Продвинутые методы работы с Terraform

Евгений Мисяков
SRE инженер в Нетологии

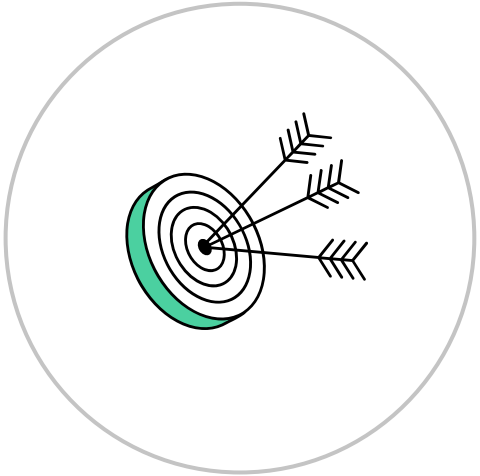
Евгений Мисяков

О спикере:

- SRE инженер в Нетологии

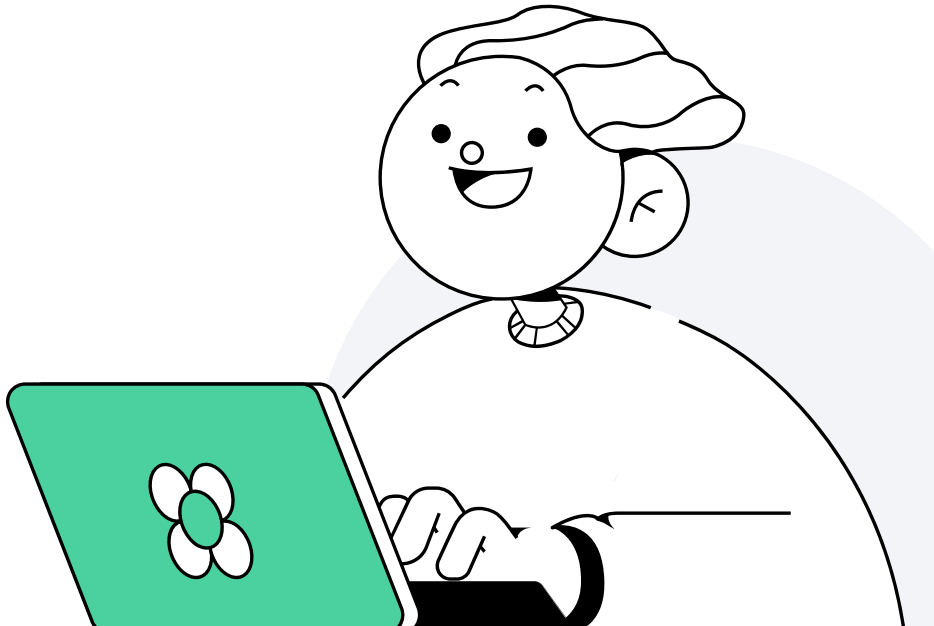
Цели занятия

- Научиться переиспользовать Terraform-код
- Детально изучить всё, что связано с Terraform state



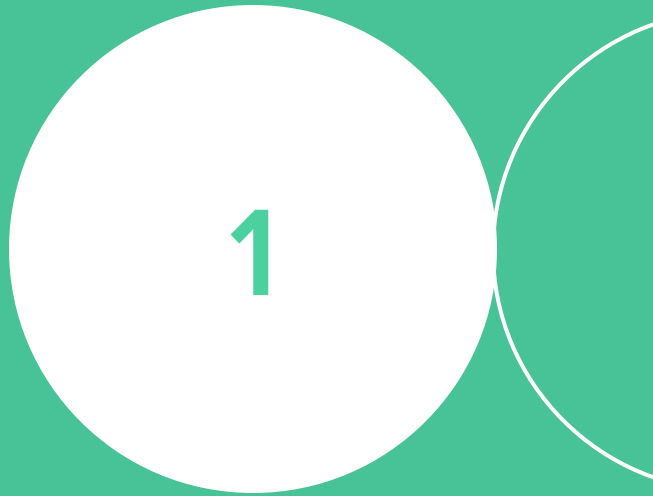
План занятия

1. Child modules
2. Управление state
3. Изолирование state
4. Полезные утилиты и инструменты
5. Итоги занятия
6. Домашнее задание



*Нажми на нужный раздел для перехода

Child modules



Child module представляет собой отдельную конфигурацию ресурсов, которая может быть вызвана внутри родительского модуля.

Все ресурсы, имена и переменные **изолированы в области действия модуля.** Это означает, что всё, что определено внутри модуля, не будет видимым за его пределами и будут исключены конфликты с другими модулями или ресурсами

Child modules

Дочерние модули могут вызываться **многократно, вызывать свои** дочерние модули и использоваться **в разных проектах.**

Это позволяет переиспользовать уже написанный и протестированный open-source-комьюнити или вашей командой Terraform-код и реализовать принцип **DRY(don't repeat yourself).**

Для вызова используется блок **module {..}.**

Обязательно указывайте метааргумент **source.**

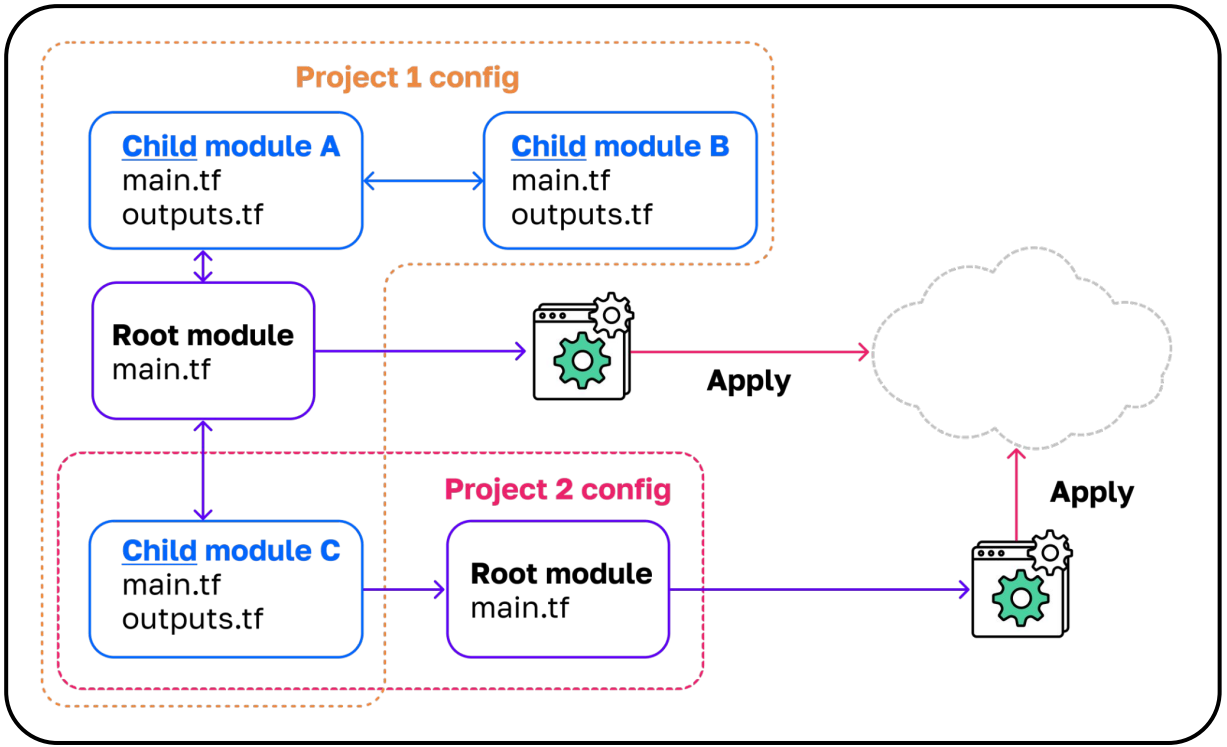
Желательно указать входящие аргументы

```
#vpc.tf

module "vpc_develop" {
    source          = "./vpc"
    env_name        = "develop"
 }

module "vpc_staging" {
    source          = "./vpc"
    env_name        = "staging"
 }
```

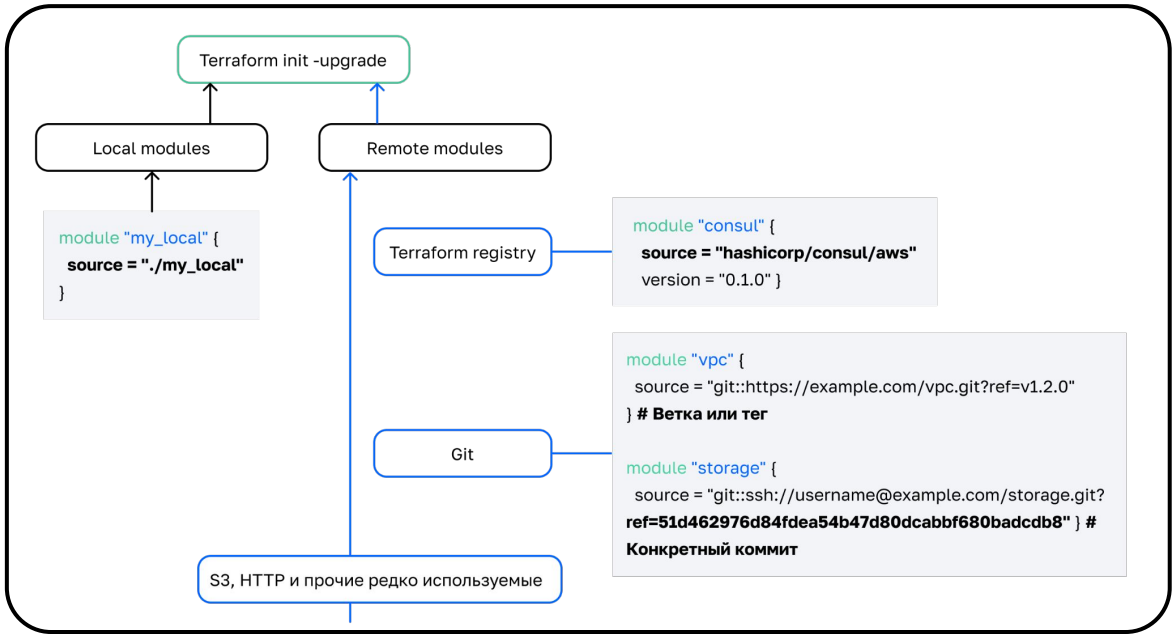
Схематичный пример использования модулей в проектах



Источники child modules

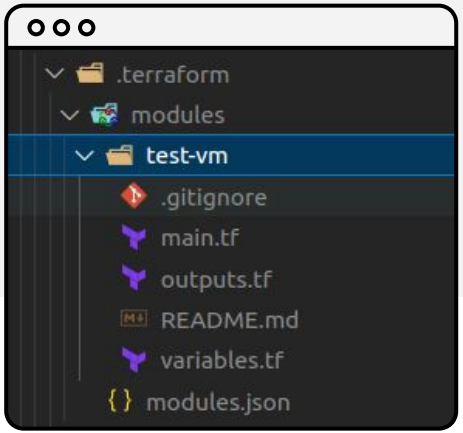
Upgrade нужен для **принудительного** обновления версий зависимостей.

Это может быть полезно, когда вы используете **Git-ветки** и хотите обновить зависимости до последней версии



Пример вызова remote child module из Git

```
module "test-vm" {
  source          = "git::https://github.com/udjin10/yandex_compute_instance.git?ref=main"
  env_name        = "develop"
  network_id      = yandex_vpc_network.develop.id
  subnet_zones    = ["ru-central1-a"]
  subnet_ids      = [ yandex_vpc_subnet.develop.id ]
  instance_name   = "web"
  instance_count  = 2
  image_family    = "ubuntu-2004-lts"
  public_ip       = true
  metadata        = {
    serial-port-enable = 1
    ssh-keys           = "ubuntu:${var.public_key}"
  }
}
```



Remote-модуль будет скачан и установлен в директорию
.terraform/modules/<label_вызываемого_модуля>

Процедура скачивания выполняется для каждого вызова модуля отдельно

Обращение к outputs дочерних модулей

module.<MODULE NAME>.<OUTPUT NAME>

#Root module: main.tf

```
module.web_vms {
  source = ./
  instance_count = 2
  name = "develop"
}
```

Input vars

```
#Child module: maint.tf
resource "yandex_compute_instance" "vm" {
  count = var.instance_count
  name = "${var.instance_name}-${count.index}"
}

variable "instance_count" {
  type = number
  default = 1 # Переменные в child module могут иметь дефолтные значения
}
variable "instance_name" {
  type = string # или требовать их указания при вызове
}
```

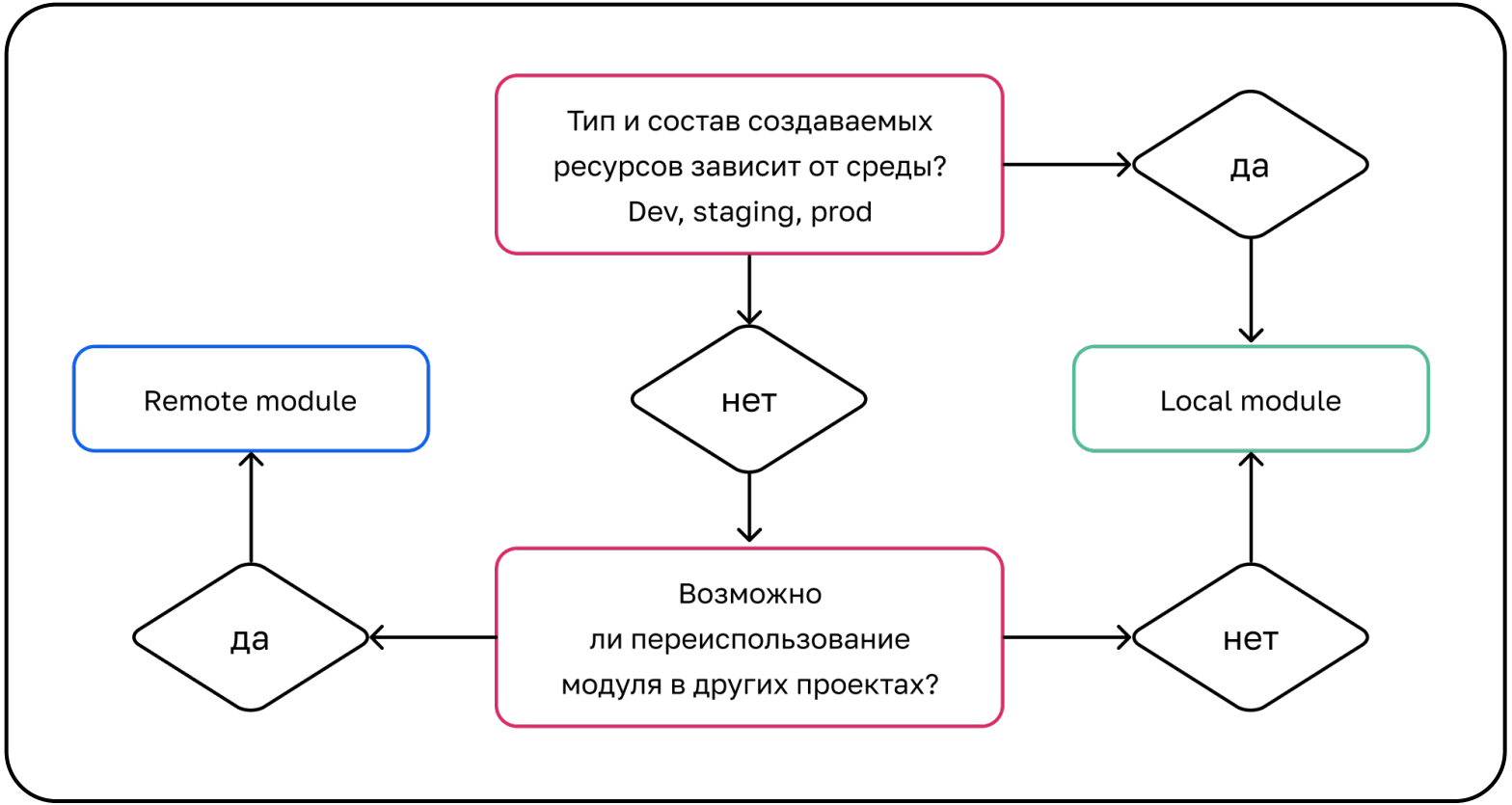
```
#Child module: outputs.tf
output "fqdn" {
  description = "The fully qualified DNS name of this instance"
  value       = yandex_compute_instance.vm.*.fqdn }
```

Output vars

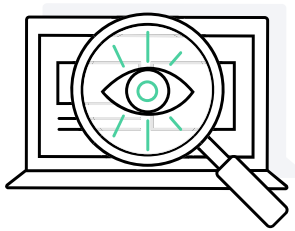
Outputs

```
module.web_vms.fqdn
[
  "develop-0.ru-central1.internal",
  "develop-1.ru-central1.internal", ]
```

Local- или remote-модуль?

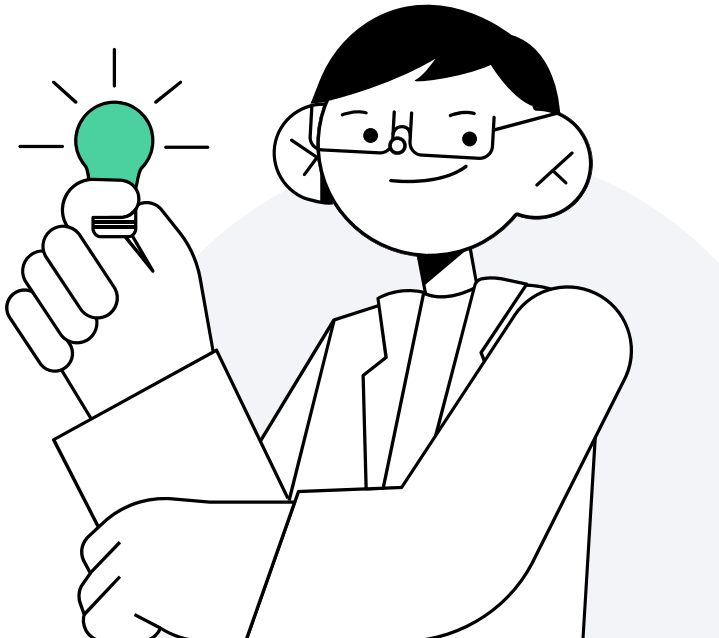


Демонстрация работы 1



Цели демонстрации

- Подключение стороннего модуля
- Разбор использованного модуля, переменных (обязательные и опциональные)
- Использование child module outputs в root module



Управление state

2

Отображение содержимого state

Terraform state list

Используется для отображения **списка** ресурсов, которые находятся в Terraform state

```
terraform state list
yandex_vpc_network.develop
yandex_vpc_security_group.example
yandex_vpc_subnet.develop
module.web_vms.data.yandex_compute_image.my_image
module.web_vms.yandex_compute_instance.vm[0]
module.web_vms.yandex_compute_instance.vm[1]
```

Terraform state show '<resource>'

Используется для отображения **подробной** информации о ресурсе, находящемся в Terraform state

```
terraform state show 'yandex_vpc_network.develop'
# yandex_vpc_network.develop:
resource "yandex_vpc_network" "develop" {
    created_at = "2023-02-12T08:11:47Z"
    folder_id  = "b1gfu61oc15cb99nqmfe"
    id         = "enp1s4k00o2bol8erupc"
    labels     = {}
    name       = "develop"
    subnet_ids = [ "e9bj7ioe4me98lpehj8t", ]
}
```

Переименование элементов state

Предположим, вы создали ресурс, но назвали его, не соблюдая принятый в команде **name convention** (соглашение об именовании).

Можно, конечно, удалить его, а затем создать заново с правильным именем.

Но это не путь DevOps-джедая ;)

```
resource "yandex_vpc_subnet" "abc123" { .. }
```

```
resource "yandex_vpc_subnet" "develop" { .. }
```

Переименование элементов state

1. Переименуйте ресурс в коде проекта
2. Для переименования ресурса внутри state используйте команду **terraform state mv <old_name> <new_name>**

```
terraform state mv  yandex_vpc_subnet.abc123   yandex_vpc_subnet.develop
Move "yandex_vpc_subnet.abc123" to "yandex_vpc_subnet.develop"
Successfully moved 1 object(s).
```

Пересоздание ресурсов. Точечное применение

В процессе эксплуатации ресурсы могут прийти в нежелательное состояние. При этом оно будет соответствовать заявленному Terraform-коду.

Пересоздание может быть более эффективным решением, чем попытка восстановления.

Точечно удалить **указанный** ресурс:

```
terraform destroy -target="module.web_vms.yandex_compute_instance.vm[0]"
```

Точечно пересоздать **указанный** ресурс:

```
terraform apply -target="module.web_vms.yandex_compute_instance.vm[0]"
```

То же самое, но без предварительного удаления:

```
terraform apply -replace="module.web_vms.yandex_compute_instance.vm[0]" \
-target="module.web_vms.yandex_compute_instance.vm[0]"
```

Жизненные примеры использования — replace

- Заменить значение пароля в ресурсе **random_password**
- Пересоздать **испорченный** backend в группе балансировки
- Yandex provider игнорирует изменение дистрибутива ОС для уже созданных ВМ. Replace поможет заменить ОС виртуальной машины, например с Ubuntu на Debian
- Обновление внешних зависимостей, например библиотек, которые устанавливаются только при первичной инициализации ресурса

Удаление и импорт ресурсов state

Если вы удаляете код, который управляет ресурсами в Terraform, **программа попытается удалить соответствующие ресурсы в облачной инфраструктуре.**

Это может быть нежелательно, если вы переносите ресурсы в другой проект или хотите сохранить данные ресурсов для последующего использования.

```
terraform state rm '<resource>'
```

Удаляет ресурс или даже целый модуль из state, **но не удаляет** соответствующие ресурсы в облачной инфраструктуре

```
terraform state rm 'yandex_vpc_network.develop'
Removed yandex_vpc_network.develop
Successfully removed 1 resource instance(s)
```

Удаление и импорт ресурсов state

```
terraform import  '<resource>' 'id'
```

При импорте ресурса Terraform сначала сравнивает его конфигурацию с той, что определена в коде, а затем импортирует его в state, если они соответствуют друг другу, **как если бы он был изначально создан Terraform.**

Необходимо указать **идентификатор ресурса.** Подробности о том, каким образом это сделать, обычно есть в документации соответствующего провайдера

```
terraform  import 'yandex_vpc_network.develop' enp1s4k00o2bol8erupc
yandex_vpc_network.develop: Importing from ID "enp1s4k00o2bol8erupc"...
yandex_vpc_network.develop: Import prepared!
  Prepared yandex_vpc_network for import
yandex_vpc_network.develop: Refreshing state... [id=enp1s4k00o2bol8erupc]
Import successful!
```

Команда **terraform import** может помочь восстановить утерянный или испорченный state Terraform.

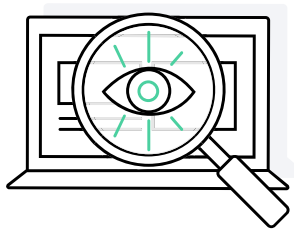
Однако процедура импорта может быть **очень кропотливой в больших проектах:** возможно, потребуется импортировать множество ресурсов, у каждого из которых есть свои многократно вложенные зависимости.

Поэтому всегда делайте бэкап state по расписанию. Дополнительно можно использовать версионирование файлов S3 для хранения истории изменений

Демонстрация работы 2

На примере первой демонстрации отработать CLI команды state.

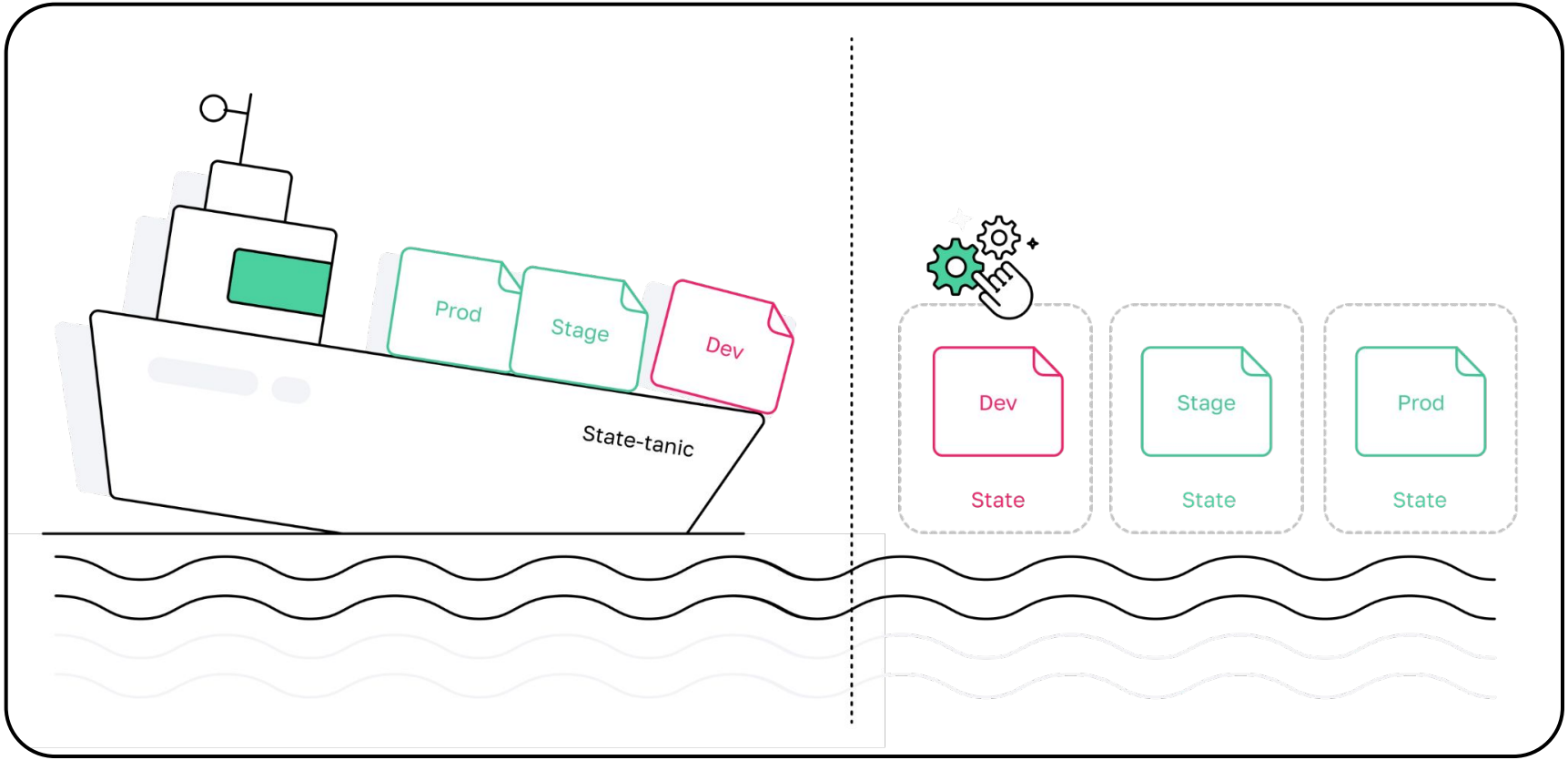
Код для демонстрации



Изолирование state

3

Не совмещайте разные окружения в одном state



Методы изоляции. Workspaces

Достоинства:

- один репозиторий на все окружения проекта
- более гибкий код

Недостатки:

- state-файл окружений изолирован логически, но не физически
- повышает сложность кода
- человеческий фактор: можно забыть, в каком workspace работали
- длительный план выполнения в больших проектах — от 30 минут до зависания

Не рекомендуем использовать этот метод, но знать о нём нужно

CLI команды для работы с workspaces

```
#Объявление workspace
terraform {
  workspace "dev" {
  #
  }

  workspace "prod" {
  #
  }
}

#Default ws переменные
variable "HA" {
  default = {
    dev   = false
    prod  = true
  }
}
```

```
$ terraform workspace list #Workspace по умолчанию
* default

$ terraform workspace new dev # Добавление workspace
Created and switched to workspace "dev"!

$ terraform workspace list   # Список workspaces
  default
* dev

$ terraform workspace select default # Выбор workspace
Switched to workspace "default".

$ terraform workspace delete dev  # Удаление workspace
Deleted workspace "dev"!
```

```
high_availability = (terraform.workspace == "prod") ? true : false
#Условные выражения с workspace
```

«Коммунальные» root-модули

```
├ dev
│  ├── main.tf [module.vpc, module.compute, module.db, module.k8s,...]
│  ├── outputs.tf
│  ├── terraform.tfstate
│  └── terraform.tfvars
└ prod
   ├── main.tf[module.vpc, module.compute, module.db, module.k8s,...]
   ├── outputs.tf
   ├── terraform.tfstate
   └── terraform.tfvars
```

«Коммунальные» root-модули

Достоинства:

- простой и интуитивно понятный метод
- state-файлы различных окружений отделены на физическом уровне

Недостатки:

- дублирование кода в root-модуле окружений усложняет поддержку кода
- длительный план выполнения в больших проектах — от 30 минут до зависания

Предпочтительный вариант для новичков и небольших проектов

Ускорение работы Terraform в большом проекте

Для ускорения работы можно **аккуратно** использовать:

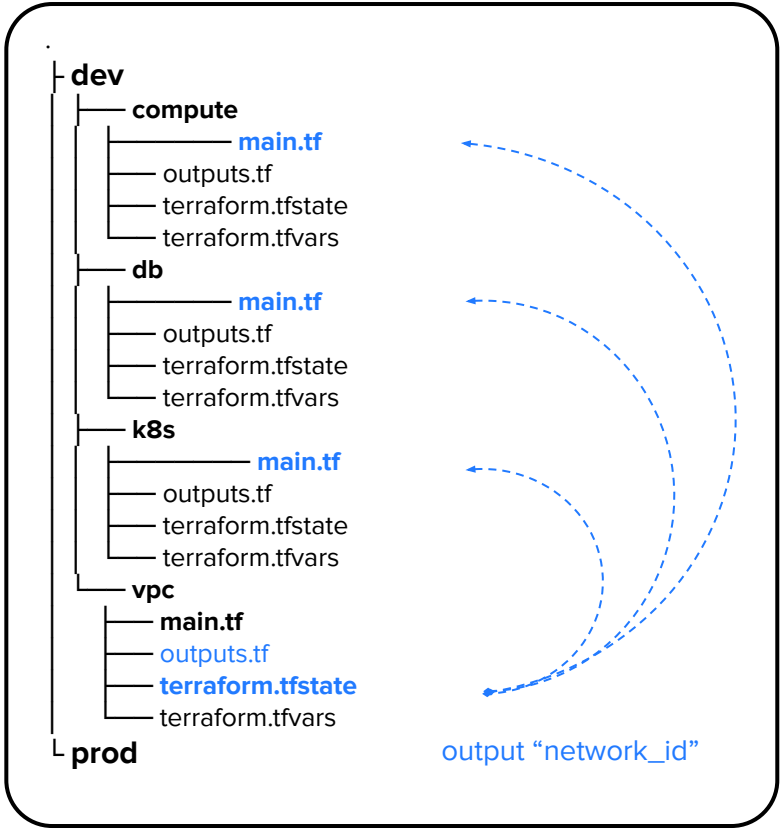
- terraform **apply -target=module.xxx** — для точечной работы с выбранным child-модулем
- terraform **apply -refresh=false** — отключает дефолтную синхронизацию state с реальными объектами. Применяйте, **когда уверены,** что state совпадает с инфраструктурой, иначе это может привести к неполному или неверному плану

Terraform предоставляет флаг **-parallelism,** который позволяет установить количество одновременно выполняемых операций. По умолчанию это значение равно **10**

Изоляция в отдельных root-модулях с помощью terraform_remote_state

Схожие наборы редко изменяемых общих ресурсов выделяются из «коммунального» root-модуля в отдельные root-модули: VPC, аккаунты и т. д.

Зависимые переменные считываются из outputs в **state** иных **root-**модулей с помощью **provider terraform_remote_state**



Изоляция в отдельных ресурсных модулях с помощью terraform_remote_state

Достоинства:

- state-файл окружений отделён на физическом уровне
- ускоряется выполнение отдельных модулей

Недостатки:

- Выделенные модули требуют предварительного исполнения своих зависимостей
- Не существует встроенного способа иерархически запустить на выполнение все модули разом. Это может сделать отдельное ПО

Provider terraform_remote_state

Для чтения данных
из **локального** state:

```
data "terraform_remote_state" "vpc" {
  backend = "local"
  config = {
    path = "../vpc/terraform.tfstate"
  }
}
```

Для чтения из **remote** state
в S3-bucket:

Про **backend** и настройку
удалённого хранения state
поговорим в следующей лекции

```
data "terraform_remote_state" "vpc" {
  backend          = "s3"
  config {
    bucket         = "tfstate"
    key            = "prod/terraform.tfstate"
    endpoint       = "storage.yandexcloud.net"
    access_key     = "..."
    region         = "ru-central1"
   }
}
```

Обращение к переменной из terraform_remote_state

Пример:

```
data.terraform_remote_state.vpc.outputs.subnet_id
```

Полезные утилиты и инструменты

4

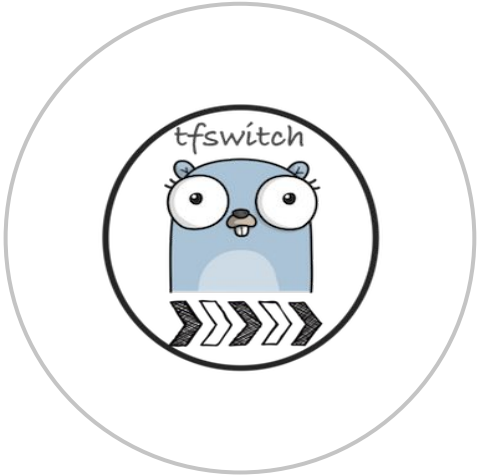
Terraform-switcher

Инструмент командной строки позволяет легко переключаться между различными версиями Terraform.

Может автоматически установить Terraform требуемой версии, если есть доступ к terraform.io.

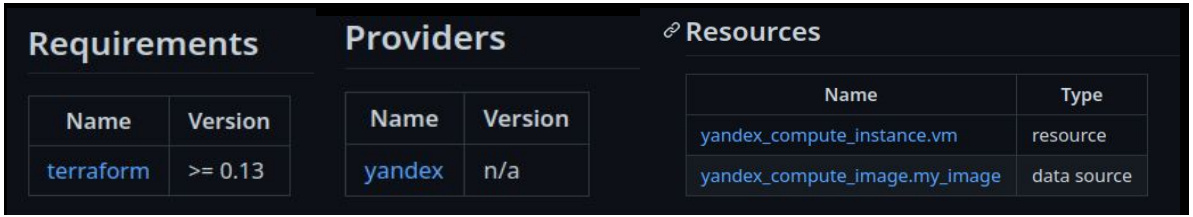
Вы можете легко переключаться между уже установленными версиями Terraform, используя команду tfswitch <version>

TFSwitch



Terraform-docs

Код Terraform — это уже неплохая документация. Но будет лучше, если все модули будут иметь сопровождающее описание. В этом вам **частично** поможет terraform-docs. Утилита автоматически генерирует описание ресурсов, переменных и зависимостей



Requirements

Name	Version
terraform	>= 0.13

Providers

Name	Version
yandex	n/a

Resources

Name	Type
yandex_compute_instance.vm	resource
yandex_compute_image.my_image	data source

```
docker run --rm --volume "$(pwd):/terraform-docs" -u $(id -u)\
quay.io/terraform-docs/terraform-docs:0.16.0 markdown /terraform-docs
```

terraform docs

Cloud-init

Предустановленный инструмент для современных дистрибутивов Linux.

Представляет собой YAML-манифест, который во время загрузки ОС производит её первичную настройку.

Может заменить Ansible в простых конфигурациях.

С помощью Terraform можно передать конфигурацию cloud-init в ресурс ВМ аргументом **user-data**

```
#Пример cloud-init.yml
users:
  - name: ubuntu
    groups: sudo
    shell: /bin/bash
    sudo: ['ALL=(ALL) NOPASSWD:ALL']
    ssh-authorized-keys:
      - ssh-ed25519 AAAAB.....
      - ssh-ed25519 AAAAC.....
package_update: true
packages_upgrade: true
packages:
  - vim
runcmd:
  - ufw allow 22
  - echo "y" | ufw enable

#Пример передачи cloud-config в ВМ
data "template_file" "cloudinit" {
 template = file("./cloud-init.yml")
}
resource "yandex_compute_instance" "vm" {
  metadata={
    user-data=data.template_file.cloudinit
}}
```

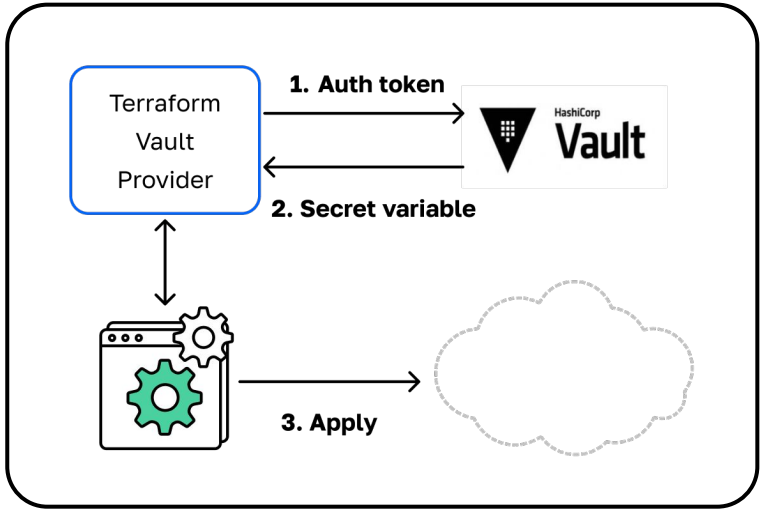
Хранение секретов в HashiCorp Vault

Provider Terraform для Vault позволяет хранить секретные данные в защищённом хранилище, используя версионирование секретов.

Безопасной настройке и полноценному использованию **HashiCorp Vault** можно посвятить отдельный учебный курс, но мы рассмотрим **основы.**

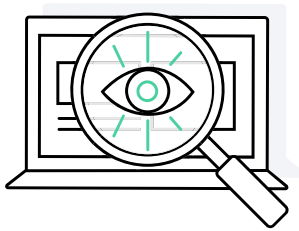
Vault можно запустить в **developer-mode,** который не требует сложной production-настройки, но позволит полностью освоить интеграцию **terraform<=>vault.**

- Docker-образ
- Документация для Vault
- Документация для Terraform Vault Provider



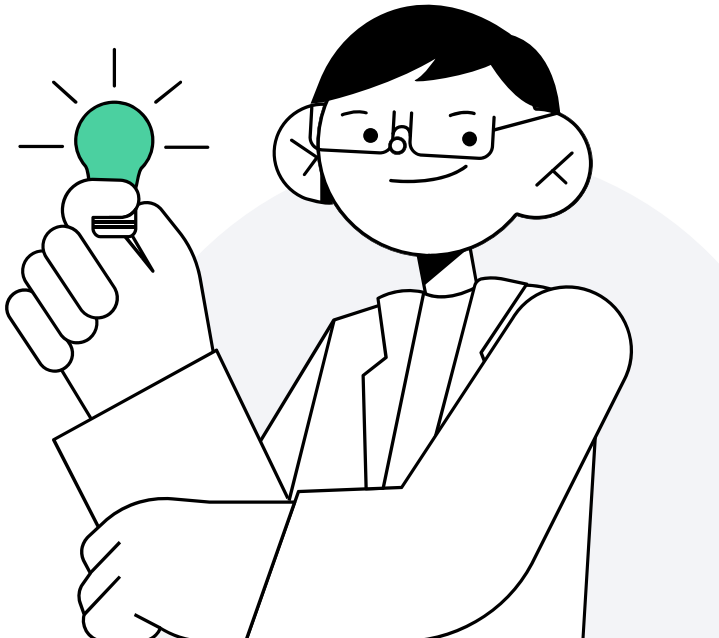

```
#Пример считывания секрета из Vault
data "vault_kv_secret_v2" "tls_example_ru" {
 mount = "tls_certs"
 name  = "wildcard.example.ru"
}
```

Демонстрация работы



На примере первой демонстрации рассмотрим:

- работу с cloud-init: пользователи, ПО, shell-команды
- использование terraform-docs
- развёртывание Vault в docker-compose
- чтение секретов из Vault



Итоги занятия

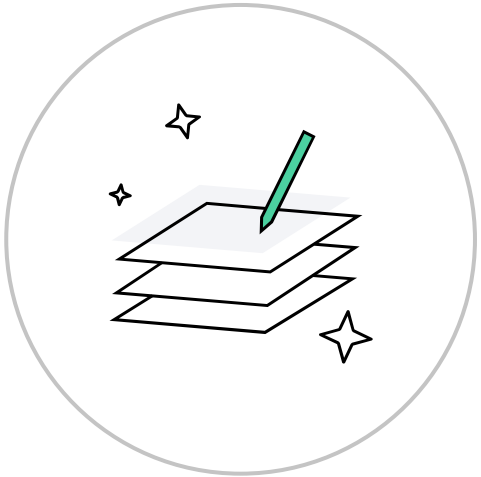
Сегодня мы:

- Рассмотрели модули и то, как их можно применять для переиспользования кода
- Узнали, как вручную управлять state
- Разобрались с организацией кода окружений
- Рассмотрели некоторые инструменты, которые могут быть полезны при работе с Terraform

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 модулей
- Документация state

Задавайте вопросы и пишите отзыв о лекции