нетология

Автоматизация администрирования инфраструктуры: **Terraform**

Александр
Зубарев

Александр Зубарев

Председатель цикловой комиссии “Информационной безопасности инфокоммуникационных систем”

АКТ (ф) СПбГУТ

Александр Зубарев

Вспоминаем прошлые занятия

Вопрос: назовите основные преимущества IaC?

Вспоминаем прошлые занятия

Вопрос: назовите основные преимущества IaC?

Ответ:

- скорость,
- воспроизводимость,
- масштабируемость,
- не нужна ручная настройка.

Вспоминаем прошлые занятия

Вопрос: что такое YAML?

Вспоминаем прошлые занятия

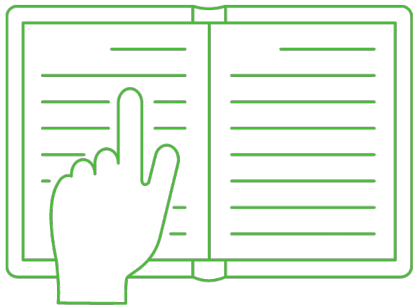
Вопрос: что такое YAML?

Ответ: язык для хранения информации в формате понятном человеку.

Предисловие

На этом занятии мы узнаем:

- где применяется и как работает IaC,
- преимущества и недостатки Terraform,
- синтаксис команд.



План занятия

1. Инфраструктура как код (IaC)
2. Проблемы и недостатки IaC
3. Преимущества и недостатки Terraform
4. Установка Terraform
5. Синтаксис Terraform
6. Итоги
7. Домашнее задание

Инфраструктура как код (IaC)

Инфраструктура как код (IaC)

IaC — модель, по которой процесс настройки инфраструктуры аналогичен процессу программирования ПО.

Приложения могут содержать скрипты, которые создают свои собственные виртуальные машины и управляют ими.

Это основа облачных вычислений и неотъемлемая **часть DevOps.**

Проблемы и недостатки IaC

Недостатки IaC

- Разработка IaC может потребовать использования дополнительных утилит, а любые ошибки при таком проектировании могут быть быстро распространены по всем окружениям проекта, поэтому IaC должен быть всесторонне протестирован.
- Конфигурация окружения была изменена администратором без внесения соответствующих изменений в IaC, поэтому особенно важно полностью интегрировать IAC в процесс системного администрирования, во все IT и DevOps-процессы и вести документацию.

Проблемы IaC

1. В большинстве случаев IaC – это какой-то dsl. A DSL, в свою очередь, – это описание структуры. В них может не быть таких привычных нам вещей, как:
   - переменные,
   - условия,
   - комментарии (например, JSON),
   - функции,
   - ООП как высокоуровневые конструкции.

   В Terraform используется HCL.

HCL

— это язык программирования, разработанный HashiCorp. Он в основном используется DevOps. Представляет собой методологию разработки, предназначенную для ускорения процесса кодирования. HCL используется для настройки программных сред и программных библиотек.

HCL совместим с JSON благодаря API HCL. Его дизайн и синтаксис более читабельны.

Синтаксис и описание HCL

Проблемы IaC

2. Гетерогенная среда.

Обычно вы работаете с одним языком, одним стеком, одной экосистемой. А тут огромное разнообразие технологий.

Вполне реальная ситуация, когда баш с питоном запускает какой-то процесс, в который подсовывается Json. Вы его анализируете, потом еще какой-то генератор выдает ещё 30 файлов. Итог прост: нужны эникейщики, а это бывает очень накладно.

Проблемы IaC

3. Тулинг.

Обычно, есть какая-то крутая штука (tool), позволяющая подсветить ошибку, или то, что можно использовать. Но, как часто бывает, в OpenSource это так не работает. К примеру, проект может состоять из 20-30 файлов, которые используют те или иные объекты друг из друга, но проблема в деталях (где-то забыли про структуру).

Тулзы создают, но не всегда успевают за версиями, и тд тп.

Хорошо, когда есть что-то типа X-Code, VSCode.

Основные игроки

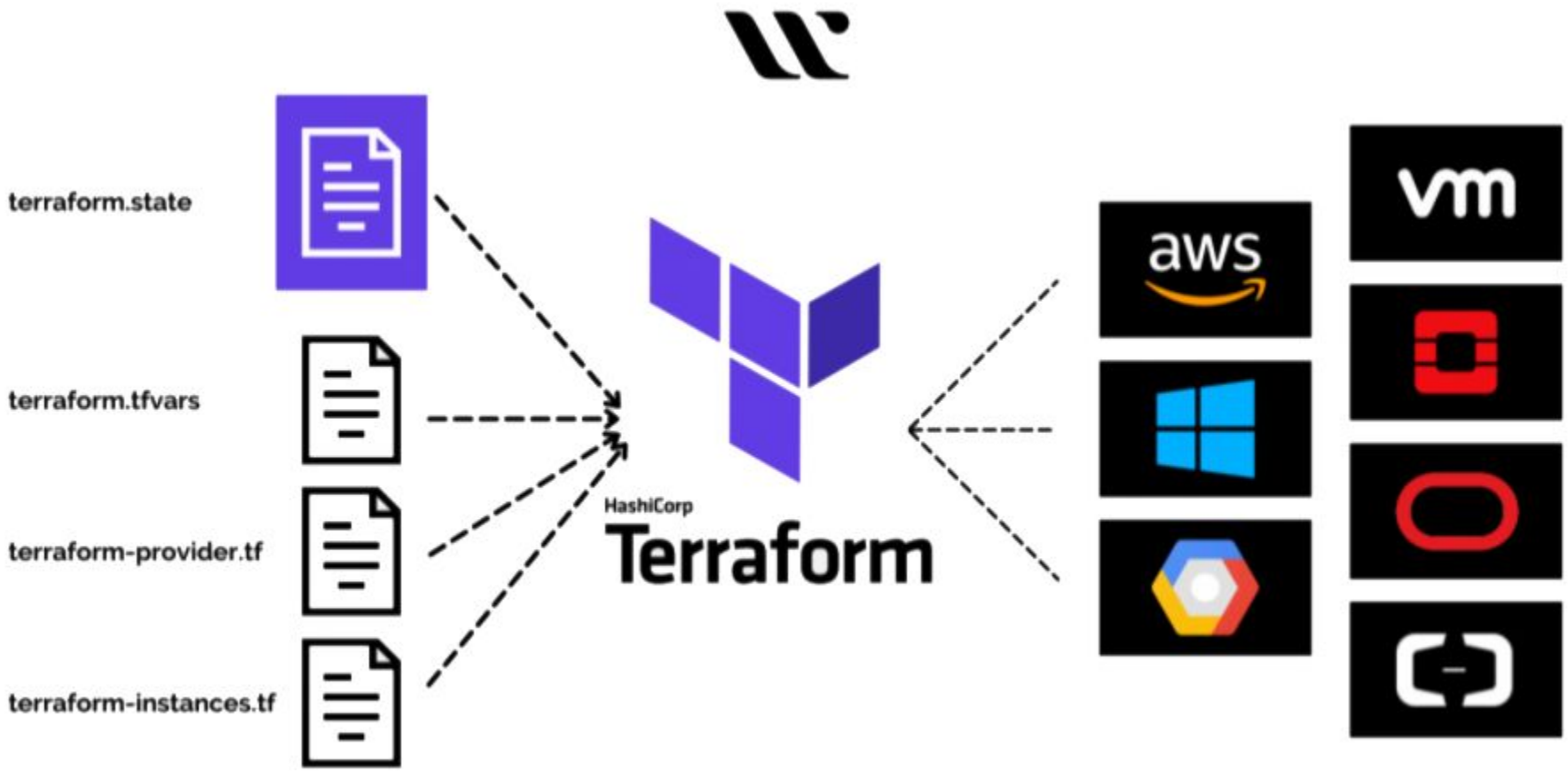
- Vagrant
- Ansible
- Packer
- Terraform
- SaltStack
- Chef

Terraform

Terraform

Terraform — это программный инструмент с открытым исходным кодом, созданный HashiCorp, который помогает управлять инфраструктурой. Пользователи определяют и предоставляют инфраструктуру центра обработки данных с помощью декларативного языка конфигурации, известного как язык конфигурации HashiCorp (HCL) или, необязательно, JSON.

Terraform


terraform.state
terraform.tfvars
terraform-provider.tf
terraform-instances.tf
HashiCorp
Terraform
aws
vm

Преимущества и недостатки Terraform

Terraform: особенности

- оркестрирование, а не просто конфигурация инфраструктуры;
- построение неизменяемой инфраструктуры;
- декларативный, а не процедурный код;
- архитектура, работающая на стороне клиента

Оркестрирование

— автоматизированный процесс управления связанными сущностями, такими как группы виртуальных машин или контейнеров.

Terraform сконцентрирован на задачах по созданию серверов с нуля, оставляя работу по размещению контейнеров с ПО платформам типа Docker или Packer.

Когда вся инфраструктура вашей облачной экосистемы работает как код, и все параметры записаны в декларативные файлы конфигурации, специалисты вашей команды могут работать с ними и изменять эти файлы, как и любой другой код.

Неизменяемая VS одноразовая

Основное отличие состоит в том, что **неизменяемая** инфраструктура привязана к железу, программе, компонентам в ней, а если нам что-то надо изменить, то мы каждый раз создаем новую инфраструктуру.

Одноразовость заключается в том, что если она стала не нужна, то ее просто удалить. Жизненный цикл закончился, за нее не держимся, а просто убираем.

Неизменяемая инфраструктура

Terraform работает по концепции неизменяемой инфраструктуры, где каждое изменение какого-либо компонента (обновление ПО, удаление или добавление компонентов, и т.д.) приводит к созданию отдельного состояния инфраструктуры, то есть к построению новой системы и удалению предыдущей конфигурации.

Это означает, что процесс обновления ПО идет легко и быстро по всей системе сразу и защищен от возможных ошибок.

Декларативный код

При помощи Terraform, вы просто указываете утилите, какие изменения нужно произвести с ТЕКУЩИМ состоянием системы, что позволяет обходиться довольно компактной и очень простой библиотекой шаблонов кода.

При использовании Chef или Ansible вы вынуждены писать пошаговые процедурные инструкции по достижению требуемого состояния системы. Напротив, Terraform, Salt или Puppet предпочитают отписывать конечные состояния системы, оставляя конфигурацию на усмотрение утилиты.

Архитектура, работающая на стороне клиента

Terraform использует API, предоставляемые поставщиком облачного хостинга. С их помощью утилита строит инфраструктуру, что означает: уход от излишних проверок безопасности, отсутствие необходимости в работе отдельного сервера для управления конфигурациями и выделение ресурсов на работу многочисленных программ-агентов.

Недостатки Terraform

- Так как Terraform появился относительно недавно, он еще далеко не идеален.
- Как дирижерскую палочку может использовать только один дирижер, так и Terraform желательно использовать одному оператору или хотя бы с одного терминала.
- Утилита была разработана только под облако и непригодна для использования с кластерами выделенных серверов.

Преимущества Terraform

Два основных преимущества Terraform:

- Супер-портативность — **одна утилита и один язык** применяется для описания облачной инфраструктуры в Google Cloud, AWS, OpenStack и работы с ЛЮБЫМ другим поставщиком услуг. Смена поставщика облачного хостинга больше никогда не будет проблемой.
- Простота полноценного запуска приложений. Предположим, на ваших серверах Amazon запущены Docker контейнеры под управлением Kubernetes, в которых работает широкий спектр приложений, и все это с легкостью **управляется с помощью одного инструмента**.

Установка Terraform

Terraform — install

1) Установка через repository git

```
cd /usr/local/src && git clone https://github.com/hashicorp/terraform.git
```

2) Вручную

```
cd /usr/local/src && curl -O
https://releases.hashicorp.com/terraform/0.15.4/terraform_0.15.4_linux_arm.zip
```

3) Установка через repository apt

```
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -

sudo apt-add-repository "deb [arch=$(dpkg --print-architecture)]
https://apt.releases.hashicorp.com $(lsb_release -cs) main"

sudo apt install terraform
```

Terraform настройка

Terraform использует конфигурационные файлы с расширением .tf.

Заносим минимальный набор переменных для успешного подключения:

```
provider "Имя провайдера" {
  user           = "ваш_логин"
  password       = "ваш_пароль"
  org            = "название_организации"
  url            = "название сайта с api "
}
```

Terraform настройка

Сохраняем конфигурацию, и пробуем подключиться:

```
terraform init
```

Запускать надо там, где находится файл xxx.tf

Пример кода доступен в репозитории GitHub.

Пример кода Terraform для развертывания кластера автомасштабирования

Синтаксис Terraform

Синтаксис Terraform

- Однострочные **комментарии** начинаются с #
- Многострочные комментарии заключаются в символы /* и */
- **Значения** присваиваются с помощью синтаксиса key = value (пробелы не имеют значения). Значение может быть любым примитивом (строка, число, логическое значение), списком или картой.
- Строки в двойных кавычках.
- Строки могут интерполировать другие значения, используя синтаксис, заключенный в ${} , например ${var.foo} .

Синтаксис Terraform

- Многострочные конструкции могут использовать синтаксис «здесь, документ» в стиле оболочки, при этом строка начинается с маркера типа <<EOF , а затем строка заканчивается на EOF в отдельной строке. Строки и конечный маркер не должны иметь отступа.
- Предполагается, что числа имеют основание 10. Если вы поставите перед числом префикс 0x , оно будет рассматриваться как шестнадцатеричное число.

Синтаксис Terraform

- **Логические значения**: true , false .
- **Списки примитивных типов** можно составлять с помощью квадратных скобок ([]). Пример: ["foo", "bar", "baz"] .
- **Карты** могут быть сделаны с фигурными скобками ({}) и двоеточие (:): { "foo": "bar", "bar": "baz" } . Кавычки могут быть опущены на ключах, если ключ не начинается с числа, и в этом случае кавычки требуются. Для однострочных карт необходимы запятые между парами ключ/значение. В многострочных картах достаточно новой строки между парами ключ/значение.

Синтаксис Terraform

Переменные User string

var. префикс, за которым следует имя переменной. Например, ${var.foo} интерполирует значение переменной foo .

Переменные карты пользователя

Синтаксис: var.MAP["KEY"] . Например, ${var.amis["us-east-1"]} получит значение ключа us-east-1 в переменной карты amis .

Переменные списка пользователей

Синтаксис: "${var.LIST}" . Например, "${var.subnets}" получит значение списка subnets в виде списка. А также может возвращать элементы списка по индексу: ${var.subnets[idx]} .

Синтаксис Terraform

Условные операторы троичная операция:

```
CONDITION ? TRUEVAL : FALSEVAL
```

Пример:

```
resource "aws_instance" "web" {

  subnet = "${var.env == "production" ? var.prod_subnet : var.dev_subnet}"

}
```

Поддерживаемые операторы:

Равенство: == и !=

Численное сравнение: > , < , >= , <=

Логическая логика: && , || , одинарный !

Синтаксис Terraform

Математика :

Поддерживаемые операции:

Сложение (+), вычитание (-), умножение (*) и деление (/) **для типов с плавающей запятой**.

Сложение (+), Вычитание (-), Умножение (*), Разделение (/) и По модулю (%) **для целочисленных типов**.

Приоритеты операторов — это стандартный математический порядок операций: умножение (*), деление (/) и модуль (%) имеют приоритет над сложением (+) и вычитанием (-). Круглые скобки могут использоваться для принудительного упорядочивания.

Синтаксис Terraform

Пример использования математических операций:

```
"${2 * 4 + 3 * 3}" # computes to 17

"${3 * 3 + 2 * 4}" # computes to 17

"${2 * (4 + 3) * 3}" # computes to 42
```

Синтаксис Terraform

Инструменты:

Шаблоны

Встроенные функции

Пример синтаксиса Terraform

```
# An AMI
variable "ami" {
  description = "the AMI to use"
}

/* A multi
   line comment. */
resource "aws_instance" "web" {
  ami               = "${var.ami}"
  count             = 2
  source_dest_check = false

  connection {
    user = "root"
  }
}
```

JSON

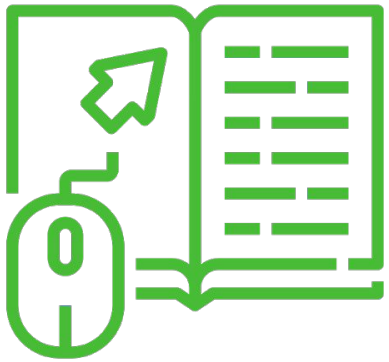
Terraform также поддерживает чтение конфигурационных файлов в формате JSON. Приведенный выше пример конвертирован в JSON.

```
{
  "variable": {
    "ami": {
      "description": "the AMI to use"
    }
  },

  "resource": {
    "aws_instance": {
      "web": {
        "ami": "${var.ami}",
        "count": 2,
        "source_dest_check": false,

        "connection": {
          "user": "root"
        }
      }
    }
  }
}
```

Итоги

Сегодня мы рассмотрели:

- инфраструктуру как код и её особенности;
- программный инструмент Terraform;
- установку и настройку Terraform.



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**.

нетология

Задавайте вопросы и пишите отзыв о лекции!

Александр Зубарев

Александр Зубарев