

# Инфраструктура как код



Андрей  
Борю



## Андрей Борю

Principal DevOps Engineer, Snapcart





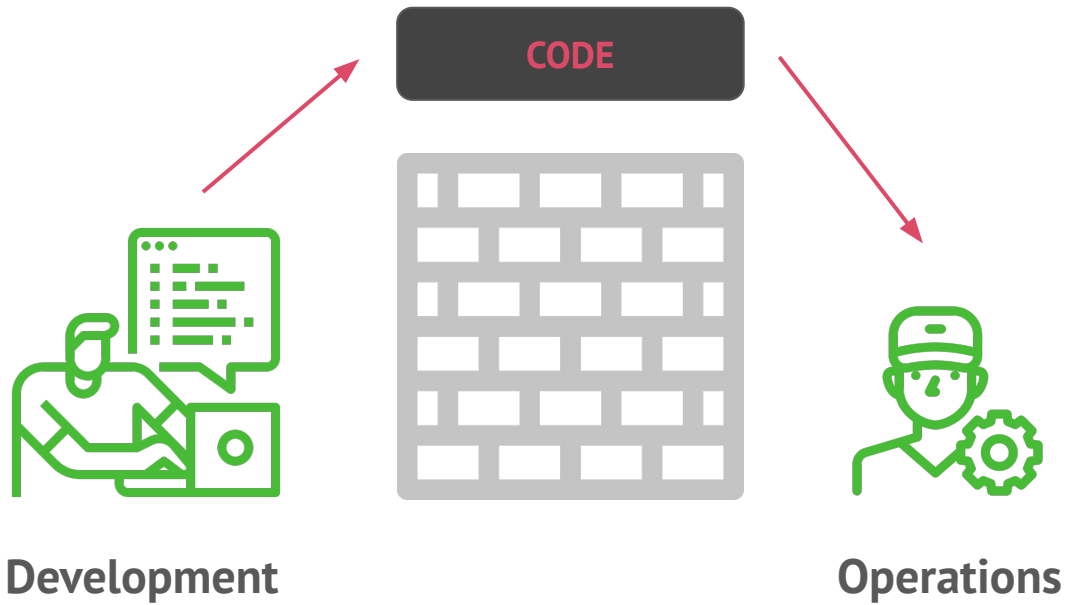
# План занятия

1. [DevOps в контексте IaC](#)
2. [Инфраструктура как код \(IaC\)](#)
3. [Выбор инструментов](#)
4. [Совместное использование инструментов](#)
5. [Резюме](#)
6. [Terraform](#)
7. [Итоги](#)
8. [Домашнее задание](#)



# DevOps в контексте IaC

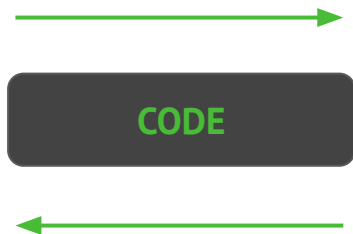
# До эры DevOps



# Появление DevOps



Development



Operations



## DevOps на этом занятии

DevOps - не название команды, должности или какой-то определенной технологии. Это набор процессов, идей и методик. Каждый понимает под DevOps что-то свое, но для этого раздела:

Цель DevOps:

**значительно повысить эффективность доставки ПО.**



# Инфраструктура как код (IaC)





# Инфраструктура как код

Идея, стоящая за IaC (infrastructure as code), заключается в том, что для определения, развертывания, обновления и удаления инфраструктуры нужно писать и выполнять код.

# Специализированные скрипты

Самый простой и понятный способ что-либо автоматизировать — написать для этого специальный скрипт.

```
#!/bin/bash
# Обновляем кэш apt-get sudo apt-get update
apt-get update
# Устанавливаем PHP и Apache
apt-get install -y php apache2
# Копируем код из репозитория
git clone https://github.com/your_account/php-app.git /var/www/html/app
# Запускаем Apache
service apache2 start
```

# Средства управления конфигурацией

**Chef, Puppet, Ansible** и **SaltStack** являются средствами управления конфигурацией. Это означает, что они предназначены для установки и администрирования программного обеспечения на существующих серверах.

Тот же скрипт на **Ansible**:

```
- name: Update the apt-get cache
  apt:
    update_cache: yes

- name: Install PHP
  apt:
    name: php

- name: Install Apache
  apt:
    name: apache2

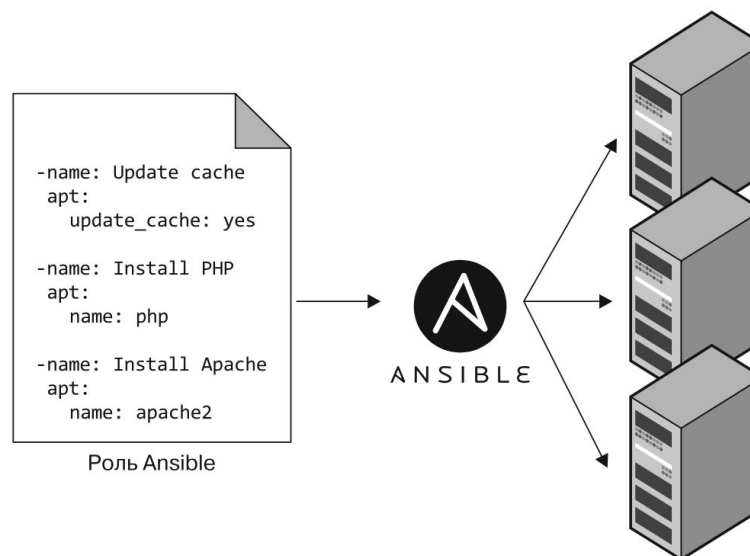
- name: Copy the code from the repository
  git: repo=https://github.com/your_account/php-app.git dest=/var/www/html/app

- name: Start Apache
  service: name=apache2 state=started enabled=yes
```

# Ansible vs bash скрипт

## Преимущества:

- Стандартизированное оформление кода.
- Идемпотентность  
(свойство объекта или операции при повторном применении операции к объекту давать тот же результат, что и при первом).
- Распределенность.



# Средства шаблонизации серверов

Такие как Docker, Packer и Vagrant.

Вместо того чтобы вводить кучу 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.

Конфиг packer:

```
"provisioners": [{  
  "type": "shell",  
  "inline": [  
    "apt-get update",  
    "apt-get install -y php",  
    "apt-get install -y apache2",  
  ]  
}]
```



---

# Средства для работы с образами

1. **Виртуальные машины:** эмулируют весь компьютер включая аппаратное обеспечение.
  - a. Для виртуализации оборудования запускается **гипервизор**.
  - b. Любой образ VM видит только виртуальное оборудование, поэтому он полностью изолирован от физического компьютера и других VM.
  - c. Много накладных расходов на виртуализацию.
  - d. Образы можно описывать при помощи кода, например, используя **Packer** и **Vagrant**.

---

## Средства для работы с образами

2. **Контейнеры:** эмулируют пользовательское пространства ОС.
  - a. Для изоляции процессов, памяти, точек монтирования и сети запускается **среда выполнения контейнеров**, такая как **Docker, CoreOs rkt** или **cri-o**.
  - b. Каждый контейнер может видеть только собственное пользовательское пространство.
  - c. Все контейнеры запущенные на одном сервере одновременно пользуются оборудованием основной ОС.
  - d. Контейнеры запускаются очень быстро.
  - e. Образы можно описывать при помощи кода, используя **Docker** и **CoreOs rkt**.



# Неизменяемая инфраструктура

**Шаблонизация серверов** — это ключевой аспект перехода на неизменяемую инфраструктуру.

Если сервер уже развернут, в него больше не вносятся никакие изменения. Если нужно что-то обновить (например, развернуть новую версию кода), вы создаете новый образ из своего шаблона и развертываете его на новый сервер.



# Средства оркестрации

Нужно выбрать какой-то способ выполнения таких действий:

- Развертывать ВМ и контейнеры с целью эффективного использования оборудования.
- Выкатка обновлений.
- Автовосстановление.
- Автомасштабирование.
- Балансировка нагрузки.
- Обнаружение сервисов.

Выполнение этих задач находится в сфере ответственности средств оркестрации, таких как **Kubernetes**, Marathon/Mesos, **Amazon Elastic Container Service (Amazon ECS)**, Docker Swarm, Nomad и др.

И это все описывается тоже в виде кода!

# Средства инициализации ресурсов

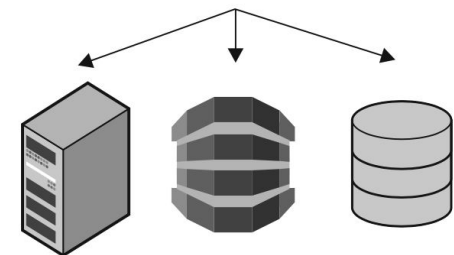
Средства инициализации ресурсов, такие как **Terraform**, **CloudFormation** и **OpenStack Heat**, отвечают за создание самих серверов.

Тот же самый сервер при помощи terraform:

```
resource "aws_instance" "app" {  
  instance_type = "t2.micro"  
  availability_zone = "us-east-2a"  
  ami = "ami-0c55b159cbfafa1f0"  
  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 админа).
- **Скорость и безопасность:** исключается человеческих фактор при развертывании очередного сервера.
- **Документация:** код IaC сам по себе хорошая документация.
- **Управление версиями:** код хранится в vcs.
- **Проверка:** тесты и код ревью.
- **Повторное использование:** переиспользование готовых модулей.
- **Радость:** больше нет рутинных действий.



# Выбор инструментов

---

## Надо выбрать из:

- Управление конфигурацией или инициализация ресурсов.
- Изменяемая или неизменяемая инфраструктура.
- Процедурный или декларативный язык.
- Наличие или отсутствие центрального сервера.
- Наличие или отсутствие агента.

---

## Управление конфигурацией или инициализация ресурсов?

- **Chef, Puppet, Ansible** и **SaltStack** управляют конфигурацией.
- **CloudFormation, Terraform** и **OpenStack Heat** инициализируют ресурсы.

Это не совсем четкое разделение, так как средства управления конфигурацией обычно в какой-то степени поддерживают инициализацию ресурсов, а средства инициализации ресурсов занимаются какого-то рода конфигурацией.

Поэтому следует выбирать тот инструмент, который лучше всего подходит для вашего случая.

---

# Изменяемая и неизменяемая архитектура?

- **Изменяемая**

- проблема дрейфа конфигурации,
- неочевидные расхождения между тестовыми прогонами и продакшеном,
- например обновление библиотеки OpenSSL приведет к отдельному ее обновлению на каждом отдельном сервере.

- **Неизменяемая**

- дает уверенность идентичности всех серверов (тоже есть нюансы),
- обновление OpenSSL – это создание нового образа,
- но даже минимальное изменение ведет к пересборке образов.

# Процедурный или декларативный подход?

- **Процедурный**

- Chef и Ansible
- код пошагово описывает как достичь желаемого результата

- **Декларативный**

- Terraform, CloudFormation, SaltStack, Puppet и Open Stack Heat
- в коде описывается нужное вам конечное состояние, а средства IaC сами разбираются с тем, как его достичь.

```
- ec2:
  count: 10
  image: ami-0c55b159cbfafa1f0
  instance_type: t2.micro
```

```
resource "aws_instance" "example" {
  count = 10
  ami = "ami-0c55b159cbfafa1f0"
  instance_type = "t2.micro"
}
```



---

# Основные проблемы процедурного подхода

- Процедурный код не полностью охватывает состояние инфраструктуры.
- Процедурный код ограничивает повторное использование.

---

## Основные особенности декларативного подхода

- Отсутствие доступа к полноценному языку программирования.
- Процедурный код ограничивает повторное использование.

---

## Наличие или отсутствие центрального сервера

- **Chef, Puppet** и **SaltStack** по умолчанию требуют наличия центрального (master) сервера для хранения состояния вашей инфраструктуры и распространения обновлений.
- У **Ansible, CloudFormation, Heat** и **Terraform** по умолчанию нет центрального сервера.

(но всегда есть нюансы)

---

## Преимущества центрального сервера

- Это единое централизованное место, где вы можете просматривать и администрировать состояние своей инфраструктуры.
- Некоторые центральные серверы умеют работать непрерывно, в фоновом режиме, обеспечивая соблюдение вашей конфигурации.

---

## Особенности центрального сервера

- **Дополнительная инфраструктура.** Вам нужно развернуть дополнительный сервер или даже кластер дополнительных серверов (для высокой доступности и масштабируемости).
- **Обслуживание.** Центральный сервер нуждается в обслуживании, обновлении, резервном копировании, мониторинге и масштабировании.
- **Безопасность.** Вам нужно сделать так, чтобы клиент мог общаться с центральным сервером, а последний — со всеми остальными серверами. Это обычно требует открытия дополнительных портов и настройки дополнительных систем аутентификации, что увеличивает область потенциальных атак.

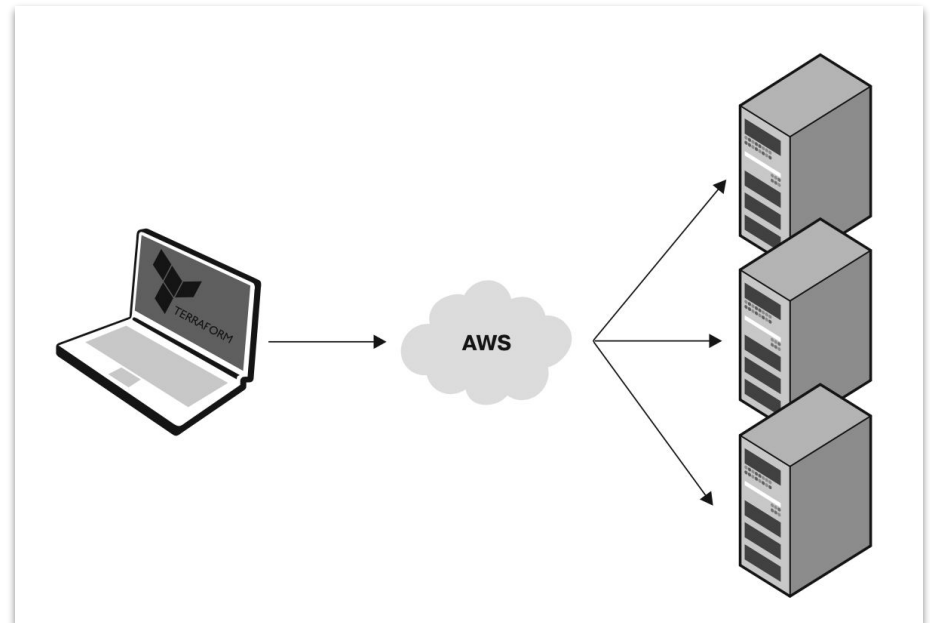
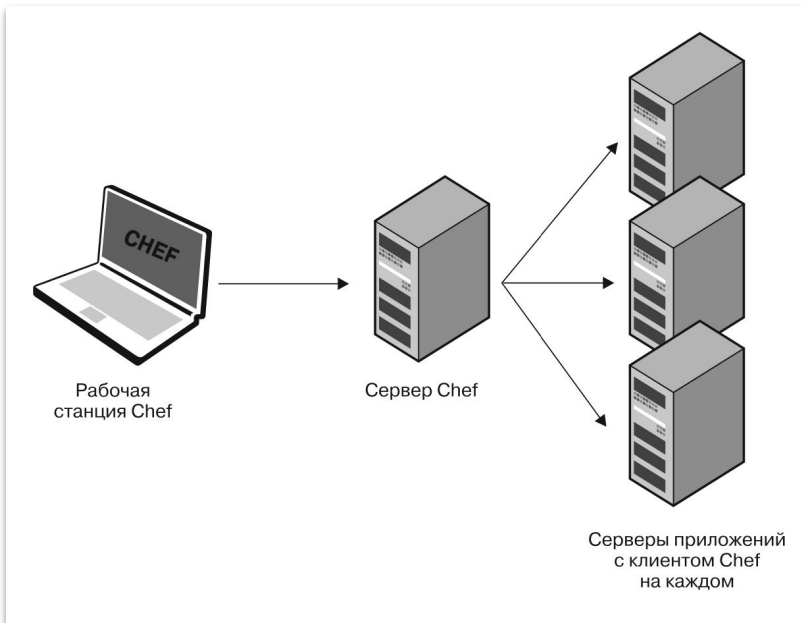
---


## Наличие или отсутствие агентов

- **Chef, Puppet** и **SaltStack** требуют установки своих агентов на каждый сервер, который вы хотите настраивать. Агент обычно работает в фоне и отвечает за установку последних обновлений конфигурации.
- **Ansible, CloudFormation, Heat** и **Terraform** не требуют установки никаких дополнительных агентов.

(но всегда есть нюансы)

# На самом деле не все однозначно





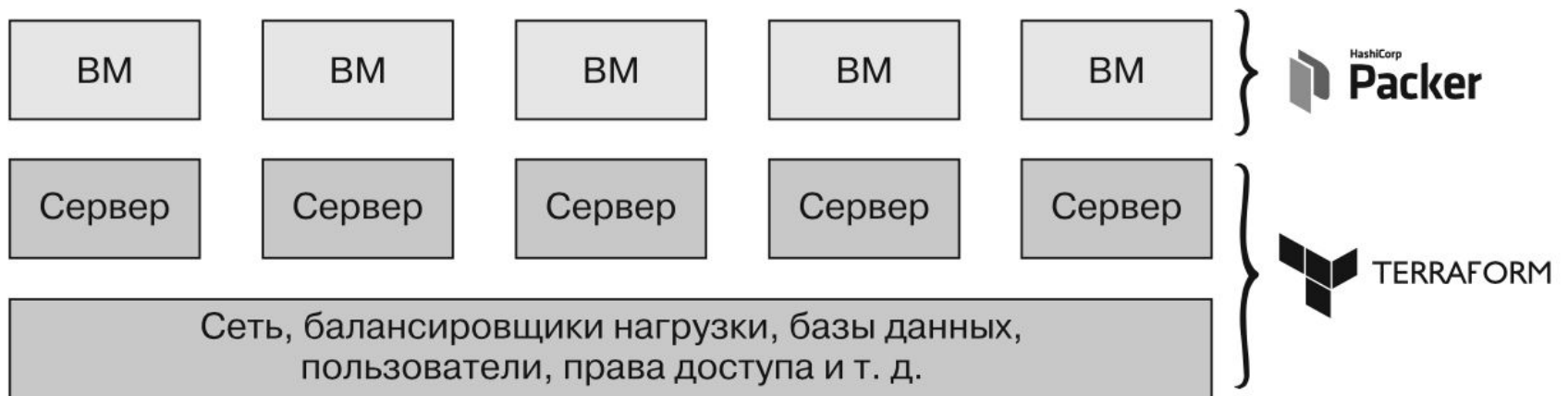
# **Совместное использование инструментов**



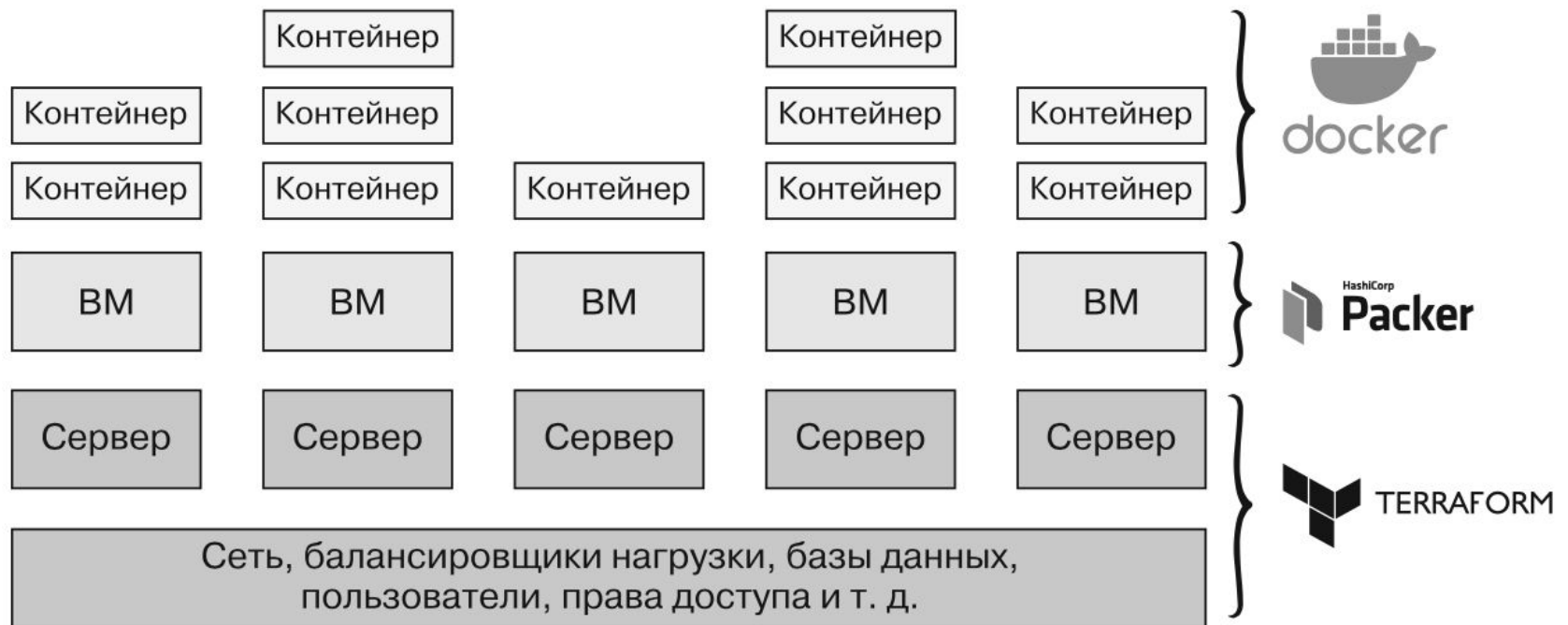
# Инициация ресурсов + управление конфигурацией



# Инициация ресурсов + шаблонизация серверов



# Инициация ресурсов + шаблонизация + оркестрация





# Резюме

## При использовании “стандартных” способов применения

Инструмент	Открытый код	Облака	Тип	Инф-ка	Язык	Агент	Вед. сервер	Сообщество
Chef	+	Все	Упр.конф.	Изм-ая	Процедурный	+	+	Большое
Puppet	+	Все	Упр.конф.	Изм-ая	Декларативный	+	+	Большое
Ansible	+	Все	Упр.конф.	Изм-ая	Процедурный	-	-	Огромное
SaltStack	+	Все	Упр.конф.	Изм-ая	Декларативный	+	+	Большое
Cloud Formation	-	AWS	Иниц. рес.	Неизм-ая	Декларативный	-	-	Маленькое
Heat	+	Все	Иниц. рес.	Неизм-ая	Декларативный	-	-	Маленькое
Terraform	+	Все	Иниц. рес.	Неизм-ая	Декларативный	-	-	Огромное



# Terraform

---

# Terraform

Терраформ это инструмент с открытым исходным кодом от компании HashiCorp, написанный на языке программирования Go.

Терраформ делает от вашего имени API-вызовы к одному или нескольким провайдерам, таким как AWS, Azure, Google Cloud, DigitalOcean, OpenStack и множеству других.

Этот позволяет развернуть инфраструктуру прямо с вашего ноутбука или либо любого другого компьютера, и для всего этого не требуется никакой дополнительной инфраструктуры.



---

# Установка Terraформа

- Скачать с [terraform.io](https://terraform.io)
- Воспользоваться менеджером пакетов (apt, brew, ...)





# Итоги



## Итоги

- Почему инфраструктуру описывают в виде кода.
- Познакомились с:
  - Средствами управления конфигурациями.
  - Средствами для работы с образами.
  - Средствами оркестрации.
  - Средствами инициализации ресурсов.
- Разобрались что такое изменяемая и неизменяемая инфраструктура.



# Домашнее задание



# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

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

**Андрей Борю**



[andreyborue](https://t.me/andreyborue)



[andreyborue](https://netology.ru/andreyborue)



[andreyborue](https://t.me/andreyborue)