

Обзор IT-систем: средства автоматизации и основные функции систем



Андрей
Тряпичников



Андрей Тряпичников

Системный администратор

Одноклассники



[Андрей Тряпичников](#)

a.tryapichnikov@gmail.com

Предисловие

Что узнали на прошлом вебинаре?

- из чего состоит компьютер
- какие задачи выполняет
- основы ИТ-систем

Что узнаем на этом вебинаре?

- современные тренды в администрировании
- какие элементы используются при создании ИТ-системы



План занятия

1. [Введение](#)
2. [Development](#)
3. [Operations](#)
4. [DevOps](#)
5. [CI/CD](#)
6. [Инструменты доставки кода](#)
7. [Виртуализация, контейнеризация](#)
8. [Облачные решения](#)
9. [Мониторинг](#)
10. [IaC, Infrastructure as Code](#)
11. [Основные практики построения IT-систем](#)
12. [Итоги](#)
13. [Домашнее задание](#)

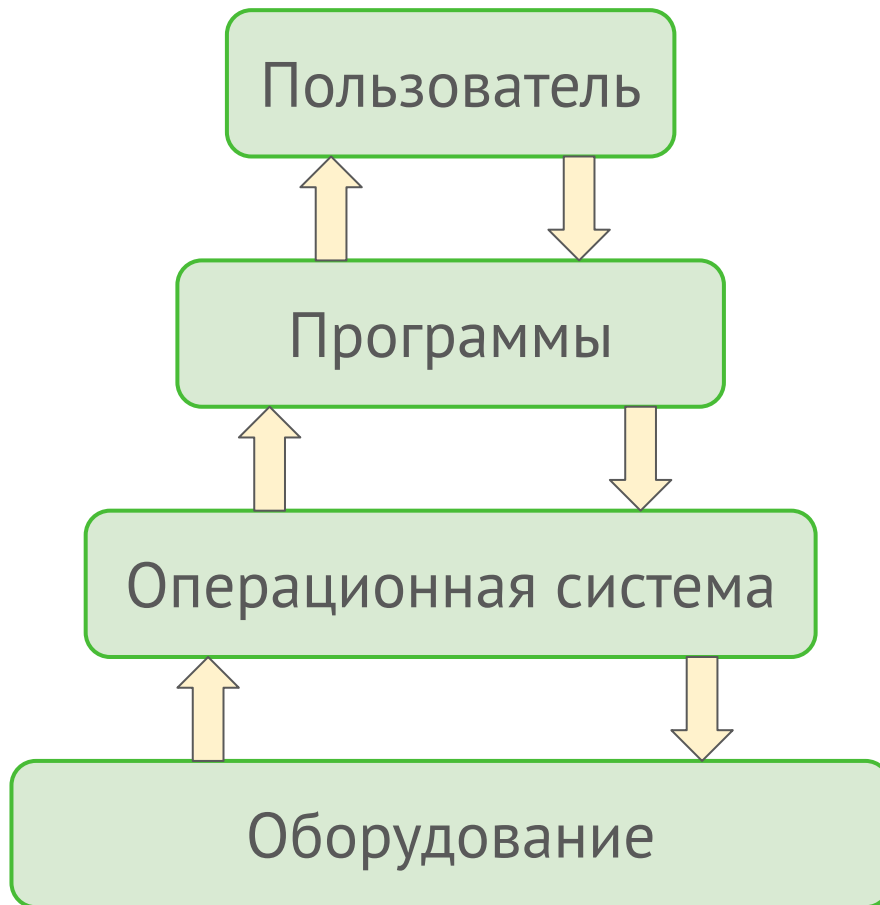


Введение

Современные тренды

- виртуализация
- контейнеризация
- облака
- автоматизация
- инструменты CI/CD
- и многое другое...

Взаимосвязь технологий





Development

Разработка



Работа программиста

Программист выучивает язык программирования, устанавливает у себя ПО для разработки и начинает разрабатывать.

Сложности появляются, когда программистов, работающих над продуктом, становится больше одного (например, вносятся изменения в одни и те же файлы).

Командная разработка

Методологии

- Agile
- Kanban
- SCRUM

ПО для командной работы

- хранение кода, версионность ➡ git
- ПО для тестирования ➡ тестовые серверы
- ПО для сборки версии (релиза) ➡ боевые серверы
- и многое другое...



Operations

Поддержка

Поддержка инфраструктуры

Основная задача системного администратора — работа инфраструктуры.

Инфраструктура:

- железо (серверы, сети)
- операционные системы
- ПО на серверах и рабочих станциях

Если работает инфраструктура и не работает услуга, вы можете услышать от админа: *“проблемы на вашей стороне”*



Цель бизнеса

Основная цель бизнеса — **прибыль**.





DevOps



Определение DevOps

DevOps (development, разработка + operations, поддержка) — методология активного взаимодействия специалистов по разработке со специалистами по информационно-технологическому обслуживанию и взаимная интеграция их рабочих процессов друг в друга для обеспечения качества продукта.

Философия DevOps

Ответственность за результат лежит на всех и каждом. И любой человек отвечает не столько за *свою работу*, сколько несет ответственность за *весь продукт*. Проблема, когда она есть, общая, и каждый в команде помогает ее решить.

Важно именно **решать проблему**, а не просто применять практики. Более того, практики внедряют не где-то, а в весь продукт. Продукту нужен не столько DevOps-инженер, сколько решение проблемы.

Что делает DevOps-инженер в команде

- помогает решить, какую архитектуру будет использовать приложение и как оно будет масштабироваться
- настраивает сервера, автоматизированную проверку и заливку кода, проверку среды
- автоматизирует тестирование
- внедряет непрерывные улучшения
- и многое другое...

Автоматизация процессов

Одна из самых страшных потерь в бизнесе — **потеря времени.**

DevOps стремится к максимальной автоматизации ➡

- сборка новых версий
- тестирование софта на всех уровнях
- сбор отчетов
- управление серверами

— происходят без участия человека и без траты времени инженеров.

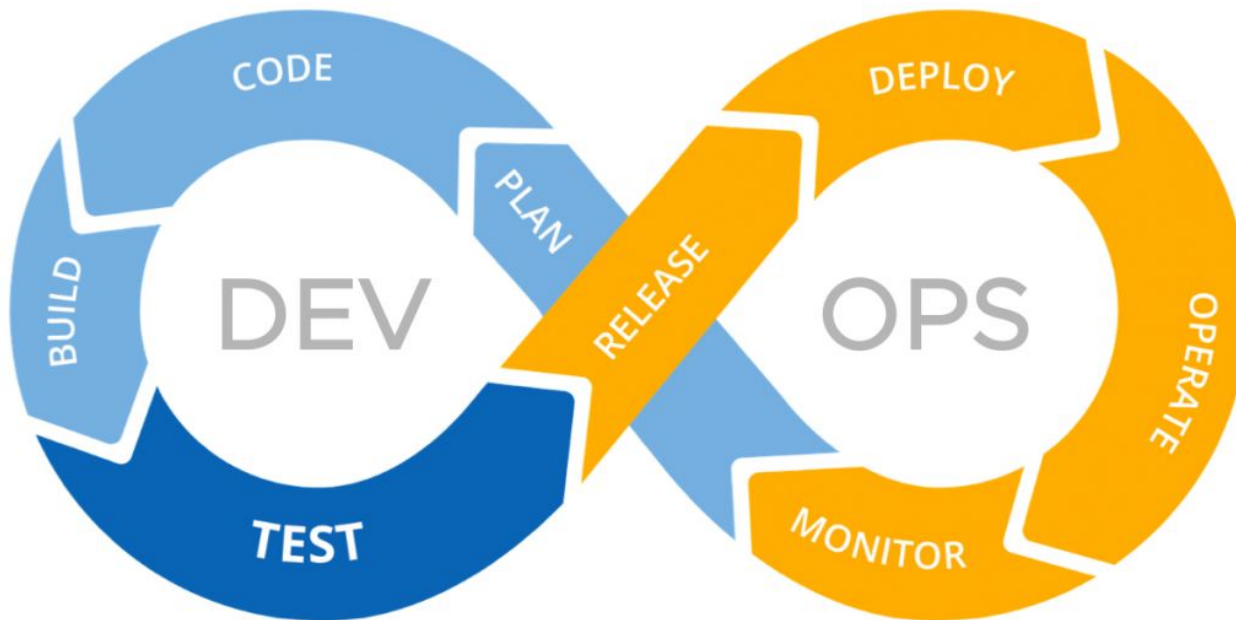


CI/CD

Непрерывная интеграция и поставка

CI/CD

Непрерывная интеграция (Continuous Integration, CI) и непрерывная поставка (Continuous Delivery, CD) представляют собой культуру, набор принципов и практик, которые позволяют разработчикам чаще и надежнее разворачивать изменения программного обеспечения.



Как разрабатывали раньше

Раньше команда много времени тратила на **планирование продукта**: что и как должно происходить.

Когда начиналась реализация, **нельзя** было **свернуть** с намеченного пути. Команда долго создавала новый продукт именно таким, какой он был запланирован.

После презентации продукта, появлялись замечания, **необходимость исправлений** — разработчики готовили следующую версию продукта. Новая версия продукта — раз в **полгода-год**.

Для некоторых решений и сейчас это лучший вариант, но не для всех!



CI, Непрерывная интеграция

Непрерывная интеграция (CI) — методология разработки и набор практик, при которых в код вносятся небольшие изменения с частыми коммитами*.

Большинство приложений разрабатываются разными людьми с использованием разных инструментов.

Появляется необходимость объединения и тестирования этих изменений.

*Коммит - фиксирование состояния всех файлов в определенный момент времени



CD, Непрерывная поставка

Непрерывная поставка (CD) – это практика автоматизации всего процесса релиза (выпуска) ПО. Идея заключается в том, чтобы выполнять CI, плюс автоматически готовить и вести релиз к продакшену.

При этом желательно добиться следующего: любой, кто обладает достаточными привилегиями для развертывания нового релиза, может выполнить развертывание в любой момент, и это можно сделать в несколько кликов.

Цель CD:

Сборка, тестирование и релиз программного обеспечения с большей скоростью и частотой. Подход позволяет уменьшить стоимость, время и риски внесения изменений путём более частых мелких обновлений в продакшн.



Agile + DevOps + CI/CD

Agile – это устранение проблем при взаимодействии заказчика и разработчика

DevOps – устранение проблем между разработчиком и администратором

CI/CD – это реализация тех стратегий и компонентов, которые есть в DevOps, на практике.

Минусы CI/CD

- Искушение перевести на Agile, DevOps и CI/CD сразу всё, включая монолитные системы.
Например, банковские автоматизированные системы.
- Поддержка должного уровня координации между CI и CD.
Всегда существует человеческий фактор - сложно налаживать здоровую командную работу, т.к. запрограммировать и автоматизировать невозможно.



Инструменты доставки кода

Инструменты доставки кода

Jenkins — это инструмент CI, поддерживает весь жизненный цикл разработки программного обеспечения от сборки и тестирования до документирования и развертывания.

TeamCity — аналог, имеет бесплатную версию для маленьких проектов. Поддерживает множество плагинов.

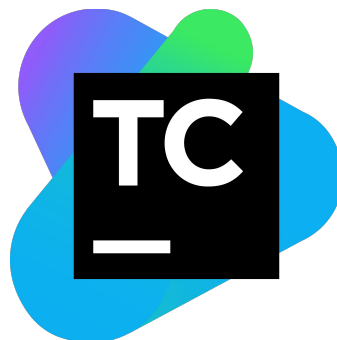
Gitlab — обеспечивает анализ представлений кода, управление ошибками и CI/CD в едином веб-хранилищ

Что используют в разных компаниях?



Jenkins

Facebook,
Netflix,
Instacart,
Lyft



TeamCity

StackOverflow,
Ebay,
Apple,
Intuit




GitLabCi

WebbyLab,
Dial Once,
Redsmin

Пример CI/CD



```
pipeline {
  stages {
    stage("Prepare container") {
      agent { docker { ... } }
    }
    stages {
      stage('Build') { steps { ... } ... }
      stage('Test') { steps { ... } ... }
      stage('Package') { steps { ... } ... }
    }
  }
  stage('Push images') {
    agent any
    when { branch 'master' }
    steps { ... }
  }
  stage('Trigger kubernetes') {
    agent any
    when { branch 'master' }
    steps { ... }
  }
}
```



Виртуализация, контейнеризация

Виды серверов и виртуализации

- **“Железный” сервер**

Bare metal, Hardware

- **Виртуальный сервер внутри ОС**

Hyper-V, VirtualBox

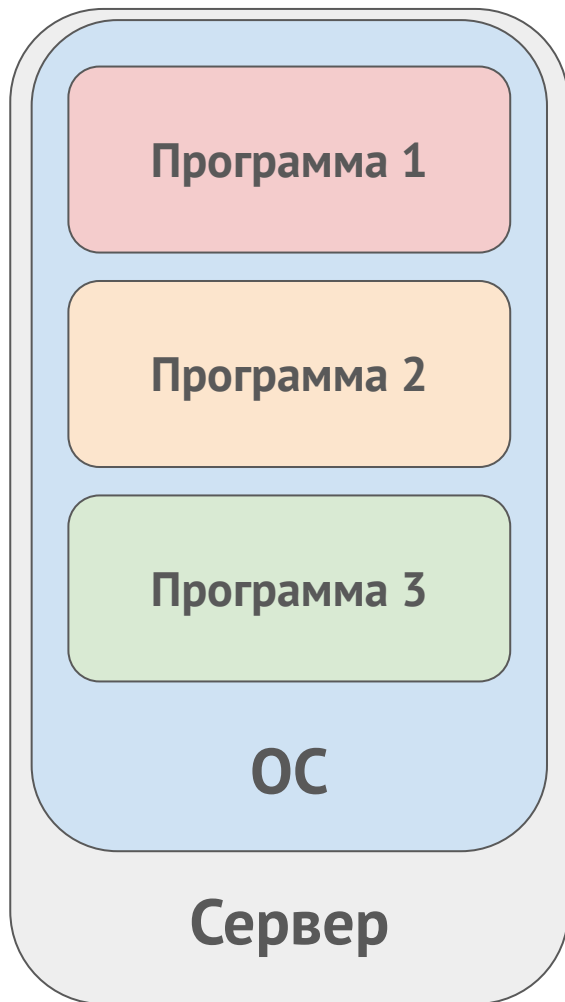
- **Виртуальный сервер внутри специальной хостовой ОС**

VmWare ESXi

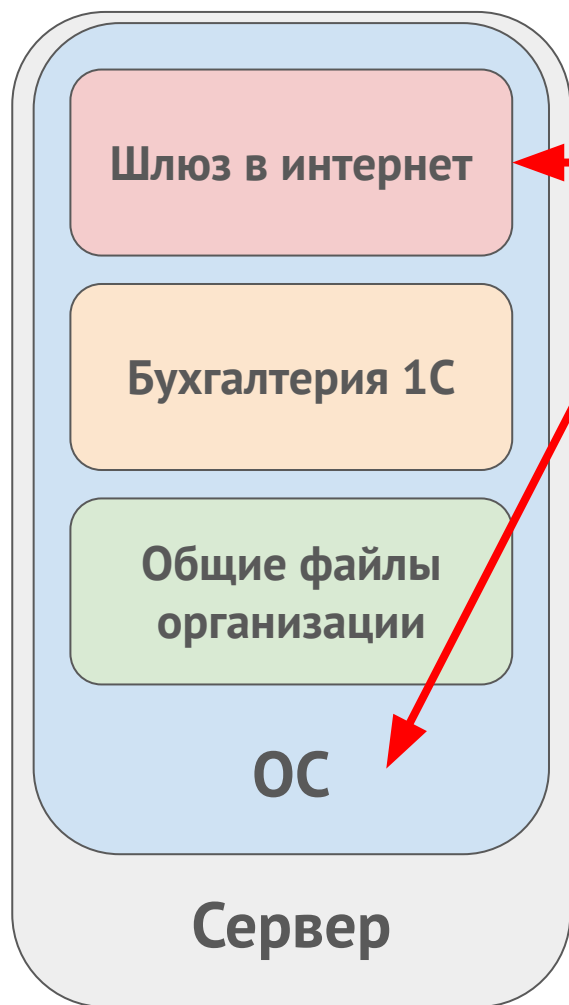
- **Изолированный контейнер внутри ОС**

Docker

“Железный” сервер



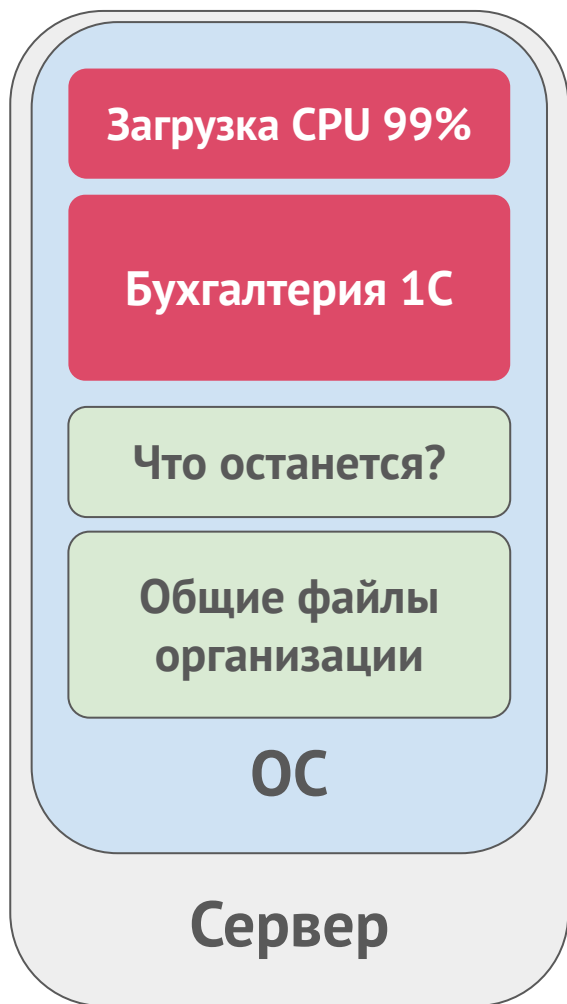
Пример “всё-в-одном”



Если взломают шлюз —
взломают всё остальное.

**Нужно убрать шлюз на
отдельное устройство!**

Пример “всё-в-одном”



Если один из процессов займёт 100% памяти или процессора — остальные не смогут работать.

Покупать сервер под каждый сервис слишком дорого.

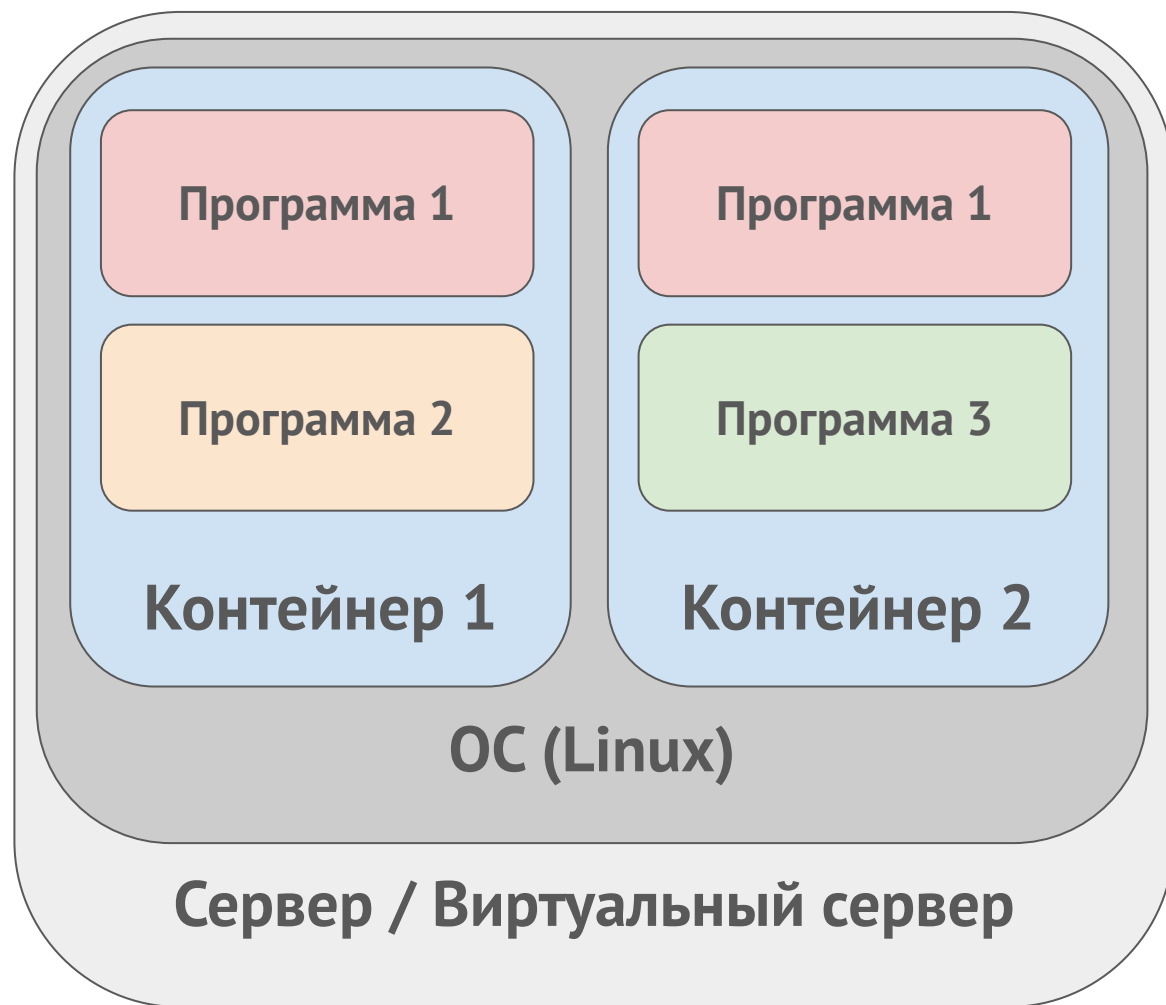
Нужно изолировать процессы друг от друга!

Виртуальный сервер



Администратор может жёстко ограничить максимальные ресурсы для каждого из виртуальных серверов

Контейнеры



Пример использования контейнеров

Распределённое приложение (онлайн-игра). Часы максимальной нагрузки - с 9 утра по 12 дня, и повышенная загрузка с 12 до 13 часов.

	Количество запущенных контейнеров (1 контейнер на 1000 пользователей)								
Время GMT	00:00-09:00	09:00-12:00	12:00-13:00	13:00-16:00	16:00-17:00	17:00-19:00	19:00-20:00	20:00-22:00	22:00-00:00
Лондон									
	1	3	2	1	1	1	1	1	1
Москва									
	1	1	1	3	2	1	1	1	1
Сеул									
	1	1	1	1	1	1	3	2	1

Преимущества контейнеров над виртуализацией

Основные преимущества контейнеров — **абсолютное совпадение кода и окружения** и **легковесность**.

Абсолютное совпадение кода и окружения - запуская контейнеры на разных компьютерах с разными версиями ОС, вы гарантированно получаете одинаковый результат.

Легковесность - между всеми процессами контейнеров разделяется единственное ядро ОС и нет необходимости тратить дополнительные ресурсы на гостевые операционные системы, а главное, запуск таких программ в контейнере происходит намного быстрее, чем эмулированных ОС.



Облачные решения

Облачные вычисления (Облако)

Модель предоставления удобного для пользователя доступа к распределенным вычислительным ресурсам, которые должны быть развернуты и запущены по запросу с минимально возможной задержкой и минимальными затратами со стороны сервис провайдера*

*Определение от NIST, National Institute of Standards and Technology

Облако **≠** виртуализация

Виртуализация — это один из кирпичиков, на котором облако строится.

Облачные решения

- **Software as a Service (SaaS)** — сервис предоставляет провайдер
- **Platform as a Service (PaaS)** — сервис настраиваете вы, но инфраструктуру предоставляет провайдер
- **Infrastructure as a Service (IaaS)** — инфраструктуру настраиваете вы - провайдер отвечает только за работоспособность базовых серверов

Облачные решения



SaaS

SaaS ➔ провайдер предоставляет сервис полностью.

Пример: Яндекс.Почта или Gmail

Особенность: При этом клиент по факту не делает ничего для работоспособности сервиса, только пользуется.



PaaS

PaaS ➔ провайдер предоставляет виртуальный сервер (набор ресурсов RAM / CPU / Диск / Сеть), ОС и необходимое ПО.

Пример: Базы данных, аналитика данных

Особенность: В PaaS система управления базами данных (СУБД) уже установлена, нужно лишь настроить ее для себя и загрузить данные. За работоспособность и резервное копирование отвечает провайдер.



PaaS

Традиционный подход
(Выполняет администратор)



Выбрать и купить HW



Сконфигурировать HW



Установить и
сконфигурировать ОС



Установить DB&FMW ПО



Сконфигурировать ПО и БД,
патчить



Добавить HW и все переконфигурировать при росте потребностей



Портал

Потребовать создать
новую БД



Настроить увеличение
конфигурации по
требованию



BCE !!!!

Пользователь не
знает об
инфраструктуре

Развертывание Platform-as-a-Service
(Выполняет пользователь)



Self-Service Provisioning

Источник - Oracle

IaaS

IaaS → можно представить как набор (пул) ресурсов:
RAM / CPU / Диск / Сеть.

Пример: несколько виртуальных серверов, соединённых в локальную сеть, размещённых в облаке

Особенность: клиент с этими ресурсами делает что хочет в рамках пула.



Средства шаблонизации серверов

Docker, Packer и Vagrant.

Вместо того чтобы **вводить** много **серверов** и **настраивать** их, запуская на каждом один и тот же код, средства шаблонизации создают **образ сервера**, содержащий полностью самодостаточный «снимок» операционной системы (ОС), программного обеспечения, файлов и любых других важных деталей.

Конфигурация Vagrant:

```
Vagrant.configure("2") do |config|
  config.vm.box = "bento/ubuntu-20.04"
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "2048"
    vb.cpus = "2"
  end
end
```



Server



Dockerfile



Virtual Servers



Мониторинг

Сбор, обработка, агрегирование и отображение в реальном времени количественных и качественных показателей системы.

Мониторинг позволяет улучшать, либо оставлять на приемлемом уровне качество обслуживания пользователей.



Взято с сайта: unsplash.com

Требования к мониторингу

Мониторинг должен:

- Отвечать на вопросы “Что случилось?” и “Почему?”.
- Быть достаточно простым.
- Иметь подходящий уровень детализации
- Содержать в себе метрики, связанные с “бизнес” частью



IaC, Infrastructure as Code

Инфраструктура как код

Инфраструктура как код

Для определения, развертывания, обновления и удаления инфраструктуры нужно **писать и выполнять код**.

То есть конфигурационные файлы должны храниться в централизованном хранилище.

Соответственно, в случае необходимости изменить конфигурацию на сервере (например, количество памяти у сервера), ответственный сотрудник меняет нужную переменную в хранилище, и оттуда она уже автоматически применяется на сервере (либо создаётся новый сервер).



Инструмент IaC

Terraform – помогает декларативно (через описание) управлять инфраструктурой.

Достаточно написать конфигурацию, в которой будет изложено, как вы видите вашу будущую инфраструктуру. Такая конфигурация создается в человеко-читаемом текстовом формате.

Средства инициализации ресурсов

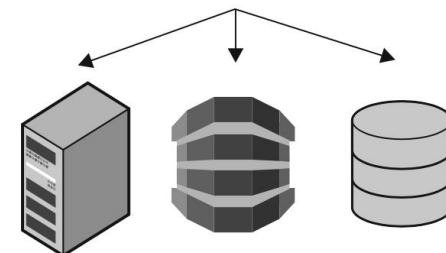
Средства инициализации ресурсов, такие как **Terraform**, **CloudFormation** и **OpenStack Heat**, отвечают за создание самих серверов.

Тот же самый сервер при помощи Terraform:

```
resource "aws_instance" "app" {  
    instance_type = "t2.micro"  
    availability_zone = "us-east-2a"  
    user_data = <<-EOF  
        #!/bin/bash  
        sudo service apache2 start  
    EOF  
}
```


```
resource  
"aws_instance" "a" {  
    ami = "ami-40d28157"  
}  
  
resource  
"aws_db_instance" "db"  
{  
    engine = "mysql"  
    name = "mydb"  
}
```

Конфигурация Terraform



Преимущества инфраструктуры в виде кода

- **Самообслуживание:** тайные знания не сосредоточены только в голове одного админа
- **Скорость и безопасность:** исключается человеческих фактор при развертывании очередного сервера
- **Документация:** описание в конфиге и является документацией — не нужно отдельно ничего писать - всё видно сразу в конфиге
- **Управление версиями:** код хранится в специальной системе управления версиями (VCS), можно посмотреть прошлые версии
- **Повторное использование:** переиспользование готовых модулей
- **Радость:** больше нет рутинных действий, например, установки ОС вручную



Основные практики построения IT-систем

Основные практики построения IT-систем

- **Производительность:** построение высокопроизводительных систем =>
➔ обрабатывается большее количество информации
- **Надёжность и отказоустойчивость:**
 - надёжность оборудования — качественное оборудование
 - надёжность ПО — отсутствие ошибок
 - отказоустойчивость - дублирование и мгновенное переключение➔ минимизация простоев, обеспечение непрерывности бизнеса, как следствие потерь денег бизнеса
- **Масштабирование:** способность системы справляться с увеличением рабочей нагрузки при добавлении ресурсов.
 - Вертикальное масштабирование - увеличение памяти, дисков.
 - Горизонтальное масштабирование - увеличение количества серверов.➔ возможность роста количества ресурсов



Резервное копирование

Сохранение копии данных вне основного места их хранения.

Главное назначение резервного копирования – восстановление данных после их потери.

Существует 2 вида системных администраторов:

- которые ещё не делают резервное копирование
- которые **уже делают** резервное копирование

Всегда делайте резервные копии!



Итоги

Итоги

Сегодня мы узнали:

- Почему появился DevOps
- Основы философии DevOps
- Виды виртуализации
- Основные инструменты системного администратора
- Основные практики построения IT-систем



Домашнее задание

Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).


- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

Материалы для дальнейшего изучения

- [The DevOps Handbook](#) (одна из самых популярных книг про философию DevOps. В книге есть общие принципы методологии, она рассказывает, на что обращать внимание в первую очередь при работе на любом проекте).
- [Почему бизнес хочет DevOps](#) (статья на bookflow.ru)
- [Что такое IaaS, PaaS, SaaS: разница простыми словами](#)
- [Статья про надёжность в сложных системах](#) (рекомендуем, несмотря на некритичные проблемы с SSL сертификатом на сайте автора)
- [Построение отказоустойчивой системы](#)

**Задавайте вопросы и
пишите отзыв о лекции!**

Андрей Тряпичников

 Андрей Тряпичников

a.tryapichnikov@gmail.com