Продвинутые методы работы с Terraform

Елисей Ильин
DevOps-инженер в Itransition

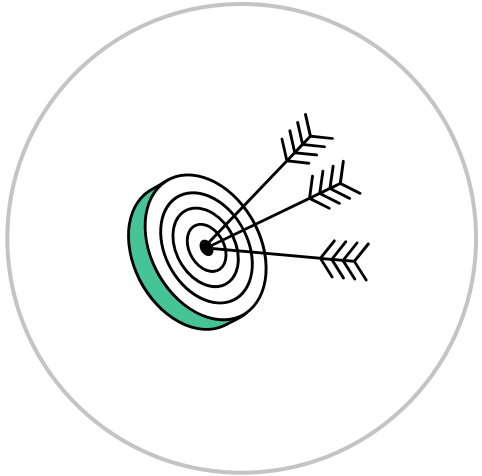
Елисей Ильин

О спикере:

- DevOps-инженер
- Опыт работы в IT 6 лет

Цели занятия

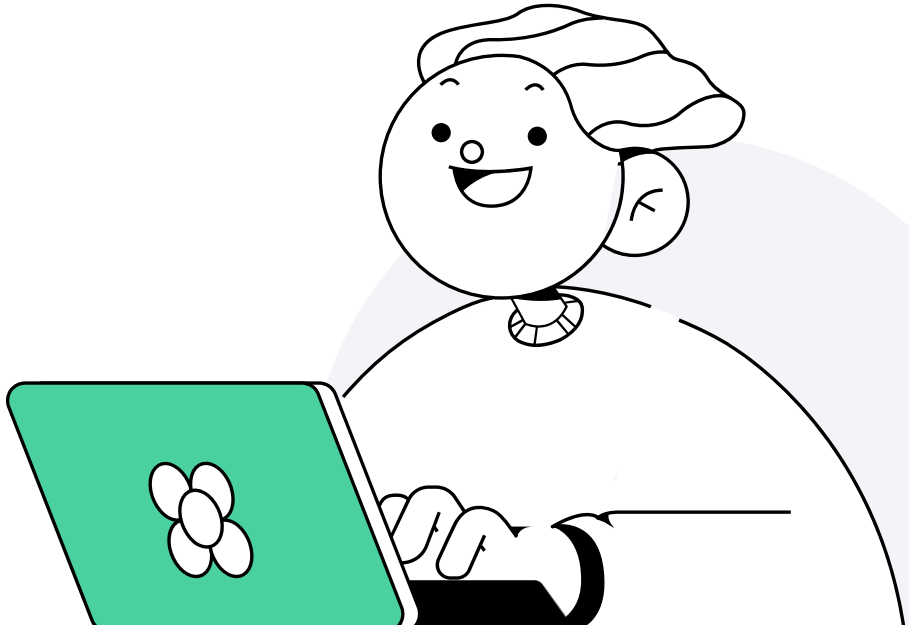
- Научиться переиспользовать terraform-код
- Детально изучить все, что связано с terraform State



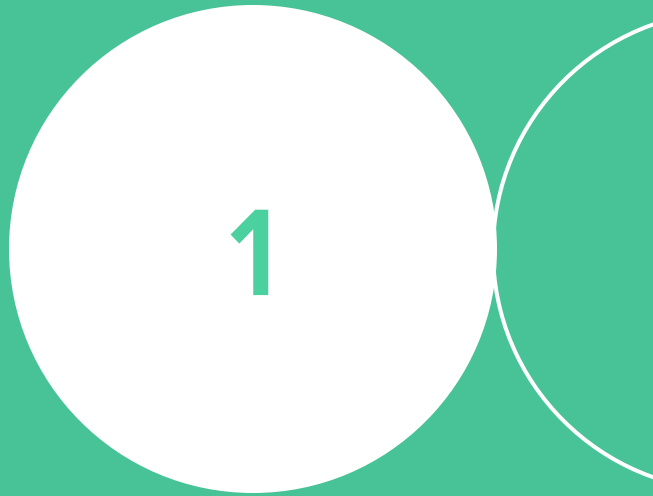
План занятия

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

Нажмите на раздел для перехода



Child modules



Child module — это отдельная, несамостоятельная конфигурация, которую можно **вызвать** в родительском модуле

Child modules

Все экземпляры ресурсов, имена и переменные **изолированы** в области действия модуля.

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

Это позволяет переиспользовать уже написанный и протестированный opensource комьюнити или вашей командой terraform-код, реализует принцип **DRY(Don't Repeat Yourself)**.

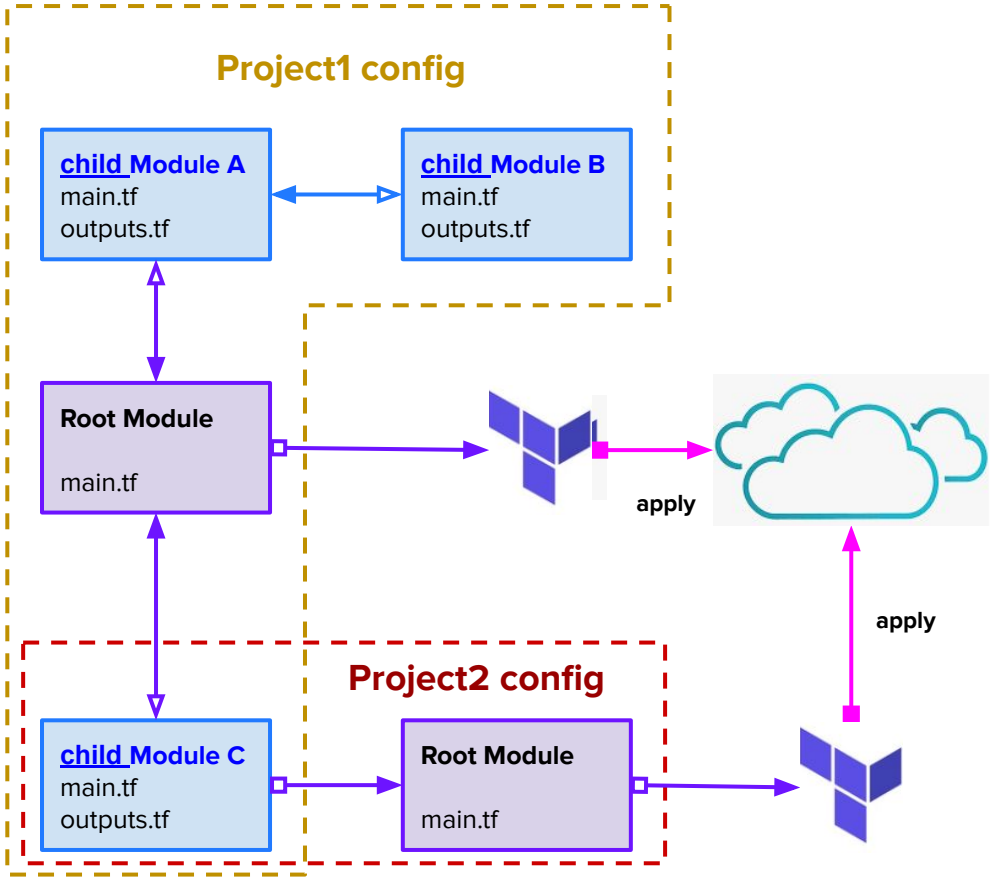
Для вызова используется блок **module {..}.**

Обязательно — указывать мета-аргумент **source.**

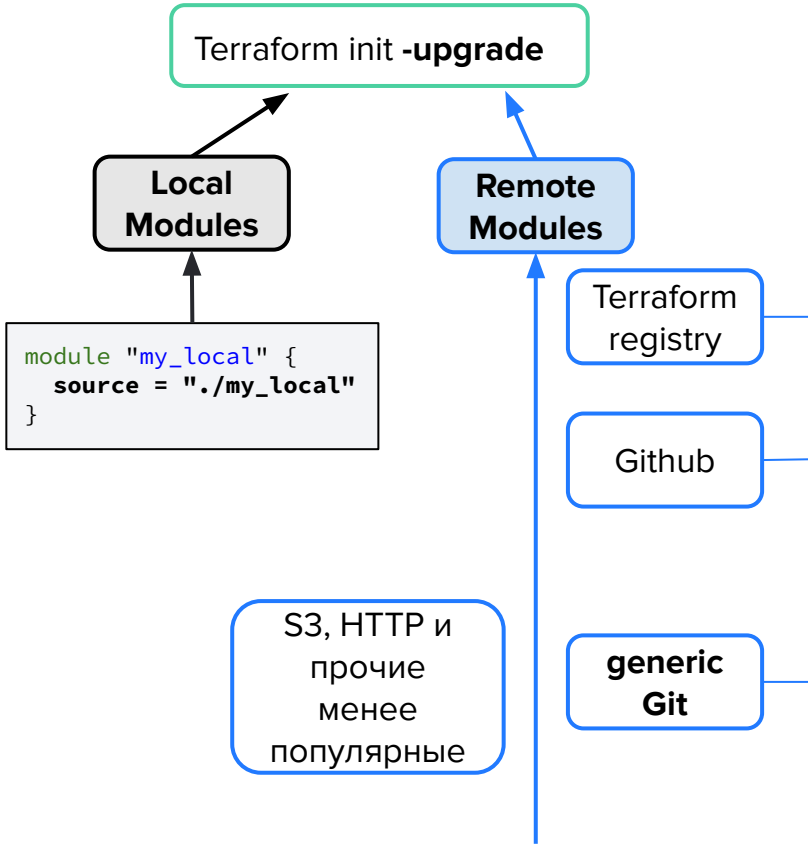
Желательно — указывать входящие аргументы/

```
module "vpc" {
    source         = "./vpc"
    env_name       = var.env_name
    nat_address    = var.nat_address
    subnet_addrs   = var.subnet_addrs
 }
```

Схематичный пример использования модулей в проектах



Источники child modules



upgrade нужен для **принудительного** обновления версий зависимостей. Полезно при использовании **git-веток**

```
module "consul" {
  source = "hashicorp/consul/aws"
  version = "0.1.0" }
```

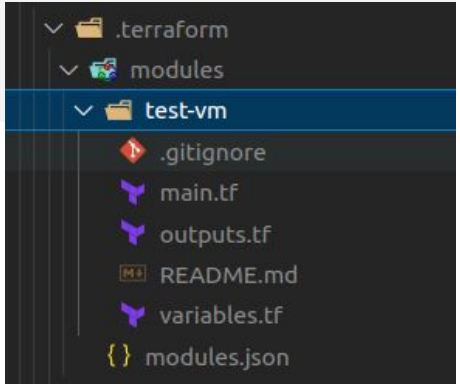
```
module "consul" {
  source = "git@github.com:hashicorp/example.git"}
```

```
module "vpc" {
  source = "git::https://example.com/vpc.git?ref=v1.2.0"
} # ветка или тег

module "storage" {
  source =
"git::ssh://username@example.com/storage.git?ref=51d462976d84f
dea54b47d80dcabbf680badcdb8" } # конкретный комит
```

Пример вызова child module из generic 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}"
  }
}
```



Модуль будет скачен и установлен в директорию
.terraform/modules

Обращение к outputs дочерних модулей

module.<MODULE NAME>.<OUTPUT NAME>

#Root module: main.tf

```
module.web_vms {
  source = ./
  instance_count = 2
}
```

input vars

#Child module: maint.tf

```
resource "yandex_compute_instance" "vm" {
  count = var.instance_count
  .....
}
variable "instance_count" { #требуемая переменная
  type = number
}
```

#Child module: outputs.tf

```
output "external_ip" {
  description = "The external IP address of the instance"
  value=
yandex_compute_instance.vm.*.network_interface.0.nat_ip_address
}

output "fqdn" {
  description = "The fully qualified DNS name of this instance"
  value       = yandex_compute_instance.this.*.fqdn }
```

output vars

Outputs

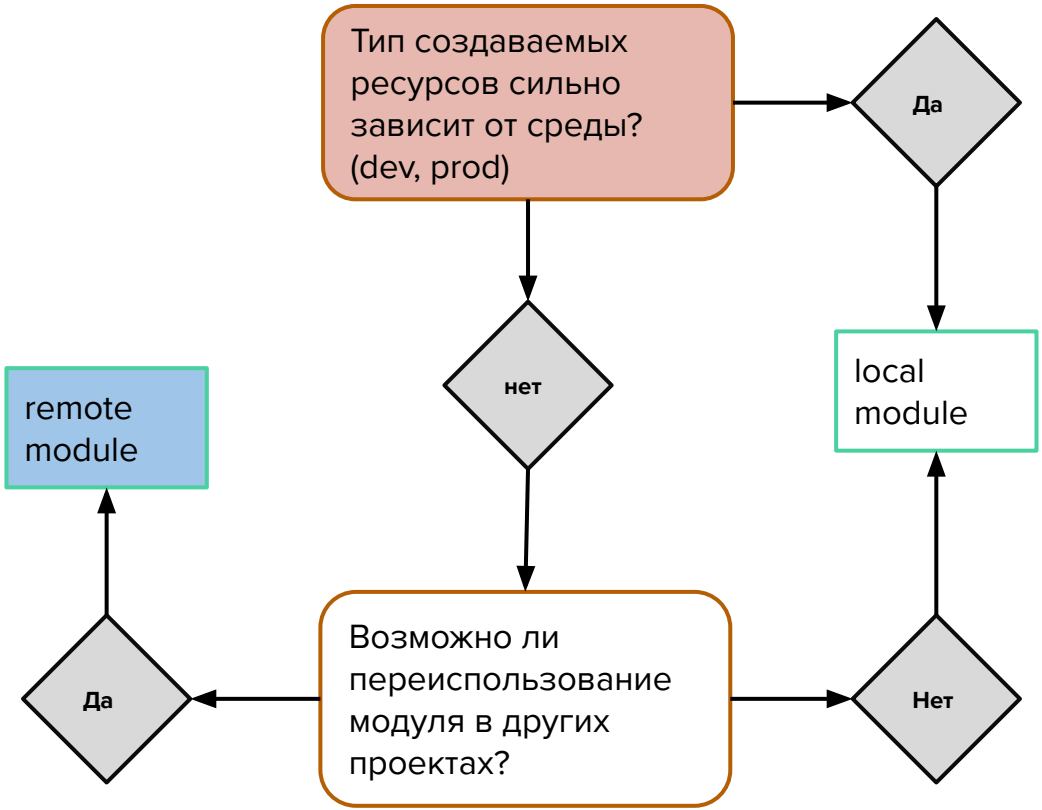
module.web_vms.external_ip_address

```
[
  "51.250.7.66",
  "51.250.87.185",
]
```

module.web_vms.fqdn

```
[
  "develop-web-0.ru-central1.internal",
  "develop-web-1.ru-central1.internal", ]
```

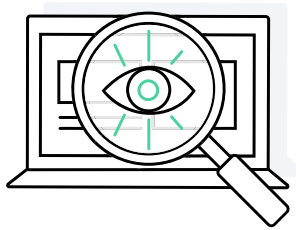
local или remote модуль?



Демонстрация работы №1

- Подключение стороннего модуля (ссылка на демо)
- Разбор использованного модуля, переменных (обязательные и опциональные) (ссылка)
- Использование child module outputs в Root Module

Стенд не разбирать — используется в демо №2 и №3



Управление State

2

Отображение содержимого State

terraform state list
Отображает список ресурсов в 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>'
Отображает параметры конкретного ресурса

```
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 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 соответствует заявленному terraform-коду.
В этом случае не стоит *чинить* ресурс. Пересоздайте его!

Точечно удалить ресурс:

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

Точечно создать ресурс:

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

То же самое без лишних манипуляций:

```
terraform apply -replace="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.

Удаление и импорт ресурсов State

Бывают нестандартные ситуации. Например, вы решили перенести ресурс в иной проект.
Если вы удалите код из текущего проекта — **Terraform запланирует удаление реальных ресурсов.**

terraform state rm '<resource>'
Удаляет ресурс или даже целый модуль из State. **Не удаляет** реальные ресурсы

```
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'
Сравнивает параметры существующего ресурса с terraform-кодом. В случае их совпадения импортирует этот ресурс в State, как если бы он был изначально создан terraform.

Необходимо указать **идентификатор ресурса**. Подробности в документации соответствующего провайдера

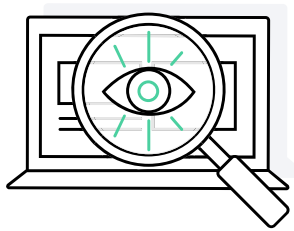
```
terraform  import 'yandex_vpc_network.develop' enp1s4k00o2bol8erupc
yandex_vpc_network.develop: Importing from ID "enp1s4k00o2bol8erupc"...
yandex_vpc_network.develop: Import prepared!
  Prepared yandex_vpc_network for import
yandex_vpc_network.develop: Refreshing state...
[id=enp1s4k00o2bol8erupc]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

С помощью import можно **воскресить утерянный или испорченный State**. Но лучше делайте **бекап** т.к. процедура в больших проектах крайне кропотливая

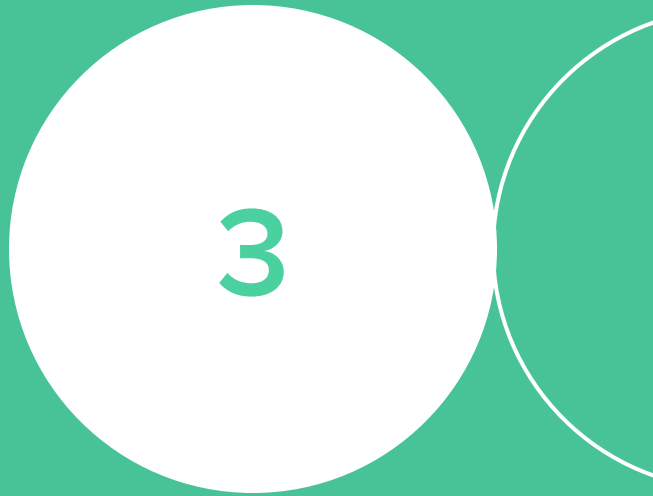
Демонстрация работы №2

На примере первой демонстрации отработать cli команды state.

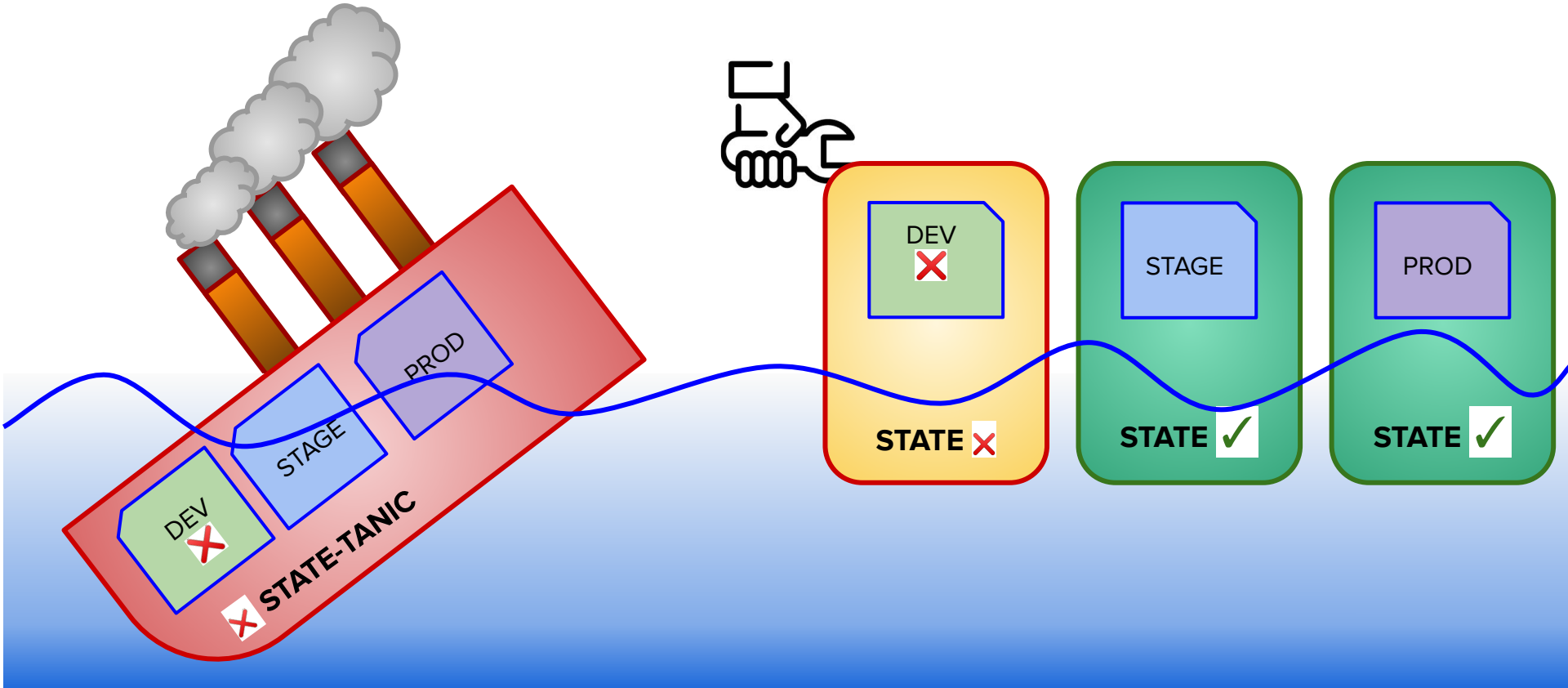
Ссылка на код для демонстрации



Изолирование State



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 (забыл, в каком WS работаешь)
- длительный план выполнения в крупных проектах — от 30 минут вплоть до зависания.

Не рекомендуем использовать этот метод, но знать о нем нужно!

CLI команды для работы с workspaces

```
#объявление ws
terraform {
  workspace "dev" {
  #
  }

  workspace "prod" {
  #
  }
}

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

```
$ terraform workspace list #ws по-умолчанию
* default

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

$ terraform workspace list   # список ws
  default
* dev

$ terraform workspace select default # выбор ws
Switched to workspace "default".

$ terraform workspace delete dev  # удаление ws
Deleted workspace "dev"!
```

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

Отдельные root модули для окружений

```
├ 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
```

Отдельные root модули для окружений

Достоинства:

- простой и интуитивно понятный метод
- State файл окружений отделен на физическом уровне

Недостатки:

- дублирование кода в Root Module разных окружений
- длительный план выполнения в крупных проектах — от 30 минут вплоть до зависания

Ускорение работы крупного проекта

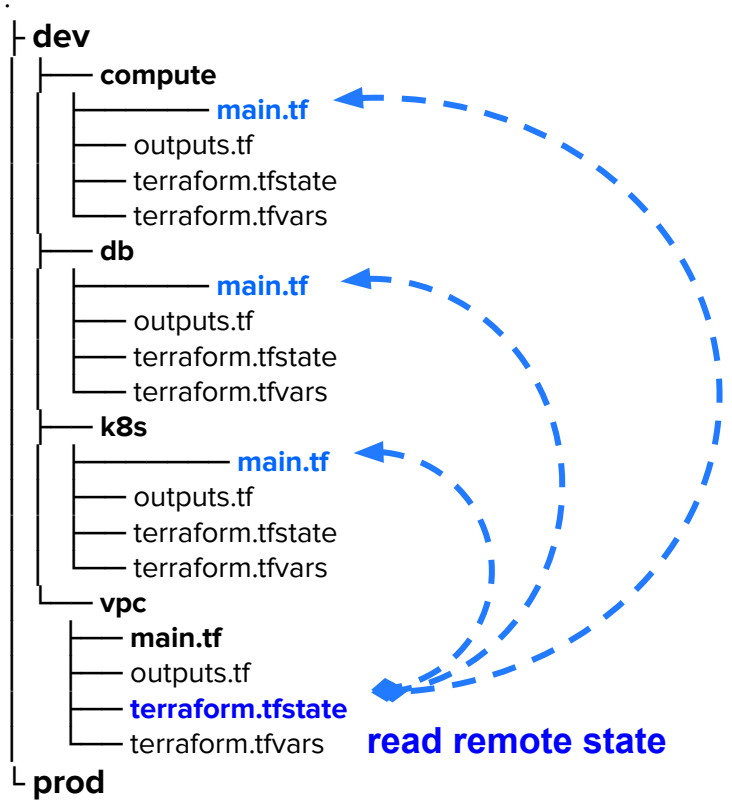
Для ускорения работы можно **очень аккуратно** использовать:

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

Изоляция в отдельных, ресурсных модулях, используя terraform_remote_state

Схожие наборы редко изменяемых, общих ресурсов выделяются из «коммунального root-модуля» в отдельные root-модули.
(vpc, аккаунты...)

Зависимые переменные считываются из State соседних модулей с помощью provider **terraform_remote_state**



Изоляция в отдельных, ресурсных модулях, используя terraform_remote_state

Достоинства:

- State файл окружений отделен 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 **Terragrunt**

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:

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

Про **backend** и настройку удаленного хранения State поговорим в следующей лекции

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).

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

Inputs

Name	Description	Type	Default	Required
boot_disk_size	n/a	number	30	no
boot_disk_type	n/a	string	"network-hdd"	no

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**.

cloud-init

```
#Пример cloud-init.yml
users:
  - name: ubuntu
    groups: sudo
    shell: /bin/bash
    sudo: ['ALL=(ALL) NOPASSWD:ALL']
    ssh-authorized-keys: 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
}}
```

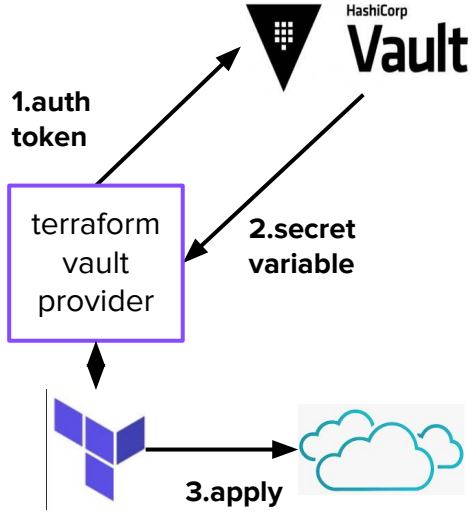
Хранение секретов в Hashicorp vault

Provider terraform для vault позволяет вам хранить секретные данные в защищенном хранилище.

Безопасной настройке и использованию **Hashicorp Vault** можно посвятить отдельный учебный курс.

Рекомендуем самостоятельно ознакомиться с основами, используя документацию и docker. **Vault** умеет работать в режиме developer-mode, который не требует сложной настройки, но позволит полностью освоить интеграцию terraform-vault.

- Docker-образ
- Документация для vault
- Документация для terraform vault provider

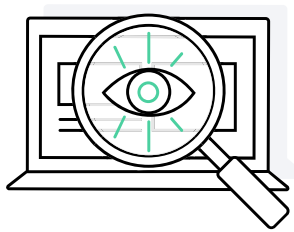



```
#Пример считывания секрета из vault
data "vault_kv_secret_v2" "tls_example_ru" {
 mount = "tls_certs"
 name  = "wildcard.example.ru"
}
```

Демонстрация работы

На примере первой демонстрации отработать:

- показать работу с cloud-init (пользователи, ПО, shell-команды)
- terraform docs
- tfswitch



Итоги занятия

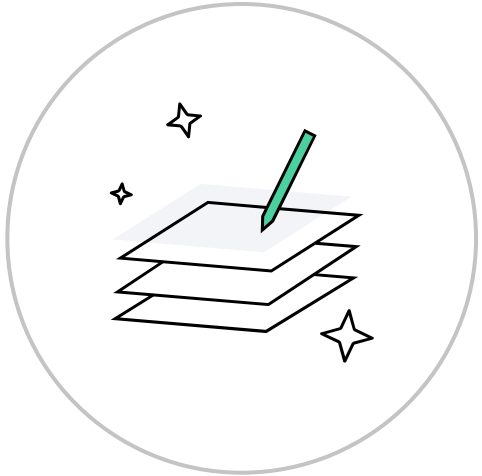
Сегодня мы

1. Узнали, как можно переиспользовать terraform код
2. Разобрались с управлением State в ручном режиме
3. Рассмотрели способы организации кода окружений
4. Узнали, какие инструменты могут помочь в 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 Модули
- Документация State

Задавайте вопросы и пишите отзыв о лекции

Елисей Ильин
DevOps-инженер в Itransition