

Практика IaC с использованием Terraform

Проект *infra* и проверка ДЗ

Создайте новую ветку в вашем инфраструктурном репозитории для выполнения данного ДЗ. Т.к. это первое задание, посвященное работе с Terraform, то ветку назовем **terraform-1**.

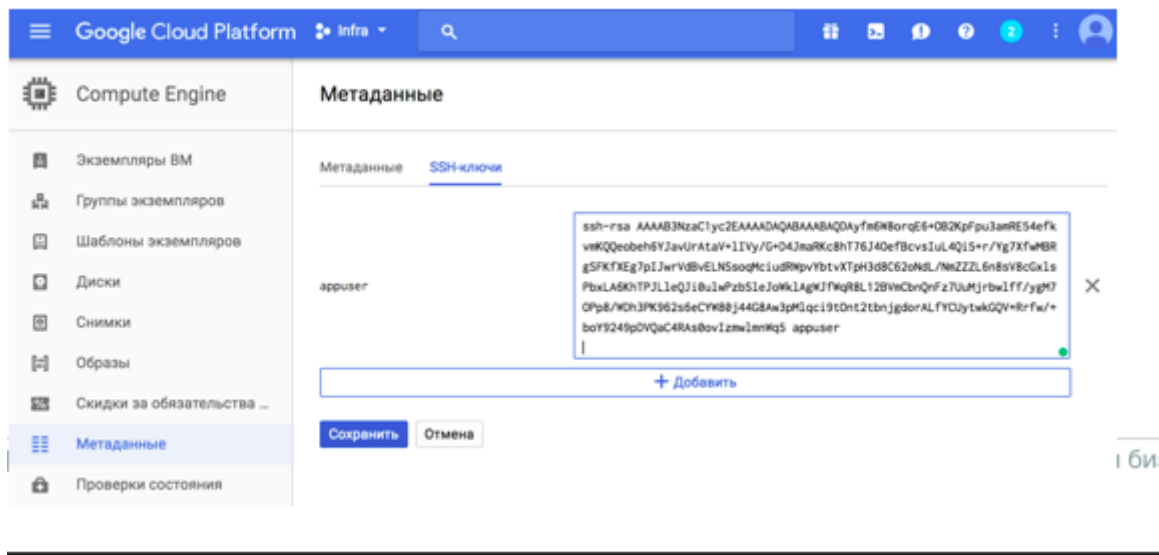
Проверка данного ДЗ, как и последующих, производится через Pull Request ветки с ДЗ к ветке мастер.

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

Предварительные требования

Перед тем как приступить к выполнению данного ДЗ, убедитесь, что у вас выполнены предыдущие ДЗ и в вашем проекте infra есть Packer шаблон для создания базового образа нашего приложения.

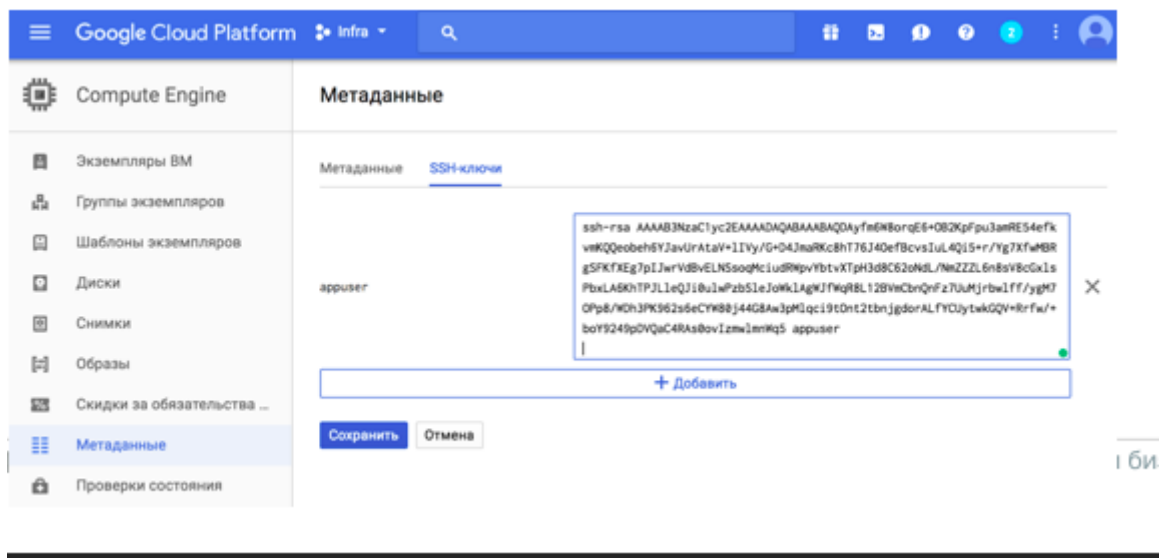
Проверьте, что у вас собран образ в предыдущем ДЗ. При необходимости, можете пересобрать образ.



Предварительные требования

Удалите ключи пользователя **appuser** из метаданных проекта.

Зайдите в *Compute Engine* -> *Метаданные* -> *SSH-ключи*.
Нажмите "Изменить", потом крестик справа от ключа пользователя, и нажмите "Сохранить".



Установка Terraform

Скачайте версию Terraform для вашей ОС, перейдя по [данной](#) ссылке. Распакуйте скачанный zip архив и поместите бинарный файл в директорию, путь до которой содержится в переменной окружения PATH.

Проверить установку Terraform можно командой:

```
terraform -v
```

В данном курсе мы используем версию 0.11.11. Версии 0.12 в курсе не поддерживаются.

Развитие проекта Infra

Создайте директорию `terraform` внутри проекта `infra`. **Внутри директории `terraform`** создайте пустой файл: `main.tf`. Это будет главный конфигурационный файл в этом задании, который будет содержать декларативное описание нашей инфраструктуры.

В **корне репозитория** создайте файл `.gitignore` с содержимым указанным в данном [gist](#).

Это необходимо для того, чтобы не коммитить в репозиторий служебные файлы и директории.

Provider

Первый делом определим секцию Provider в файле main.tf, которая позволит Terraform управлять ресурсами GCP через API
ВЫЗОВЫ

```
terraform {  
  # Версия terraform  
  required_version = "0.11.11"  
}  
  
provider "google" {  
  # Версия провайдера  
  version = "2.0.0"  
  
  # ID проекта  
  project = "infra-908766"  
  
  region = "europe-west-1"  
}
```

Terraform init

Провайдеры Terraform являются загружаемыми модулями, начиная с версии 0.10. Для того чтобы загрузить провайдер и начать его использовать выполните следующую команду в директории terraform:

```
$ terraform init
```

```
Initializing provider plugins...
```

- Checking **for** available provider plugins on <https://releases.hashicorp.com>...
- Downloading plugin **for** provider "google" (2.0.0)...

```
Terraform has been successfully initialized!
```

```
You may now begin working with Terraform. Try running "terraform plan" to see any changes that are required for your infrastructure. All Terraform commands should now work.
```

```
If you ever set or change modules or backend configuration for Terraform, rerun this command to reinitialize your working directory. If you forget, other commands will detect it and remind you to do so if necessary
```

Ресурсная модель

Terraform предоставляет широкий набор примитивов (resources) для управления ресурсами различных сервисов GCP.

Полный список предоставляемых terraform'ом ресурсов для работы с GCP можно посмотреть слева на [данной странице](#).

Чтобы запустить VM при помощи terraform, нам нужно воспользоваться ресурсом [google compute instance](#), который позволяет управлять инстансами VM.

В файле `main.tf` после определения провайдера, добавьте ресурс для создания инстанса VM в GCP.

```
resource "google_compute_instance" "app" {
  name          = "reddit-app"
  machine_type  = "g1-small"
  zone          = "europe-west1-b"
  # определение загрузочного диска
  boot_disk {
    initialize_params {
      image = "reddit-base"
    }
  }
  # определение сетевого интерфейса
  network_interface {
    # сеть, к которой присоединить данный интерфейс
    network = "default"
    # использовать ephemeral IP для доступа из Интернет
    access_config {}
  }
}
```

Отметим, что в определении загрузочного диска для VM, мы передаем имя семейства образа, в данном случае "reddit-base". Также можно передать полное имя образа, например "reddit-base-1513757169".

```
# определение загрузочного диска
boot_disk {
  initialize_params {
    image = "reddit-base"
  }
}
```

Планируем изменения

Перед тем как дать команду terraform'у применить изменения, хорошей практикой является предварительно посмотреть, какие изменения terraform собирается произвести относительно состояния известных ему ресурсов (tfstate файл), и проверить, что мы действительно хотим сделать именно эти изменения. Выполните команду планирования изменений в директории terraform:

```
terraform plan
```

Планируем изменения

Знак "+" перед наименованием ресурса означает, что ресурс будет добавлен. Далее приведены атрибуты этого ресурса. "`<computed>`" означает, что данные атрибуты еще не известны terraform'у и их значения будут получены во время создания ресурса.

```
+ google_compute_instance.app
  boot_disk.#: "1"
  boot_disk.0.auto_delete: "true"
  boot_disk.0.device_name: "<computed>"
  boot_disk.0.disk_encryption_key_sha256: "<computed>"
  boot_disk.0.initialize_params.#: "1"
  boot_disk.0.initialize_params.0.image: "reddit-base"
```

Планируем изменения

Внизу приводятся итоги планируемых изменений: количество ресурсов, которые будут добавлены, изменены и удалены

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

Создаем VM согласно описанию

Для того чтобы запустить инстанс VM, описание характеристик которого мы описали в конфигурационном файле `main.tf`, используем команду:

```
$ terraform apply
```

- Начиная с версии 0.11 **terraform apply** запрашивает дополнительное подтверждение при выполнении. Необходимо добавить **-auto-approve=true** для отключения этого.

В результате применения команды увидим, какие изменения были произведены terraform'ом:

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

terraform.tfstate

Результатом выполнения команды также будет создание файла `terraform.tfstate` в директории `terraform`.

Terraform хранит в этом файле состояние управляемых им ресурсов. Загляните в этот файл и найдите внешний IP адрес созданного инстанса.

terraform show

Искать нужные атрибуты ресурсов по state файлу не очень удобно, поэтому terraform предоставляет команду show для чтения стейт файла.

Найдите внешний IP адрес созданного инстанса, но уже используя команду show:

```
$ terraform show | grep nat_ip  
  
network_interface.0.access_config.0.nat_ip = 35.205.174.46
```

Подключение к инстансу по SSH

Зная внешний IP адрес, попробуем подключиться к инстансу по SSH, как мы делали до этого в предыдущих ДЗ, используя следующую команду:

```
ssh appuser@<внешний_IP>
```

Получилось ли подключиться? Если нет, то есть ли догадки, почему не удастся подключиться?

Добавим SSH ключ для пользователя `appuser`

Порт 22 для подключения к инстансу по SSH открыт по умолчанию и проблема с подключением заключается в том, что мы удалили в начале этого ДЗ публичный SSH ключ пользователя `appuser` из метаданных проекта.

Убедитесь, что на вашей локальной машине есть пара ключей для пользователя `appuser` (см. ДЗ **cloud-bastion** "Знакомство с облачной инфраструктурой")

Определим SSH ключ пользователя `appuser` в метаданных нашего инстанса. Помним, что публичные ключи доступны инстансам VM именно через метаданные. Добавим в `main.tf` внутри описание инстанса:

```
resource "google_compute_instance" "app" {  
  ...  
  metadata {  
    # путь до публичного ключа  
    ssh-keys = "appuser:${file("~/ssh/appuser.pub")}"  
  }  
  ...  
}
```

Здесь мы используем встроенную функцию `file`, которая позволяет считывать содержимое файла и вставлять его в наш конфигурационный файл.

Планируем изменения и смотрим, что terraform собирается изменить относительно известного предыдущего состояния.

```
terraform plan
```

Понимаете ли вы какие изменения произойдут, если мы применим изменения? Если да, то примените изменения

```
terraform apply
```

Обратите внимание на то, что terraform не удалял запущенный инстанс VM, т.к. его состояние уже соответствует описанию в `main.tf`.

Проверка SSH подключения

Вновь найдите внешний адрес нашего инстанса через `terraform show` и используйте его для подключения по SSH:

```
ssh appuser@<внешний_IP>
```

На этот раз подключение должно быть успешным.

Output vars

Согласитесь, что каждый раз использовать `grep` для поиска внешнего IP адреса по результатам команды `terraform show` довольно неудобно. К тому же, если у нас будет запущено несколько VM, то понять к какой машине относится какой адрес становится еще сложнее.

Поэтому вынесем интересующую нас информацию - внешний адрес VM - в выходную переменную (output variable), чтобы облегчить себе поиск.

outputs.tf

Чтобы не мешать выходные переменные с основной конфигурацией наших ресурсов, создадим их в отдельном файле, который назовем `outputs.tf`. Помним, что название файла может быть любым, т.к. terraform загружает все файлы в текущей директории, имеющие расширение `.tf`.

Создайте файл `outputs.tf` в директории `terraform` со следующим содержимым.

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



Идентифицируем
ресурс, указывая его
тип и имя



Указываем нужные
атрибуты ресурса

Используем команду `terraform refresh`, чтобы выходная переменная приняла значение.

```
Outputs:  
app_external_ip = 104.155.68.69
```

Значение выходных переменным можно посмотреть, используя команду `terraform output`:

```
$ terraform output  
app_external_ip = 104.155.68.69  
$ terraform output app_external_ip  
104.155.68.69
```

Откроем порт для приложения

Определим правило фаерволла для нашего приложения. Добавим в `main.tf` следующий ресурс:

```
resource "google_compute_firewall" "firewall_puma" {
  name = "allow-puma-default"
  # Название сети, в которой действует правило
  network = "default"
  # Какой доступ разрешить
  allow {
    protocol = "tcp"
    ports = ["9292"]
  }
  # Каким адресам разрешаем доступ
  source_ranges = ["0.0.0.0/0"]
  # Правило применимо для инстансов с перечисленными тэгами
  target_tags = ["reddit-app"]
}
```

Правило фаерволла для приложения

Определим правило фаерволла для нашего приложения. Добавим в `main.tf` следующий ресурс:

Планируем и применяем изменения:

```
terraform plan
```

```
terraform apply
```

Добавляем тег инстансу VM

В описании правила фаерволла мы указали, что оно применимо только для инстансов VM, имеющих тег **reddit-app**. Сейчас это правило не применимо для нашего инстанса, так как в его описании мы не указали тег.

Добавим тег в определении ресурса.

```
resource "google_compute_instance" "app" {  
  name = "reddit-app"  
  machine_type = "g1-small"  
  zone = "europe-west1-b"  
  tags = ["reddit-app"]  
  ...  
}
```

Добавляем тег инстансу VM

Планируем изменения

```
terraform plan
```

```
...  
~ google_compute_instance.app  
tags.#: "0" => "1"  
tags.1799682348: "" => "reddit-app"  
Plan: 0 to add, 1 to change, 0 to destroy.
```

Видим, что ресурс виртуальной машины будет изменен.

Добавляем тег экземпляру VM

Применяем изменения

```
terraform apply
```

Provisioners

Provisioners в terraform вызываются в момент создания/удаления ресурса и позволяют выполнять команды на удаленной или локальной машине. Их используют для запуска инструментов управления конфигурацией или начальной настройки системы.

Используем провижинеры для деплоя последней версии приложения на созданную VM.

Внутри ресурса, содержащего описание VM, вставьте секцию провижинера типа `file`, который позволяет копировать содержимое файла на удаленную машину.

```
provisioner "file" {  
  source = "files/puma.service"  
  destination = "/tmp/puma.service"  
}
```

В нашем случае мы говорим, провижинеру скопировать локальный файл, располагающийся по указанному относительному пути (**files/puma.service**), в указанное место на удаленном хосте.

Unit file для Puma

В определении провижинера мы указали путь до systemd unit файла для Puma. Systemd использует unit файлы для запуска, остановки сервиса или добавления его в автозапуск. С его помощью мы можем запускать сервер приложения, используя команду `systemctl start puma`.

Создадим директорию **files** внутри директории **terraform** и создадим внутри нее файл `puma.service` с содержимым, указанным по [данной ссылке](#).

Добавим еще один провиженер для запуска скрипта деплоя приложения на создаваемом инстансе. Сразу же после определения провижинера `file` (провижинеры выполняются по порядку их определения), вставьте секцию провижинера `remote-exec`:

```
provisioner "remote-exec" {  
  script = "files/deploy.sh"  
}
```

В определении данного провижинера мы указываем относительный путь до скрипта, который следует запустить на созданной VM.

Создайте файл `deploy.sh` в директории `terraform/files` с содержимым, указанным в [данном gist](#).

Параметры подключения provisioners

Определим параметры подключения провиженеров к VM. **Внутри ресурса VM**, перед определением провижинеров, добавьте следующую секцию:

```
connection {  
  type = "ssh"  
  user = "appuser"  
  agent = false  
  # путь до приватного ключа  
  private_key = "${file("~/ssh/appuser")}"  
}
```

В данном примере мы указываем, что провижинеры, определенные в ресурсе VM, должны подключаться к созданной VM по SSH, используя для подключения приватный ключ пользователя appuser.

Ресурс для VM сейчас должен выглядеть следующим образом ([gist](#))

```
resource "google_compute_instance" "app" {
  name = "reddit-app"
  machine_type = "g1-small"
  zone = "europe-west1-b"
  boot_disk {
    initialize_params {
      image = "reddit-base"
    }
  }
  metadata {
    ssh-keys = "appuser:${file("~/ssh/appuser.pub")}"
  }
  tags = ["reddit-app"]
  network_interface {
    network = "default"
    access_config {}
  }
  connection {
    type = "ssh"
    user = "appuser"
    agent = false
    private_key = "${file("~/ssh/appuser")}"
  }
  ...
}
```

Продолжение. Ресурс для VM сейчас должен выглядеть следующим образом ([gist](#))

```
...
provisioner "file" {
  source = "files/puma.service"
  destination = "/tmp/puma.service"
}
provisioner "remote-exec" {
  script = "files/deploy.sh"
}
```

Проверяем работу провижинеров

Так как провижинеры по умолчанию запускаются сразу после создания ресурса (могут еще запускаться после его удаления), чтобы проверить их работу нам нужно удалить ресурс VM и создать его снова.

Terraform предлагает команду **taint**, которая позволяет пометить ресурс, который terraform должен пересоздать, при следующем запуске **terraform apply**.

Говорим terraform'у пересоздать ресурс VM при следующем применении изменений:

```
$ terraform taint google_compute_instance.app
The resource google_compute_instance.app in the module root
has been marked as tainted!
```

Планируем изменения:

```
$ terraform plan
...
-/+ google_compute_instance.app (tainted) (new resource required)
boot_disk.#: "1" => "1"
boot_disk.0.auto_delete: "true" => "true"
```

-/+ означает, что ресурс будет удален и создан вновь.

Применяем изменения

```
terraform apply
```

```
google_compute_instance.app (remote-exec): Connecting to remote host via SSH...
google_compute_instance.app (remote-exec):   Host: 104.199.32.99
google_compute_instance.app (remote-exec):   User: appuser
google_compute_instance.app (remote-exec):   Password: false
google_compute_instance.app (remote-exec):   Private key: true
google_compute_instance.app (remote-exec):   SSH Agent: false
google_compute_instance.app (remote-exec): Connected!
google_compute_instance.app: Still creating... (50s elapsed)
google_compute_instance.app (remote-exec): Cloning into 'reddit'...
google_compute_instance.app (remote-exec): remote: Counting objects: 139, done.
```

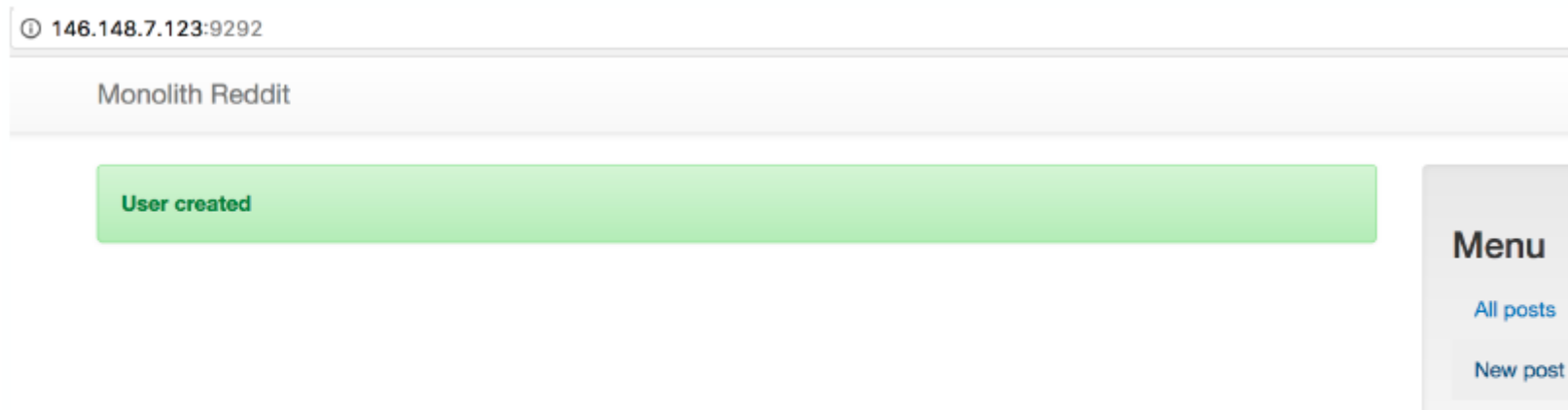
провиджеру удалось
подключиться к VM

Скрипт запустился и
началось
клонирование
репозитория

40

Проверяем работу приложения

Открываем в браузере страницу с указанием внешнего адреса инстанса (помните terraform output?) и номер порта сервера приложения.



41

Input vars

Входные переменные позволяют нам параметризовать конфигурационные файлы.

Для того чтобы использовать входную переменную ее нужно сначала определить в одном из конфигурационных файлов. Создадим для этих целей еще один конфигурационный файл `variables.tf` в директории `terraform`.

variables.tf

Определим переменные в файле variables.tf

```
variable project {
  description = "Project ID"
}
variable region {
  description = "Region"
  # Значение по умолчанию
  default = "europe-west1"
}
variable public_key_path {
  # Описание переменной
  description = "Path to the public key used for ssh access"
}
variable disk_image {
  description = "Disk image"
}
```

Используем input переменные

Теперь можем использовать input переменные в определении других ресурсов. Чтобы получить значение пользовательской переменной внутри ресурса используется синтаксис `"${var.var_name}"`.

Определим соответствующие параметры ресурсов `main.tf` через переменные:

```
provider "google" {  
  version = "2.0.0"  
  project = "${var.project}"  
  region = "${var.region}"  
}
```

Используем input переменные

```
...
boot_disk {
  initialize_params {
    image = "${var.disk_image}"
  }
}
...
metadata {
  ssh-keys = "appuser:${file(var.public_key_path)}"
}
...
```

Определим переменные

Определим переменные используя специальный файл `terraform.tfvars`, из которого terraform загружает значения автоматически при каждом запуске.

В директории **terraform** создайте файл `terraform.tfvars`, в котором определите ваши переменные. Это должно выглядеть примерно так:

```
project = "infra-179015"  
public_key_path = "~/.ssh/appuser.pub"  
disk_image = "reddit-base"
```

Финальная проверка

Пересоздадим все ресурсы созданные при помощи terraform.

Для начала удалим все созданные ресурсы:

```
$ terraform destroy
...
Do you really want to destroy?
Terraform will delete all your managed infrastructure, as shown above.
There is no undo. Only 'yes' will be accepted to confirm.
Enter a value: yes
```

Затем создадим ресурсы вновь. Terraform автоматически будет использовать переменные, определенные в `terraform.tfvars`.

```
terraform plan
terraform apply
```

Финальная проверка

146.148.7.123:9292

Monolith Reddit

User created

Menu

[All posts](#)

[New post](#)

41

Самостоятельные задания

1. Определите input переменную для приватного ключа, используемого в определении подключения для провижинеров (connection);
2. Определите input переменную для задания зоны в ресурсе "google_compute_instance" "app". У нее * **должно** быть значение по умолчанию*;
3. Отформатируйте **все** конфигурационные файлы используя команду **terraform fmt**;
4. Так как в репозиторий не попадет ваш **terraform.tfvars**, то нужно сделать рядом файл **terraform.tfvars.example**, в котором будут указаны переменные для образца. [Пример в gist](#).

Задание со *

В начале данного задания мы удалили ssh ключи пользователей из метаданных проекта и описали в коде их добавление в метаданные инстанса.

- Опишите в коде terraформа добавление ssh ключа пользователя `appuser1` в метаданные проекта. Выполните `terraform apply` и проверьте результат (публичный ключ можно брать пользователя `appuser`)
- Опишите в коде terraформа добавление ssh ключей нескольких пользователей в метаданные проекта (можно просто один и тот же публичный ключ, но с разными именами пользователей, например `appuser1`, `appuser2` и т.д.).
Выполните `terraform apply` и проверьте результат

Задание со *

- Добавьте в веб интерфейсе ssh ключ пользователю `appuser_web` в метаданные проекта. Выполните `terraform apply` и проверьте результат
- Какие проблемы вы обнаружили? Добавьте описание в `README.md`
- Не забудьте закоммитить добавленный код в репозиторий и добавить описание в `README.md`

Документация

P.S. Использование директивы `sshKeys` на уровне инстансов `deprecated`, т.е. их использовать нежелательно.

Задание с **

В данный момент у нас с помощью terraform создается один инстанс с запущенным приложением и правило для firewall.

Задания:

- Создайте файл `lb.tf` и опишите в нем в коде terraform создание HTTP балансировщика, направляющего трафик на наше развернутое приложение на инстансе **reddit-app**. Проверьте доступность приложения по адресу балансировщика. Добавьте в output переменные адрес балансировщика.

Задание с **

- Добавьте в код еще один terraform ресурс для нового инстанса приложения, например **reddit-app2**, добавьте его в балансировщик и проверьте, что при остановке на одном из инстансов приложения (например `systemctl stop puma`), приложение продолжает быть доступным по адресу балансировщика; Добавьте в output переменные адрес второго инстанса; Какие проблемы вы видите в такой конфигурации приложения? Добавьте описание в README.md.

Задание с **

- Как мы видим, подход с созданием доп. инстанса копированием кода выглядит нераационально, т.к. копируется много кода. Удалите описание **reddit-app2** и попробуйте подход с заданием количества инстансов через параметр ресурса **count**. Переменная **count** должна задаваться в параметрах и по умолчанию равна 1.

Задание с **

- Не забудьте закоммитить добавленный код в репозиторий и добавить описание в README.md;

[Документация](#)

Удалим созданные ресурсы

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

```
terraform destroy
```

Проверка ДЗ

- Результаты вашей работы находятся в ветке **terraform-1** вашего инфраструктурного репозитория.
- В README добавлено описание того, что сделано.
- Создайте Pull Request к ветке master (**описание PR нужно заполнять**);
- В ревьюеры можно никого не добавлять;
- Добавьте "Labels" **terraform** и **terraform-1** к вашему Pull Request;
- После того, как один из преподавателей сделает approve пул реквеста, ветку с ДЗ можно смерджить и закрыть PR.

Проверка ДЗ

- **TravisCI** делает валидацию кода тераформа и пакера, проверяет наличие файлов и наличие пустой строки в конце файла. В файле **terraform.tfvars.example** должны использоваться ключи пользователя **appuser**