Введение в Terraform

Елисей Ильин
DevOps-инженер в Itransition

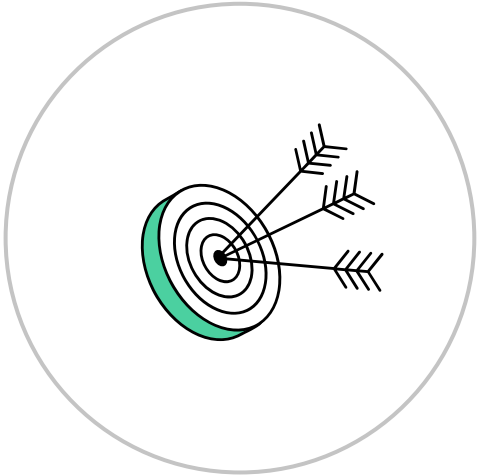
Елисей Ильин

О спикере:

- DevOps-инженер
- Опыт работы в IT 6 лет

Цели занятия

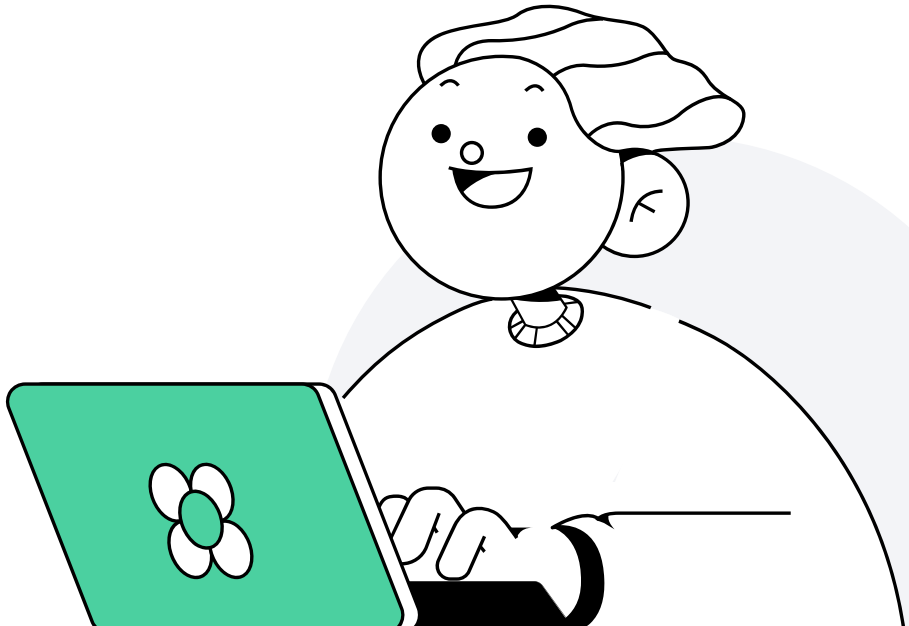
- Изучить аспекты управления инфраструктурой и многообразие инструментов
- Разобраться, почему выбираем именно Terraform
- Познакомиться с компонентами архитектуры Terraform
- Узнать, как писать и выполнять Terraform-код
- Создать несколько локальных ресурсов



План занятия

1. Инфраструктура как код (IaC)
2. Архитектура Terraform
3. Hashicorp Language (HCL)
4. Инициализация инфраструктуры
5. Итоги занятия
6. Домашнее задание

*Нажми на нужный раздел для перехода



Инфраструктура как код (IaC)

1

IaC или **IaaC** — это способ создания и управления инфраструктурой через описание в коде специализированных инструментов. Полностью исключает ручное управление

Подходы к реализации IaC

Императивный (процедурный)	**Декларативный (функциональный)**
Последовательность конкретных действий в коде IaC-инструмента	Описание целевого состояния в коде IaC-инструмента
Примеры: скрипт с aws-cli или yc-tools, ansible-playbook	Примеры: terraform-манифест, k8s-манифест, helm-манифест

Основные преимущества IaC

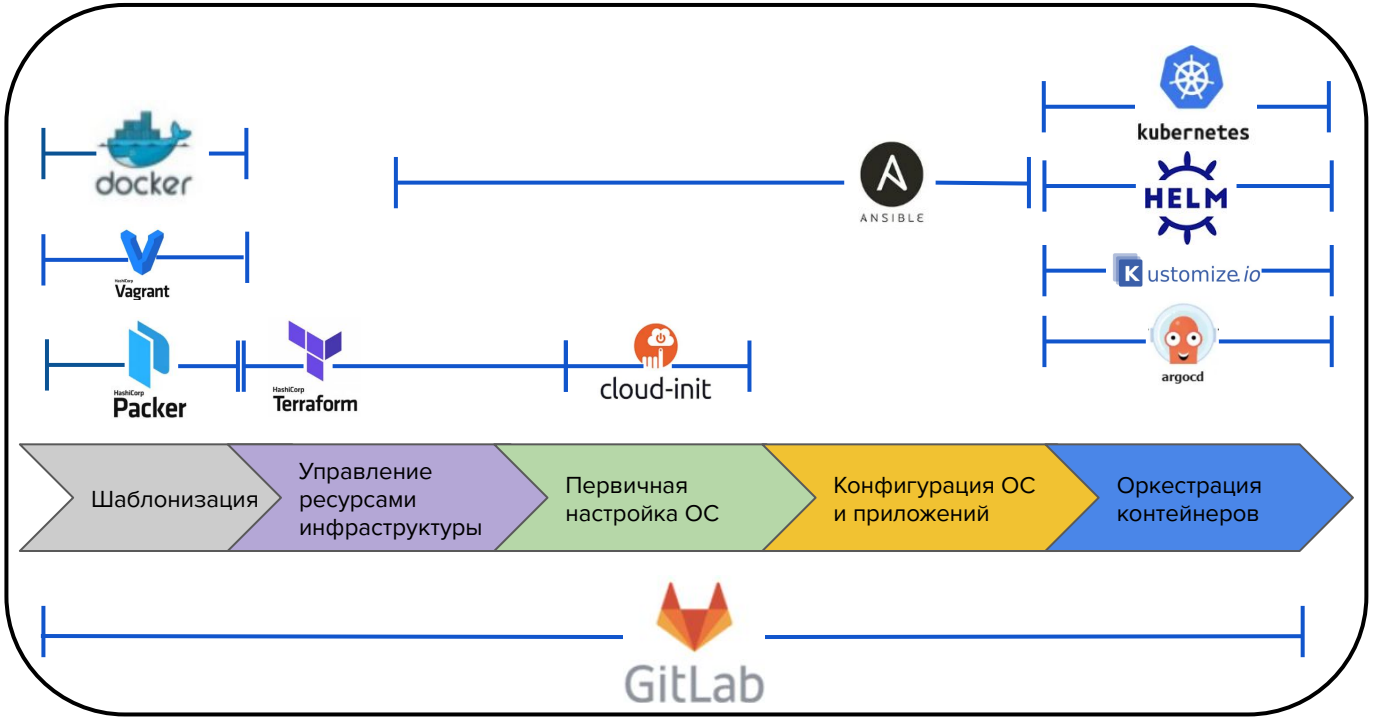
1. Скорость и снижение затрат
2. Масштабируемость и стандартизация
3. Безопасность и документация

Основные недостатки IaC

1. Необходимость соглашения об IaC коде
2. Сложности с ролевой моделью доступа (RBAC)
3. Запаздывание фич

Рекомендуемый стек IaC инструментов

Совместное использование **наиболее** популярных, open source инструментов



Архитектура Terraform

2

Terraform — это **кроссплатформенный** IaC-инструмент с открытым исходным кодом от компании HashiCorp, написанный на языке программирования Golang. Распространяется по лицензии **Mozilla Public License (MPL 2.0).**

Инфраструктура описывается **декларативным** языком HCL (крайне редко JSON)

Для чего используется Terraform

Terraform позволяет **управлять**:

- **ресурсами** в облачных провайдерах, таких как AWS, Azure, Google Cloud, DigitalOcean, Yandex Cloud и т. д.
- **локальной инфраструктурой**, например, OpenStack, Docker, VirtualBox, K8S и прочие.

С помощью golang вы можете расширить область его применения. В 2019 году Nat Henderson в шутку опубликовал в GitHub Terraform-код для заказа пиццы из ресторанов Domino's

Преимущества Terraform

1. Открытый исходный код, кроссплатформенность
2. Большое активное комьюнити
3. Поддерживает множество облачных провайдеров
4. Декларативный язык HCL с простым, читаемым синтаксисом
5. Хранение состояния инфраструктуры, отслеживание изменений
6. Поддержка импорта ресурсов, созданных вручную
6. Не требует установки агентов

Архитектура Terraform

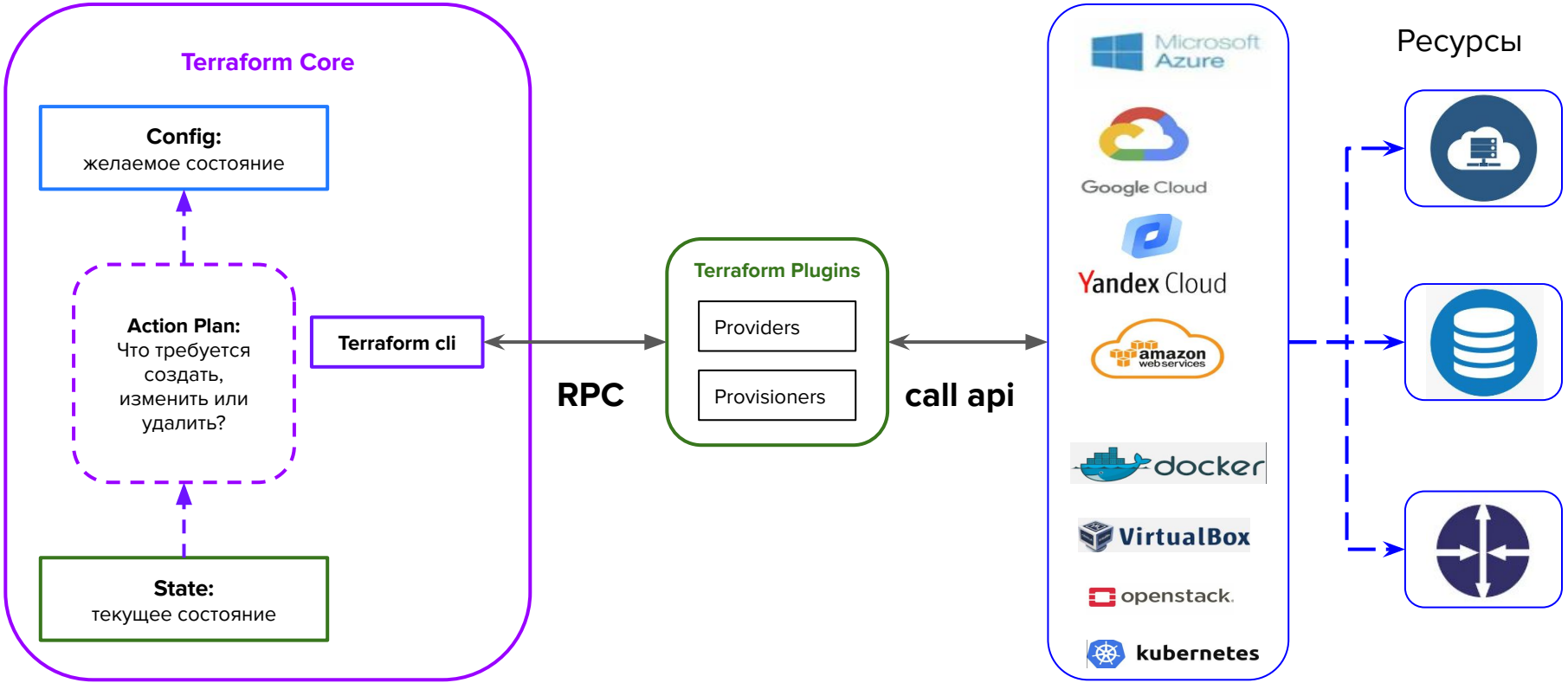
1 Terraform Core:

- Исполняемый (бинарный) файл **Terraform**
- Конфигурационные файлы проекта
- State

2 Terraform Plugins:

- **Providers** — управление инфраструктурой через команды API
- **Provisioners** — запуск команд и скриптов применительно к ресурсам

Архитектура Terraform



Варианты установки

- Скачать исполняемый файл с сайта **terraform.io**
- Установить с помощью пакетных менеджеров (apt, yum, brew, chocolatey и т. д.)
- Скачать исходные файлы с **GitHub,** скомпилировать с помощью **Golang**
- Скачать с **Docker Hub** образ с установленным Terraform
- Для установки **без VPN** можно скачать Terraform из зеркала

State

Terraform сохраняет ***возвращаемую*** информацию о созданных им ресурсах в виде JSON-файла **terraform.tfstate**.

В том числе секретные данные в открытом виде.

Основная цель Terraform state — хранить связь между реальными объектами инфраструктуры и экземплярами ресурсов, объявленными в конфигурации.

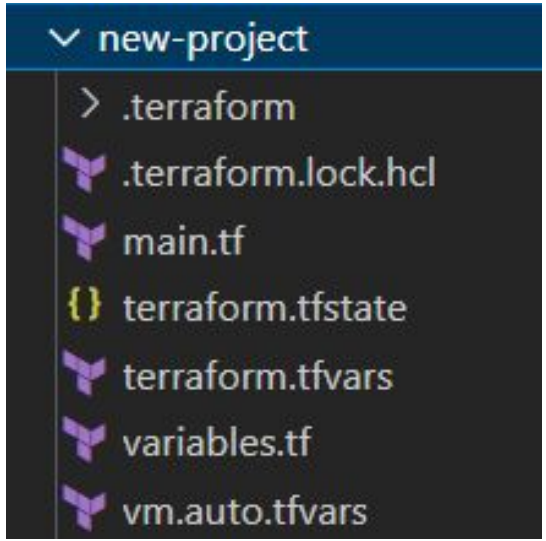
Для коллективной работы используется **remote state**.


```
{ "version": 4,
  "terraform_version": "1.3.7",
  "serial": 69,
  "lineage": "9628672d-ce5b-6d26-36dd-d5539c722be2",
  "outputs": {},
  "resources": [
    { "mode": "managed",
      "type": "random_password",
      "name": "random_string",
      "provider":
"provider[\"registry.terraform.io/hashicorp/random\"]",
      "instances": [
        { "schema_version": 3,
          "attributes": {
            "bcrypt_hash":
$10$B4ZgL2GrOYtBItiI.OGG7O9Zo9lDgsTqYNyBe34,
            "id": "none","keepers": null,"length": 16,"lower":
true,"min_lower": 1,
            "min_numeric": 1,"min_special": 0,"min_upper":
1,"number": true,
            "numeric": true,"override_special": null,"result":
"pECLPAnA83eMa83Y",
            "special": false,"upper": true
          },"sensitive_attributes": []
        }
      ]
    }
  ],"check_results": null
}
```

Root Module — это директория с файлами конфигурации ***.tf**, из которой происходит запуск проекта. Вложенные директории игнорируются

Конфигурационные файлы в Root Module

Полная конфигурация проекта состоит из **Root Module** и вызываемых им **Child Modules** (рассмотрим в следующих лекциях).



- ***.tf** — файлы конфигурации
- **terraform.tfvars** — автоматически загружаемый файл, в котором можно задавать значения переменных
- ***.auto.tfvars** — то же, что и terraform.tfvars, позволяет именовать файлы с переменными (в том числе секретными)
- **.terraform.lock.hcl** — создается при инициализации проекта. Фиксирует версии используемых провайдеров и зависимостей проекта
- **terraform.tfstate** — файл, в котором сохраняется текущее состояние инфраструктуры проекта
- **каталог .terraform** — локальный архив скачанных providers и Child Modules

Hashicorp Language (HCL)

Блоки и аргументы

Код Terraform пишется на языке конфигурации HCL.

- Элементы кода заключаются в **блоки с фигурными скобками {...}**
- Очередность блоков не имеет значения
- Каждый блок имеет **type** и от 0 до 2-х **label**
- Содержимое блоков может включать в себя **аргументы** и вложенные **блоки**

```
#Block:
<Block Type> "<Block Label>" "<Block Label>" {
  <IDENTIFIER> = <EXPRESSION> #Argument
}
```

Terraform код

Terraform код всегда начинается с блока **terraform {...}**.
Внутри списком указываются необходимые **providers**, версия terraform, иные параметры.

Комментарий внутри кода:

- #Однострочный комментарий
- /* Многострочный

  комментарий */

```
terraform {
  required_providers {
    yandex = {
      source = "yandex-cloud/yandex"
    }
    aws = {
      source = "hashicorp/aws"
      version = "~> 2.0"
    }
  }
  required_version = ">= 0.13"
}
```

Блок provider

Конфигурация provider задается в отдельном блоке:

```
provider "provider_name" {..}
```

Любой сторонний провайдер должен быть предварительно загружен из **репозитория** (registry).

Terraform содержит в себе встроенные провайдеры. Они не нуждаются в конфигурации и загрузке

```
provider "yandex" {
     version       = "~> 0.85"
     token         = "t1.9euelZ567…..."
     cloud_id      = "b1g78mf….."
     folder_id     = "b1g6vgh….."
     zone          = "ru-central1-a"
}
```

Версионность в terraform

- **>=** 1.2.0 : версия 1.2.0 или новее
- **<=** 1.2.0 : версия 1.2.0 или более ранняя
- **~>** 1.2.0 : любая 1.2.X — версия
- **~>** 1.2 : любая 1.X.Y — версия
- **>=** 1.0.0, <= 2.0.0 : любая версия от 1.0.0 до 2.0.0

Registry — общедоступный репозиторий, предоставляемый HashiCorp. Содержит в себе множество **opensource providers**

Mirror registry

Существуют частные **зеркала**, которые дублируют содержимое репозитория HashiCorp.

Для смены репозитория по умолчанию необходимо отредактировать файл конфигурации:

- ~/.terraformrc для **Linux/Mac**
- %APPDATA%/terraform.rc для **Windows**

```
provider_installation {
  network_mirror {
    url =
"https://terraform-mirror.yandexcloud.net/"
    include = ["registry.terraform.io/*/*"]
  }
  direct {
    exclude = ["registry.terraform.io/*/*"]
  }
}
```

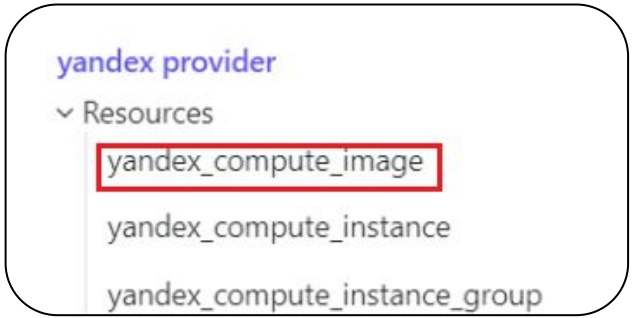
Подробная инструкция от Yandex Cloud

Документация к провайдерам

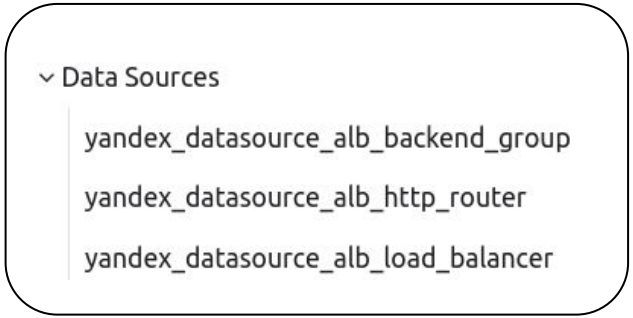
Общедоступное зеркало документации providers.

С помощью provider Terraform может:

- создавать ресурсы (раздел **Resources**)



- считывать информацию о существующих объектах (раздел **Data Sources**)



Блок resourse

Создает объекты, поддерживаемые **provider** — сети, виртуальные машины, базы данных, dns-записи, пароли, файлы и т. п.

Описываются блоком **resource "type" "name" {..}**, содержащим:

- тип объекта из классификатора provider
- уникальное имя в текущем проекте
- аргументы для создания ресурса

```
resource "random_password"  "uniq_name" {
  length    = 16
}
```

Блок resourse

В примере создается ресурс **«пароль»**.

Для дальнейшего использования его значения необходимо обратиться к ресурсу в формате:

```
type.name.параметр

random_password.uniq_name.result
```

Блок datasource

Считывает параметры **уже существующих** объектов инфраструктуры, поддерживаемых **provider**.

Описываются блоком кода `data "type" "name" {..}`, содержащим:

- тип объекта из классификатора provider
- уникальное имя в текущем проекте
- фильтр-запрос

```
data "local_file" "version" {
  filename    = "/proc/version"
}
```

Блок datasource

В примере считывается дата-ресурс **«файл»**.

Для дальнейшего использования **всех** его параметров необходимо обратиться к дата-ресурсу в формате: `data.type.name`

```
data.local_file.version
```

Или отфильтровать конкретный параметр: `data.type.name.параметр`

```
data.local_file.version.content
```

Инициализация инфраструктуры

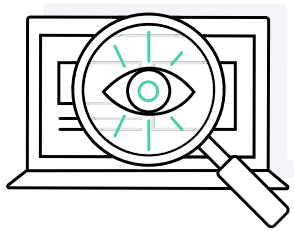
4

Базовые команды

terraform init	Скачивание зависимостей
terraform validate	Проверка синтаксиса конфигурации и доступности зависимостей
terraform plan	**terraform validate** + Отображение планируемых изменений в инфраструктуре (DRY RUN)
terraform apply	**terraform plan** + Внесение изменений в инфраструктуру (если они есть)
terraform destroy	Уничтожение ранее созданных Terraform объектов инфраструктуры
terraform fmt	Встроенное автоформатирование текста в конфигурации проекта

Демонстрация работы

- init
- plan
- apply
- destroy



Пример простой конфигурации

Используем только встроенные провайдеры от HashiCorp.

Создаем ресурс "случайный пароль".

Обращаемся к значению сгенерированного пароля и записываем его в файл /tmp/from_resource.txt.

Считываем файл /proc/version в дата-ресурс.

Обращаемся к содержимому дата-ресурса и записываем его в файл /tmp/from_data_source.txt

```
terraform {
  required_providers { }
    required_version =   ">=0.13"
}

resource "random_password"  "any_uniq_name" {
  length    = 16
}

resource "local_file"  "from_resourse" {
  content    = random_password.any_uniq_name.result
  filename = "/tmp/from_resource.txt"
}


data  "local_file"  "version" {
  filename    = "/proc/version"
}


resource "local_file"  "from_dataresourse" {
  content    = data.local_file.version.content
  filename = "/tmp/from_data_source.txt"
}
```

План исполнения

```
terraform apply
  # random_password.any_uniq_name will be
created
  + resource "random_password"
"any_uniq_name" {
      + bcrypt_hash = (sensitive value)
      + id          = (known after apply)
      + length      = 16
      + lower       = true
      + number      = true
      + special     = true
      +upper       = true
      + numeric     = true
      + result      = (sensitive value)
…..
    }
Do you want to perform these actions?
Plan: 1 to add, 0 to change, 0 to destroy.
Apply complete! Resources: 1 added,
 0 changed, 0 destroyed
```

```
terraform destroy
  # random_password.any_uniq_name will be
destroyed
  - resource "random_password"
"any_uniq_name" {
      - bcrypt_hash = (sensitive value)
      - id          = "none" -> null
      - length      = 16 -> null
      - lower       = true -> null
      - number      = true -> null
      - special     = true -> null
      - upper       = true -> null
      - result      = (sensitive value)
   }

Do you want to perform these actions?

Plan: 0 to add, 0 to change, 1 to destroy.

Destroy complete! Resources: 1 destroyed
```

Итоги занятия

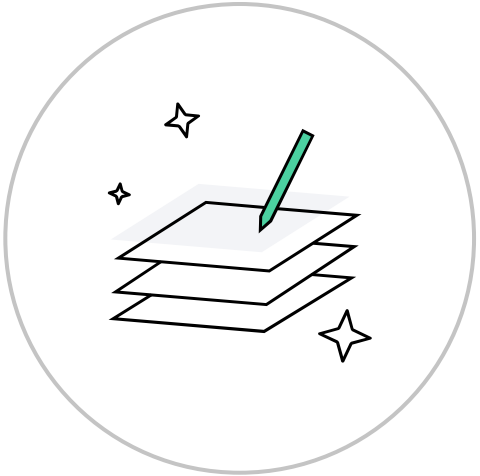
Сегодня мы

1. Узнали про аспекты управления инфраструктурой и используемые инструменты
2. Разобрались в преимуществах Terraform
3. Рассмотрели компоненты архитектуры Terraform
4. Поняли, как писать и выполнять Terraform-код
5. Попробовали создать локальные ресурсы

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

- Синтаксис HCL
- Resource Blocks
- Data Sources Blocks
- Провайдеры

Задавайте вопросы и пишите отзыв о лекции

Елисей Ильин
DevOps-инженер в Itransition