

День 1: Изучение Terraform на практике

Спикер:
Павел Замошин





План



здать
вопрос

- Что такое Infrastructure as Code

- Terraform и его отличие от других утилит, Terraform State

- Описание HashiCorp Configuration Language

- Базовые команды для работы с Terraform

- Resources, Providers, Data-sources, Outputs, Variables, Locals

- Файловая структура кода

- Мета-аргументы, динамические блоки, условные операторы





В случае проблем



задать
вопрос

Если вдруг вы застряли, у вас ошибка и что-то не получается:

- Попробуйте внимательно прочитать ошибку. Разработчики Terraform постарались сделать его удобным и ошибки вполне читаемые;
- Внимательно ознакомьтесь с предоставленными по теме ссылками, возможно там будет ответ на проблему;
- Посмотрите, как это было решено в коде следующего задания;
- И конечно же, задавайте вопросы в чате.

Если вдруг вы что-то не успели или пропустили — можно взять исходный код следующего задания и продолжить выполнение.





Infrastructure as Code



задать
вопрос

Infrastructure as Code (IaC) — практика управления инфраструктурой через конфигурационные файлы, а не через ручное редактирование.

Основные плюсы:



скорость



цена



уменьшение рисков





Terraform



здать
вопрос

Terraform — open-source cloud-agnostic инструмент оркестрации инфраструктуры от компании HashiCorp.

Инфраструктура описывается с помощью специального декларативного языка HCL или JSON.

HCL

JSON



Terraform



здать
вопрос

terraform.state



terraform.tfvars



terraform-provider.tf



terraform-instances.tf





Terraform State



здать
вопрос

Стейт — это файл в формате JSON, в котором хранится описание ранее созданной инфраструктуры.





Пример Terraform State



задать
вопрос

```
> cat terraform.tfstate
{
  "version": 4,
  "terraform_version": "0.12.4",
  "serial": 13,
  "lineage": "6395ed5e-49d9-d469-2bc4-75c44303c8b7",
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "aws_route53_health_check",
      "name": "LouisJohnBichardRoute53HealthCheck",
      "provider": "provider.aws",
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "child_health_threshold": 0,
            "child_healthchecks": [],
            "cloudwatch_alarm_name": null,
            "cloudwatch_alarm_region": null,
```




О важности стейта



здать
вопрос

Стейт — очень важная часть Terraform-кода. Потеря стейта ни в коем случае недопустима — это равносильно удалению всей информации о когда-либо созданных ресурсах.

Автоматического восстановления стейта сейчас не существует, и решением является либо **ручное добавление** ресурсов в стейт, либо **ручное удаление** ресурсов из консоли.





Сравнение Terraform другими инструментами

Инструменты IaC



здать
вопрос

Configuration Management

(Ansible, Chef, Puppet, SaltStack)

— инструменты настройки
готовой инфраструктуры

Orchestrators

(Terraform, Pulumi, CloudFormation)

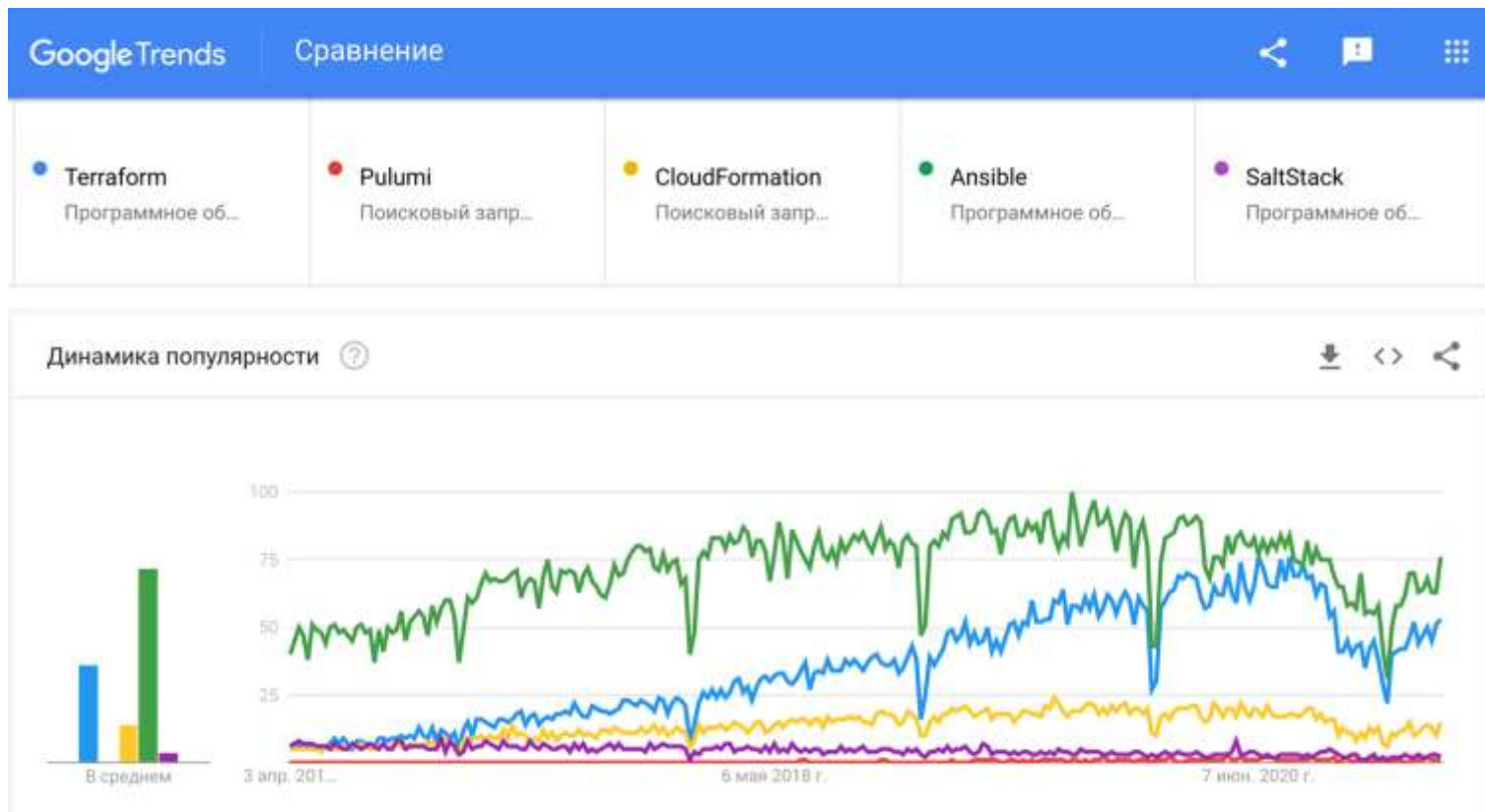
— инструменты создания
инфраструктуры



Сравнение по частоте использования



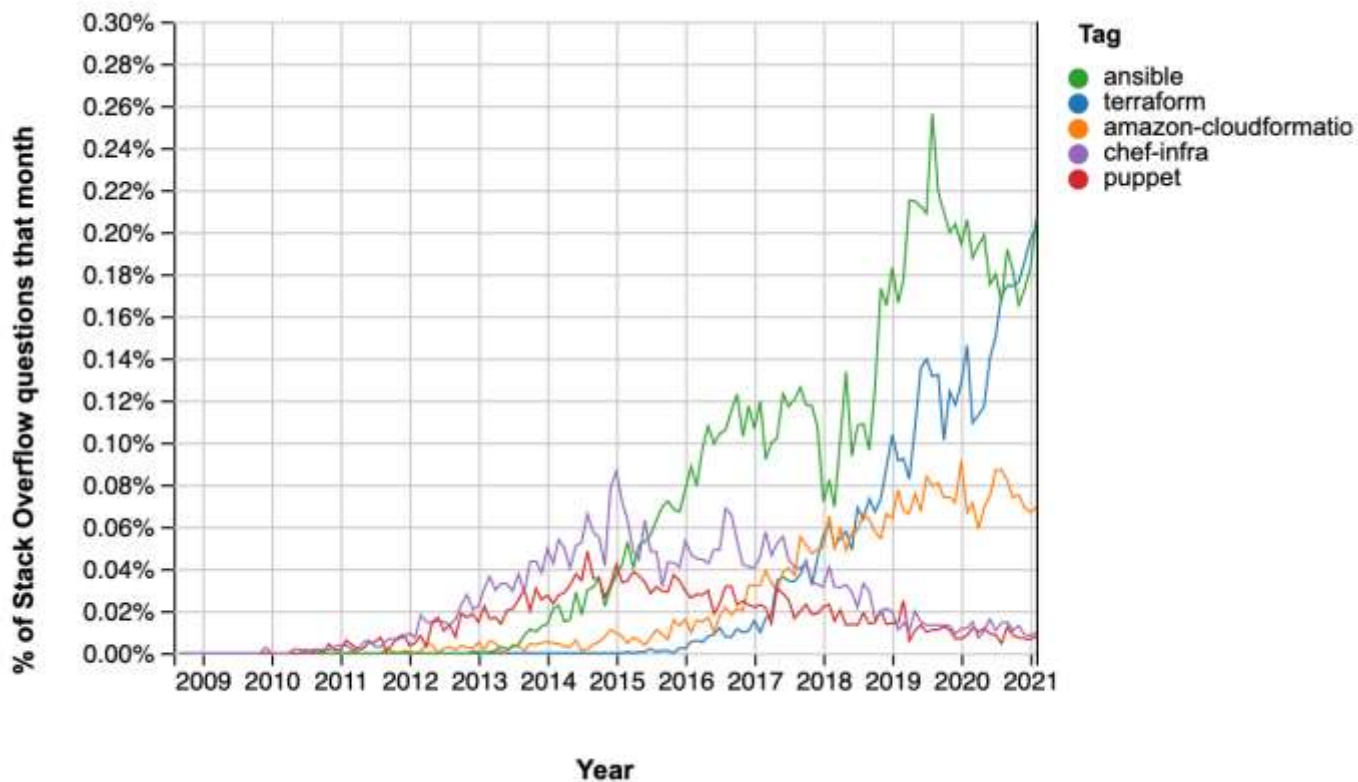
задать
вопрос



Сравнение по частоте использования



задать
вопрос





задать
вопрос

Вопросы





Настройка доступа



здать
вопрос

- Введите в консоль `aws configure`;

- Укажите AWS Access Key ID, AWS Secret Access Key, полученные от Amazon;

- "Default region name" можно выбрать любой, например, "us-east-1";

- "Default output format" можно оставить пустым.



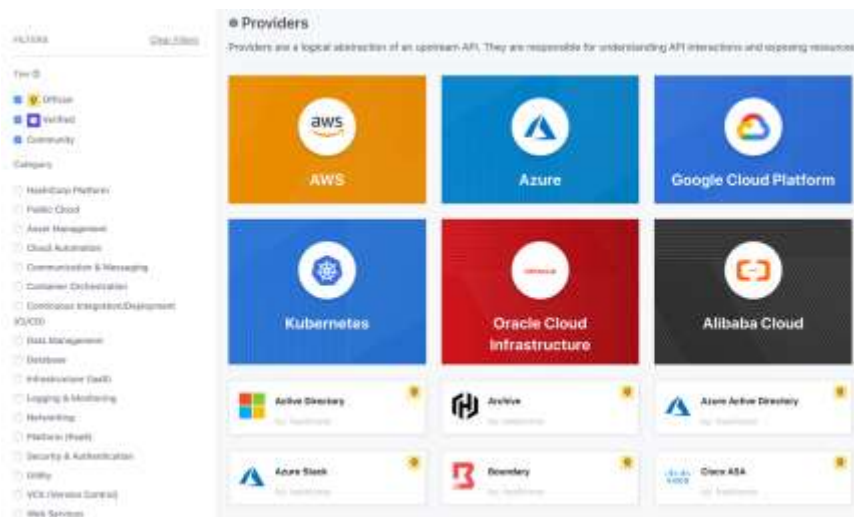
Terraform Provider



задать
вопрос

Провайдер — подключаемый плагин для взаимодействия с внешними системами (например AWS).

Количество провайдеров огромное, их можно найти в **Terraform Registry**:





Terraform Resources



задать
вопрос

Ресурсы — любые объекты, которые мы создаем (бакет S3, виртуальная машина, доменное имя и т.д.).

Провайдеры предоставляют возможность создания ресурсов в системах.





Terraform Resources

Примеры ресурсов из провайдера AWS:

- > DynamoDB
- > DynamoDB Accelerator (DAX)
- ▼ EC2
 - ▼ Resources
 - aws_ami
 - aws_ami_copy
 - aws_ami_from_instance
 - aws_ami_launch_permission
 - aws_ebs_default_kms_key
 - aws_ebs_encryption_by_default
 - aws_ebs_snapshot
 - aws_ebs_snapshot_copy
 - aws_ebs_volume
 - aws ec2 availability zone group



задать
вопрос





HashiCorp Configuration Language



задать
вопрос

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.27"
    }
  }
}

provider "aws" {
  profile = "default"
  region  = "us-west-2"
}

resource "aws_instance" "main" {
  ami          = "ami-830c94e3" # Ubuntu
  instance_type = "t2.micro"

  tags = {
    Name = "Example Instance"
  }
}
```



Типы данных в HCL



задать
вопрос

string

- строка, последовательность символов юникода.
Пример: "example"

number

- число, целое или дробное. Примеры: 10, 3.14

bool

- логический, true или false

list (или tuple)

- список значений. Пример: ["us-east-1", "us-east-2"]

map (или object)

- словарь. Пример: {name = "region", type = "us-east-1"}

null

- специальный тип отсутствующего значения



Команды для работы с Terraform



задать
вопрос

terraform init

— скачивание внешних зависимостей

terraform plan

— отображение вносимых изменений

terraform apply

— применение изменений

terraform destroy

— уничтожение описанной инфраструктуры





Задание 1: работа с существующим Terraform-кодом



задать
вопрос

Создадим с помощью Terraform ресурсы из готового кода:

1. Откройте директорию задания и изучите исходный код;
2. Проинициализируйте репозиторий, спланируйте и примените изменения;
3. Проверьте наличие созданных ресурсов в AWS;
4. Изучите созданный файл стејта — его синтаксис и содержание;
5. Уничтожьте ресурсы и убедитесь, что они исчезли в AWS.





Задание 1: подробное описание



здать
вопрос

1. Откройте директорию с исходным кодом задания;
2. Выполните **"terraform init"** для инициализации, дождитесь загрузки файлов провайдера;
3. Выполните **"terraform plan"**, проверьте вывод — какие ресурсы будут созданы;
4. Выполните **"terraform apply"** для применения изменений;
5. Зайдите в консоль AWS и проверьте созданные ресурсы;
6. В директории с проектом появился файл **terraform.tfstate**. Выведите его на экран и осмотрите;
7. Уничтожьте ресурсы, выполнив команду **"terraform destroy"**;
8. Проверьте, что ресурсы были уничтожены и в консоли AWS.



здать
вопрос

Вопросы



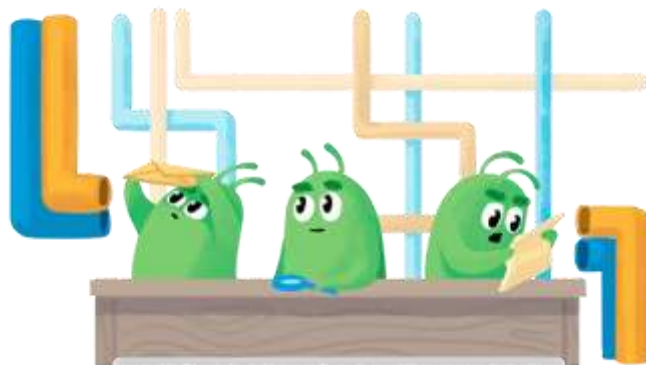


Terraform Data Sources



задать
вопрос

Дата-ресурсы — внешние ресурсы, которые уже были созданы в системе, с которыми мы хотели бы взаимодействовать в исходном коде (например, уже созданный айпи-адрес для виртуальной машины)





Обращение к ресурсам



здать
вопрос

Для использования данных одного ресурса для создания другого, к ним можно обращаться. Например, данные из дата-ресурса можно использовать при создании виртуальной машины.

Обращение к дата-ресурсу идет в формате: `data.тип_ресурса.имя.переменная`

Пример: `data.aws_ami.example.arn`

При обращении к обычным ресурсам указывается только:

`тип_ресурса.имя.переменная`

Пример: `aws_instance.example.arn`



Задание 2: объявление data source



здать
вопрос

1. Получите информацию о образе Ubuntu 20.04 с помощью дата-ресурса. Выберите тип виртуализации hvm. Для помощи воспользуйтесь [документацией](#);
2. В качестве владельца образа укажите идентификатор Canonical: `099720109477` ([если хотите знать, как его получить](#));
3. После создания замените идентификатор `ami` в ресурсе `aws_instance` на `id` из созданного дата-ресурса;
4. Примените внесенные изменения;
5. После выполнения удалите созданную инфраструктуру с помощью Terraform.





Задание 2: подробное описание



здать
вопрос

1. Откройте документацию по [aws_ami](#), изучите пример создания ресурса;
2. Добавьте в свой код блок типа `aws_ami`, назовите его `ubuntu`;
3. Установите переменную `most_recent = true`, это позволит получить последний образ из всех;
4. Установите переменную `owners = ["099720109477"]`, это ID владельца образа - Canonical;
5. Создайте блоки со следующими фильтрами:
 1. `name: ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-*`
 2. `virtualization-type: hvm`
1. В ресурсе виртуальной машины замените параметр `ami`, заданный строкой, ссылкой на ресурс: `data.aws_ami.ubuntu.id`. Важно: кавычки ставить не нужно!
2. Проверьте свою конфигурацию с помощью команды `terraform validate`
3. Попробуйте построить план вносимых изменений в инфраструктуру (`terraform plan`);
4. Если все в порядке — применяйте изменения (`terraform apply`), если нет — внимательно изучите ошибку и внесите исправления;
5. Удалите созданную инфраструктуру (напоминание: `terraform destroy`)



Terraform Outputs



здать
вопрос

Outputs — это значения, которые возвращаются объектами Terraform (например, resource, data-source, module). Каждое возвращаемое значение задается отдельным блоком класса output.

Пример:

```
output "instance_ip_addr" {  
  description = "The private IP address of the instance."  
  value = aws_instance.server.private_ip  
}
```



Также существует параметр **sensitive**, принимающий значения **true** или **false** — является ли значение секретом, который нельзя показывать в консоли.



здать
вопрос

Получение значений output

Возвращаемые значения можно получить из сте́йта с помощью команды `terraform output`. Это позволяет интегрировать Terraform в различные скрипты и пайплайны.

Пример:

```
$ terraform output
instance_ips = [
  "54.43.114.12",
  "52.122.13.4",
  "52.4.116.53"
]
lb_address = "my-app-alb-1657023003.us-east-
1.elb.amazonaws.com"
password = <sensitive>
```



Задание 3: объявление outputs



здать
вопрос

1. Объявите возвращаемые значения для создаваемых ресурсов: айпи-адрес виртуальной машины и идентификатор образа, из которого она создана. Назовите их `instance_ip` и `instance_ami` соответственно;
2. В качестве помощи откройте документацию по [outputs](#) и ресурсам: [instance](#), [ami](#);
3. Примените изменения, а после получите значения output отдельно.





Задание 3: подробное описание



здать
вопрос

1. Добавьте блок **output** под именем **instance_ip** и присвойте ему значение **aws_instance.main.public_ip**, возвращаемое ресурсом виртуальной машины. Также в поле **description** опишите, что это за возвращаемое значение;
2. Добавьте блок **output** под именем **instance_ami** и присвойте ему значение **id**, возвращаемое дата-ресурсом образа виртуальной машины (этот id мы уже использовали при создании виртуальной машины);
3. Обратите внимание: ссылки на resource и на data-source отличаются. В случае с resource не нужно указывать тип ресурса. В случае же с data-source мы начинаем обращаться с **data**;
4. Проверьте и примените внесенные изменения. Проверьте, чтобы возвращаемые значения были выведены в консоли;
5. Используйте команду terraform **output**, чтобы отдельно получить список возвращаемых значений.



Terraform Variables



здать
вопрос

Variables — это способ передать в Terraform-код входящие значения. Если проводить аналогию с программированием, variables — это аргументы функции, а output — это return функции.

Пример:

```
variable "image_id" {  
  type      = string  
  description = "The id of the machine image (AMI) to use for the  
server."  
}
```



Впоследствии, переменную можно будет использовать в исходном коде с помощью приставки **var.** и имени переменной.



Другие параметры variables



здать
вопрос

default

— значение переменной по умолчанию. Делает необязательной передачу переменной;

type

— тип переменной (подробнее на следующем слайде);

description

— описание переменной;

validation

— набор правил и функций для проверки переменной;

sensitive

— является ли переменная секретом. Ограничивает ее вывод при использовании.



Типы переменных



здать
вопрос

Помимо стандартных типов данных Terraform (string, number, bool), в variable можно использовать и сложные типы:

`list(<TYPE>);`

`set(<TYPE>);`

`map(<TYPE>);`

`object({<ATTR NAME> = <TYPE>, ... })`

`tuple([<TYPE>, ...])`

В данном случае, **<TYPE>** представляет собой один из базовых типов, описанных выше.

В случае, если сложный тип принимает любой из базовых типов (например, лист произвольных типов), используется ключевое слово **any**.



Примеры переменных



задать
вопрос

Пример 1, число: `type = number`

Пример 2, лист строк: `type = list(string)`

Пример 3, объект: `object({ name = string, count = number })`





Пример tfvars файла

```
$ cat terraform.tfvars
image_id = "ami-abc123"
availability_zone_names = [
    "us-east-1a",
    "us-west-1c",
]
```



здать
вопрос





здать
вопрос

Порядок загрузки variables

Terraform загружает переменные в следующем порядке:

- Из environment-переменных системы

- Из файла **terraform.tfvars** , если существует

- Из файла **terraform.tfvars.json** , если существует

- Из файлов **.auto.tfvars** или **.auto.tfvars.json** в алфавитном порядке

- Из аргументов командной строки **var** и **var-file** при вызове Terraform, в порядке их употребления



Перед выполнением задания — интерполяция



задать
вопрос

Для выполнения следующего задания нам понадобится использовать **интерполяцию** — использование переменных в строке.

Пример:

```
"line-begining-${var.something}-and-ending"
```

Переменная `var.something` будет добавлена в центре строки.
Синтаксис `${}` внутри строки обозначает вставку выражения.



Задание 4: объявление переменных



здать
вопрос

1. Объявите переменную. Дайте ей имя **ubuntu_version**, задайте описание, тип и значение по умолчанию — версию, которую мы использовали;
2. Используйте интерполяцию в месте использования версии AMI;
3. Примените внесенные изменения. С виртуальной машиной ничего не должно произойти;
4. Теперь создайте файл `terraform.tfvars` и задайте в нем **ubuntu_version = "bionic-18.04"**
5. Примените изменения. Виртуальная машина должна пересоздаться с другой версией AMI.





Terraform Local Values



здать
вопрос

Local value — это значение, которое можно задать один раз и после использовать в разных блоках кода. Очевидный вопрос: "В чем отличие от variable?"

variables

— это аргументы функции;

outputs

— это return функции;

locals

— это временные переменные внутри функции, которые нужны только во время работы функции.



здать
вопрос

Пример locals

Local-переменные задаются в специальном блоке **locals**. И в отличие от **variable/output**, в **locals** можно задавать сразу несколько переменных.

Пример:

```
locals {  
  service_name = "forum"  
  owner        = "Community Team"  
}
```

Чтобы сослаться на local-переменную — используется префикс **local.** и имя переменной. Пример: **local.service_name**



Terraform функции



здать
вопрос

В Terraform есть огромное количество [функций](#), которые можно применять в коде.

Примеры:

max — выявление максимального из списка;

min — минимального;

join — склейка списка строк с помощью разделителя;

concat — соединение списков;

merge — объединение словарей и так далее.

Категории функций:

▼ Functions

• Overview

› Numeric Functions

› String Functions

› Collection Functions

› Encoding Functions

› Filesystem Functions

› Date and Time Functions

› Hash and Crypto Functions

› IP Network Functions

› Type Conversion Functions



Задание 5: объявление locals



задать
вопрос

Объявим с помощью локальных переменных несколько стандартных тегов, которые после будут использоваться для всех ресурсов:

1. Создайте блок **locals** и объявите в нем переменную-словарь **aws_tags**.
Заполните ее парами ключ-значение: **environment="dev"**, **project="slurm"**
2. В блоке ресурса виртуальной машины объедините уже объявленные теги и общие теги из **locals** с помощью функции **merge**. [Документация](#);
3. Проверьте и примените изменения





Файловая структура проекта



задать
вопрос

variables.tf

— блоки variable

outputs.tf

— блоки output

data.tf

— блоки data

versions.tf

— блок terraform

providers.tf

— блоки provider

остальные ресурсы хорошо
разделять по их типам
(например, создание ec2
выделить в ec2.tf)



Задание 6: разделение проекта



здать
вопрос

Давайте разделим проект на описанные файлы.

1. Перенесите ресурсы описанных типов по файлам `variables.tf`, `outputs.tf`, `data.tf`, `versions.tf`, `providers.tf`, `locals.tf`. Сам блок создания виртуальных машин пока оставим в файле `main.tf`;
1. Проверьте и примените внесенные изменения. Если все правильно — изменений в инфраструктуру быть не должно.





задать
вопрос

Мета-аргументы Terraform

Мета-аргументы — аргументы, которые можно использовать с любым типом ресурса (вспомогательные).

depends_on

— список ресурсов, от которых зависит

count

— количество объектов ресурса, которое будет создано

for_each

— как count, только с параметрами

provider

— какой провайдер использовать для создания ресурса

lifecycle

— описание поведения при изменении/удалении ресурса



задать
вопрос

Мета-аргумент depends_on

Список ресурсов, от которых зависит описываемый ресурс.

Пример:

```
resource "aws_instance" "instance1"
{
  ami          = "ami-a1b2c3d4"
  instance_type = "t2.micro"
}

resource "aws_instance" "instance2"
{
  ami          = "ami-a1b2c3d4"
  instance_type = "t2.micro"
  depends_on = [
    aws_instance.instance1,
  ]
}
```



Мета-аргумент count



здать
вопрос

Количество объектов ресурса, которое будет создано.

Пример:

```
resource "aws_instance" "server" {  
  count = 4  
  
  ami          = "ami-a1b2c3d4"  
  instance_type = "t2.micro"  
  
  tags = {  
    Name = "Server ${count.index}"  
  }  
}
```




здать
вопрос

Мета-аргумент for_each

Какие ресурсы требуется создать. Проходит по словарю или множеству.

Пример:

```
resource "aws_instance" "server" {  
  for_each = {  
    "team 1" = "t2.micro"  
    "team 2" = "i3.large"  
  }  
  
  ami          = "ami-a1b2c3d4"  
  instance_type = each.value  
  
  tags = {  
    Name = "Server for ${each.key}"  
  }  
}
```



здать
вопрос

Мета-аргумент provider

Какой провайдер использовать для создания ресурса.

Пример:

```
provider "google" {  
  region = "us-central1"  
}  
  
provider "google" {  
  alias   = "europe"  
  region = "europe-west1"  
}  
  
resource "google_compute_instance" "example" {  
  provider = google.europe  
  # ...  
}
```



здать
вопрос

Мета-аргумент lifecycle

Описание поведения при изменении/удалении ресурса. Словарь, принимающий следующие ключи:

create_before_destroy
(bool)

— создавать ресурс перед его удалением

prevent_destroy
(bool)

— запретить удаление ресурса, план с его пересозданием вызовет ошибку

ignore_changes **(list)**

— игнорировать изменения в следующих атрибутах



Мета-аргумент lifecycle



задать
вопрос

Описание поведения при изменении/удалении ресурса. Словарь, принимающий следующие ключи:

Пример:

```
resource "aws_instance" "example" {  
  # ...  
  lifecycle {  
    create_before_destroy = true  
  }  
}
```



Обращение к списку ресурсов



задать
вопрос

При создании нескольких ресурсов с помощью `count` или `for_each` может потребоваться обратиться к [НИМ](#);

Для обращения к одному ресурсу можно использовать индексы.

Пример: `aws_instance.example[0].id`

А для обращения ко всем сразу использовать `*`.

Пример: `aws_instance.example[*].id`





Задание 7: Мета-аргументы



задать
вопрос

Сделаем так, чтобы создавалось две идентичных виртуальных машины, а к их имени добавлялся номер:

1. Объявите переменную для задания количества виртуальных машин (для начала двух);
2. Используйте ее с мета-аргументом `count` в ресурсе;
3. Измените объявленный аутпут на обращение к списку ресурсов;
4. Проверьте изменения и примените.





Задание 7: Подробное описание



здать
вопрос

Сделаем так, чтобы создавалось две идентичных виртуальных машины, а к их имени добавлялся номер:

1. Создайте числовую переменную (**variable**) и назовите ее **instance_count**;
2. Добавьте мета-аргумент **count** в блок создания виртуальной машины и присвойте ему значение переменной (например, 2);
3. В тегах виртуальной машины добавьте значение **count.index** к имени виртуальной машины;
4. В **instance_ip** output измените **value** на **aws_instance.example.*.public_ip**
5. Проверьте изменения и примените их.



Использование условных выражений



задать
вопрос

В Terraform также можно использовать условия.

Синтаксис: `condition ? true_val : false_val`

Пример:

```
var.a != "" ? var.a : "default-  
a"
```

Такие условия можно использовать и в мета-аргументе `count`, чтобы включить или выключить создание ресурса.





Блок dynamic



здать
вопрос

Блок dynamic дает возможность динамически генерировать блоки в **resource**, **data**, **provider**, **provisioner**.

Пример:

```
resource "example_resource" "example" {  
  name = "tf-test-name"  
  setting {  
    name = ...  
    value = ...  
  }  
  setting {  
    name = ...  
    value = ...  
  }  
}
```

Мы можем использовать **for_each** для создания нескольких ресурсов, но нельзя использовать его для создания нескольких блоков **setting**.



Пример с dynamic



здать
вопрос

```
data "aws_ami" "ubuntu" {
  most_recent = true
  dynamic "filter" {
    for_each = local.aws_ami_filters
    content {
      name = filter.value["name"]
      values = filter.value["values"]
    }
  }
  owners = ["099720109477"] # Canonical
}

locals {
  aws_ami_filters = [
    {
      name = "name"
      values = ["ubuntu/images/hvm-ssd/ubuntu-${var.ubuntu_version}-amd64-server-*"]
    },
    {
      name = "virtualization-type"
      values = ["hvm"]
    }
  ]
}
```





Задание 8: Работа с dynamic



задать
вопрос

При объявлении `aws_ami` мы использовали два блока `filter`. Давайте переделаем их объявление на использование `dynamic`. При выполнении пользуйтесь [документацией](#).

1. Объявите локальную переменную-лист `aws_ami_filters`, элементами которой будут два словаря со значениями `name` и `values`;
1. Создайте блок `dynamic "filter"` в блоке `aws_ami`, вместо блоков фильтров. Используйте `local.aws_ami_filters` для `for_each`;
1. Проверьте и примените изменения.





Резюме урока



задать
вопрос

Поздравляю! Это завершение дня 1, мы на полпути к поставленной цели.

Сегодня мы изучили:

- Что такое Infrastructure as Code;
- Для чего нужен Terraform и какое место он занимает в своей области;
- Как разворачивать инфраструктуру с помощью Terraform;
- А также основные концепции Terraform:
 - Resources, Providers, Data-sources, Outputs, Variables, Locals;
 - Мета-аргументы, динамические блоки, условные операторы;
 - Файловую структуру проектов.

Завтра будет больше и интереснее!



До завтра!

Спикер:
Павел Замошин

