

# Использование Terraform в команде

Евгений Мисяков  
SRE инженер в Нетологии



# Евгений Мисяков

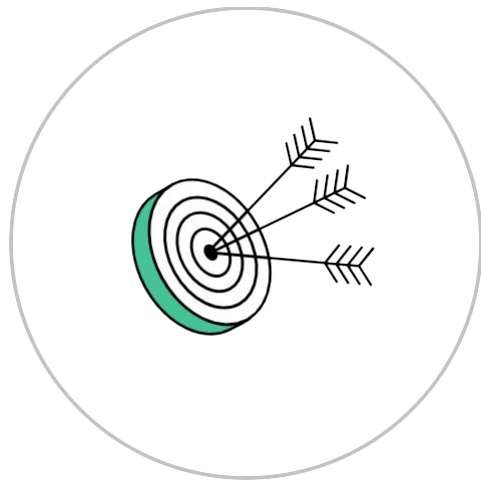
О спикере:

- SRE инженер в Нетологии



# Цели занятия

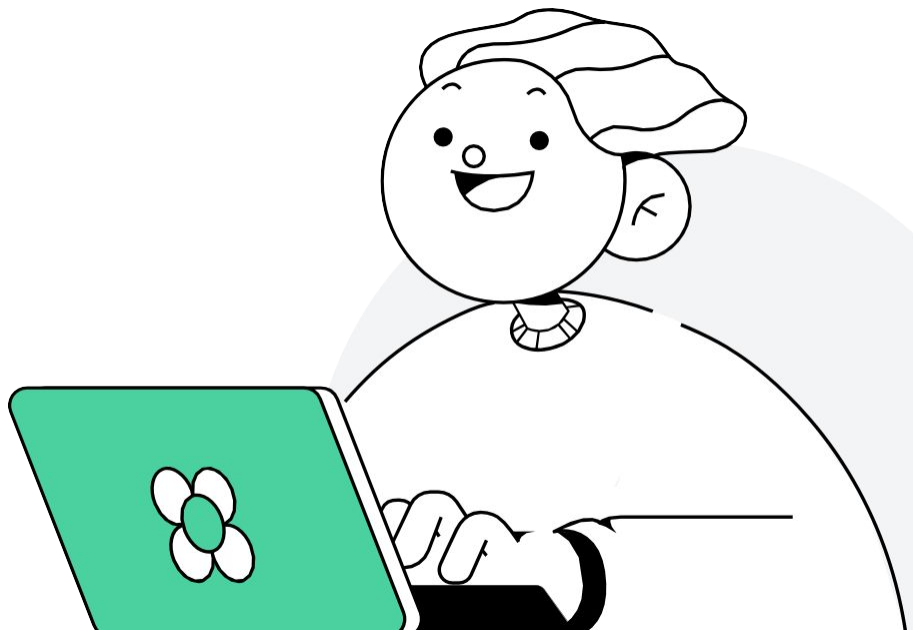
- Узнать, чем отличаются одиночная и командная разработка кода Terraform
- Научиться безопасно хранить и использовать remote state



# План занятия

- 1 Работа в команде
- 2 Name & code conventions
- 3 Terraform backend
- 4 Настройка backend S3 с блокировкой в YDB
- 5 Security scan
- 6 Code & plan review
- 7 Полезные практики

\*Нажми на нужный раздел для перехода

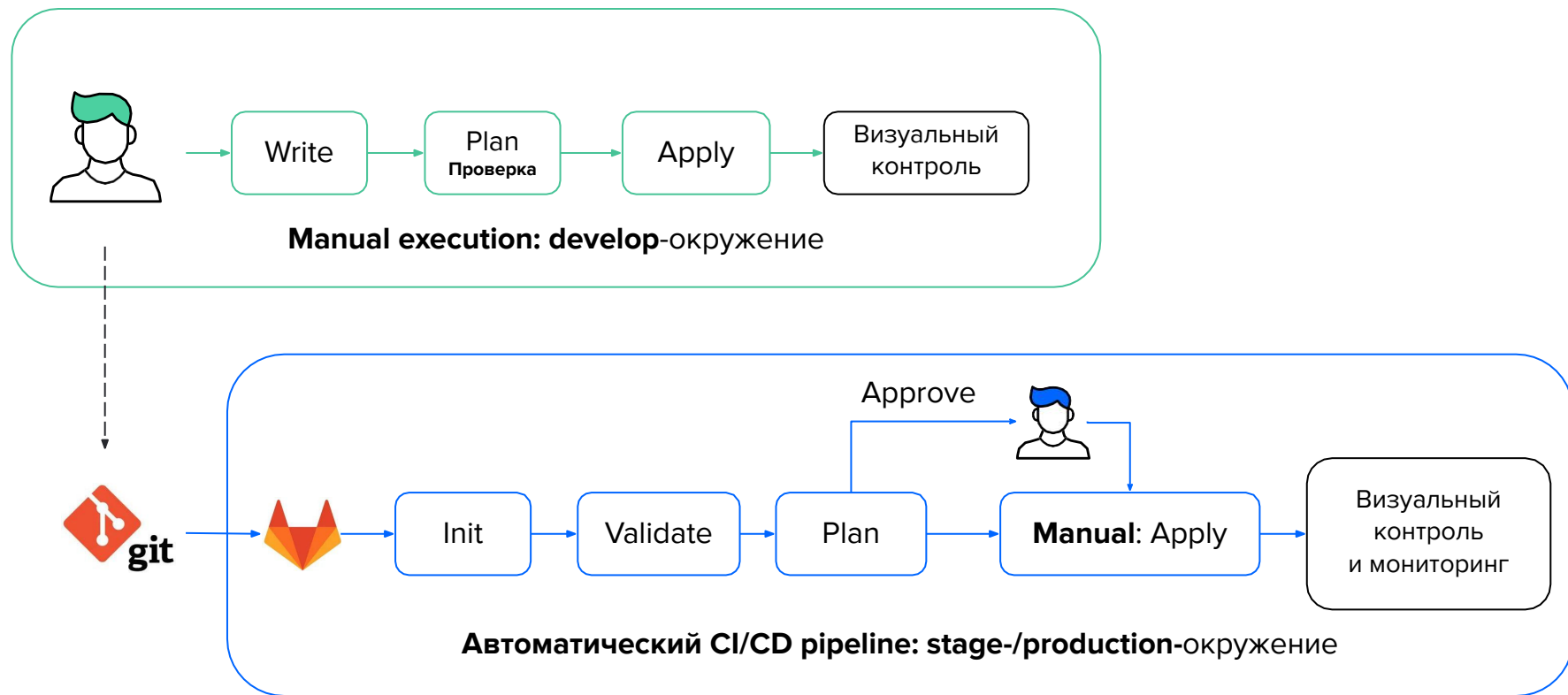


# Работа в команде



1

# Одиночная или small-team-разработка в Terraform



# Подводные камни Terraform при работе в команде

Проблема	Способы решения
Документация модулей и их переменных, окружений, костылей	Валидация переменных в модулях, Terraform-docs, контроль через PR
Name & code conventions	<a href="#">terraform-style-guide</a> ; <b>TFLINT</b>
Версионирование Terraform, плагинов, модулей	Привязка версий к пайплайнам CI/CD
Совместный доступ к state	Remote state
Повреждение state при одновременной записи	State locking
Code & plan review; CI/CD	GitLab; GitHub + any CI/CD
Security	Static code analysis <a href="#">checkov</a> , шифрование remote state

# Name & code conventions



2



# TFLint

Помогает найти в коде неиспользуемые переменные, незафиксированные версии, неопределённый тип переменных и прочее по [списку правил](#).

Также можно прикрепить дополнительные плагины по name convention.

[Релизы проекта](#)

```
Warning: variable "vm_db_name"  
is declared but not used  
(terraform_unused_declarations)  
  
on variables.tf line 51:  
51: variable "tflint_example"
```

**Вспомните, как во втором домашнем задании вы вручную искали неиспользуемые более переменные :)**



# TFLint: настройка

Конфигурация правил **TFLint**  
задаётся в файле `~/.tflint.hcl`  
или с помощью ключа `--config=<path>`.

Поддерживается подключение  
плагинов

```
config {  
  format = "compact"  
  plugin_dir = "~/.tflint.d/plugins"  
  module = true  
}  
plugin "terraform" {  
  enabled = true  
  
  preset      = "recommended"  
}
```



# TFLint: запуск

Проще всего запустить через Docker из каталога с кодом Terraform

```
docker run --rm -v "$(pwd):/tflint" ghcr.io/terraform-linters/tflint/tflint
```

# Блок валидации переменных

Если вы используете модули, написанные разнородной командой, важно не только предоставлять пример вызова модуля в документации к нему, но и валидировать переменные по типу и содержимому

```
variable "platform" {  
  type      = string  
  default   = "standard-v1"  
  description = "Example to validate VM platform."  
  validation {  
    condition = contains(["standard-v1", "standard-v2", "standard-v3"],  
      var.platform) error_message = "Invalid platform provided."  
  }  
}  
  
variable "api_token" {  
  type      = string  
  description = "API token"  
  validation {  
    condition = length(var.api_token) >= 32  
    error_message = "Must be at least 32 character long API token."  
  }  
}
```



# Блок валидации переменных

```
module "test-vm" {  
  ...  
  platform = "standart-v4"  
  ...  
}
```

**Error:** Invalid value for variable

on main.tf line 42, in module  
"test-vm":  
42: ~~platform = "standard-v4"~~  
| var.platform is "standard-v4"

Invalid platform provided.

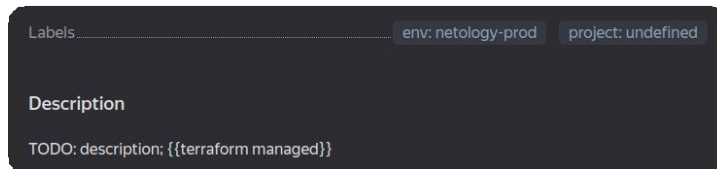
This was checked by the validation rule at  
.terraform/modules/test-vm/variables.tf:27,4-14.

# Лейблы ресурсов

Лейблы помогают выгружать финансовую статистику по проектам из облака.

Пример labels по умолчанию в модуле VM, если они не были переданы при вызове модуля:

```
locals {  
  labels = length(keys(var.labels)) > 0 ?  
  var.labels: {  
    "env"    = var.env_name  
    "project" = "undefined"  
  }  
}
```

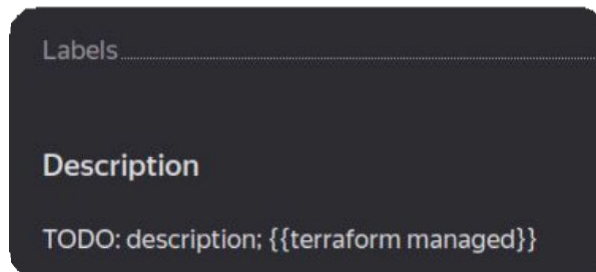


# Описание ресурсов

Описание может пригодиться другим сотрудникам — финансистам, бухгалтерам, сотрудникам техподдержки.

Дефолтное описание можно использовать для недавно созданных ресурсов на этапе разработки. Сразу видно, что его требуется доработать перед промышленным внедрением

```
variable "description" {  
  type = string  
  default = "TODO: description;  
  {{terraform managed}}"  
}
```

A screenshot of a code editor showing a Terraform variable definition. The variable is named "description" and is of type string. The default value is "TODO: description; {{terraform managed}}". The code is displayed in a dark-themed editor with syntax highlighting.

Labels \_\_\_\_\_

Description

TODO: description; {{terraform managed}}

# Terraform backend



3





**Terraform backend** — это метод хранения state и доступа к нему. По типам делится на **local** и **remote**. Некоторые виды backend поддерживают state locking



**State locking** — отметка о монопольном доступе к state. Запрещает операции записи в state для иных источников. Исключает повреждение state

# Сравнение популярных видов backend

Вид	Название backend	Поддержка блокировки
Local	backend "local"	Нет
Generic S3	backend "s3"	Нет
Amazon S3 bucket	backend "s3"	Да, хранение в DynamoDB
Yandex Object Storage	backend "s3"	Да, хранение в YDB (аналог DynamoDB)
Google Cloud Storage	backend "gcs"	Да, хранение в файле .tflock прямо в S3
HashiCorp Consul	backend "consul"	Да, интегрирована
Postgres DB	backend "pg"	Да, на уровне блокировок БД
HTTP (например, GitLab)	backend "http"	Да, 200: OK , 423: Locked or 409: Conflict

# Local backend

- Backend по умолчанию. Настройка опциональна, но особого смысла не имеет
- State сохраняется в root module
- Переопределить путь к state можно с помощью блока **backend "local" {..}**

```
terraform {  
  backend "local" {  
    path = "<относительный путь>/terraform.tfstate"  
  }  
}
```

# Remote s3 backend

Блок backend "s3" {..}

```
terraform {  
  backend "s3" {  
    bucket = "bucket_name"  
    key     = "path/terraform.tfstate"  
    region = "us-east-1"  
    access_key = "..." #Только для примера. Секретные данные нельзя хардкодить  
    secret_key = "..." #Только для примера. Секретные данные нельзя хардкодить  
    dynamodb_table = "tfstate-lock" #Таблица блокировок  
    encrypt = true #Шифрование state сервером Terraform  
  }  
}
```

# Backend Security

Использовать переменные в **backend** нельзя, но и хардкодить секреты недопустимо

#Недопустимый пример

```
backend "s3" {  
  ...  
  access_key = var.s3_access_key  
  secret_key = var.s3_secret_key  
  ...  
}
```

**Error: Variables not allowed**

```
on providers.tf lines 13,14 , in  
terraform:    access_key = var.s3_access_key  
13:          secret_key = var.s3_secret_key  
14:
```

**Variables may not be used here.**



# Backend Security

**Вместо этого** нужно инициализировать Terraform командой:

```
terraform init -backend-config="access_key=<s3_access_key>" -backend-config="secret_key=<s3_secret_key>"
```

Ключи доступа к backend будут сохранены в открытом виде в локальном файле **.terraform/terraform.tfstate**

# Миграция local backend -> backend S3

```
terraform init -backend-config="access_key=<s3_access_key>" -backend-config="secret_key=<s3_secret_key>"
```

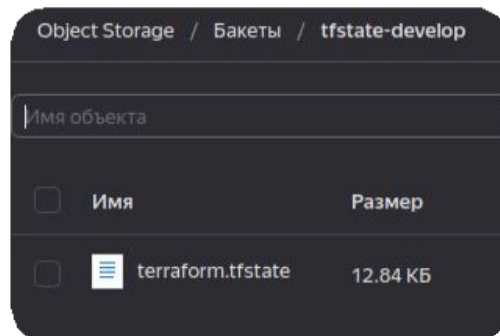
## Initializing the backend...

### Do you want to copy existing state to the new backend?

Pre-existing state was found while migrating the previous "local" backend to the newly configured "s3" backend. No existing state was found in the newly configured "s3" backend. Do you want to copy this state to the new "s3" backend? Enter "yes" to copy and "no" to start with an empty state.

Enter a value: yes

Terraform has been successfully initialized





# Полезные команды при работе с remote s3 state

<code>terraform state pull &gt; example.tfstate</code>	Скачивает state из backend в файл ( <b>backup</b> )
<code>terraform state push example.tfstate -force</code>	Принудительная загрузка state в backend ( <b>restore backup</b> )
<code>terraform plan -out=TFPLAN_FILENAME</code>	Сохраняет план изменения в файл-артефакт. Необходимо для пайплайнов CI/CD и code & plan review
<code>terraform apply TFPLAN_FILENAME</code>	Исполняет ранее сохранённый план изменений
<code>terraform force-unlock &lt;LOCK_ID&gt;</code>	Ручная разблокировка state в случае проблем. Помогает при прерывании процесса, например Ctrl + C

Если ваш Terraform Console блокирует remote state для всей команды, то решение следующее:

```
terraform state pull > ~/test.tfstate && terraform console -state=~/test.tfstate
```

# Настройка backend S3 с блокировкой в YDB



4

# Free tier S3 в Yandex Cloud

## Yandex Object Storage

Каждый месяц не тарифицируются ресурсы стандартного хранилища:

- первый 1 ГБ в месяц хранения
- первые 10 000 операций PUT, POST
- первые 100 000 операций GET, HEAD

default Object Storage / Бакеты / Новый бакет

### Новый бакет

Имя\* ? tfstate-develop

Макс. размер 1 ГБ ☐ Без ограничения

Доступ на чтение объектов ? ☒ Ограниченный ☐ Публичный

Доступ к списку объектов ? ☒ Ограниченный ☐ Публичный

Доступ на чтение настроек ? ☒ Ограниченный ☐ Публичный

Класс хранилища ? ☒ Стандартное ☐ Холодное ☐ Ледяное

Создать бакет

Имя	Занято	Кол-во объектов	
tfstate-develop	0 Б / 1 ГБ	0 объектов	...

# Free tier Managed YDB в Yandex Cloud

## Serverless Managed Service for YDB

Каждый месяц не тарифицируются первые:

- 1 000 000 операций (в единицах Request Unit)
- 1 ГБ/месяц хранения данных

[Инструкция YDB](#)

The screenshot shows the 'Новая база данных' (New Database) configuration page in the Yandex Cloud console. The breadcrumb navigation at the top reads 'Managed Service for YDB / Базы данных / Создать'. The form includes the following sections:

- Имя** (Name): A text field containing 'tfstate-lock'.
- Метки** (Tags): A section with 'Ключ' (Key) and 'Значение' (Value) input fields, and a 'Добавить' (Add) button.
- Тип базы данных** (Database Type): Radio buttons for 'Serverless' (selected) and 'Dedicated'.
- Ограничения** (Limits):
  - Пропускная способность, RU/c** (Throughput, RU/c): A text field with '10' and a checkbox for 'Без ограничения' (No limit).
  - Объем данных, ГБ** (Data volume, GB): A text field with '1'.
- Тарификация** (Billing):
  - Выделенная пропускная способность, RU/c** (Dedicated throughput, RU/c): A text field with '0'.
  - A note below states: 'Почасовой тариф на выделенную пропускную способность не применяется. Все запросы оцениваются по тарифу «за фактическое потребление».' (Hourly rate for dedicated throughput is not applied. All requests are evaluated by the 'pay-as-you-go' rate.)

At the bottom, there are two buttons: 'Создать базу данных' (Create database) in blue and 'Отменить' (Cancel) in grey.

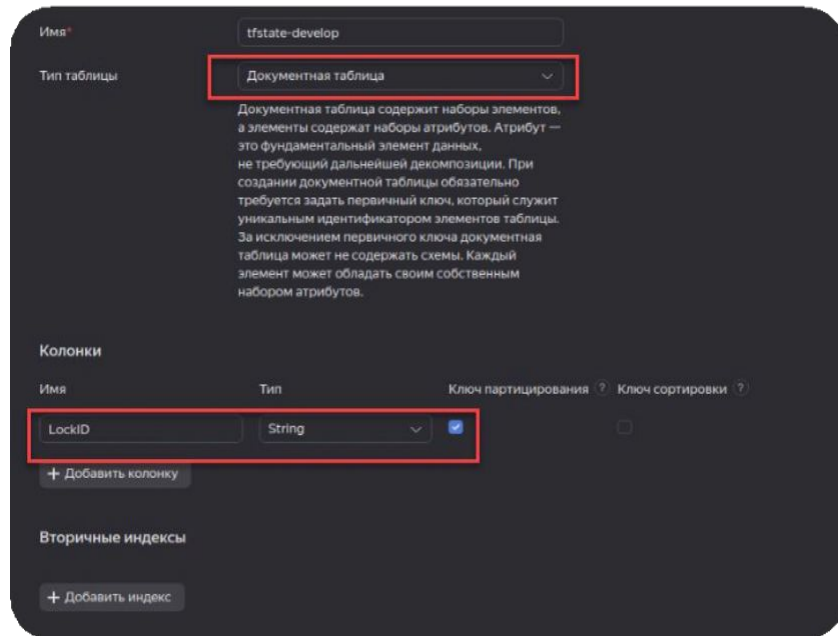
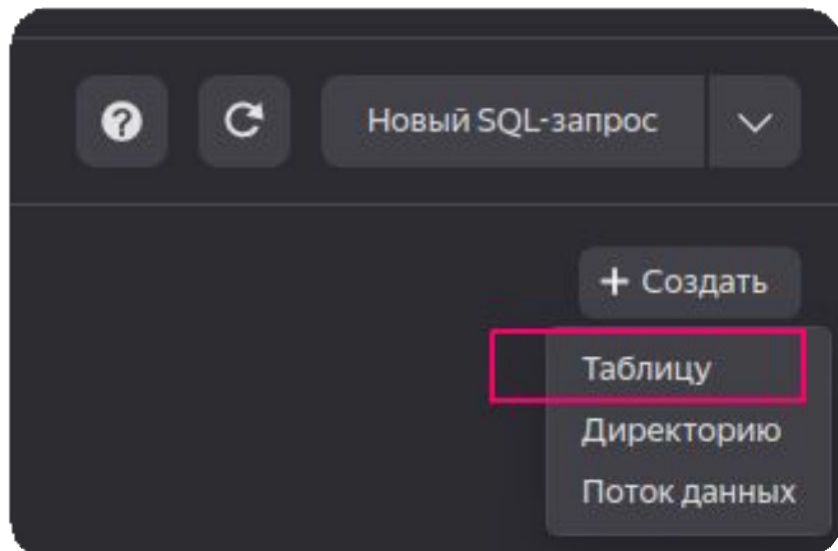
# YDB API endpoint

Во вкладке «Обзор» найдите Document API  
эндпоинт, он нужен для подключения к YDB

## Document API эндпоинт

Эндпоинт ..... <https://docapi.serverless.yandexcloud.net/ru-central1/b1gn3ndpua1j6jaabf79/etnij6ph9brodq9ohs8d>

# Создание таблицы блокировок в YDB



**Имя колонки:** LockID

**Тип:** string

# Создание сервисного аккаунта для S3 и YDB в Yandex Cloud

[Статья](#) о сервисных аккаунтах

Создание сервисного аккаунта

Имя ?

tfstate

Описание ?

handmade account for terraform tfstate

Роли в каталоге

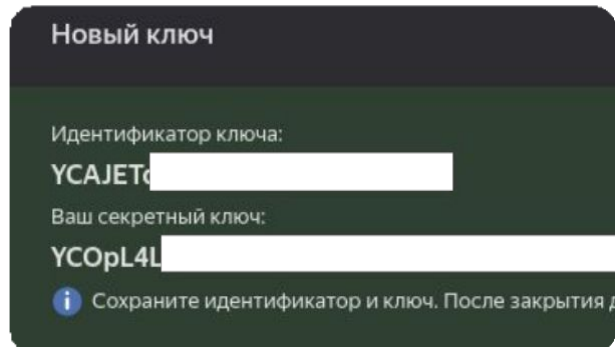
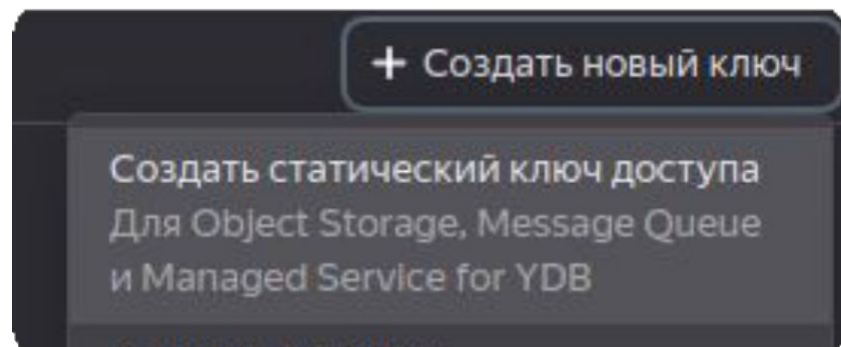
+ Добавить роль

Отменить

Создать

# Создание ключа доступа для сервисного аккаунта в Yandex Cloud


[Статья](#) о создании статических ключей доступа





# Выдача доступов в S3 для сервисного аккаунта в Yandex Cloud

Редактирование ACL


 tfstate-develop 0 Б, 0 объектов

Выберите пользователя

READ

Добавить

Пользователь

 tfstate  
Сервисный аккаунт

Разрешения

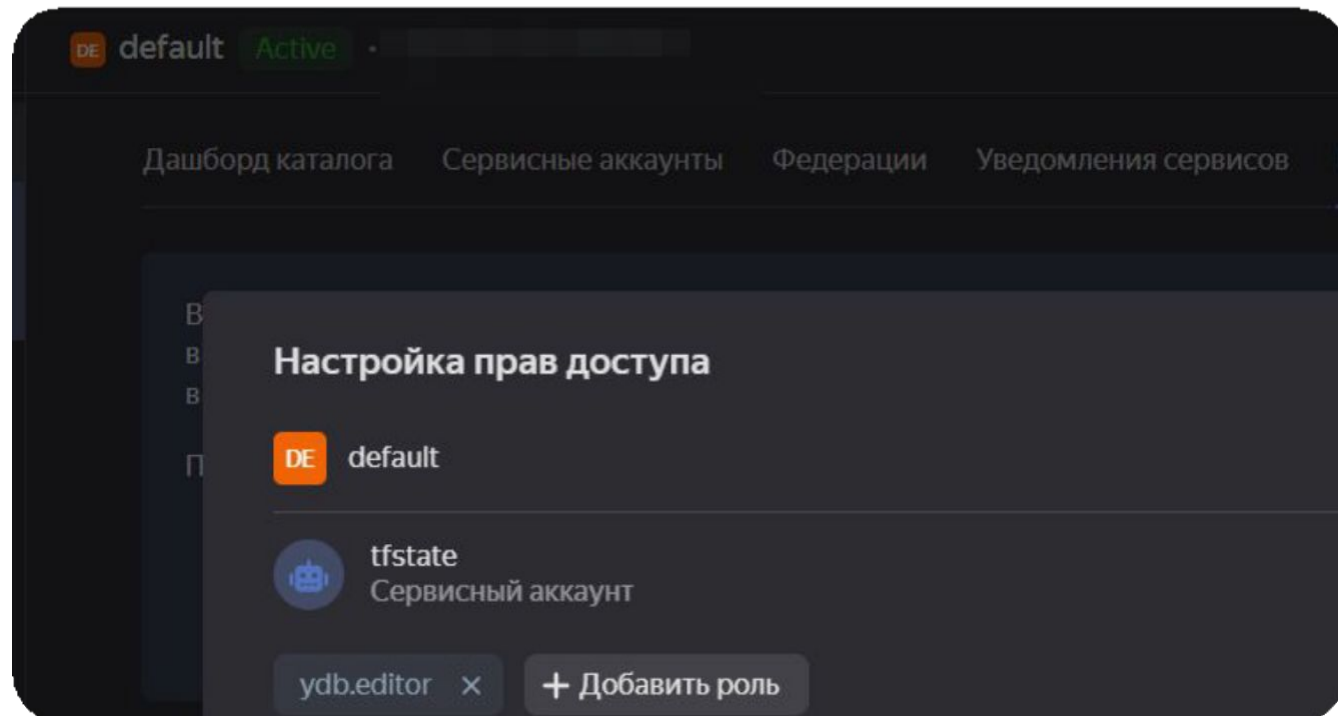
READ и WRITE

Отменить

Отменить

Сохранить

# Выдача доступов в YDB для сервисного аккаунта Yandex Cloud



# Пример настройки backend s3 для Yandex Cloud storage с блокировками

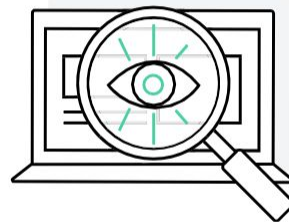
state

```
backend "s3" {  
    endpoint      = "storage.yandexcloud.net"  
    bucket        = "tfstate-develop"  
    region        = "ru-central1"  
    key           = "terraform.tfstate"  
  
    skip_region_validation    = true  
    skip_credentials_validation = true  
  
    dynamodb_endpoint = "https://docapi.serverless.yandexcloud.net/ru-central1/xxx/yyy"  
    dynamodb_table    = "tfstate-lock-develop"  
}
```

# Демонстрация

## работы

- Настройка remote state в Yandex Object Storage с блокировкой в YDB
- Миграция local state в S3 на примере демонстрации в лекции «Продвинутые методы работы с Terraform»



# Securityscan



5

# Chekov

[Инструкция](#) по установке и настройке.

Поддерживается множество **laC tools**:

- Terraform
- K8s
- Helm
- Kustomize
- Ansible
- Dockerfile
- GitLab и другие

Terraform Yandex Provider  
поддерживается checkov

В примере checkov проверил настройки security groups и обнаружил секретный ключ от s3 bucket:

Check: CKV\_SECRET\_6: "Base64 High Entropy String"

FAILED for resource:

9ac6c31563686ca9a2347974391f8555b3cef93c

Severity: LOW

File: /providers.tf:14-15

Guide:

[https://docs.bridgecrew.io/docs/git\\_secrets\\_6](https://docs.bridgecrew.io/docs/git_secrets_6)

```
14 |         # secret_key =  
"YC0pL4*****"
```



# Запуск chekov

Проще всего использовать Docker-образ для проверки текущего каталога:

```
docker pull bridgecrew/checkov
```

```
docker run --rm --tty --volume $(pwd):/tf --workdir /tf bridgecrew/checkov \  
--download-external-modules true --directory /tf
```

# Code & plan review



6





## В CI/CD всегда используйте конструкцию вида:

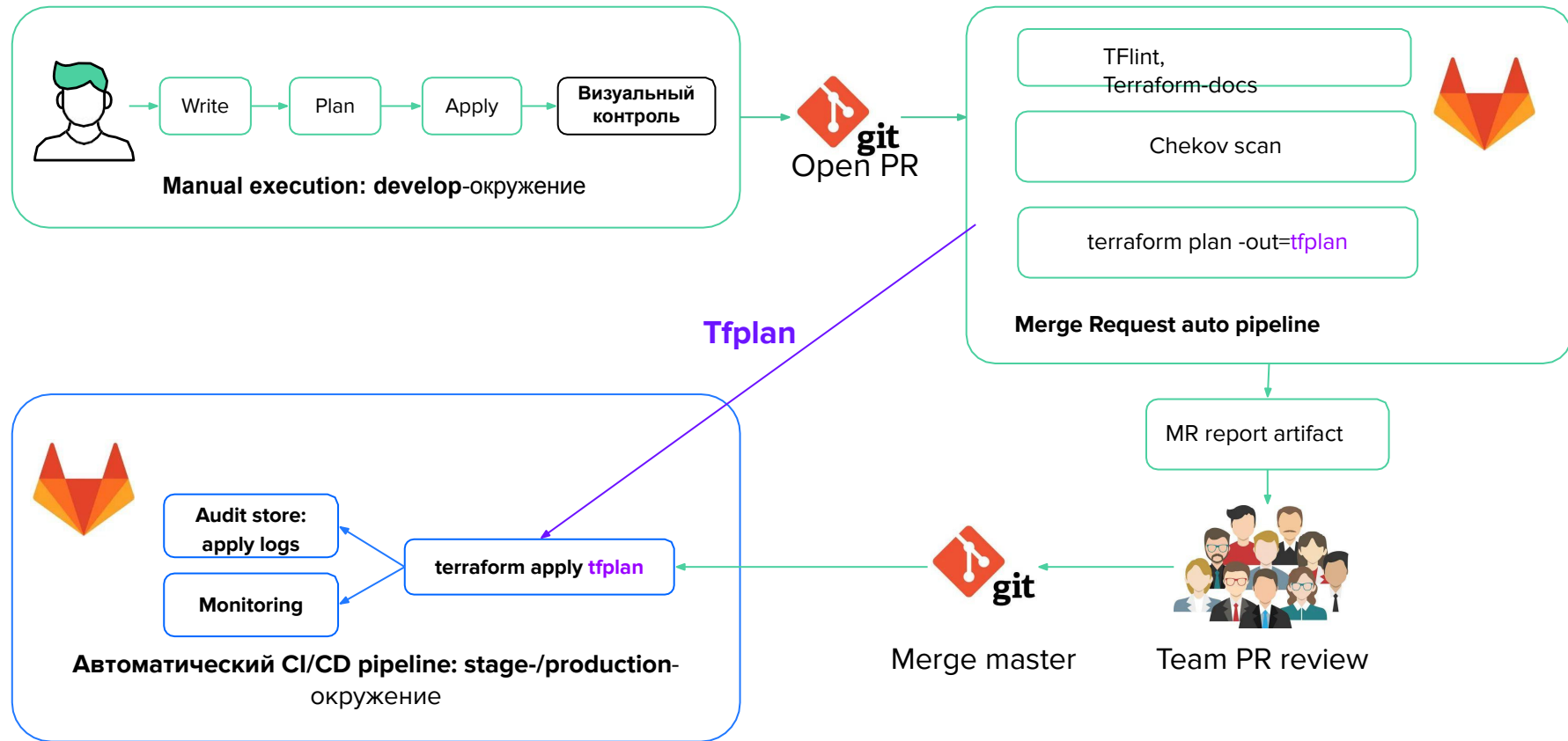
- 1 terraform plan -out=**tfplan**
- 2 plan review
- 3 terraform apply **tfplan**

Это не критично при ручном запуске Terraform.

В CI/CD-пайплайне между **terraform plan** и **terraform apply** в репозитории могут появиться новые коммиты, и новый plan будет отличаться от того, что был проверен в ходе ревью кода.

**Важно:** не забудьте добавить **tfplan** в **.gitignore**

# CI/CD на примере GitLab



# Скрипт пайплайна на примере GitLab

```
workflow:
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
      when: always
  stages:
    - lint_plan
    - apply
  before_script:
    - yc config set service-account-key "${CI_CLOUD_ROBOT}"
    - export cloud_access_token=$(yc iam create-token)
    - terraform init -backend-config="secret_key=${CI_S3_SECRET_KEY_NETOTFSTATE}"
  lint_plan:
    script:
      - tflint>tflint; checkov>checkov
    -terraform plan -var="cloud_access_token=${cloud_access_token}" -out=outfile
    allow_failure:
      exit_codes: [ 2, 3 ]
    artifacts:
      when: always
      paths: [ "outfile", "tflint", "checkov" ]
  apply:
    stage: apply
    script:
      - terraform apply "outfile"
  when: manual
  dependencies: [ "lint_plan" ]
```

[GitHub](#)



# Скрипт пайплайна на примере GitLab

Пайплайн запускается для каждого merge request и выполняет следующие действия:

- 1 Авторизуется в Yandex Cloud через сервисную учётную запись, используя переменную `$CI_CLOUD_ROBOT`, которая должна быть заранее сохранена в GitLab Variables
- 2 Создаёт временный токен доступа к Yandex Cloud через `us iam create-token`, который используется в качестве переменной `cloud_access_token` в дальнейшем
- 3 Авторизуется в Terraform backend, используя переменную `CI_S3_SECRET_KEY_NETOTFSTATE`, которая также должна быть сохранена в GitLab Variables
- 4 Запускает линтеры TFLint и checkov, сохраняя результаты в артефакт `tflint` и `checkov`. Также запускает Terraform plan и сохраняет результаты в файл `outfile`
- 5 На этапе Apply применяет сохранённую конфигурацию из файла `outfile`. Этап Apply запускается вручную с помощью кнопки в интерфейсе GitLab или после автоматического подтверждения MR несколькими коллегами

# Полезные практики



7

# Полезные практики

- Не храните секреты в Git, используйте HashiCorp Vault
- Фиксируйте версии зависимостей
- Пользуйтесь линтерами и анализаторами кода
- Разделяйте dev-, stage-, prod-окружения
- Разделяйте компоненты окружения на логические элементы: сеть, права и доступы, БД, VM
- Используйте модули и валидацию переменных.  
Полезно приложить пример вызова модуля
- Используйте remote state + locking + versioning
- Вносите изменения в master-ветку Git только через Pull/Merge request

# Полезные практики

- Вносите изменения в Git только через Pull/Merge request
- Используйте `sensitive=true`, чтобы не показывать важные переменные в плане
- Тегируйте все ресурсы хотя бы тегами по умолчанию:  
окружение + «terraform managed»
- Вносите изменения в production-среду только через CI/CD
- Изучите [Golang](#) и освоите тестирование Terraform-кода
- Пишите сопровождающую документацию к своему проекту
- Изучайте release notes, когда хотите обновить версию Terraform
- Прочитайте всю доступную документацию HashiCorp по Terraform

# Итоги занятия

Сегодня мы:

- 1 Изучили, какие проблемы могут возникнуть при работе с Terraform в команде
- 2 Научились успешно решать эти проблемы

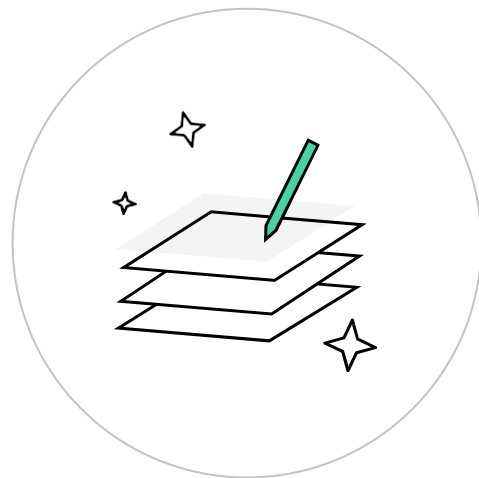




# Домашнее задание

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

- [Naming conventions](#)
- [Backends](#)
- [Yandex Object Storage](#)
- [Таблицы DynamoDB в YDB](#)



# Подготовьте вопросы к разборному вебинару

Евгений Мисяков  
SRE инженер в Нетологии

