами, делая их независимыми и безопасными.

## Namespaces (пространства имен)

Namespaces создают изолированное пространство имен, благодаря которому процессы контейнера видят только ресурсы, предоставленные им, как если бы они работали на отдельной системе.

#### Основные типы namespaces

- PID (Process ID): изолирует процессы, так что процессы в одном контейнере не видны в другом.
- NET (Network): предоставляет каждому контейнеру свой собственный стек сетевых интерфейсов, IP-адресов и маршрутизации.
- IPC (Interprocess Communication): разделяет ресурсы межпроцессного взаимодействия, например, семафоры и очереди сообщений.
- MNT (Mount): изолирует файловую систему, предоставляя каждый контейнеру своё файловое окружение.
- UTS (Unix Time Sharing): позволяет задавать отдельные имена хостов и доменные имена для контейнеров.
- User: изолирует идентификаторы пользователей, позволяя разным контейнерам использовать одного и того же пользователя с разными правами.

# Control Groups (cgroups)

cgroups (группы управления) отвечают за контроль и ограничение ресурсов, доступных контейнерам.

Основные возможности cgroups:

- Ограничение использования ресурсов: контроль использования СРU, памяти, сетевой полосы пропускания и I/O для контейнеров.
- Приоритет ресурсов: возможность распределять ресурсы по приоритету, чтобы важные контейнеры получали больше мощности.
- Отслеживание и статистика: мониторинг потребления ресурсов каждым контейнером.
- Изоляция ресурсов: позволяет ограничивать ресурсы, доступные каждому контейнеру, предотвращая их конкуренцию за ресурсы.

## Namespaces и cgroups в Docker

Namespaces предоставляют контейнерам изолированное окружение, имитирующее отдельную операционную систему. cgroups обеспечивают контроль и ограничение ресурсов, предотвращая конкуренцию и обеспечивая стабильность работы.

Вместе, namespaces и cgroups создают безопасные, независимые контейнерные среды, которые позволяют Docker поддерживать высокую степень изоляции при эффективном использовании ресурсов.

# До следующей лекции!