Управляющие конструкции в коде Terraform

Елисей Ильин
DevOps-инженер в Itransition

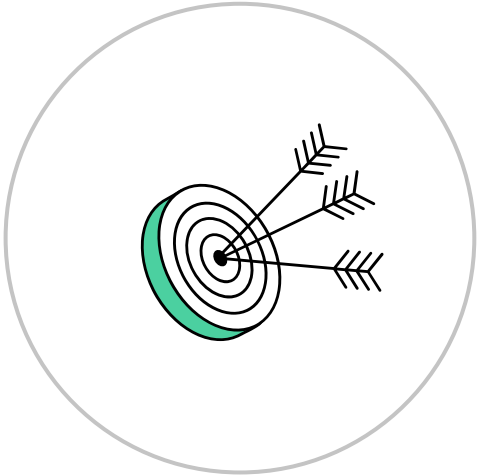
Елисей Ильин

О спикере:

- DevOps-инженер
- Опыт работы в IT 6 лет

Цели занятия

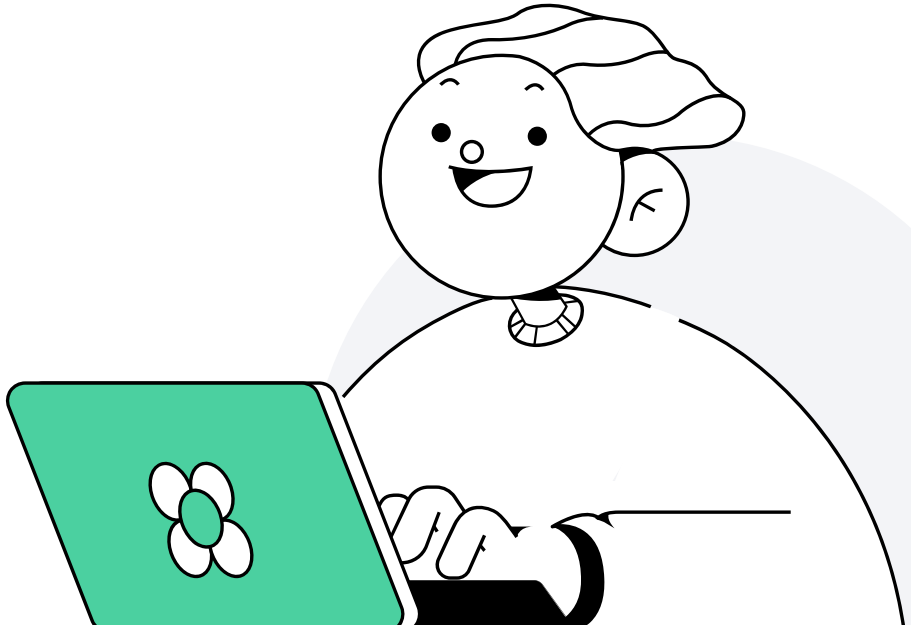
- Сделать наш код более динамичным, добавив в него логику
- Научиться дополнительно настраивать созданные Terraform ресурсы



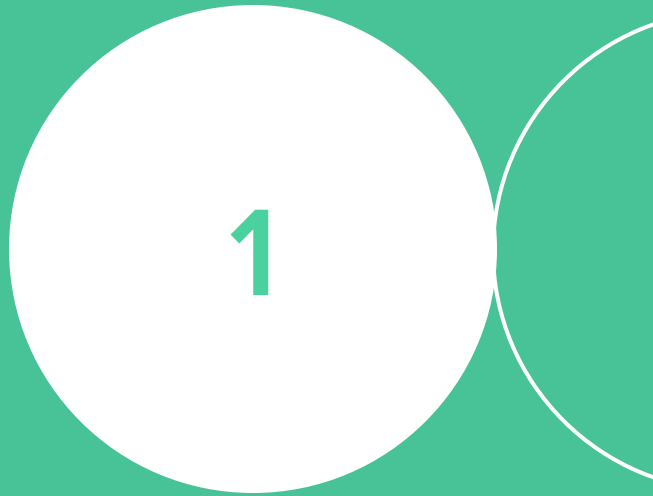
План занятия

1. Meta аргументы
2. Expressions
3. Provisioners
4. Итоги занятия
5. Домашнее задание

*Нажми на нужный раздел для перехода



Meta аргументы



Мета-аргументы – специальные конструкции, расширяющие функционал terraform-blocks

depends_on

Terraform provider самостоятельно выстраивает правильную последовательность создания **ресурсов**. Но возможны исключения и ошибки.

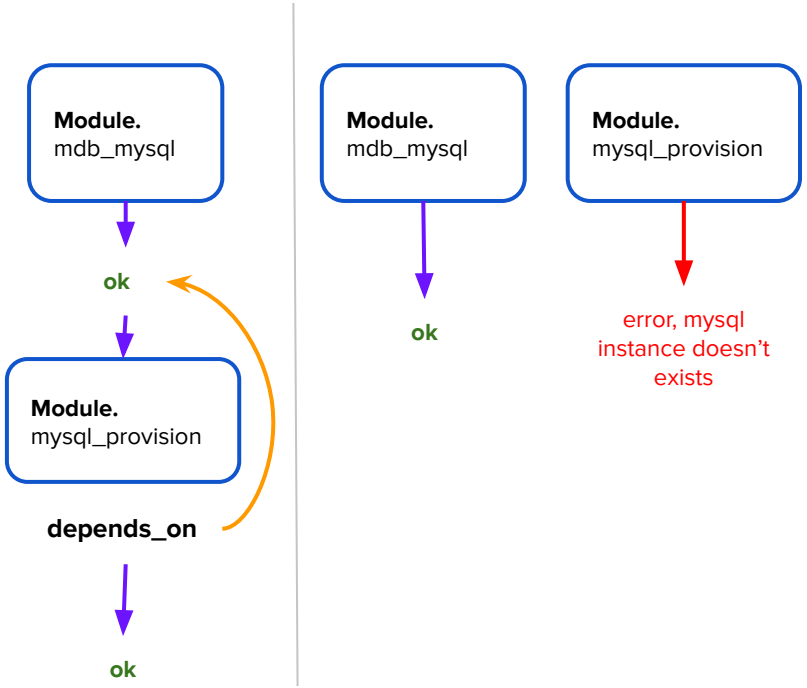
Аргумент **depends_on** в блоках позволяет управлять порядком создания **ресурсов и модулей.**

depends_on = [**resource.A , module.B, ...**]

depends_on

Обычно **depends_on** используется **только** для определения порядка создания **child modules**. Работу с модулями мы изучим в следующей лекции

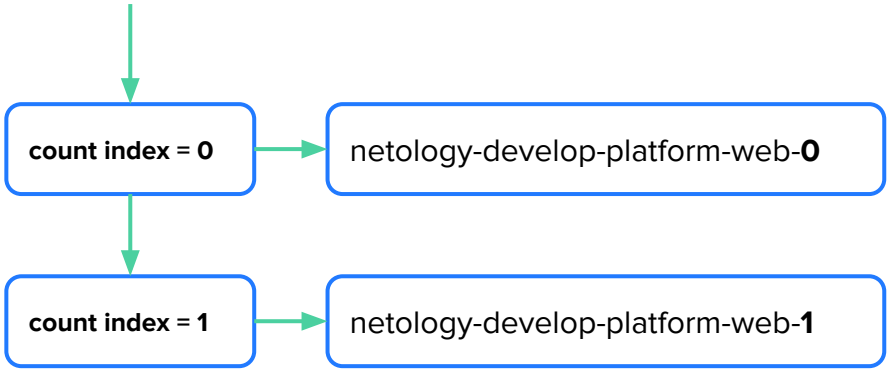
```
#Объявление блока модуля
module "mdb_mysql" {
  instance_name = "test_instance"
}

module "mysql_provision" {
  depends_on = [module.mdb_mysql]
  database    = "test_db"
  user_name   = "test_user"
  instance_id = module.mdb_mysql.id
}
```



count loop

- Позволяет указать число экземпляров данного ресурса, которые необходимо создать
- Инициализирует итерируемую переменную **count.index**
- Подходит для создания **идентичных** ресурсов
- Если требуется создать отличающиеся ресурсы, стоит использовать мета аргумент **for_each**

```
resource "yandex_compute_instance" "web" {
  count = 2
  name =
"netology-develop-platform-web-${count.index}"
…
}
```



for_each loop with set

В отличие от count в качестве указателя количества экземпляров принимает переменную типа **set** или **map**.

Доступны атрибуты:

- **each.key**
- **each.value**

В случае set each.key==each.value

```
resource "yandex_compute_instance" "web" {
  for_each = toset([ 0, 1 ])
  name =
"netology-develop-platform-web-${each.key}"
}
```

```
"netology-develop-platform-web-0"

"netology-develop-platform-web-1"
```

for_each loop with map

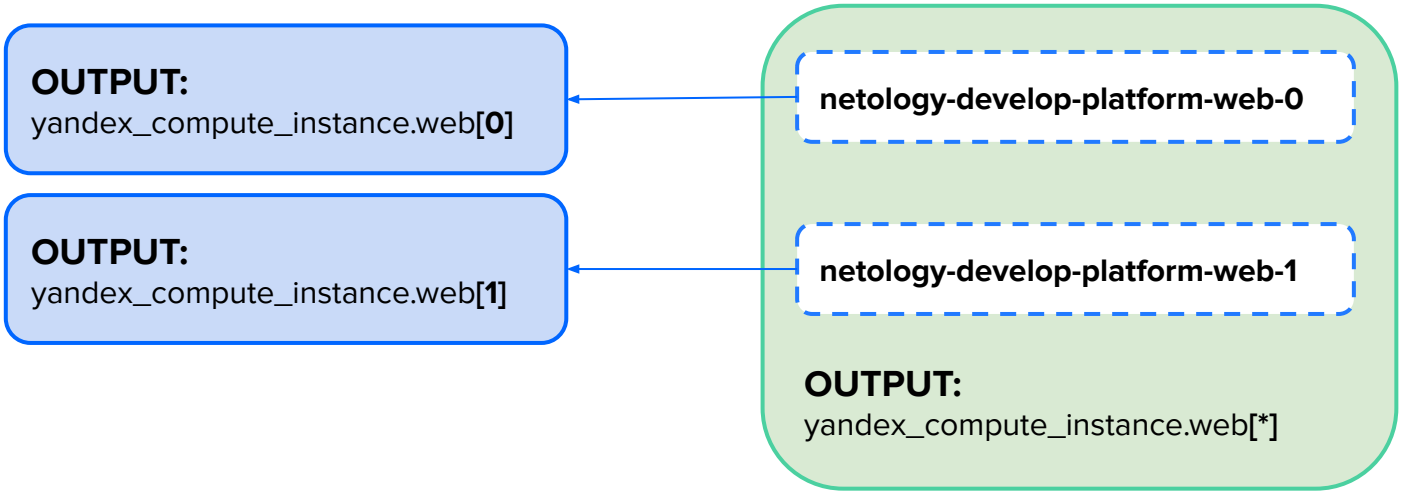
```
resource "yandex_compute_instance" "web" {
  for_each = {
     0 = "first"
     1 = "second"
}
  name = "netology-develop-platform-web-${each.key}"
  tags     =  {
    Name = ${each.value}
  }
}
```

```
"netology-develop-platform-web-0" tags=["first"]

"netology-develop-platform-web-1" tags=["second"]
```

Обращение к ресурсам при использовании count или for_each

Для конкретного ресурса используется индекс: yandex_compute_instance.web**[0]**

Для всех: yandex_compute_instance.web**[*]** **или** yandex_compute_instance.web**.***



lifecycle

Мета-аргумент lifecycle позволяет изменить поведение при внесении изменений в ресурсы:

- **create_before_destroy** = true

  default =false

- **prevent_destroy** = true, защита от случайного удаления
- **ignore_changes** = [tags], игнорировать изменения, например, тегов. Полезно, если тегами управляет стороннее ПО

```
resource "yandex_compute_instance" "platform" {
  name        = "netology-develop-platform-web"
  …
  lifecycle {
    prevent_destroy = true
  }
}
```

provider

Позволяет **переопределить настройки** провайдера для выбранного ресурса.

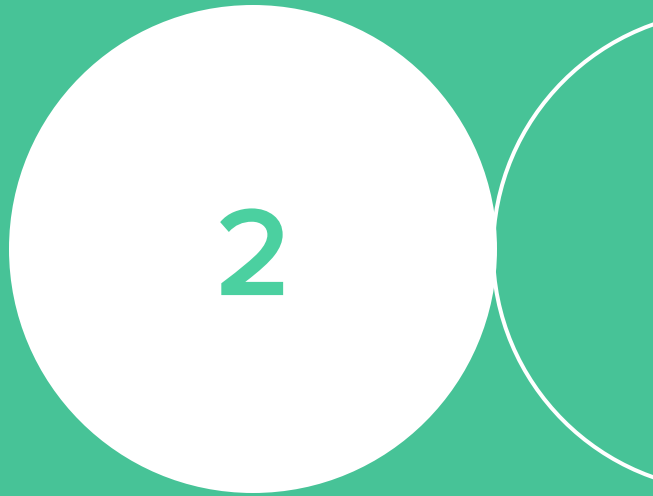
Обычно этот аргумент имеет смысл использовать только в мульти-региональных облаках, таких как: **aws, gcp, azure, digital_ocean, selectel** для выбора региона, в котором создается ресурс.

Yandex Cloud на данный момент имеет только один регион

```
provider "aws" {
  alias  = "eu-west-1"
  region = "eu-west-1"
  access_key = "var.aws_ak_west"
  secret_key = "var.aws_sk_west"
}

provider "aws" {
  alias  = "eu-central-1"
  region = "eu-central-1"
  access_key = "var.aws_ak_eu"
  secret_key = "var.aws_sk_eu"
}

resource "aws_instance" "ec2_eu_central1" {
   provider      = aws.eu-central-1
   ami           = "ami-0ff338189efb7ed37"
   instance_type =  "t2.micro"
   count = 1
}

#В примере показан выбор провайдера по alias
```

Expressions

Выражения



Simple expressions

Выражения — это **значения**, вычисляемые в процессе выполнения кода. Они позволяют сделать ваш код более гибким.

- Любой тип данных terraform
- Индексы и атрибуты
- Ссылка на именованные значения
- Арифметические и логические операторы
- Интерполяция строк
- A here document (HereDoc) строки

- `list[5], map["key"]`
- `var.<NAME>, <RESOURCE TYPE>.<NAME>`
- `5+5, a!=b,a==b,a >=5, &&, ||`
- `${...}`

```
block {
  value = <<-EOL
  hello
    world
  EOL
}
```

Результат:

```
hello
  world
```

Functions

Язык HCL не позволяет добавлять пользовательские функции, но предоставляет множество встроенных.

Вызов функции:

```
<FUNCTION NAME>(<ARGUMENT 1>, <ARGUMENT 2>...<ARGUMENT N>)
```

Примеры:

>**join**(",", ["Hello ", "world ", "!"])	**Результат:** "Hello world!"
>**split**("_", "A_B_C_D")	**Результат:** ["A", "B", "C", "D",]
>**concat**([1,2,3], [4,5,6])	**Результат:** [1, 2, 3, 4, 5, 6,]
>**merge**({ "1": "A " }, { "2": "B" })	**Результат:** { "1" = "A" , "2" = "B" }

Functions

Виды функций

- Числовые
- Строковые
- Коллекции
- Дата и время
- Хэш и шифр
- Сеть IP
- Файловая система
- Кодирование
- Преобразование типов данных

Условные выражения

Используются для логического выбора между двумя значениями.

Синтаксис:

```
условие ? истинное значение : ложное значение
```

Пример:

```
mysql_hosts_count = var.env_name == "production" ? 3 : 1
```

Также, они используются в meta аргументе **count** для создания ресурсов по условию.

Пример:

```
count = var.bastion_instance == true ? 1 : 0
```

for loop позволяет итерироваться по содержимому list и map, применяя к нему функции, условные выражения, преобразование данных

for loop

Для list: [for <ITEM> in <LIST> : <OUTPUT_KEY> => <OUTPUT_VALUE>]
Пример: `test_list = ["develop", "staging", "production"]`

```
[ for i in local.test_list : upper(i) if env !="develop" ]
[
  "STAGING",
  "PRODUCTION", ]
```

Для map: { for <KEY>, <VALUE> in <MAP> : <OUTPUT_KEY> = <OUTPUT_VALUE> }
Пример: `test_map = { John = "admin", Alex = "user" }`

```
> [for k,v in local.test_map : "${k} is ${v}" ]
[
  "Alex is user",
  "John is admin", ]
```

Directives

Конструкция **%{ ... }** позволяет итерироваться по list, map или set.
Поддерживает функции, выражения и условную логику.

Синтаксис:
```
%{ for <ITEM> in <COLLECTION> }<BODY>%{ endfor}
```

В строковой директиве можно указать маркер **~** , чтобы удалить все пробелы и переносы строки:

- в начале **%{~ .. }**
- в конце **%{ .. ~}**
- в начале и в конце **%{~ .. ~}**

Directives

```
Пример:
locals{
  test_list = ["develop", "staging", "production"]
}

> "%{ for env in local.test_list **${upper(env)}_!%{endfor}"

"**DEVELOP_!**STAGING_!**PRODUCTION_!"
```

Dynamic Blocks используются для динамической генерации многократно повторяющихся, вложенных блоков

Dynamic Blocks

Рассмотрим пример создания группы безопасности в YC (firewall для ресурсов). Внимание — **сервис находится на стадии Preview.**
Необходимо создать **отдельный** блок **ingress** (входящее правило) или **egress** (исходящее правило) для **каждой** записи.

Недостатки:

1. Правил может быть **огромное** количество
2. Со временем количество правил может изменяться, что потребует править код
3. Для каждого окружения (dev, prod) придётся хардкодить свой код

На следующих слайдах рассмотрим преимущества dynamic block

```
#Хардкод способ без dynamic block
resource "yandex_vpc_security_group" "all_to_all" {
  name        = "web-server"
  ……..
  ingress {
    protocol             = "TCP"
    v4_cidr_blocks =  ["0.0.0.0/0"]
    port                      = 22
  }
  ingress {
    protocol             = "TCP"
    v4_cidr_blocks =  ["0.0.0.0/0"]
    port                      = 80
  }
  ingress {
    protocol             = "TCP"
    v4_cidr_blocks =  ["0.0.0.0/0"]
    port                      = 443
  }
  egress {
    protocol             = "TCP"
    v4_cidr_blocks =  ["0.0.0.0/0"]
    from_port       = 0
     to_port          = 65365
  }
  ……..A long-long story in a git far-far away..……
}
```

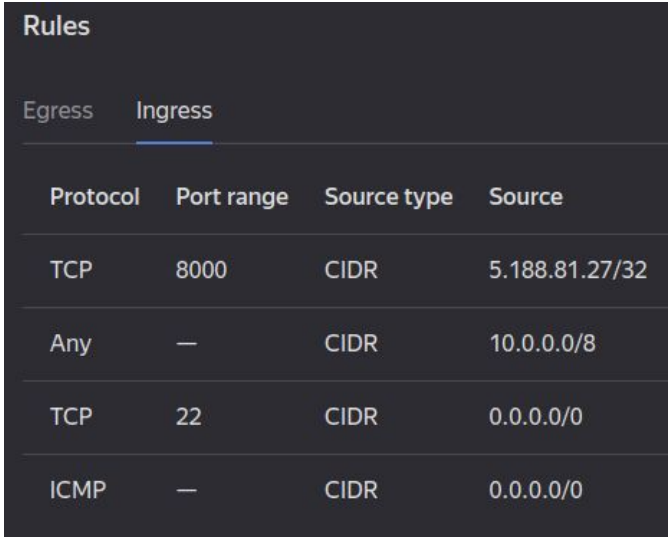
 # Описание переменной в файле security.tf

```
variable "security_group_ingress" {
  type = list(object(
    {
      protocol    = string
      description = string
      v4_cidr_blocks = list(string)
      port        = optional(number)
      from_port   = optional(number)
      to_port     = optional(number)
  }))
  default = []
}


variable "security_group_eggress" {
  type = list(object(
    {
      protocol    = string
      description = string
      v4_cidr_blocks = list(string)
      port        = optional(number)
      from_port   = optional(number)
      to_port     = optional(number)
   }))
  default = []
}
```

#Загружаем переменные из файла security.auto.tfvars

```
security_group_ingress = [
    {
      protocol            = "TCP"
      description        = "разрешить входящий ssh"
      v4_cidr_blocks =  ["0.0.0.0/0"]
      port                    = 22
    },
    {
      protocol             = "TCP"
      description        = "разрешить входящий  http"
      v4_cidr_blocks = ["0.0.0.0/0"]
      port                    = 80
    },
    {
     protocol             = "TCP"
      description        = "разрешить входящий https"
      v4_cidr_blocks = ["0.0.0.0/0"]
      port                    = 443
    },
]

security_group_egress = [
{     protocol           = "TCP"
      description        = "разрешить весь исходящий трафик"
      v4_cidr_blocks = ["0.0.0.0/0"]
      from_port       = 0
     to_port           = 65365
 },
]
```

```
#Способ с использованием dynamic block
resource "yandex_vpc_security_group" "example" {
  name        = "example_dynamic"
  network_id  = yandex_vpc_network.develop.id
  folder_id   = var.folder_id

  dynamic "ingress" {
    for_each = var.security_group_ingress
    content {
      protocol       = lookup(ingress.value, "protocol", null)
      description    = lookup(ingress.value, "description", null)
      port           = lookup(ingress.value, "port", null)
      from_port      = lookup(ingress.value, "from_port", null)
      to_port        = lookup(ingress.value, "to_port", null)
      v4_cidr_blocks = lookup(ingress.value, "v4_cidr_blocks",
null)
    }
  }
  dynamic "egress" {
    for_each = var.security_group_eggress
    content {
      protocol       = lookup(egress.value, "protocol", null)
      description    = lookup(egress.value, "description", null)
      port           = lookup(egress.value, "port", null)
      from_port      = lookup(egress.value, "from_port", null)
      to_port        = lookup(egress.value, "to_port", null)
      v4_cidr_blocks = lookup(egress.value, "v4_cidr_blocks", null)
    }
  }
}
```



Шаблонизация

Функция tepmplatefile

```
(<путь к файлу-шаблону>, <переменные>)
```

Рекомендуемые расширение файлов-шаблонов — ***.tftpl или *.tpl**

Для наполнения файлов шаблонов используются **for loop и directives**

Пример шаблона для Ansible inventory

```
> templatefile("./ansible.tftpl",  { webservers = { server1="1.1.1.1", server2="2.2.2.2" }})
```

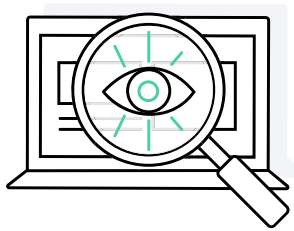
```
[servers]

%{~ for k,v in webservers ~}
${k}   ansible_host = ${v}
%{~ endfor ~}
```

Результат:

```
[servers]

server1   ansible_host = 1.1.1.1
server2   ansible_host = 2.2.2.2
```

Демонстрация работы

- Terraform console (ссылка на демо-код на GitHub)
- Dynamic blocks



Provisioners

Блок provisioner используется для выполнения команд на локальном или удаленном сервере для первоначальной «*донастройки*» (**bootstrap**) целевого ресурса

Блок provisioner

Объявляются внутри блока **resourse {} и выполняются только один раз, сразу после создания этого ресурса.**

Если поместить **provisioner** в специальный **null_resource**, то его можно запускать по условию с помощью блока **trigger {}**.

Типы provisioners:

- file
- local-exec
- remote-exec

Вместо provisioners разработчики Terraform рекомендуют использовать подготовленные **образы Packer или cloud-init,** входящий во все Linux ОС

```
resource "yandex_compute_instance" "web" {
…
     provisioner "<provisioner type>" {
         command 1
         command 2
     }

     provisioner "<provisioner type>" {
         command 1
         command 2
     }
…
}
```

file provisioner

Копирует файлы или директории с локального Terraform-сервера на удаленную ВМ.

Требует блок **connection { .. }**
для настройки авторизации.

Может пригодится для копирования:

- скриптов
- сертификатов
- конфиг-файлов

```
resource "yandex_compute_instance" "web" {
…
  provisioner "file" {
    source      = "conf/myapp.conf"
    destination = "/etc/myapp.conf"
  }

  provisioner "file" {
    content     = "this is test content"
    destination = "/tmp/content.md"
  }

…
}
```

local-exec provisioner позволяет выполнить shell-команду на локальном сервере (там, где запускается Terraform)

local-exec provisioner

Часто используется в **null_resource** для запуска ansible-playbook с сервера, где запускается Terraform. Команды внутри данного ресурса **выполняются по порядку.**

Триггером для запуска provisioner «local-exec» в данном примере служит **изменение значения текущего времени** (т.е. запускается всегда).

Таким образом Terraform сначала создает ВМ, а после запускает ansible для её настройки

```
resource "null_resource" "web_hosts_provision" {
depends_on = [yandex_compute_instance.web]

#Добавление ssh ключа в ssh-agent
  provisioner "local-exec" {
    command = "echo '${var.private_key}' | ssh-add -"
  }
#Создание inventory из файла шаблона
 provisioner "local-exec" {
    command = <<-EOA
    echo "${templatefile("ansible_inventory.yml.tftpl",
   { hosts = yandex_compute_instance.web[*] })}" > hosts.yml
    EOA
  }
#Запуск ansible-playbook
  provisioner "local-exec" {
    command  = "ansible-playbook -i hosts.yml provision.yml"
    interpreter = ["bash"]
    environment = { ANSIBLE_HOST_KEY_CHECKING = "False" }

    triggers = {  always_run = "${timestamp()}" }
  }
}
```

remote-exec provisioner

Выполняет команды на ВМ удаленно.

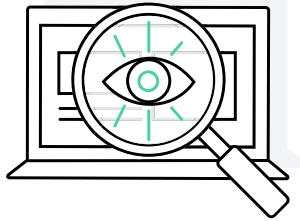
Требует блок **connection { }** для настройки авторизации

Для указания адреса подключения используется специальная переменная **self**

```
resource "yandex_compute_instance" "web" {
…
    connection {
      type                = "ssh"
      user                = "root"
      private_key = var.id_rsa
      host                = self.public_ip
    }

    provisioner "local-exec" {
      command = " …"
    }
…
}
```

Демонстрация работы

- teplatefile: ansible inventory
- local-exec ansible



Итоги занятия

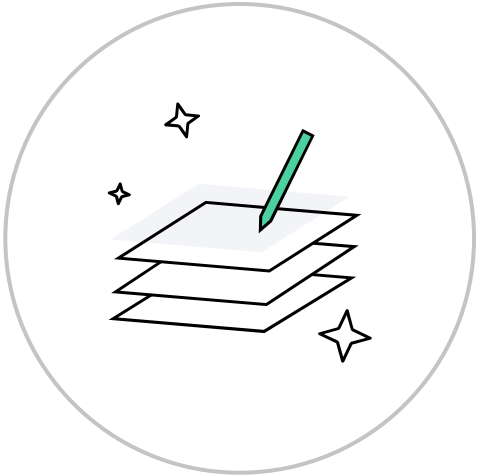
Сегодня мы

1. Узнали, для чего нужны Мета-аргументы
2. Разобрали виды выражений
3. Познакомились с provisioners, совместным использованием terraform+ansible

Домашнее задание

Давайте посмотрим ваше домашнее задание.

1. Вопросы по домашней работе задавайте в чате группы
2. Задачи можно сдавать по частям
3. Зачёт по домашней работе ставят после того, как приняты все задачи



Дополнительные материалы

- Документация expressions
- Документация provisioners

Задавайте вопросы и пишите отзыв о лекции

Елисей Ильин
DevOps-инженер в Itransition