

Методы и средства сборки и развертывания ПО, 25

ФИО преподавателя: Смирнов Михаил Вячеславович

e-mail: smirnovmgupi@gmail.com

Лекция 7

Базовые принципы IaC. IaC в непрерывном развертывании.

Что такое Инфраструктура как Код (IaC)?



Infrastructure as Code (IaC) - это практика создания и управления вычислительной инфраструктурой (серверы, сети, базы данных) с помощью специальных конфигурационных файлов.

Сисадмин (настройка конфигов сервера вручную через CLI) -> ДевОПес (конфигурация сервера в текстовом файле: размер, ОС, устанавливаемые пакеты). Инструмент IaC автоматически создает и настраивает сервер в точном соответствии с описанием.

Инфраструктура становится версионизуемым, повторно используемым, тестируемым и совместно используемым активом, таким же как и код приложения (IaC = SCM, предпосылки к интеграции).

Проблемы традиционного подхода: «Снежинки» (Snowflake servers)



Снежинка-сервер настраивается вручную и имеет уникальные, нигде не задокументированные изменения. Воссоздать его в случае падения практически невозможно.

Проблемы ручного управления:

Медлительность: Развертывание нового сервера может занимать дни.

Нестабильность (Configuration Drift): Со временем конфигурации серверов «дрейфуют» из-за ручных «быстрых исправлений», что приводит к расхождениям между средами.

Отсутствие документации: Знания об инфраструктуре хранятся в головах у нескольких инженеров.

Ошибки: Человеческий фактор - главный источник сбоев.

Масштабирование: Сложно и долго тиражировать успешную конфигурацию.

Результат: Классическая фраза «У меня на локальной машине все работало!» становится кошмаром для всей команды.

Преимущества IaC



Скорость и эффективность: Развертывание всей инфраструктуры занимает несколько минут, одним скриптом.

Стабильность и предсказуемость: Идемпотентность гарантирует, что результат каждого запуска кода будет идентичным. Исключается «дрейф» конфигураций, так как код является единственным источником.

Масштабируемость: Идентичная инфраструктура может быть развернута где угодно в каком угодно количестве.

Документирование и прозрачность: Код сам по себе является актуальной и исполняемой документацией. Любой член команды может посмотреть в репозиторий и понять, что развернуто.

Контроль версий и возможность отката: Используя системы вроде Git, можно отслеживать каждое изменение инфраструктуры, видеть, кто и что изменил, и при необходимости мгновенно откатиться к предыдущей, рабочей версии.

Классная штука IaC со странным названием

Технический пример идемпотентности: В условном скрипте IaC указано: «создай виртуальную машину с именем web-server-01». При первом запуске средство IaC создаст ее. При втором и последующих запусках оно уже не будет создавать новую машину с тем же именем, а проверит: «Ага, web-server-01 уже существует, состояние этой машины соответствует описанному в коде. Значит, менять ничего не нужно».

Почему это важно? Это основа предсказуемости и надежности. Можно без страха запускать свои скрипты многократно, например, в CI/CD-конвейере, будучи уверенным, что они не создадут хаос и дублирующие ресурсы.



Декларативный подход к IaC

Декларативный подход (Что я хочу).

В скрипте описывается **желаемое конечное состояние** вашей инфраструктуры («Молодой человек, будьте любезны, 1 сервер типа t3.large с ОС Ubuntu 20.04, с открытым 80-м портом и установленным nginx»).

При этом последовательность действий для выполнения заказа (Как?) определяется инструментом IaC автоматически.

Преимущества: Более простой для чтения и поддержки код, меньше шансов ошибиться, инструмент сам заботится о порядке операций.

Инструменты: Terraform, AWS CloudFormation, Ansible.



Императивный подход к IaC

Императивный подход (Как сделать).

В скрипте описывается **точная последовательность команд**, которые необходимо выполнить для получения нужного инстанса («1. Вызови API RunInstances.

2. Вызови API AuthorizeSecurityGroupIngress для открытия порта.

3. Подключись по SSH и выполни apt-get update && apt-get install -y nginx»).

Преимущества: Большой контроль на низком уровне.

Инструменты: AWS CLI, Shell-скрипты, Puppet.



Инструменты IaC. Terraform



Terraform

Terraform (HashiCorp). Самый популярный инструмент IaC со строго декларативным подходом.

Используется HCL (HashiCorp Configuration Language). Считается, что он проще, чем JSON или YAML, и удобен для использования в качестве документации.

План: Перед внесением любых изменений Terraform выполняет команду `terraform plan` для детального отчета: что будет создано, изменено или уничтожено.

Интерфейсы для работы с API различных IaaS называются **провайдеры**. Существуют провайдеры для AWS, Azure, Google Cloud, Kubernetes, GitHub, Datadog и других систем.

Состояние: Terraform хранит состояние развернутой инфраструктуры в файле. Этот файл используется для сопоставления развернутых инстансов с оригинальной конфигурацией и является критически важным артефактом.

Базовая структура проекта Terraform

```
my-infrastructure/  
├─ main.tf          # Основное определение ресурсов  
  (виртуальные машины, сети)  
├─ variables.tf     # Объявление входных переменных  
  (например, размер инстанса, регион)  
├─ outputs.tf       # Объявление выходных значений  
  (например, публичный IP-адрес созданного сервера)  
├─ terraform.tfvars # Файл с конкретными значениями  
  для переменных (может быть .gitignore'd для секретов)  
├─ providers.tf     # Конфигурация провайдеров  
  (версия, регион по умолчанию)
```

Фрагмент скрипта Terraform

```
# Генерация случайного числа
resource "random_integer" "random_num" {
  min = 1
  max = 100
}

# Создание локального файла
resource "local_file" "hello_world" {
  filename = "hello.txt"
  content = "Hello, Terraform! Random
number:
${random_integer.random_num.result}"

  # Зависимость от random_integer
  depends_on =
[random_integer.random_num]
}
```

```
# Вывод результата
output "random_number" {
  value =
random_integer.random_num.result
  description = "Сгенерированное
случайное число"
}

output "file_path" {
  value =
local_file.hello_world.filename
  description = "Путь к созданному
файлу"
}
```

Инструменты IaC. Ansible



A N S I B L E

Ansible (Red Hat). Инструмент для **конфигурационного менеджмента** и развертывания приложений. В конвейерах CI/CD дополняет Terraform.

Допускает использование императивных команд.

Работает без агентов, напрямую через SSH (для Linux) или WinRM (для Windows).

Сценарии использования:

Настройка уже существующих серверов, созданных Terraform (установка ПО, настройка сервисов, управление файлами конфигурации).

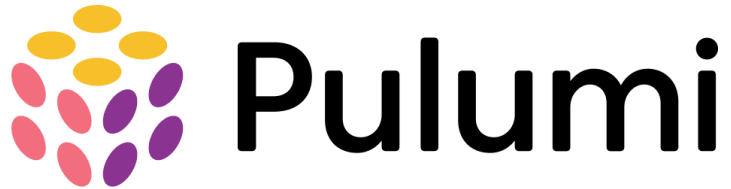
Оркестрация: Координация сложных процессов развертывания, затрагивающих несколько серверов («остановить сервер приложения, сделать бэкап БД, обновить код, перезапустить сервер»).

Язык: Конфигурации пишутся в YAML-файлах, «плейбуках»

Фрагмент плейбука Ansible

```
- name: Установка Nginx на веб-серверах
hosts: webservers
become: yes
tasks:
  - name: Установить nginx
    apt: name: nginx state: present
  - name: Убедиться, что nginx запущен
    service: name: nginx state: started enabled: yes
```

Инструменты IaC. Pulumi



Pulumi. «Инфраструктура как код на настоящих языках программирования».

Вместо специализированных языков (HCL, YAML) скрипт создания или изменения инфраструктуры пишется на Python (+), Go, C# и других языках программирования. Это стирает грань между разработчиками приложений и DevOps.

Преимущество: В скрипте можно использовать циклы, условия, функции, классы, библиотеки.

Жизненный цикл IaC

Написание кода (Write). Создание или изменение файлов конфигурации (например, .tf).

Проверка кода (Review & Lint). Код проверяется через Pull/Merge Request в Git. Коллеги проводят ревью, а автоматические инструменты (например, tflint) проверяют синтаксис и стиль.

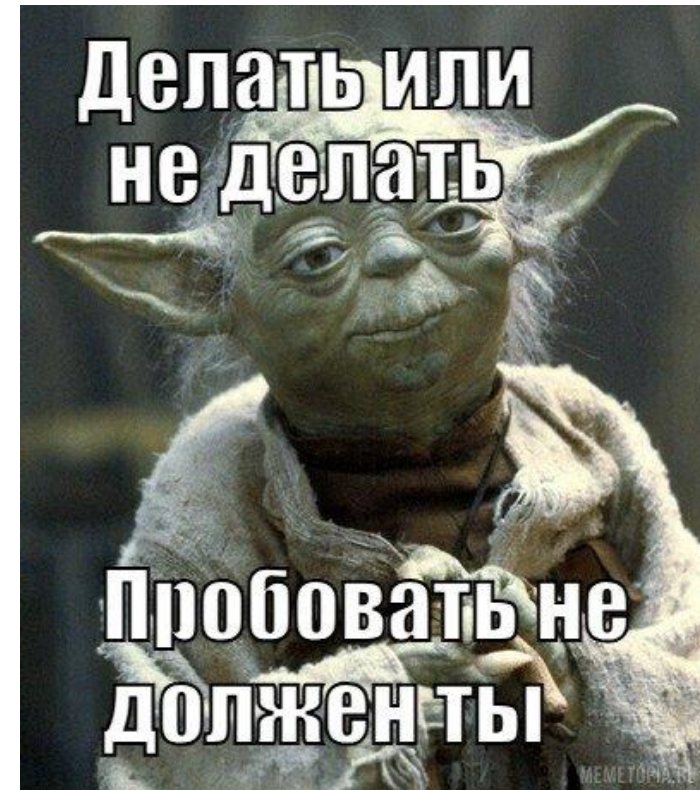
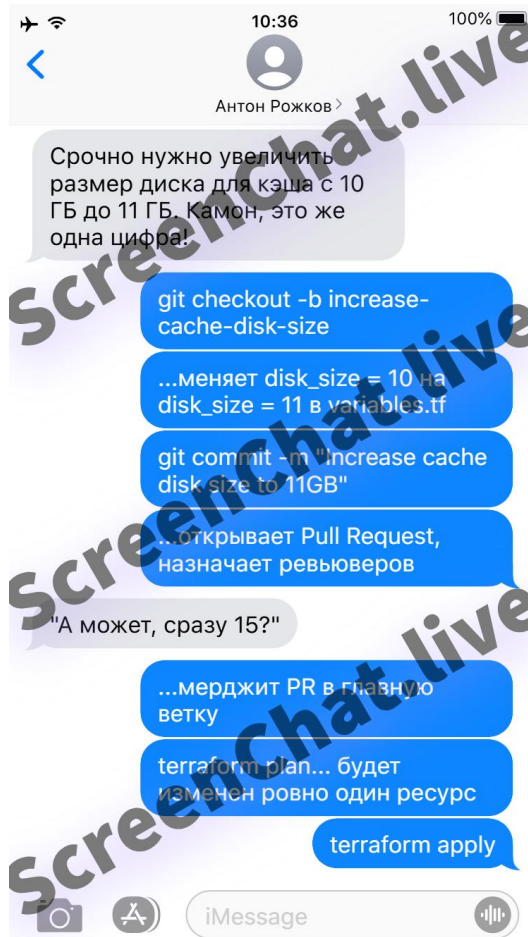
Планирование (Plan). Выполняется команда terraform plan. Инструмент сравнивает код с текущим состоянием (state) и генерирует детальный план предстоящих изменений.

Применение (Apply). После утверждения плана выполняется terraform apply. Инструмент выполняет необходимые вызовы API к облачным провайдерам для приведения инфраструктуры в соответствие с кодом.

Управление состоянием (State Management). После применения состояние обновляется.



Процесс внесения изменений через IaC (☺)



Интеграция IaC в непрерывное развертывание

Код инфраструктуры хранится в том же репозитории (или в соседнем), что и код приложения. Конвейер CD расширяется этапами для работы с этим кодом.

Изменение в коде приложения *или* инфраструктуры инициирует запуск конвейера.

Этап «Plan» для IaC, выполняется terraform plan, и результат (что будет изменено в инфраструктуре) прикрепляется к сборке для проверки.

Этап «Apply» для IaC, После успешного прохода всех тестов приложения и мануального подтверждения плана, выполняется terraform apply.

В итоге, изменения в инфраструктуре (например, обновление версии базы данных) так же легко и контролируемо попадают в продакшен, как и новый функционал приложения.



Безопасность в IaC (DevSecOps)



«Сдвиг влево» (Shift Left). Вопросы безопасности решаются на самых ранних этапах разработки, а не в конце или после инцидента.

(SAST) для IaC. tfsec/checkov/terrascan: Сканирование скрипта на наличие известных уязвимостей и нарушений безопасности *до того*, как инфраструктура будет развернута («Заблокирован ли S3-бакет от публичного доступа?», «Использует ли база данных шифрование?», «Что там с портами в security group?»)

Политики как код (Policy as Code). Использование инструментов IaC для реализации политик безопасности компании. («Разрешать создание инстансов только определенных типов», «Требовать наличие тега owner для всех ресурсов»)

Управление секретами: Секреты (пароли, ключи API) никогда не должны храниться в коде IaC, или храниться в правильно организованном месте.

Спасибо за внимание!