Erlang Solutions

Infrastructure as Code with Terraform

9/2023

WHAT WE HAVE DONE IN LAST MEETING

We know about Infrastructure as Code (IaC)

We know what is Terraform and the reason to use it

We know how Terraform works, Terraform's workflow

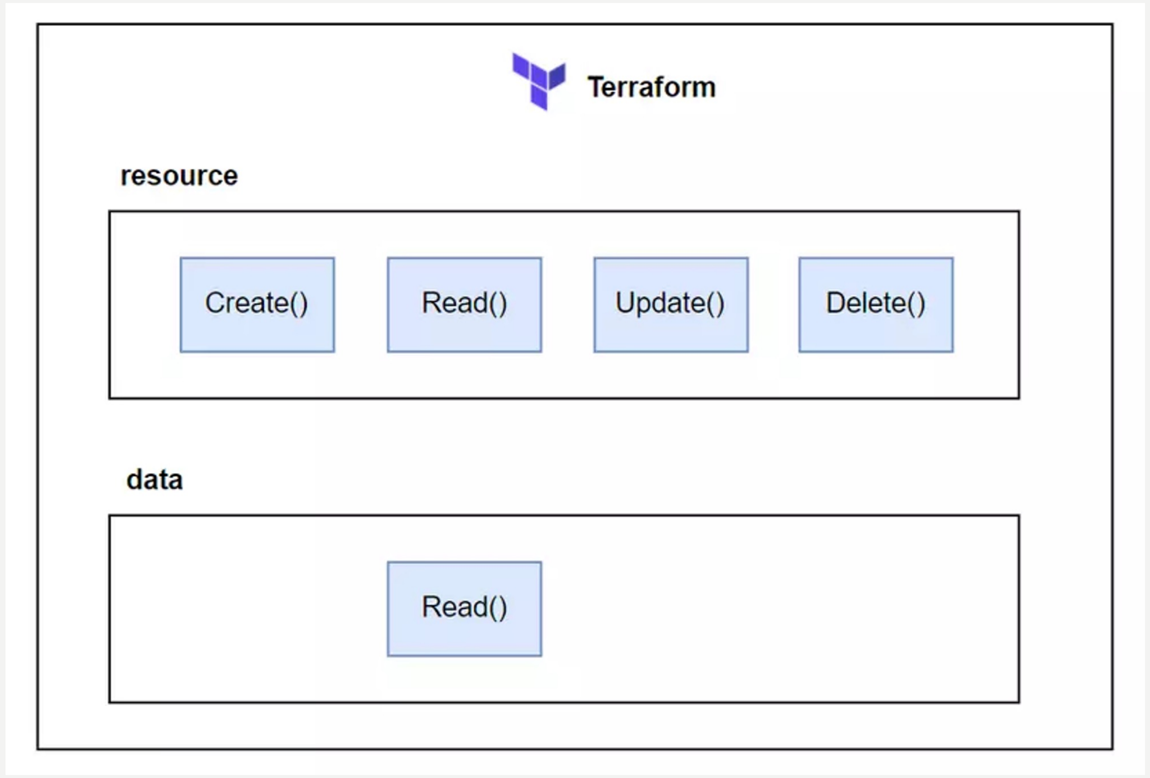
Terraform VS similar technologies

Getting started with Terraform - Demo

For this week, we continue to discover Terraform with the below Agenda

- CRUD operations (create, read, update, delete) overview
- Example about an AWS Simple Cloud Storage (S3): Plan, Create, No-op, Update, Delete.
- Resource Drift
- Summary about the Terraform life circle
- What's next?

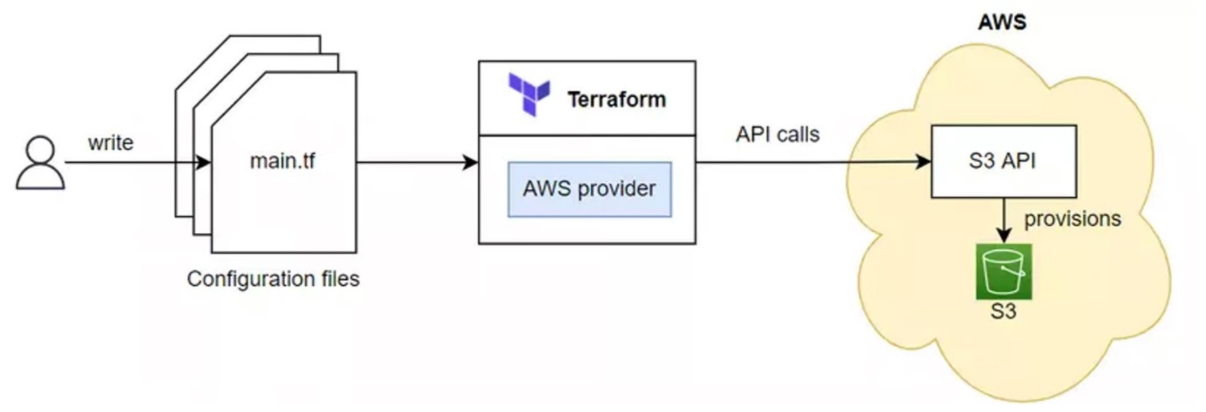
CRUD OPERATIONS

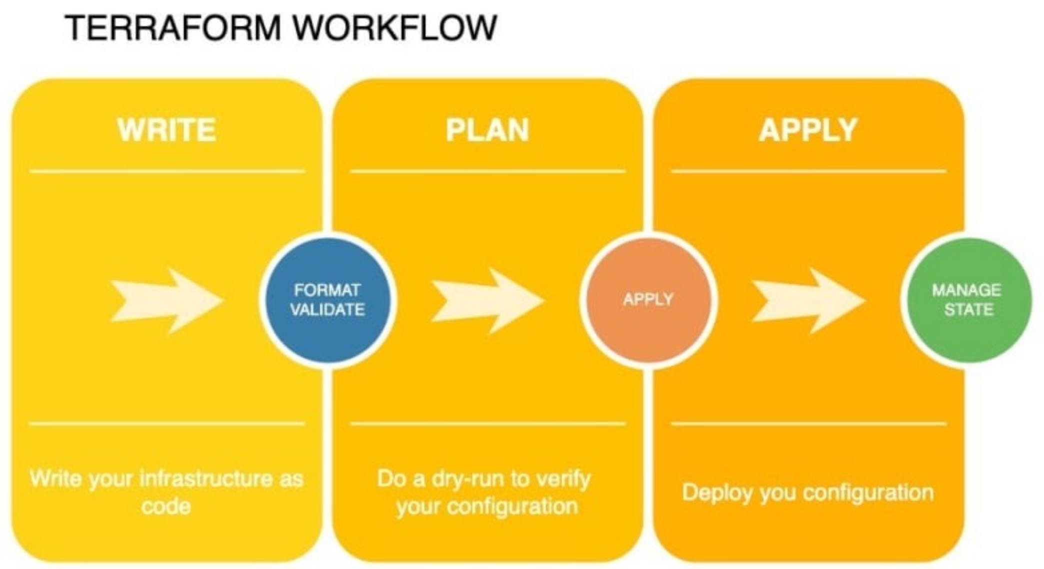


- All resource types of Terraform implement a CRUD Interface.
- In this CRUD Interface there will be functions like Create(), Read(), Update(), Delete() and these functions will be executed if the right conditions are met.
- The two resources in Terraform: a managed resource and an unmanaged data source. The managed resource implements full CRUD, while the data source only implements Read().

Let’s do a demo about AWS simple cloud storage (S3)

We will use Terraform to create an S3 (AWS Simple Cloud Storage) on AWS to learn about the lifecycle of a resource.



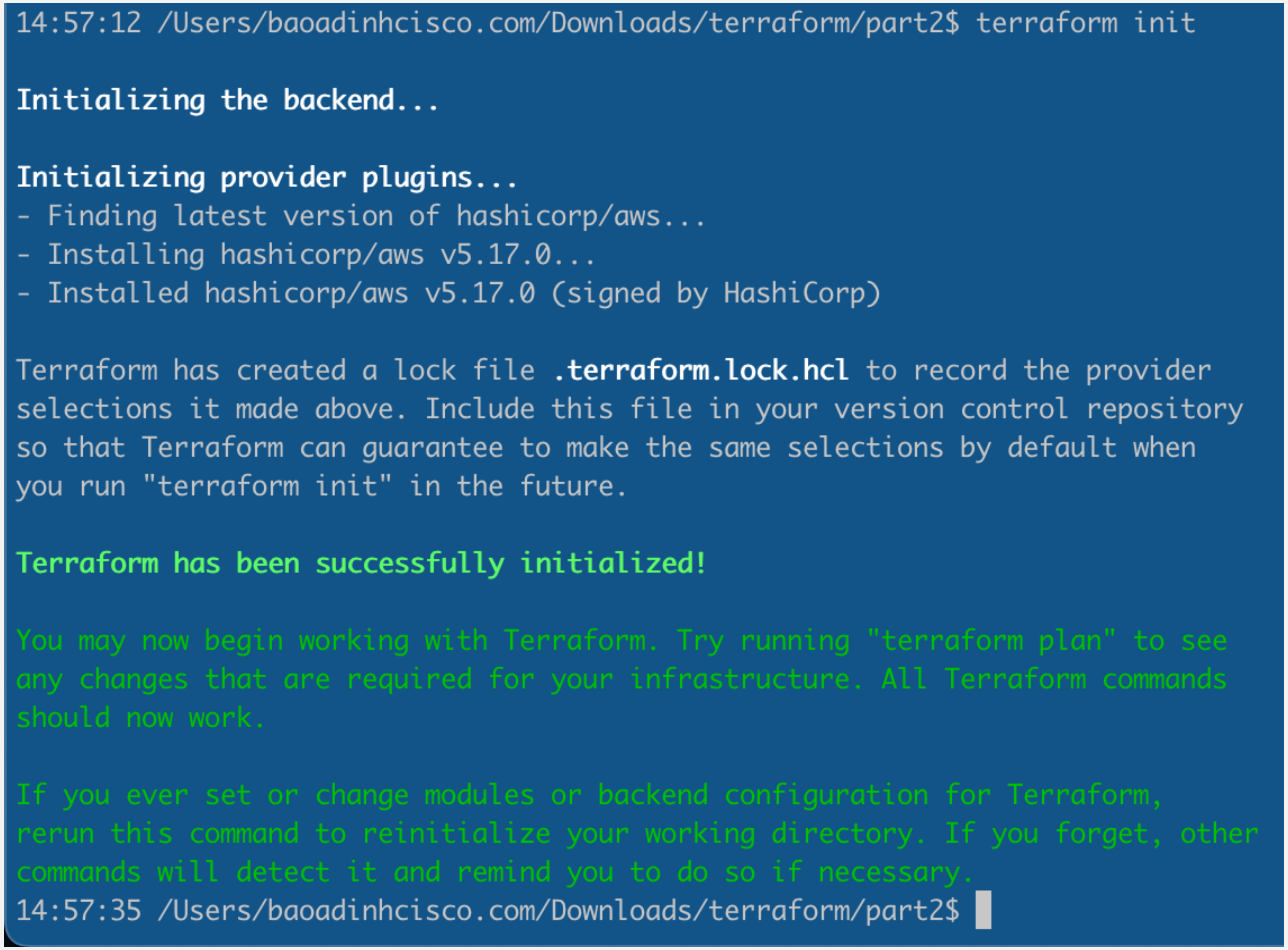


Declaring the configuration file

Now we will write the Terraform file to create S3 and talk about each of the above functions. Create a Wordspace named S3, then create a file named `main.tf` with the following code:

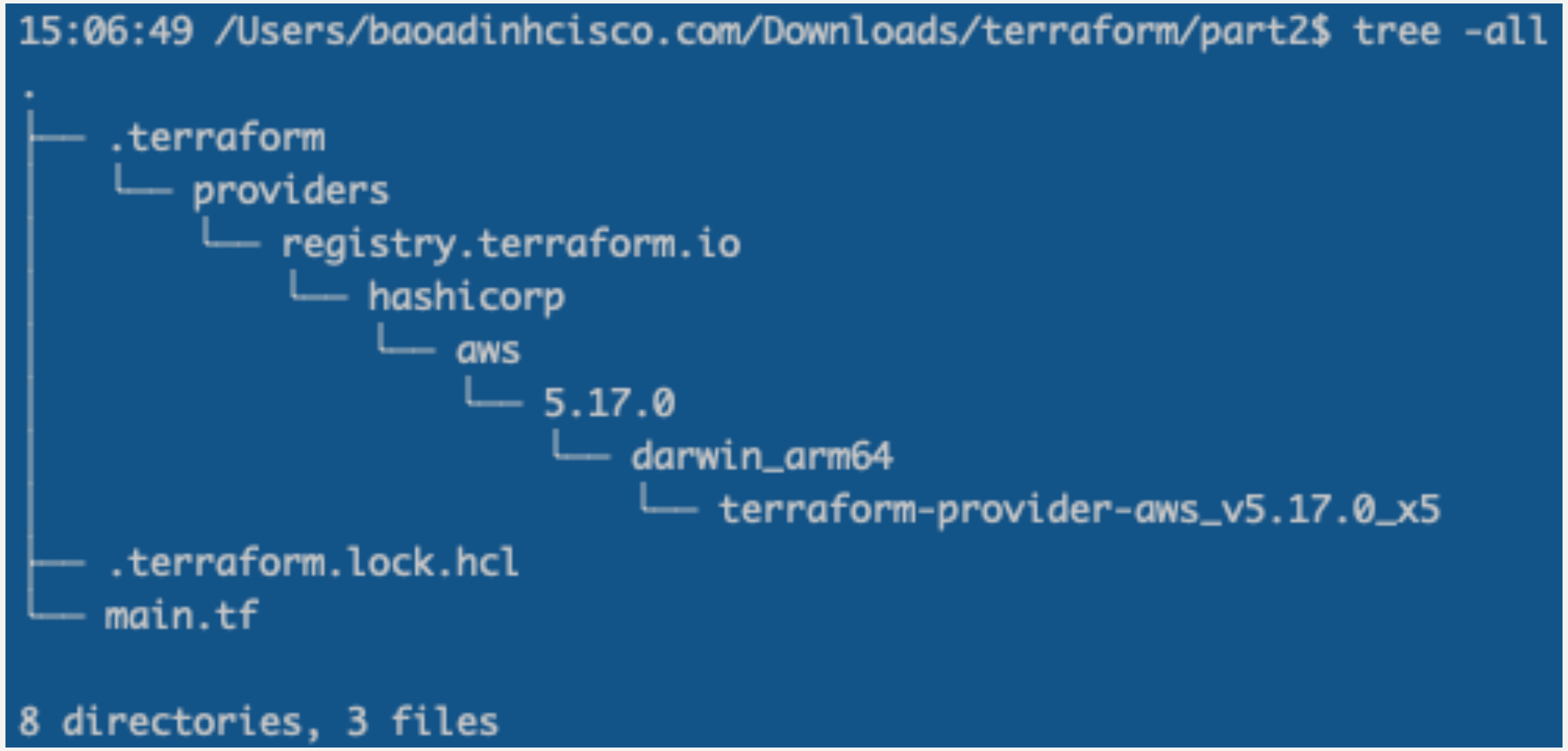
```
provider "aws" {
  region = "us-west-2"
}

resource "aws_s3_bucket" "terraform-bucket" {
  bucket = "terraform-series-bucket-bao"

  tags = {
    Name        = "Terraform Series"
  }
}
```

Initializing the workspace



```
14:57:12 /Users/baoadinhcisco.com/Downloads/terraform/part2$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.17.0...
- Installed hashicorp/aws v5.17.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

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.
14:57:35 /Users/baoadinhcisco.com/Downloads/terraform/part2$
```

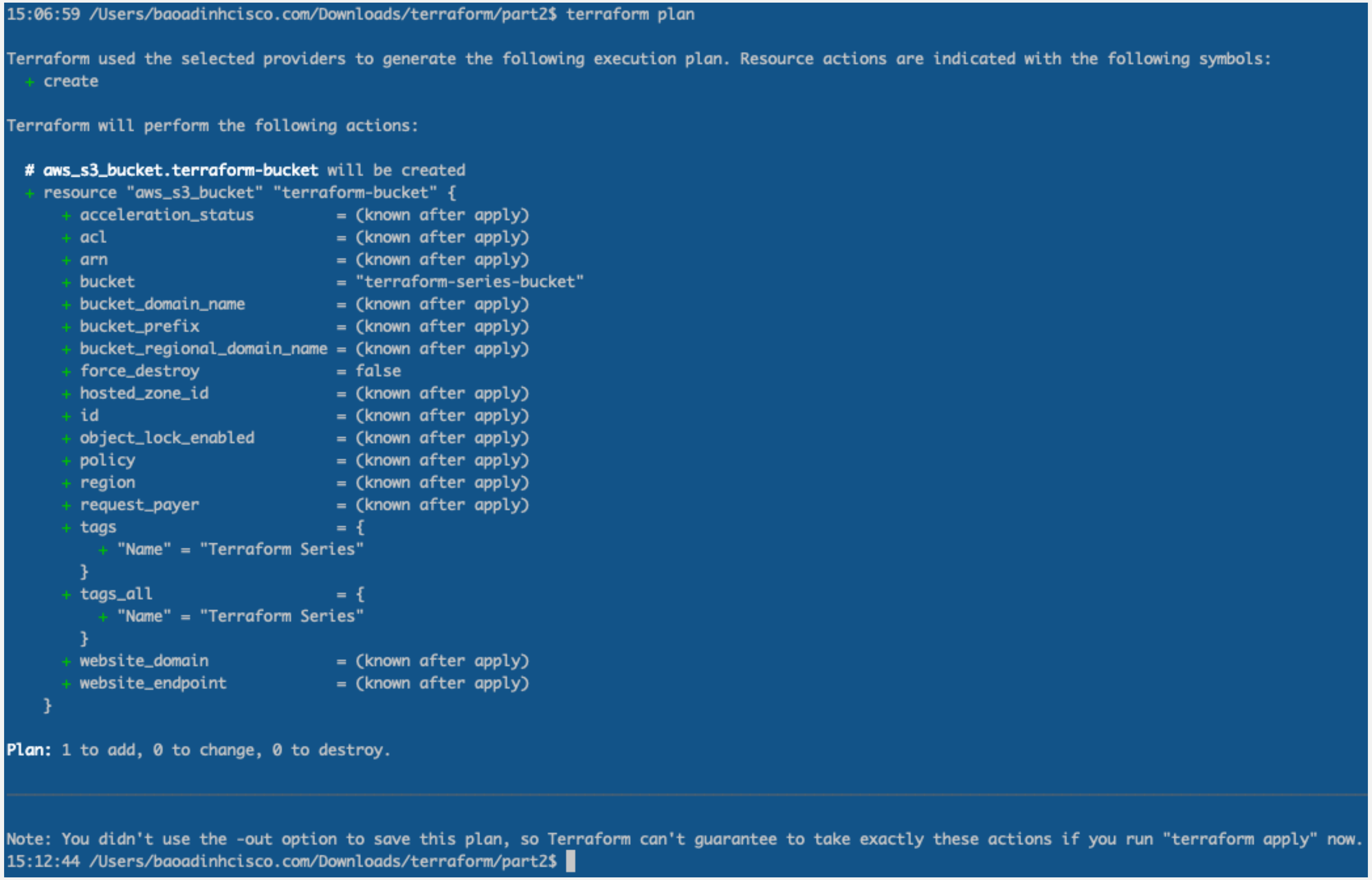
After initialization, Terraform creates a hidden `.terraform` directory for installing plugins and modules. The directory structure for the current Terraform workspace is the following:



```
15:06:49 /Users/baoadinhcisco.com/Downloads/terraform/part2$ tree -all
.
├── .terraform
│   └── providers
│       └── registry.terraform.io
│           └── hashicorp
│               └── aws
│                   └── 5.17.0
│                       └── darwin_arm64
│                           └── terraform-provider-aws_v5.17.0_x5
├── .terraform.lock.hcl
└── main.tf

8 directories, 3 files
```

Read() operation

Check which resources will be created

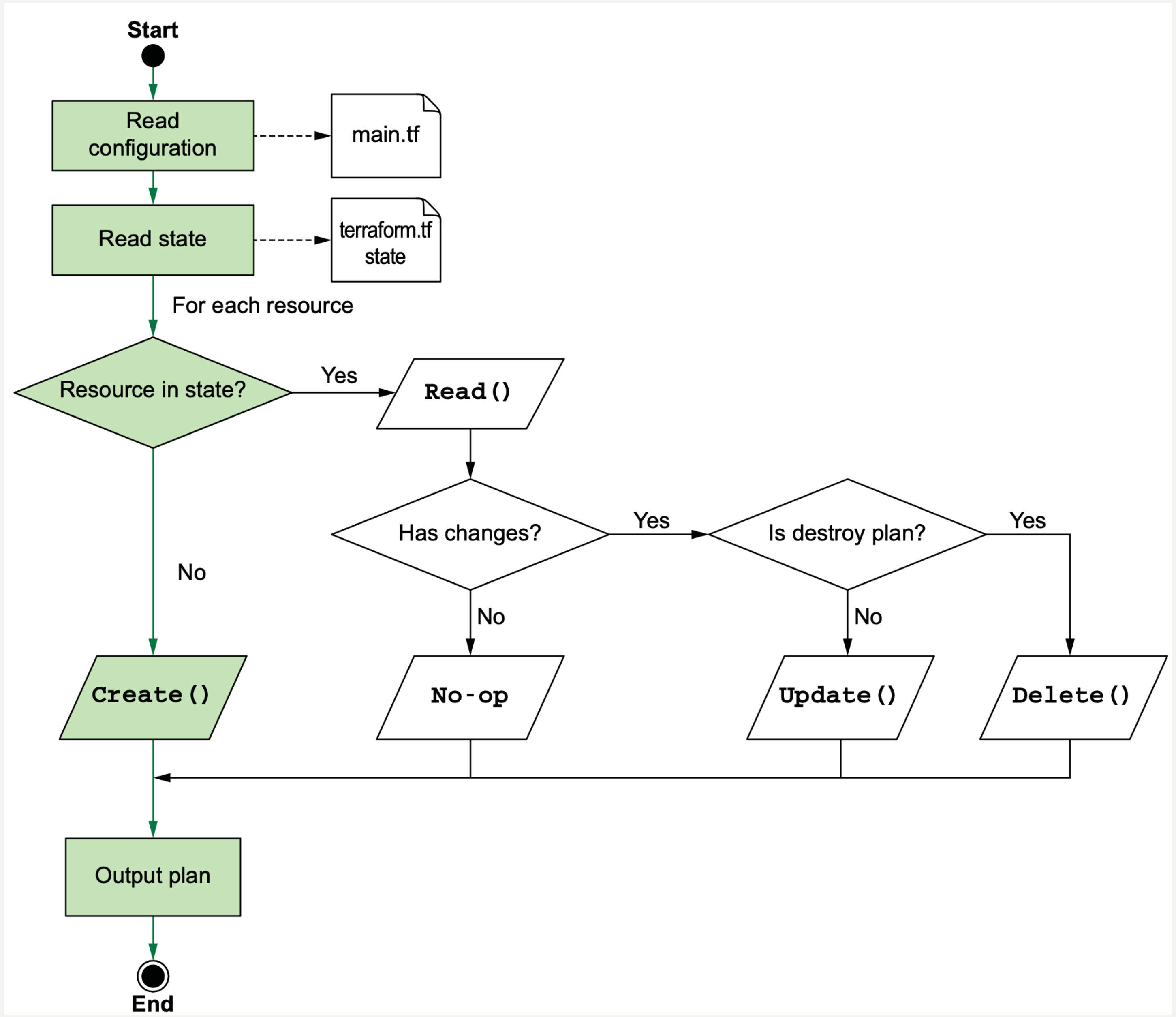


```
15:06:59 /Users/baoadinhcisco.com/Downloads/terraform/part2$ terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_s3_bucket.terraform-bucket will be created
  + resource "aws_s3_bucket" "terraform-bucket" {
      + acceleration_status         = (known after apply)
      + acl                         = (known after apply)
      + arn                         = (known after apply)
      + bucket                      = "terraform-series-bucket"
      + bucket_domain_name          = (known after apply)
      + bucket_prefix               = (known after apply)
      + bucket_regional_domain_name = (known after apply)
      + force_destroy               = false
      + hosted_zone_id              = (known after apply)
      + id                          = (known after apply)
      + object_lock_enabled         = (known after apply)
      + policy                      = (known after apply)
      + region                      = (known after apply)
      + request_payer               = (known after apply)
      + tags                        = {
          + "Name" = "Terraform Series"
        }
      + tags_all                    = {
          + "Name" = "Terraform Series"
        }
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)
    }

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

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
15:12:44 /Users/baoadinhcisco.com/Downloads/terraform/part2$
```

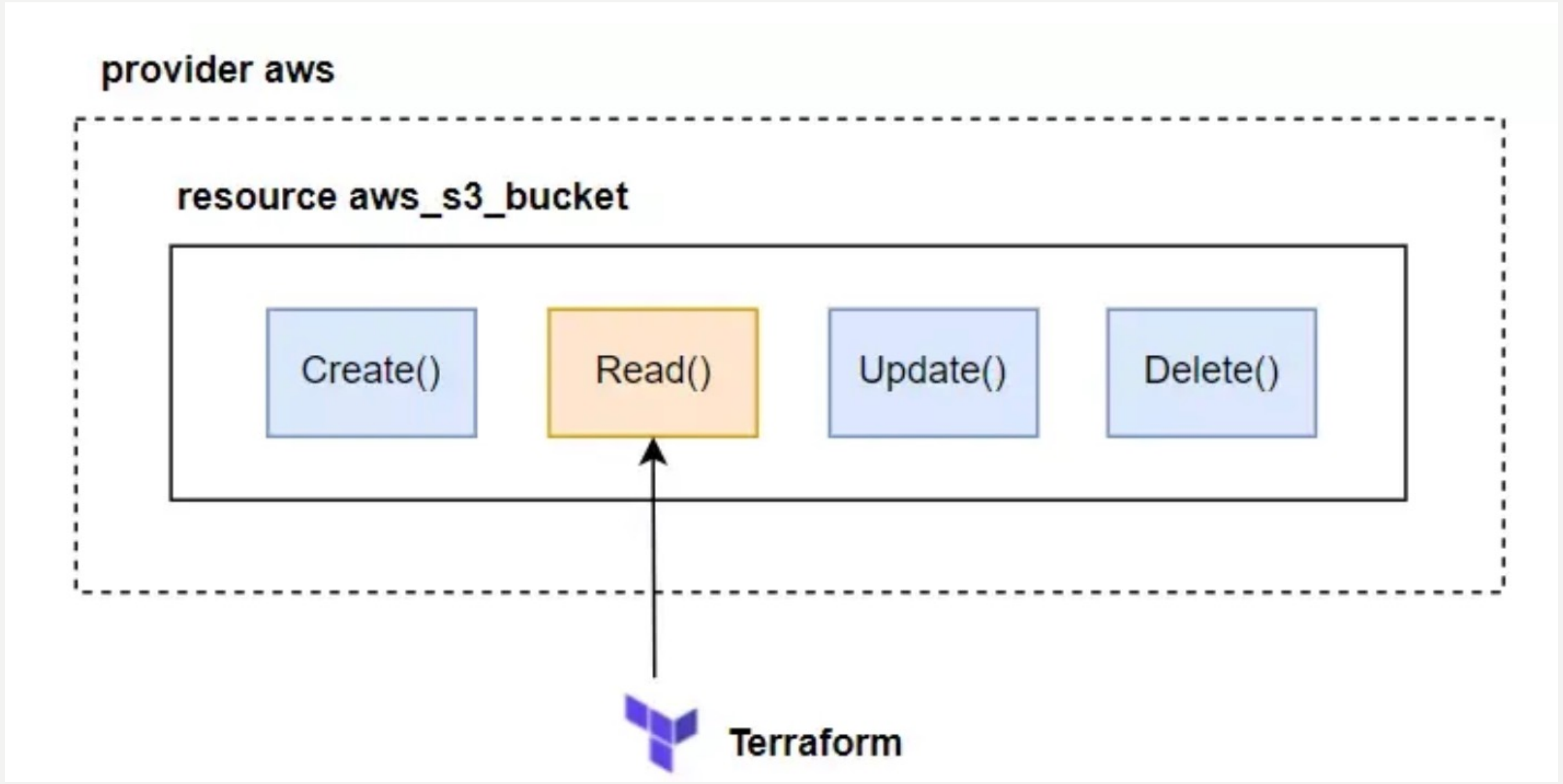
If you need to save the results of the plan statement, you use the `–out` attribute at runtime. For example, we will save the results of the plan statement in a JSON file.

```
terraform plan -out plan.out

terraform show -json plan.out > plan.json
```

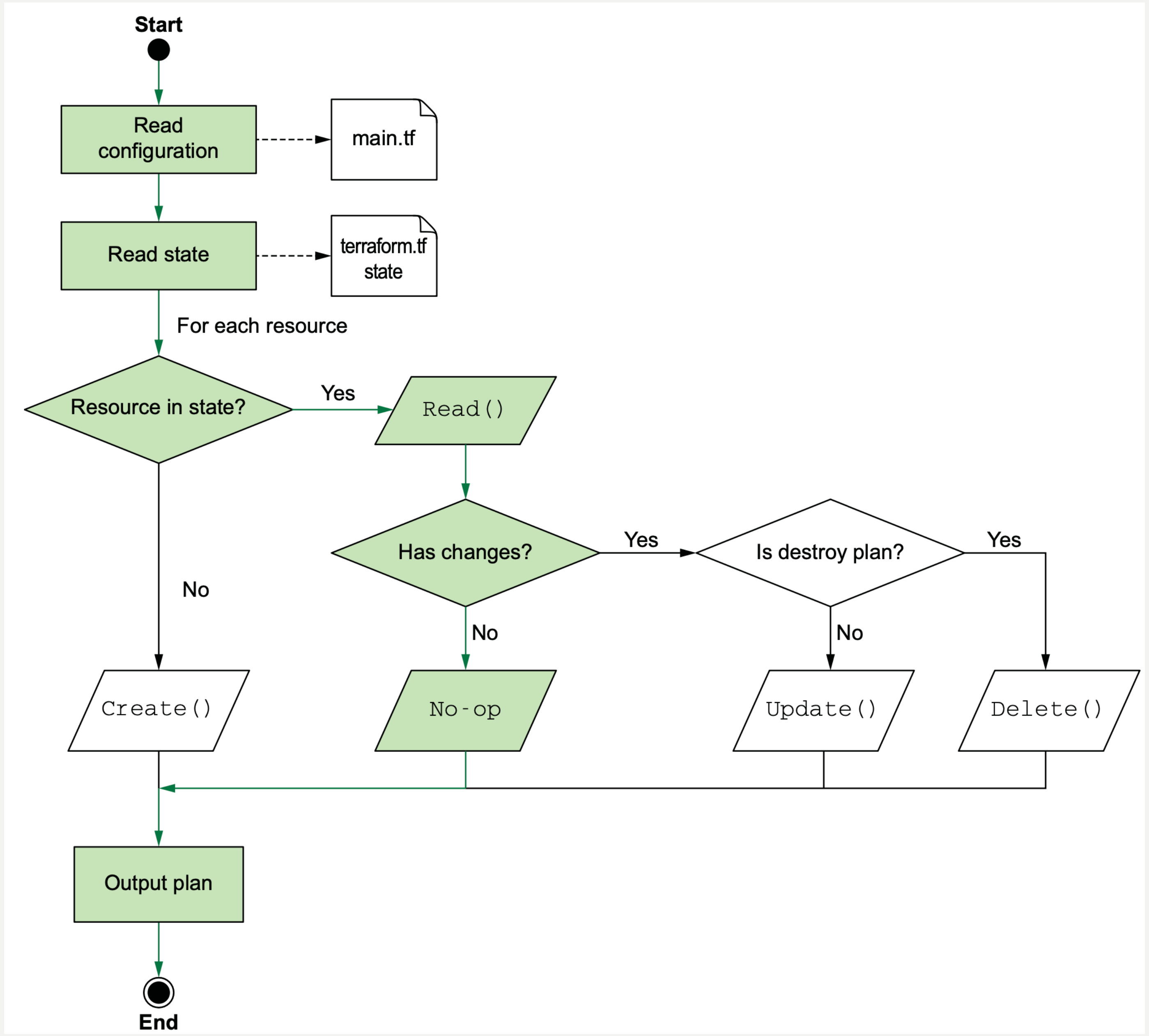
Illustrative diagram of the plan process



When `terraform plan` is run, Terraform call `Read()` on each resource in the state file.



If there are no changes when running `terraform plan`, the resulting action is a no-operation (no-op).



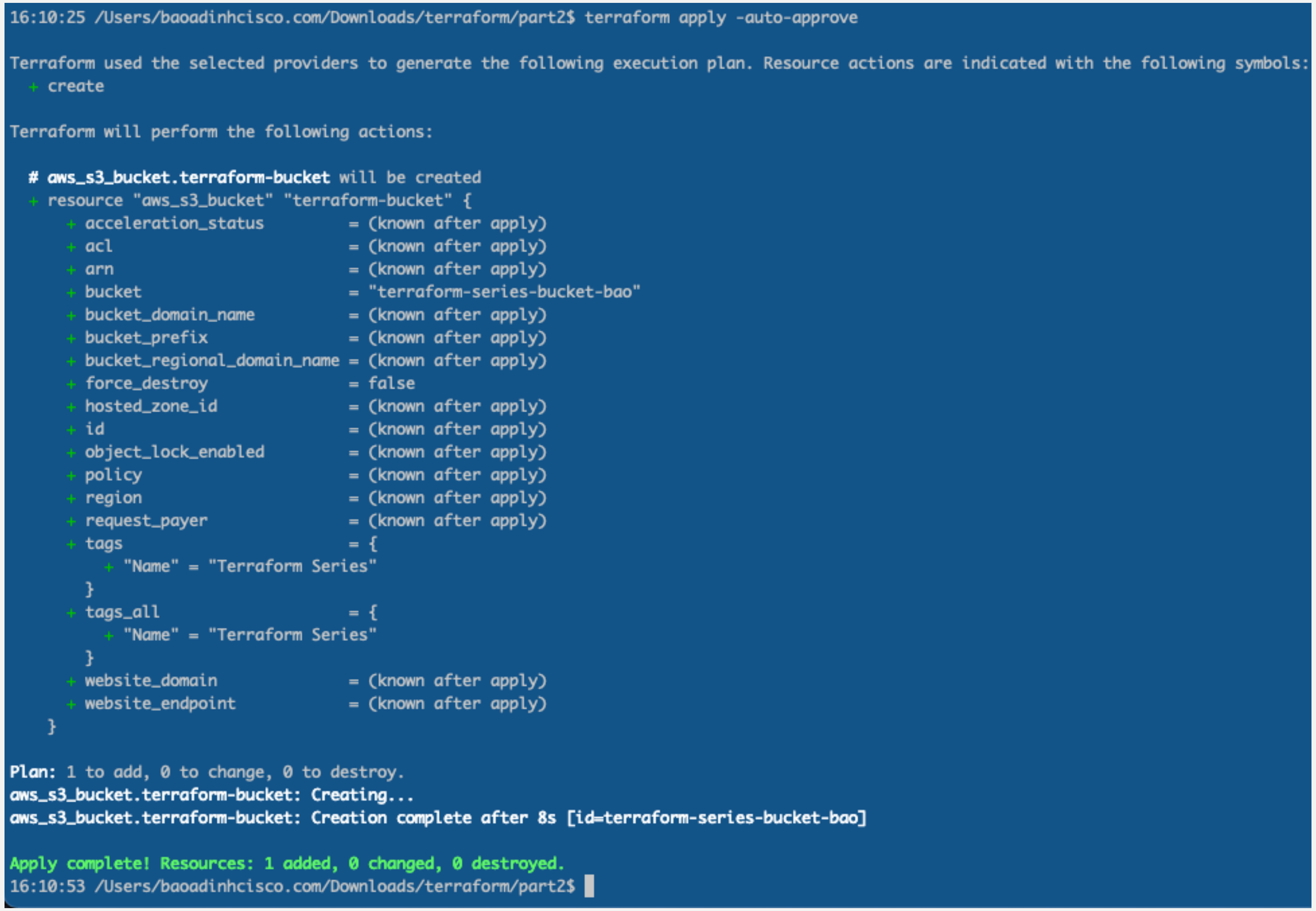
Create() operation

INITIALIZING THE WORKSPACE

Now we will run the apply command to create S3 on AWS. When we run the apply command,

it will have an additional confirmation step and require us to enter yes.

If you want to skip the confirmation step, then when running, add the attribute -auto-approve.



```
16:10:25 /Users/baoadinhcisco.com/Downloads/terraform/part2$ terraform apply -auto-approve

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  # aws_s3_bucket.terraform-bucket will be created
  + resource "aws_s3_bucket" "terraform-bucket" {
      + acceleration_status         = (known after apply)
      + acl                         = (known after apply)
      + arn                         = (known after apply)
      + bucket                      = "terraform-series-bucket-bao"
      + bucket_domain_name          = (known after apply)
      + bucket_prefix               = (known after apply)
      + bucket_regional_domain_name = (known after apply)
      + force_destroy               = false
      + hosted_zone_id              = (known after apply)
      + id                          = (known after apply)
      + object_lock_enabled         = (known after apply)
      + policy                      = (known after apply)
      + region                      = (known after apply)
      + request_payer               = (known after apply)
      + tags                        = {
          + "Name" = "Terraform Series"
        }
      + tags_all                    = {
          + "Name" = "Terraform Series"
        }
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
aws_s3_bucket.terraform-bucket: Creating...
aws_s3_bucket.terraform-bucket: Creation complete after 8s [id=terraform-series-bucket-bao]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
16:10:53 /Users/baoadinhcisco.com/Downloads/terraform/part2$
```

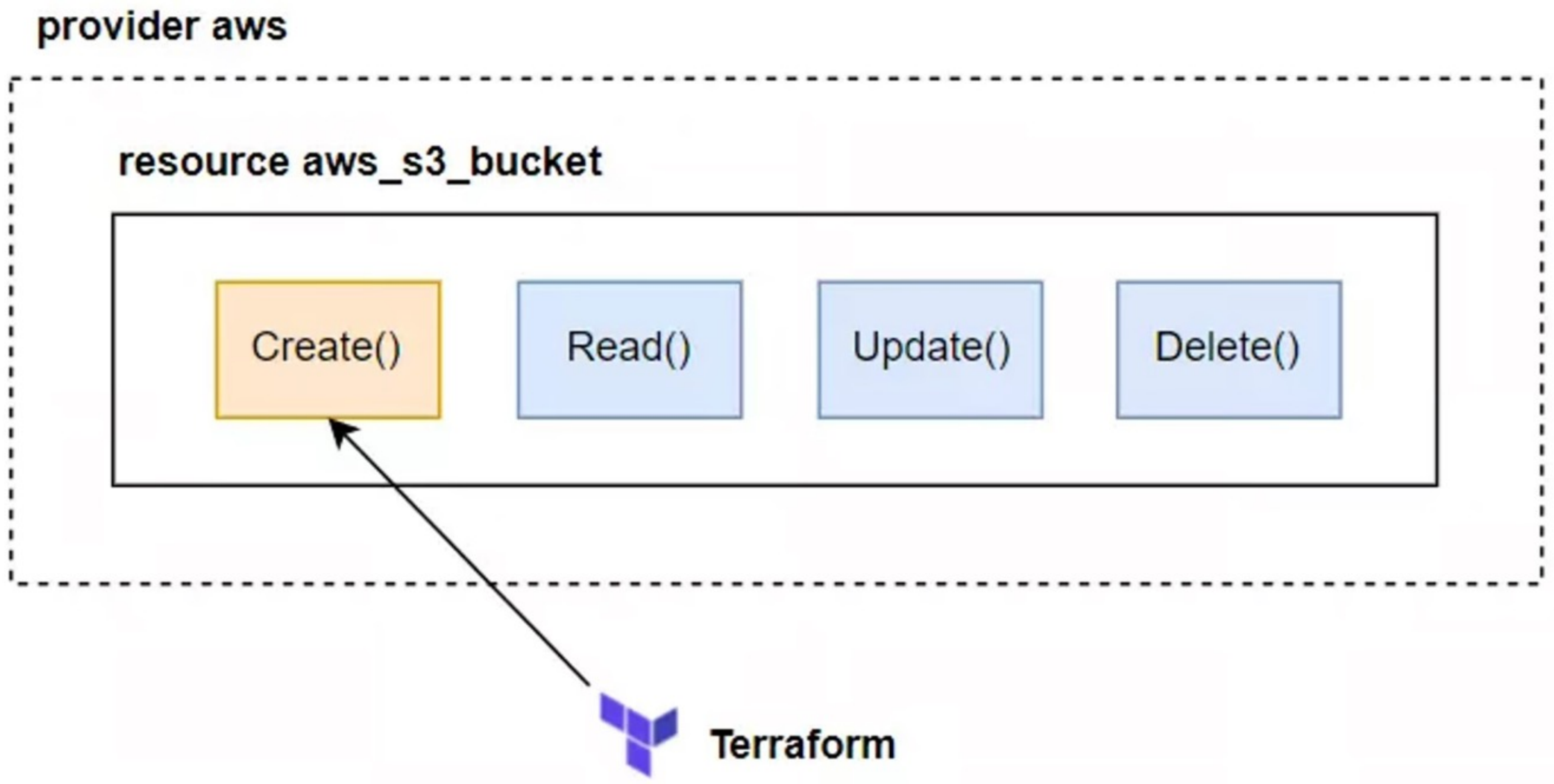
When finished running, you will see a new file created `terraform.tfstate`

The terraform.tfstate file you see here is the state file that Terraform uses
to keep track of the resources it manages. It's used to perform diffs during the `plan`
and detect configuration drift. Here's what the current state file looks like.

<Open and check the new file in terminal>

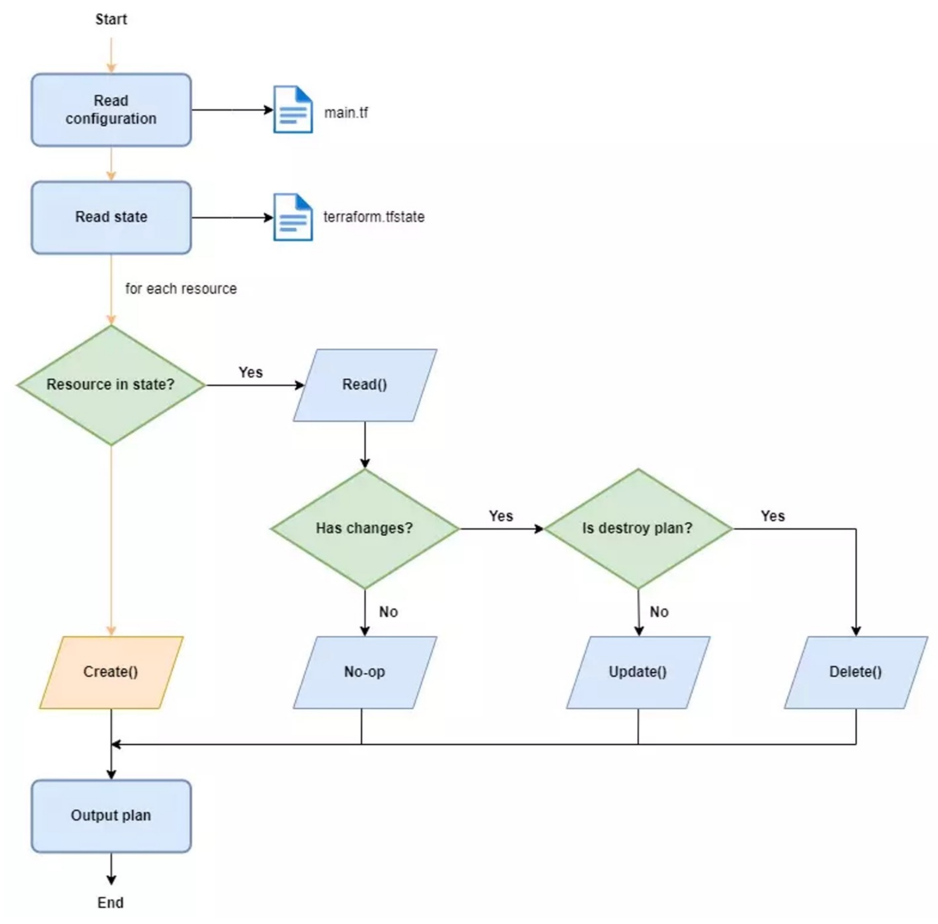
<Open AWS and check that S3 is created after running `apply`>

During the apply process, Terraform will call the `Create()` function of the `aws_s3_bucket` resource.



The `Create()` function contains code to call the API to AWS to create an S3 Bucket.

When Terraform calls this function, the S3 Bucket will be created, illustrated as this figure.

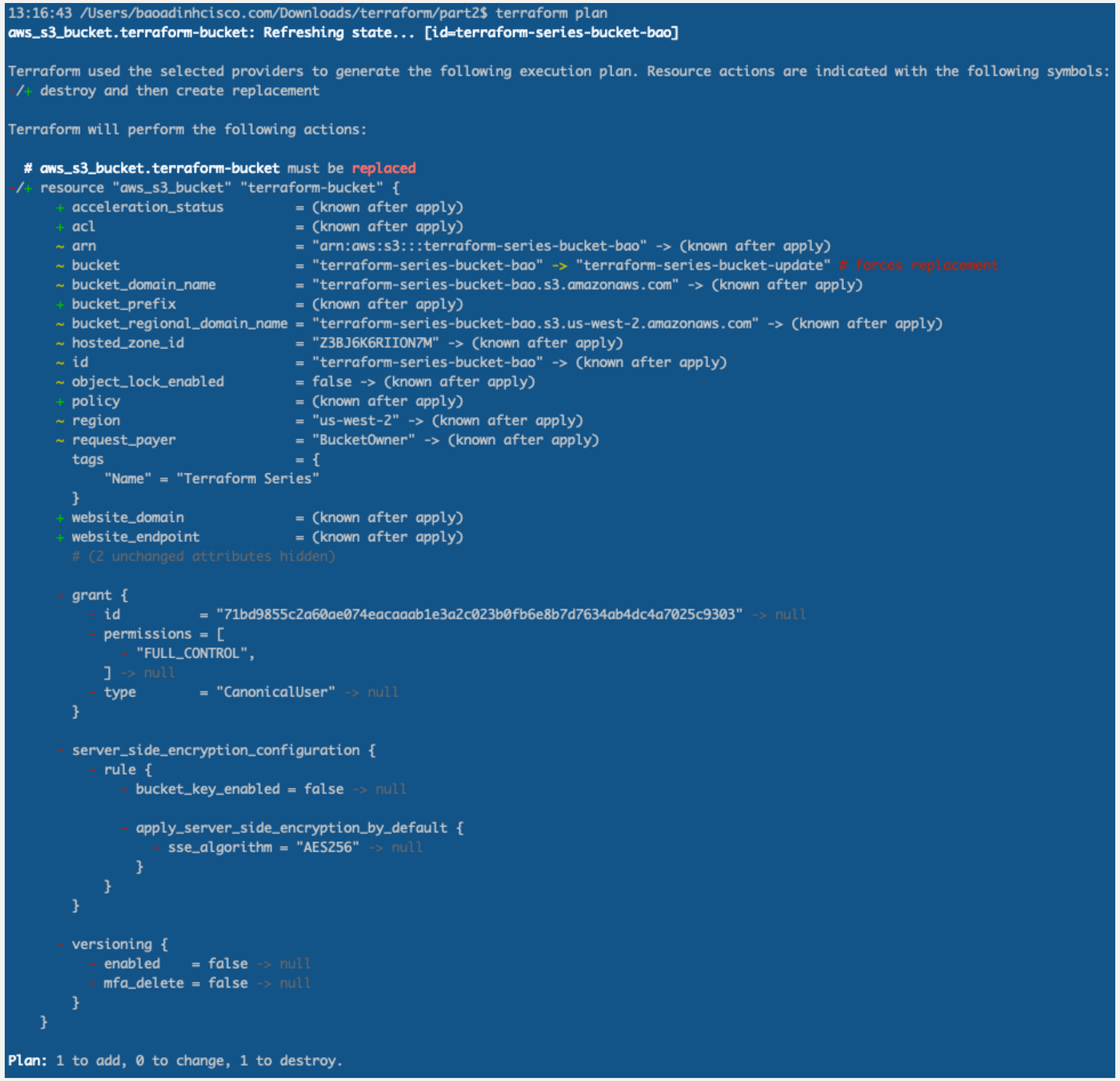


Update() operation

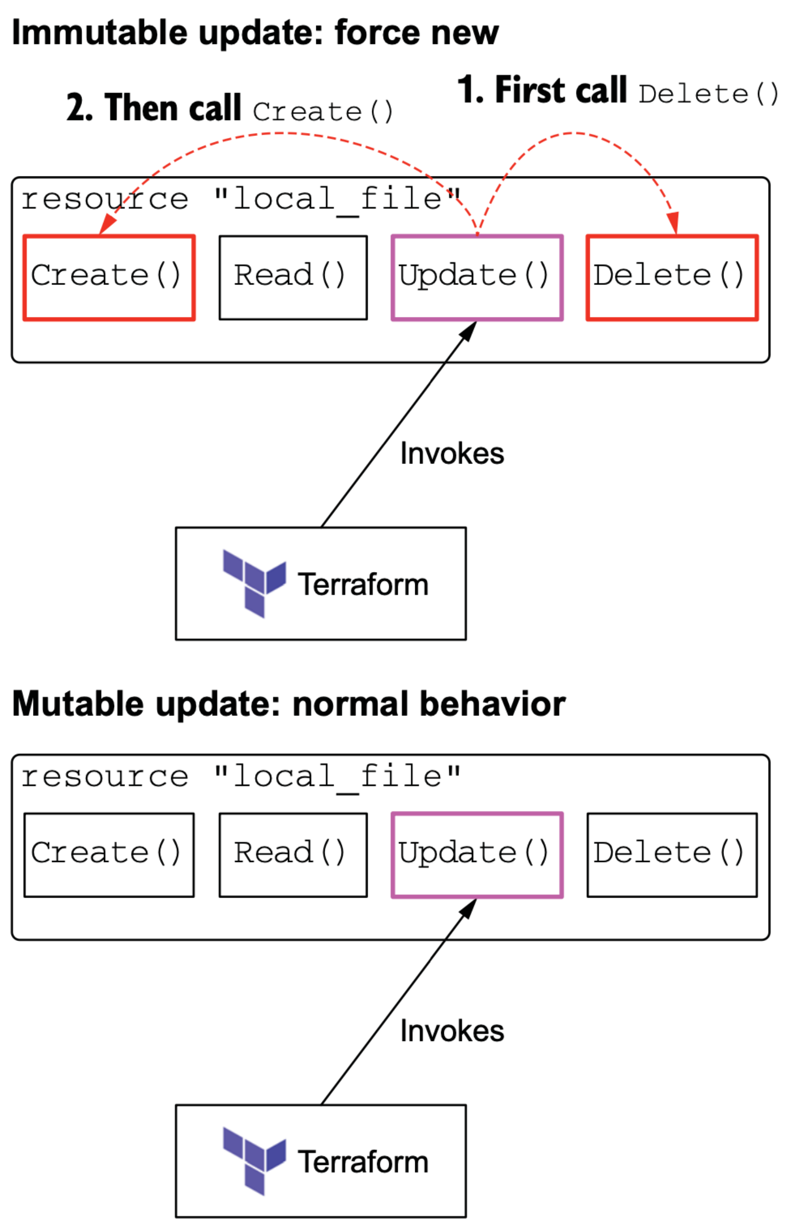
In Terraform there is no `update` command, we just need to edit the configuration file and run the `apply` command again, Terraform will automatically determine whether to update the resource or not. We changed the name of the S3 Bucket.

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_s3_bucket" "terraform-bucket" {
  bucket = "terraform-series-bucket-rabbitmq"

  tags = {
    Name = "Terraform Series"
  }
}
```

Reviewing the changes by running `terraform plan`



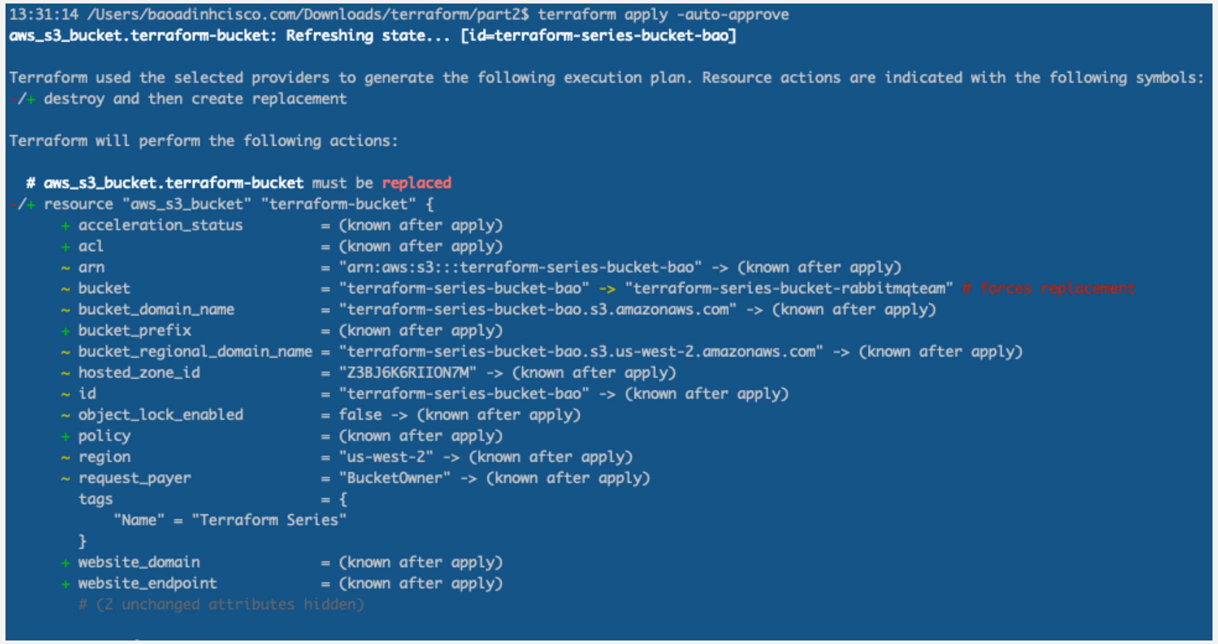
```
13:16:43 /Users/baoadinhcisco.com/Downloads/terraform/part2$ terraform plan
aws_s3_bucket.terraform-bucket: Refreshing state... [id=terraform-series-bucket-bao]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # aws_s3_bucket.terraform-bucket must be replaced
-/+ resource "aws_s3_bucket" "terraform-bucket" {
      + acceleration_status         = (known after apply)
      + acl                         = (known after apply)
      ~ arn                         = "arn:aws:s3:::terraform-series-bucket-bao" -> (known after apply)
      ~ bucket                      = "terraform-series-bucket-bao" -> "terraform-series-bucket-update" # forces replacement
      ~ bucket_domain_name          = "terraform-series-bucket-bao.s3.amazonaws.com" -> (known after apply)
      + bucket_prefix               = (known after apply)
      ~ bucket_regional_domain_name = "terraform-series-bucket-bao.s3.us-west-2.amazonaws.com" -> (known after apply)
      ~ hosted_zone_id              = "Z3BJ6K6RIION7M" -> (known after apply)
      ~ id                          = "terraform-series-bucket-bao" -> (known after apply)
      ~ object_lock_enabled         = false -> (known after apply)
      + policy                      = (known after apply)
      ~ region                      = "us-west-2" -> (known after apply)
      ~ request_payer               = "BucketOwner" -> (known after apply)
        tags                        = {
            "Name" = "Terraform Series"
        }
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)
        # (2 unchanged attributes hidden)

      - grant {
          - id          = "71bd9855c2a60ae074eacaaab1e3a2c023b0fb6e8b7d7634ab4dc4a7025c9303" -> null
          - permissions = [
              - "FULL_CONTROL",
            ] -> null
          - type        = "CanonicalUser" -> null
        }

      - server_side_encryption_configuration {
          - rule {
              - bucket_key_enabled = false -> null

              - apply_server_side_encryption_by_default {
                  - sse_algorithm = "AES256" -> null
                }
            }
        }

      - versioning {
          - enabled    = false -> null
          - mfa_delete = false -> null
        }
    }

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



Difference between immutable and mutable updates

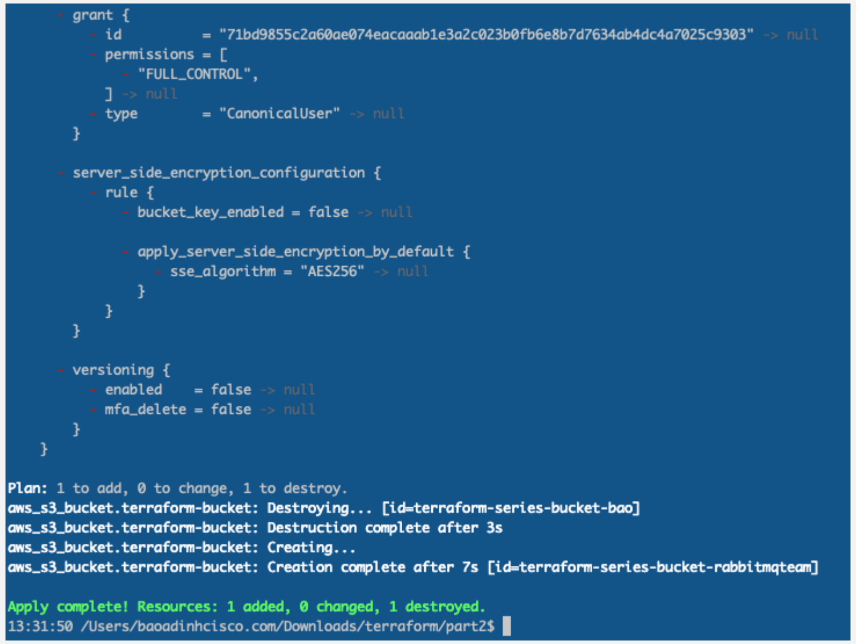
Because our S3 Bucket was just created and there is nothing in it, we just run `terraform apply` so it updates normally.

1



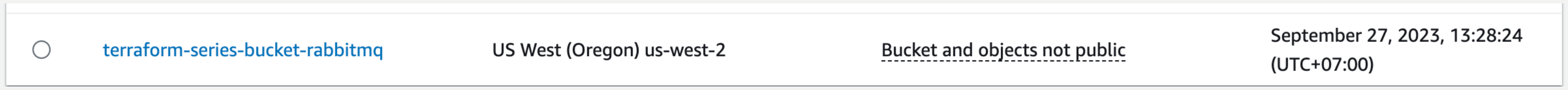
```
13:31:14 /Users/baoadinhcisco.com/Downloads/terraform/part2$ terraform apply -auto-approve
aws_s3_bucket.terraform-bucket: Refreshing state... [id=terraform-series-bucket-bao]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # aws_s3_bucket.terraform-bucket must be replaced
-/+ resource "aws_s3_bucket" "terraform-bucket" {
      + acceleration_status         = (known after apply)
      + acl                         = (known after apply)
      ~ arn                         = "arn:aws:s3:::terraform-series-bucket-bao" -> (known after apply)
      ~ bucket                      = "terraform-series-bucket-bao" -> "terraform-series-bucket-rabbitmqteam" # forces replacement
      ~ bucket_domain_name          = "terraform-series-bucket-bao.s3.amazonaws.com" -> (known after apply)
      + bucket_prefix               = (known after apply)
      ~ bucket_regional_domain_name = "terraform-series-bucket-bao.s3.us-west-2.amazonaws.com" -> (known after apply)
      ~ hosted_zone_id              = "Z3BJ6K6RIION7M" -> (known after apply)
      ~ id                          = "terraform-series-bucket-bao" -> (known after apply)
      ~ object_lock_enabled         = false -> (known after apply)
      + policy                      = (known after apply)
      ~ region                      = "us-west-2" -> (known after apply)
      ~ request_payer               = "BucketOwner" -> (known after apply)
        tags                        = {
            "Name" = "Terraform Series"
        }
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)
        # (2 unchanged attributes hidden)
```

2



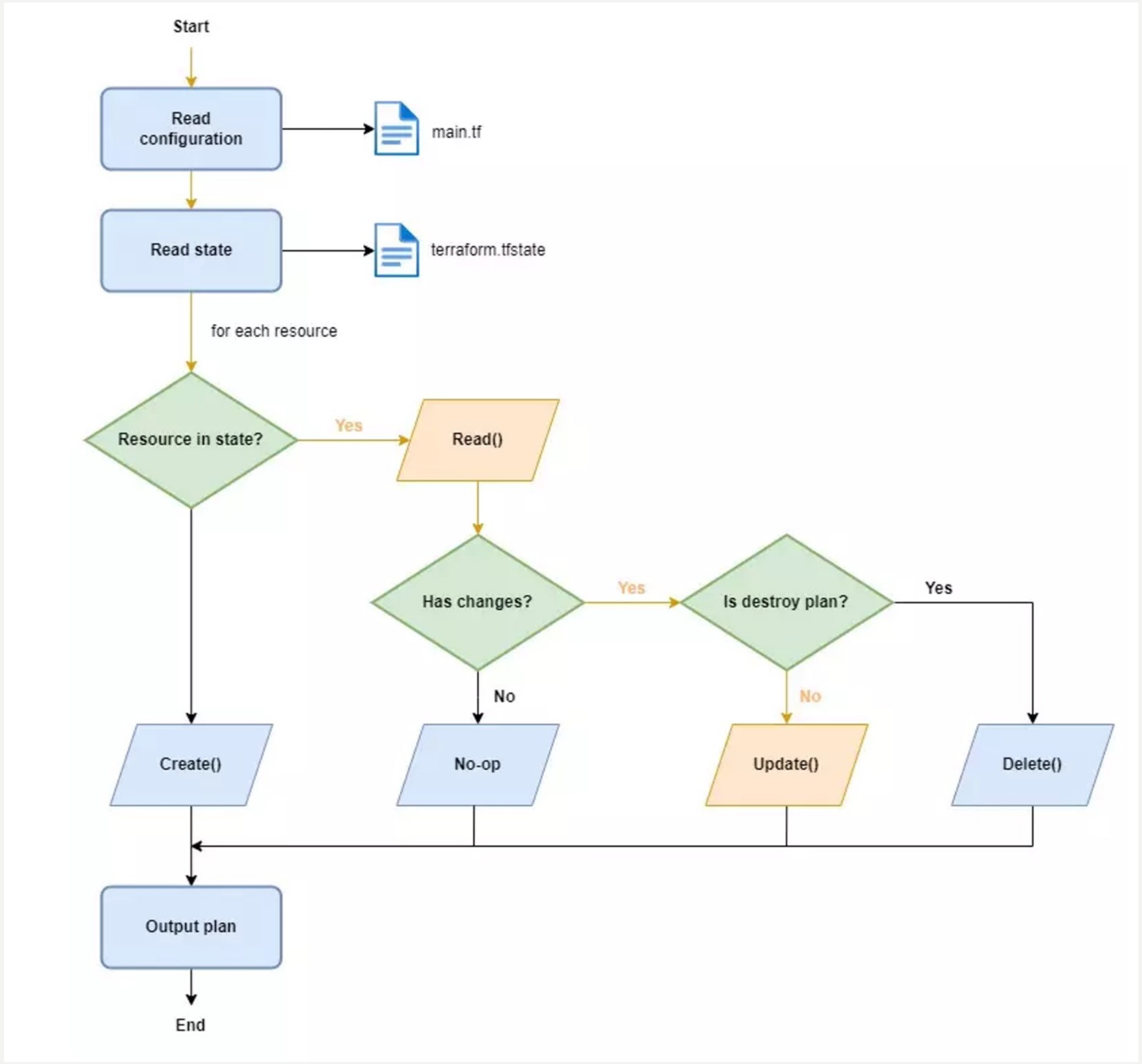
```
      - grant {
          - id          = "71bd9855c2a60ae074eacaaab1e3a2c023b0fb6e8b7d7634ab4dc4a7025c9303" -> null
          - permissions = [
              - "FULL_CONTROL",
            ] -> null
          - type        = "CanonicalUser" -> null
        }

      - server_side_encryption_configuration {
          - rule {
              - bucket_key_enabled = false -> null

              - apply_server_side_encryption_by_default {
                  - sse_algorithm = "AES256" -> null
                }
            }
        }

      - versioning {
          - enabled    = false -> null
          - mfa_delete = false -> null
        }
    }

Plan: 1 to add, 0 to change, 1 to destroy.
aws_s3_bucket.terraform-bucket: Destroying... [id=terraform-series-bucket-bao]
aws_s3_bucket.terraform-bucket: Destruction complete after 3s
aws_s3_bucket.terraform-bucket: Creating...
aws_s3_bucket.terraform-bucket: Creation complete after 7s [id=terraform-series-bucket-rabbitmqteam]

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
13:31:50 /Users/baoadinhcisco.com/Downloads/terraform/part2$
```

S3 Bucket with new name updated:



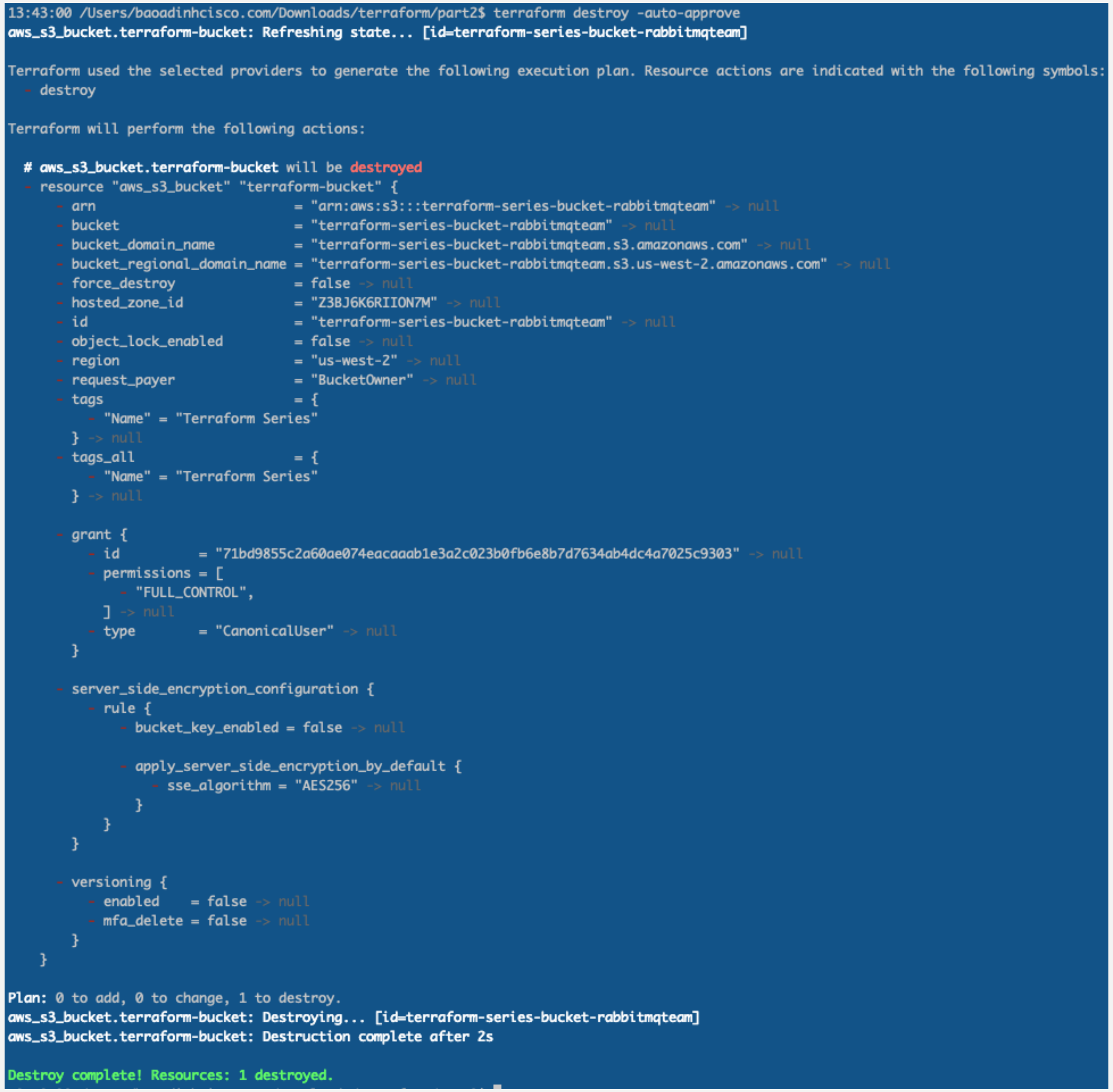
○	terraform-series-bucket-rabbitmq	US West (Oregon) us-west-2	Bucket and objects not public	September 27, 2023, 13:28:24 (UTC+07:00)

Illustrative diagram of the update process



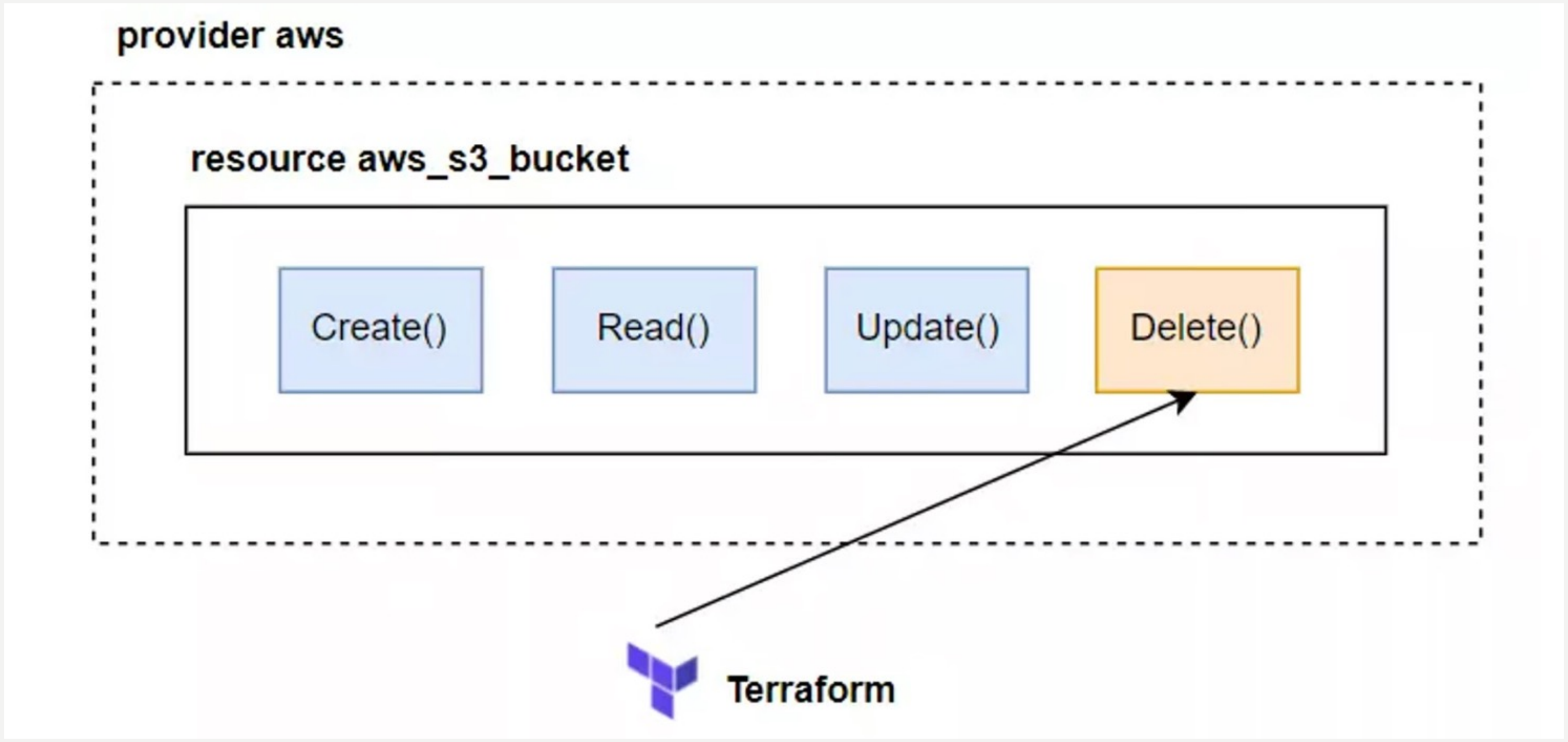
Delete() operation

Let’s clean up by running `terraform destroy`

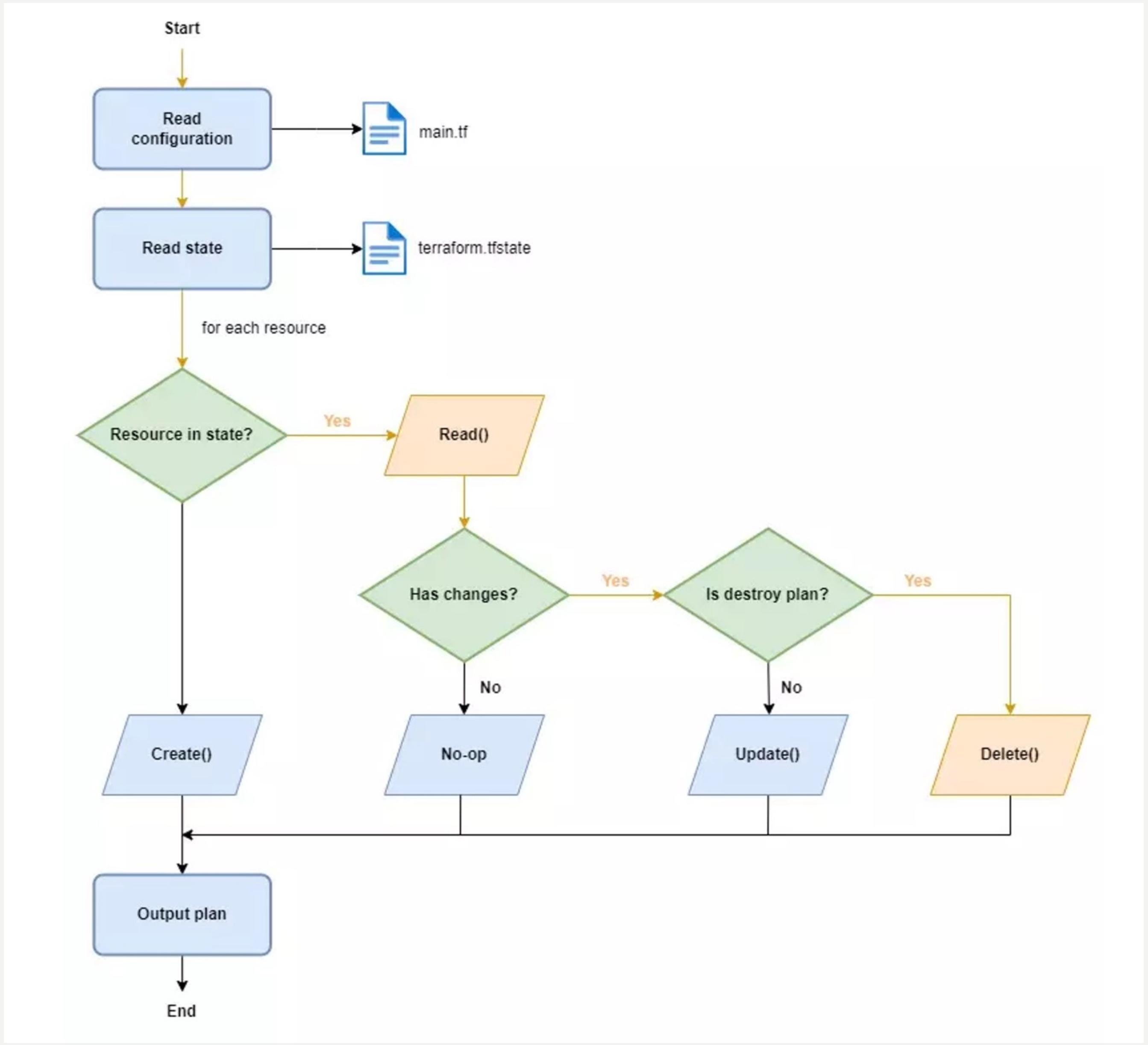


```
13:43:00 /Users/baoadinhcisco.com/Downloads/terraform/part2$ terraform destroy -auto-approve
aws_s3_bucket.terraform-bucket: Refreshing state... [id=terraform-series-bucket-rabbitmqteam]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  - destroy

Terraform will perform the following actions:

  # aws_s3_bucket.terraform-bucket will be destroyed
  - resource "aws_s3_bucket" "terraform-bucket" {
      - arn                         = "arn:aws:s3:::terraform-series-bucket-rabbitmqteam" -> null
      - bucket                      = "terraform-series-bucket-rabbitmqteam" -> null
      - bucket_domain_name          = "terraform-series-bucket-rabbitmqteam.s3.amazonaws.com" -> null
      - bucket_regional_domain_name = "terraform-series-bucket-rabbitmqteam.s3.us-west-2.amazonaws.com" -> null
      - force_destroy               = false -> null
      - hosted_zone_id              = "Z3BJ6K6RIION7M" -> null
      - id                          = "terraform-series-bucket-rabbitmqteam" -> null
      - object_lock_enabled         = false -> null
      - region                      = "us-west-2" -> null
      - request_payer               = "BucketOwner" -> null
      - tags                        = {
          - "Name" = "Terraform Series"
        } -> null
      - tags_all                    = {
          - "Name" = "Terraform Series"
        } -> null

      - grant {
          - id          = "71bd9855c2a60ae074eacaaab1e3a2c023b0fb6e8b7d7634ab4dc4a7025c9303" -> null
          - permissions = [
              - "FULL_CONTROL",
            ] -> null
          - type        = "CanonicalUser" -> null
        }

      - server_side_encryption_configuration {
          - rule {
              - bucket_key_enabled = false -> null

              - apply_server_side_encryption_by_default {
                  - sse_algorithm = "AES256" -> null
                }
            }
        }

      - versioning {
          - enabled    = false -> null
          - mfa_delete = false -> null
        }
    }

Plan: 0 to add, 0 to change, 1 to destroy.
aws_s3_bucket.terraform-bucket: Destroying... [id=terraform-series-bucket-rabbitmqteam]
aws_s3_bucket.terraform-bucket: Destruction complete after 2s

Destroy complete! Resources: 1 destroyed.
```

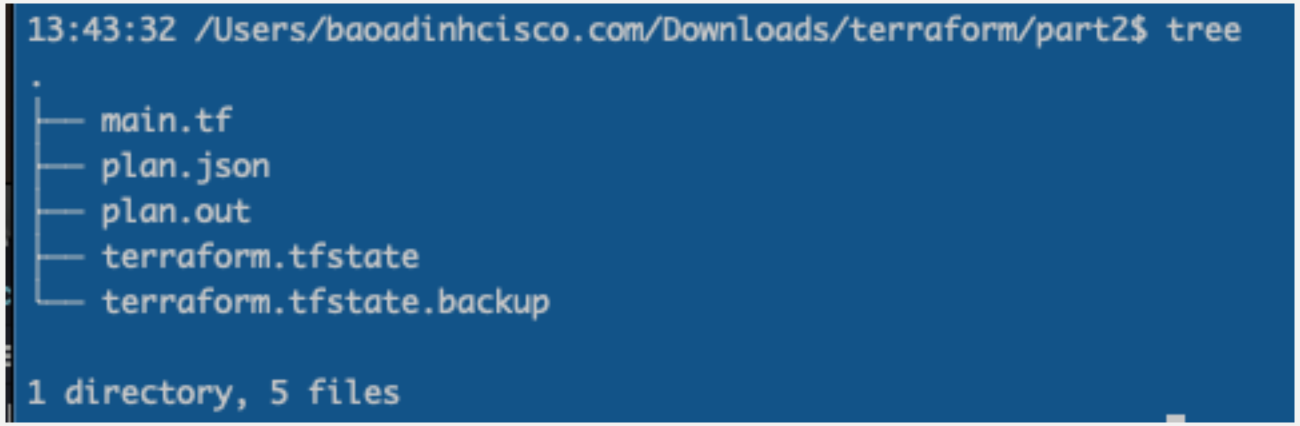
When we run the destroy statement, it will read in our State file to see if there is that resource. If so, it will execute the `Delete()` function of the `aws_s3_bucket` resource.



Illustrative diagram of the delete process

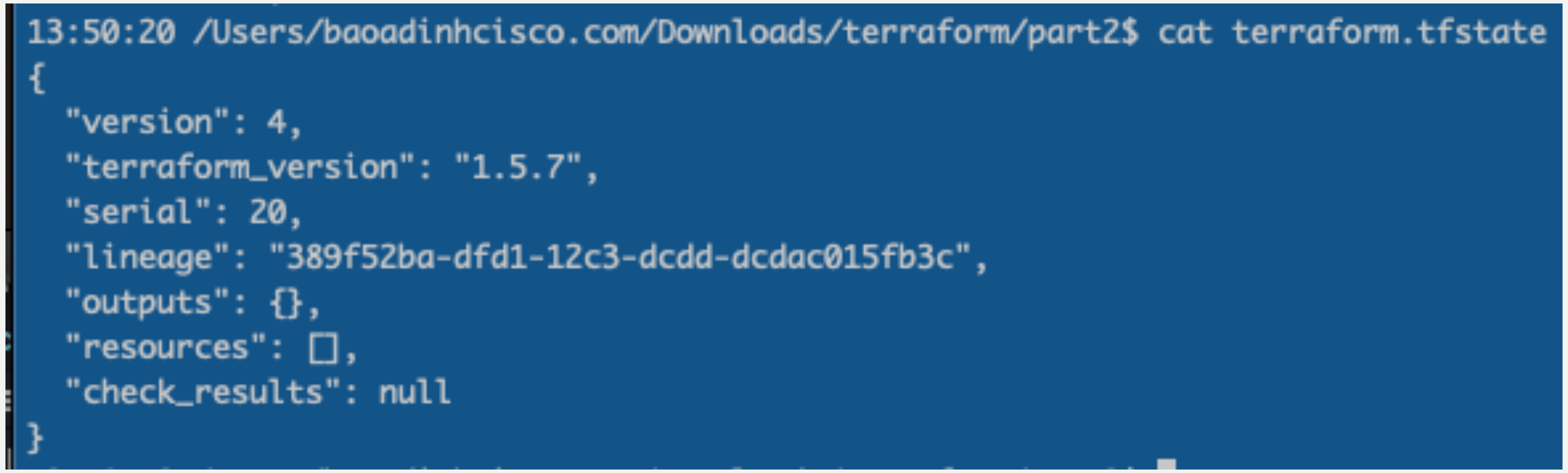


Although it's gone, its memory lives on in a new file, `terraform.tfstate.backup`



```
13:43:32 /Users/baoadinhcisco.com/Downloads/terraform/part2$ tree
.
├── main.tf
├── plan.json
├── plan.out
├── terraform.tfstate
└── terraform.tfstate.backup

1 directory, 5 files
```

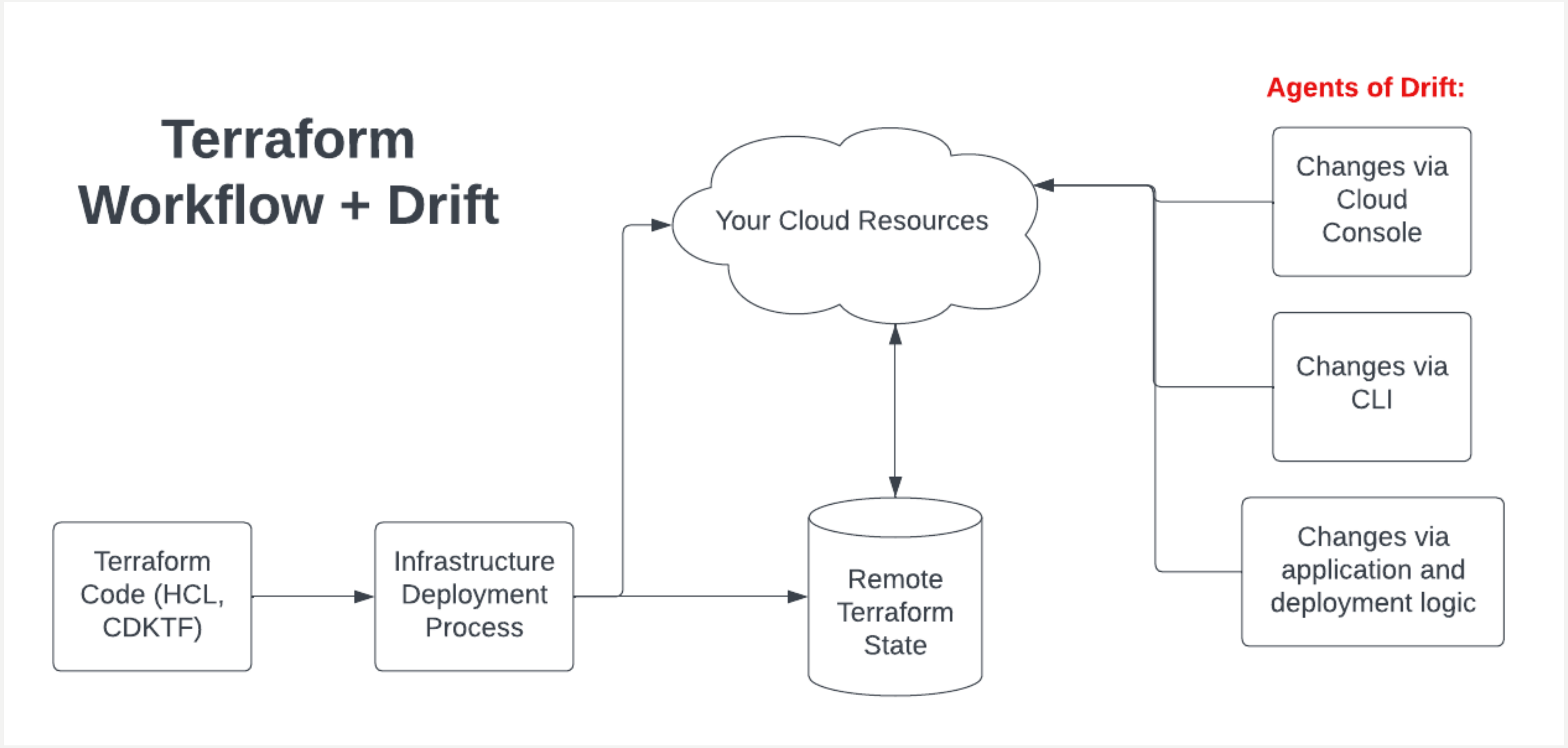
And our current state file is empty



```
13:50:20 /Users/baoadinhcisco.com/Downloads/terraform/part2$ cat terraform.tfstate
{
  "version": 4,
  "terraform_version": "1.5.7",
  "serial": 20,
  "lineage": "389f52ba-dfd1-12c3-dcdd-dcdac015fb3c",
  "outputs": {},
  "resources": [],
  "check_results": null
}
```

RESOURCE DRIFT

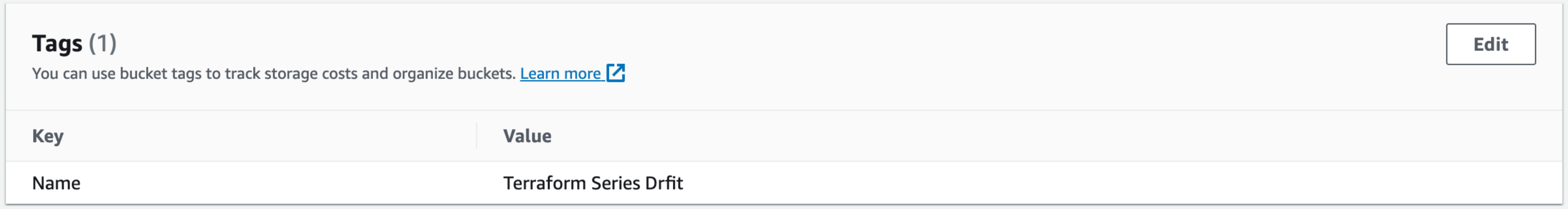
Any change – not triggered by Terraform – that occurs to the infrastructure being managed by Terraform is known as a **drift**


Terraform Workflow + Drift
Agents of Drift:
Your Cloud Resources
Changes via Cloud Console
Changes via CLI
Changes via application and deployment logic
Terraform Code (HCL, CDKTF)
Infrastructure Deployment Process
Remote Terraform State

We will re-use the main.tf file above, apply it to create the S3 on AWS

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_s3_bucket" "terraform-bucket" {
  bucket = "terraform-series-bucket-rabbitmq"

  tags = {
    Name = "Terraform Series"
  }
}
```

Then, we come to AWS Web Console and edit the `Tags` field

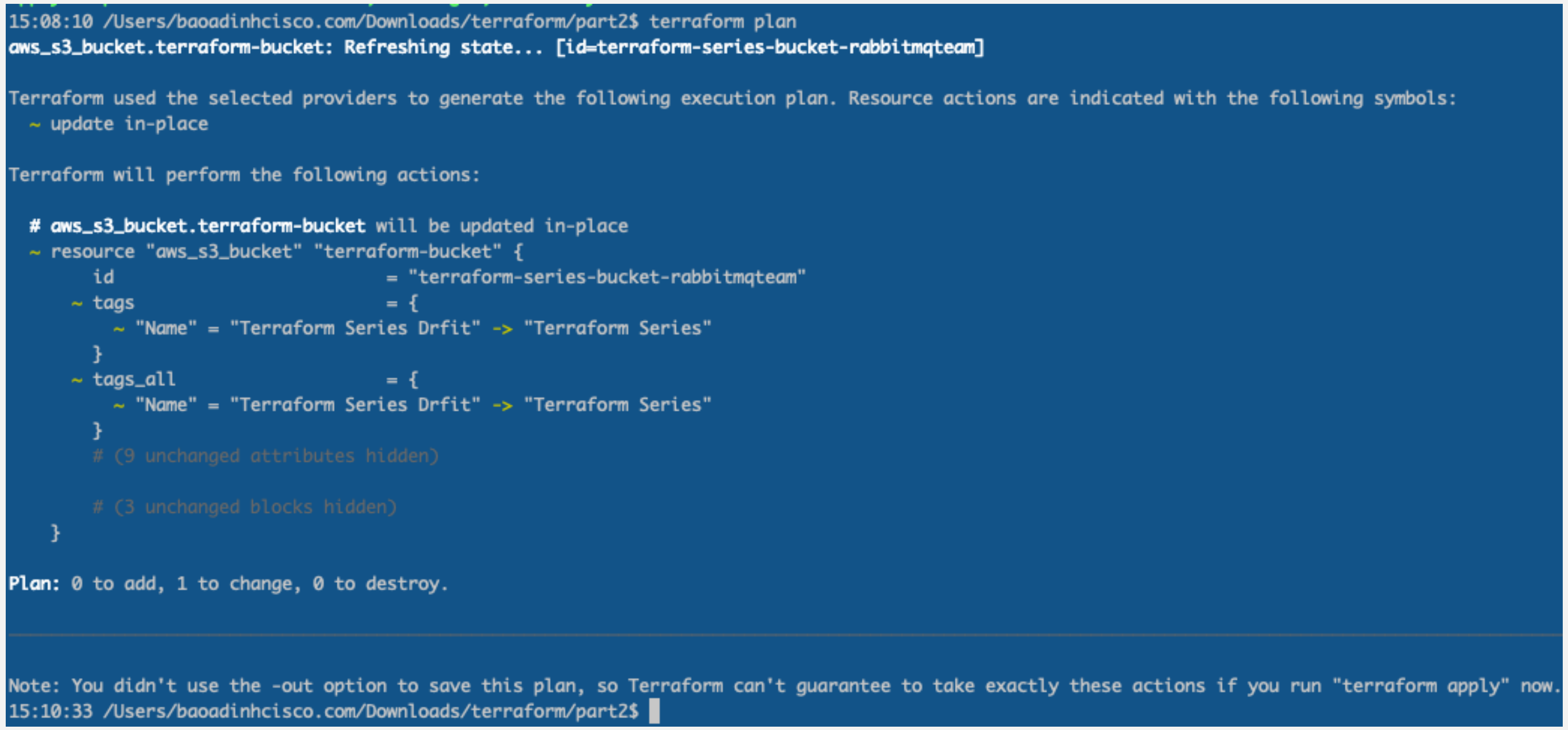


Tags (1)

You can use bucket tags to track storage costs and organize buckets. Learn more

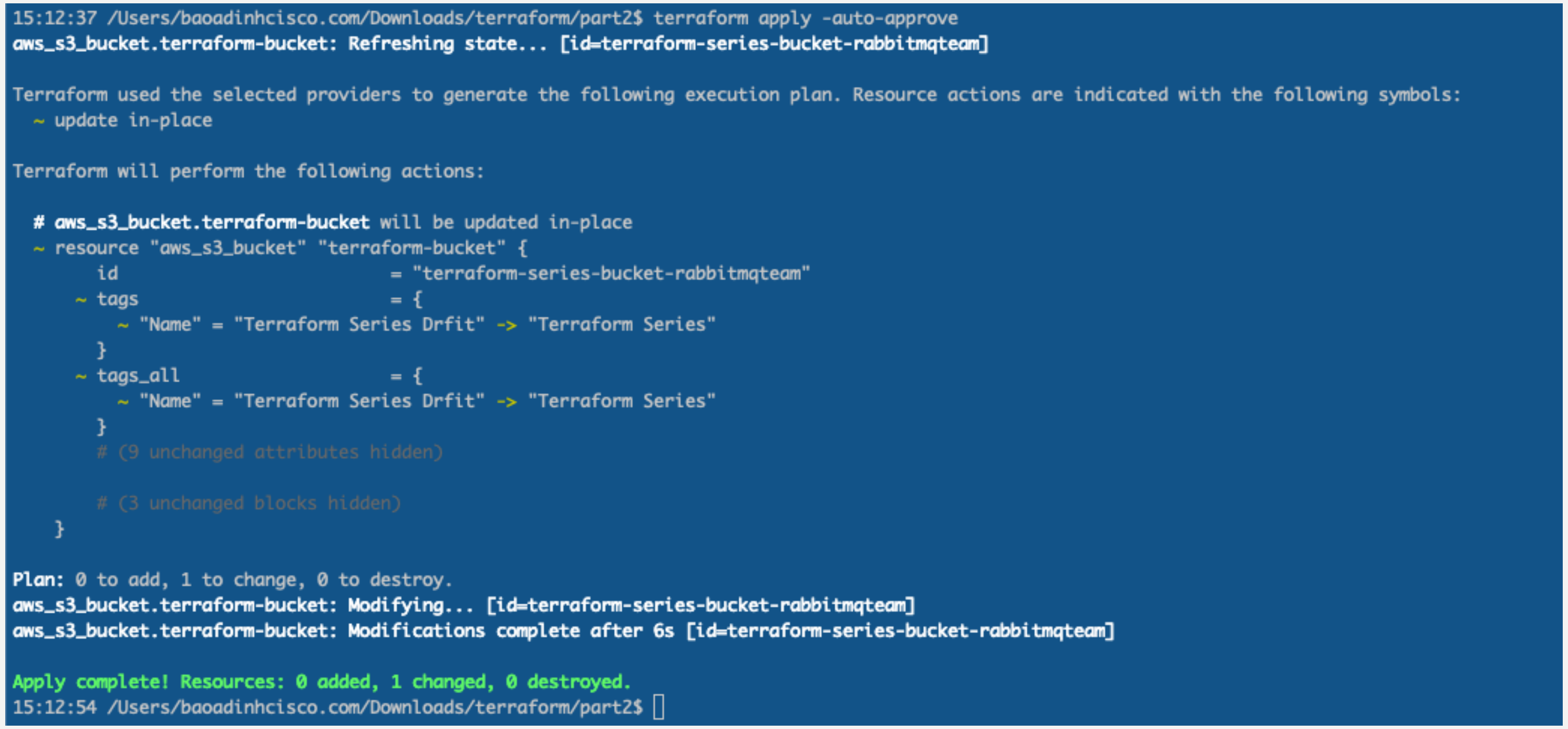
Key	Value
Name	Terraform Series Drfit

Run `terraform plan` to see what Terraform has to say for itself:

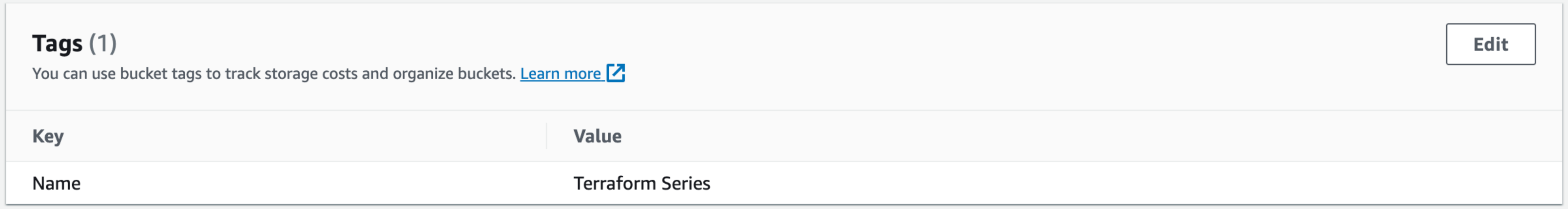


```
15:08:10 /Users/baoadinhcisco.com/Downloads/terraform/part2$ terraform plan
aws_s3_bucket.terraform-bucket: Refreshing state... [id=terraform-series-bucket-rabbitmqteam]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  # aws_s3_bucket.terraform-bucket will be updated in-place
  ~ resource "aws_s3_bucket" "terraform-bucket" {
        id                          = "terraform-series-bucket-rabbitmqteam"
      ~ tags                        = {
          ~ "Name" = "Terraform Series Drfit" -> "Terraform Series"
        }
      ~ tags_all                    = {
          ~ "Name" = "Terraform Series Drfit" -> "Terraform Series"
        }
        # (9 unchanged attributes hidden)

        # (3 unchanged blocks hidden)
    }

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

─────────────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run "terraform apply" now.
15:10:33 /Users/baoadinhcisco.com/Downloads/terraform/part2$
```

Rerun the `apply` command and we will see that the `tags` will be updated as before.



```
15:12:37 /Users/baoadinhcisco.com/Downloads/terraform/part2$ terraform apply -auto-approve
aws_s3_bucket.terraform-bucket: Refreshing state... [id=terraform-series-bucket-rabbitmqteam]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  # aws_s3_bucket.terraform-bucket will be updated in-place
  ~ resource "aws_s3_bucket" "terraform-bucket" {
        id                          = "terraform-series-bucket-rabbitmqteam"
      ~ tags                        = {
          ~ "Name" = "Terraform Series Drfit" -> "Terraform Series"
        }
      ~ tags_all                    = {
          ~ "Name" = "Terraform Series Drfit" -> "Terraform Series"
        }
        # (9 unchanged attributes hidden)

        # (3 unchanged blocks hidden)
    }

Plan: 0 to add, 1 to change, 0 to destroy.
aws_s3_bucket.terraform-bucket: Modifying... [id=terraform-series-bucket-rabbitmqteam]
aws_s3_bucket.terraform-bucket: Modifications complete after 6s [id=terraform-series-bucket-rabbitmqteam]

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.
15:12:54 /Users/baoadinhcisco.com/Downloads/terraform/part2$ 
```



Tags (1)

You can use bucket tags to track storage costs and organize buckets. Learn more

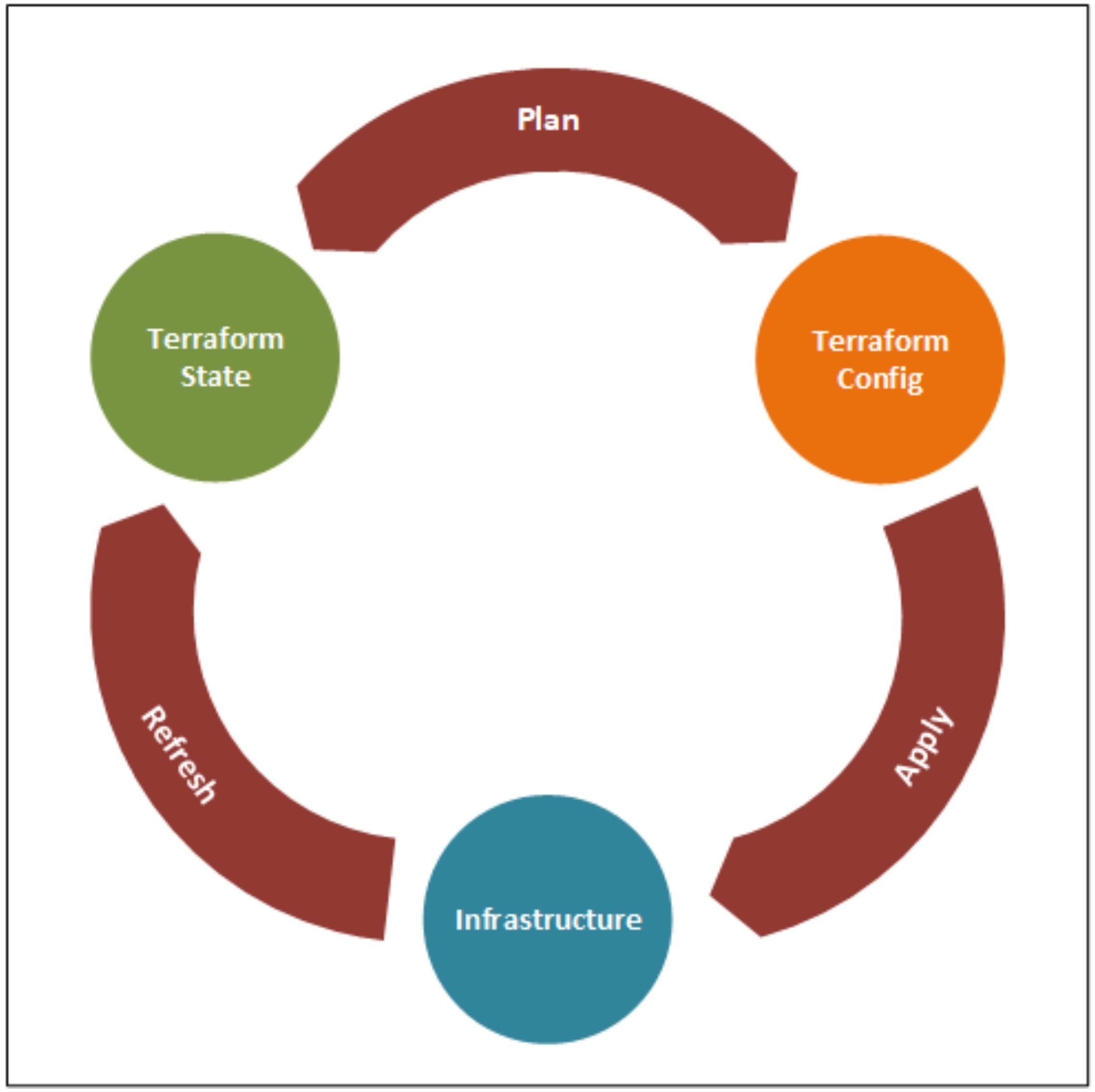
Key	Value
Name	Terraform Series

IDENTIFYING DRIFTS

- The terraform refresh command helps refresh the state file, and the plan
command provides a plan of action by analyzing the state file and current configuration.
 The output provided by the plan command helps us identify drifts.

- Running refresh does not modify infrastructure, but does modify the state file.
If the state has drifted from the last time Terraform ran, refresh allows that drift to be detected

THE LIFECYCLE OF A RESOURCE IN TERRAFORM


Plan
Terraform State
Terraform Config
Refresh
Apply
Infrastructure

SUMMARY

- Each managed resource has life cycle function hooks associated with it:
`Create()`, `Read()`, `Update()`, and `Delete()`.
Terraform invokes these function hooks as part of its normal operations.

- Changing Terraform configuration code and running `terraform apply`
will update an existing managed resource. You can also use `terraform refresh`
to update the state file based on what is currently deployed.

- Terraform reads the state file during a plan to decide what actions to take during an `apply`.
It's important not to lose the state file, or Terraform will lose track of all the resources it's managing

To be continued...

Chapter 3 is our first look at variables and functions

Q&A