

Основы работы с Terraform

Yandex Cloud

Евгений Мисяков
SRE инженер в Нетологии



Евгений Мисяков

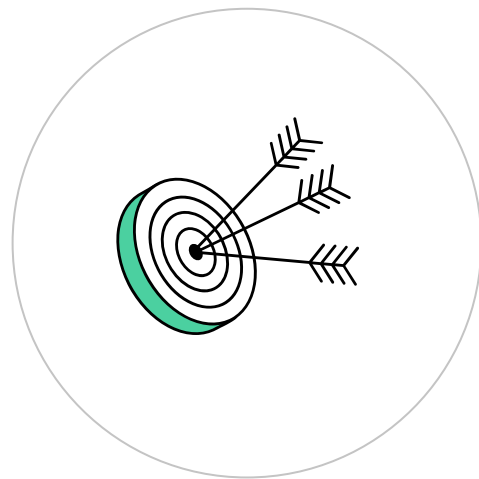
О спикере:

- SRE инженер в Нетологии



Цели занятия

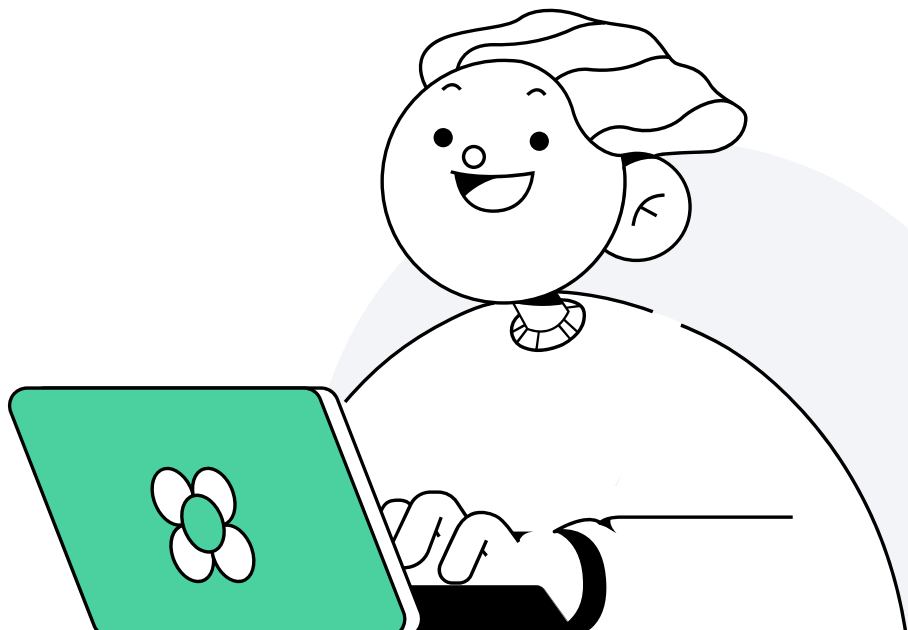
- Рассмотреть возможности облачных решений
- Понять, как работать с облачной платформой на примере Yandex Cloud
- Узнать, как использовать Terraform для создания, изменения и удаления ресурсов в Yandex Cloud
- Изучить типы переменных в Terraform и их назначение



План занятия

- 1 Облачные вычисления
- 2 Основы работы с Yandex Cloud
- 3 Переменные в Terraform
- 4 Итоги занятия
- 5 Домашнее задание

*Нажми на нужный раздел для перехода



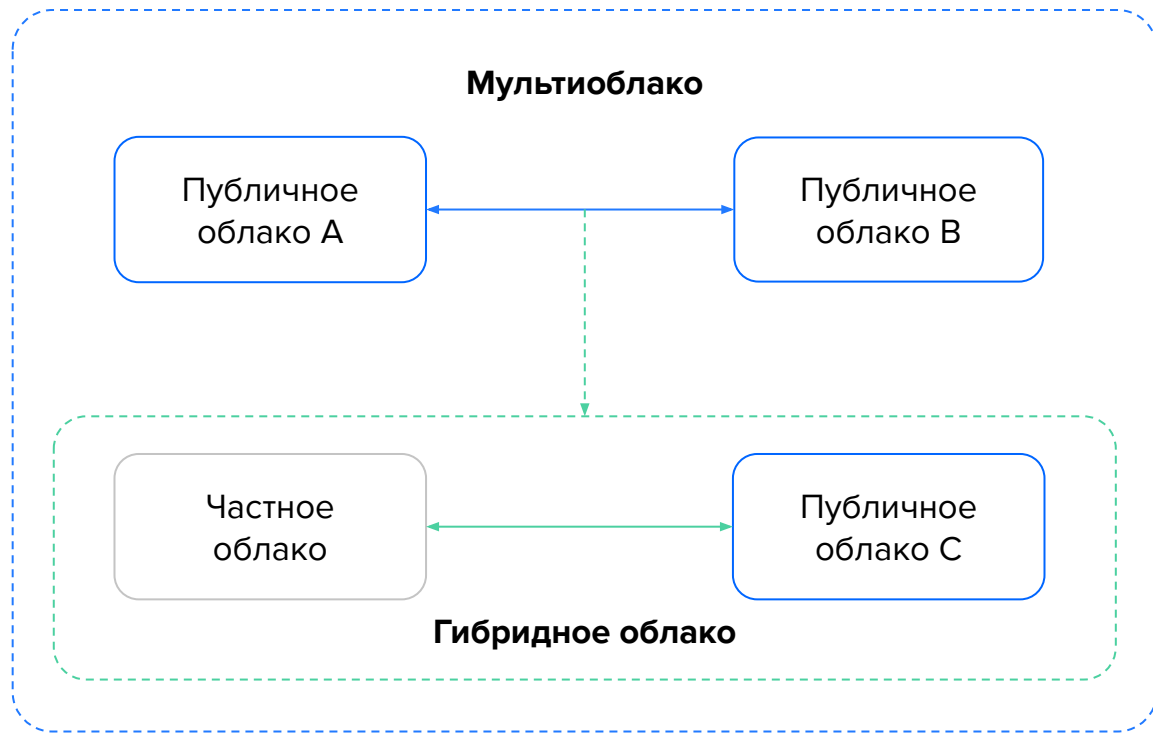
Облачные вычисления

Cloud computing



1

Типы облаков



Типы облачных услуг, границы управляемости



Распределение ответственности за безопасность



Наиболее популярные облачные провайдеры в России

У каждого провайдера свои преимущества и недостатки

Yandex Cloud

 VK Cloud

#CloudMTS

 SBER CLOUD

Selectel

CROC Cloud

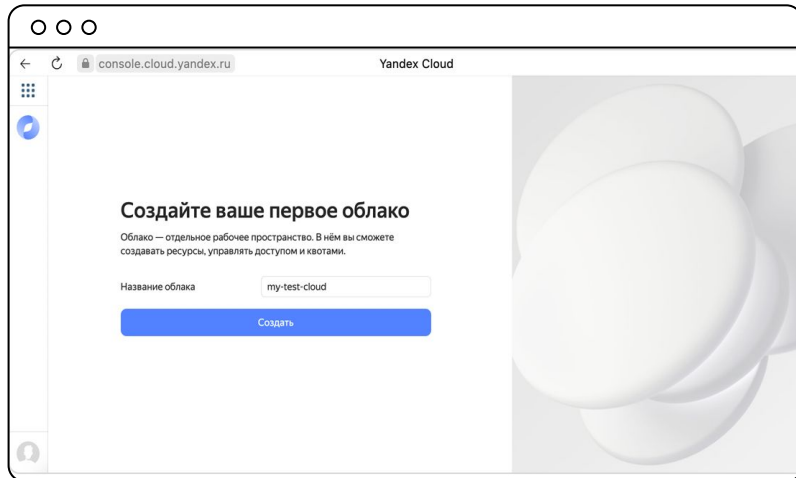
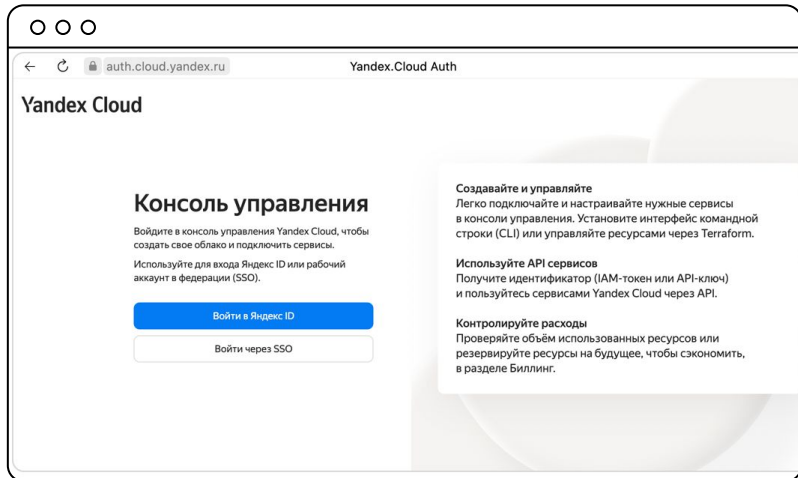
Основы работы с Yandex Cloud



2

Регистрация аккаунта

Вам будет присвоена главная роль: владелец облака, **cloud.owner**

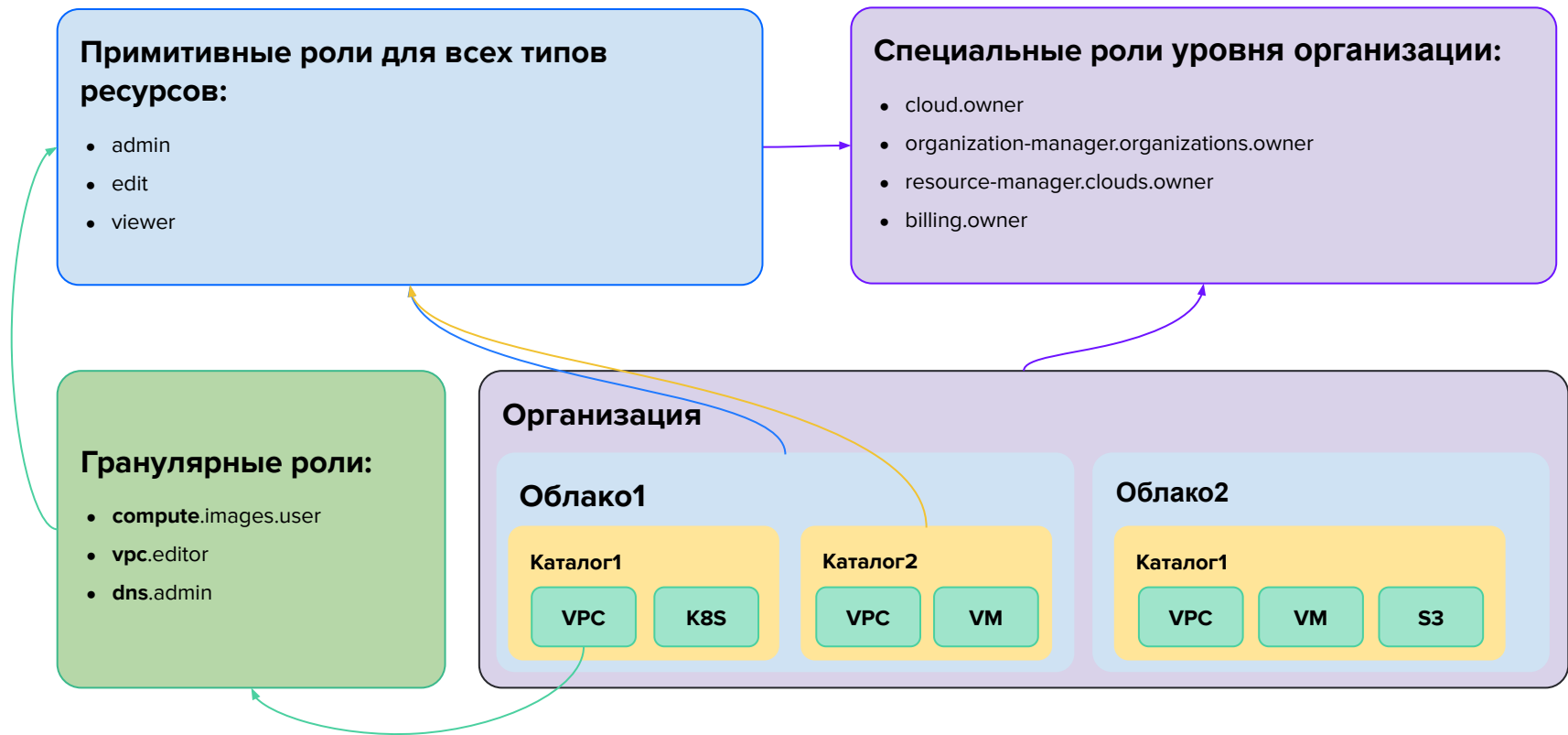




Иерархия ролей и ресурсов в Yandex Cloud позволяет гибко управлять доступом к ресурсам и обеспечивает безопасность данных в облаке.

Каждый уровень иерархии может быть точно настроен в соответствии с потребностями организации и проектов: начиная от организации и заканчивая отдельными ресурсами

Иерархия ролей и ресурсов

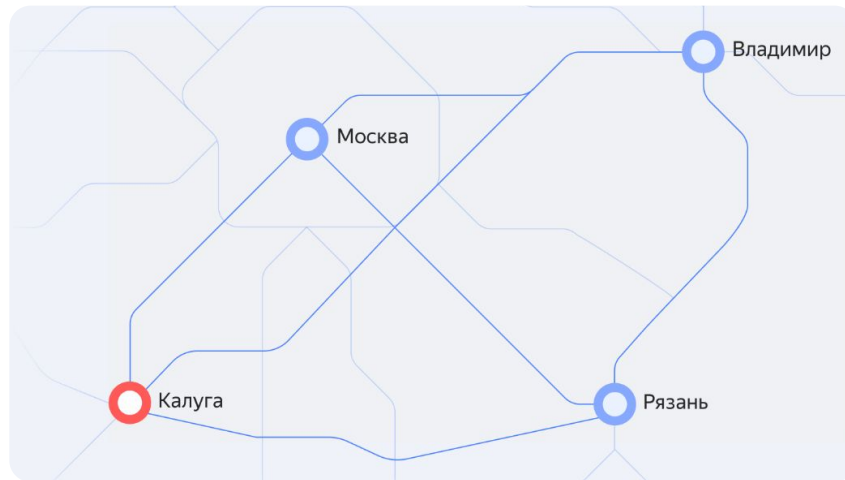




Зоны доступности (availability zones) — это физически отдельные локации, расположенные в пределах одного региона. Пока что у YC только 1 регион: ru-central1.

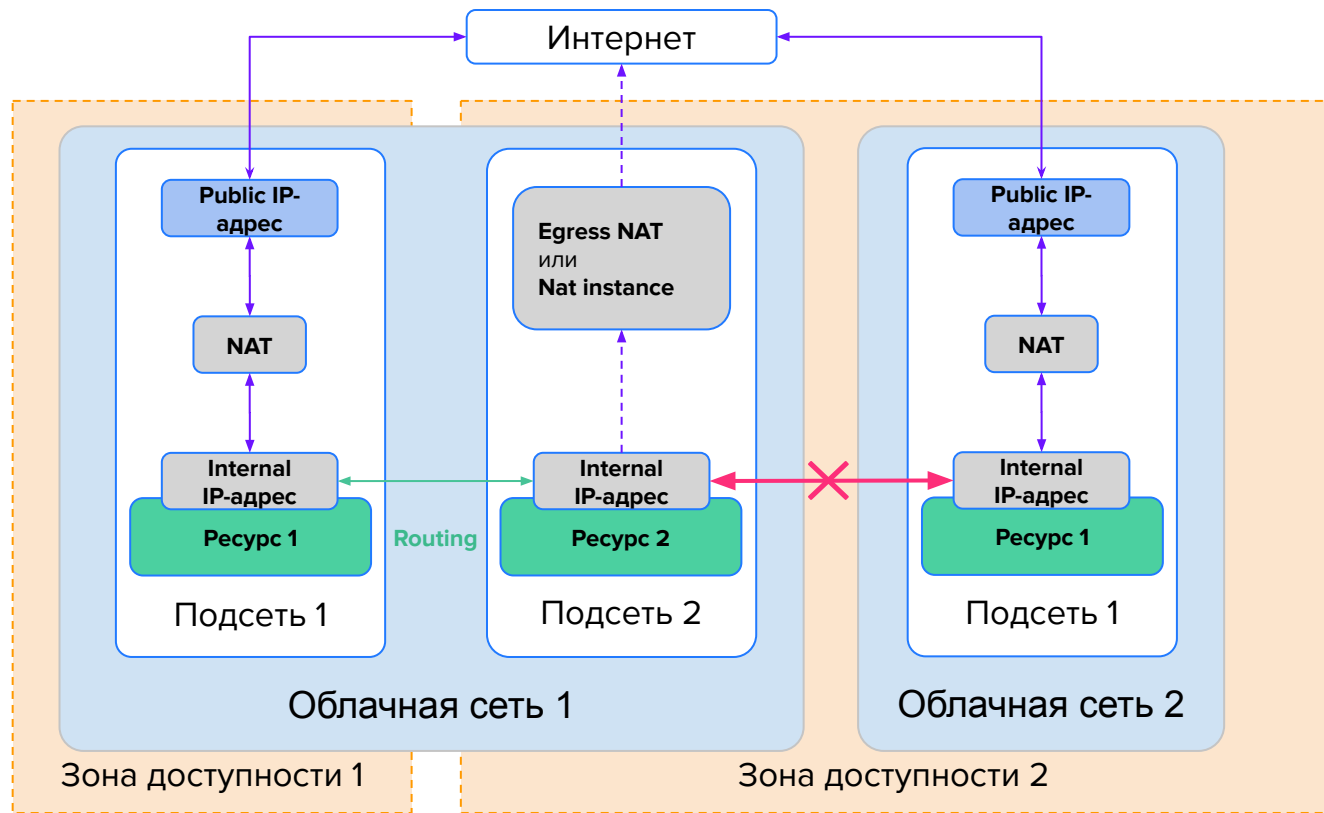
У каждой зоны собственная инфраструктура, энергоснабжение и сетевое оборудование, что обеспечивает высокую доступность и надёжность сервисов в облаке

Зоны доступности (availability zones)



○○○		
Регион	Зоны	Географическая привязка
ru-central1	ru-central1-a ru-central1-b ru-central1-c	Владимирская, Московская и Рязанская области.

Virtual private cloud (VPC)



Квоты Yandex Cloud

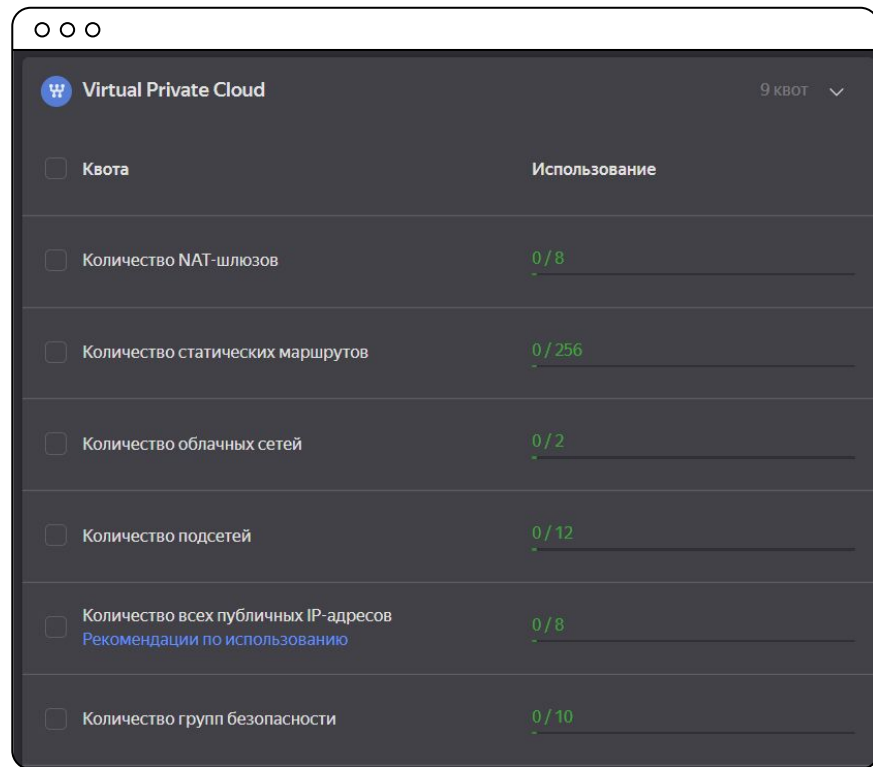
Yandex Cloud устанавливает квоты на разные ресурсы в облаке, например:

- количество виртуальных машин
- общий объём дискового пространства
- общее количество CPU и RAM
- количество VPC (дефолт VPC = 2, **частая ошибка — не удалена инфраструктура из предыдущего задания**)

Начальные квоты можно увеличить, написав запрос в техническую поддержку Yandex Cloud.

Но тогда, чтобы обеспечить финансовую безопасность, у учётной записи должны быть соответствующие права. Администратор или хакер не смогут создать ресурсов больше, чем разрешил биллинг-аудитор

Пример стартовых квот в Yandex Cloud



<input type="checkbox"/> Квота	Использование
<input type="checkbox"/> Количество NAT-шлюзов	0 / 8
<input type="checkbox"/> Количество статических маршрутов	0 / 256
<input type="checkbox"/> Количество облачных сетей	0 / 2
<input type="checkbox"/> Количество подсетей	0 / 12
<input type="checkbox"/> Количество всех публичных IP-адресов Рекомендации по использованию	0 / 8
<input type="checkbox"/> Количество групп безопасности	0 / 10

Источник: материалы эксперт

Настройка yandex provider для авторизации в YC

- 1 Установите [yc-tools](#)
- 2 Получите [OAuth-токен](#). Срок его жизни — 1 год, используется для первичной настройки. Можно перевыпустить
- 3 Получите [идентификатор облака](#)
- 4 Получите [идентификатор каталога](#)
- 5 Создайте [профиль](#) для yc-tools
- 6 Используйте команду **yc iam create-token**, чтобы получить IAM-токен (срок жизни — 12 часов)

Аутентификация для Terraform provider на примере Yandex

Используйте полученные данные в блоке **provider {..}**

```
provider "yandex" {  
  #Небезопасный способ, только для изучения!  
  token      = "<iam-token или OAuth-token>"  
  cloud_id   = "<идентификатор_облака>"  
  folder_id  = "<идентификатор_каталога>"  
  zone       = "ru-central1-a"  
}
```

Безопасность токенов доступа

- **Никогда не сохраняйте секреты в коде Terraform или в открытом виде в конфигурационных файлах.**
Используйте переменные окружения или файлы конфигурации, которые хранятся в защищённом месте, или добавляйте чувствительные файлы в **.gitignore**
- Используйте отдельные токены для каждого проекта или окружения, чтобы ограничить их область действия и снизить риск компрометации
- Ограничьте права доступа для токенов: они должны быть на минимально необходимом уровне, чтобы предотвратить несанкционированный доступ к вашим ресурсам
- Используйте инструменты управления доступом, такие как Identity and Access Management (IAM) в Yandex Cloud, чтобы доступ к токенам был только у уполномоченных пользователей
- Периодически обновляйте токены, чтобы их нельзя было использовать после истечения срока действия
- Используйте механизмы мониторинга и журналирования, чтобы обнаруживать и отслеживать необычную активность в вашей инфраструктуре

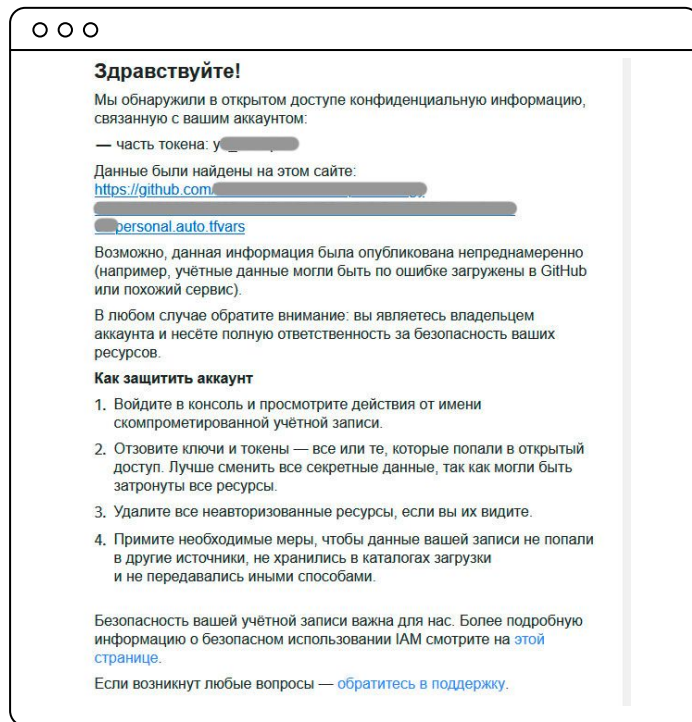
Обработка секретов, попавших в открытый доступ

Yandex Cloud ищет такие типы секретов в открытых источниках:

- API-ключи
- IAM Cookies
- IAM-токены
- статические ключи доступа
- OAuth-токены

Yandex Cloud подключён к программам поиска секретов:

- GitHub Secret scanning partner program
- GitLab Secret Detection
- поисковый индекс Яндекс





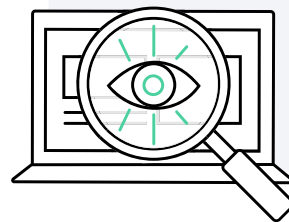
Хардкод (англ. hardcoding) — это нежелательная практика в программировании, когда значения переменных и констант вставляются непосредственно в исходный код программы.

В конечном итоге это всегда приводит к усложнению поддержки кода в больших проектах.

Использование хардкода возможно на этапе испытаний, т. е. MVP (minimum viable product, минимально жизнеспособный продукт)

Демонстрация работы

Создание ресурсов в УС с помощью Terraform без использования input variables (хардкод)



[Ссылка](#)

Переменные в Terraform



3



Input variables (входящие переменные)
позволяют задавать параметры для значений
аргументов извне при выполнении кода.

С **input variables** код становится более гибким и настраиваемым,
легче переносится между разными средами

Блок input variables

Объявляются блоком кода **variable "<уникальное_имя>" {..}**, содержащим:

- [type](#) — тип переменной (опционально)
- [description](#) — назначение (опционально)
- [default](#) — значение по умолчанию (опционально)
- иные опциональные параметры: [validation](#), [sensitive](#), [nullable](#)

```
variable "image_name" {  
    type          = "string"  
    description   = "ubuntu release name"  
    default       = "ubuntu-2004-lts"  
}
```

Переменная без default значения: **variable "image_name" {}**

Sensitive value

В Terraform можно использовать sensitive-свойство для переменных, которые содержат конфиденциальную информацию — пароли или токены.

Значение этой переменной будет скрыто в выводе команды terraform plan/apply/output/console. Такая практика необходима при CI/CD Pipeline

```
variable "database_password" {  
  type      = string  
  sensitive = true  
}
```

```
+ resource "db_instance" "example" {  
  # ...  
  password = (sensitive value)  
  # ...  
}
```

Обращение к input variables

Использование **input variable** в коде проекта: **var.<имя_переменной>**

```
data "yandex_compute_image" "ubuntu-2004-lts" {  
  family = var.image_name  
}
```

Переменные окружения TF_VAR_

Формат: `export TF_VAR_image_name = "ubuntu-2004-lts"`

Префикс TF_VAR_ отбрасывается

Безопасная аутентификация для `yandex provider`:

содержимое `~/.bashrc`

```
export TF_VAR_yc_token = $(yc iam create-token)
```

код terraform:

```
provider "yandex" {  
  token      = var.yc_token  
  cloud_id   = var.yc_cloud_id  
  folder_id  = var.yc_folder_id  
  zone       = "ru-central1-a"  
}
```

Переменные окружения можно использовать для изменения некоторых параметров работы Terraform.
Например, для детализации логов работы: **`export TF_LOG=trace`**

Файлы .tfvars

Terraform загружает переменные из tfvars-файлов модуля:

- **terraform.tfvars** — файл по умолчанию
- ***.auto.tfvars** — именованные файлы

Любые другие tfvars-файлы игнорируются.

Содержат пары «ключ — значение» переменных, например:

```
instance_type = "b2.micro"
region        = "ru-central1"
```

Чтобы **безопасно** записать аутентификационные данные, например, в файл `personal.auto.tfvars`, **добавьте его в `.gitignore`**

Файлы .tfvars

Можно загрузить дополнительные файлы переменных, указав путь с помощью одного или нескольких флагов **-var-file**.

Таким образом можно подгружать переменные для разных окружений

```
terraform apply -var-file=./develop/env.tfvars
```

```
terraform apply -var-file=./prod/env.tfvars -var-file=/prod/additional.tfvars
```


Порядок загрузки значений переменных

- 1 Считывание default-значения из блока variable
- 2 Считывание из environment переменных `export TF_VAR_image_name="ubuntu-2004-lts"`
- 3 Считывание переменных из файла terraform.tfvars (если существует)
- 4 Считывание переменных из автоматически загружаемых файлов .tfvars
- 5 Считывание из командной строки `terraform apply -var="image_name=ubuntu-2004-lts"`

Порядок загрузки значений переменных

Если переменная не определена, Terraform запросит её в командной строке.

Если одна и та же переменная определена разными методами, используется последнее в порядке приоритета загрузки значение. Исключение — переменные типа map, их значения объединяются

Базовые типы переменных

string	Последовательность символов в кавычках : "hello"
number	Целые числа и десятичные дроби без кавычек : 10
bool	Логическое значение без кавычек : true или false
null	Несуществующее (незаданное) значение без кавычек
any	Допускается любой тип данных, значение по умолчанию

Комплексные типы переменных. Collections

1

```
list(<VAR_TYPE>)
```

Список — **индексированный, упорядоченный** набор значений **одинакового** типа.

Пример:

```
list(string)=["abc", "123"];  
list(number)=[1, 2, 5]
```

Обращение к индексу списка:

```
<тип переменной>.<имя переменной>[индекс]
```

Обращение к индексу списка:

```
var.list[0]
```

Комплексные типы переменных. Collections

2

```
map(<VAR_TYPE>)
```

Набор уникальных ключей = строковое значение.

Пример:

```
dict = { first = "A", second = "B" }
```

Обращение к элементу словаря:

```
<тип переменной>.<имя переменной>.<ключ>
```

Пример:

```
var.dict.first или var.dict["first"]
```

Комплексные типы переменных. Collections

3

```
set(<VAR_TYPE>)
```

Неиндексированный набор **уникальных** значений одинакового **типа**.

Примеры:

```
set(string) = ["abc", "123"] set(number) = [1, 2, 3]
```

```
set(list) = [ ["abc", "123"], [1, 2, 3] ]
```

Операция преобразования **list** -> **set** удаляет дубликаты

Комплексные типы переменных. Collections

4

```
tuple([<VAR_TYPE>, ...])
```

Неизменяемый, **индексированный** список значений любого типа, соответствующих структуре переменной.

Пример:

```
tuple([string, number, bool])
```

```
default = ["abc", 123, false]
```

Комплексные типы переменных. Collections

5

```
object({<ATTR_NAME>=<VAR_TYPE>, ... }
```

Неизменяемый набор уникальных «ключ = любой тип данных», соответствующих структуре переменной.

Пример:

```
object({ name = string, age = number, married = bool})  
  
default = { name = "John Connor", age = 15, married = false}
```

Возможны ещё более сложные, многократно вложенные комбинации. В таких случаях часто используют тип **any**:

```
map(object(list(string))) или map(any)
```




Любой объект при создании сохраняет в State значения своей конфигурации.

С помощью блока **output** можно извлечь эти значения или их часть в **output-переменную** для последующего использования при создании ресурсов

Блок output variables

Объявляется блоком **output** "" {...}, содержащим:

- **value** — значение переменной, которое может быть string, list или map

После применения конфигурации Terraform отобразит в консоли значения всех объявленных **outputs в root module**.

Так же действует и **terraform output**. Его вывод можно использовать в скриптах автоматизации

Пример:

```
output "vm_external_ip_address" {  
    value =  
yandex_compute_instance.vm.*.network_interface.[0].nat_ip_address  
    description = "vm external ip"  
}
```

Организация кода проекта

Для удобства разработки и поддержки кода принято группировать блоки по типам и назначению в отдельные файлы ***.tf**. Как именно — решает команда, но технически весь код можно сложить в один файл

outputs.tf	Блоки output
locals.tf	Локальные переменные
providers.tf	Блок провайдеров
main.tf	Root-модуль
variables.tf	Описание общих переменных
vpc.tf vm.tf	Именованные файлы для ресурсов и переменных
db.auto.tfvars personal.auto.tfvars	Именованные файлы для загрузки переменных. Если они содержат личные токены доступа, добавьте их в .gitignore



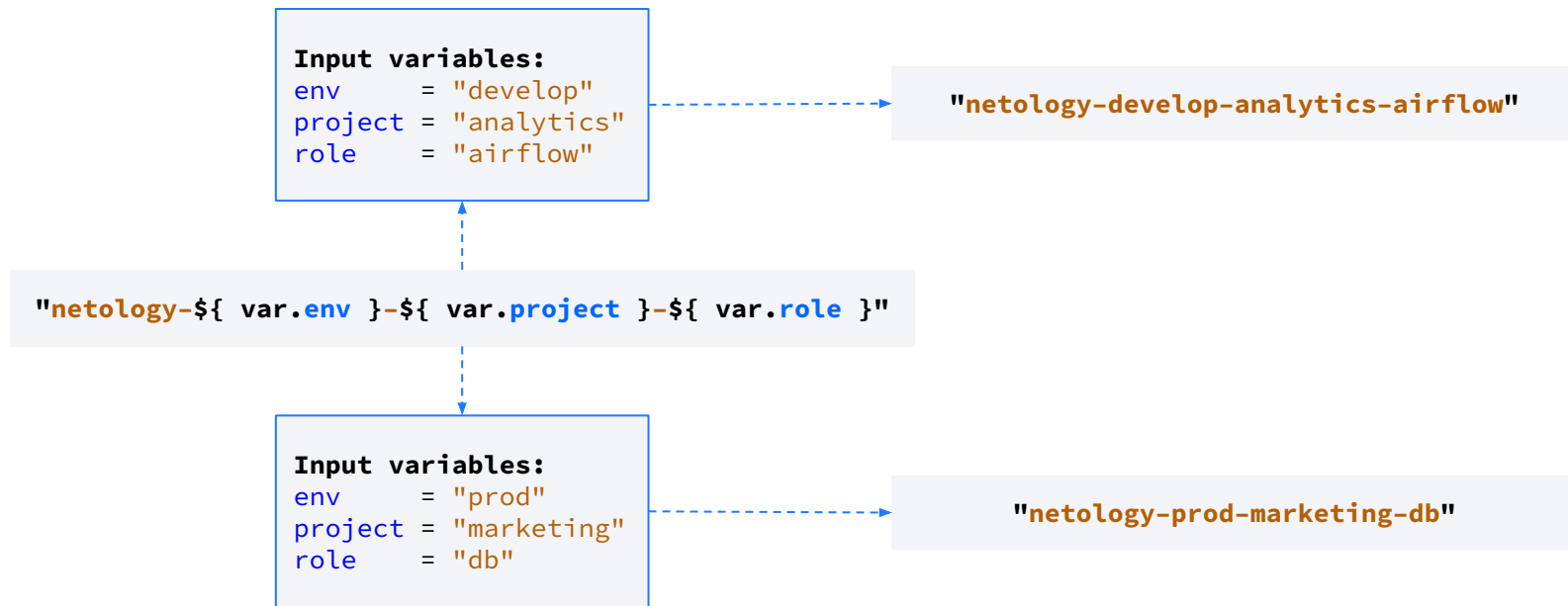
В программировании интерполяция — это процесс замены или вставки значений переменных в строку-шаблон.

В результате получается новая строка, которую можно использовать при дальнейшем выполнении программы

String interpolation

С помощью конструкции `"${ ... }"` можно составить шаблон строки с использованием условий, функций и выражений.

Пример универсального шаблона названия виртуальной машины:



Input variables

Input variables не могут принимать в качестве значения вычисляемое выражение, т. е. не могут содержать переменные, функции и прочие выражения.

Недопустимый пример:

```
variable "bad_example" { default = "${ var.env }-${ var.project }" }
```



Локальные значения позволяют создавать вычисляемые значения на основе других переменных или функций. Например, вы можете создать локальное значение, которое будет объединять значения двух входящих переменных

Блок locals

Локальные переменные объявляются блоком **locals { .. }**, содержащим **одну** или **несколько** переменных в виде «ключ = значение».

```
locals {  
  name = "${ var.env }-${ var.project }"  
  test = "123"  
}
```

Обращение к local-переменной в модуле:

local.<Имя Переменной>

```
local.name  
local.test
```




Количество блоков `locals` не ограничено, но если использовать слишком много локальных значений, код может стать менее читаемым и понятным для других разработчиков, а спустя некоторое время — и для вас самих.

Добавляйте локальные значения, только когда это действительно необходимо.

Комментируйте свой код и объясняйте, как и зачем вы используете локальные значения

Интерактивная консоль Terraform

Команда **terraform console** открывает интерактивную оболочку.

Она позволяет тестировать Terraform-выражения в интерактивном режиме, используя текущее состояние инфраструктуры и переменные, **определённые** в вашем коде Terraform.

Создавать новые переменные в консоли нельзя.

Знак > во всех последующих примерах будет означать работу с консолью

Примеры **terraform console**:

```
> 5+5
```

```
10
```

```
> [ 1,2,3 ]
```

```
[
```

```
  1,
```

```
  2,
```

```
  3,
```

```
]
```

```
> { a=1, b=2 }
```

```
{
```

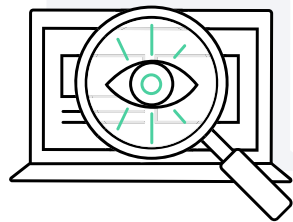
```
  "a" = 1
```

```
  "b" = 2
```

```
}
```

Демонстрация работы

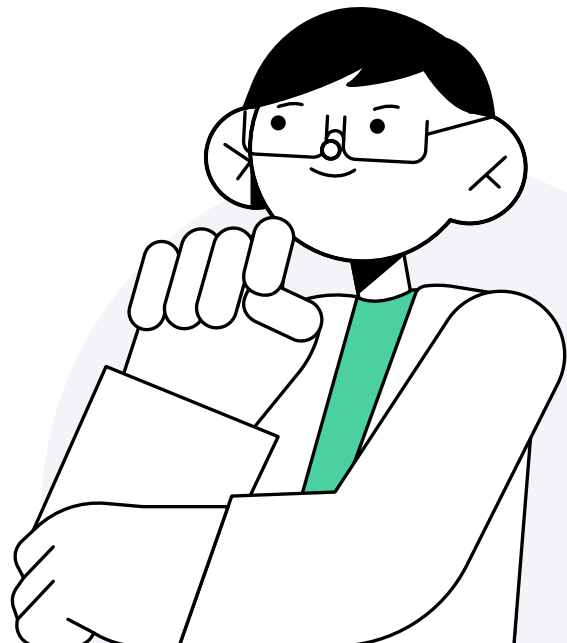
Интерактивная консоль Terraform



Разбор типов переменных на практике

Итоги занятия

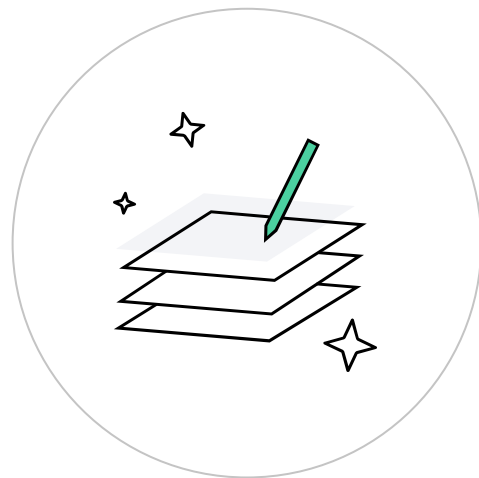
- 1 Рассмотрели виды облачных вычислений
- 2 Научились создавать ресурсы в Yandex Cloud с помощью Terraform
- 3 Изучили типы переменных в Terraform



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 ставится после того, как приняты все задачи



Дополнительные материалы

Документация:

- [input variables](#)
- [outputs](#)
- [local values](#)



**Задавайте
вопросы и пишите
отзыв о лекции**

