Section 3

Managing your Infrastructure with the Terraform CLI

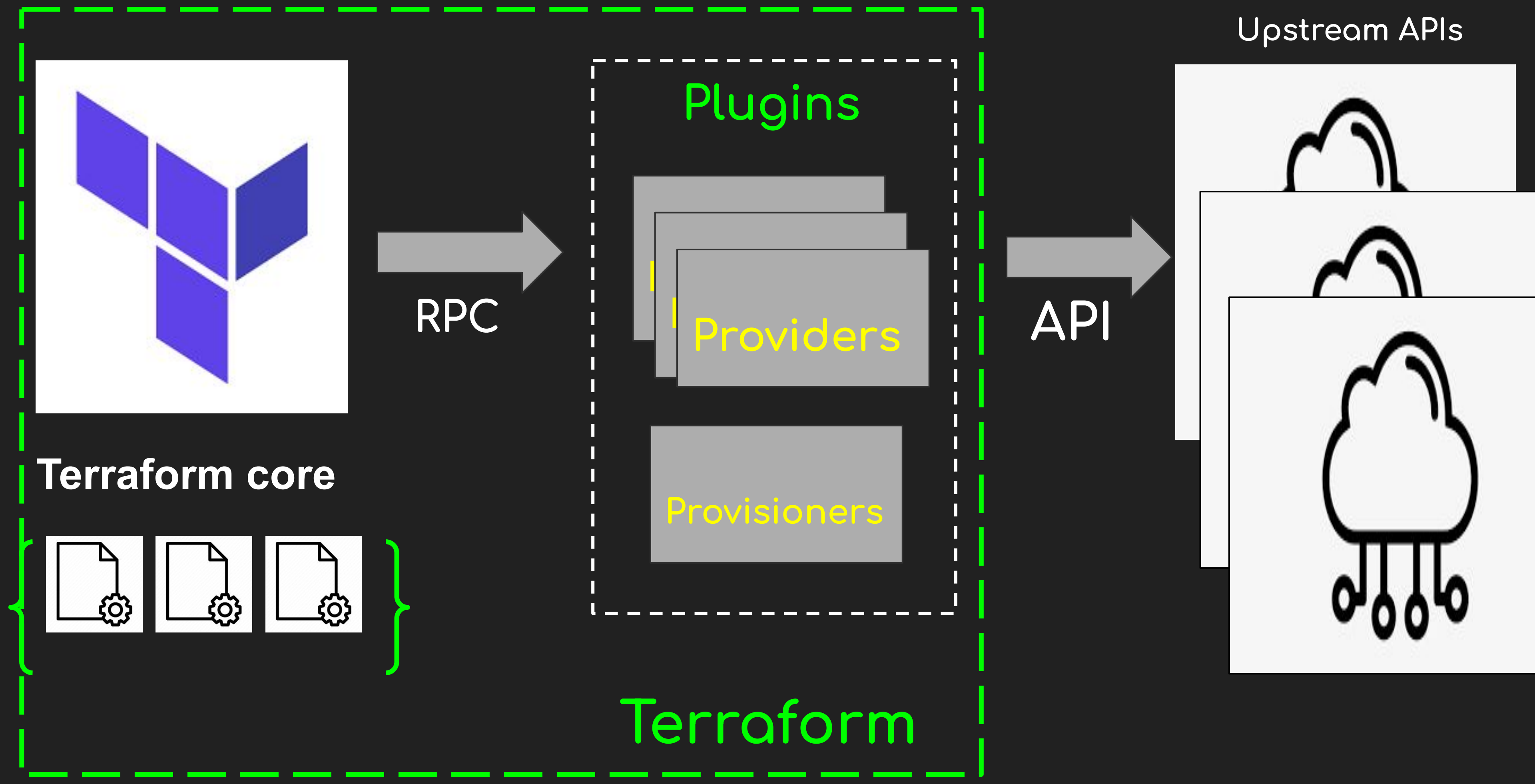
The Terraform Architecture

The Terraform Architecture

**TF == Terraform

Terraform is a plugin based architecture

"A Plugin or add-on is a software component that adds functionality to an existing computer program.


Upstream APIs
Plugins
Providers
Provisioners
RPC
API
Terraform core
Terraform

TF Core is the binary that provides the CLI and HCL.

It manages the configuration files and execution plan with plan() and apply()

The core interacts with the providers and provisioners plugins over RPC.

It ensure that the configuration desired end state is the same as the real world using a state file. The core will do a diff()

TF Providers plugins main job is to manage the communication with the infrastructure providers, expose the infrastructure resources and manage their lifecycle over the API

Lifecycle is create/read/update/delete

TF Provisioners plugins execute scripts or commands locally or remotely .

The terraform syntax

Terraform configuration files are composed of one or multiple code blocks

Each block is a container of code delimited or constrained with { }.

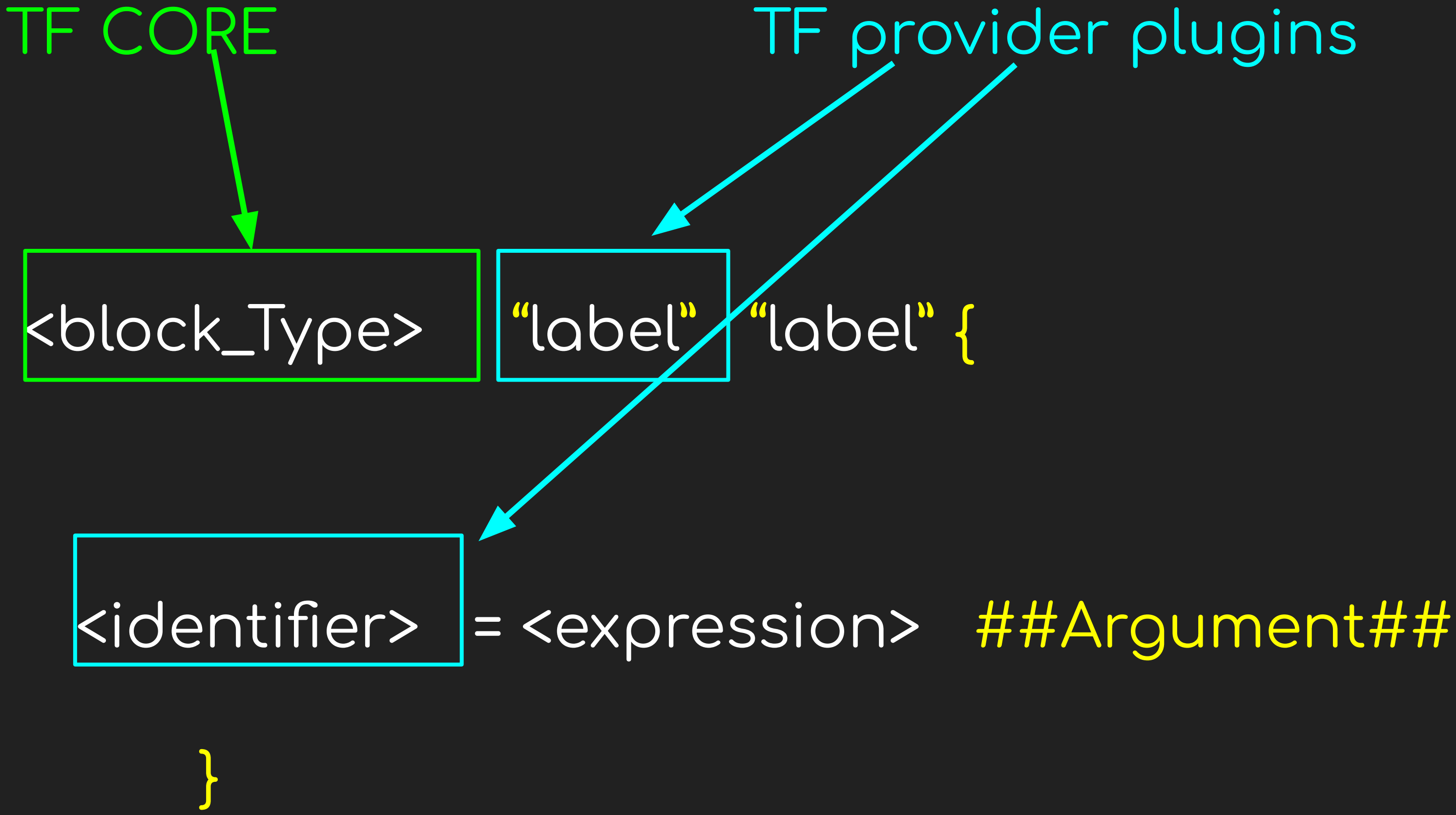
Multiple types of code blocks

- Resource, Data Source, Terraform, Provider, Variable ...

Syntax for Terraform code block:

```
<block_Type> “label” “label” {
      <identifier> = <expression>   ##Argument##
 }
```

- Comments can be added to a block using #, /* */ or //
- Labels are used for identification of the block type.
- Arguments to assign a value to a name



```
<block_Type>  "label"  "label" {

  <identifier>  = <expression>   ##Argument##

      }
```

Arguments vs Meta-Arguments

Arguments apply only to a specific block

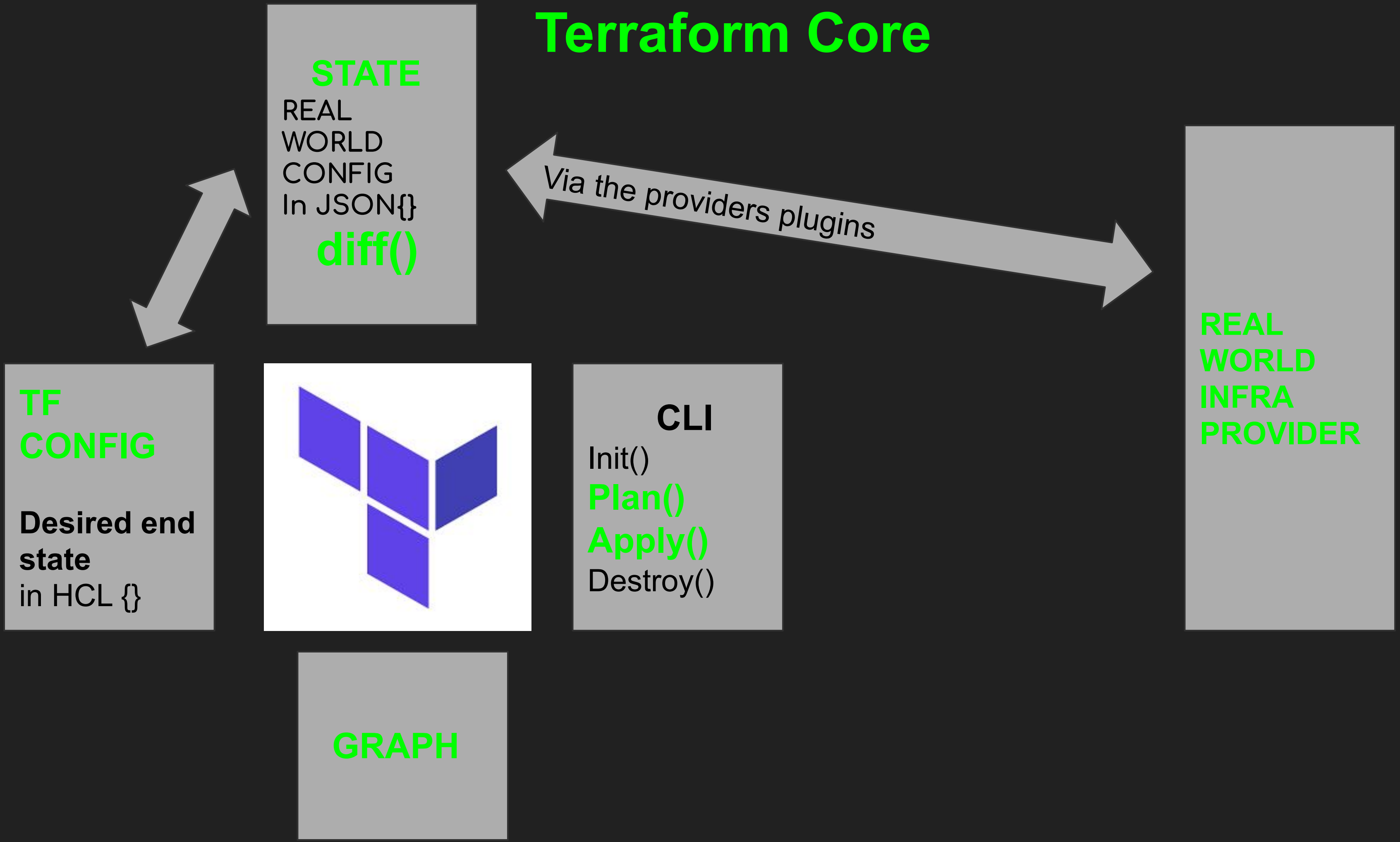
Meta-arguments apply to the all blocks.

```
  <block_Type> "label" "label" {

      <block_Type> {     ##Nested block:

          <identifier> = <expression>   ##only literal values.

        }

}
```

> Dynamic blocks will allow to accept non literal values like input variables.

The terraform Core

Terraform Core

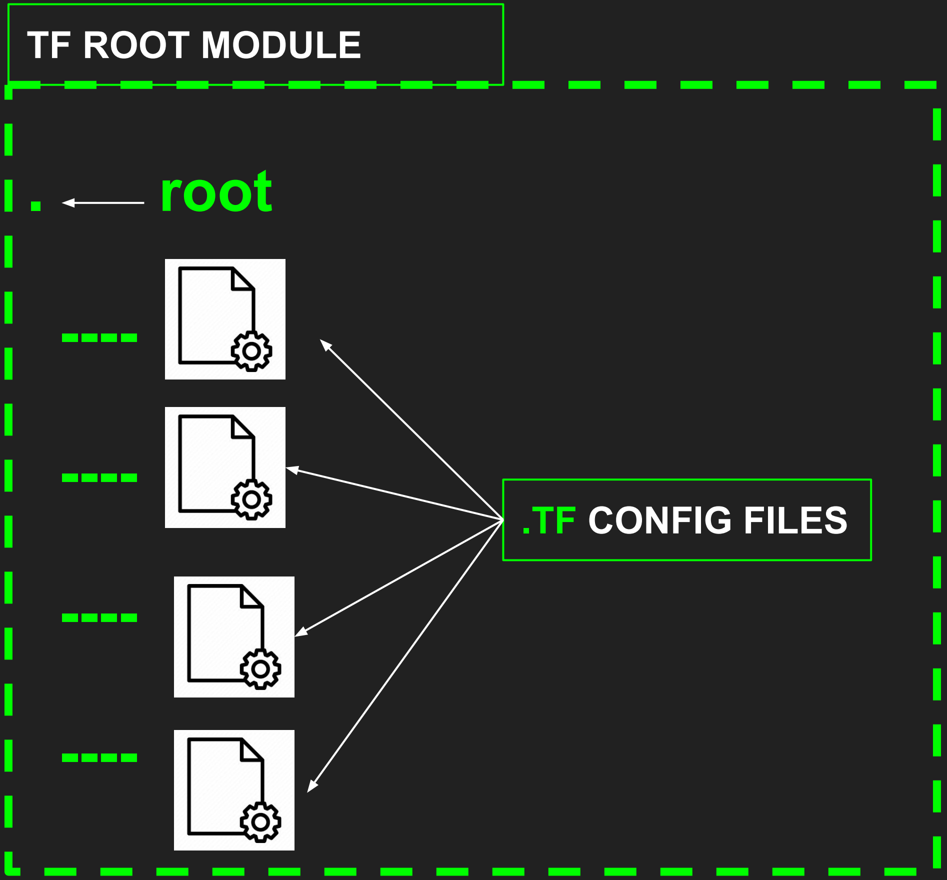

STATE
REAL WORLD CONFIG In JSON{}
diff()
Via the providers plugins
REAL WORLD INFRA PROVIDER
TF CONFIG
Desired end state in HCL {}
CLI
Init()
Plan()
Apply()
Destroy()
GRAPH

TF configuration files uses the .tf extension or can use json with .tf.json extension.

TF Configuration files are saved in a local directory called the root module.

Module == collection of TF configuration files

These .tf files can be saved in a Version Control System


TF ROOT MODULE
.
root
.TF CONFIG FILES

Terraform core is as single GO binary and can be installed on the following Operating Systems..

- Linux
- Windows
- MacOs
- Solaris
- FreeBSD
- OpenBSD

Terraform core is versioned using the 0.X.Y numbering system

The latest version is 0.13

no need to have GO installed as a prerequesite

Upgrade between major version could cause config breakage.

Terraform setting configurations is

It can be used to configure the Terraform core version, provider versions and backends.
Backend allows to define where the state is located.

Terraform { ... }

The terraform version is configured using the required_version setting using operators

= equal to
> < >= <= compared to
!= not equal to
~> pessimist constraint operator

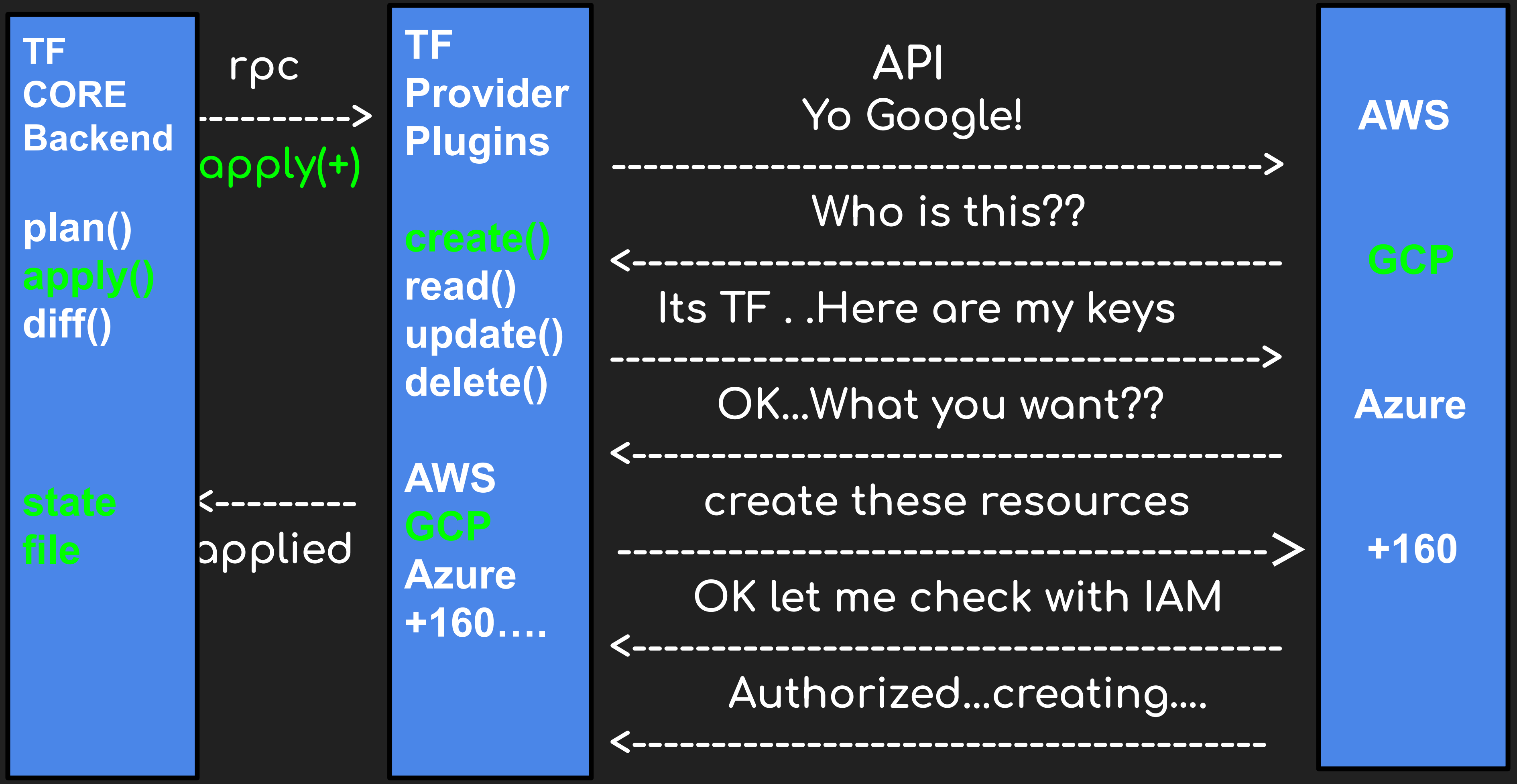
>> ~> 0.12.0 is version 0.12.0 to 0.12.29 or
>= 0.12.0 and < 0.13

The terraform Providers

A "Terraform provider plugin" represent a particular infrastructure provider using the plugin and the API...could be IaaS, PaaS or SaaS. 160+ and growing...

The Terraform Providers manages the lifecycle of infrastructure providers resources that are managed by Terraform using a CRUD operation:

create/read/update/delete



Yo Google!

Who is this??

Its TF . .Here are my keys

OK...What you want??

create these resources

OK let me check with IAM

Authorized...creating....

Provider configuration are configured with a provider block

The provider block will define how and where to provision and access resources.
For example:

- Region or Location
- Version **
- Credentials, Tenant id etc..

```
provider  “provider_name” {

        arguments
        ....
}
```

For example:

```
provider  “aws” {

    Region = “us-east-1”

}
```

Terraform does not install any provider plugin by default

Once the TF provider is configured and initialized then TF will download the provider plugin.

TF creates a .terraform/plugins directory in the root module where it will download the plugin binary for each TF provider.

NOTE: Only plugins from providers distributed by Hashicorp will be downloaded and 3rd party needs to be manually installed

Terraform providers plugins are versioned to extend functionality as quickly as possible and thus are versioned independently from the terraform core version.

By default the latest version will be set. It is possible to set the version of the provider plugin

```
 Provider "aws" {
    Version = "~> version_number"  }
```

** No longer recommended in 0.13

~> all minor within major version

> or >= any version higher than

< or<= any version lower than

The recommended way set the provider version under the terraform block using the required_providers block

```
terraform {
    required_providers {
        aws = {
            version = “~> 3.0”
            }
        }
    }
```

IMPORTANT: It is recommended not to hard code your provider credentials in a provider block

Terraform can read the aws credentials using the credential file in the ~/.aws directory create with aws configure.

The aws credentials can be set as well as environment variables using the follow command:

Export TF_VAR_<variable-name> in linux/macos

```
export TF_VAR_AWS_ACCESS_KEY_ID="anaccesskey"

export TF_VAR_AWS_SECRET_ACCESS_KEY="asecretkey"
```

Terraform providers command can be used to verify:

```
$ terraform providers
.
└── provider.aws
```

The latest version of providers and modules can be installed every time a terraform init using

```
$terraform init -upgrade
```

Using alias for many configurations in a provider.

Alias allows to support multiple regions in the same provider

This is done with one default provider and additional provider blocks with the alias argument.

The region can be referenced in a resource block with the provider meta-argument
Provider = provider_name.alias

```
provider  “aws” {
    region   =  “us-east-1”
    version  =  “~> 2.6”
 }


provider “aws” {
    alias    = “apac”
    region  =  “ap-south-1”
}

resource “type” “name” {
   provider =
provider_type.alias_name
  .......
 }
```

Multiple providers can be set on a per resource type or per module basis using the alias option.

The alias value can be anything you chose.

When not specified a resource will chose the non alias option

Terraform resources

A terraform resource block enables the configuration of one or more infrastructure objects

It is the most important configuration element of the Terraform configuration

A resource configuration block is defined between constraints { }

The resource block can use many resource block types that are defined by the TF provider plugin

Arguments are all the configurable options of the provider with some mandatary and some optional.

The resource configuration block type:

```
Resource “<resource_type>” “<local_name>” {
    <identifier> =  <expression>  → argument
}
```

There are several meta-arguments

- provider
- count
- depends_on
- for_each
- lifecycle
- provisioner and connection

Expressions can refer to Attributes which are a piece of data representing an object like the id of an vpc or the id of an ami

There can be dependencies between resources configuration blocks like the vpc id or security group id used by a EC2 instance

The dependencies will define the order of creation in the TF plan of execution.

The Terraform State

Terraform state file == real world infrastructure provider configuration

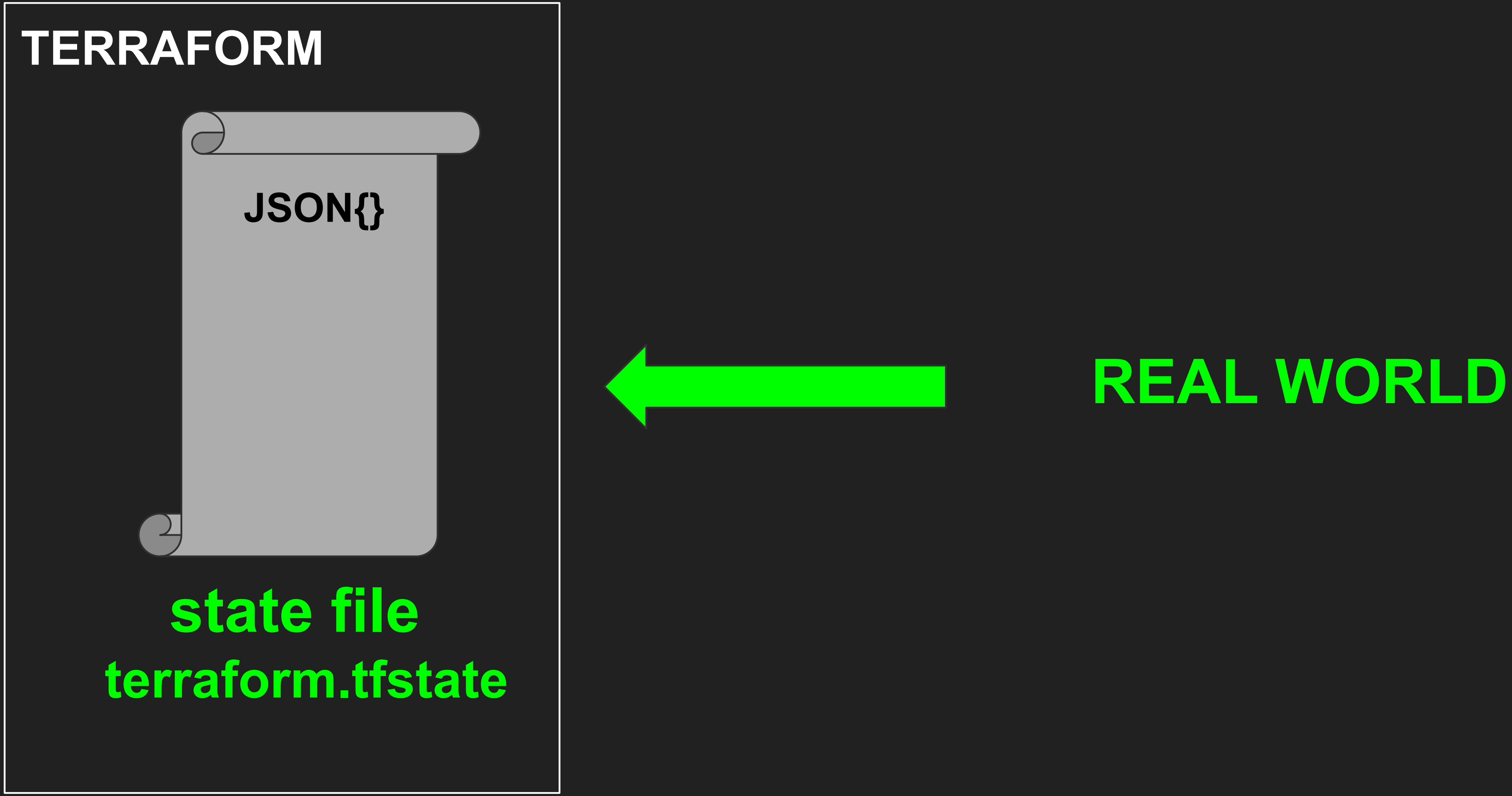
Created after terraform apply command is issued to provision infrastructure

all the provisioned objects in the infrastructure provider with all their attributes are captured in the state file

Terraform uses the state file to compare the desired end state to the real world.

This enables Terraform to be idempotent as without the state file, Terraform will have to destroy and recreate ALL resources in the provider anytime Terraform creates or updates a resource.

The TF state improves performance for large Infras!


TERRAFORM
JSON{}
state file
terraform.tfstate
REAL WORLD

The state file and its backup are automatically created in a json{} format:
Terraform.tfstate and terraform.tfstate.backup

You can give it another name like mystate.tfstate but need to to pass it using -state flag with the terraform plan and apply commands.

The Terraform Backend stores the state file in root module by default but can be stored remotely.

NOT encrypted at rest in root module

State should NOT be edited manually and only using $terraform state command

The state file content can be viewed with the $terraform show command

The $terraform refresh command can be used to determine the actual state and automatically update the state file but does not provision anything!.

Terraform plan and apply include that as well.