



Terraform. But why? But how?

Ervin Weber

Цель №1

Если ты принимаешь участие в создании продукта у которого есть API - подумай о создании Terraform Provider для вашего API

Цель №2

Убедись что у человека отвечающего за твою инфраструктуру есть история изменений не только самого кластера в котором работают сервисы но и скажем DNS Zone, Hard Disk Backups и прочего что обычно нужно чтобы запустить и адресовать сам кластер...

Внимание! далее следует история которая возможно была или даже не была в далеком прошлом, любые совпадения с реальностью случайны... Пожалуйста, не делайте по-старинке =)

Дано: 2 сервера приложений + 1 база

- Все делаем ручками на сервере и радуемся жизни
- Задачи решаются быстро, все рады
- Единственный в команде человек о котором ничего нет в системе контроля версий.
- Если попасть под автобус - потомки не разберутся что и почему



Дано: 2 сервера приложений + 1 база

- Делаем новое хранилище в системе версионирования кода туда кладем пару текстовых файлов “в свободной форме”:
 - Список серверов (адрес, контакт провайдера и кто может добавить новый ключ доступа в случае необходимости)
 - “Сборник ссылок советов и костылей по тонкой настройке сервисов”
 - “Файл-почемучка” ссылается на поставленные задачи и немного помогает разобраться “зачем”
- Теперь на каждую выполненную задачу есть “некий след”™



Дано: 3 сервера приложений + 1 база

- “Разок не копи-паста” перестало работать: теперь есть сервер на котором пишется и тестируется под нагрузкой конфигурация и ее нужно скопировать на еще два сервера.
- “Файл почемучка” начинает обрастать коллекцией команд которые можно просто запустить на всех серверах и получить одинаковый результат
- Становится лениво делать однообразные действия руками и возникает большое желание что-то как то автоматизировать...



Автоматизация 10 лет назад

- Уже есть ansible, capistrano, chef, puppet.
 - Каждый мне чем-то не понравился....
- УРА! Мы пишем свою автоматизацию!!!!
- Первая версия: только настройка сервисов
 - Написано на любимом руби, конфигурация json
 - На входе “Роли” (скрипты) “Серверы” (узлы на которые ставить роли).
 - Внутри
 - решение зависимостей “ролей”
 - Обнаружение ручных изменений, внесенных оператором прямо на сервер в обход автоматизации
 - Тесная интеграция с системой контроля версий
 - И все могло бы быть очень хорошо.....




Облачный провайдер - машин становится много


- Добавление новых машин делается ручками в интерфейсе облачных услуг
- Затем машину добавляют в настройщик сервисов и назначают роли
- Это тоже можно и нужно автоматизировать
 - Пишем костыль который на входе берет название машины и роли которые мы хотим на ней видеть. Оно общается с программным интерфейсом облака и что-то там делает потом добавляет новый адрес в наш самописный настройщик сервисов и он делает свою работу
 - Программный интерфейс облака быстро развивается, нам нужно больше функций и постепенно количество кода “костыля” растет слишком быстро
 - “А давайте отдадим костыль сторонним разработчикам”



Так в мою жизнь и пришел terraform (2019г.)

- Альтернатив по стандартному функционалу нет.
 - У поставщиков облачных услуг есть альтернативы, но только для их сервиса
 - Декларируем все компоненты которые нам нужны в облаке
 - Просим terraform чтобы после создания инфраструктуры передал все новые узлы в наш старый настройщик и чтобы он настроил сервисы на узлах
 - profit?!
- 

Зачем нам все это?

- Автоматизация лени
 - Воспроизводимость и история изменений
 - Создал инфраструктуру - потестировал - снес (выгодно при поминутной оплате)
 - Стандартизация: купил инженера а он уже работал с этим инструментом
 - Делегирование - команда разработчиков может сама создать или исправить описание своей инфраструктуры
 - Сертификация и требования безопасности: “вот этот наш шаблон отвечает требованиям SOC 2 Framework”
 - Людям не нужен “доступ Админа” каждый день.
- 

terraform

1.2.4 (June 29, 2022)

```
curl --silent --fail-with-body
https://checkpoint-api.hashicorp.com/v1/check/terraform
| jq .

{
  "product": "terraform",
  "current_version": "1.2.4",
  "current_release": 1656525661,
  "current_download_url":
  "https://releases.hashicorp.com/terraform/1.2.4",
  "current_changelog_url":
  "https://github.com/hashicorp/terraform/blob/v1.2.4/CHANGELOG.md",
  "project_website": "https://www.terraform.io",
  "alerts": []
}
```

Краткое содержание этой серии

- Компоненты terraform: ядро и расширения
- Пиши! - Планируй! - Применяй!
 - Конфигурация - HCL 2.0 (а может все-таки JSON?)
 - У терраформа есть какой-то план и он будет его придерживаться
 - Иголлка в яйце - если вся известная вселенная это один файл - то где его хранить
- Разработка дорожает, как прокормить разработчиков.

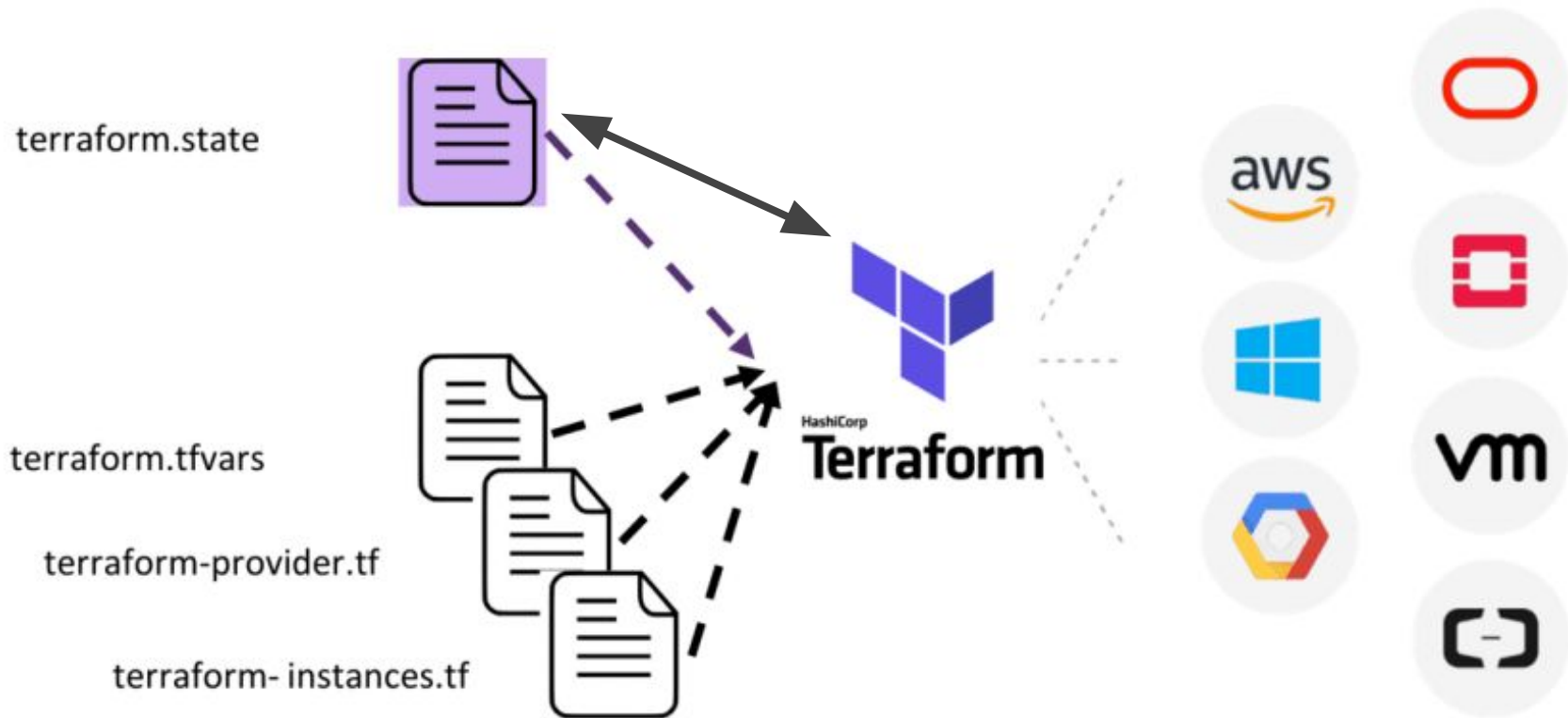


Terraform

Приложение для командной строки, позволяющее декларировать желаемое состояние инфраструктуры и приводящее реальность к требуемому.

Из коробки ничего создавать не умеет, но имеет расширения для всех популярных API.





Как это работает?

- Ядро Terraform отвечает за:
 - Чтение и интерполяцию конфигурации и всех модулей
 - Запись и чтение “состояния” в разных интересных местах
 - Создание графа зависимостей
 - Выполнение плана
 - Общение со всеми плагинами по gRPC



Кто такой terraform provider

- Расширение написанное на golang
- Может быть локальным, из приватного репозитория или из “магазина”
- Добавляет новые типы ресурсов или источники данных
- Проверяет правильность желаемой конфигурации ресурсов
- Вычисляет необходимые изменения в ресурсах
- Делает всю работу в стороннем сервисе
- Создает информацию о ресурсах для файла состояния



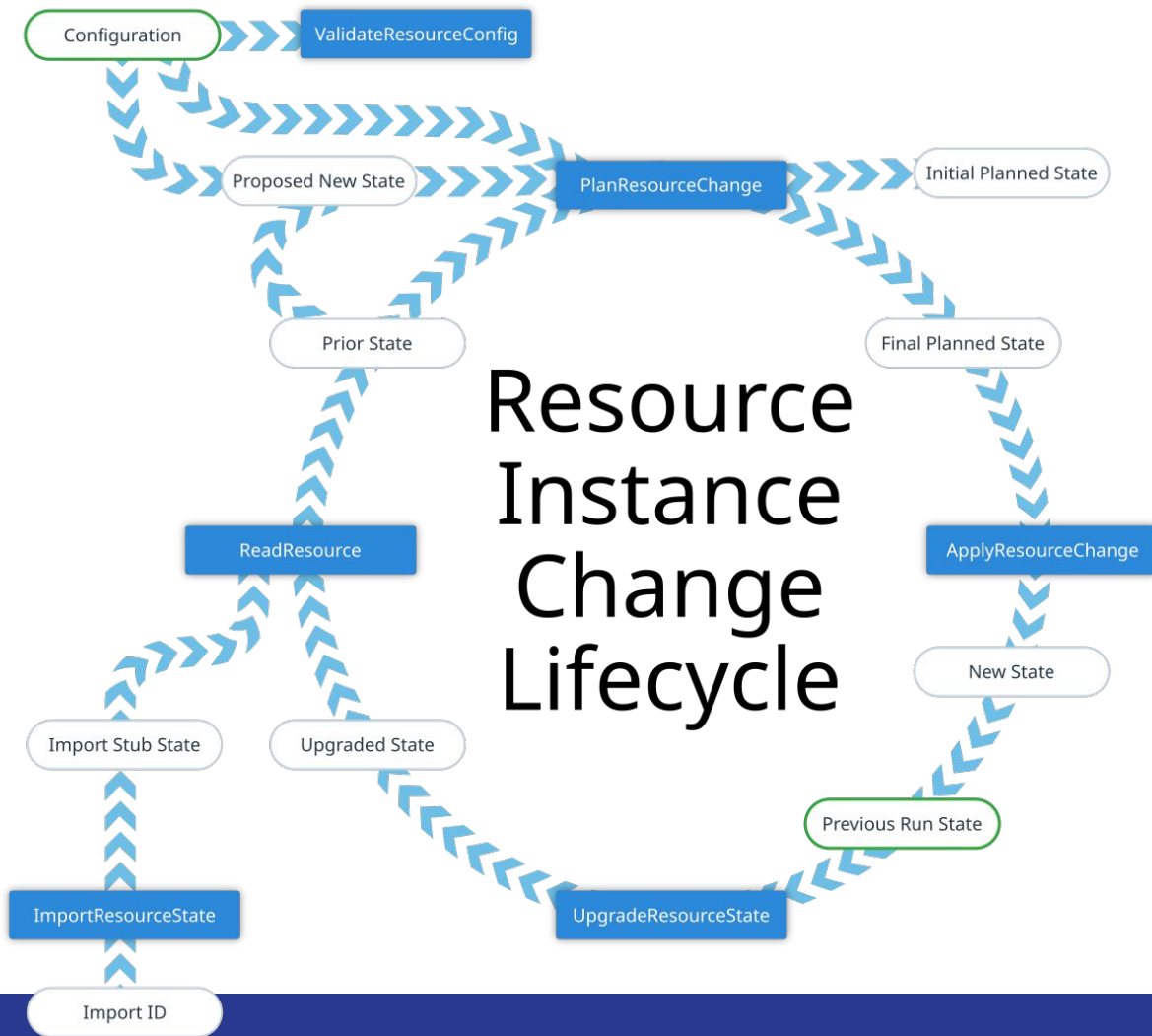
Official



Verified

Community

Archived



Описываем инфраструктуру - модуль

- “Модуль” это все файлы .tf (или .tf.json) в одной папке
- Модуль с которого начинается работа “корневой” но можно “вызывать” другие модули
- Модули пишутся на языке HCL в который добавлены расширения terraform - набор “блоков верхнего уровня” и набор полезных “пользовательских” функций.




HashiCorp configuration language

v2.13.0 (June 22, 2022)

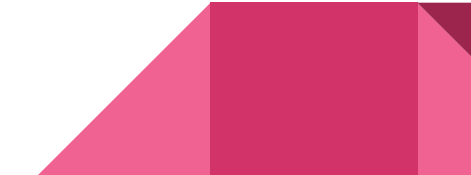
Но на самом деле версия языка
встроена в версию terraform

HashiCorp configuration language

<https://github.com/hashicorp/hcl>

- Придуман для людей, но для машин есть возможность писать и читать JSON
 - Атрибуты (структура, “имя” = “значение”)
 - Блоки (группировка, “тип” “этикетка1” “этикетка2” { “атттрибуты или блоки” })
 - Строчные и блочные комментарии
 - Интерполяция внутри строк
 - Текстовые шаблоны (интерполяция + if + for)
 - Простая арифметика для выражений
 - Возможность добавлять свои функции в своей программе.
- 

```
resource "aws_vpc" "main" {  
    cidr_block = var.base_cidr_block  
}  
/*  
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>"  
{  
    # Block body  
    <IDENTIFIER> = <EXPRESSION> # Argument  
}  
*/
```



Выражение, блок
или ошибка?

A meme featuring Steve Harvey on the set of the game show Jeopardy!. He is wearing his signature tan suit and has a mustache. He has a wide-eyed, intense expression, looking slightly to his left. Overlaid on the bottom half of the image are four blue, rounded rectangular buttons with white text, arranged in two rows. The top button contains the text 'a = { b = c }'. The bottom row contains four buttons labeled 'A:', 'B:', 'C:', and 'D:' followed by the words 'Expression', 'Error', 'Block', and 'Unblock' respectively. In the bottom left corner, there is a small white watermark that reads 'imgflip.com'.

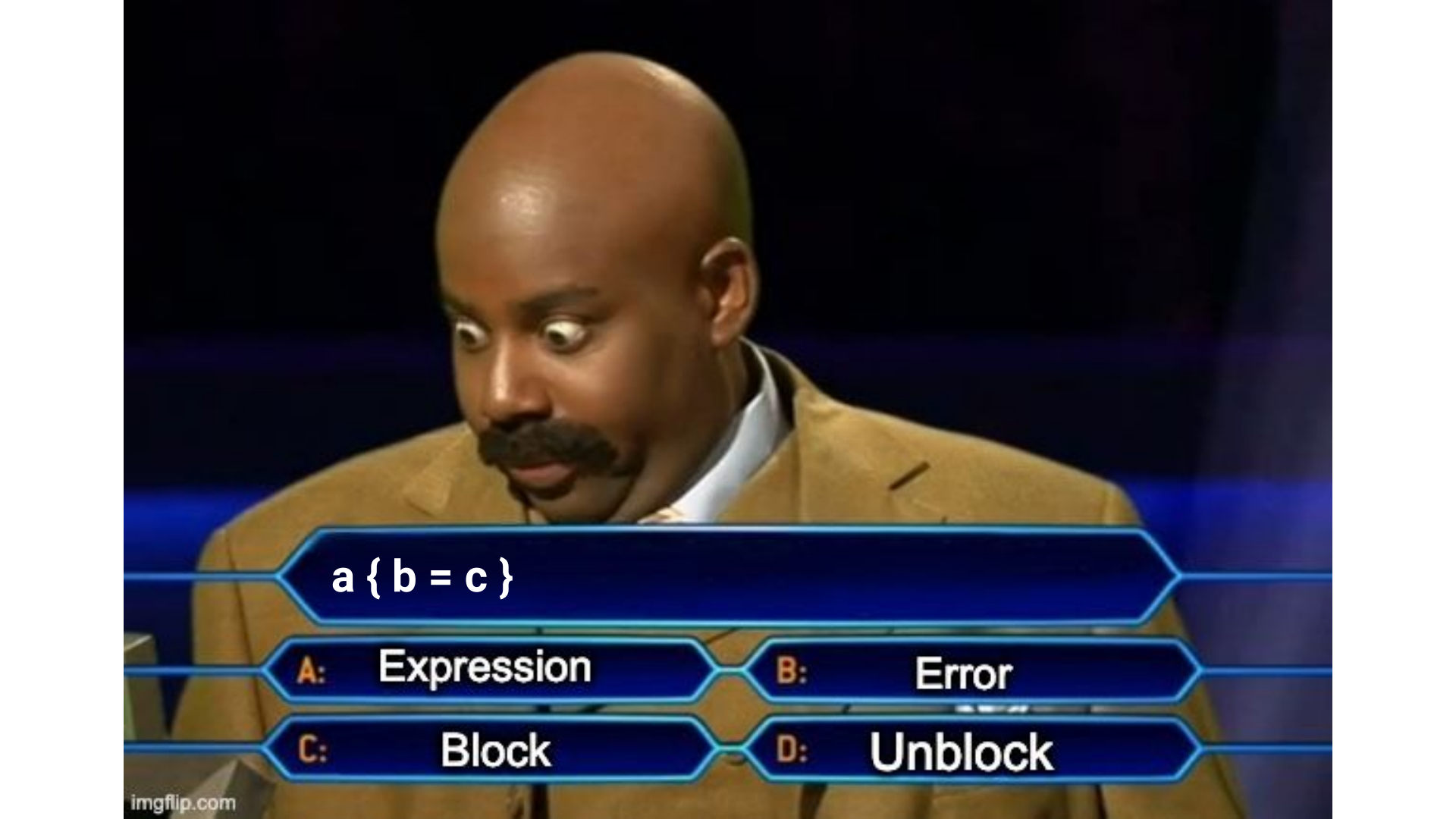
$a = \{ b = c \}$

A: Expression

B: Error

C: Block

D: Unblock

A meme featuring Steve Harvey on the set of the game show Jeopardy!. He is wearing a tan suit and has a mustache. He has a wide-eyed, questioning expression on his face, looking slightly to the left. Overlaid on the bottom half of the image are four blue, rounded rectangular buttons with white text, arranged in two rows of two. The top button contains the code snippet 'a { b = c }'. The bottom row contains four buttons labeled 'A:', 'B:', 'C:', and 'D:' followed by the words 'Expression', 'Error', 'Block', and 'Unblock' respectively. In the bottom left corner, there is a small white watermark that reads 'imgflip.com'.

`a { b = c }`

A: Expression

B: Error

C: Block

D: Unblock

A meme featuring Steve Harvey on the set of the game show Jeopardy!. He is wearing his signature tan suit and has a mustache. He has a wide-eyed, questioning expression on his face, looking slightly to the left. Overlaid on the bottom half of the image are four blue, rounded rectangular buttons with white text, arranged in two rows of two. The top button contains the text 'a b { c = d }'. The bottom row contains four buttons labeled 'A:', 'B:', 'C:', and 'D:' followed by the words 'Expression', 'Error', 'Block', and 'Unblock' respectively. In the bottom left corner, there is a small white watermark that reads 'imgflip.com'.

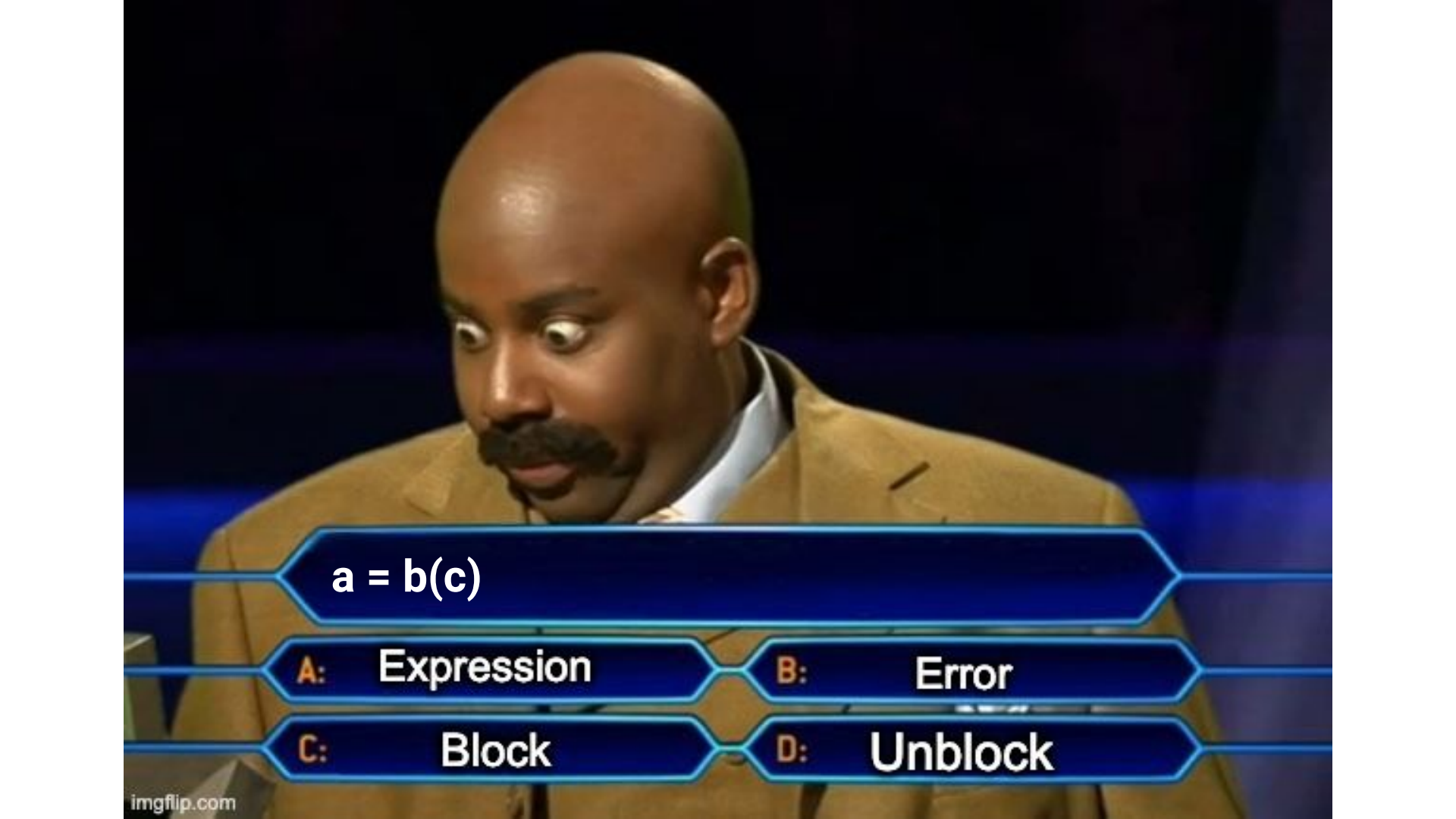
$a\ b\ \{c = d\}$

A: Expression

B: Error

C: Block

D: Unblock

A meme featuring Steve Harvey on the set of the game show Jeopardy!. He is wearing his signature tan suit and has a mustache. He has a wide-eyed, intense expression, looking slightly to his left. Overlaid on the bottom half of the image are four blue, rounded rectangular buttons with white text, arranged in two rows. The top button contains the text 'a = b(c)'. The bottom row contains four buttons labeled 'A:', 'B:', 'C:', and 'D:' followed by the words 'Expression', 'Error', 'Block', and 'Unblock' respectively. In the bottom left corner, there is a small white watermark that reads 'imgflip.com'.

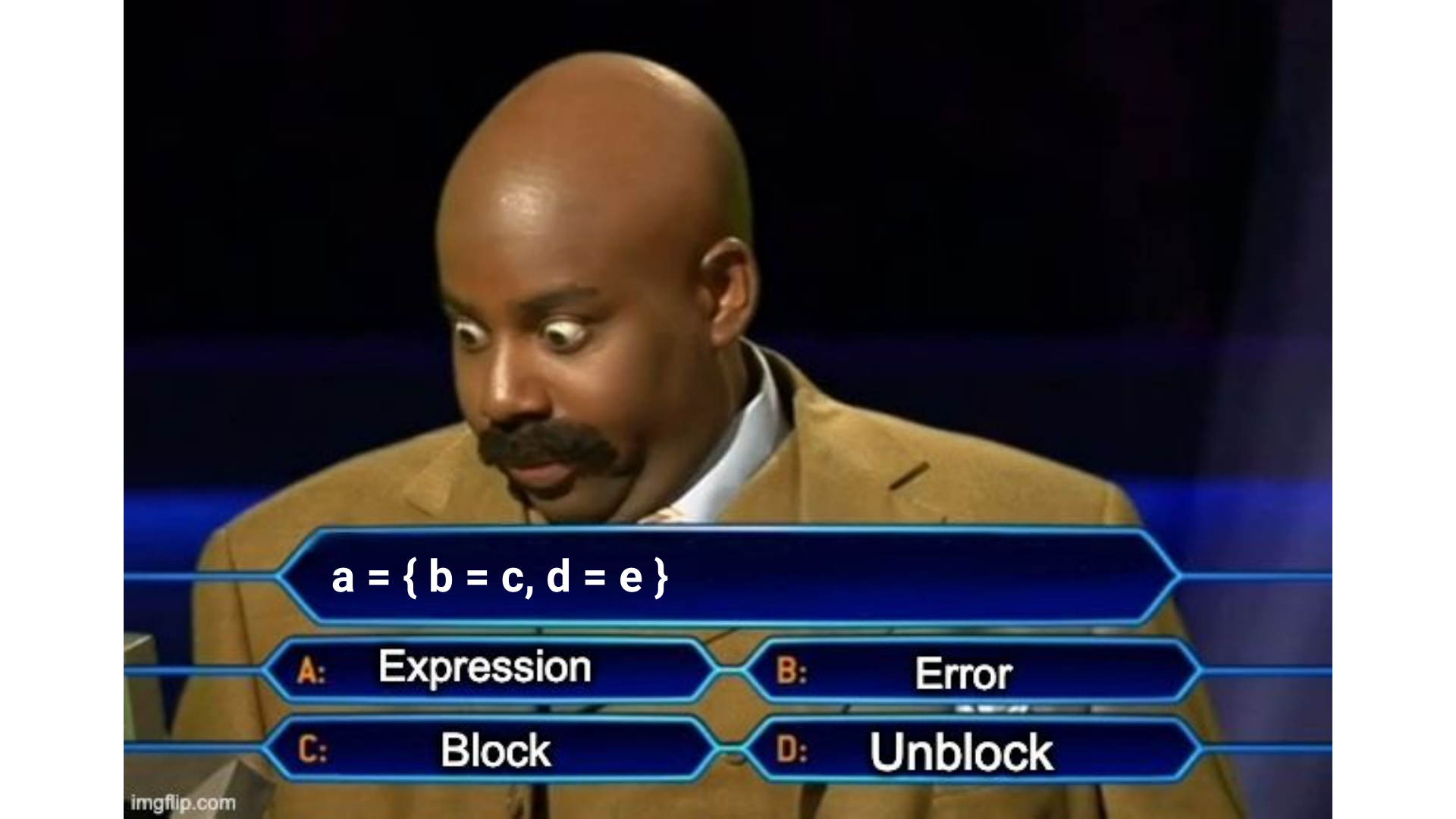
$a = b(c)$

A: Expression

B: Error

C: Block

D: Unblock

A meme featuring Steve Harvey on the set of the game show Jeopardy!. He is wearing a tan suit and has a mustache. He has a wide-eyed, questioning expression on his face, looking slightly to the left. Overlaid on the bottom half of the image are four blue, rounded rectangular buttons with white text, arranged in two rows. The top button contains the text 'a = { b = c, d = e }'. The bottom row contains four buttons: 'A: Expression', 'B: Error', 'C: Block', and 'D: Unblock'. The background is dark and out of focus, typical of a game show set.

$a = \{ b = c, d = e \}$

A: Expression

B: Error

C: Block

D: Unblock

HCL - Операторы и разделители

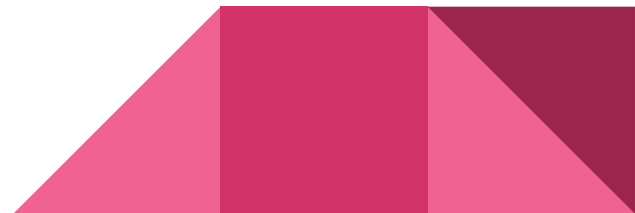
+ && == < : { [(\${

- || != > ? }]) %{

* ! <= = .

/ >= => ,

% ...



HCL - Формальный синтаксис

`ConfigFile = Body;`

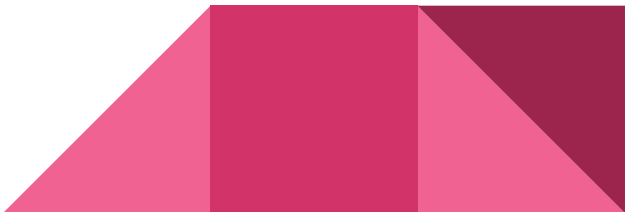
`Body = (Attribute | Block | OneLineBlock)*;`

`Attribute = Identifier "=" Expression Newline;`

`Block = Identifier (StringLit|Identifier)* "{" Newline Body "}" Newline;`

`OneLineBlock = Identifier (StringLit|Identifier)* "{" (Identifier "=" Expression)? "}" Newline;`

`Conditional = Expression "?" Expression ":" Expression;`



HCL - For loop

`[for v in ["a", "b"]: v]` результат `["a", "b"]`

`[for k, v in ["a", "b"]: k]` результат `[0, 1]`

`{for k, v in ["a", "b"]: v => k}` результат `{a = 0, b = 1}`

`{for k, v in ["a", "a", "b"]: v => k}` ошибка такой объект
НЕВОЗМОЖЕН


`{for k, v in ["a", "a", "b"]: v => k...}` результат `{a = [0, 1],
b = [2]}`

`[for k, v in ["a", "b", "c"]: v if k < 2]` результат `["a", "b"]`

HCL - “SPLAT” оператор (ну и как это по-русски?)

`[for v in tuple: v.foo.bar]` короче записать так:
`tuple.*.foo.bar` (возвращает коллекцию значений “bar” со всех элементов)

`[for v in tuple: v.foo.bars[0]]` короче записать так:
`tuple[*].foo.bars[0]` (возвращает коллекцию из первых элементов bar)





Demo 1:

Hello devclub

Local Terraform Lifecycle

<ОПИСЫВАЕМ СОСТОЯНИЕ>

```
$ terraform init
```

```
$ terraform plan
```

```
$ terraform apply
```

```
$ terraform destroy
```



Local Terraform Lifecycle

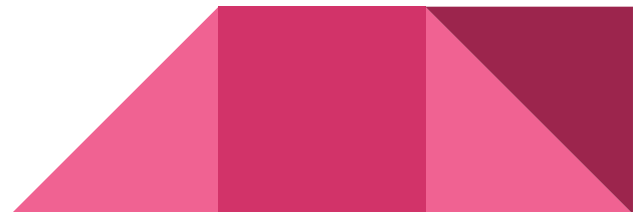
<ОПИСЫВАЕМ СОСТОЯНИЕ>

\$ terraform init

\$ terraform plan

\$ terraform apply

\$ terraform destroy

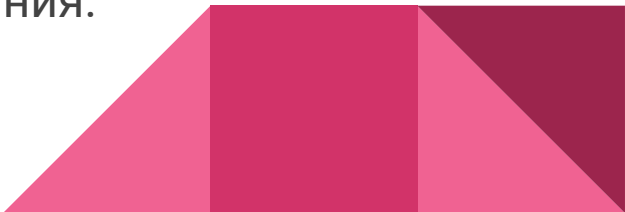


terraform init

Готовит систему к работе.

На входе: папка с желаемой конфигурацией

На выходе готовая к другим командам папка с конфигурацией и зависимостями

- Скачивает расширения
 - Скачивает дополнительные модули
 - Опционально проверяет “связь” с файлом состояния.
 - Dependency lock-file
- 

terraform plan

Опрашивает через расширение поставщика услуг объекты перечисленные в файле состояния и их текущее состояние, сравнивает с требуемым в конфигурации и создает граф нужных изменений. Умеет создавать план для удаления всего.

На входе:

- готовая к другим командам папка с конфигурацией и зависимостями
- Значения для переменных корневого модуля
- “файл состояния”

На выходе:

- “План действий” которые нужно предпринять чтобы реальность приблизилась к желаемому

terraform apply

Применяет план действий на реальную инфраструктуру.

На входе: план действий

На выходе:

- Измененная реальность
- Обновленный файл состояния



terraform destroy

Удаляет все известное.

На входе: план удаления

На выходе:


- Опустевший файл состояния
- Реально удаленная инфраструктура

There is no **terraform undo**



Где запускать terraform

Задача:

- не хранить секретные значения переменных в git
 - ограничить доступ к файлу состояния (там тоже можно увидеть все секреты)
 - хранение ключей высокого уровня доступа к инфраструктуре
-
- Локально / CI/CD / своя установка <https://www.runatlantis.io/> или подобного сервиса
 - HashiCorp Terraform cloud - terraform as a service
- 

Где хранить файл состояния

- Замок на одновременные запросы
 - Надежность и резервное копирование
 - Безопасность - файл содержит все секреты
-
- Local / ~~remote~~ / ~~artifactory~~ / ~~azurerm~~ / consul / cos / ~~eted~~ / ~~etcdv3~~ / gcs / http / Kubernetes / ~~manta~~ / oss / pg / s3 / ~~swift~~
 - Мой любимый вариант: gcs



CI/CD

```
export TF_IN_AUTOMATION=true
```

```
terraform init -input=false
```

```
terraform plan -out=tfplan -input=false
```

```
# visualize tfplan for reviewers and obtain consent to  
continue
```

```
terraform apply -input=false -auto-approve tfplan
```



Основные блоки
специфичные для
terraform...

Описываем инфраструктуру - блоки

terraform { }

variable "name" { type = string }

output "name" { value = expression }

locals { k = v }

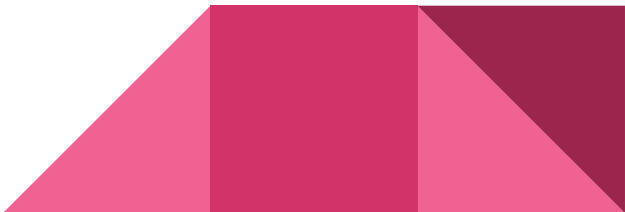
module "name" { source = uri }

provider "name" { }

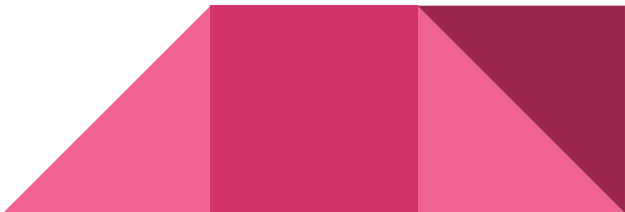
resource "provider_type" "name" { }

data "provider_type" "name" { }


moved { }




```
terraform {  
  cloud { ... } | backend "backend_type" { ... }  
  required_version = ">= 1.2.3"  
  required_providers {  
    devclub = {  
      version = ">= 0.0.0"  
      source = "devclub/data"  
    }  
  }  
  experiments = [name1, name2]  
}
```




```
variable "var_name" {  
    type = string  
    description = "human readable block"  
    sensitive = false  
    nullable = true  
    default = "devclub"  
    validation {  
        condition      = length(var.var_name) > 4 && substr(var.var_name, 0, 3) == "dev"  
        error_message = "The var_name value must be starting with \"dev\"."  
    }  
}
```



```
output "my_result_name" {  
    value = ( expression)  
    description = "документация"  
    precondition {  
        condition      = true  
        error_message = "Что-то пошло совсем не так."  
    }  
    sensitive = false  
    depends_on = [ ... ]  
}
```



```
locals {  
  name1 = "Клуб!"  
  (expression) = 42  
  name3 = {  
    key1 = [ 1,2,3,7 ]  
    key2 = {  
      Key3 = "${local.name1}"  
    }  
  }  
}
```





```
module "consul" {  
    source = "hashicorp/consul/aws"  
    version = "0.0.5"  
    depends_on = [ ... ]  
    count = number | for_each = ( expression )  
    providers = {  
        provider_name = alias | name  
    }  
    depends_on  
}
```




```
provider "my_provider" {  
    configuration1 = "east"  
}  
  
provider "my_provider" {  
    alias  = "west"  
    configuration1 = "west"  
}
```




```
resource "my_provider_specific_resource_name"  
  "resource_name" {  
  
    provider = my_provider  
  
    count = 100 | for_each = (expression)  
  
    depends_on = [ .. ]  
  
    lifecycle {  
  
      ...  
    }  
  }  
}
```



```
resource "my_provider_specific_name" {  
  lifecycle {  
    create_before_destroy = false  
    prevent_destroy = false  
    ignore_changes = [ ] | all  
    # new in 1.2.0 (May 18, 2022)  
    replace_triggered_by = []  
    precondition {}  
    postcondition {}  
  }  
}
```



```
data "my_provider_specific_resource_name" "resource_name" {  
  count = 0 | for_each = ( expression )  
  provider = my_provider  
  depends_on = [ ]  
  lifecycle {  
    precondition { }  
    postcondition { }  
  }  
}
```



Demo 2:

Refactoring Joy :)

```
moved {  
    from = some_resource.old  
    to    = some_resource.old  
}  
  
# Since: 1.1.0 (December 08, 2021)
```





Demo 3:

Simple password rotation

Broke in version 2.3.4

Dennis

Simple password

```
alias tfd='TF_LOG=TRACE terraform '
```

```
alias tfmt='terraform fmt -recursive'
```

```
alias tfp='terraform validate && terraform plan'
```

```
alias tf_moves='terraform validate && terraform plan -no-color | tee plan.txt && (  
grep "\" will be created\"" plan.txt > create.txt; grep "\" will be destroyed\"" plan.txt >  
destroy.txt; sed \"s/ # /moved {\n from = /g\" destroy.txt | sed \"s/ will be  
destroyed/\n to = \n}\n/g \" > move.txt || true)
```

Спасибо за внимание!

<https://www.linkedin.com/in/webervin>
webervin@gmail.com