

# Создаем собственные провайдеры



Сергей  
Андрюнин



# Сергей Андрюнин

DevOps-инженер

RT Labs



Сергей Андрюнин



# План занятия

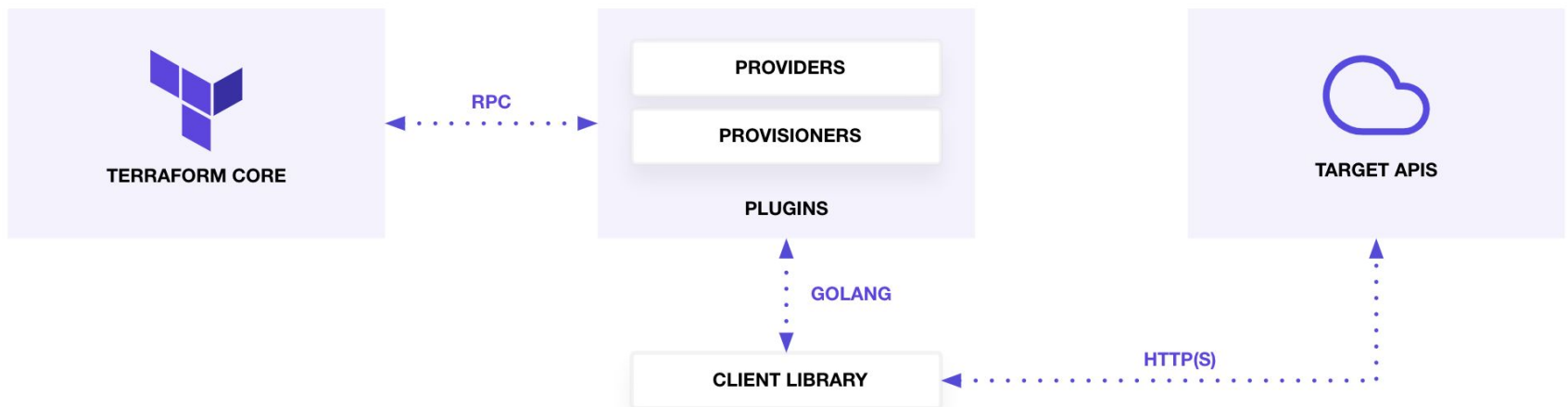
1. [Работа с собственным провайдером](#)
2. [Устройство провайдера](#)
3. [Чтение ресурсов](#)
4. [Авторизация](#)
5. [Создание ресурса](#)
6. [Обновление ресурса](#)
7. [Удаление ресурса](#)
8. [Итоги](#)
9. [Домашнее задание](#)



# Работа с собственным провайдером

# Схема http провайдера

Давайте научим терраформ варить кофе при помощи собственного провайдера:



# Подготовка среды

Установим кофемашину hashicups:

```
$ git clone https://github.com/hashicorp/learn-terraform-hashicups-provider  
$ cd learn-terraform-hashicups-provider
```

Запустим кофемашину:

```
$ cd docker_compose  
$ docker-compose up
```

Проверим ее:

```
$ curl localhost:19090/health  
ok
```

# Установка провайдера для Mac

Скачиваем провайдер:

```
$ curl -Lo terraform-provider-hashicups  
https://github.com/hashicorp/terraform-provider-hashicups/releases/download/v0.2/ter  
raform-provider-hashicups_0.2_darwin_amd64
```

Делаем его исполняемым:

```
$ chmod +x terraform-provider-hashicups
```

Создаем каталог и копируем провайдер:

```
$ mkdir -p ~/.terraform.d/plugins/hashicorp.com/edu/hashicups/0.2/darwin_amd64  
$ mv terraform-provider-hashicups  
~/.terraform.d/plugins/hashicorp.com/edu/hashicups/0.2/darwin_amd64
```

# Установка провайдера для Linux

Скачиваем провайдер:

```
$ curl -Lo terraform-provider-hashicups_0.3.1_darwin_amd64.zip  
https://github.com/hashicorp/terraform-provider-hashicups/releases/download/v0.3.1/terraform-provider-hashicups\_0.3.1\_darwin\_amd64.zip  
$ unzip terraform-provider-hashicups_0.3.1_darwin_amd64.zip  
$ mv terraform-provider-hashicups_v0.3.1 terraform-provider-hashicups  
$ chmod +x terraform-provider-hashicups
```

Создаем каталог и копируем провайдер:

```
$ mkdir -p ~/.terraform.d/plugins/hashicorp.com/edu/hashicups/0.2/linux_amd64  
$ mv terraform-provider-hashicups  
~/.terraform.d/plugins/hashicorp.com/edu/hashicups/0.2/linux_amd64
```



# Установка провайдера для Windows

Скачиваем провайдер:

```
$ curl -Lo terraform-provider-hashicups  
https://github.com/hashicorp/terraform-provider-hashicups/releases/download/v0.2/ter  
raform-provider-hashicups_0.2_windows_amd64
```

Делаем его исполняемым:

```
$ icacls ".\terraform-provider-hashicups" /grant USER:RX
```

Создаем каталог и копируем провайдер:

```
$ mkdir -p  
%APPDATA%\terraform.d\plugins\hashicorp.com\edu\hashicups\0.2\windows_amd64  
  
$ move terraform-provider-hashicups.exe  
%APPDATA%\terraform.d\plugins\hashicorp.com\edu\hashicups\0.2\windows_amd64
```

# Создаем нового HashiCups пользователя

Создаем пользователя **education** с паролем **test123**:

```
$ curl -X POST localhost:19090/signup -d '{"username":"education",  
"password":"test123"}'  
{"UserID":1,"Username":"education","token":"..."}
```

Проходим аутентификацию:

```
$ curl -X POST localhost:19090/signin -d '{"username":"education", "password":"test123"}'  
{"UserID":1,"Username":"education","token":"..."}
```

Сохраняем полученный токен:

```
$ export HASHICUPS_TOKEN=...
```

# Создаем нового HashiCups пользователя

Создаем пользователя **education** с паролем **test123**:

```
$ curl -X POST localhost:19090/signup -d '{"username":"education",  
"password":"test123"}'  
{"UserID":1,"Username":"education","token":"..."}
```

Проходим аутентификацию:

```
$ curl -X POST localhost:19090/signin -d '{"username":"education", "password":"test123"}'  
{"UserID":1,"Username":"education","token":"..."}
```

Сохраняем полученный токен:

```
$ export HASHICUPS_TOKEN=...
```

# Варим кофе

```
terraform {
  required_providers {
    hashicups = {
      versions = ["0.2"]
      source =
"hashicorp.com/edu/hashicups
"
    }
  }
}

provider "hashicups" {
  username = "education"
  password = "test123"
}
```

```
resource "hashicups_order"
"edu" {
  items {
    coffee {id = 3}
    quantity = 2
  }
  items {
    coffee {id = 2}
    quantity = 2
  }
}

output "edu_order" {
  value = hashicups_order.edu
}
```

# Процесс варки кофе

Логи докера:

```
api_1 | 2020-11-18T03:15:58.123Z [INFO] Handle User | signin  
api_1 | 2020-11-18T03:16:02.056Z [INFO] Handle User | signin  
api_1 | 2020-11-18T03:16:02.067Z [INFO] Handle Orders | CreateOrder  
api_1 | 2020-11-18T03:16:02.083Z [INFO] Handle Orders | GetUserOrder
```

Создаем, изменяем, проверяем стейты:

```
$ terraform plan  
$ terraform apply  
$ terraform state show hashicups_order.edu  
$ curl -X GET -H "Authorization: ${HASHICUPS_TOKEN}" localhost:19090/orders/1
```



# Устройство провайдера

# Подготовка среды

Клонируем исходники:

```
$ cd ~/go/src/github.com/hashicorp/  
$ git clone --branch boilerplate https://github.com/hashicorp/terraform-provider-hashicups
```

Запустим тестовую кофемашину:

```
$ cd docker_compose  
$ docker-compose up
```

Проверим ее:

```
$ curl localhost:19090/health  
ok
```

# Makefile

Makefile содержит вспомогательные функции, используемые для сборки, упаковки и установки провайдера.

Весь список **GOARCH** и **GOOS**: <https://golang.org/doc/install/source#environment>

Например для windows:

```
- BINARY=terraform-provider-${NAME}
+ BINARY=terraform-provider-${NAME}.exe
- OS_ARCH=darwin_amd64
+ OS_ARCH=windows/amd64
```





## hashicups/provider.go

Сейчас здесь определен пустой провайдер.

В самом простом случае здесь можно задать доступные:

- ресурсы (блок resources),
- источники данных (блок data).

# Билдим провайдер

Запустите команду **go mod init**, чтобы указать что этот каталог является корнем модуля:

```
$ go mod init terraform-provider-hashicups  
go: creating new go.mod: module terraform-provider-hashicups
```

Затем запустите **go mod vendor**, чтобы выкачать зависимости:

```
$ go mod vendor
```

Билдим провайдер:

```
$ make build  
go build -o terraform-provider-hashicups
```



# Чтение ресурсов

# Определим структуру кофе

Узнаем структуру:

```
$ curl localhost:19090/coffees | jq
[
  {
    "id": 1,
    "name": "Packer Spiced Latte",
    "teaser": "Packed with goodness to spice up your images",
    "description": "",
    "price": 350,
    "image": "/packer.png",
    "ingredients": [{"ingredient_id": 1}, ... ]
  }, ...
]
```

# Создаем data\_source\_coffee.go

```
package hashicups

import (...)

func dataSourceCoffees() *schema.Resource {
    return &schema.Resource{
        ReadContext: dataSourceCoffeesRead,
        Schema: map[string]*schema.Schema{ /* ... */},
    }
}

func dataSourceCoffeesRead(ctx context.Context, d *schema.ResourceData, m
interface{}) diag.Diagnostics {
    /* ... */
}
```

# Декларируем структуру и функцию чтения

```
Schema: map[string]*schema.Schema{
    "coffees": &schema.Schema{
        Type:     schema.TypeList,
        Computed: true,
        Elem: &schema.Resource{
            Schema: map[string]*schema.Schema{
                "id": &schema.Schema{
                    Type:     schema.TypeInt,
                    Computed: true,
                },
            },
        },
    },
    ...
}
```

```
func dataSourceCoffeesRead(ctx context.Context, d *schema.ResourceData, m
interface{}) diag.Diagnostics { ...
}
```



# Авторизация

## provider.go: добавляем переменные

```
func Provider() *schema.Provider {
    return &schema.Provider{
        Schema: map[string]*schema.Schema{
            "username": &schema.Schema{
                Type:     schema.TypeString,
                Optional: true,
                DefaultFunc: schema.EnvDefaultFunc("HASHICUPS_USERNAME", nil),
            },
            "password": &schema.Schema{
                Type:     schema.TypeString,
                Optional: true,
                Sensitive: true,
                DefaultFunc: schema.EnvDefaultFunc("HASHICUPS_PASSWORD", nil),
            },
        },
        ResourcesMap: map[string]*schema.Resource{},
        DataSourcesMap: map[string]*schema.Resource{
            "hashicups_coffees": dataSourceCoffees(),
        },
    }
}
```



---

# Определение схемы

Обращаем внимание на:

- список переменных при определении провайдера,
- тип этих переменных,
- при помощи каких переменных окружений их можно задать,
- значение по-умолчанию.

## provider.go: определяем конфигурации

```
func providerConfigure(ctx context.Context, d *schema.ResourceData)
(interface{}, diag.Diagnostics) {

    username := d.Get("username").(string)

    password := d.Get("password").(string)

    ...

    return c, diags
}
```



# Создание ресурса

---

# Создадим заказ на кофе

Шаги:

- зарегистрировать ресурс для работы с **order**,
- определить схему данных для **order**,
- реализовать функцию создания **order**,
- реализовать функцию чтения **order**.

---

## resource\_order.go

```
func resourceOrder() *schema.Resource {  
    return &schema.Resource{  
        CreateContext: resourceOrderCreate,  
        ReadContext:    resourceOrderRead,  
        UpdateContext: resourceOrderUpdate,  
        DeleteContext: resourceOrderDelete,  
        Schema: map[string]*schema.Schema{  
        }  
    }  
}
```

## Декларируем схему

Протестировать создание заказа можно так:

```
$ curl -X POST -H "Authorization: ${HASHICUPS_TOKEN}"  
localhost:19090/orders -d '[{"coffee": { "id":1 }, "quantity":4}, {"coffee": { "id":3  
, "quantity":3}]'
```

И потом заменить `Schema: map[string]*schema.Schema{}`, на настоящую схему.

## Создаем функцию создания ресурса

```
func resourceOrderCreate(ctx context.Context, d *schema.ResourceData, m
interface{}) diag.Diagnostics {
    // ...
}
```

---

## Реализуем функцию чтения ресурса

```
func resourceOrderRead(ctx context.Context, d
*schema.ResourceData, m interface{}) diag.Diagnostics {
    // ...
}
```



## Добавляем order ресурс в провайдер

```
func Provider() *schema.Provider {  
    return &schema.Provider{  
        Schema: map[string]*schema.Schema{ /* ... */},  
        ResourcesMap: map[string]*schema.Resource{  
            "hashicups_order": resourceOrder(),  
        },  
        DataSourceMap: map[string]*schema.Resource{  
            "hashicups_coffees": dataSourceCoffees(),  
        },  
    }  
}
```



# Обновление ресурса



## Обновляем заказ на кофе

Шаги:

- реализуем функцию `resourceOrderUpdate()` в `resource_order.go`

---

## Функция resourceOrderUpdate()

```
func resourceOrderUpdate(ctx context.Context, d *schema.ResourceData,  
m interface{}) diag.Diagnostics {  
    // ...  
    return resourceOrderRead(ctx, d, m)  
}
```



# Удаление ресурса

---

## Удаляем заказ на кофе

Шаги:

- реализуем функцию `resourceOrderDelete()` в `resource_order.go`

---

## Функция resourceOrderDelete()

```
func resourceOrderDelete(ctx context.Context, d *schema.ResourceData, m
interface{}) diag.Diagnostics {
    // ...

    var diags diag.Diagnostics

    return diags
}
```

---

# Итоги

Что мы разобрали:

- как установить и работать с собственным провайдером,
- как устроен провайдер внутри,
- как реализовать авторизацию,
- как реализовать работу с ресурсом.





# Домашнее задание



# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и  
пишите отзыв о лекции!**

**Сергей Андрюнин**