

Практика IaC с использованием Terraform

Развитие проекта `infra`

В прошлых ДЗ вы создали инфраструктурный репозиторий `infra` на GitHub. Убедитесь что данный проект находится у вас на локальной машине.

Если у вас нет репозитория `infra` на GitHub, выполните сначала предыдущие ДЗ.



Проект *infra* и проверка ДЗ

Создайте новую ветку в вашем локальном репозитории для выполнения данного ДЗ. Т.к. это второе задание, посвященное работе с Terraform, то ветку можно назвать **terraform-2**.

Проверка данного ДЗ, как и многих последующих, будет производиться через Pull Request ветки с ДЗ к ветке мастер и добавлению в Reviewers пользователей **Artemmkin** и **serjs**.

После того, как **один** из преподавателей сделает approve пул реквеста, ветку с ДЗ можно смерджить.



Поднимем тестовый стенд

Используя наработки из предыдущего дз, создайте инфраструктуру при помощи тераформа:

```
$ terraform apply
```

Т.к. наша инфраструктура полностью описана в конфигурационных файлах terraform-а и мы ее можем создать в любой момент при помощи одной команды, то после выполнения каждого задания мы также рекомендуем вам выполнять `terraform destroy`, чтобы не тратить кредит на ресурсы GCP.

Правила файервола

Помните, как в предыдущем задании мы настраивали SSH доступ к создаваемой VM, создавая пользовательские ключи? Мы также говорили, что по умолчанию в новом проекте создается правило файервола, открывающее SSH доступ ко всем инстансам, запущенным в сети default (которая тоже создается по умолчанию в новом проекте). Давайте найдем это правило в консоли...



Google Cloud Platform

infra

VPC network

VPC networks

External IP addresses

Firewall rules

Routes

VPC network peering

Shared VPC

Firewall rules

+ CREATE FIREWALL RULE

REFRESH

Firewall rules control incoming or outgoing traffic to an instance. By default, incoming traffic from outside your network is blocked. [Learn more](#)

IngressEgress

| <input type="checkbox"/> Name | Targets | Source filters | Protocols / p |
|---|--------------|-------------------------|---------------|
| <input type="checkbox"/> allow-puma-default | Apply to all | IP ranges: 0.0.0.0/0 | tcp:9292 |
| <input type="checkbox"/> default-allow-icmp | Apply to all | IP ranges: 0.0.0.0/0 | icmp |
| <input type="checkbox"/> default-allow-internal | Apply to all | IP ranges: 10.128.0.0/9 | tcp:0-65535 |
| <input type="checkbox"/> default-allow-rdp | Apply to all | IP ranges: 0.0.0.0/0 | tcp:3389 |
| <input type="checkbox"/> default-allow-ssh | Apply to all | IP ranges: 0.0.0.0/0 | tcp:22 |

Проблема в том, что данное правило файервола не содержится в наших конфигурационных файлах terraform-а. Поэтому у нас все еще нет контроля над управлением нужными нам правилами файервола.

Определим ресурс файервола

Посмотрев информацию об интересующем нас правиле файервола, создайте ресурс в вашем конфиг файле main.tf (после правила файервола для пумы) с такой же конфигурацией, что у уже имеющегося правила:

```
resource "google_compute_firewall" "firewall_ssh" {  
  name      = "default-allow-ssh"  
  network   = "default"  
  
  allow {  
    protocol = "tcp"  
    ports    = ["22"]  
  }  
  
  source_ranges = ["0.0.0.0/0"]  
}
```

Выполним команду применения изменений:

```
$ terraform apply
google_compute_instance.app: Refreshing state... (ID: reddit-app)

...

* google_compute_firewall.firewall_ssh: 1 error(s) occurred:

* google_compute_firewall.firewall_ssh: Error creating firewall:
googleapi: Error 409: The resource 'projects/infra-179014/global/
firewalls/default-allow-ssh' already exists, alreadyExists
```

Возникла ошибка. Как вы думаете почему?

Импортируем существующую инфраструктуру в Terraform

Т.к terraform ничего не знает о существующем правиле файервола (а всю информацию, об известных ему ресурсах, он хранит в state файле), то при выполнении команды apply terraform пытается создать новое правило файервола. Для того чтобы сказать terraform-у не создавать новое правило, а управлять уже имеющимся, в его "записную книжку" (state файл) о всех ресурсах, которыми он управляет, нужно занести информацию о существующем правиле.

Импортируем существующую инфраструктуру в Terraform

Команда `import` позволяет добавить информацию о созданном без помощи Terraform ресурсе в state файл. В директории terraform выполните команду:

```
$ terraform import google_compute_firewall.firewall_ssh  
default-allow-ssh
```

```
google_compute_firewall.firewall_ssh: Importing from ID "default-allow-ssh"...
```

```
google_compute_firewall.firewall_ssh: Import complete!
```

```
  Imported google_compute_firewall (ID: default-allow-ssh)
```

```
google_compute_firewall.firewall_ssh: Refreshing state... (ID: default-allow-ssh)
```

Import successful!

Выполним планирование изменений:

```
$ terraform plan  
Refreshing Terraform state in-memory prior to plan...
```

```
~ google_compute_firewall.firewall_ssh  
  description: "Allow SSH from anywhere" => ""
```

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

Из планируемых изменений видим, что description(описание) существующего правила будет удалено. Можете добавить свое описание правила файервола в конфигурацию ресурса firewall_ssh. После чего выполните terraform apply.

Взаимосвязи ресурсов

Ресурс IP адреса

Зададим IP для инстанса с приложением в виде внешнего ресурса. Для этого определим ресурс `google_compute_address` в конфигурационном файле `main.tf`

```
resource "google_compute_address" "app_ip" {  
    name = "reddit-app-ip"  
}
```

Применим изменения

Удалим созданные до этого ресурсы:

```
$ terraform destroy
```

Создадим их вновь:

```
$ terraform apply
google_compute_firewall.firewall_ssh: Creating...
google_compute_firewall.firewall_puma: Creating...
google_compute_address.app_ip: Creating...
google_compute_instance.app: Creating...
```

Обратите внимание как тераформ параллельно начал создавать определенные нами ресурсы.

Ссылаемся на атрибуты другого ресурса

Для того чтобы использовать созданный IP адрес в нашем ресурсе VM нам необходимо сослаться на атрибуты ресурса, который этот IP создает, внутри конфигурации ресурса VM. В конфигурации ресурса VM определите, IP адрес для создаваемого инстанса.

```
network_interface {  
  network      = "default"  
  access_config = {  
    nat_ip = "${google_compute_address.app_ip.address}"  
  }  
}
```

указываем ресурс, на
который ссылаемся
type.name

указываем атрибут
ресурса

Неявная зависимость

Ссылку в одном ресурсе на атрибуты другого тераформ понимает как зависимость одного ресурса от другого. Это влияет на очередность создания и удаления ресурсов при применении изменений.

Вновь пересоздадим все ресурсы и посмотрим на очередность создания ресурсов сейчас (см. след. слайд)


```
$ terraform destroy
$ terraform plan
$ terraform apply
```

```
google_compute_address.app_ip: Creating...
google_compute_firewall.firewall_puma: Creating...
google_compute_firewall.firewall_ssh: Creating...
...
google_compute_address.app_ip: Creation complete after 12s (ID: reddit-app-ip)
google_compute_instance.app: Creating...
```

Видим, что ресурс VM начал создаваться только после завершения создания IP адреса в результате неявной зависимости этих ресурсов.

Terraform поддерживает также явную зависимость используя параметр depends_on.

Структуризация ресурсов

Несколько VM

Вынесем БД на отдельный инстанс VM. Для этого необходимо в директории `packer`, где содержатся ваши шаблоны для билда VM, создать два новых шаблона `db.json` и `app.json`. При помощи шаблона `db.json` должен собираться образ VM, содержащий установленную MongoDB. Шаблон `app.json` должен использоваться для сборки образа VM, с установленными Ruby. В качестве базового образа для создания образа возьмите `ubuntu16.04`.

Для выполнения задания, нужно лишь скопировать и слегка подкорректировать уже имеющийся шаблон `ubuntu16.json`. Постарайтесь выполнить самостоятельно, но на всякий случай, здесь находятся готовые файлы.



Создадим две VM

Разобьем конфиг main.tf на несколько конфигов.

Создадим файл app.tf, куда вынесем конфигурацию для VM с приложением. Пока пренебрежем провижинерами.

Обратите внимание, что мы вводим новую переменную для образа приложения. Не забудьте объявить ее в variables.tf:

```
variable app_disk_image {  
    description = "Disk image for reddit app"  
    default     = "reddit-app-base"  
}
```

app.tf

```
resource "google_compute_instance" "app" {
  name          = "reddit-app"
  machine_type  = "g1-small"
  zone          = "europe-west1-b"
  tags          = ["reddit-app"]
  boot_disk {
    initialize_params {
      image = "${var.app_disk_image}"
    }
  }
  network_interface {
    network      = "default"
    access_config = {
      nat_ip = "${google_compute_address.app_ip.address}"
    }
  }
  metadata {
    sshKeys = "appuser:${file(var.public_key_path)}"
  }
}
```

Также добавим в app.tf определение правила фаервола для сервера приложения и создание IP адреса.

```
resource "google_compute_address" "app_ip" {  
    name = "reddit-app-ip"  
}  
resource "google_compute_firewall" "firewall_puma" {  
    name      = "allow-puma-default"  
    network   = "default"  
  
    allow {  
        protocol = "tcp"  
        ports     = ["9292"]  
    }  
    source_ranges = ["0.0.0.0/0"]  
    target_tags   = ["reddit-app"]  
}
```

Создадим файл db.tf, в котором определим ресурсы для запуска VM с БД.

```
resource "google_compute_instance" "db" {
  name          = "reddit-db"
  machine_type  = "g1-small"
  zone          = "europe-west1-b"
  tags          = ["reddit-db"]
  boot_disk {
    initialize_params {
      image = "${var.db_disk_image}"
    }
  }
  network_interface {
    network      = "default"
    access_config = {}
  }
  metadata {
    sshKeys = "appuser:${file(var.public_key_path)}"
  }
}
```

Не забудьте объявить переменную в variables.tf

```
variable db_disk_image {  
    description = "Disk image for reddit db"  
    default     = "reddit-db-base"  
}
```


Также добавим в db.tf правило файервола, которое даст доступ приложению к БД

```
resource "google_compute_firewall" "firewall_mongo" {  
  name      = "allow-mongo-default"  
  network   = "default"  
  
  allow {  
    protocol = "tcp"  
    ports    = ["27017"]  
  }  
  # правило применимо к инстансам с тегом ...  
  target_tags = ["reddit-db"]  
  # порт будет доступен только для инстансов с тегом ...  
  source_tags = ["reddit-app"]  
}
```

Разбиваем остальную конфигурацию по файлам

Создадим файл `vpc.tf`, в который вынесем правило фаервола для `ssh` доступа, которое применимо для всех инстансов нашей сети.

```
resource "google_compute_firewall" "firewall_ssh" {  
    name      = "default-allow-ssh"  
    network   = "default"  
  
    allow {  
        protocol = "tcp"  
        ports    = ["22"]  
    }  
    source_ranges = ["0.0.0.0/0"]  
}
```

В итоге, в файле main.tf должно остаться только определение провайдера:

```
provider "google" {  
    project = "${var.project}"  
    region  = "${var.region}"  
}
```

Применяем изменения

Применим новую конфигурацию. Планируем и применяем изменения одной командой:

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

Если у вас все прошло успешно, можете проверить, что хосты доступны, и на них установлено необходимое ПО. Затем удалите созданные ресурсы, используя `terraform destroy`.

Если у вас возникли трудности с разбиением на отдельные конфигурационные файлы, то вы можете посмотреть, как оно должно выглядеть в данном репозитории.

Модули

Разбивая нашу конфигурацию нашей инфраструктуры на отдельные конфиг файлы, мы готовили для себя почву для работы с модулями.

Внутри директории terraform создайте директорию modules, в которой мы будем определять модули.

DB module

Внутри директории `modules` создайте директорию `db`, в которой создайте три привычных нам файла `main.tf`, `variables.tf`, `outputs.tf`.

Скопируем содержимое `db.tf`, который мы создали ранее, в `modules/db/main.tf`.

Затем определим переменные, которые у нас используются в `db.tf` и объявляются в `variables.tf` в файл переменных модуля `modules/db/variables.tf`

modules/db/variables.tf

```
variable public_key_path {  
    description = "Path to the public key used to connect to instance"  
}  
  
variable db_disk_image {  
    description = "Disk image for reddit db"  
    default     = "reddit-db-base"  
}
```



App module

Создадим по аналогии модуль приложения: в директории `modules` создадим директорию `app`, в которой создайте три привычных нам файла `main.tf`, `variables.tf`, `outputs.tf`.

Скопируем содержимое `app.tf`, который мы создали ранее, в `modules/app/main.tf`.

Затем определим переменные, которые у нас используются в `app.tf` и объявляются в `variables.tf` в файл переменных модуля `modules/app/variables.tf`.

modules/app/variables.tf

```
variable public_key_path {  
    description = "Path to the public key used to connect to instance"  
}
```

```
variable app_disk_image {  
    description = "Disk image for reddit app"  
    default     = "reddit-app-base"  
}
```

Не забудем про выходные переменные

modules/app/outputs.tf

```
output "app_external_ip" {  
  value = "${google_compute_instance.app.network_interface.0.access_config.0.assigned_nat_ip}"  
}
```



Проверим работу модулей

Прежде чем вызывать и проверять модули, для начала удалим `db.tf` и `app.tf` в нашей директории (или поменяем расширение файлов), чтобы тераформ перестал их использовать.

В файл `main.tf`, где у нас определен провайдер вставим секции вызова созданных нами модулей.

terraform/main.tf

Источник, откуда
копировать модуль

```
...  
module "app" {  
    source          = "modules/app"  
    public_key_path = "${var.public_key_path}"  
    app_disk_image  = "${var.app_disk_image}"  
}
```

```
module "db" {  
    source          = "modules/db"  
    public_key_path = "${var.public_key_path}"  
    db_disk_image  = "${var.db_disk_image}"  
}
```

Входные переменные
модуля

Чтобы начать использовать модули, нам нужно сначала их загрузить из указанного источника (source). В нашем случае источником модулей будет просто локальная папка на диске. Используем команду для загрузки модулей. В директории terraform:

```
$ terraform get
```

Модули будут загружены в директорию .terraform, в которой уже содержится провайдер (см. след. слайд)

```
$ terraform get
```

```
Get: file:///Users/artemkin/hw09/modules/app
```

```
Get: file:///Users/artemkin/hw09/modules/db
```

```
$ tree .terraform
```

```
.terraform
```

```
├── modules
```

```
|   ├── 9926d1ca5a4ce00042725999e3b3a90f -> /Users/artemkin/hw09/modules/db
```

```
|   └── dea8bdea57c956cc3317d254e5822e13 -> /Users/artemkin/hw09/modules/app
```

```
└── plugins
```

```
    ├── darwin_amd64
```

```
        ├── lock.json
```

```
        └── terraform-provider-google_v0.1.3_x4
```

Используем ресурсы модулей, которые мы загрузили, для настройки инфраструктуры:

```
$ terraform plan  
1 error(s) occurred:
```

```
* module root: 1 error(s) occurred:
```

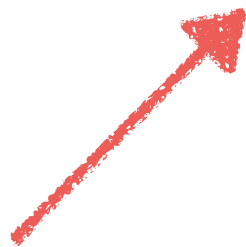
```
* output 'app_external_ip': unknown resource  
'google_compute_instance.app' referenced in variable  
google_compute_instance.app.network_interface.0.access_config.  
0.assigned_nat_ip
```

Возникает ошибка :(В файле outputs.tf осталась выходная переменная, которую мы определяли получения внешнего IP инстанса приложения. Сейчас она ссылается на несуществующий ресурс. Давайте поправим это.

Получаем output переменные из модуля

В созданном нами модуле app мы определили выходную переменную для внешнего IP инстанса. Чтобы получить значение этой переменной, переопределим ее:

```
output "app_external_ip" {  
    value = "${module.app.app_external_ip}"  
}
```



Говорим ссылаться на
атрибут модуля app

Повторяем попытку:

```
$ terraform plan
```

```
Refreshing Terraform state in-memory prior to plan...
```

```
+google_compute_firewall.firewall_ssh  
+module.app.google_compute_address.app_ip  
+module.app.google_compute_instance.app  
+module.app.google_compute_firewall.firewall_puma  
+module.db.google_compute_instance.db  
+module.db.google_compute_firewall.firewall_mongo
```

```
Plan: 6 to add, 0 to change, 0 to destroy.
```

Самостоятельное задание

Аналогично предыдущим модулям создайте модуль `vpc`, в котором определите настройки файервола в рамках сети.

Используйте созданный модуль в основной конфигурации `terraform/main.tf`.

Самопроверка

После применения конфигурации с помощью **terraform apply** в соответствии с нашей конфигурацией у нас должен быть SSH доступ ко обоим инстансам

```
$ ssh appuser@<instance_external_ip>
```

В браузерной консоли можете посмотреть созданные правила файрвола.

Параметризация модулей

Приведем пример параметризации модулей за счет использования input переменных.

В созданном вами модуле vrs используем переменную для конфигурации допустимых адресов.

terraform/vpc/main.tf

```
resource "google_compute_firewall" "firewall_ssh" {  
  name      = "default-allow-ssh"  
  network   = "default"  
  allow {  
    protocol = "tcp"  
    ports     = ["22"]  
  }  
  source_ranges = "${var.source_ranges}"  
}
```

terraform/vpc/variables.tf

```
variable source_ranges {  
  description = "Allowed IP addresses"  
  default      = ["0.0.0.0/0"]  
}
```

Теперь мы можем задавать диапазоны IP адресов для правила файервола при вызове модуля.

terraform/main.tf

```
...  
module "vpc" {  
    source           = "modules/vpc"  
    source_ranges = ["80.250.215.124/32"]  
}
```

Правило выше допускает SSH доступ только одному внешнему IP адресу. Здесь я использовал мой внешний IP, который можно посмотреть на myip.ru Таким образом инстансы будут доступны по SSH только мне. Не нужно выполнять terraform apply, конфиг выше дан для примера и потребуется в дальнейшем.

Переиспользование модулей

Основную задачу, которую решают модули - это увеличивают переиспользуемость кода и помогают нам следовать принципу DRY. Инфраструктуру, которую мы описали в модулях, теперь можно использовать на разных стадиях нашего конвейера непрерывной поставки с необходимыми нам изменениями.

Создадим инфраструктуру для двух окружений (stage и prod), используя созданные модули.

Stage & Prod

В директории terraform создайте две директории: stage и prod. Скопируйте файлы main.tf, variables.tf, outputs.tf, terraform.tfvars из директории terraform в каждую из созданных директорий. Поменяйте пути к модулям в main.tf на `../modules/xxx` вместо `modules/xxx`.

Инфраструктура в обоих окружениях будет идентична, однако будет иметь небольшие различия: мы откроем SSH доступ для всех IP адресов в окружении Stage, а в окружении Prod откроем доступ только для своего IP.

terraform/stage/main.tf

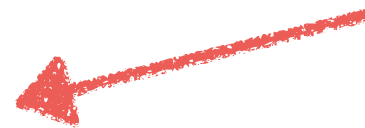
```
provider "google" {  
  project = "${var.project}"  
  region  = "${var.region}"  
}  
module "app" {  
  source          = "../modules/app"  
  public_key_path = "${var.public_key_path}"  
  app_disk_image  = "${var.app_disk_image}"  
}  
module "db" {  
  source          = "../modules/db"  
  public_key_path = "${var.public_key_path}"  
  db_disk_image   = "${var.db_disk_image}"  
}  
module "vpc" {  
  source          = "../modules/vpc"  
  source_ranges = ["0.0.0.0/0"]  
}
```

Открываем
доступ для всех
адресов

terraform/prod/main.tf

```
provider "google" {  
  project = "${var.project}"  
  region  = "${var.region}"  
}  
module "app" {  
  source          = "../modules/app"  
  public_key_path = "${var.public_key_path}"  
  app_disk_image  = "${var.app_disk_image}"  
}  
module "db" {  
  source          = "../modules/db"  
  public_key_path = "${var.public_key_path}"  
  db_disk_image  = "${var.db_disk_image}"  
}  
module "vpc" {  
  source          = "../modules/vpc"  
  source_ranges  = ["82.155.222.156/32"]  
}
```

Открываем доступ
для своего
внешнего IP
(поменяйте на свой)



Проверьте конфигурацию окружений

Проверьте правильность настроек инфраструктуры каждого окружения. Для этого нужно запустить `terraform apply` в каждом из них.

Не забывайте удалять ресурсы после проверок.

Самостоятельные задания

1. Отформатируйте конфигурационные файлы, используя команду `terraform fmt`.
2. Параметризируйте конфигурацию модулей насколько считаете нужным.

Не забывайте удалять ресурсы в конце

После выполнения ДЗ не забывайте удалить созданные terraform-ом ресурсы:

```
$ terraform destroy
```

Задание со звездочкой

Настроить хранение стейта файла в удаленном бекенде (remote backends), используя Google Cloud Storage в качестве бекенда.