Введение в Terraform

Евгений Мисяков
SRE инженер в Нетологии

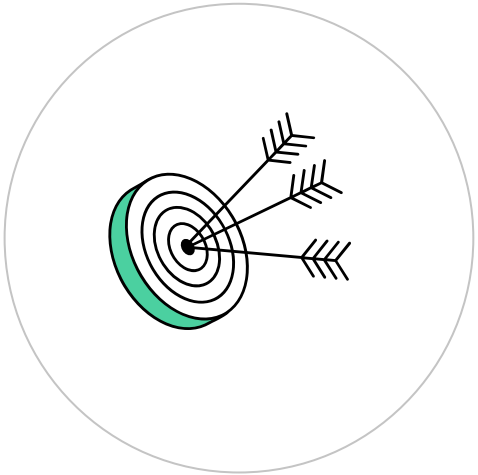
Евгений Мисяков

О спикере:

- SRE инженер в Нетологии

Цели занятия

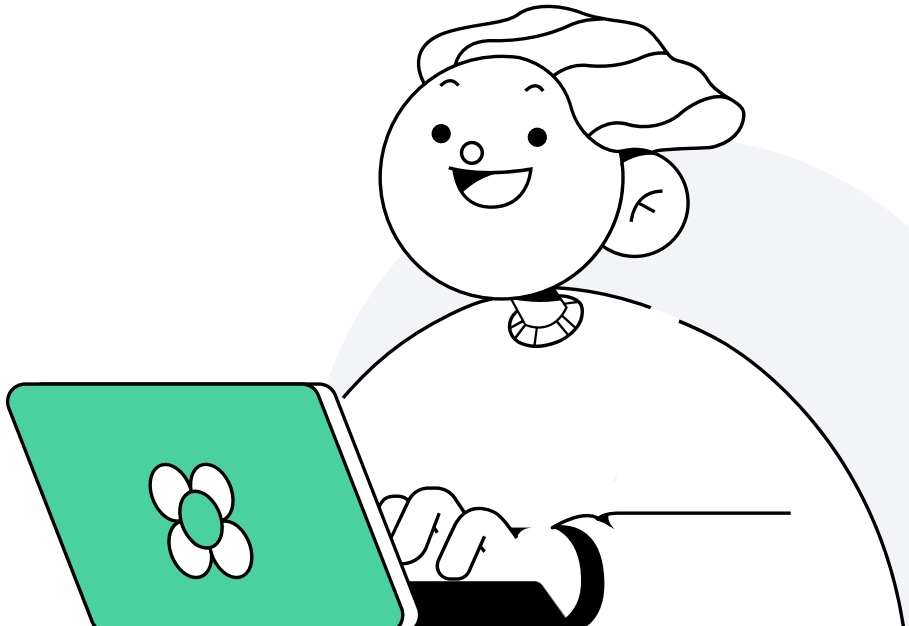
- Изучить аспекты управления инфраструктурой и многообразие используемых для этого инструментов
- Разобраться, почему выбираем именно Terraform
- Познакомиться с компонентами архитектуры Terraform
- Узнать, как писать и выполнять Terraform-код
- Создать несколько локальных ресурсов



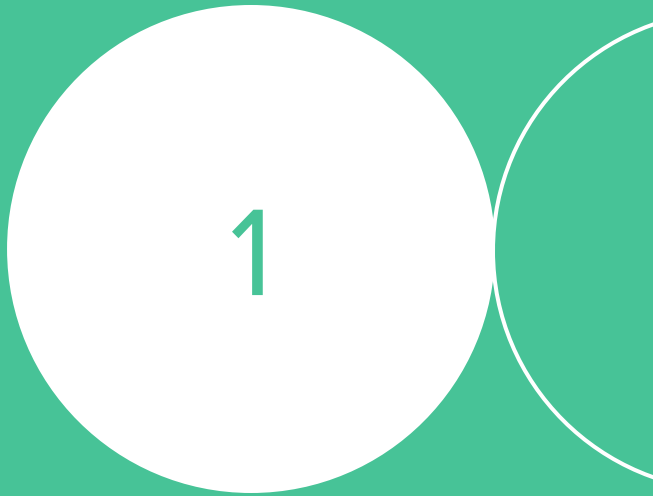
План занятия

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)



IaaC (IaC)

Метод создания и управления инфраструктурой посредством написания кода с использованием специализированных инструментов. Этот подход полностью исключает необходимость в ручном управлении

Подходы к реализации IaC

Императивный (процедурный)	**Декларативный (функциональный)**	**Гибридный**
Инфраструктура описывается в виде набора команд, которые нужно выполнить для достижения желаемого состояния. Это позволяет более точно регулировать процесс создания инфраструктуры и управления ею, но требует более высокого уровня экспертности и необходимости в ручной настройке	Инфраструктура описывается в виде манифестов, в которых указывается, как она должна выглядеть. Инструменты IaC автоматически создают инфраструктуру и управляют ресурсами, чтобы достичь этого состояния	Объединяет декларативный и императивный подходы, позволяя использовать их преимущества
Примеры: скрипт с aws-cli или yc-tools, ansible-playbook	**Примеры:** Terraform-манифест, K8s-манифест, Helm-манифест	**Пример:** Terraform создаёт виртуальную машину, затем Ansible её настраивает

Основные преимущества IaC

1. **Скорость и снижение затрат**
   - Автоматизация создания, настройки и управления инфраструктурой
   - Воспроизводимость создания инфраструктуры в разных средах

2. **Масштабируемость и стандартизация**
   - Быстрое масштабирование инфраструктуры
   - Стандартизация процессов создания инфраструктуры и управления ею

3. **Безопасность и документация**
   - Учёт безопасности при создании инфраструктуры
   - Документирование конфигурации инфраструктуры

Основные недостатки IaC

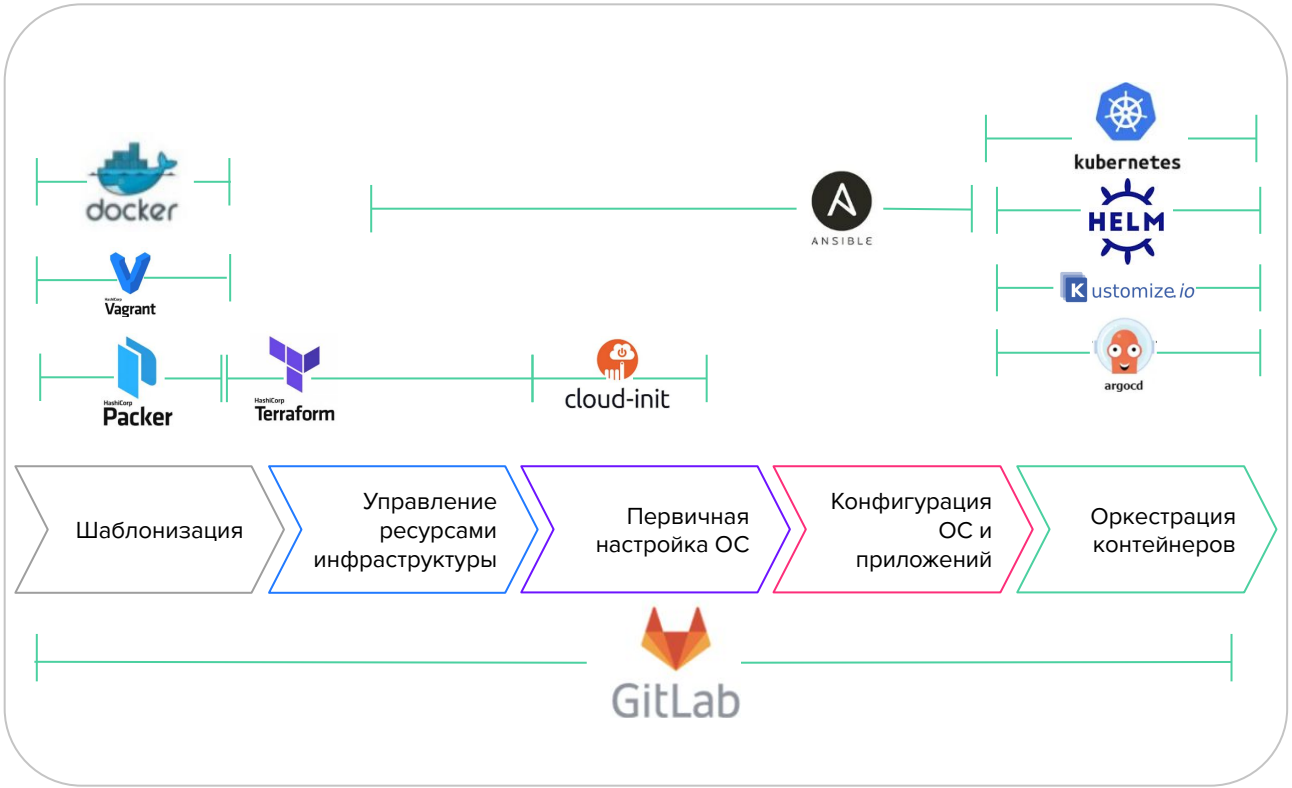
1. **Сложность и затратность внедрения**
   - Необходимость изучать инструменты и языки программирования для создания конфигурационных файлов
   - Затраты провайдера на создание и поддержку инфраструктуры для автоматизации процессов

2. **Риск появления глобальных ошибок и уязвимостей**
   - Возможность допустить ошибки при создании конфигурационных файлов или скриптов
   - Риск возникновения уязвимостей в инфраструктуре, если конфигурационные файлы содержат ошибки

3. **Необходимость постоянного обновления и тестирования**
   - Необходимость постоянно обновлять и тестировать конфигурационные файлы для поддержания корректной работы инфраструктуры
   - Риск возникновения проблем при обновлении инфраструктуры, если конфигурационные файлы не обновляются или не тестируются достаточно часто

Рекомендуемый стек IaC-инструментов

Совместное использование наиболее популярных open-source-инструментов



Архитектура Terraform

2

Terraform — это бесплатный кросс-платформенный инструмент для управления инфраструктурой, созданный компанией HashiCorp.

Terraform написан на языке программирования **Golang** и распространяется по лицензии Mozilla Public License (MPL 2.0).

Позволяет описывать инфраструктуру как код с использованием декларативного языка HCL (редко — JSON), что упрощает создание и изменение различной инфраструктуры, а также управление ею

Для чего используется Terraform

Terraform позволяет управлять:

- ресурсами в облачных провайдерах: AWS, Azure, Google Cloud, DigitalOcean, Yandex Cloud и т. д.
- локальной инфраструктурой: OpenStack, Docker, VirtualBox, K8s и пр.

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

Преимущества Terraform

- **Открытый исходный код и кросс-платформенность** позволяют использовать Terraform на различных платформах и расширять его функциональность. Большое активное комьюнити обеспечивает поддержку и развитие инструмента
- **Поддержка множества облачных провайдеров** даёт возможность применять Terraform для управления инфраструктурой в различных облачных средах: AWS, Azure, Google Cloud, DigitalOcean и др.
- **Декларативный язык HCL** с простым, читаемым синтаксисом позволяет описывать инфраструктуру как код и управлять ею с помощью привычных инструментов разработки

Преимущества Terraform

- **Хранение состояния инфраструктуры** и отслеживание изменений дают возможность контролировать изменения в инфраструктуре и быстро восстанавливать её в случае сбоев или ошибок
- **Поддержка импорта ресурсов**, созданных вручную, позволяет интегрировать уже существующую инфраструктуру в управляемый Terraform-проект
- **Terraform не имеет выделенного сервера управления.** Вы одинаково настраиваете его как на локальных рабочих станциях администраторов, так и на сервере CI/CD

Архитектура Terraform

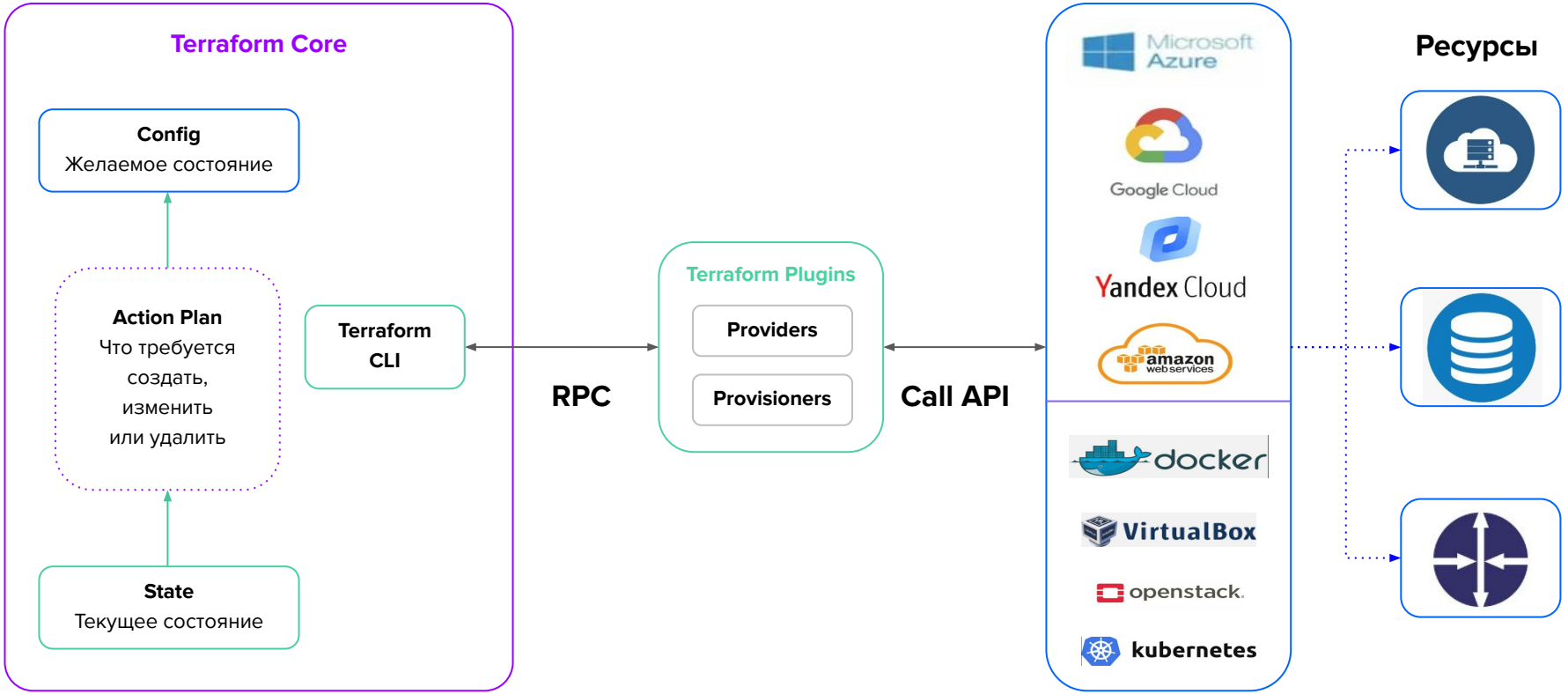
1 Terraform Core

- Исполняемый файл **Terraform**, который обрабатывает конфигурационные файлы проекта и управляет состоянием инфраструктуры
- Конфигурационные файлы проекта, написанные на языке **HCL** или **JSON**
- Файл состояния — **State file**

2 Terraform Plugins

- **Providers** — модули, которые позволяют Terraform управлять инфраструктурой через команды API, предоставляемые различными облачными провайдерами
- **Provisioners** — модули, которые позволяют Terraform запускать команды и скрипты на управляемых ресурсах после их создания или обновления

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 **.tfstate** все секретные данные содержатся 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 в Terraform — это директория, которая содержит файлы конфигурации Terraform с расширением .tf и из которой Terraform запускается для создания и управления инфраструктурой

Root module

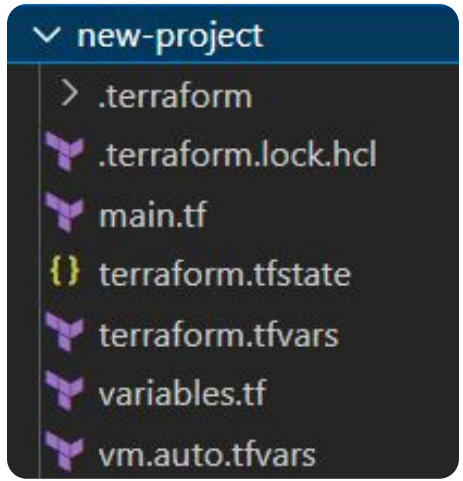
В этой директории может находиться несколько файлов конфигурации, каждый из которых определяет ресурсы и провайдеры для управления инфраструктурой. Это позволяет удобно организовывать код и разделять ресурсы по файлам.

Terraform при запуске считывает все файлы конфигурации в **root module** как единый файл.

Вложенные директории в root module Terraform игнорируются и не используются в создании инфраструктуры, если они не подключены в качестве дочерних модулей

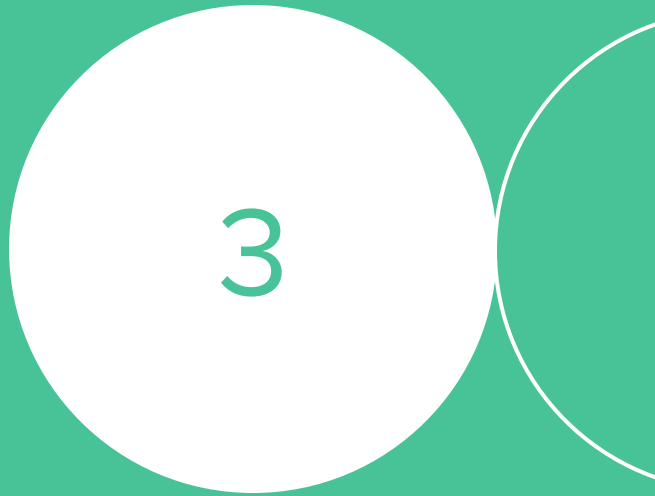
Конфигурационные файлы в root module

Полная конфигурация проекта состоит из **root module** и вызываемых им **child modules** (их 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** — создаётся 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

- Элементы кода заключаются в **блоки с фигурными скобками** {...}
- Очерёдность блоков не имеет значения
- У каждого блока есть **block type** и от 0 до 2 **block label**
- Содержимое блоков может включать **аргументы** и вложенные **блоки**
- Аргументы определяют свойства и параметры ресурсов
- Вложенные блоки используются для определения связей между ресурсами и другими элементами конфигурации

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

Блоки и аргументы

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 задаётся 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

В Terraform используется 2 типа версий: версия Terraform и версия провайдера. Лучше всего указывать конкретные версии, чтобы избежать проблем совместимости и неожиданных изменений в инфраструктуре в результате обновления Terraform или провайдеров.

- = 1.2.1 — фиксированная версия 1.2.1
- >= 1.2.0 — требуется версия 1.2.0 или новее
- <= 1.2.0 — требуется версия 1.2.0 или более ранняя
- ~> 1.3.0 — требуется любая версия Terraform или провайдера, начиная с версии 1.3 и **до** следующей главной версии (например, 1.4.0 или 1.5.0)
- >= 1.0.0, <= 2.0.0 — требуется любая версия начиная с 1.0.0 и заканчивая 2.0.0 включительно

Registry — это общедоступный репозиторий, предоставляемый компанией HashiCorp и содержащий множество публичных модулей, провайдеров и других ресурсов, которые могут быть использованы в Terraform.

В настоящее время Terraform Registry содержит более 700 провайдеров и тысячи модулей, созданных сообществом и разработчиками

Mirror registry

Для работы Terraform в ограниченных сетевых условиях может потребоваться настройка частного зеркала для Terraform Registry.

Зеркала могут содержать копию репозитория HashiCorp, а также дополнительные модули и провайдеры, которые применяются внутри организации.

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

Для использования зеркала необходимо отредактировать файл конфигурации:

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

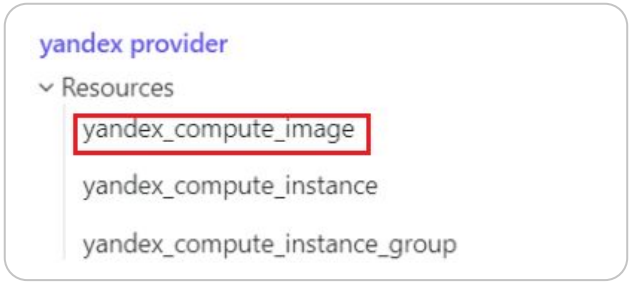
Подробная инструкция от Yandex Cloud

Документация к провайдерам

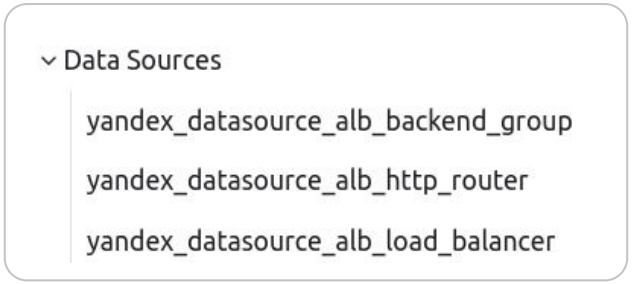
Существует ряд общедоступных зеркал с документацией для Terraform providers.

С помощью provider Terraform может:

- создавать ресурсы (раздел Resources)



- считывать информацию о существующих объектах, которые были созданы вне текущей конфигурации Terraform (раздел Data Sources)



Блок resource

Создаёт объекты, поддерживаемые **provider**: сети, виртуальные машины, базы данных, DNS-записи, пароли, файлы и т. п.

Объекты 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 resource

В примере создаётся 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**.

Объекты 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
```

IDE (integrated development environment) — интегрированная среда разработки, которая объединяет в себе различные инструменты и функции, необходимые для разработки программного обеспечения в конкретной среде.

Обычно включает редактор кода с подсветкой синтаксиса и автодополнением, отладчик, систему контроля версий, средства анализа кода, инструменты для создания пользовательского интерфейса и другие полезные функции.

Рекомендуем установить Visual Studio Code и расширение Terraform от HashiCorp для валидации кода. Также полезно включить автоматическое форматирование кода при сохранении файла.

Инициализация инфраструктуры

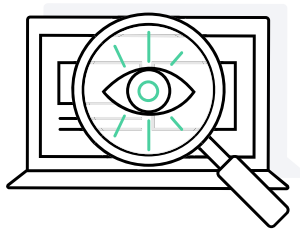
4

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 **всех** ранее созданных Terraform-объектов инфраструктуры
terraform fmt	Встроенное автоформатирование текста в конфигурации проекта

Демонстрация работы

- IDE Visual Studio Code
- init
- plan
- apply
- Изменение ресурсов
- destroy



Пример простой конфигурации

Используем только встроенные провайдеры от HashiCorp.

Создаём 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:

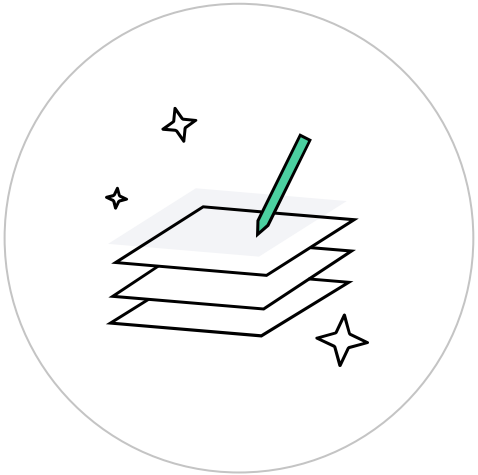
1. Рассмотрели основные аспекты управления инфраструктурой и инструменты, используемые для этого
2. Изучили преимущества использования Terraform для управления инфраструктурой
3. Разобрались в компонентах архитектуры Terraform: провайдерах и ресурсах
4. Узнали, как писать и выполнять Terraform-код с использованием команд CLI и файлов конфигурации
5. Попробовали создать локальные ресурсы, используя Terraform

Всё это послужит отправной точкой для дальнейшего изучения инструмента и разработки более сложных конфигураций

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 ставится 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, Itransition