Introduction to AWS With Terraform

Almost one month ago, I released Practical AWS, a training concerned With the actual use of AWS rather than with theory & ideas.

Even if the training is released, but since it will be lifetime updated with new contents for free, this blog post is a prototype of the new lesson that will be added to the training.

This blog post is an introduction to managing an AWS infrastructure using Terraform.

Downloading & Installing Terraform

Start by downloading Terraform from [the official download page](https://www.terraform.io/downloads.html).

Choose your OS and CPU architecture and start the download.

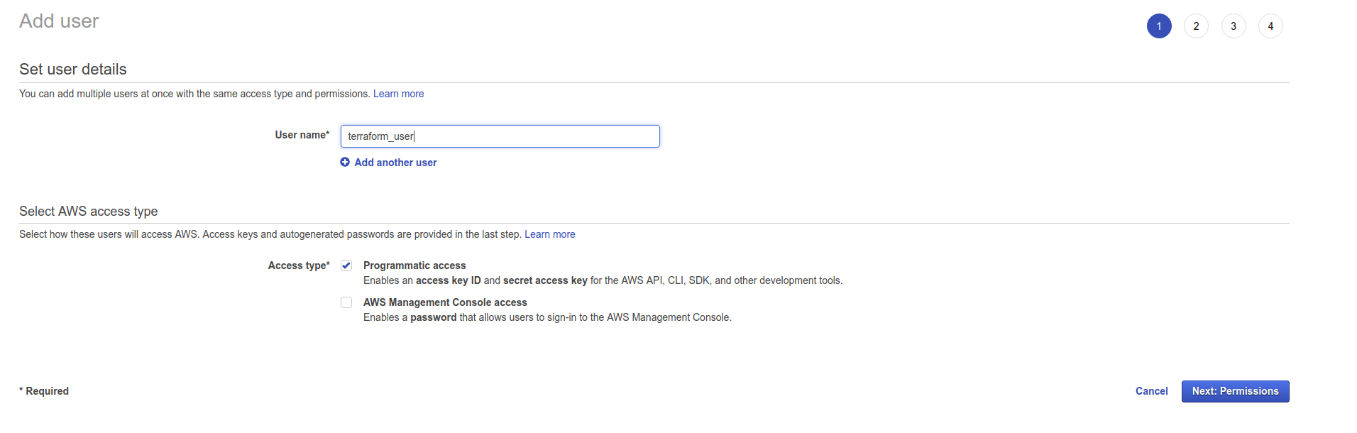
Terraform is a single binary that you should move to /usr/bin and make it executable.

sudo mv terraform /usr/bin && sudo chmod +x

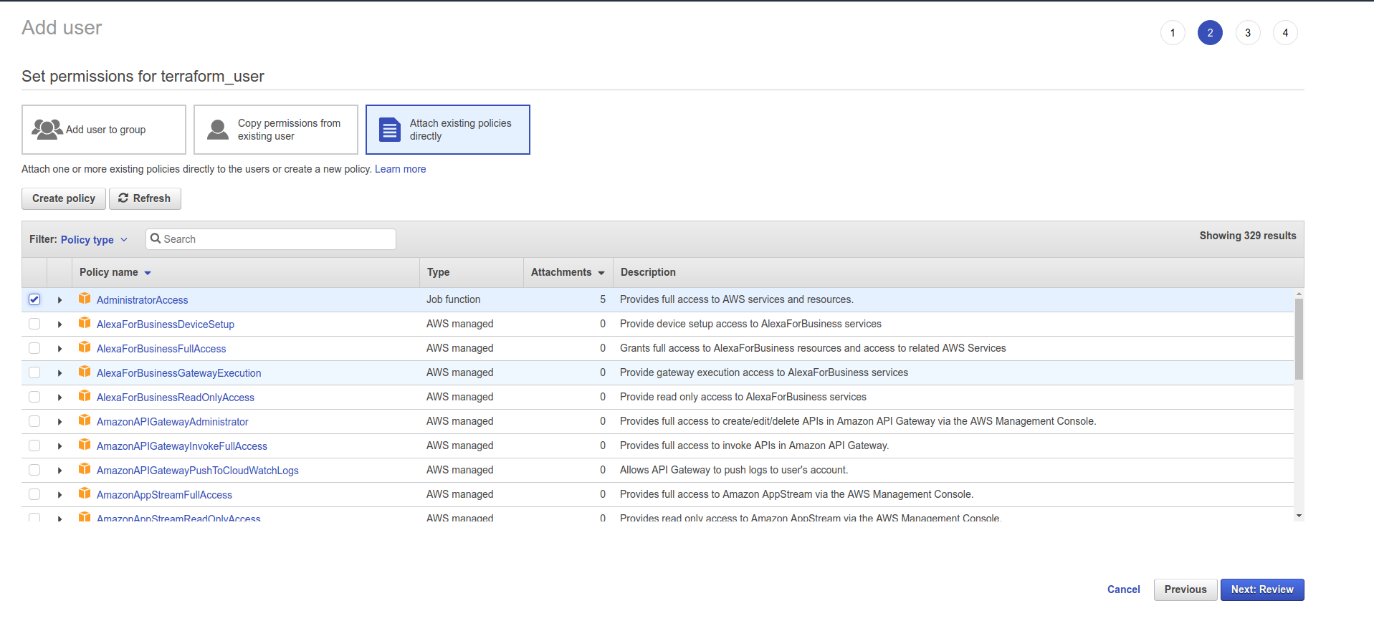
If you are using another OS, please refer to [the documentation](http://dome.readthedocs.io/en/latest/dmc/deployment/install-terra.html).

Configuring AWS

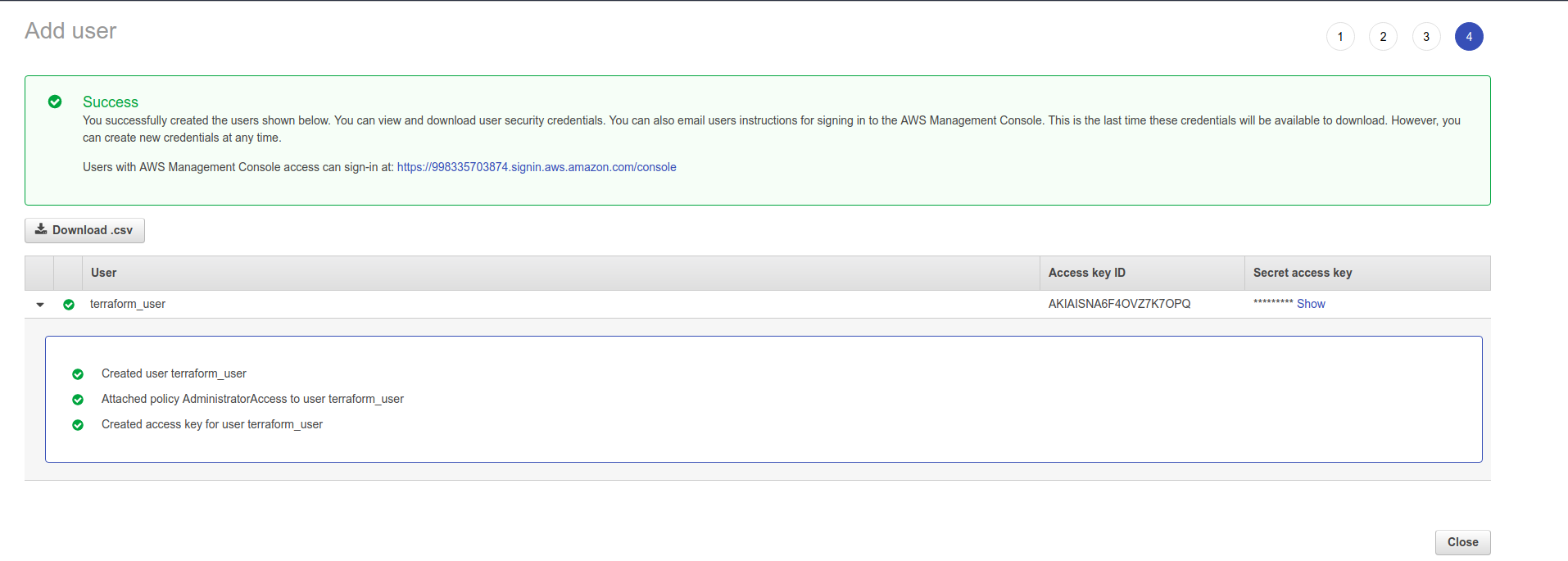
In order to follow the best practices, let’s create a user for Terraform. Go to your AWS console and create terraform\_user user:



Give it the good rights. In my example, I need Terraform to be able to manage all of my AWS Cloud resources:



Don’t forget to store the AWS access key id and secret access key:



Copy them in your AWS credential file:

.aws/credentials

You can also execute aws configure to add a new user.

In both cases, your keys will be stored in the AWS credentials file:

**[terraform]**  
aws\_access\_key\_id = xxxxxxxxxxxxxxxxxxx

aws\_secret\_access\_key = xxx/xxxxxxxxxxxxx/xxxx

Terraform Hello World !

Go to your workspace and create a folder called terraform:

**mkdir** terraform

Add these lines to main.tf :

**provider** "aws" {

**region** = "eu-west-3"

shared\_credentials\_file = "/home/eon01/.aws/credentials"  
 profile = "terraform"  
}

The above is the configuration for AWS, adapt the credential file path to your own configuration, the profile name, as well as the region. In my example, I am using the Paris region.

In order to create our first AWS machine, let’s add these lines:

**resource** "aws\_instance" "web" {  
 **ami** = "ami-0e55e373"  
 instance\_type = "t1.micro"  
 tags {  
 **Name** = "eralabs"  
 }

Our file main.tf will look like this:

**provider** "aws" {  
 **region** = "eu-west-3"  
 shared\_credentials\_file = "/home/eon01/.aws/credentials"  
 profile = "terraform"  
}

resource "aws\_instance" "web" {  
 **ami** = "ami-0e55e373"  
 instance\_type = "t1.micro"  
 tags {  
 **Name** = "eralabs"  
 }

}

In the example above, I am creating a machine on the region “eu-west-3” using the profile “terraform”.

My machine size is t1.micro and it is using the AMI ami-0e55e373 , which is a Ubuntu 17.04 image available for the region “eu-west-3”.

**Note**: Ubuntu 17.04 image doesn’t have the same AMI id in two different regions.

If you prefer using Ubuntu like in this example, you can visit cloud-images.ubuntu.com where you can find the id of the AMI you should use.

You can also use the CLI in order to describe AWS IAMs:

e.g: Describing Windows AMIs that are backed by Amazon EBS.

**aws** ec2 describe-images --filters "Name=platform,Values=windows" "Name=root-device-type,Values=ebs"

e.g: Describing Ubuntu AMIs

**aws** ec2 describe-images --filters "Name=name,Values=ubuntu\*"

**Note:**Check the AWS cheat sheet that comes with this training in order to get more examples.

After choosing the AMI, go into the folder where you created main.tf and initialize Terraform:

**terraform** init

You will see a similar output to this one:

Initializing provider plugins...  
- Checking for available provider plugins on https://releases.hashicorp.com...  
- Downloading plugin for provider "aws" (1.13.0)...

The following providers **do** **not** have **any** **version** **constraints** **in** configuration,  
so the latest **version** was installed.

**To** prevent automatic upgrades **to** new major versions that may contain breaking  
changes, it **is** recommended **to** **add** **version** = "..." **constraints** **to** the  
**corresponding** provider blocks **in** configuration, **with** the **constraint** strings  
suggested below.

\* provider.aws: **version** = "~> 1.13"

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.

Now execute:

**terraform** plan

This command will not create any resource on your AWS cloud. It lets you know what Terraform will do.

You will see this output:

+ aws\_instance.web  
 id: <computed>  
 ami: "ami-0e55e373"  
 associate\_public\_ip\_address: <computed>  
 availability\_zone: <computed>  
 ebs\_block\_device.#: <computed>  
 ephemeral\_block\_device.#: <computed>  
 get\_password\_data: "false"  
 instance\_state: <computed>  
 instance\_type: "t1.micro"  
 ipv6\_address\_count: <computed>  
 ipv6\_addresses.#: <computed>  
 key\_name: <computed>  
 network\_interface.#: <computed>  
 network\_interface\_id: <computed>  
 password\_data: <computed>  
 placement\_group: <computed>  
 primary\_network\_interface\_id: <computed>  
 private\_dns: <computed>  
 private\_ip: <computed>  
 public\_dns: <computed>  
 public\_ip: <computed>  
 root\_block\_device.#: <computed>  
 security\_groups.#: <computed>  
 source\_dest\_check: "true"  
 subnet\_id: <computed>  
 tags.%: "1"  
 tags.Name: "eralabs"  
 tenancy: <computed>  
 volume\_tags.%: <computed>  
 vpc\_security\_group\_ids.#: <computed>

Note that the (+) sign indicates that a resource will be created. In the other hand, when showing a minus sign (-), Terraform means that a resource will be deleted.

Working With Variables

Let’s discover how to use Terraform variables to write a cleaner configuration file.

We can consider that the AWS region could be variable, that’s why we are going to add this code to the main.tf file:

**variable** "region" {  
 **default** = "eu-west-3"  
}

You can now call it from a Terraform file using:

${var.region}

This how our main.tf will look like:

**variable** "region" {  
 **default** = "eu-west-3"  
}

provider "aws" {  
 **region** = "${var.region}"  
 shared\_credentials\_file = "/home/eon01/.aws/credentials"  
 profile = "terraform"  
}

resource "aws\_instance" "web" {  
 **ami** = "ami-0e55e373"  
 instance\_type = "t1.micro"  
 tags {  
 **Name** = "eralabs"  
 }

}

Right !

Let’s do the same thing to “shared\_credentials\_file” and “profile”:

**variable** "region" {  
 **default** = "eu-west-3"  
}

variable "shared\_credentials\_file" {  
 **default** = "/home/eon01/.aws/credentials"  
}

variable "profile" {  
 **default** = "terraform"  
}

provider "aws" {  
 **region** = "${var.region}"  
 shared\_credentials\_file = "${var.shared\_credentials\_file}"  
 profile = "${var.profile}"  
}

resource "aws\_instance" "web" {  
 **ami** = "ami-0e55e373"  
 instance\_type = "t1.micro"  
 tags {  
 **Name** = "eralabs"  
 }

}

Using Terraform Maps

In my example, I am using the Paris region (eu-west-3) but what if I need to add new regions like Dublin (eu-west1) for instance !?

The above code will deploy an EC2 instance to a single region.

In order to seolve this problem, the first step to follow here, is finding the AMI we want to use (depending on the region) and then create a variable with the type “map”:

variable "my\_ami" {  
 type = "map"  
 default = {  
 eu-west-1 = "ami-f90a4880"  
 eu-west-3 = "ami-0e55e373"  
 }

description = "I added only 2 regions: Paris and Dublin. You can use as many regions as you want."

}

According to the used region Terraform should create an EC2 machine with a different AMI.

This is done by changing the old AMI line by changing ami = "ami-0e55e373" to ami = "${lookup(var.my\_ami, var.region)}" .

To test this, type terraform plan and you will get this output:

+ aws\_instance.web  
 id: <computed>  
 ami: "ami-0e55e373"  
 associate\_public\_ip\_address: <computed>  
 availability\_zone: <computed>  
 ebs\_block\_device.#: <computed>  
 ephemeral\_block\_device.#: <computed>  
 get\_password\_data: "false"  
 instance\_state: <computed>  
 instance\_type: "t1.micro"  
 ipv6\_address\_count: <computed>  
 ipv6\_addresses.#: <computed>  
 key\_name: <computed>  
 network\_interface.#: <computed>  
 network\_interface\_id: <computed>  
 password\_data: <computed>  
 placement\_group: <computed>  
 primary\_network\_interface\_id: <computed>  
 private\_dns: <computed>  
 private\_ip: <computed>  
 public\_dns: <computed>  
 public\_ip: <computed>  
 root\_block\_device.#: <computed>  
 security\_groups.#: <computed>  
 source\_dest\_check: "true"  
 subnet\_id: <computed>  
 tags.%: "1"  
 tags.Name: "eralabs"  
 tenancy: <computed>  
 volume\_tags.%: <computed>  
 vpc\_security\_group\_ids.#: <computed>

If you manually change the region to eu-west-1 , you will notice that terraform plan will use the other AMI:

+ aws\_instance.web  
 id: <computed>  
 ami: "ami-f90a4880"  
 associate\_public\_ip\_address: <computed>  
 availability\_zone: <computed>  
 ebs\_block\_device.#: <computed>  
 ephemeral\_block\_device.#: <computed>  
 get\_password\_data: "false"  
 instance\_state: <computed>  
 instance\_type: "t1.micro"  
 ipv6\_address\_count: <computed>  
 ipv6\_addresses.#: <computed>  
 key\_name: <computed>  
 network\_interface.#: <computed>  
 network\_interface\_id: <computed>  
 password\_data: <computed>  
 placement\_group: <computed>  
 primary\_network\_interface\_id: <computed>  
 private\_dns: <computed>  
 private\_ip: <computed>  
 public\_dns: <computed>  
 public\_ip: <computed>  
 root\_block\_device.#: <computed>  
 security\_groups.#: <computed>  
 source\_dest\_check: "true"  
 subnet\_id: <computed>  
 tags.%: "1"  
 tags.Name: "eralabs"  
 tenancy: <computed>  
 volume\_tags.%: <computed>  
 vpc\_security\_group\_ids.#: <computed>

Using Input Arguments

In the latest example we changed the value of the region from “eu-west-3” to “eu-west-1” manually. The goal was testing if the map function was working right.

In practice, you don’t need to manually change your main.tf file, but you can override the value of region by using a new region as an argument:

Try using “eu-west-1” instead of the default value “eu-west-3”:

terraform plan -var region=eu-west-1

It is possible to input other variables in the same line. As an example, we can change the used profile from “terraform” to “default” using the following command:

terraform plan -var region=eu-west-1 -var profile=default

Using Variable Files

We want to separate the configuration from the execution code, that’s why we are going to create a file containing the variables we are using (variables.tfvars):

region = "eu-west-1"

shared\_credentials\_file = "/home/eon01/.aws/credentials"  
profile = "terraform"

my\_ami = {  
 "eu-west-1" = "ami-f90a4880"  
 "eu-west-3" = "ami-0e55e373"  
 }

After removing the variables from the main.tf file, this is how it becomes:

**variable** "region" {}  
**variable** "shared\_credentials\_file" {}  
**variable** "profile" {}  
**variable** "my\_ami" {  
 **type** = "map"  
}  
provider "aws" {  
 **region** = "${var.region}"  
 shared\_credentials\_file = "${var.shared\_credentials\_file}"  
 profile = "${var.profile}"  
}

resource "aws\_instance" "web" {  
 **ami** = "${lookup(var.my\_ami, var.region)}"  
 instance\_type = "t1.micro"  
 tags {  
 **Name** = "eralabs"  
 }

}

Let’s execute the plan command to see if there the EC2 machine will be created:

terraform plan -var-file=variables.tfvars

Terraform Apply

In order to execute create our EC2 machine, we need to execute: terraform apply .   
Because we are using a file to store our variables, we need to execute:

terraform apply -var-file=variables.tfvars

After executing the command above, we will have a similar output to this one:

An execution plan has been generated and is shown below.  
Resource actions are indicated with the following symbols:  
 + create

Terraform will perform the following actions:

+ aws\_instance.web  
 id: <computed>  
 ami: "ami-f90a4880"  
 associate\_public\_ip\_address: <computed>  
 availability\_zone: <computed>  
 ebs\_block\_device.#: <computed>  
 ephemeral\_block\_device.#: <computed>  
 get\_password\_data: "false"  
 instance\_state: <computed>  
 instance\_type: "t1.micro"  
 ipv6\_address\_count: <computed>  
 ipv6\_addresses.#: <computed>  
 key\_name: <computed>  
 network\_interface.#: <computed>  
 network\_interface\_id: <computed>  
 password\_data: <computed>  
 placement\_group: <computed>  
 primary\_network\_interface\_id: <computed>  
 private\_dns: <computed>  
 private\_ip: <computed>  
 public\_dns: <computed>  
 public\_ip: <computed>  
 root\_block\_device.#: <computed>  
 security\_groups.#: <computed>  
 source\_dest\_check: "true"  
 subnet\_id: <computed>  
 tags.%: "1"  
 tags.Name: "eralabs"  
 tenancy: <computed>  
 volume\_tags.%: <computed>  
 vpc\_security\_group\_ids.#: <computed>

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

Do you want to perform these actions?  
 Terraform will perform the actions described above.  
 Only 'yes' will be accepted to approve.

You will be asked to confirm, enter “yes”:

Enter a value: yes

aws\_instance.web: Creating...  
 ami: "" => "ami-f90a4880"  
 associate\_public\_ip\_address: "" => "<computed>"  
 availability\_zone: "" => "<computed>"  
 ebs\_block\_device.#: "" => "<computed>"  
 ephemeral\_block\_device.#: "" => "<computed>"  
 get\_password\_data: "" => "false"  
 instance\_state: "" => "<computed>"  
 instance\_type: "" => "t1.micro"  
 ipv6\_address\_count: "" => "<computed>"  
 ipv6\_addresses.#: "" => "<computed>"  
 key\_name: "" => "<computed>"  
 network\_interface.#: "" => "<computed>"  
 network\_interface\_id: "" => "<computed>"  
 password\_data: "" => "<computed>"  
 placement\_group: "" => "<computed>"  
 primary\_network\_interface\_id: "" => "<computed>"  
 private\_dns: "" => "<computed>"  
 private\_ip: "" => "<computed>"  
 public\_dns: "" => "<computed>"  
 public\_ip: "" => "<computed>"  
 root\_block\_device.#: "" => "<computed>"  
 security\_groups.#: "" => "<computed>"  
 source\_dest\_check: "" => "true"  
 subnet\_id: "" => "<computed>"  
 tags.%: "" => "1"  
 tags.Name: "" => "eralabs"  
 tenancy: "" => "<computed>"  
 volume\_tags.%: "" => "<computed>"  
 vpc\_security\_group\_ids.#: "" => "<computed>"  
aws\_instance.web: Still creating... (10s elapsed)  
aws\_instance.web: Creation complete after 19s (ID: i-055aaa2cab2436ab4)

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

Terraform & Immutable Infrastructure

To simply define this concept, an immutable resource or component is replaced for every deployment. For instance, servers are never modified after the deployment. When an updated is needed, a new server should be created from a base/common image with the new updates.

In order to see this in practice, I made it explicit to forget adding the SSH key to the EC2 description file, without it you can create an EC2 machine but you can’t access it using SSH. Let’s now add a key pair to the EC2 machine:

In the main.tf file:

**resource** "aws\_instance" "web" {  
 **ami** = "${lookup(var.my\_ami, var.region)}"  
 instance\_type = "t1.micro"

key\_name = "${var.key\_name}"

tags {  
 **Name** = "eralabs"  
 }

In variables.tfvs file:

**key\_name** = "my\_key.kp"

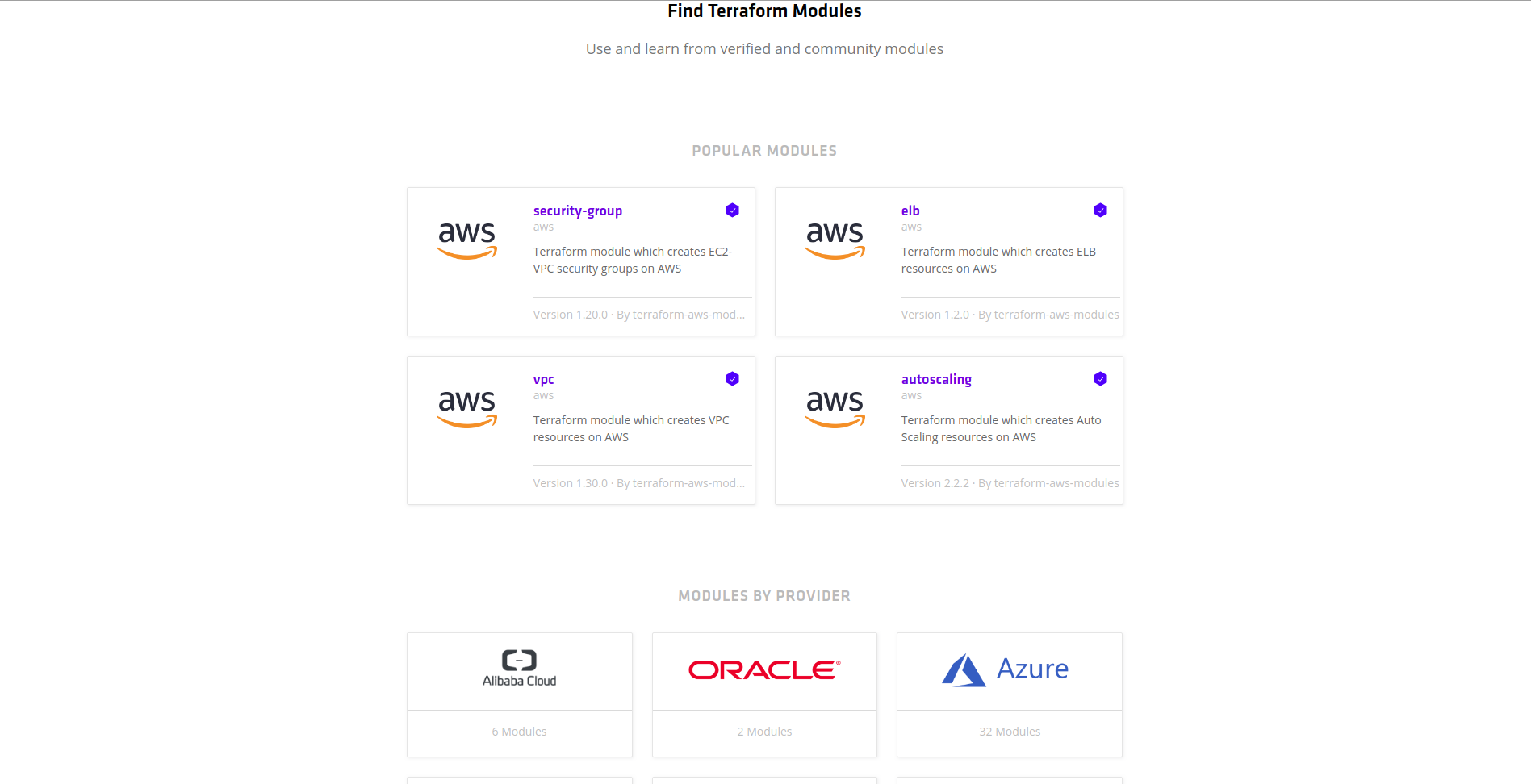
Now you can execute the plan command than the apply command and you will notice that Terraform will not update the machine to add the new key but will destroy it and create a new one with a new configuration:

-/+ aws\_instance.web (**new** resource required)  
 id: "i-055aaa2cab2436ab4" => <computed> (forces **new** resource)  
 ami: "ami-f90a4880" => "ami-f90a4880"  
 associate\_public\_ip\_address: "true" => <computed>  
 availability\_zone: "eu-west-1a" => <computed>  
 ebs\_block\_device.#: "0" => <computed>  
 ephemeral\_block\_device.#: "0" => <computed>  
 get\_password\_data: "false" => "false"  
 instance\_state: "running" => <computed>  
 instance\_type: "t1.micro" => "t1.micro"  
 ipv6\_address\_count: "" => <computed>  
 ipv6\_addresses.#: "0" => <computed>  
 key\_name: "" => "my\_key.kp" (forces **new** resource)  
 network\_interface.#: "0" => <computed>  
 network\_interface\_id: "eni-ddcf2fd9" => <computed>  
 password\_data: "" => <computed>  
 placement\_group: "" => <computed>  
 primary\_network\_interface\_id: "eni-ddcf2fd9" => <computed>  
 private\_dns: "ip-172-31-43-75.eu-west-1.compute.internal" => <computed>  
 private\_ip: "172.31.43.75" => <computed>  
 public\_dns: "ec2-52-211-29-100.eu-west-1.compute.amazonaws.com" => <computed>  
 public\_ip: "52.211.29.100" => <computed>  
 root\_block\_device.#: "1" => <computed>  
 security\_groups.#: "1" => <computed>  
 source\_dest\_check: "true" => "true"  
 subnet\_id: "subnet-7f510f27" => <computed>  
 tags.%: "1" => "1"  
 tags.Name: "eralabs" => "eralabs"  
 tenancy: "default" => <computed>  
 volume\_tags.%: "0" => <computed>  
 vpc\_security\_group\_ids.#: "1" => <computed>

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

Using Terraform Modules

Terraform hots [a public registry](https://registry.terraform.io/)where you can find common reusable modules.



You can use this registry modules for your projects. [Some of them](https://registry.terraform.io/browse?verified=true)are verified by HashiCorp, the company behind Terraform.

In order to put this in practice, we are going to do the same operation(creating an EC2 machine), but using [this module](https://registry.terraform.io/modules/terraform-aws-modules/ec2-instance/aws/1.5.0).

Create a new file (for example: main2.tf) and add these lines:

**module** "ec2\_cluster" {  
 **source** = "terraform-aws-modules/ec2-instance/aws"

name = "my-cluster"  
 instance\_count = 5

ami = "ami-ebd02392"  
 instance\_type = "t2.micro"  
 key\_name = "user1"  
 monitoring = true  
 vpc\_security\_group\_ids = ["sg-12345678"]  
 subnet\_id = "subnet-eddcdzz4"

tags = {  
 **Terraform** = "true"  
 Environment = "dev"  
 }  
}

Now type terraform init and the module files will be downloaded. You can use the plan then the apply command.