Принципы организации инфраструктурного кода и работа над инфраструктурой в команде на примере Terraform.

Не забудь включить запись!

План

- Декларативное vs процедурное описание
- Зависимости ресурсов
- Data sources
- Remote state
- Modules
- Продвинутое управление state

Процедурное описание

- Описываем действия по достижению желаемого состояния
- Желаемое состояние достижимо, но сильно зависит от метода использования инструмента
- Отвечает на вопрос "как и что сделать?"

Создание ресурсов

```
$ gcloud compute instances create example-instance-1 example-instance-2 \
    --machine-type f1-micro \
    --zone europe-west1-b \
    --image-family ubuntu-1804-lts \
    --image-project ubuntu-os-cloud

NAME ZONE MACHINE_TYPE PREEMPTIBLE INTERNAL_IP EXTERNAL_IP STATUS
example-instance-1 europe-west1-b f1-micro 10.132.0.3 35.195.239.50 RUNNING
example-instance-2 europe-west1-b f1-micro 10.132.0.2 35.187.163.47 RUNNING
```

Повторное применение команды

```
$ gcloud compute instances create example-instance-1 example-instance-2 \
    --machine-type f1-micro \
    --zone europe-west1-b \
    --image-family ubuntu-1804-lts \
    --image-project ubuntu-os-cloud

ERROR: (gcloud.compute.instances.create) Could not fetch resource:
- The resource 'projects/infra-179014/zones/europe-west1-b/instances/example-instance-1'
already exists
- The resource 'projects/infra-179014/zones/europe-west1-b/instances/example-instance-2'
already exists
```

Удаление

```
$ gcloud compute instances delete example-instance-2

Do you want to continue (Y/n)? y
```

Декларативное описание

- Описываем желаемое состояние
- Инструмент отвечает за приведение инфраструктуры в соответствие с описанием
- Отвечает на вопрос "что должно быть?", а не "как сделать?"

Создание ресурсов

```
resource "google_compute_instance" "app" {
  name         = "example-instance-${count.index + 1}"
  machine_type = "f1-micro"
  count        = 2

  boot_disk {
    initialize_params {
      image = "ubuntu-1804-lts"
    }
  }

  network_interface {
    network       = "default"
    access_config = {}
  }
}
```

Создание ресурсов

```
$ terraform apply -auto-approve=true

google_compute_instance.app.0: Creation complete after 16s (ID: example-instance-1)
google_compute_instance.app.1: Creation complete after 16s (ID: example-instance-2)

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

Повторное применение команды

```
$ terraform apply -auto-approve=true

google_compute_instance.app.1: Refreshing state... (ID: example-instance-2)
google_compute_instance.app.0: Refreshing state... (ID: example-instance-1)

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

Удаление ресурса

```
resource "google_compute_instance" "app" {
  name         = "example-instance-${count.index + 1}"
  machine_type = "f1-micro"
  count        = 1

  boot_disk {
    initialize_params {
      image = "ubuntu-1804-lts"
    }
  }

  network_interface {
    network       = "default"
    access_config = {}
  }
}
```

Говорим инструменту привести в желаемое состояние

```
$ terraform apply -auto-approve=true

google_compute_instance.app.1: Destruction complete after 46s

Apply complete! Resources: 0 added, 0 changed, 1 destroyed.
```

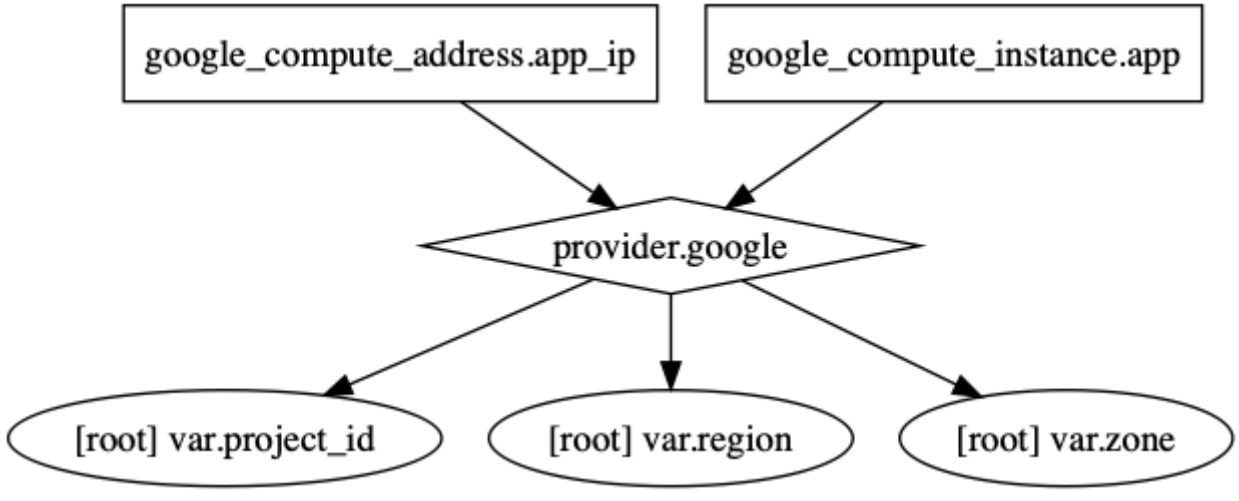
Взаимосвязи ресурсов

Ресурсы независимы

```
resource "google_compute_instance" "app" {
  name         = "example-instance"
  machine_type = "f1-micro"

  boot_disk {
    initialize_params {
      image = "ubuntu-1804-lts"
    }
  }

  network_interface {
    network       = "default"
    access_config = {}
  }
}

resource "google_compute_address" "app_ip" {
  name = "example-ip"
}
```

Ресурсы независимы

```
terraform graph -no-color | grep -v -e 'meta.count-boundary' -e 'provider.google
(close)' | dot -T png >graph.png
```



Независимые ресурсы создаются одновременно

```
$ terraform apply -auto-approve=true

google_compute_address.app_ip: Creating...
 address: "" => "<computed>"
 name: "" => "example-ip"
 self_link: "" => "<computed>"
google_compute_instance.app: Creating...
 boot_disk.#:

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

Неявная зависимость

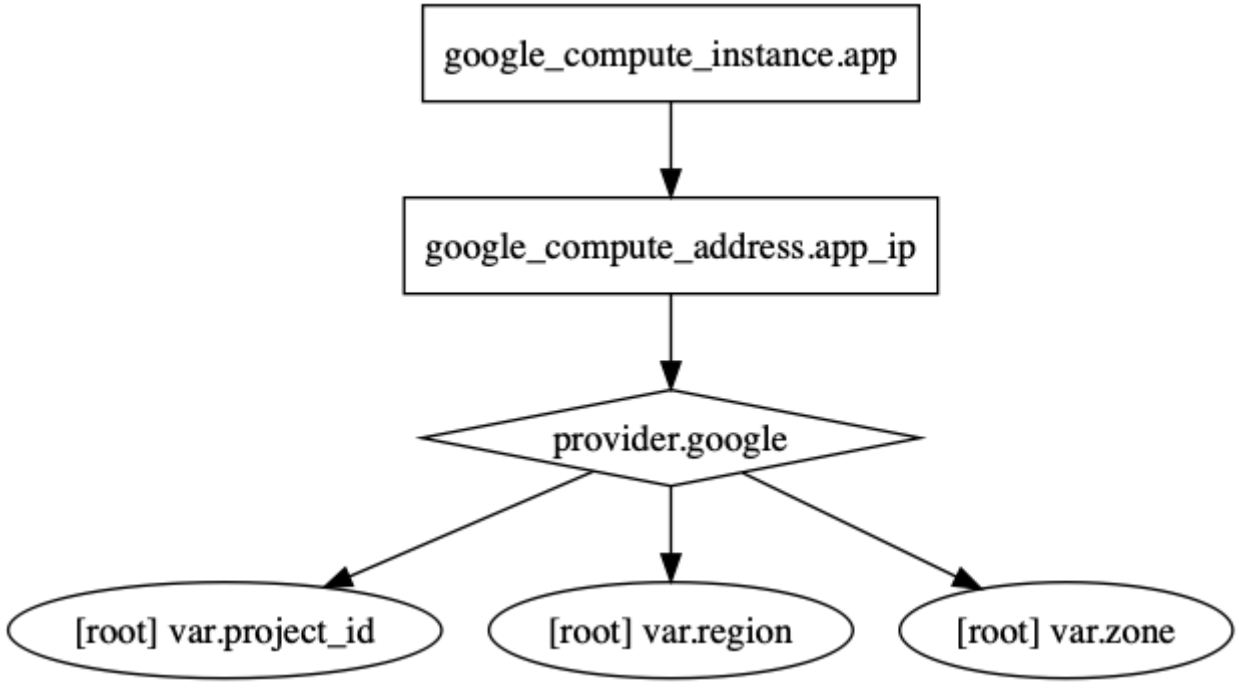
```
resource "google_compute_instance" "app" {
  name         = "example-instance"
  machine_type = "f1-micro"

  boot_disk {
    initialize_params {
      image = "ubuntu-1804-lts"
    }
  }

  network_interface {
    network = "default"

    access_config = {
      nat_ip = "${google_compute_address.app_ip.address}"
    }
  }
}

resource "google_compute_address" "app_ip" {
  name = "example-ip"
}
```

Ссылка на атрибут ресурса

```
"${google_compute_address.app_ip.address}"
```

Неявная зависимость

```
terraform graph -no-color | grep -v -e 'meta.count-boundary' -e 'provider.google
(close)' | dot -T png >graph.png
```



Зависимые ресурсы создаются в соответствии с зависимостями

```
$ terraform apply -auto-approve=true

google_compute_address.app_ip: Creating...
 address: "" => "<computed>"
 name: "" => "example-ip"
 self_link: "" => "<computed>"
google_compute_address.app_ip: Creation complete after 13s (ID: example-ip)
google_compute_instance.app: Creating...
 boot_disk.#:
```

Зависимые ресурсы удаляются в соответствии с зависимостями

```
$ terraform destroy

google_compute_instance.app: Destroying... (ID: instance-1)
google_compute_instance.app: Still destroying... (ID: instance-1, 10s elapsed)
google_compute_instance.app: Still destroying... (ID: instance-1, 20s elapsed)
google_compute_instance.app: Still destroying... (ID: instance-1, 30s elapsed)
google_compute_instance.app: Destruction complete after 36s
google_compute_address.app_ip: Destroying... (ID: example-ip)
```

Явные зависимости

```
resource "google_storage_bucket" "app_data" {
  name = "app-data-store-bucket"
}

resource "google_compute_instance" "app" {
  name         = "example-instance"
  machine_type = "f1-micro"

  depends_on = ["google_storage_bucket.app_data"]

  boot_disk {
    initialize_params {
      image = "ubuntu-1804-lts"
    }
  }

  network_interface {
    network       = "default"
    access_config = {}
  }
}
```

Data source

```
data "google_compute_image" "base_image" {
  family  = "ubuntu-1804-lts"
  project = "ubuntu-os-cloud"
}

resource "google_compute_instance" "app" {
  name         = "example-instance"
  machine_type = "f1-micro"

  boot_disk {
    initialize_params {
      image = "${data.google_compute_image.base_image.self_link}"
    }
  }

  network_interface {
    network       = "default"
    access_config = {}
  }
}
```

Управление стейтом и работа в команде

State

State - хранит информацию о соответсвии конфигурации Terraform реальным инфраструктурным единицам

```
...
"resources": {
  "google_compute_firewall.firewall_puma": {
    "type": "google_compute_firewall",
    "depends_on": [],
    "primary": {
      "id": "allow-puma-default",
      "attributes": {
        "allow.#": "1",
        "allow.931060522.ports.#": "1",
        "allow.931060522.ports.0": "9292",
        "allow.931060522.protocol": "tcp",
        "deny.#": "0",
        "description": "",
        "destination_ranges.#": "0",
...
```

Проблемы управления стейтом

- Доступность всем членам команды
- Блокировка стейт файла
- Разделение по окружениям

Проблемы хранения в гите

- Небезопасно

```
resource "google_service_account" "myaccount" {
  account_id = "myaccount"
  display_name = "My Service Account"
}

resource "google_service_account_key" "mykey" {
  service_account_id = "${google_service_account.myaccount.name}"
  public_key_type = "TYPE_X509_PEM_FILE"
}
```

Проблемы хранения в гите

- Небезопасно

```
"google_service_account_key.mykey": {
"type": "google_service_account_key",
...
"attributes": {
    "id": "projects/infra-234612/serviceAccounts/myaccount@infra-
234612.iam.gserviceaccount.com/keys/8cc72b47202cde9cb902362a4750fcf0190a8866",
    "key_algorithm": "KEY_ALG_RSA_2048",
    "name": "projects/infra-234612/serviceAccounts/myaccount@infra-
234612.iam.gserviceaccount.com/keys/8cc72b47202cde9cb902362a4750fcf0190a8866",
    "private_key": "ewogICJ0eXBlIjo....IzNDYxMi5pYW0uZ3NlcnZpY2VhY2NvdW50LmNvbSIKfQo=",
    "private_key_type": "TYPE_GOOGLE_CREDENTIALS_FILE",
    "public_key": "LS0tL.....tLQo=",
    "public_key_type": "TYPE_X509_PEM_FILE",
    "service_account_id": "projects/infra-234612/serviceAccounts/myaccount@infra-
234612.iam.gserviceaccount.com",
    "valid_after": "2019-04-01T12:48:25Z",
    "valid_before": "2029-03-29T12:48:25Z"
},
```

Проблемы хранения в гите

- Забываем обновлять

Плюсы remote backends

- Централизованное хранение стейта
- Автоматическое обновление при каждом изменении
- Поддержка блокирования стейта

Remote backends

Храним state в удалённом хранилище

- gcs (+lock)
- s3 (+lock, но через DynamoDB)
- consul (+lock)
- etcd (+lock)
- artifactory (-lock)
- pg (+lock)
- http (+lock, если реализован)

Пример

```
terraform {
  backend "gcs" {
    bucket = "terraform-2-otus-demo-storage-bucket"
    prefix = "stage"
  }
}
```

Обычно выносится в отдельный файл `backend.tf`. Инициализация нового бекенда прозводится во время вызова `terrafrom init`. При этом если state уже существует, то он копируется в backend.

Настройка remote backend

Так как backend инициализируется в самом начале, то нам недоступны переменные! Можно опустить часть параметров backend и задать их:

- Интерактивно
- Через файл
- Через командную строку

Пример

```
terraform {
  backend "pg" {}
}
```

```
terraform init \
  -backend-config="conn_str=postgres://user:password@db.example.com/terraform_backend"
```

Remote state как data source

```
data "terraform_remote_state" "vpc" {
  backend = "gcs"
  config {
    bucket = "my-state-bucket"
    prefix = "global/vpc"
  }
}

resource "google_compute_instance" "app" {
  # ...
  network_interface {
    network = "${data.terraform_remote_state.vpc.app_network}"
  }
}
```

Небольшая хитрость

Можно сохранить информацию о bucket, в котором мы храним state в том же bucket.

Демо.

Модули

Модули

- Набор конфигурационных файлов для управления частью инфраструктуры
- Позволяют переиспользовать уже написанный код
- Способствуют однородности окружений
- Важна параметризация
- Поддержка версионирования функционала
- Важна документация по использованию

Пример

modules/app/main.tf

```
resource "google_compute_instance" "app" {
  name         = "example-instance-${count.index + 1}"
  machine_type = "f1-micro"
  count        = "${var.instances_count}"

  boot_disk {
    initialize_params {
      image = "${var.app_image}"
    }
  }

  network_interface {
    network       = "default"
    access_config = {}
  }
}
```

Пример

prod/main.tf

```
provider "google" {
  version = "1.19.1"
  project = "${var.project_id}"
  region  = "${var.region}"
  zone    = "${var.zone}"
}

module "app" {
  source          = "../modules/app"
  app_image       = "ubuntu-1604-lts"
  instances_count = 6
}
```

Пример

stage/main.tf

```
provider "google" {
  version = "1.19.1"
  project = "${var.project_id}"
  region  = "${var.region}"
  zone    = "${var.zone}"
}

module "app" {
  source          = "../modules/app"
  app_image       = "ubuntu-1804-lts"
  instances_count = 3
}
```

Этот пример - плохой

Хороший модуль должен описывать полноценную часть приложения (frontend, DB, etc)

Почему:

Если модуль сложно назвать иначе чем `instance`, то это плохой модуль.

Исключение:

Модуль прячет ресурсы, которые трудно конфигурировать.

Где искать хороший пример

Terraform registry

- Marketplace для модулей - registry
- Поддержка Private registry
- Единое место для обмена модулями в Terraform
- Взаимодействие сообщества + развитие внутри компании

Пример использования модуля из Registry

Этот модуль определяет 19 ресурсов.

```
module "cloudsql" {
  source  = "google-terraform-modules/cloudsql/google"
  version = "1.13.0"

  general = {
    name       = "mydatabase"
    env        = "dev"
    region     = "${var.region}"
    db_version = "POSTGRES_9_6"
  }

  instance = {
    zone              = "b"
    availability_type = "REGIONAL"
  }
}
```

Продвинутое управление state

```
terraform state list
terraform state show

terraform state rm
terraform state mv

terraform state pull > path_to_state.tfstate
terraform state push path_to_state.tfstate
```

Принимают опцию -state=some_state.tfstate, по умолчанию terraform.tfstate. Если remote backend включён, то -state игнорируется!

Перенос существующего окружения в terraform

Добавляем описание ресурса в код

```
resource "google_compute_instance" "f2_a_clicks_counter" {
  name = "f2-a-clicks-counter"

  machine_type = "g1-small"

  boot_disk = {}

  network_interface = {
    network       = "default"

    access_config = {}
  }
}
```

Перенос существующего окружения в terraform

Добавляем его в state

```
$ terraform import \
  google_compute_instance.f2_a_clicks_counter \
  infra-123123/europe-west1-b/f2-a-clicks-counter

google_compute_instance.f2_a_clicks_counter: Importing from ID "infra-123123/europe-
west1-b/f2-a-clicks-counter"...
google_compute_instance.f2_a_clicks_counter: Import complete!
  Imported google_compute_instance (ID: f2-a-clicks-counter)
google_compute_instance.f2_a_clicks_counter: Refreshing state... (ID: f2-a-clicks-
counter)

Import successful!

The resources that were imported are shown above. These resources are now in
your Terraform state and will henceforth be managed by Terraform.
```

Перенос существующего окружения в terraform

Проверяем, что ресурс описан правильно

```
$ terraform plan

...
~ google_compute_instance.f2_a_clicks_counter
      machine_type:                         "n1-standard-1" => "g1-small"
....
```

Ссылки

- Terraform blog
- Доклады HashiConf 2017
- Рекомендуемые практики использования Terraform
- Reusable modules