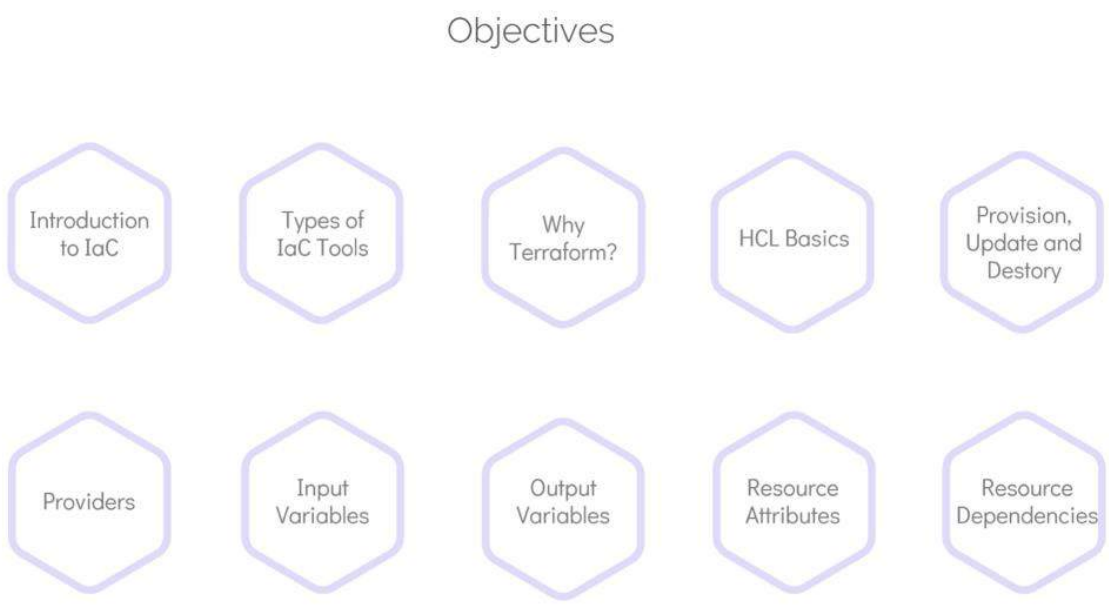
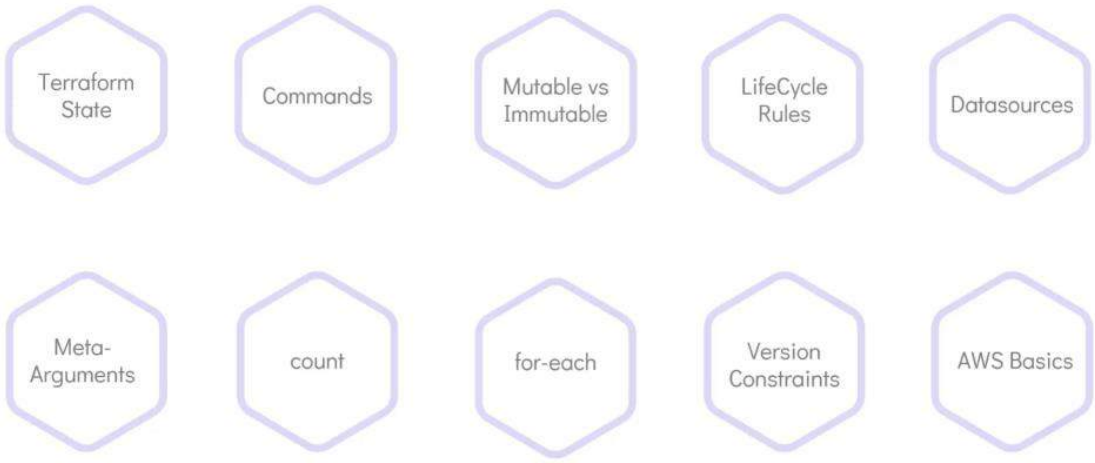
https://cisco.udemy.com/course/terraform-for-the-absolute-beginners/

Section 1: Introduction
Section 2: Introduction to infrastructure as code
Section 3: getting started with terraform
Section 4: Terraform basics
Section 5: Terraform state
Section 6: working with terraform
Section 7: terraform with aws
Section 8: Remote state
Section 9: terraform provisioners
Section 10: terraform import tainting resources and debugging
Section 11: terraform modules
Section 12: terraform functions and conditional expression

Section 1: Introduction



Objectives

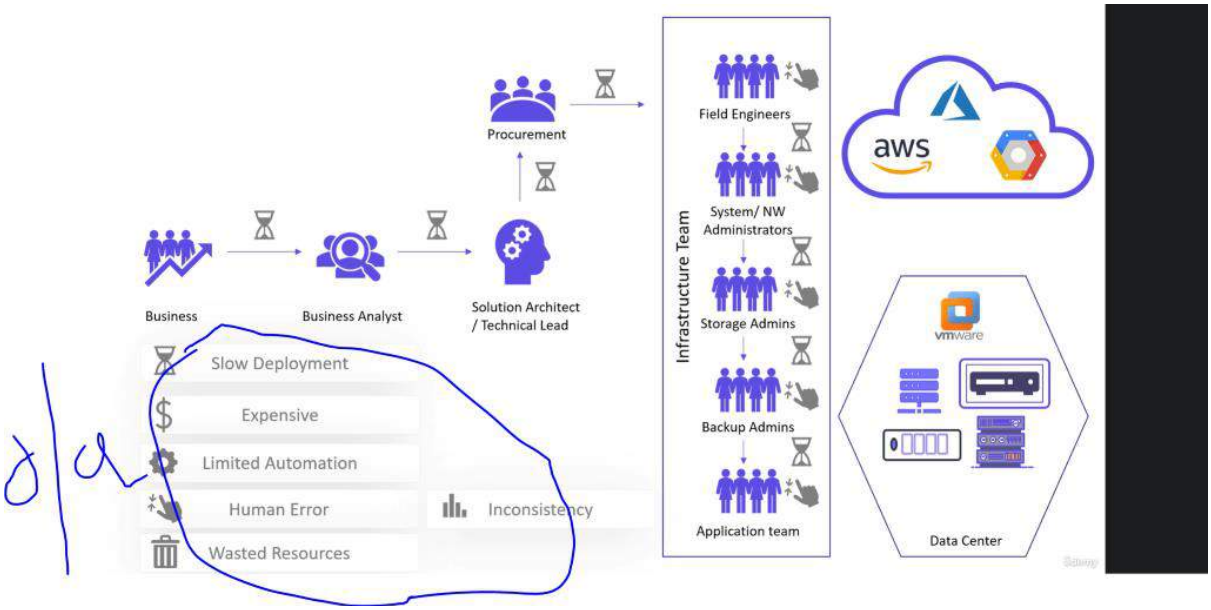


Objectives

Join Our Community
Join this community of students on discussions about the course: https://community.kodekloud.com/c/terraform-for-beginners
Join our slack channel https://join.slack.com/t/kodekloud/shared_invite/zt-1eg9wa4bl-iccmKnUM7pUXpqUdKUNlgQ

Section 2: Introduction to infrastructure as code

Challenges with traditional it infrastructure



This model have quite d/a.

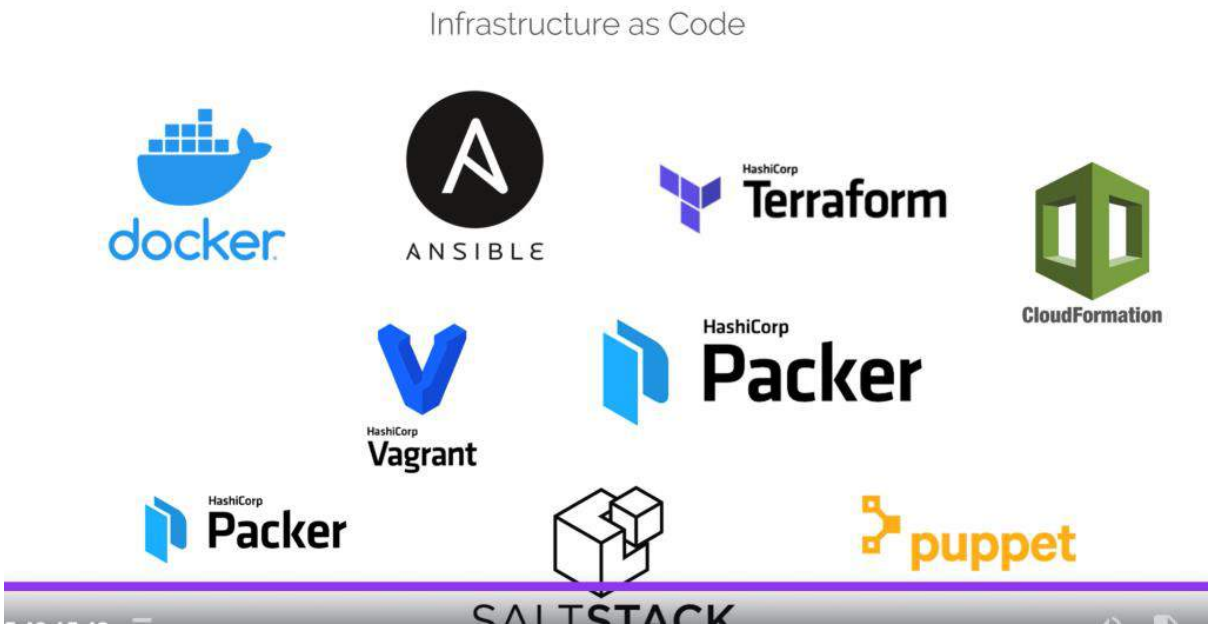
Scaling up or scaling down on demand cannot achieve quickly

Note :
By moving to cloud the time spin up infrastructure are reduced

The data center, server and services are manged by the cloud provider
Infrastructure cost reduced

But still human error is there while configuring infrastructure in aws.
Everyone was solving same problem trying to automate infrastructure provisioning
To deploy environment faster.

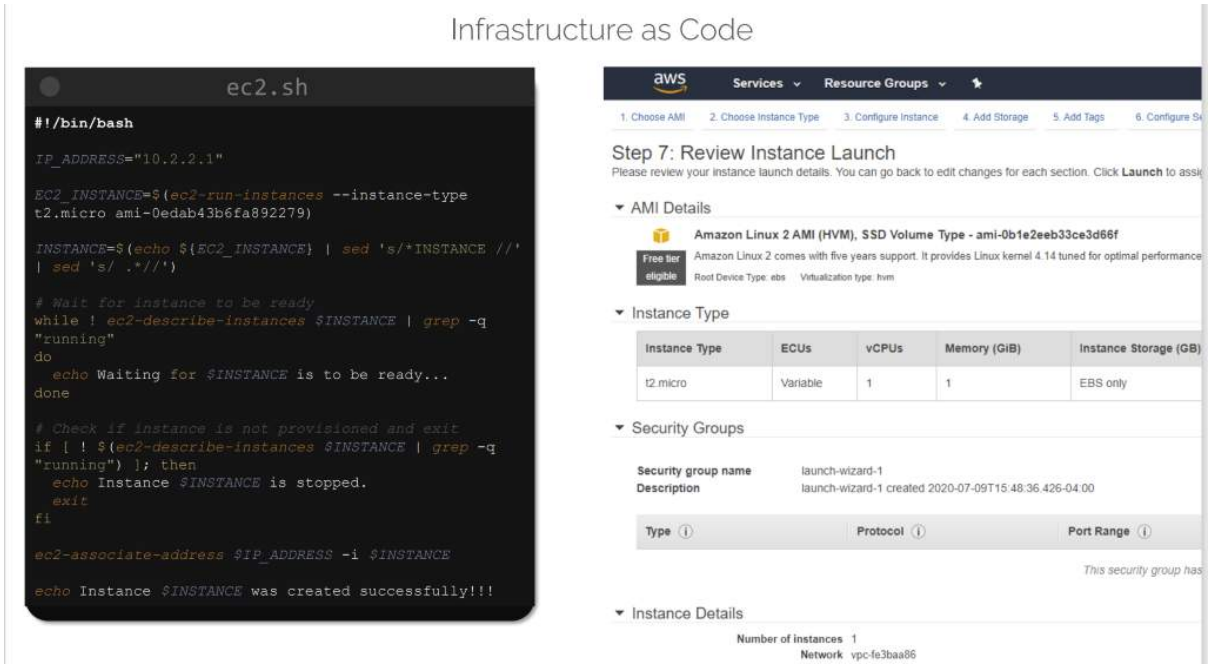
These evolves set of tool that's is known as infrastructure as code



Type of IAC(Infrastructure as code) tool

What is meant by infrastructure as a code?

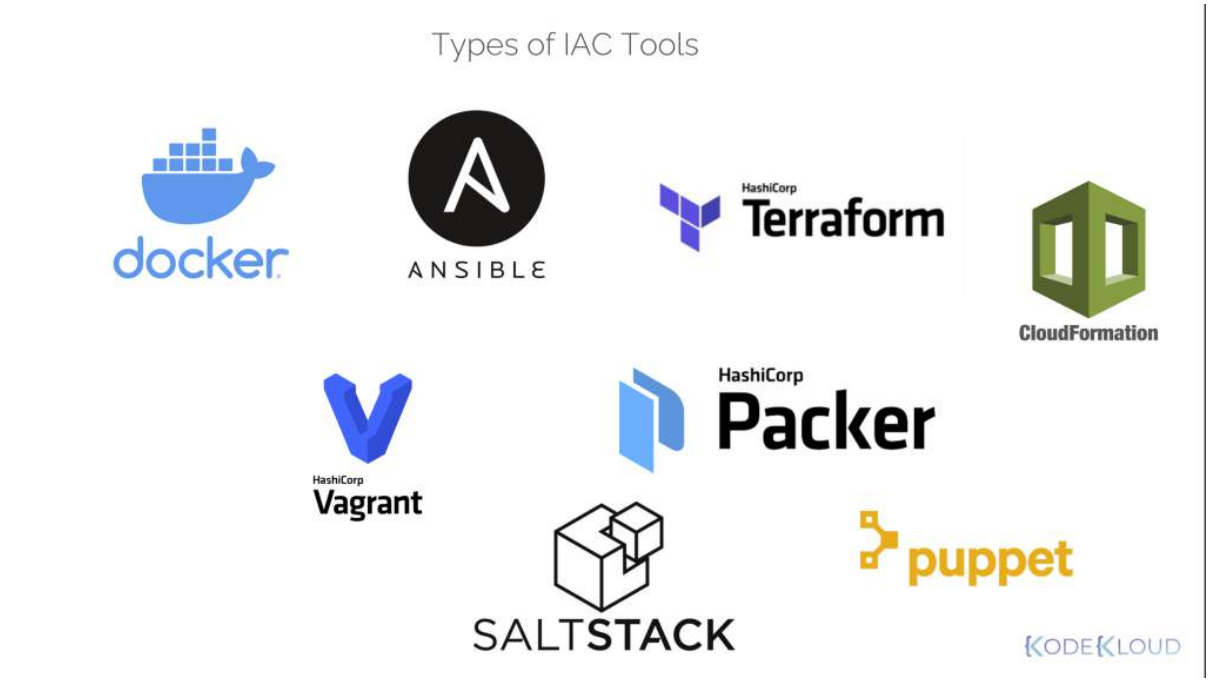
Overview. Infrastructure as Code (IaC) is **the managing and provisioning of infrastructure through code instead of through manual processes**. With IaC, configuration files are created that contain your infrastructure specifications, which makes it easier to edit and distribute configurations

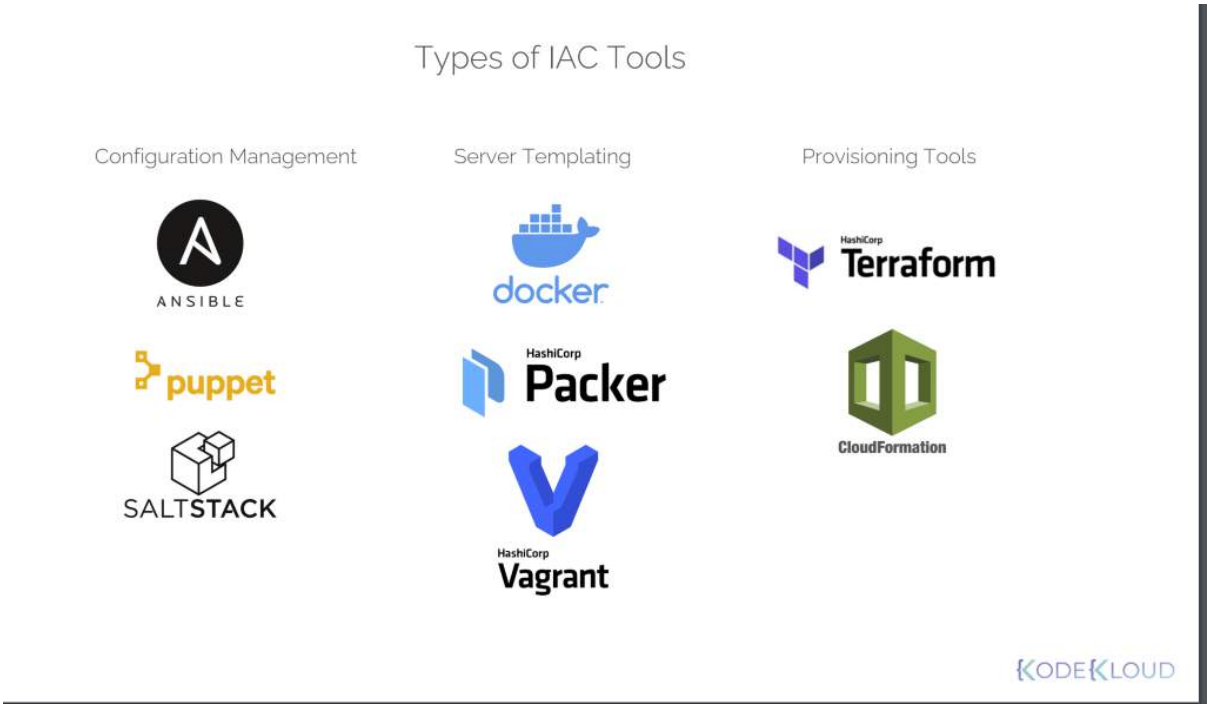


The code here we see is shell script which is not easy to maintain since it is required development sill to learn

That's why terraform is useful . A large shell script we can convert into simple terraform files

Note :
There are several IAC tool but they all are addressed to a very specific tool





Configuration management:

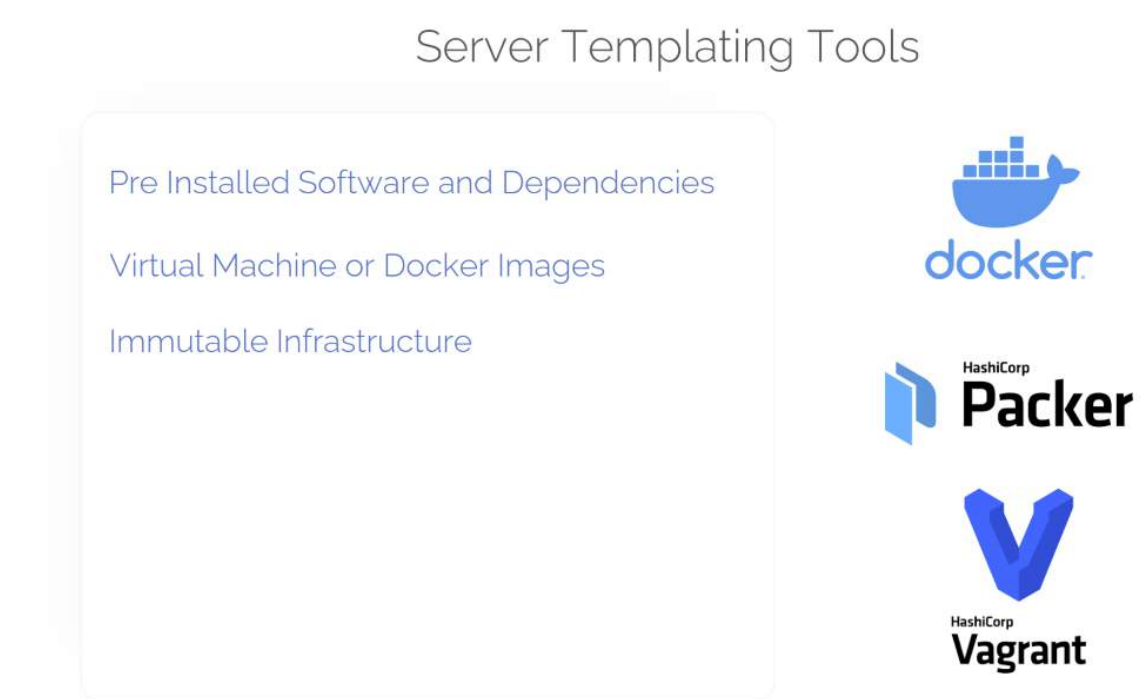
These are usually use to install and manage sw on existing infrastructure resources such as server d/b etc

Idempotent: everytime we run it it will change only desirable changes

What do you mean by idempotency?
Idempotence, in programming and mathematics, is **a property of some operations such that no matter how many times you execute them, you achieve the same result**

Server templating tools

This remove the need of installing sw after vm or containter deploy



Provisioning tools

Cloud formation is for aws
And terraform provides plugins for all major cloud providers

Why terraforms

Terraform is iac tool which is specially use as infrastructure provisioning tool

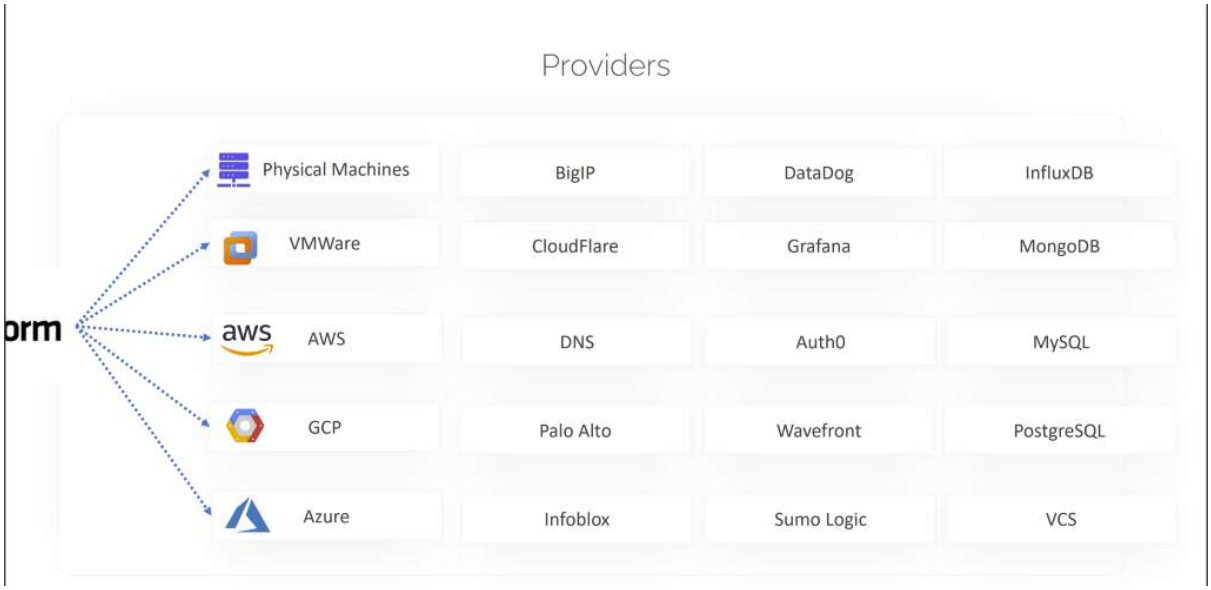
Terraform is free and open source tool which is developed by **hashicorp.**

Advantages:
It can deploy infrastructure across multiple platform

How terraform manages infrastructure on so many diff kind of platforms

This is achieved through providers.
Provider help terraform manage third party platform through api

Terraform has 100 of such provider



Terraform uses HCL (hashicorp configuration language)

Extension is :- .tf

Note : the configuration is easy to read and write
This code is declarative and can be control is version control system

HashiCorp Configuration Language

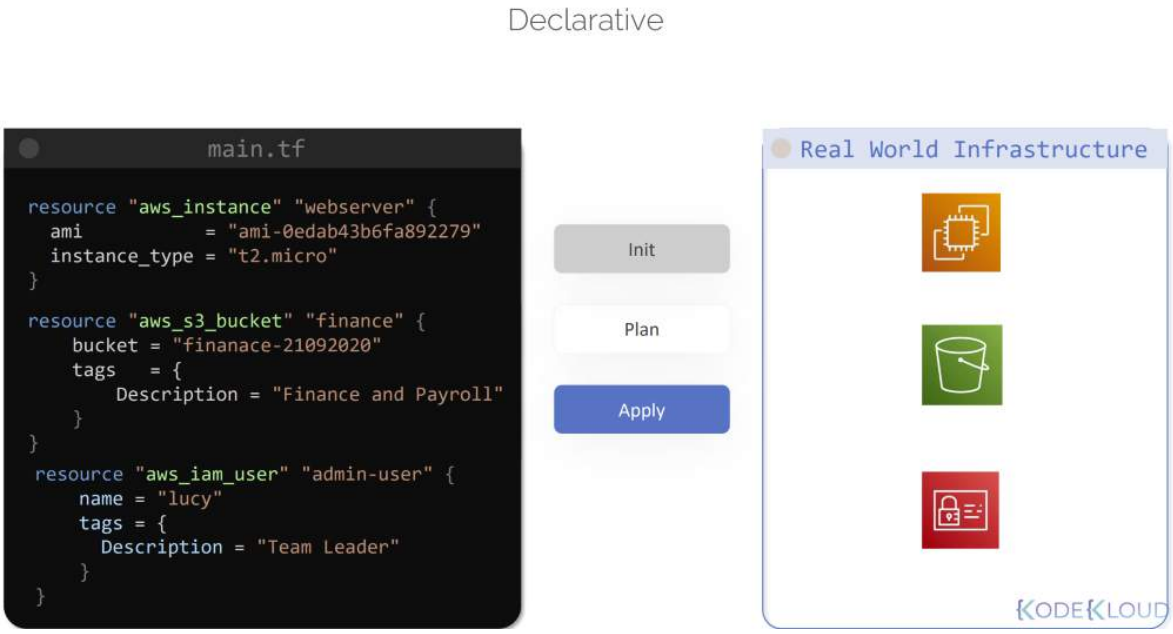
main.tf

```
resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"
}

resource "aws_s3_bucket" "finance" {
    bucket = "finanace-21092020"
    tags   = {
        Description = "Finance and Payroll"
    }
}
 resource "aws_iam_user" "admin-user" {
     name = "lucy"
     tags = {
       Description = "Team Leader"
     }
 }
```

Terraform work in **three phase init , plan and apply**

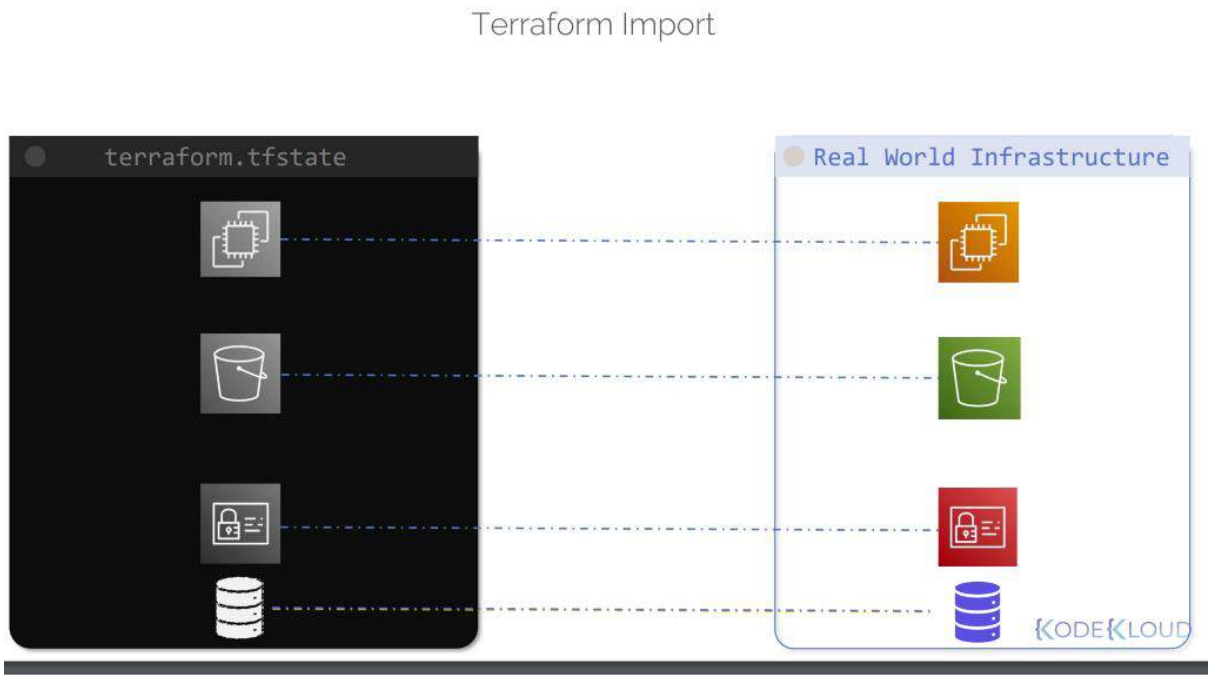
During the init phase terraform initialize the project and identify the providers to be use for target environment

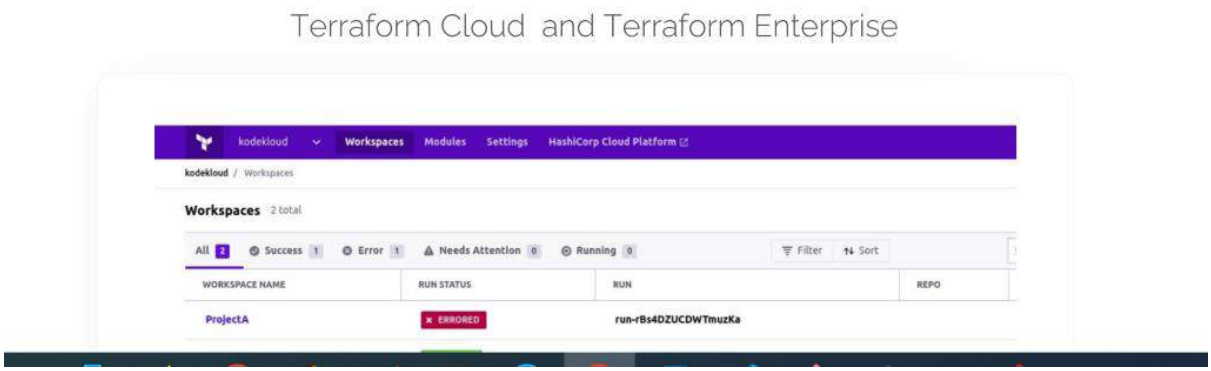
During the apply phase terraform did the necessary changes on the target environment to bring it to the desired state



Note :

Every object that terraform manages is called resource
Terraform can read existing attribute of infrastructure by configuring datasource
Terraform can also import other resources outside terraform either manually or by means of other iac tool





What is a Terraform used for?

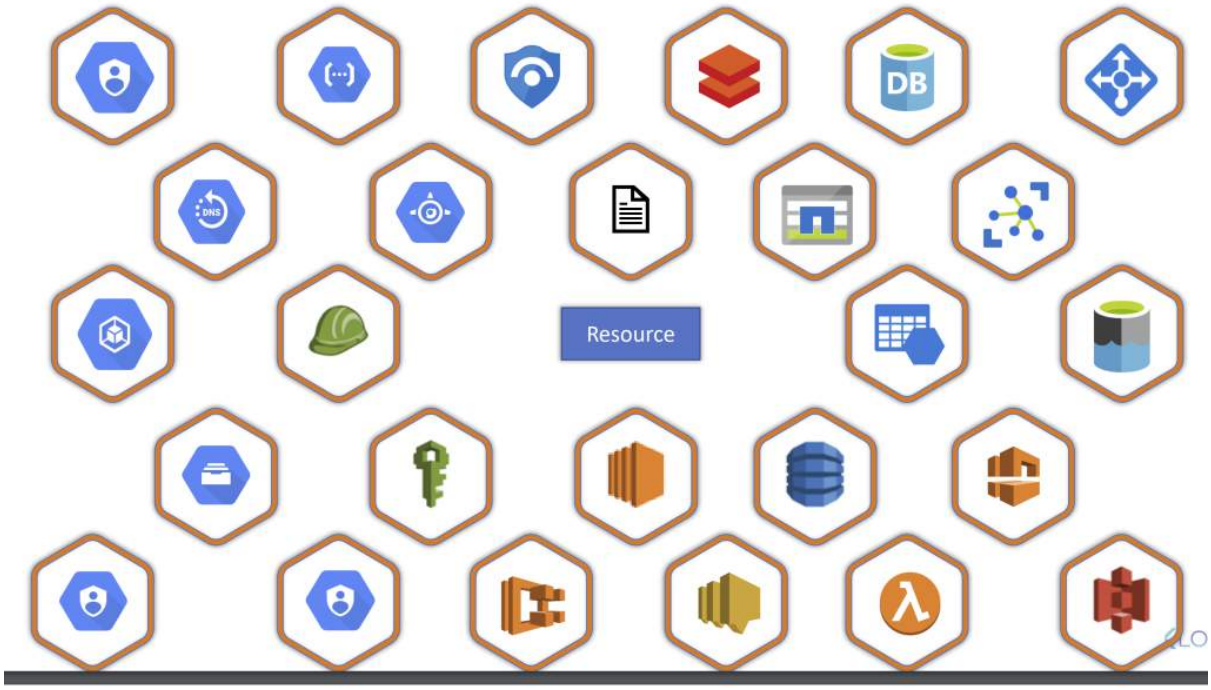
Terraform is an IAC tool, used primarily by DevOps teams **to automate various infrastructure tasks**. The provisioning of cloud resources, for instance, is one of the main use cases of Terraform. It's a cloud-agnostic, open-source provisioning tool written in the Go language and created by HashiCorp

Section 3: getting started with terraform

Installing terraform

Resource : is an object that terraform manages

It could be a file on local machine
It could be a virtual machine on cloud such as ec2 instances
It could be a services like s3, ecs , dynamo db, iam users , roles, provider etc



Hashicorp configuration language – basics

HCL syntax

Hcl file consist of **block and argument**

Each resource type consider a specific argument

```
>_
$ mkdir /root/terraform-local-file
$ cd /root/terraform-local-file
```

local.tf

```
resource "local_file" "pet" {
  filename = "/root/pets.txt"
  content = "We love pets!"
}
```


```
C:\Users\saurakes>terraform -version
Terraform v1.3.7
on windows_386
```

aws-ec2.tf

```
resource "aws_instance" "webserver" {
    ami = "ami-0c2f25c1f66a1ff4d"
    instance_type = "t2.micro"
}
```

aws-s3.tf

```
resource "aws_s3_bucket" "data" {
    bucket = "webserver-bucket-org-2207"
    acl    = "private"
}
```

Terraform init
Terraform plan
Terraform apply
Terraform destroy

Init

The terraform init command **initializes a working directory containing Terraform configuration files**. This is the first command that should be run after writing a new Terraform configuration or cloning an existing one from version control. It is safe to run this command multiple times.

Plan

The terraform plan command **creates an execution plan, which lets you preview the changes that Terraform plans to make to your infrastructure**. By default, when Terraform creates a plan it: Reads the current state of any already-existing remote objects to make sure that the Terraform state is up-to-date.

Apply

What is Terraform Apply? The terraform apply command **executes the actions proposed in a terraform plan** . It is used to deploy your infrastructure. Typically apply should be run after terraform init and terraform plan

A simple terraform workflow consist of four step

First : write the configuration file

Second : run the terraform **init** command
Third : review the execution plan using the terraform **plan** command
Fourth : once we are ready we should run apply command

Terraform init : this command will check the configuration file and intialie the working directory containing the .tf file

It will download the plugin to work with .tf file

Terraform plan command will display output similar to diff command in git.
This command will help user to review it

Terraform apply

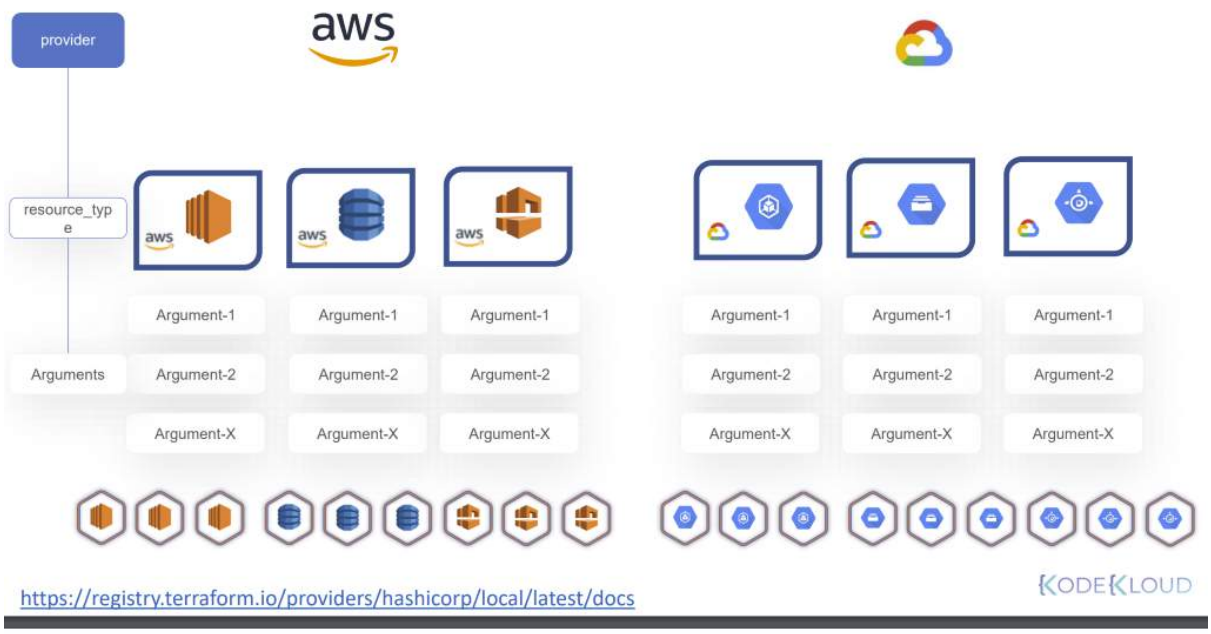
Terraform show **c**ommand we can use after

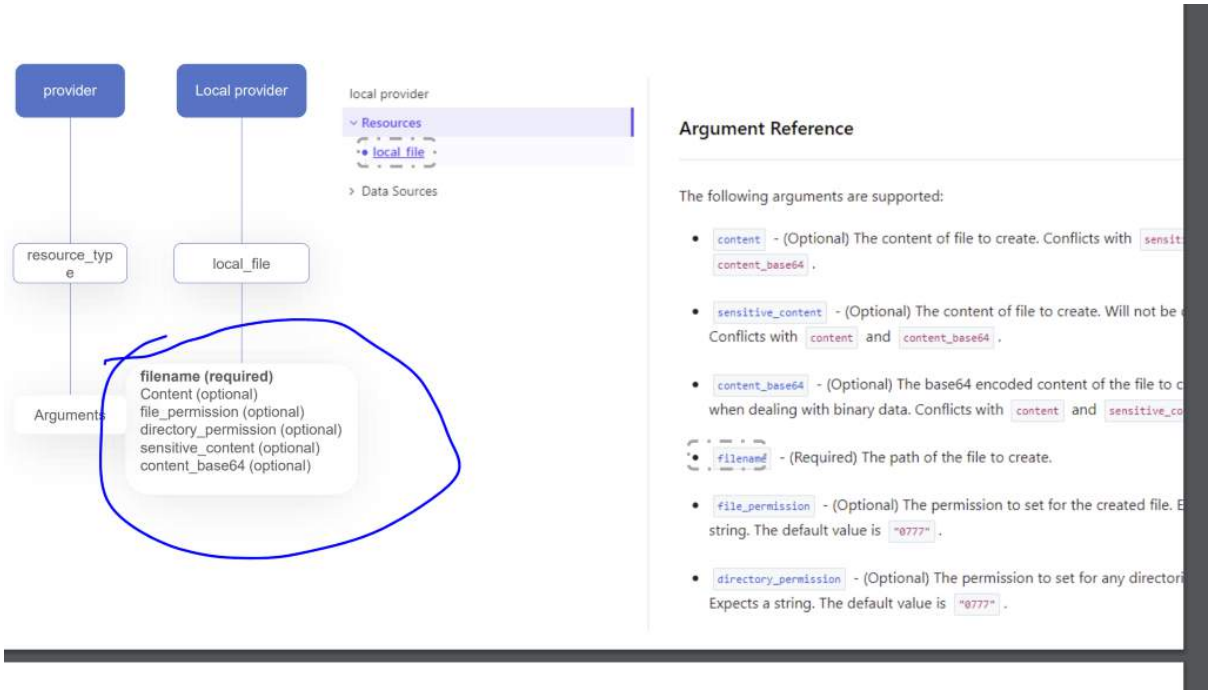
Local.tf

```
resource "local_file" "test"{
filename="C://Users//saurakes//test//pet.txt"
content="we love pets"
}
```

```
C:\Users\saurakes\OneDrive - Cisco\Desktop\teraform>terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/local...
- Installing hashicorp/local v2.2.3...
- Installed hashicorp/local v2.2.3 (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.

C:\Users\saurakes\OneDrive - Cisco\Desktop\teraform>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:

  # local_file.test will be created
  + resource "local_file" "test" {
      + content              = "we love pets"
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "C://Users//saurakes//test//pet.txt"
      + id                   = (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.
```

```
C:\Users\saurakes\OneDrive - Cisco\Desktop\teraform>terraform apply

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:

  # local_file.test will be created
  + resource "local_file" "test" {
      + content              = "we love pets"
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "C://Users//saurakes//test//pet.txt"
      + id                   = (known after apply)
    }

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.

  Enter a value: yes

local_file.test: Creating...
local_file.test: Creation complete after 0s [id=d87a5018d4e1c1e1644d11eafcdf53a95c8968ef]

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

C:\Users\saurakes\OneDrive - Cisco\Desktop\teraform>terraform show
# local_file.test:
resource "local_file" "test" {
    content              = "we love pets"
    directory_permission = "0777"
    file_permission      = "0777"
    filename             = "C://Users//saurakes//test//pet.txt"
    id                   = "d87a5018d4e1c1e1644d11eafcdf53a95c8968ef"
}

C:\Users\saurakes\OneDrive - Cisco\Desktop\teraform>
```





Update and destroy infrastructure

Requirement :

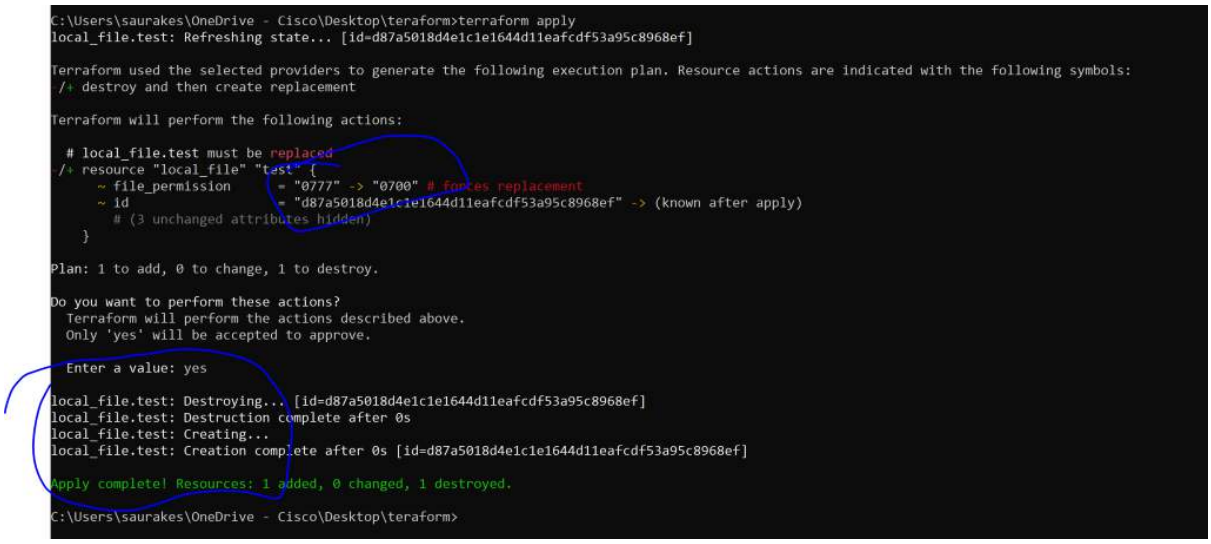
How to update and destroy local file using terraform

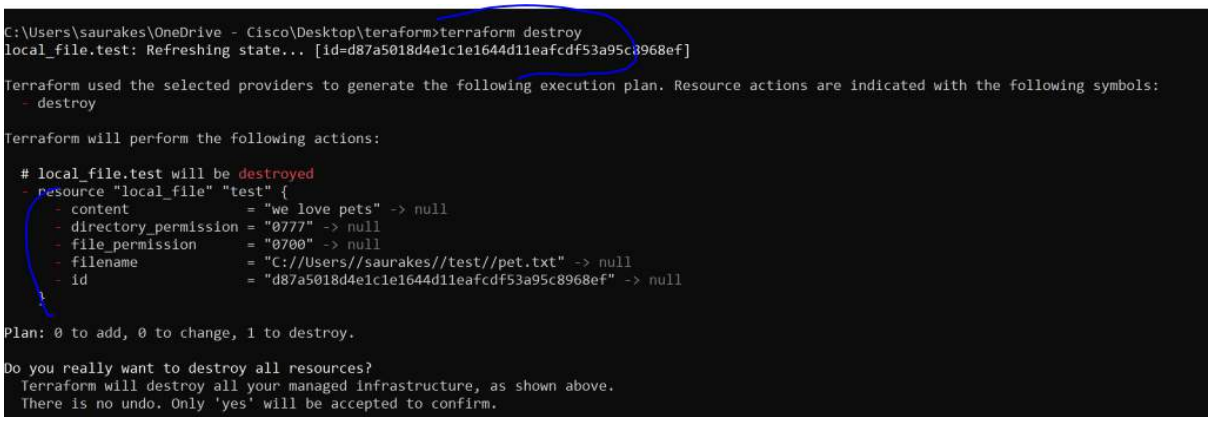
-/+ symbol :- file is destroyed and create again

This form of infrastructure is immutable

```
resource "local_file" "test"{
filename="C://Users//saurakes//test//pet.txt"
content="we love pets"
file_permission="0700"
}
```

```
C:\Users\saurakes\OneDrive - Cisco\Desktop\teraform>terraform plan
local_file.test: Refreshing state... [id=d87a5018d4e1c1e1644d11eafcdf53a95c8968ef]

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:

  # local_file.test must be replaced
-/+ resource "local_file" "test" {
      ~ file_permission      = "0777" -> "0700" # forces replacement
      ~ id                   = "d87a5018d4e1c1e1644d11eafcdf53a95c8968ef" -> (known after apply)
        # (3 unchanged attributes hidden)
    }

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

C:\Users\saurakes\OneDrive - Cisco\Desktop\teraform>
```





Terraform destroy : will deletes the file

Section 4: Terraform basics

Using Terraform providers

Terraform init : terraform downloads and install plugin

Terraform init command when run shows the version of the plugin that has been installed

The plugin is downloaded into the hidden directory .terraform in the working directory

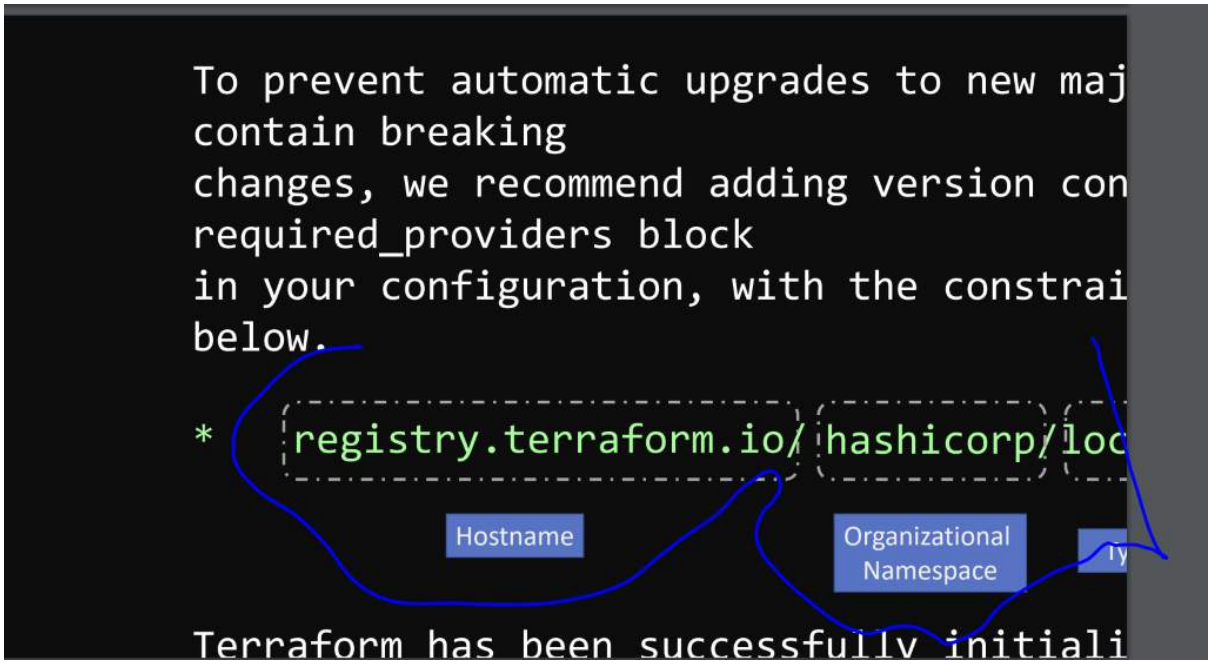
KODEKLOUD

registry.terraform.io

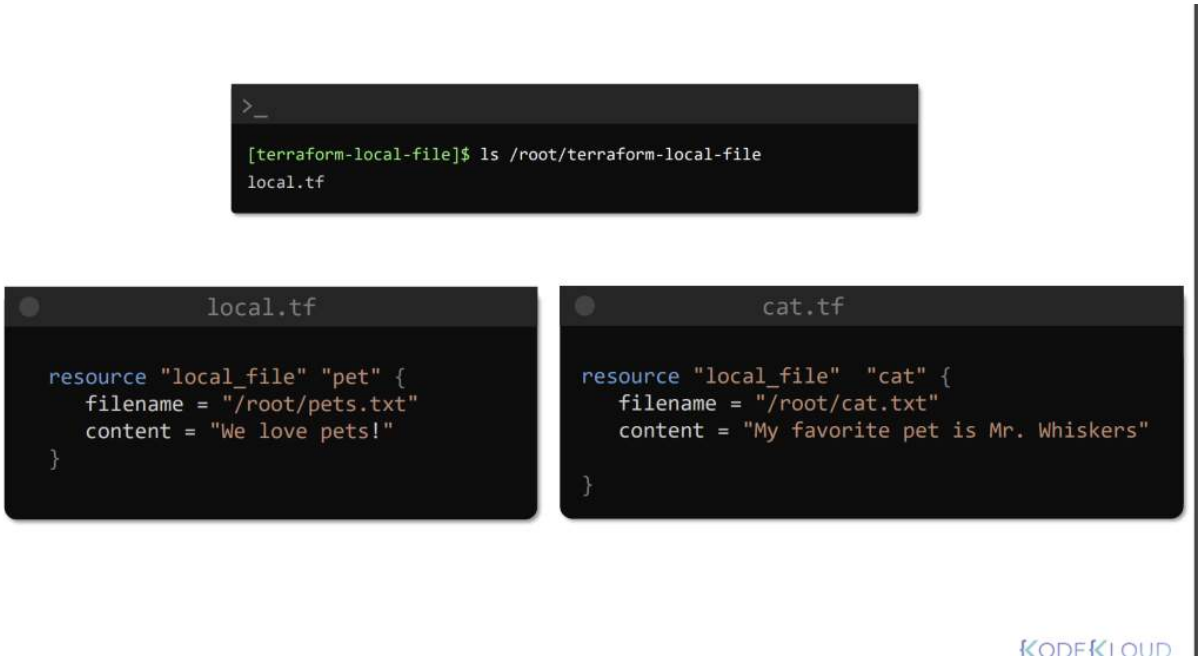
KODEKLOUD

https://registry.terraform.io/

This is the public hostname where plugin is there
By default terraform install the latest version of plugin



Configuration Directory



Single configuration file we can use

main.tf

```
resource "local_file" "pet" {
    filename = "/root/pets.txt"
    content = "We love pets!"
}
resource "local_file"  "cat" {
    filename = "/root/cat.txt"
    content = "My favorite pet is Mr. Whiskers"

}
```

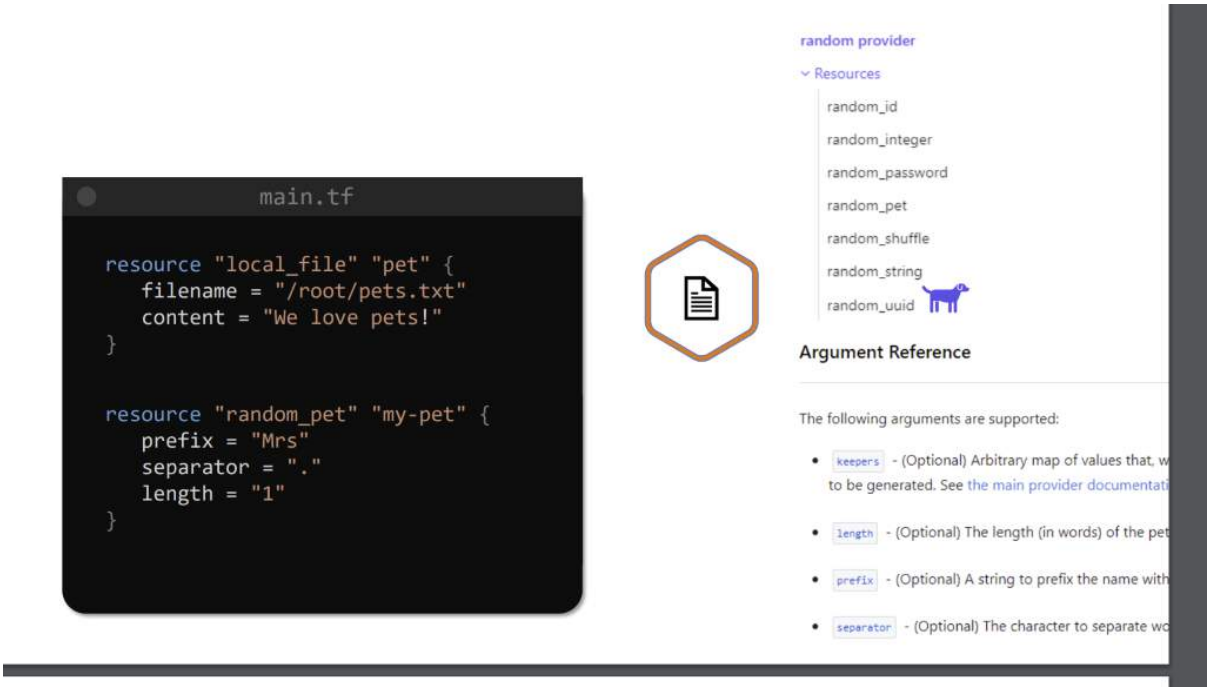
File Name	Purpose
main.tf	Main configuration file containing resource definition
variables.tf	Contains variable declarations
outputs.tf	Contains outputs from resources
provider.tf	Contains Provider definition

KODEKLOUD

Multiple providers

How to use multiple providers and resources in terraform

Until now we use single provider local. Terraform also suppose we can use multiple provider within same configuration file



```
>_

$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but
will not be
persisted to local or remote state storage.

local_file.pet: Refreshing state...
[id=d1a31467f206d6ea8ab1cad382bc106bf46df69e]

.
.
  # random_pet.my-pet will be created
  + resource "random_pet" "my-pet" {
      + id        = (known after apply)
      + length    = 1
      + prefix    = "Mrs"
      + separator = "."
    }

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

KODEKLOUD

```
>_

$ terraform init
Initializing the backend...

Initializing provider plugins...
- Using previously-installed hashicorp/local v2.0.0
- Finding latest version of hashicorp/random...
- Installing hashicorp/random v2.3.0...
- Installed hashicorp/random v2.3.0 (signed by HashiCorp)

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, we recommend adding version constraints in a required_providers
block
in your configuration, with the constraint strings suggested below.

* hashicorp/local: version = "~> 2.0.0"
* hashicorp/random: version = "~> 2.3.0"

Terraform has been successfully initialized!
```

KODEKLOUD

Using Input Variables

They are considered to be hard coded value. Hard coded value does not a good idea.
This limit the reusability of code.

We need to make sure same file can be deploy based on input variable which we will provide while execution

```
resource "local_file" "test"{
filename=var.filename
content=var.content
}
```

```
variable "filename"{
default="C://Users/saurakes//test//test.txt"
}
variable "content"{
default="hello how are you"
}
```

main.tf

```
resource "local_file" "pet" {
    filename = var.filename
    content = var.content

}

resource "random_pet" "my-pet" {
    prefix = var.prefix
    separator = var.separator
    length = var.length
}
```

variables.tf

```
variable "filename" {
        default = "/root/pets.txt"
}
variable "content" {
        default = "We love pets!"
}
variable "prefix" {
        default = "Mrs"
}
variable "separator" {
        default = "."
}
variable "length" {
        default = "1"
}
```

main.tf

```
resource "aws_instance" "webserver" {
  ami           = var.ami
  instance_type = var.instance_type
}
```

variables.tf

```
variable "ami" {
  default = "ami-0edab43b6fa892279"
}
variable "instance_type" {
  default = "t2.micro"
}
```

Understanding the various block

Variable block uses three parameter

Default
Type (optional)
Description (it is optional)

Note : if type is not specified in block then by default type is **any**

variables.tf

```
variable "filename" {
   default = "/root/pets.txt"
   type = string
   description = "the path of local file"

}
variable "content" {
    default = "I love pets!"
    type = string
    description = “the content of the file"

}
variable "prefix" {
    default = "Mrs"
    type = string
    description = "the prefix to be set"

}
variable "separator" {
    default = "."
```

variables.tf

```
variable "filename" {
   default = "/root/pets.txt"
   type = string
   description = "the path of local file"

}
variable "content" {
    default = "I love pets!"
    type = string
    description = "the content of the file"

}
variable "prefix" {
    default = "Mrs"
    type = string
    description = "the prefix to be set"

}
variable "separator" {
    default = "."
```

Type	Example
string	"/root/pets.txt"
number	1
bool	true/false
any	Default Value

variables.tf

```
variable "length" {
  default = "2"
  type = number
  description = "length of the pet name"
}

variable "password_change" {
  default = "true"
  type = bool
}
```

Type	Example
string	"/root/pets.txt"
number	1
bool	true/false
any	Default Value
list	["cat", "dog"]
map	pet1 = cat pet2 = dog
object	Complex Data Structure
tuple	Complex Data Structure

List

variables.tf

```
variable "prefix" {
  default = ["Mr", "Mrs", "Sir"]
  type = list  0     1      2
}
```

maint.tf

```
resource "random_pet" "my-pet" {
  prefix    = var.prefix[0]
}
```

Index	Value
0	Mr
1	Mrs
2	Sir

KODEKLOUD

Map

variables.tf

```
variable file-content {
  type        = map
  default     = {
      "statement1" = "We love pets!"
      "statement2" = "We love animals!"
  }
}
```

maint.tf

```
resource local_file my-pet {
  filename  = "/root/pets.txt"
  content = var.file-content["statement2"]
}
```

Key	Value
statement1	We love pets!
statement2	We love animals!

KODEKLOUD

List of a Type

variables.tf

```
variable "prefix" {
  default = ["Mr", "Mrs", "Sir"]
  type = list(string)
}
```

variables.tf

```
variable "prefix" {
  default = ["Mr", "Mrs", "Sir"]
  type = list(number)
}
```

variables.tf

```
variable "prefix" {
  default = ["1", "2", "3"]
  type = list(number)
}
```

>_

```
$ terraform plan
Error: Invalid default value for variable

  on variables.tf line 3, in variable "prefix":
   3:   default     = ["Mr", "Mrs", "Sir"]

This default value is not compatible with the
variable's type constraint: a number is required.
```

Map of a Type

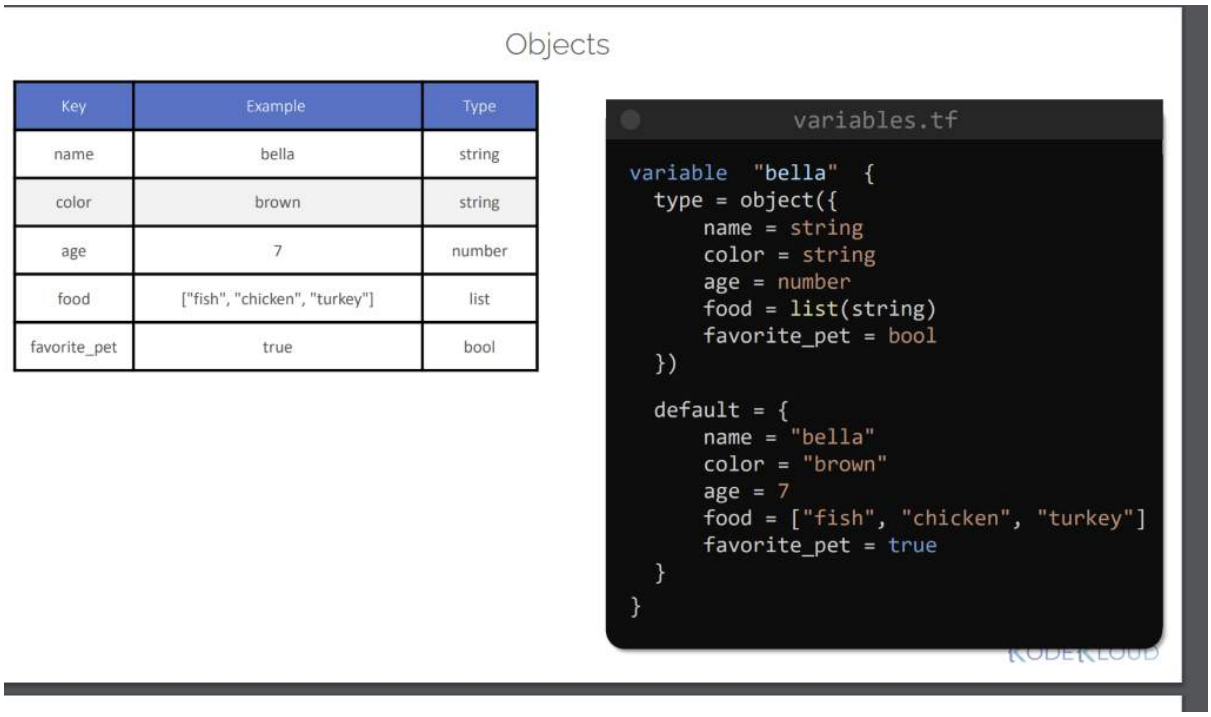
variables.tf

```
variable "cats" {
  default = {
    "color" = "brown"
    "name" = "bella"
  }
  type = map(string)
}
```

variables.tf

```
variable "pet_count" {
  default = {
    "dogs" = "3"
    "cats" = "1"
    "goldfish" = "2"
  }
  type = map(number)
}
```

Object : we can create complex data structure by combining all the variables types



Tuples is similar to list and consist of sequence of element.
List uses same variable type but in case of tuples we can use different variable type

Tuples

variables.tf

```
variable kitty {
  type        = tuple([string, number, bool])
  default     = ["cat", 7, true]

}
```

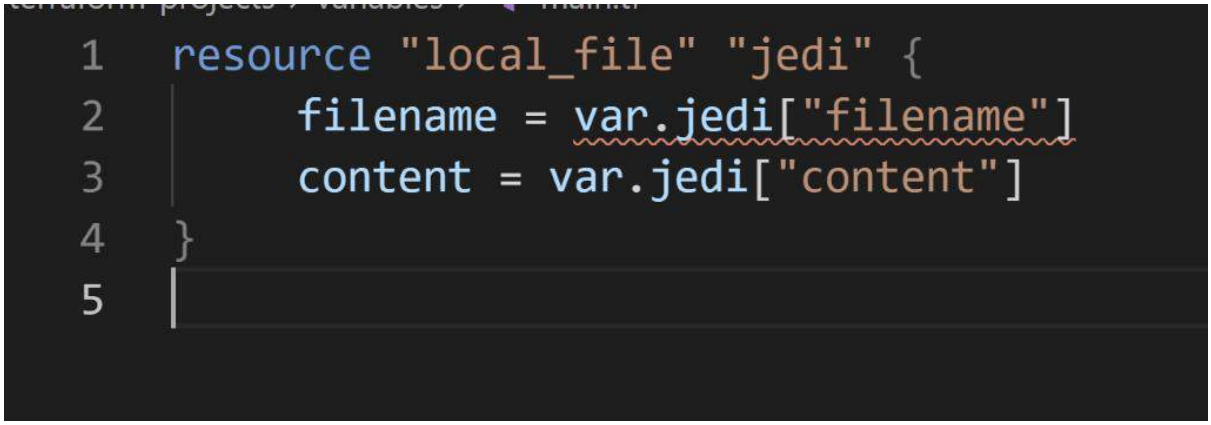
variables.tf

```
variable kitty {
  type        = tuple([string, number, bool])
  default     = ["cat", 7, true, "dog"]

}
```

>_

```
$ terraform plan
  Error: Invalid default value for variable

    on variables.tf line 3, in variable "kitty":
     3:   default     = ["cat", 7, true, "dog"]

  This default value is not compatible with the
  variable's type constraint:
  tuple required.
```

```
variable "name" {
     type = string
     default = "Mark"

}
variable "number" {
     type = bool
     default = true

}
variable "distance" {
     type = number
     default = 5

}
variable "jedi" {
     type = map
     default = {
     filename = "/root/first-jedi"
     content = "phanius"
     }

}

variable "gender" {
     type = list(string)
     default = ["Male", "Female"]
}
variable "hard_drive" {
     type = map
     default = {
          slow = "HHD"
          fast = "SSD"
     }
}
variable "users" {
     type = set(string)
     default = ["tom", "jerry", "pluto", "daffy", "donald", "jerry", "chip", "dale"]


}
```



Using variables in terraform

Will see different ways in which we can use input for terraforms

Local.tf

```
resource "local_file" "test"{
filename=var.filename
content=var.content
}
```

Variables.tf

```
variable "filename"{
default="C://Users/saurakes//test//test.txt"
}
variable "content"{
type =string
}
```

Terraform.tfvars

```
content="i m updated"
```

Default is optional

main.tf

```
resource "local_file" "pet" {
    filename = var.filename
    content = var.content

}

resource "random_pet" "my-pet" {
    prefix = var.prefix
    separator = var.separator
    length = var.length
}
```

variables.tf

```
variable "filename" {
        default = "/root/pets.txt"
}
variable "content" {
        default = "We love pets!"
}
variable "prefix" {
        default = "Mrs"
}
variable "separator" {
        default = "."
}
variable "length" {
        default = 2
}
```


main.tf

```
resource "local_file" "pet" {
    filename = var.filename
    content = var.content

}

resource "random_pet" "my-pet" {
    prefix = var.prefix
    separator = var.separator
    length = var.length
}
```

variables.tf

```
variable "filename" {

}
variable "content" {

}
variable "prefix" {

}
variable "separator" {

}
variable "length" {

}
```

Interactive Mode

```
>_
$ terraform apply
var.content
  Enter a value: We love Pets!

var.filename
  Enter a value:  /root/pets.txt

var.length
  Enter a value:  2

var.prefix
  Enter a value:  Mrs.

var.separator
  Enter a value: .
```

KODEKLOUD

Command Line Flags

```
>_
$ terraform apply -var "filename=/root/pets.txt" -var "content=We love
Pets!"  -var "prefix=Mrs" -var "separator=." -var "length=2"
```

Environment Variables

```
>_

$ export TF_VAR_filename="/root/pets.txt"
$ export TF_VAR_content="We love pets!"
$ export TF_VAR_prefix="Mrs"
$ export TF_VAR_separator="."
$ export TF_VAR_length="2"
$ terraform apply
```

KODEKLOUD

Variable Definition Files

terraform.tfvars

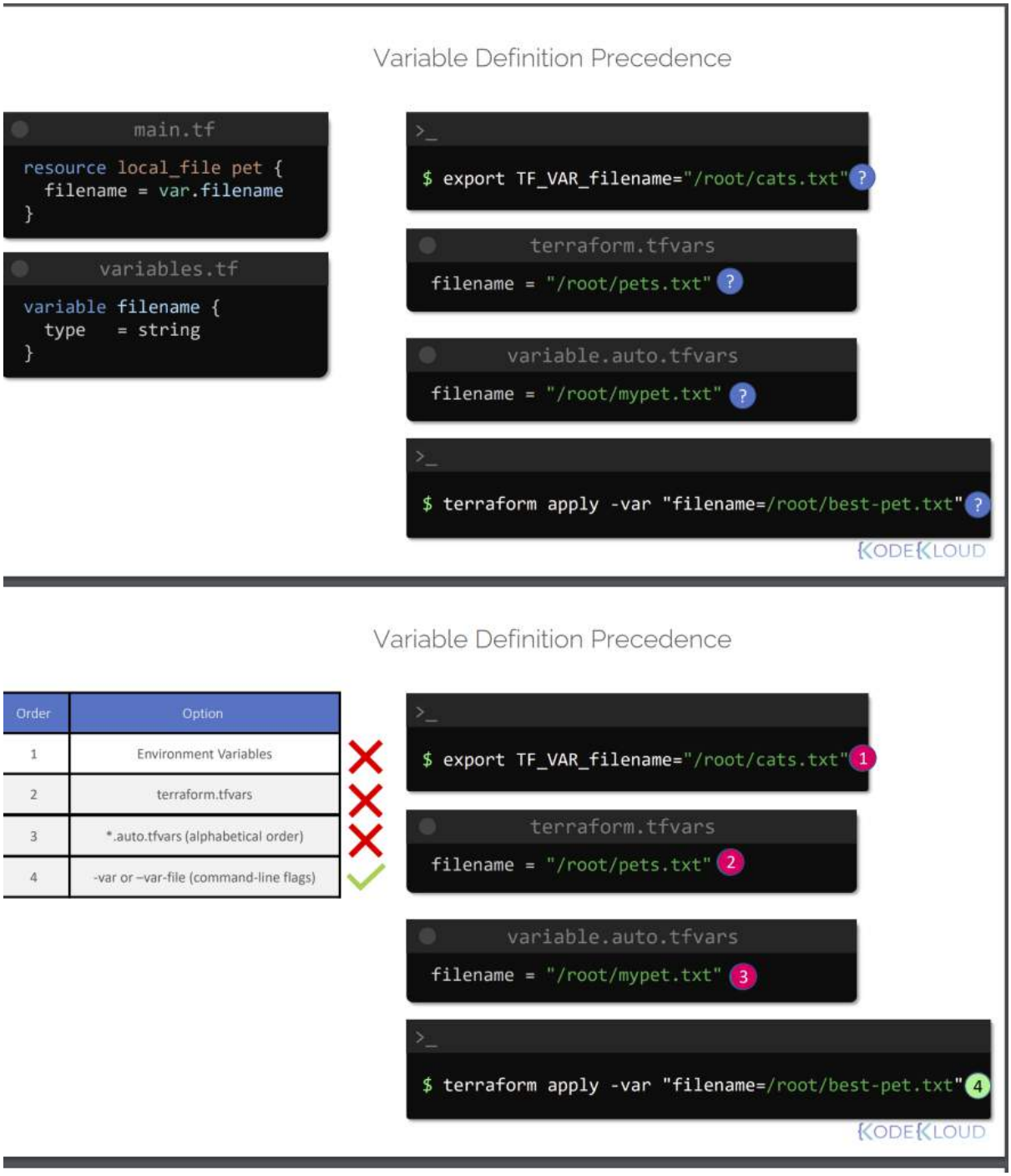
```
filename = "/root/pets.txt"
content = "We love pets!"
prefix = "Mrs"
separator = "."
length = "2"
```

```
>_

$ terraform apply -var-file variables.tfvars
```

Automatically Loaded

terraform.tfvars	terraform.tfvars.json
*.auto.tfvars	*.auto.tfvars.json

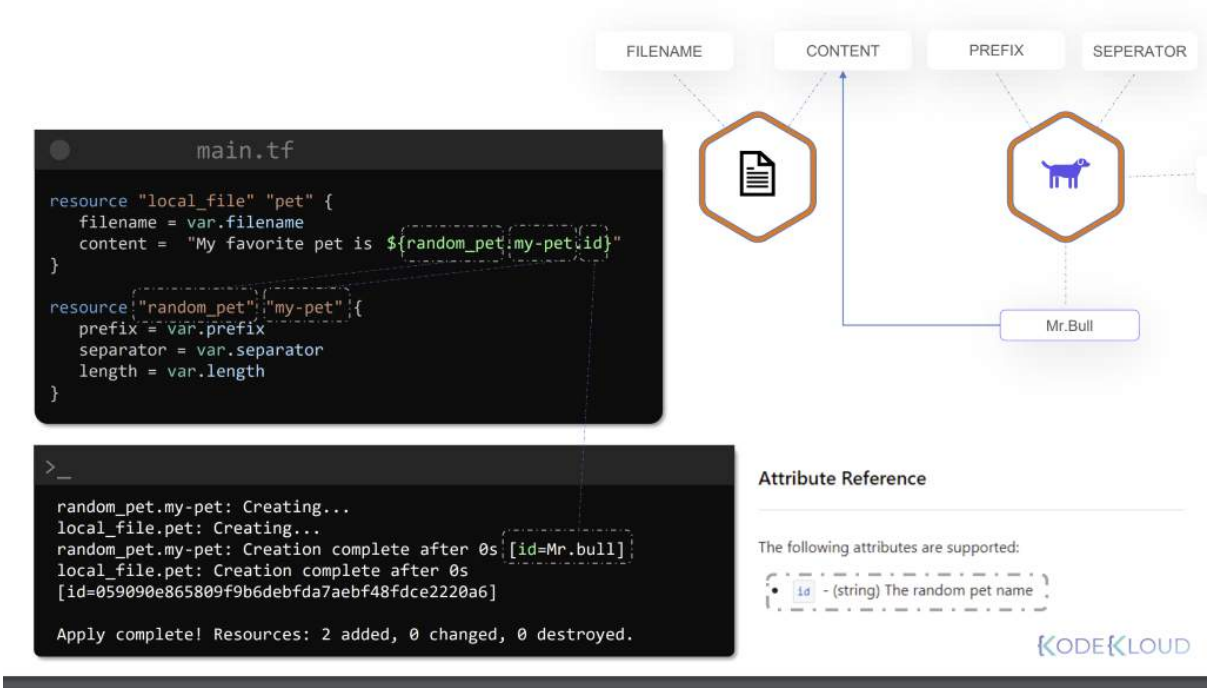
KODEKLOUD



Resource attribute

Objective : how to link two resource together by making use of resource attribute

There is no dependency between two resources but in real world it is not like that.



Requirement
We want to make of use of id in other resource as content of local_file

```
resource "local_file" "test"{
filename="C://Users//saurakes//test//pet.txt"
content="we love pets ${random_pet.my-pet.id}"
file_permission="0700"
}

resource "random_pet" "my-pet"{
prefix="Mrs"
separator="."
length="1"
}
```

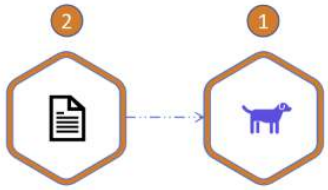
Resource dependencies

In previous lecture we saw how to link one resource to other resource using reference

Implicit Dependency

main.tf

```
resource "local_file" "pet" {
    filename = var.filename
    content =  "My favorite pet is ${random_pet.my-pet.id}"
}

resource "random_pet" "my-pet" {
    prefix = var.prefix
    separator = var.separator
    length = var.length
}
```



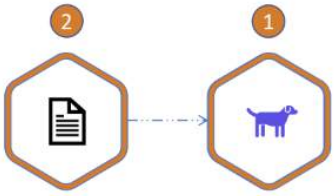
Here we are not telling which resource is depend on which . This type of dependency is called implicit

---

Explicit Dependency

main.tf

```
resource "local_file" "pet" {
    filename = var.filename
    content =  "My favorite pet is Mr.Cat"
    depends_on = [
      random_pet.my-pet
    ]

}

resource "random_pet" "my-pet" {
    prefix = var.prefix
    separator = var.separator
    length = var.length
}
```



Output variables

Along with input variables terraform also suppose output variables

Section 5: Terraform state

Introduction to terraform state

Query : how terraform know that the local file already exist

Terraform.tfstate :-- this file is terraform state file which will create after run **terraform apply** command

```
>_
$ ls terraform-local-file
main.tf variables.tf
```

main.tf

```hcl
resource "local_file" "pet" {
  filename = var.filename
  content  = var.content
}
```

variables.tf

```hcl
variable "filename" {
  default = "/root/pets.txt"
}
variable "content" {
  default = "I love pets!"
}
```


```
>_
$ cd terraform-local-file
[terraform-local-file]$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/local...
- Installing hashicorp/local v1.4.0...
- Installed hashicorp/local v1.4.0 (signed by HashiCorp)

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, we recommend adding version constraints in a
required_providers block
in your configuration, with the constraint strings
suggested below.

* hashicorp/local: version = "~> 1.4.0"

Terraform has been successfully initialized!
```

```
>_
$ ls terraform-local-file
main.tf variables.tf
```

main.tf

```hcl
resource "local_file" "pet" {
  filename = var.filename
  content  = var.content
}
```

variables.tf

```hcl
variable "filename" {
  default = "/root/pets.txt"
}
variable "content" {
  default = "I love pets!"
}
```


```
>_
[terraform-local-file]$ terraform plan

Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan,
persisted to local or remote state storage.


------------------------------------------------------------

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

Terraform will perform the following actions:

  # local_file.pet will be created
  + resource "local_file" "pet" {
      + content              = "I love pets!"
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "/root/pets.txt"
      + id                   = (known after apply)
    }

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

------------------------------------------------------------

Note: You didn't specify an "-out" parameter to save thi
```

```
>_
$ ls terraform-local-file
main.tf variables.tf
```

main.tf

```
resource "local_file" "pet" {
  filename = var.filename
  content  = var.content
}
```

variables.tf

```
variable "filename" {
  default = "/root/pets.txt"
}
variable "content" {
  default = "I love pets!"
}
```


```
>_
[terraform-local-file]$ terraform apply
An execution plan has been generated and is shown below
Resource actions are indicated with the following symbo
  + create

Terraform will perform the following actions:

  # local_file.pet will be created
  + resource "local_file" "pet" {
      + content              = "I love pets!"
      + directory_permission = "0777"
      + file_permission      = "0777"
      + filename             = "/root/pets.txt"
      + id                   = (known after apply)
    }

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.

  Enter a value: yes

local_file.pet: Creating...
local_file.pet: Creation complete after 0s
[id=7e4db4fbfdbb108bdd04692602bae3e9bd1e1b68]
```

```
>_
[terraform-local-file]$ cat /root/pets
I love pets!
```


```
>_
[terraform-local-file]$ terraform apply
local_file.pet: Refreshing state...
[id=7e4db4fbfdbb108bdd04692602bae3e9bd1e1b68]

Apply complete! Resources: 0 added, 0 changed, 0 destro
```

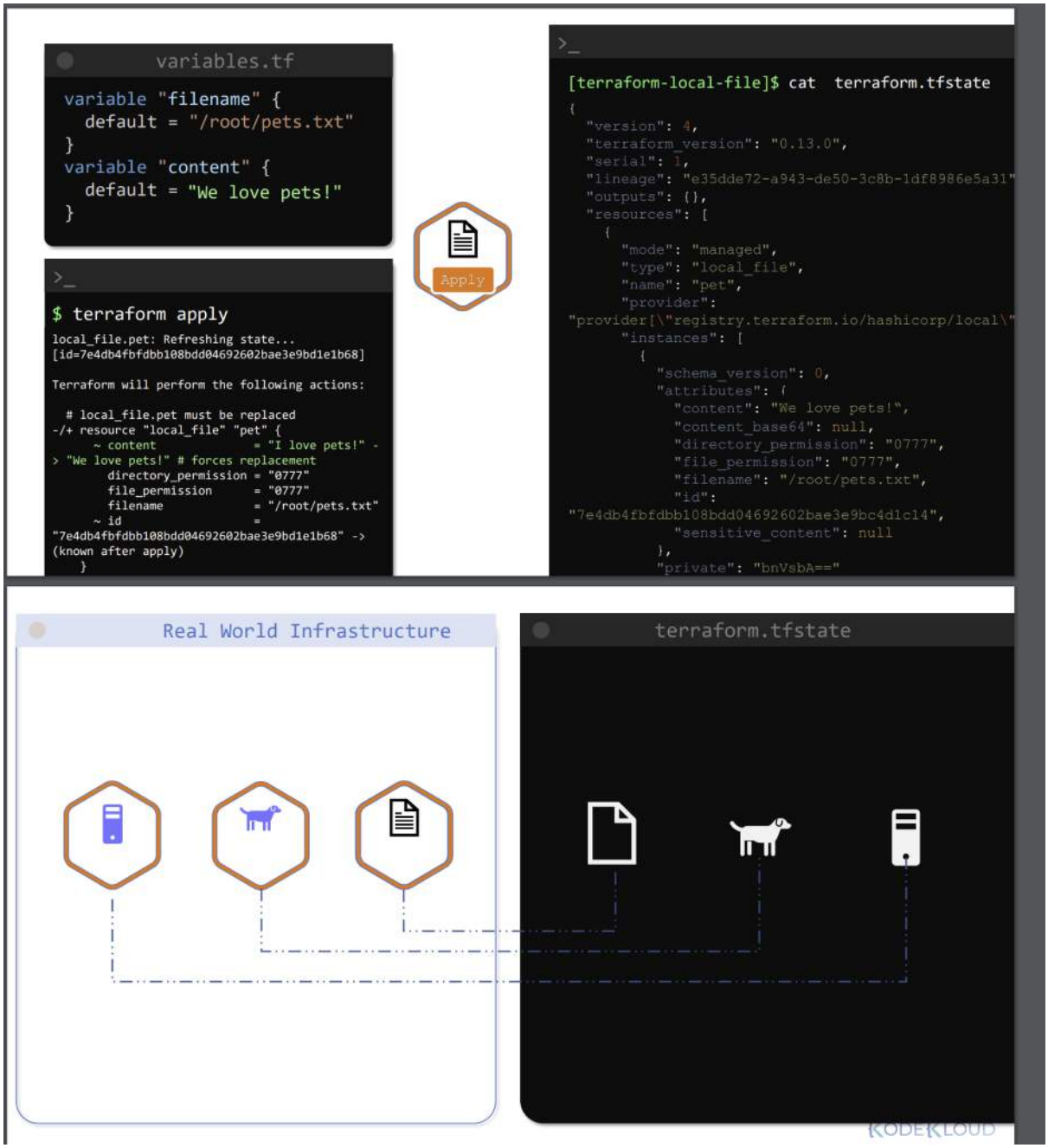
```
>_
[terraform-local-file]$ ls
main.tf variables.tf terraform.tfstate
```


```
>_
[terraform-local-file]$ cat  terraform.tfstate
{
  "version": 4,
  "terraform_version": "0.13.0",
  "serial": 1,
  "lineage": "e35dde72-a943-de50-3c8b-1df8986e5a31"
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "local_file",
      "name": "pet",
      "provider":
"provider[\"registry.terraform.io/hashicorp/local\"
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "content": "I love pets!",
            "content_base64": null,
            "directory_permission": "0777",
            "file_permission": "0777",
            "filename": "/root/pets.txt",
            "id":
"7e4db4fbfdbb108bdd04692602bae3e9bd1e1b68",
            "sensitive_content": null
          },
          "private": "bnVsbA=="
```

```
variables.tf
variable "filename" {
  default = "/root/pets.txt"
}
variable "content" {
  default = "We love pets!"
}
```

```
>_
$ terraform plan
Refreshing Terraform state in-memory
prior to plan...
The refreshed state will be used to
calculate this plan, but will not be
persisted to local or remote state
storage.

local_file.pet: Refreshing state...
[id=7e4db4fbfdbb108bdd04692602bae3e9bd1e
1b68]
.
.
.
[Output Truncated ]
```


```
>_
[terraform-local-file]$ cat  terraform.tfstate
{
  "version": 4,
  "terraform_version": "0.13.0",
  "serial": 1,
  "lineage": "e35dde72-a943-de50-3c8b-1df8986e5a31"
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "local_file",
      "name": "pet",
      "provider":
"provider[\"registry.terraform.io/hashicorp/local\"
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "content": "I love pets!",
            "content_base64": null,
            "directory_permission": "0777",
            "file_permission": "0777",
            "filename": "/root/pets.txt",
            "id":
"7e4db4fbfdbb108bdd04692602bae3e9bd1e1b68",
            "sensitive_content": null
          },
          "private": "bnVsbA=="
```



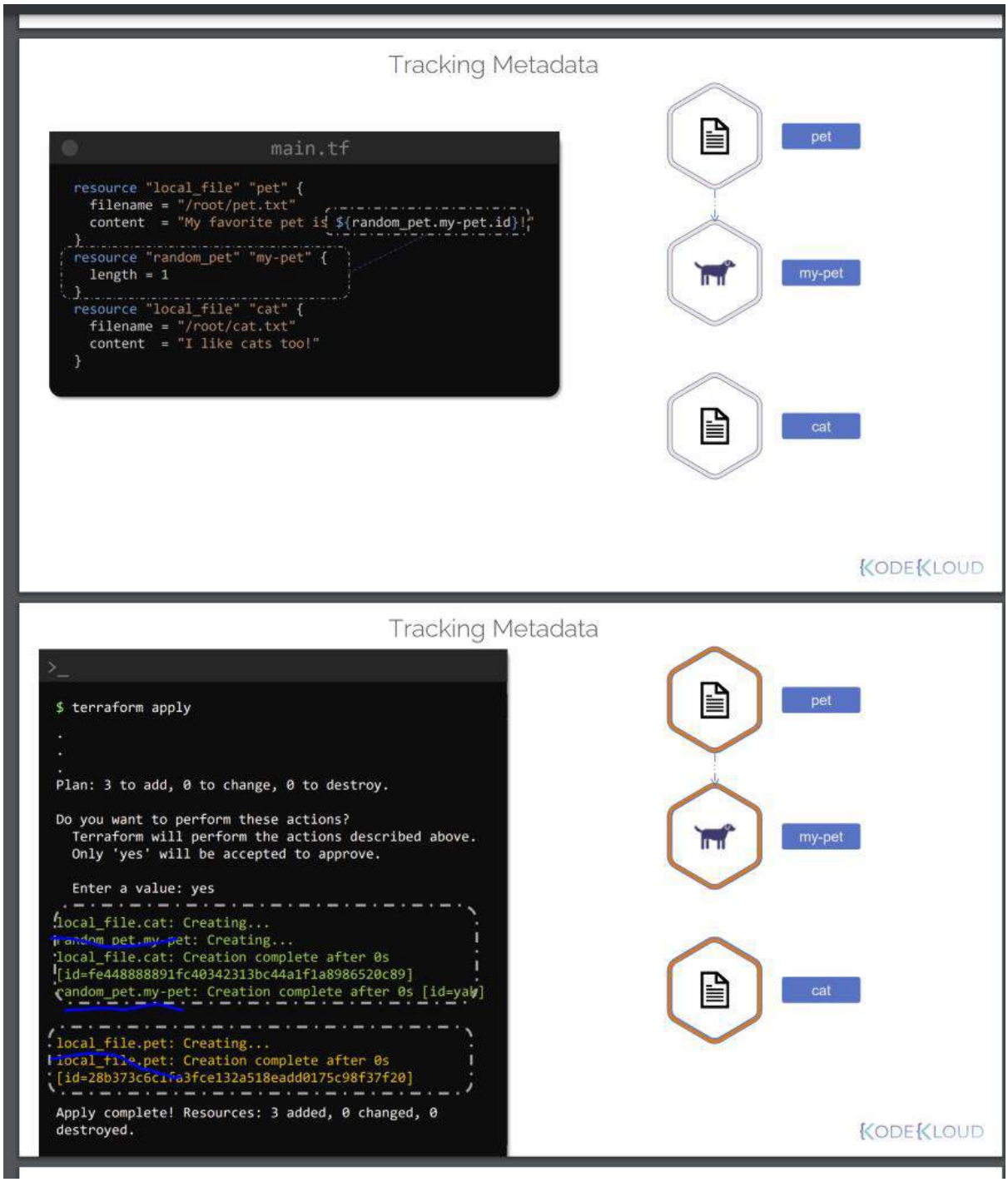
Purpose of state

It map resource configuration to the real world infrastructure

State file can considered as for all the resources that are manages in the real world
When terraform create a resources it record the identity in the state file.
Each resource created by the terraform will have **unique id**

Note:

Resource one depend on resource 2 and we run apply command now we remove resource from configuration file and again run apply command the how terraform will know which resource we need to remove first that will came to know by seeing state file



Tracking Metadata

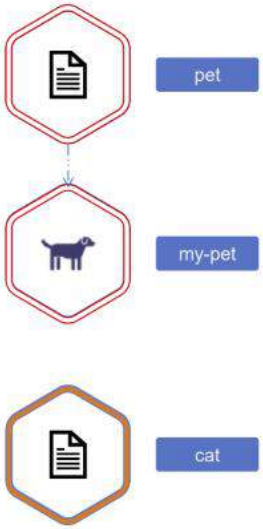
main.tf

```
resource "local_file" "pet" {
  filename = "/root/pet.txt"
  content  = "My favorite pet is ${random_pet.my-pet.id}!"
}
resource "random_pet" "my-pet" {
  length = 1
}
resource "local_file" "cat" {
  filename = "/root/cat.txt"
  content  = "I like cats too!"
}
```

Tracking Metadata

main.tf

```
resource "local_file" "cat" {
  filename = "/root/cat.txt"
  content  = "I like cats too!"
}
```

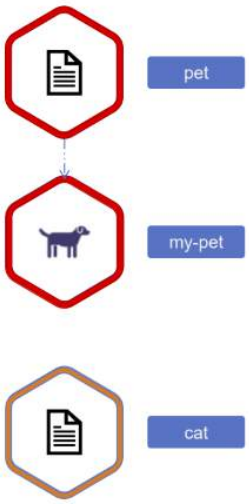


Tracking Metadata

main.tf

```
resource "local_file" "cat" {
  filename = "/root/cat.txt"
  content  = "I like cats too!"
}
```

```
>_
$ cat terraform.tfstate
{
      "mode": "managed",
      "type": "local_file",
      "name": "pet",
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "content": "My favorite pet is yak!",
          },
          "private": "bnVsbA==",
          "dependencies": [
            "random_pet.my-pet"
          ]
```



KODEKLOUD

Performance

terraform.tfstate

KODEKLOUD

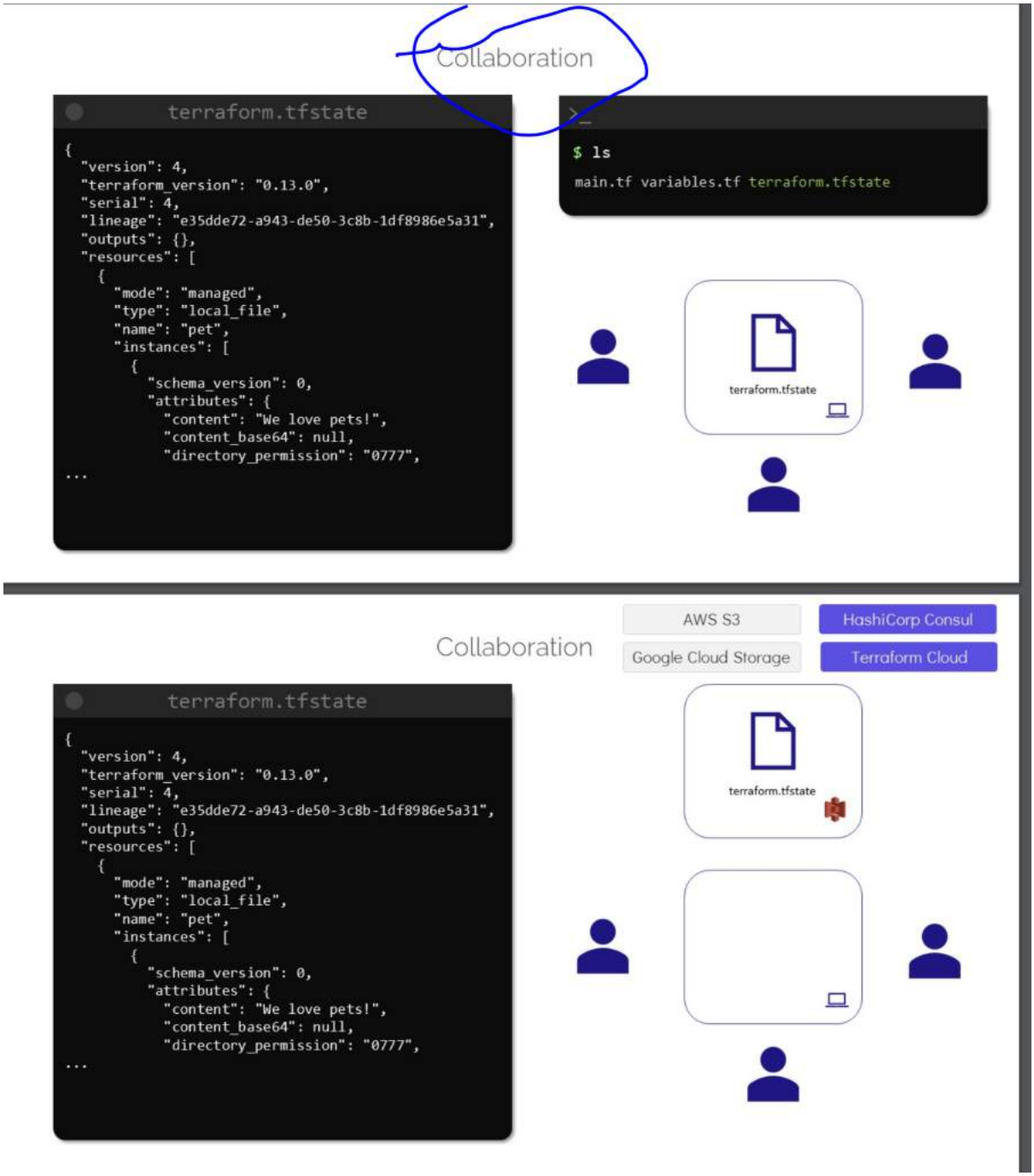
Performance

terraform.tfstate

```
{
  "version": 4,
  "terraform_version": "0.13.0",
  "serial": 4,
  "lineage": "e35dde72-a943-de50-3c8b-1df8986e5a31",
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "local_file",
      "name": "pet",
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "content": "We love pets!",
            "content_base64": null,
            "directory_permission": "0777",
...
```

>_

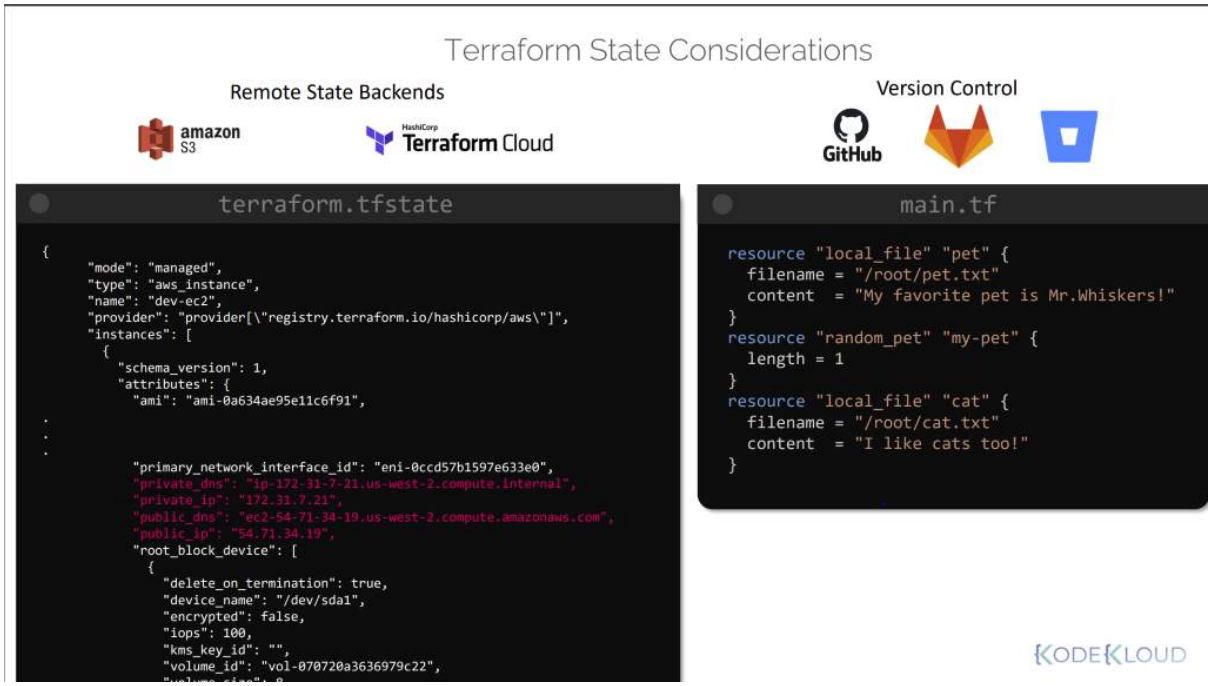
```
$ terraform plan --refresh=false

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

Terraform will perform the following actions:

  # local_file.cat must be replaced
-/+ resource "local_file" "pet" {
      ~ content              = "I like cats too!" ->
"Dogs are awesome!" # forces replacement
        directory_permission = "0777"
        file_permission      = "0777"
        filename             = "/root/pets.txt"
      ~ id                   =
"cba595b7d9f94ba1107a46f3f731912d95fb3d2c" -> (known
after apply)
    }

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

------------------------------------------------------
```



Terraform state consideration

State file contain sensitive information within it contain every little detail of infrastructure.

The state file should always be stored in secure storage.



Section 6: working with terraform

Terraform commands

Terraform validate : if everything is correct with file we would see success

Terraform fmt

terraform fmt

main.tf

```
resource "local_file" "pet" {
   filename        = "/root/pets.txt"
   content         = "We love pets!"
   file_permission = "0700"
}
```

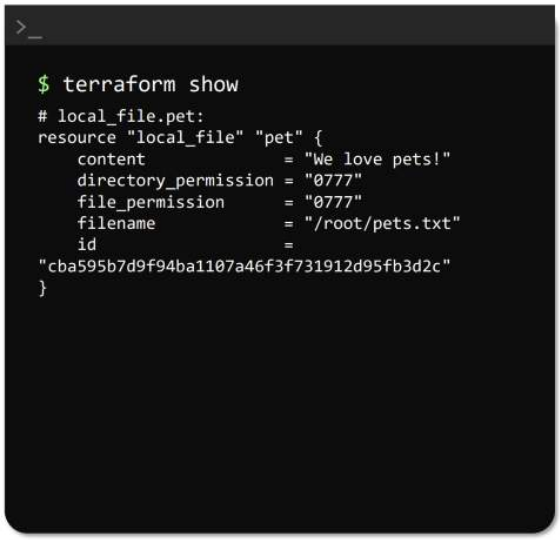
```
>_

$ terraform fmt
main.tf
```

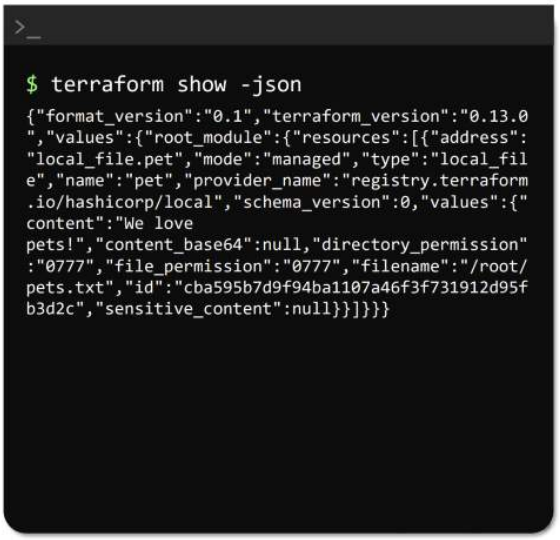
Terraform show :

Print out current state of infrastructure

```
C:\Users\saurakes\OneDrive - Cisco\Desktop\teraform>terraform show
# local_file.test:
resource "local_file" "test" {
    content              = "we love pets Mrs.buck"
    directory_permission = "0777"
    file_permission      = "0700"
    filename             = "C://Users//saurakes//test//pet.txt"
    id                   = "5708acf2660274cab0f1827220dfd2fc9d0e6e46"
}

# random_pet.my-pet:
resource "random_pet" "my-pet" {
    id        = "Mrs.buck"
    length    = 1
    prefix    = "Mrs"
    separator = "."
}
```

terraform show

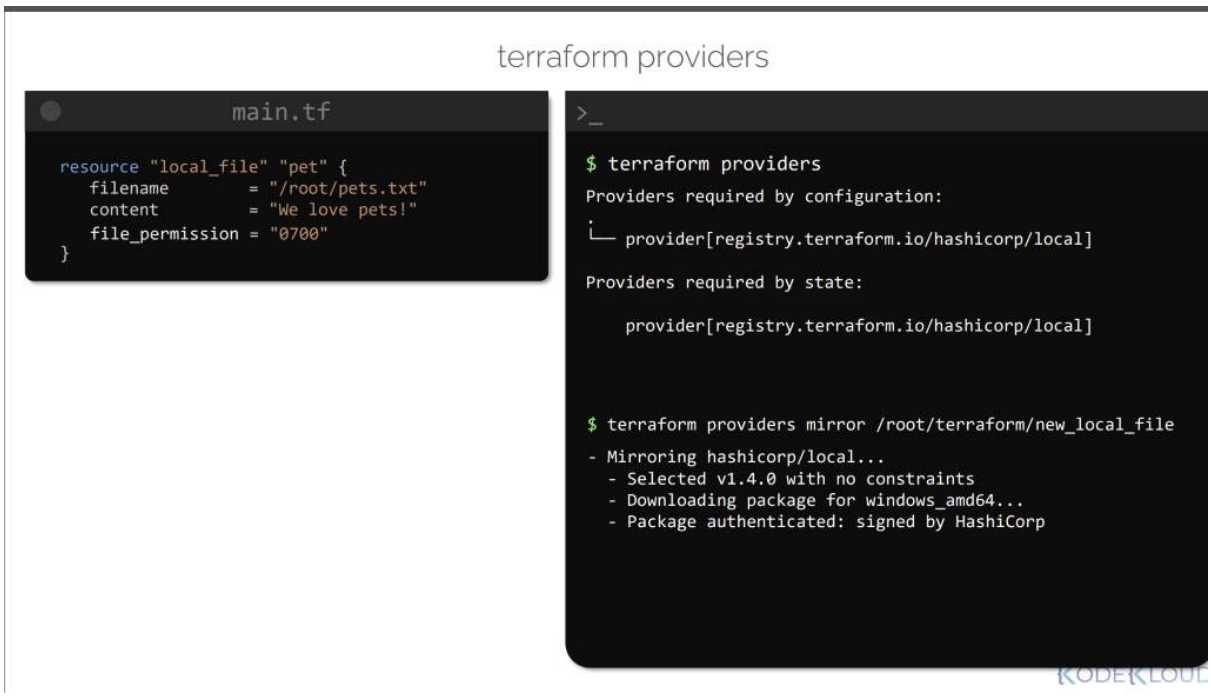




Terraform providers

To see list of all providers use in the configuration directory

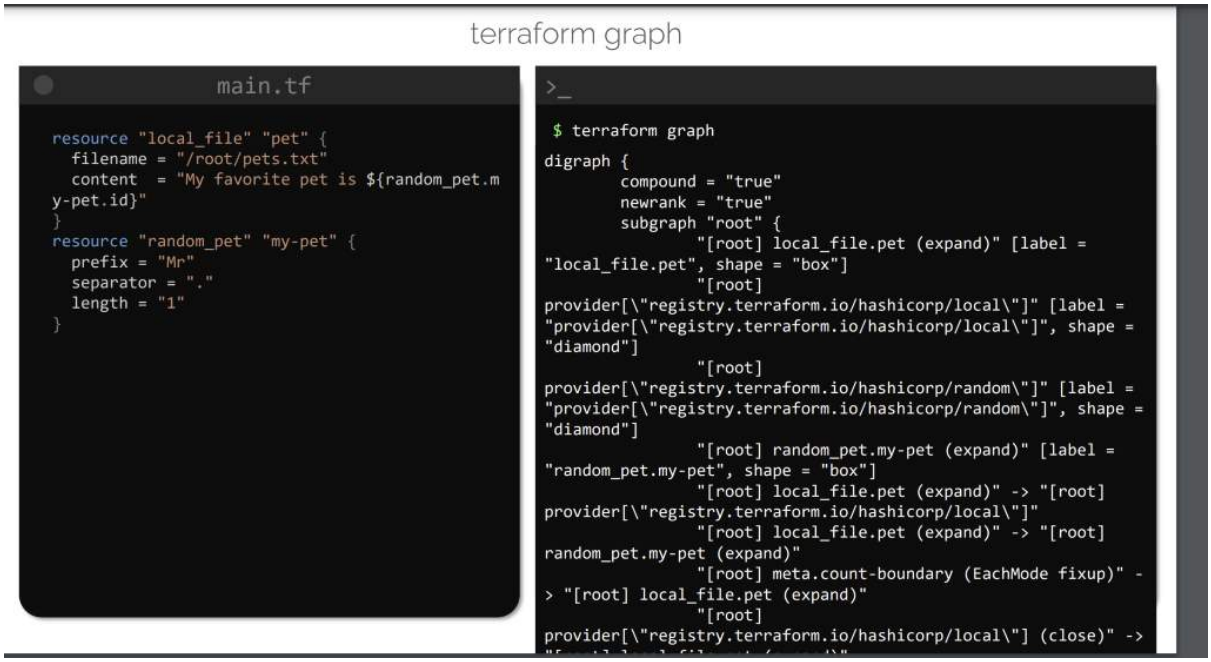
```
C:\Users\saurakes\OneDrive - Cisco\Desktop\teraform>terraform providers

Providers required by configuration:
.
├── provider[registry.terraform.io/hashicorp/local]
└── provider[registry.terraform.io/hashicorp/random]

Providers required by state:

    provider[registry.terraform.io/hashicorp/local]

    provider[registry.terraform.io/hashicorp/random]
```

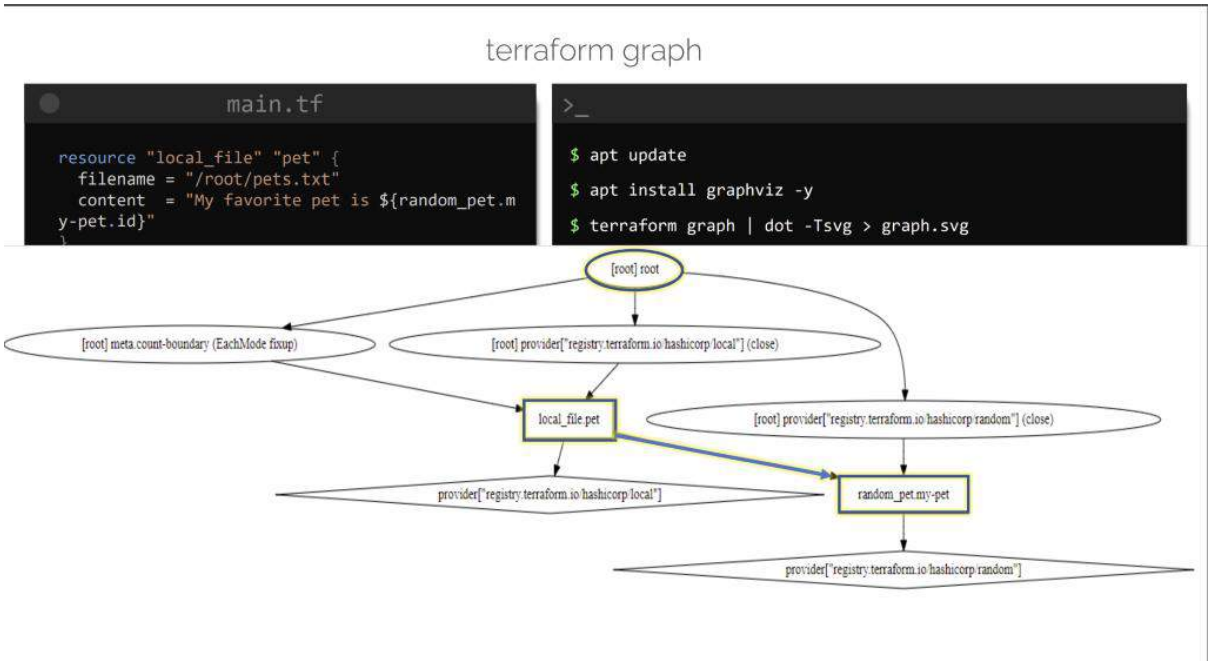


Terraform output

Terraform graph :

Will show visual representation of dependency





What is a provider in Terraform?

A provider in Terraform is **a plugin that enables interaction with an API**. This includes Cloud providers and Software-as-a-service providers. The providers are specified in the Terraform configuration code. They tell Terraform which services it needs to interact with.

Mutable vs Immutable infrastructure

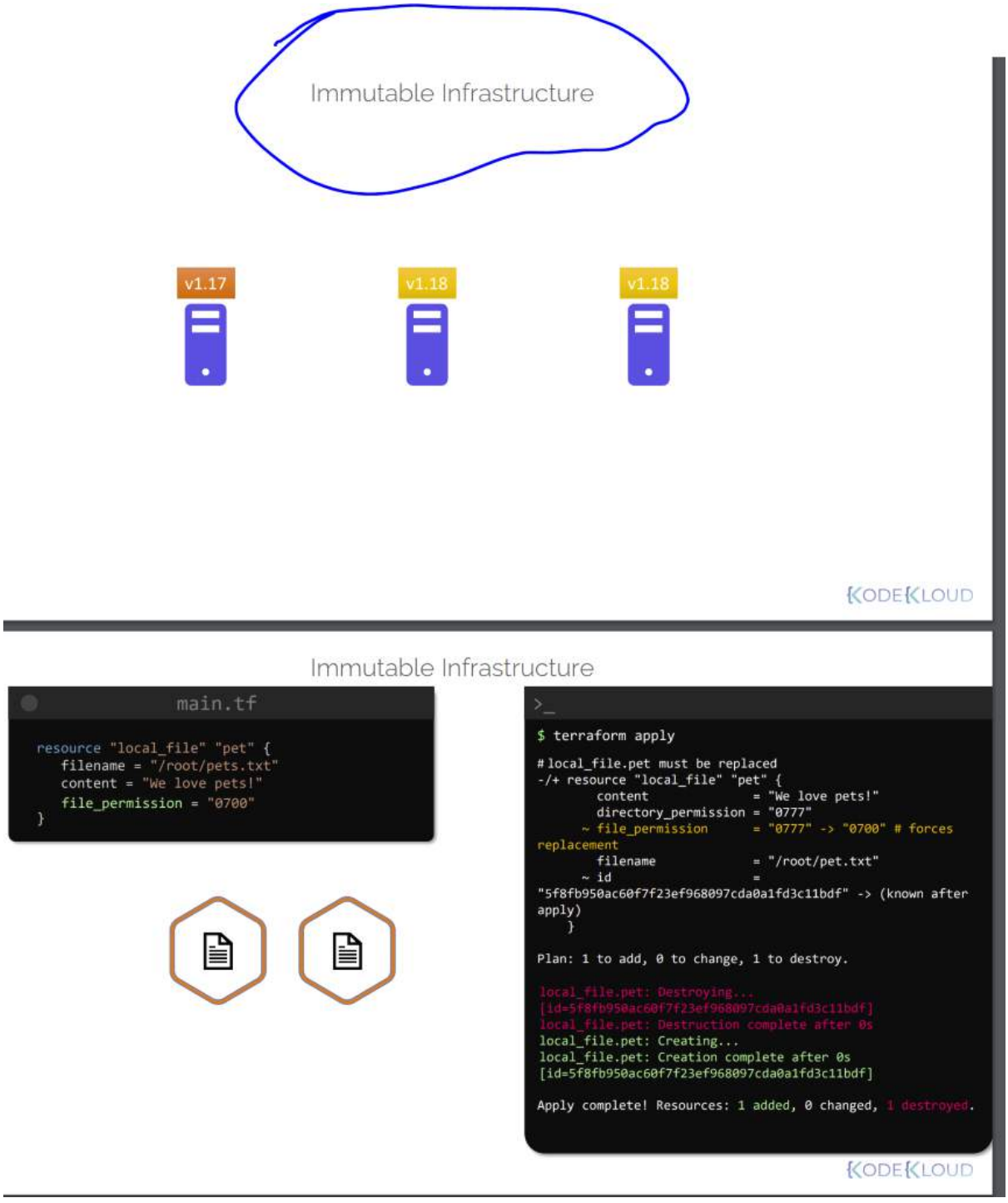
Mutable infrastructures allow for regular updates and modifications after the software has been deployed, whereas i**mmutable infrastructures** do not allow modifications once the software has been deployed.

What is configuration drift in terraform?
Drift Detection for Terraform Cloud **provides continuous checks against infrastructure state to detect and notify operators of changes in your infrastructure to minimize your risk exposure, application downtime, and costs**

Immutable :
make easy role back and role forward between version

By default terraform destroy the resource first before creating a new one and placed

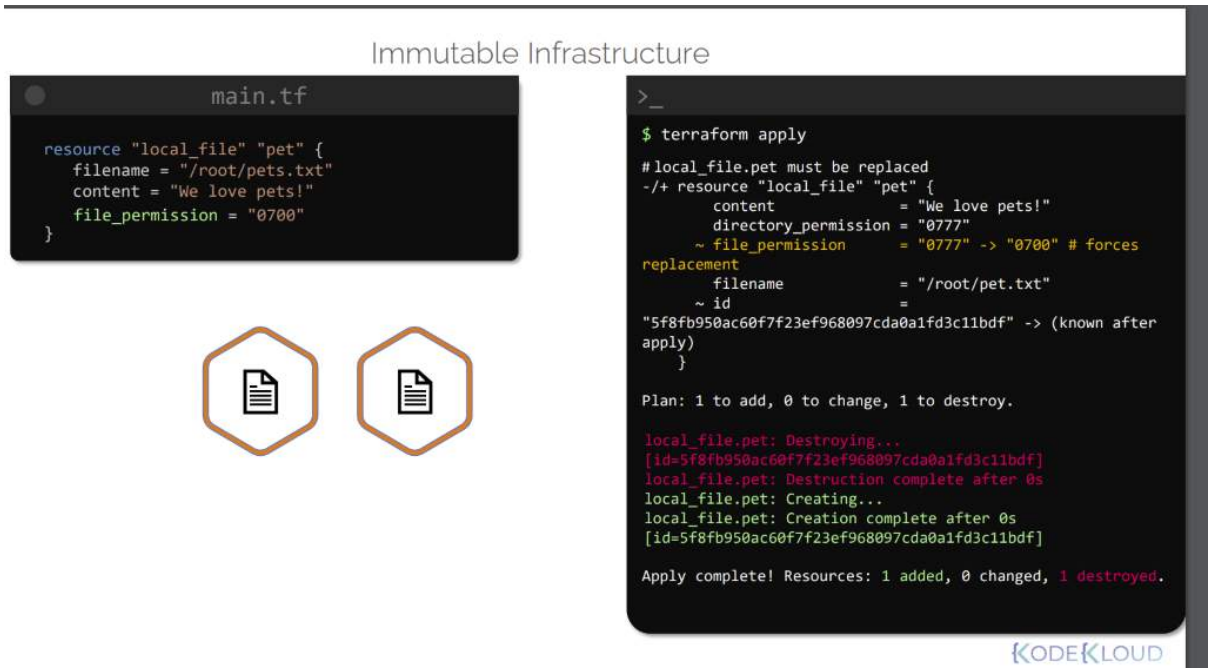
But what if we want to create resource first before deleting this can be achieved by life cycles rules



An immutable infrastructure is **another infrastructure paradigm in which servers are never modified after they're deployed**. If something needs to be updated, fixed, or modified in any way, new servers built from a common image with the appropriate changes are provisioned to replace the old ones.

LifeCycle rules

Immutable: terraform delete the resource first before creating a new one with updated configuration



Older file destroyed first. This may be desirable approach in all cases.
And sometimes we want updated version to be created first before the older one is deleted or we do not want resource to be deleted at all.

This can be achieved in terraform by making use of lifecycle rules

Create_before_destroy :- a new resource will be created first

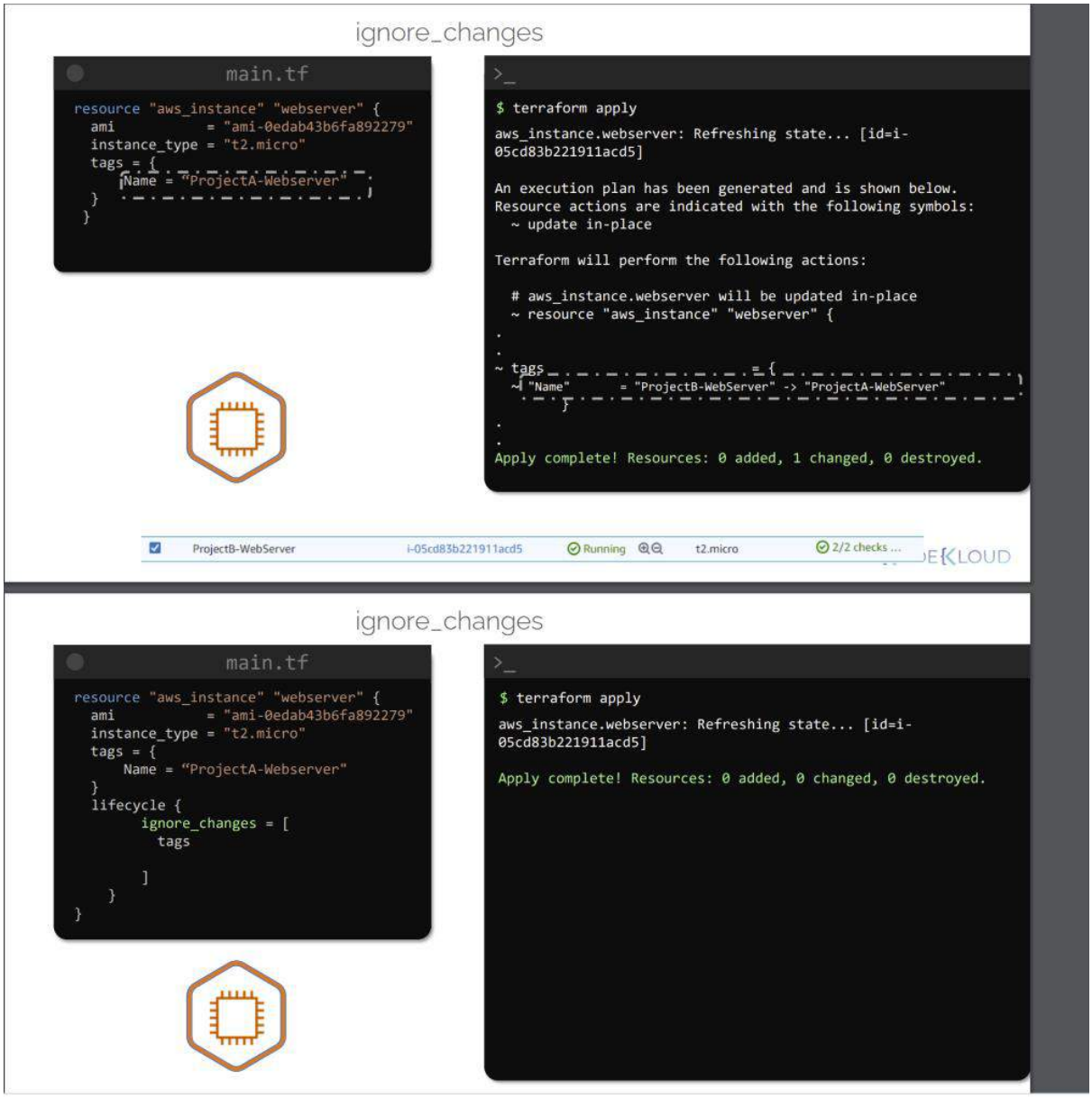
There could be cases where we do not want to destroy resources for any reason

This can be achieved by use o prevent_destroy but we can still destroy it by use of terraform destroy

This will especially use to resource accidently deleted

Ignore_changes:

This lifecycle rule when apply to prevent the resource from being updated.
If we do no change in attribute of resource will use ignore_changes



Order	Option	
1	create_before_destroy	Create the resource first and then destroy older
2	prevent_destroy	Prevents destroy of a resource
3	ignore_changes	Ignore Changes to Resource Attributes (specific/all)

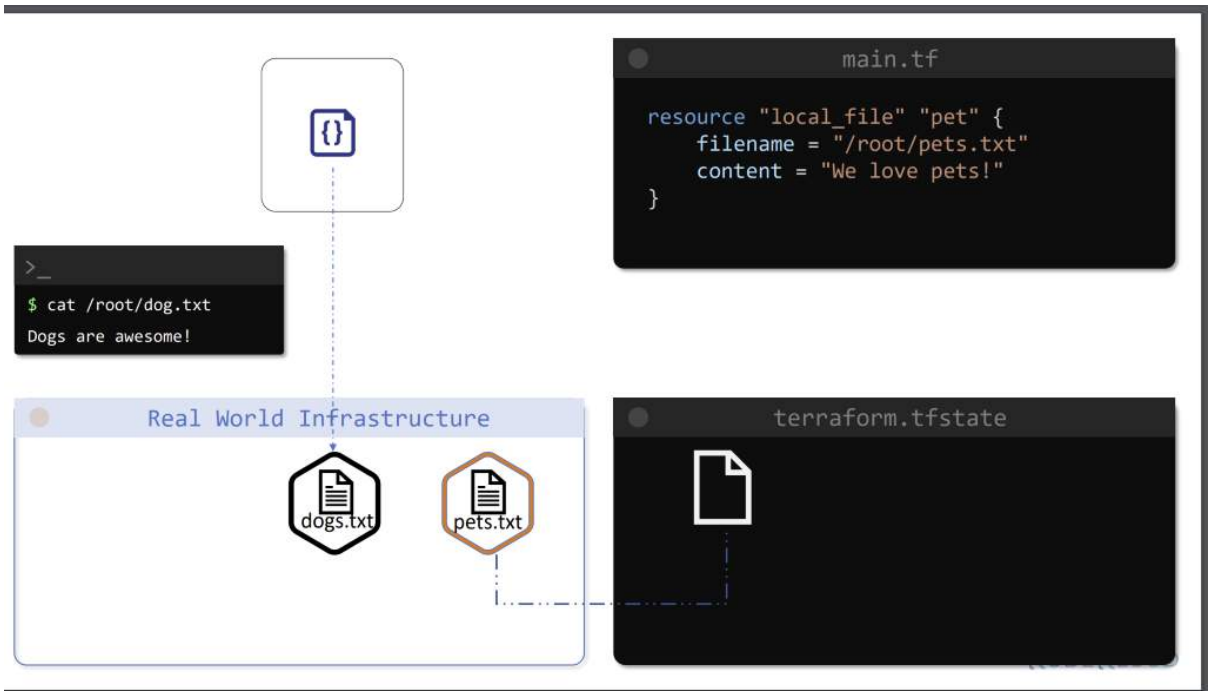
Datasource

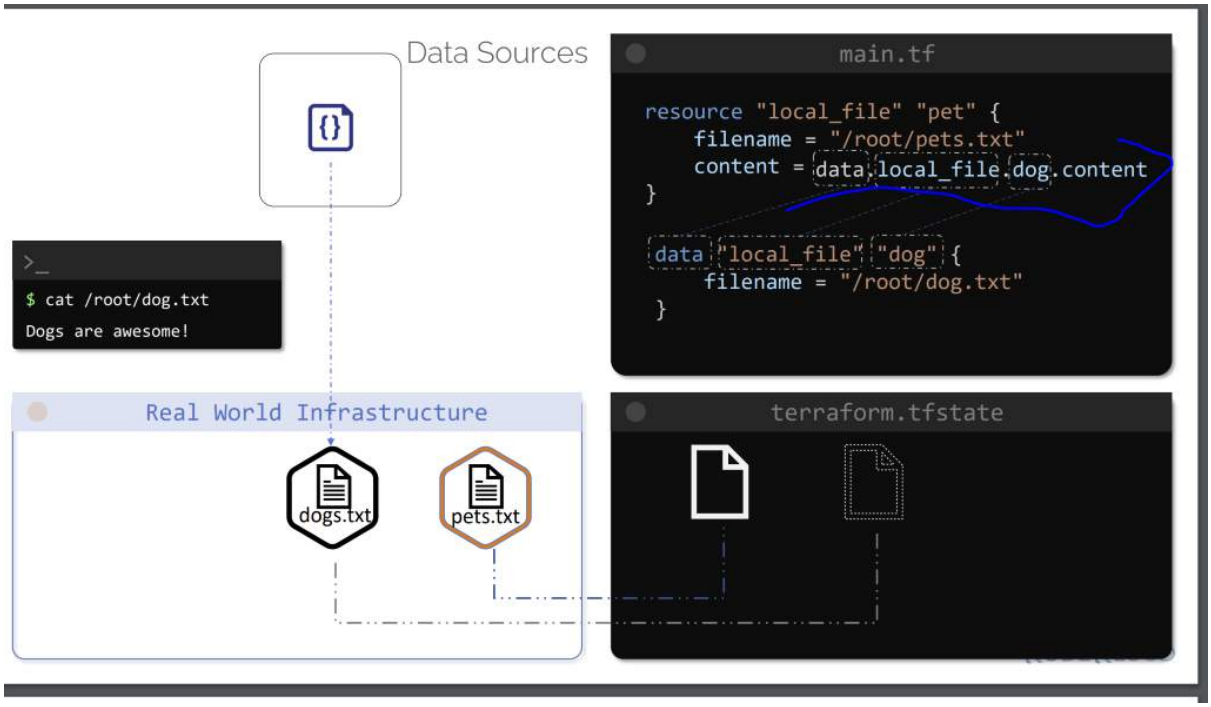
Will look datasources in terraform.

We know that terraform make use of configuration files along with the state file to provision infrastructure resources

Example :
pets.txt created by terraform and dog.txt created by bash.
But if we want to use the content of dog.txt in pet.txt then we use datasource

Data will be available in data source





Resource	Data Source
Keyword: **resource**	Keyword: **data**
Creates, Updates, Destroys Infrastructure	Only **Reads** Infrastructure
Also called **Managed Resources**	Also called **Data Resources**

Meta Arguments

Requirement: create multiple instance of same resource

This can be achieved by meta argument of terraforms.

main.tf

```
resource "local_file" "pet" {
   filename = var.filename
   content = var.content

}
```

variables.tf

```
variable "filename" {
  default = "/root/pets.txt"
}
variable "content" {
  default = "I love pets!"
}
```

KODEKLOUD

Shell Scripts

create_files.sh

```
#!/bin/bash

for i in {1..3}
  do
    touch /root/pet${i}
  done
```

>_

```
$ ls -ltr /root/
-rw-r--r-- 1 root root    0 Sep  9 02:04 pet2
-rw-r--r-- 1 root root    0 Sep  9 02:04 pet1
-rw-r--r-- 1 root root    0 Sep  9 02:04 pet3
```

Meta Arguments

depends_on

main.tf

```
resource "local_file" "pet" {
  filename = var.filename
  content  = var.content
  depends_on = [
    random_pet.my-pet
  ]
}
resource "random_pet" "my-pet" {
  prefix    = var.prefix
  separator = var.separator
  length    = var.length
}
```

lifecycle

main.tf

```
resource "local_file" "pet" {
   filename = "/root/pets.txt"
   content = "We love pets!"
   file_permission = "0700
   lifecycle  {
     create_before_destroy  = true
   }
}
```

Count

Count meta argument : to create multiple instance of local file

Problem in this approach : file name is not unique

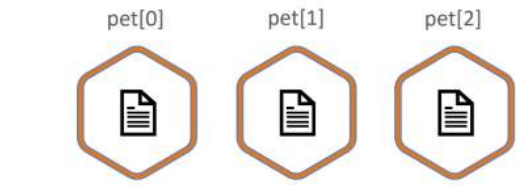
Length Function

main.tf

```
resource "local_file" "pet" {
    filename = var.filename[count.index]
    count    = length(var.filename)
}
```

variables.tf

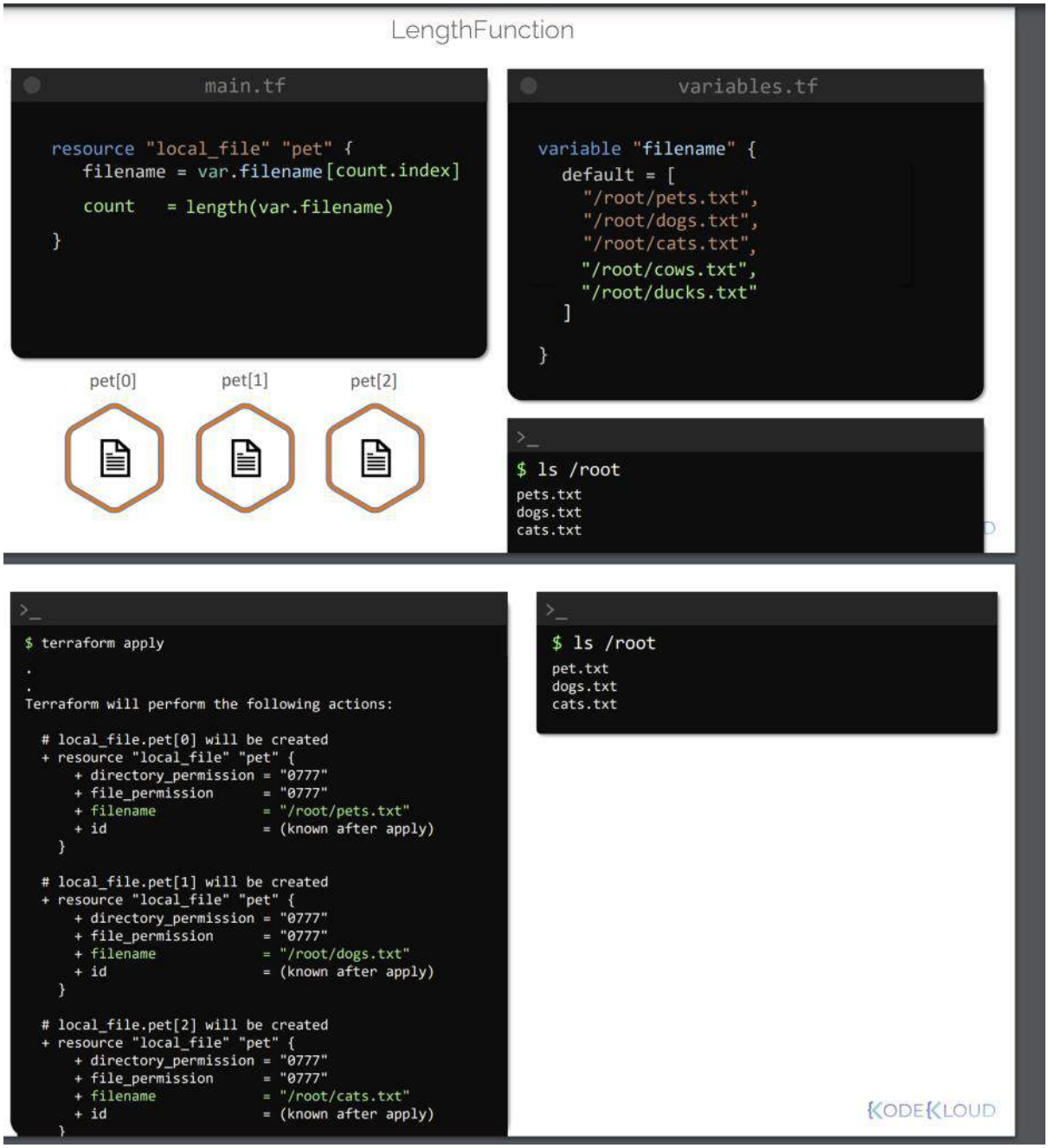
```
variable "filename" {
  default = [
    "/root/pets.txt",
    "/root/dogs.txt",
    "/root/cats.txt",
    "/root/cows.txt",
    "/root/ducks.txt"
  ]
}
```




```
>_
$ ls /root
pets.txt
dogs.txt
cats.txt
```

Length Function

variable	function	value
fruits = ["apple", "banana", "orange"]	length(fruits)	3
cars = [“honda”, “bmw”, “nissan”, “kia”]	length(cars)	4
colors = [“red”, “purple”]	length(colors)	2



Note : if we delete first element then second and third one will destroy and replace but we don't want this if we delete one then other also need to be destroy and replace

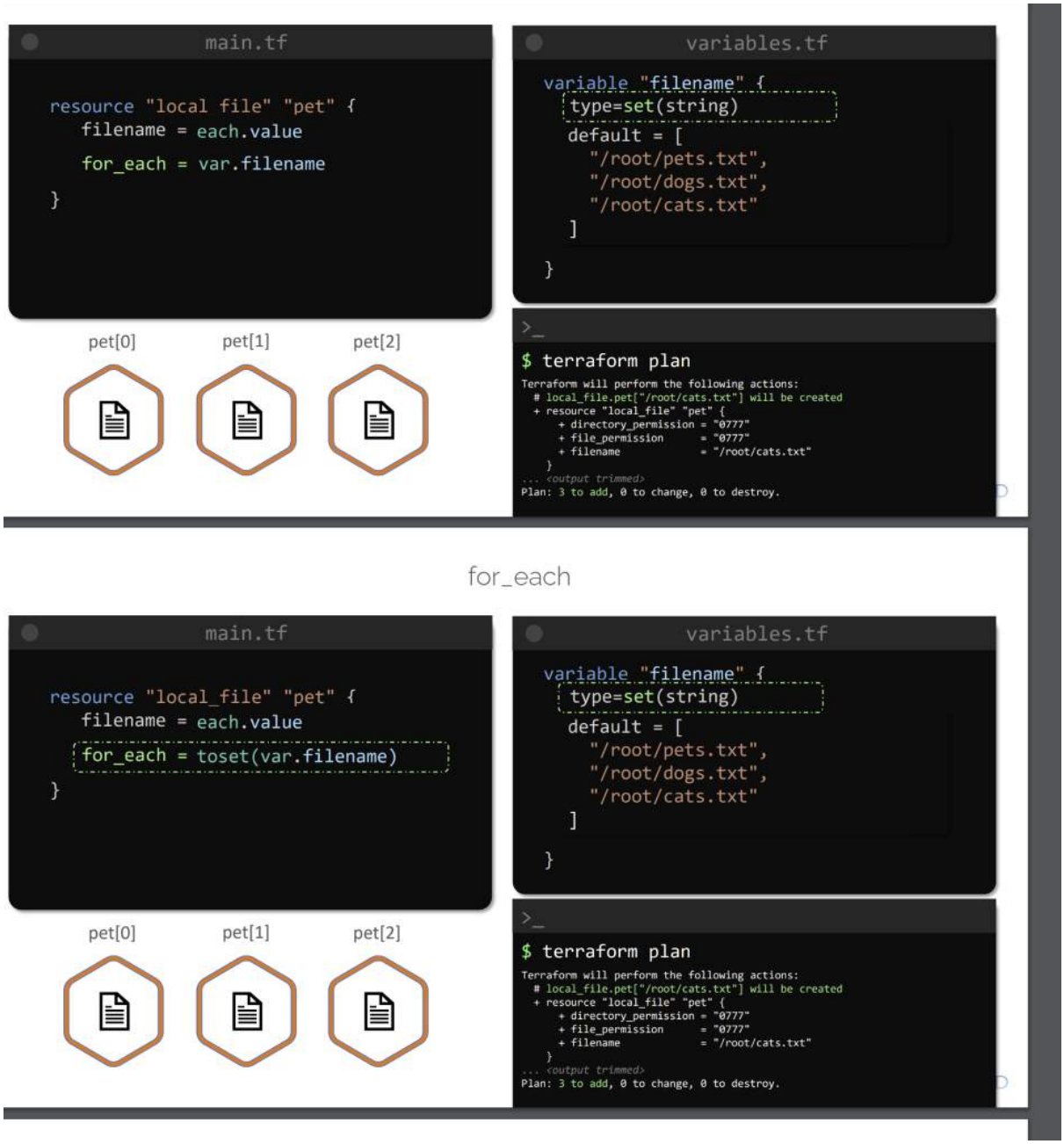
Will fix this

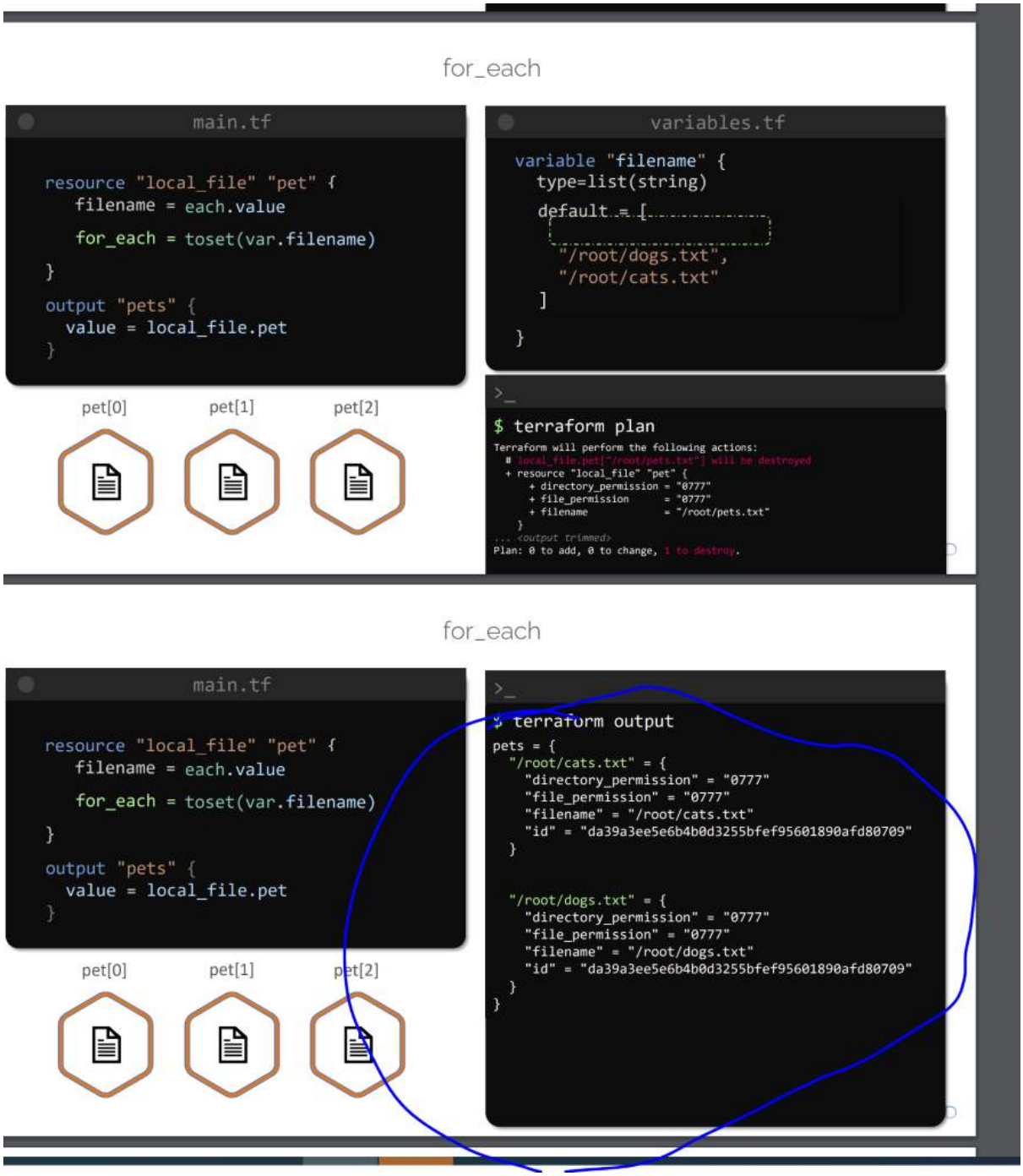
For each

Note :
For each argument only work with map or set

ToSet :-- will convert variable from a list to set

Now the resources will stored as map not list





Count option will create as list and for each will create as map.
So it will destroy only first element.

count

```
$ terraform output
pets = [
  {
    "directory_permission" = "0777"
    "file_permission" = "0777"
    "filename" = "/root/pets.txt"
    "id" = "da39a3ee5e6b4b0d3255bfef95601890afd80709"
  },
  {
    "directory_permission" = "0777"
    "file_permission" = "0777"
    "filename" = "/root/dogs.txt"
    "id" = "da39a3ee5e6b4b0d3255bfef95601890afd80709"
  },
  {
    "directory_permission" = "0777"
    "file_permission" = "0777"
    "filename" = "/root/cats.txt"
    "id" = "da39a3ee5e6b4b0d3255bfef95601890afd80709"
  },
]
```

for_each

```
$ terraform output
pets = {
  "/root/cats.txt" = {
    "directory_permission" = "0777"
    "file_permission" = "0777"
    "filename" = "/root/cats.txt"
    "id" = "da39a3ee5e6b4b0d3255bfef95601890afd80709"
  }


  "/root/dogs.txt" = {
    "directory_permission" = "0777"
    "file_permission" = "0777"
    "filename" = "/root/dogs.txt"
    "id" = "da39a3ee5e6b4b0d3255bfef95601890afd80709"
  }
}
```

Version Constraints

Provider use a plugin based architecture and most of the popular one are available in public terraform registry

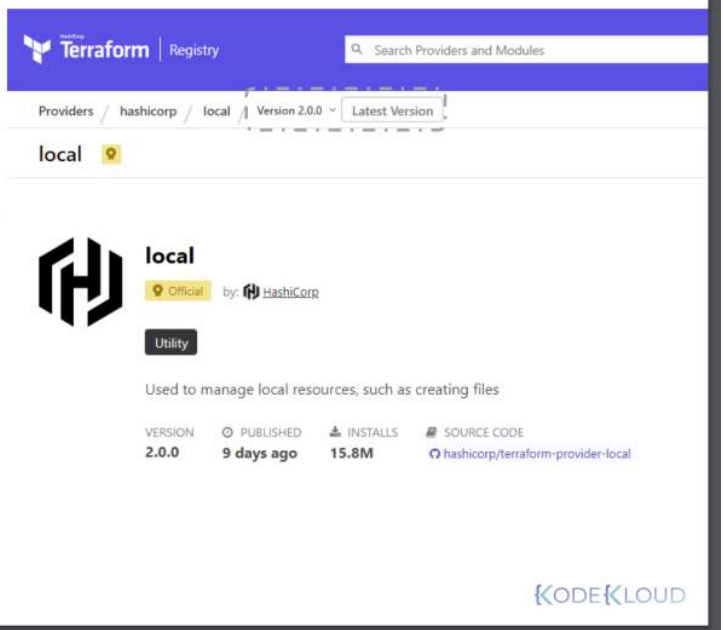
Without additional configuration terraform init command download latest version of provider plugin needed by the configuration file.

However this is not needed everytime . The functionality of provider plugin changes drastically from one version to another.
Or terraform configuration file may not work as expected when we uses different version

How to use specific version of provider

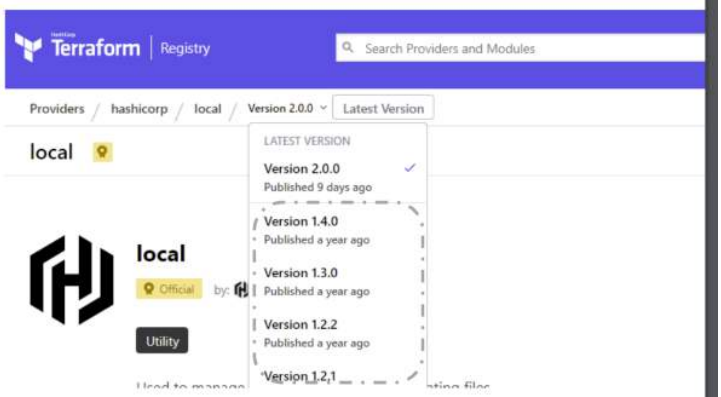
main.tf

```
resource "local_file" "pet" {
  filename     = "/root/pet.txt"
  content  = "We love pets!"
}
```



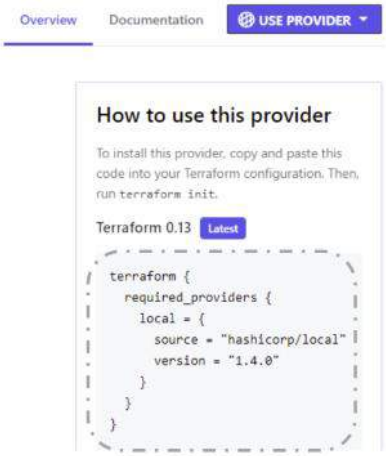
main.tf

```
resource "local_file" "pet" {
  filename     = "/root/pet.txt"
  content  = "We love pets!"
}
```



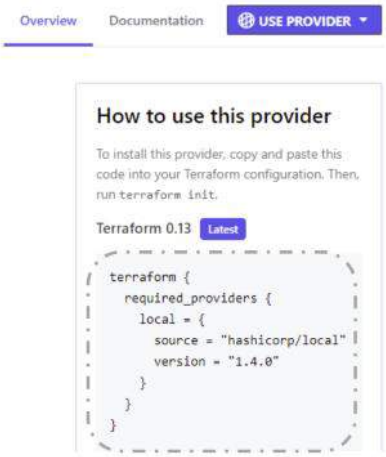
main.tf

```
resource "local_file" "pet" {
  filename     = "/root/pet.txt"
  content  = "We love pets!"
}
```



main.tf

```
terraform {
  required_providers {
    local = {
      source = "hashicorp/local"
      version = "1.4.0"
    }
  }
}

resource "local_file" "pet" {
  filename     = "/root/pet.txt"
  content  = "We love pets!"
}
```

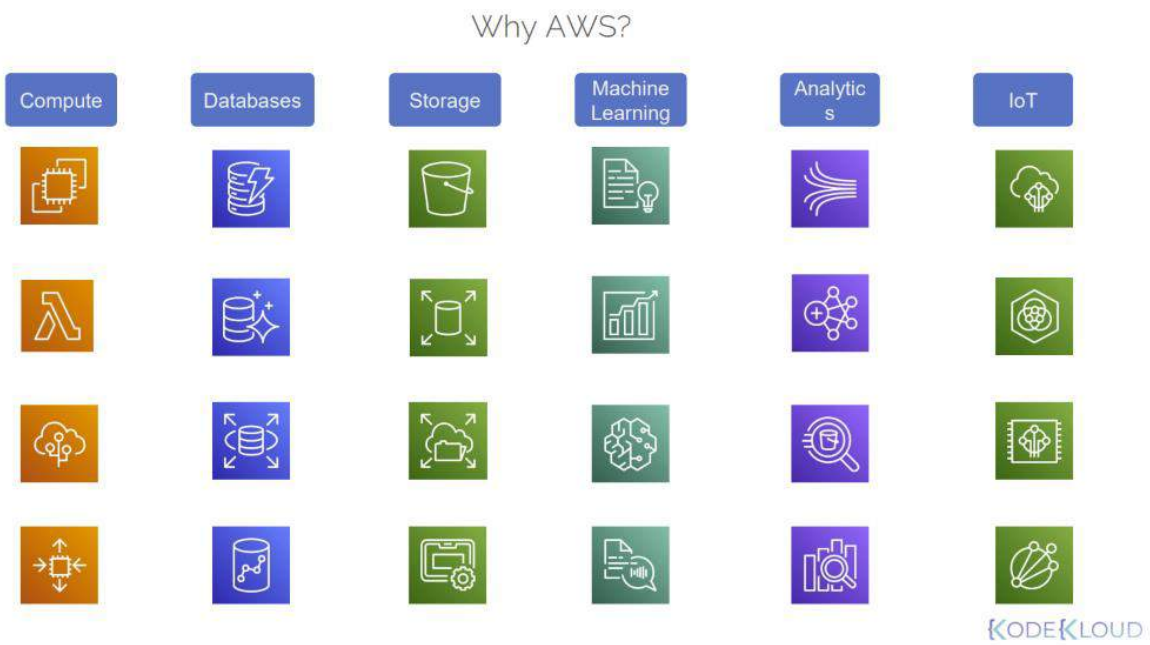


Terraform will download either 1.2 or incremental version such as 1.3 , 1.4 etc

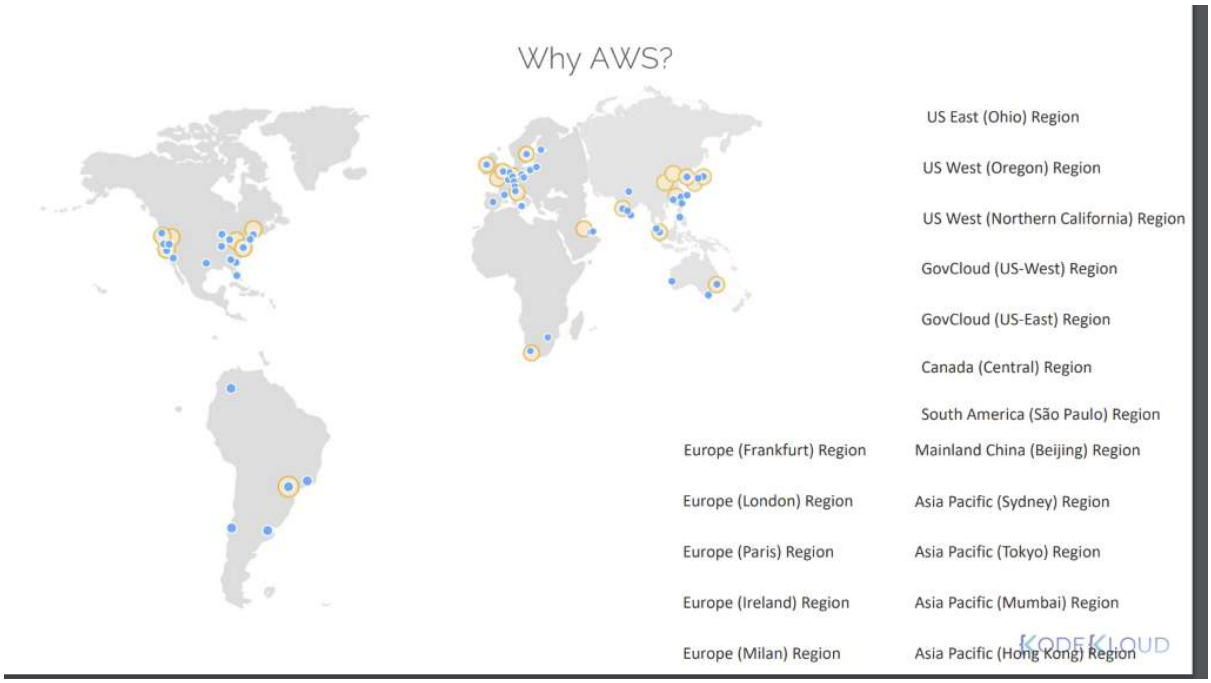
Section 7: terraform with aws

Getting started with aws

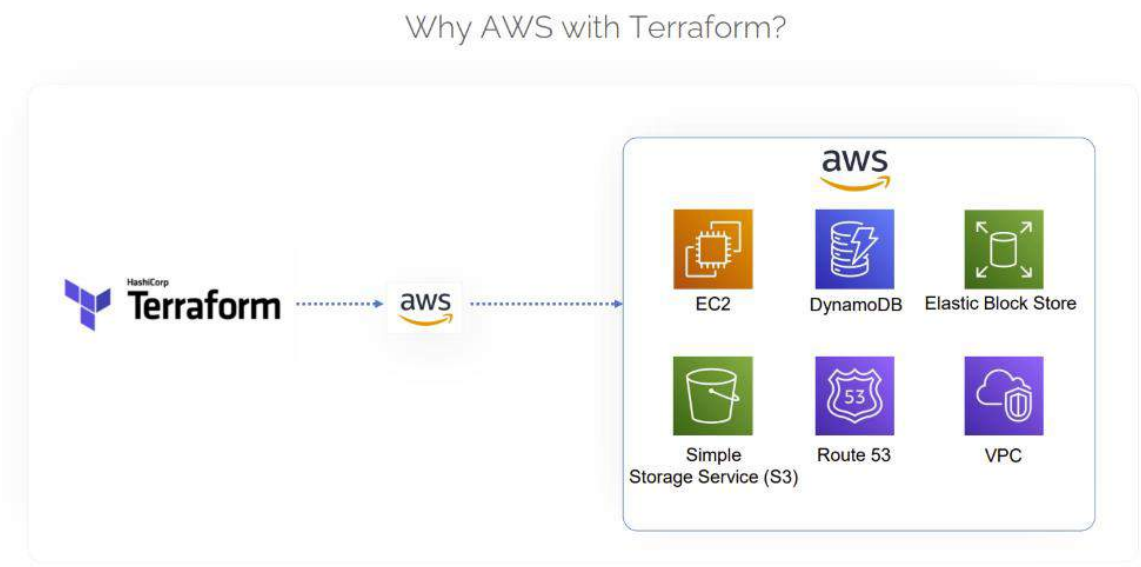
Aws: cloud computing platform. It offers 100 of services



Aws allow global cloud infrastructure. This allow to deploy services in number of globally distributed region



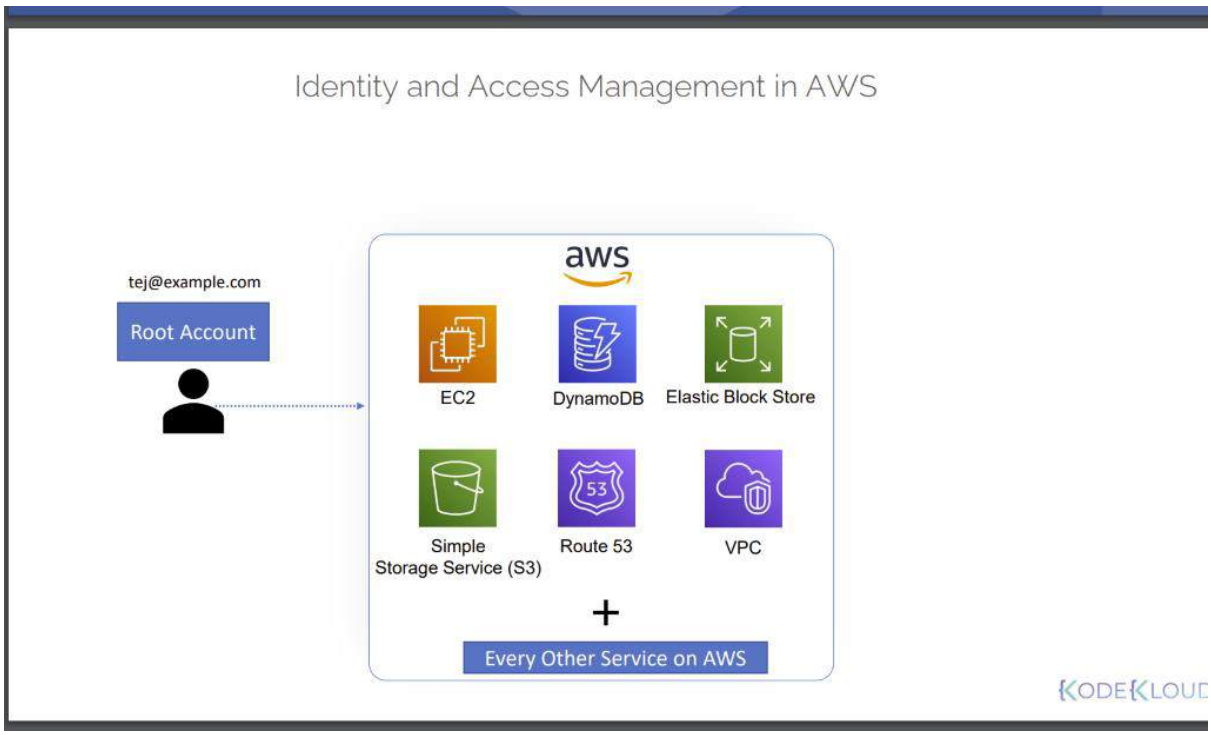
Terraform manage aws cloud based service



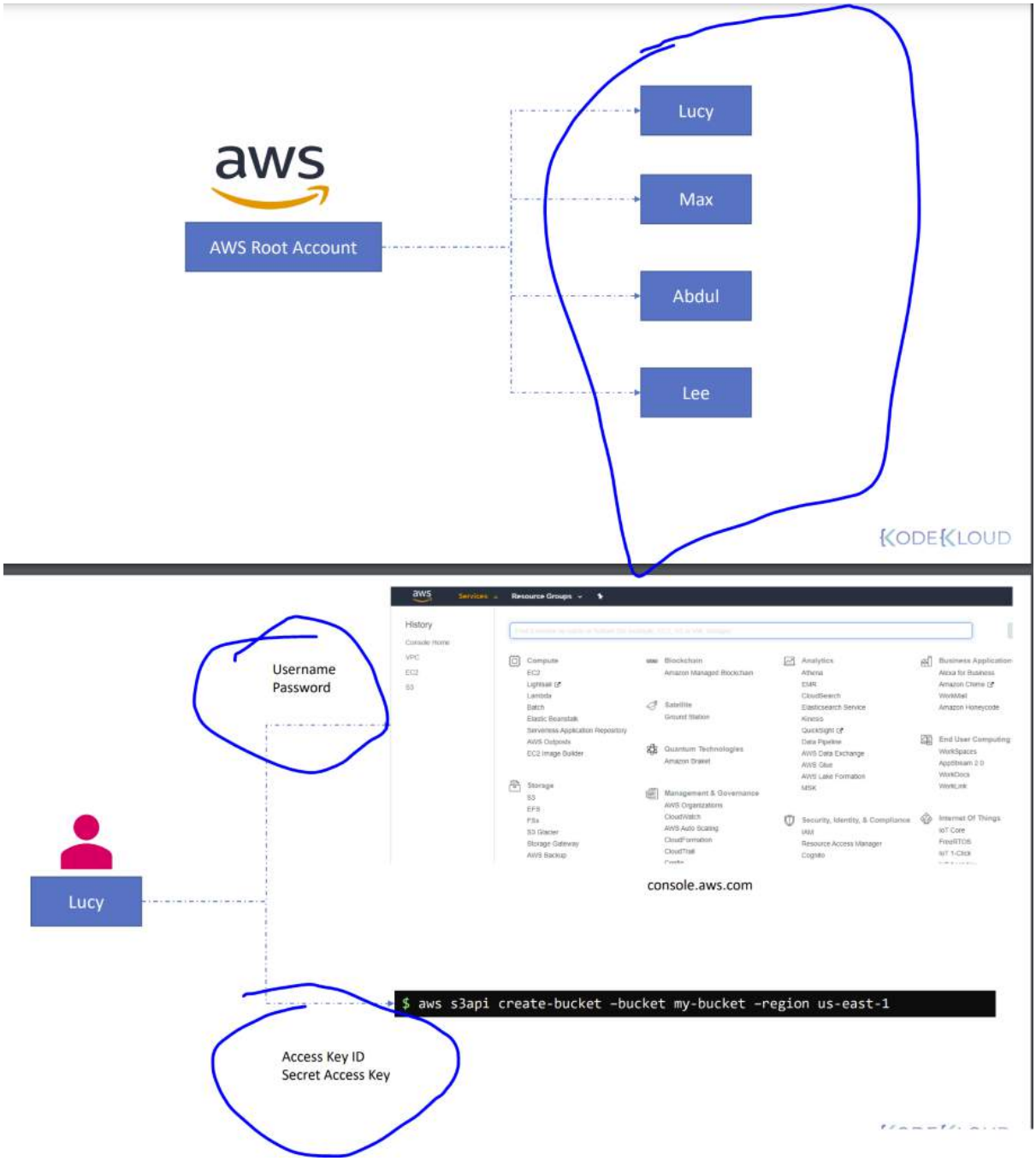
Introduction to IAM

Root account : has complete administrative privileges but this is not the recommended approach

Using root account we can create new account and assign privileges



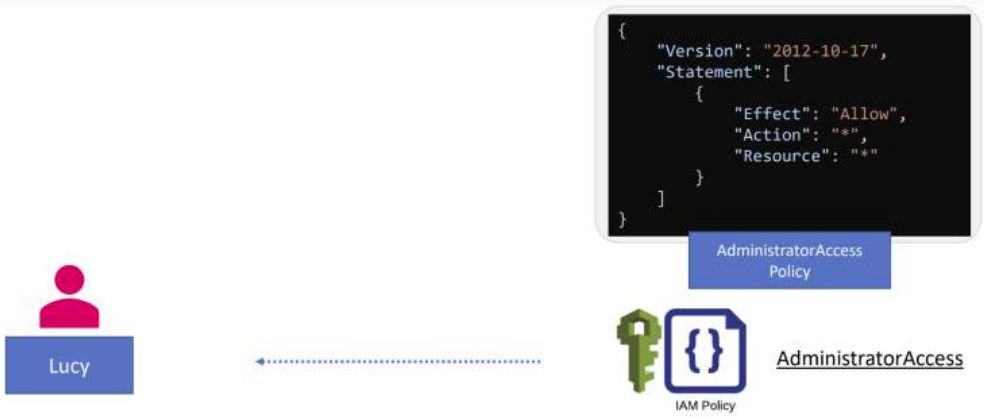
Create some user



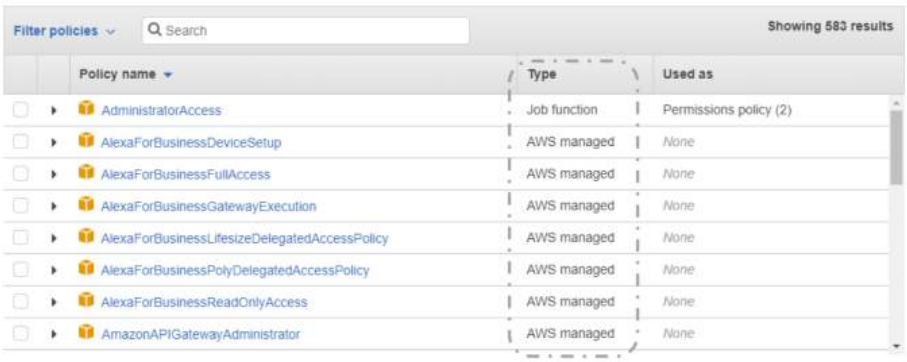
What a user can do or not that will be decide by IAM policy

IAM *policy defined in json format*

Lucy got admin access



KODEKLOUD

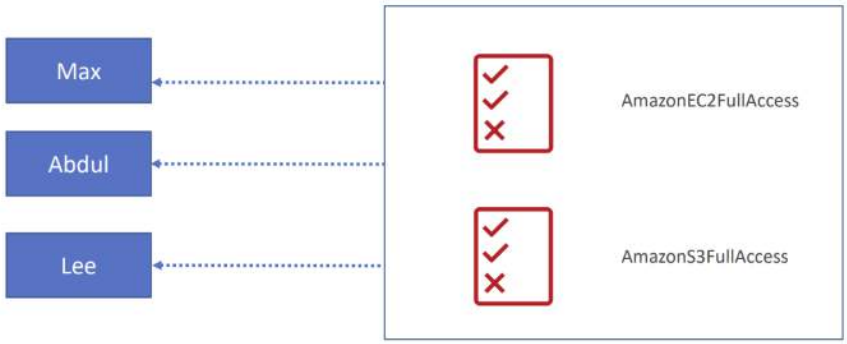


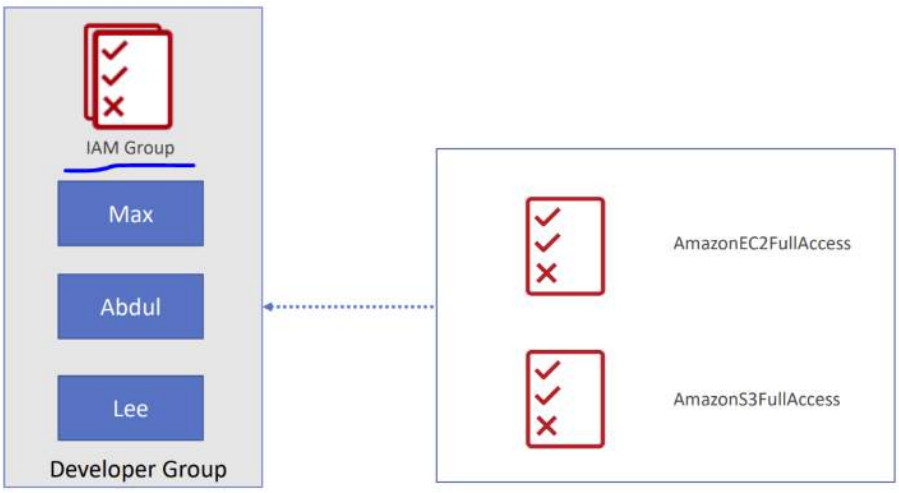
KODEKLOUD

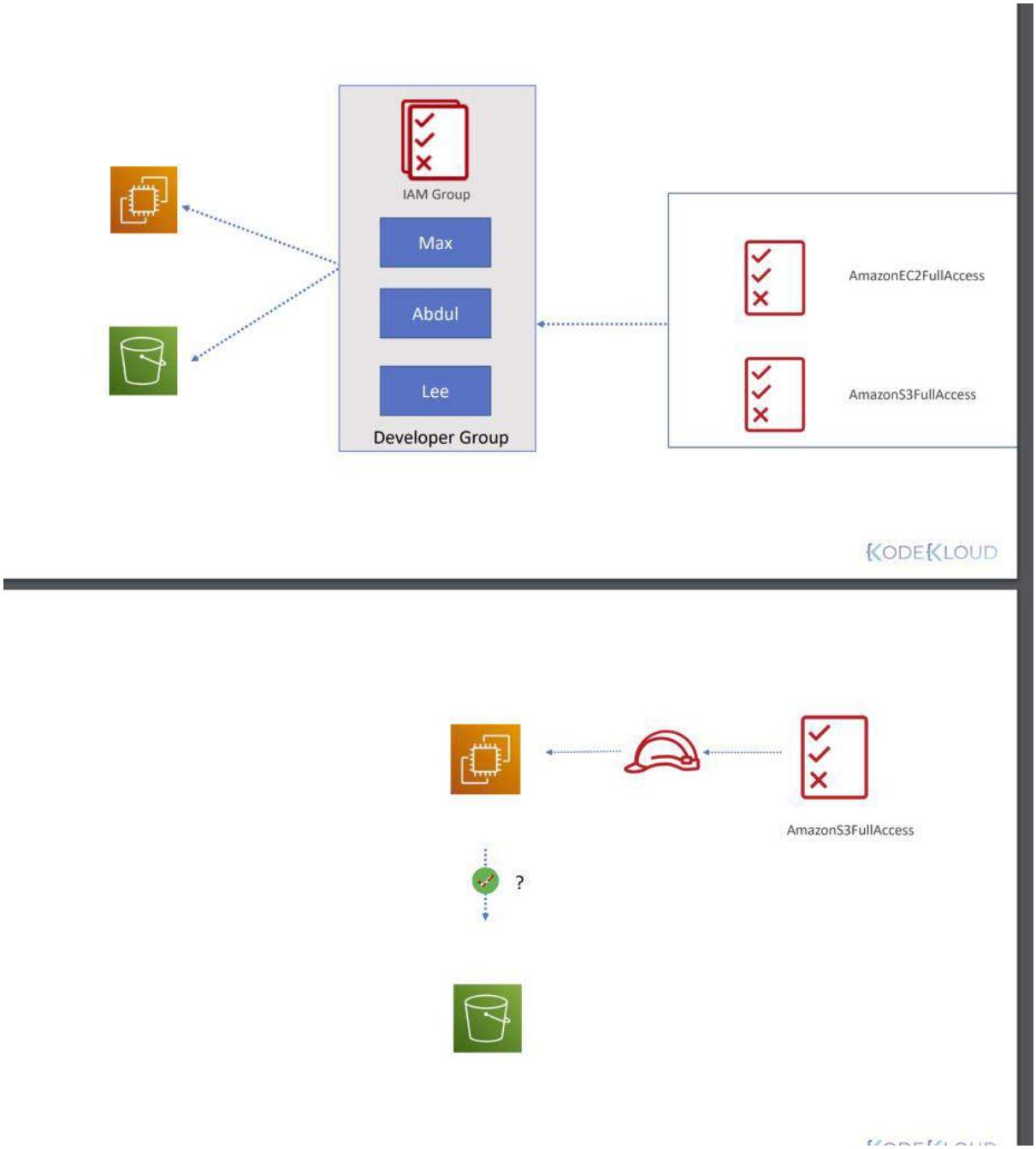
https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_job-functions.html

Job Function	Policy Name
Administrator	AdministratorAccess
Billing	Billing
Database Administrator	DatbaseAdministrator
Network Administrator	NetworkAdministrator
View-Only User	ViewOnlyAccess

IAM group *: is a collection of user . This allow to give permission to multiple users at once*


Max
Abdul
Lee
AmazonEC2FullAccess
AmazonS3FullAccess


IAM Group
Max
Abdul
Lee
Developer Group
AmazonEC2FullAccess
AmazonS3FullAccess



Note : imp concept

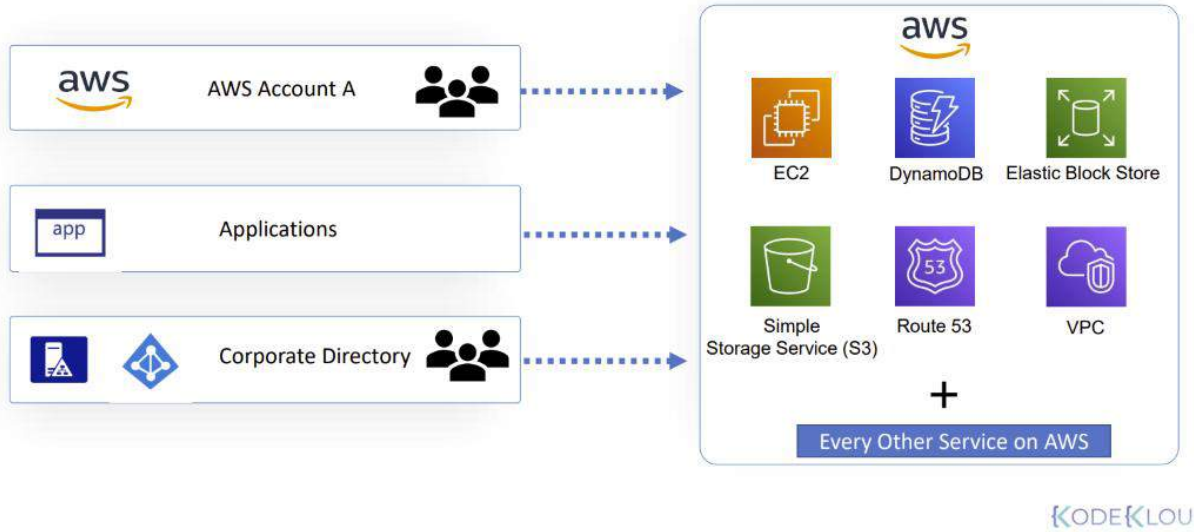
Imp point : just like regular user resources in aws(ec2 , dynamo db) also do not have permission to interact with each other by default. They need to be granted access specifically.

However unlike user we cannot attached policy to service directly. However in order to allow ec2 instance to interact with s3 service we have to create ***role.***

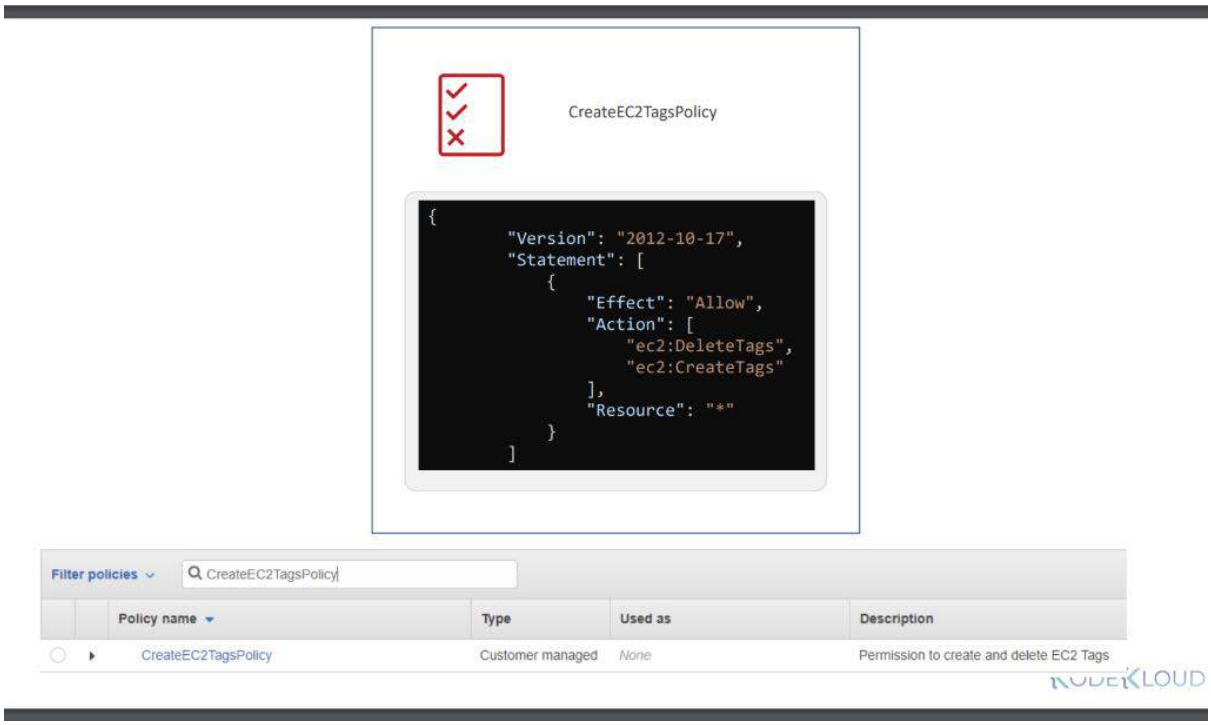
An *IAM role grant access to aws service*

An IAM role is **an identity within your AWS account that has specific permissions**. It is similar to an IAM user, but is not associated with a specific person.

IAM ROLE : it can be use to access another aws account



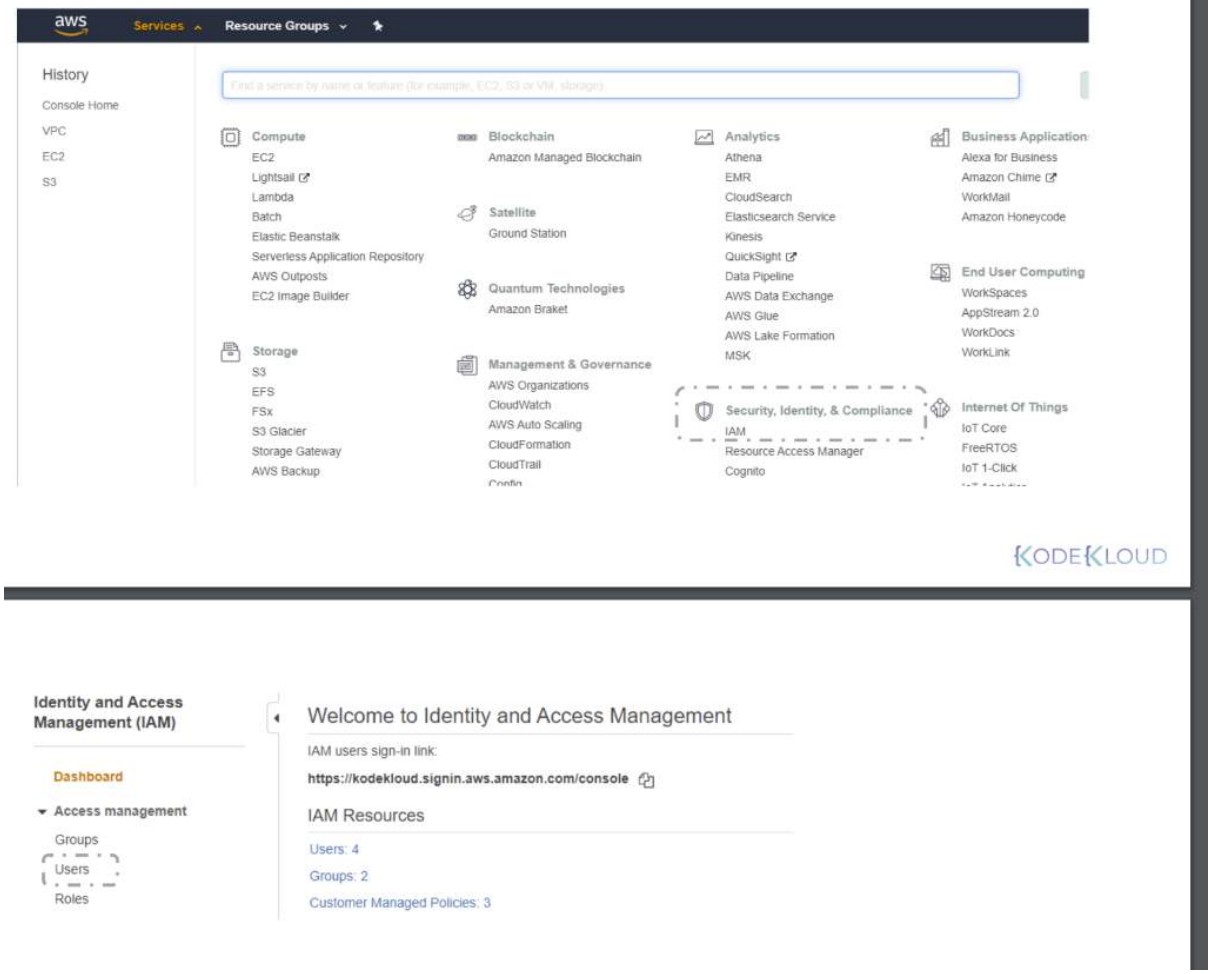
We can create custom policy also. Once created policy will be available to aws account



Demo IAM

Will create user , group , attach policy, create custom IAM policy and finally create IAM role

NOTE:
Iam is independent of region . It is global service. Once we create user or group , it will be available in all region of account



Objective : create ec2 read only custom policy

*Click policies ---- create policy ----**ec2-list-read***
The policy will now be available in policy list.
This policy now can be attached user, group or roles

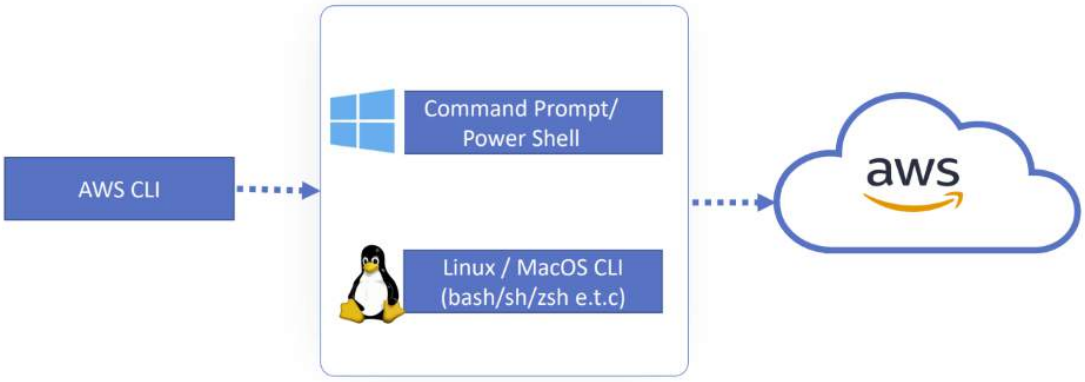
*Create another policy read and list for **s3 service ------ s3-readonly-policy***

Let now create a role for this usecase

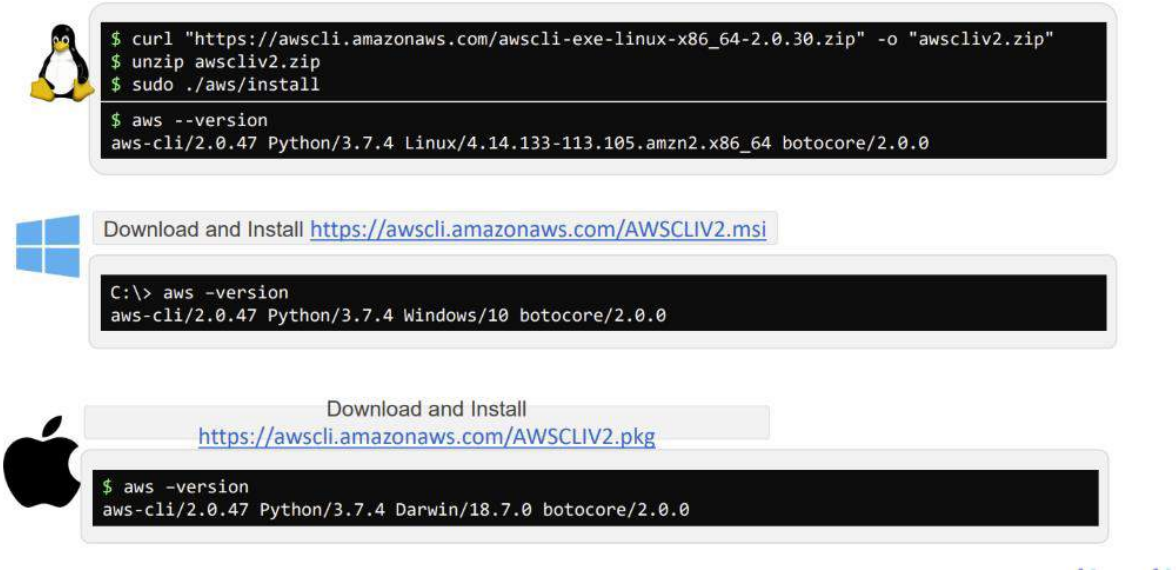
Roles: grant ec2 instance to read only access of s3 service

Programmatic access

Aws cli : *is a open source tool that allow us to interact with aws services*



Download and install aws cli



Configure aws account

```
$ aws configure
  AWS Access Key ID [None]: AKIAI44QH8DHBEXAMPLE
  AWS Secret Access Key [None]: je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
  Default region name [None]: us-west-2
  Default output format [None]: json
```

```
$ cat .aws/config/config
[default]
region = us-west-2
output = text
```

```
$ cat .aws/config/credentials
[default]
aws_access_key_id = AKIAI44QH8DHBEXAMPLE
aws_secret_access_key = je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
```

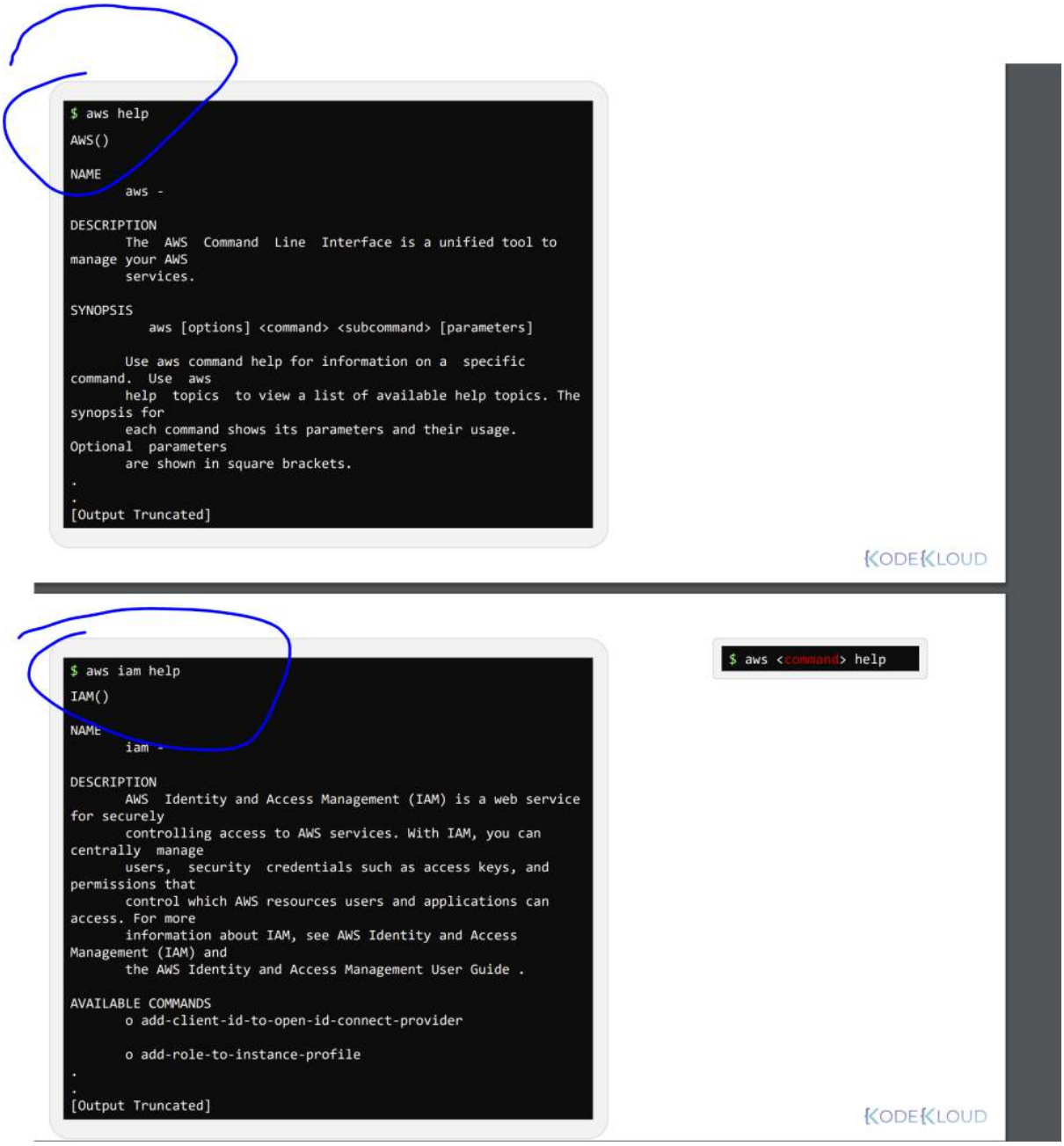
Credential will be stored in .aws directory

Amazon Resource Names (ARNs) **uniquely identify AWS resources**
Arn : is unique name which is assigned to every resource in aws

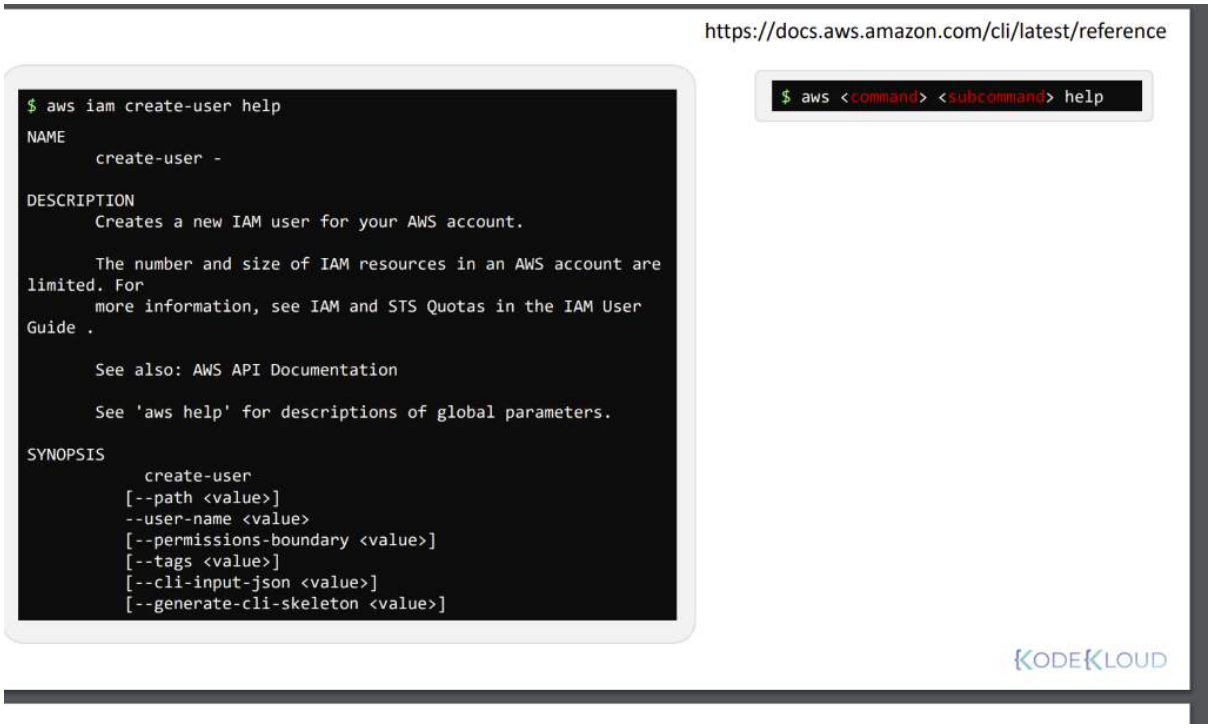
```
$ aws <command> <subcommand> [options and parameters]
```

```
$ aws iam create-user --user-name lucy
{
    "User": {
        "UserName": "lucy",
        "Tags": [],
        "CreateDate": "2020-09-15T23:40:11.168Z",
        "UserId": "h9r2sc5br8ss7uzhs2qm",
        "Path": "/",
        "Arn": "arn:aws:iam::000000000000:user/lucy"
    }
}
```

Command	Value
command	iam
subcommand	create-user
option	--user-name
parameter	lucy

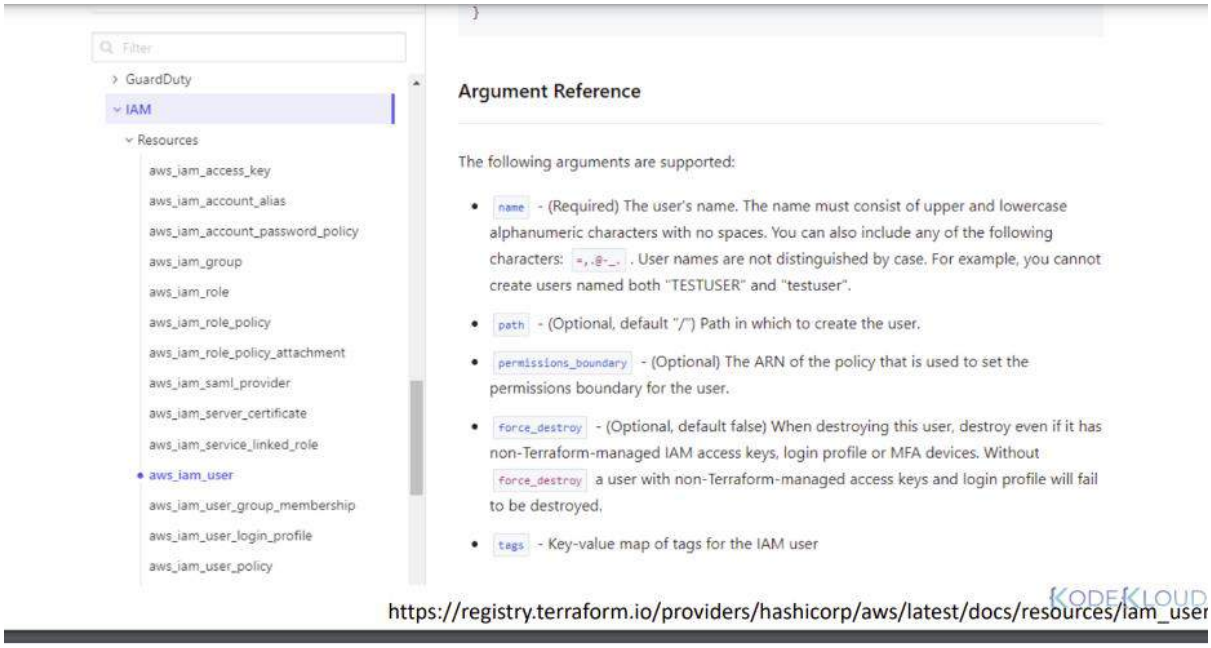


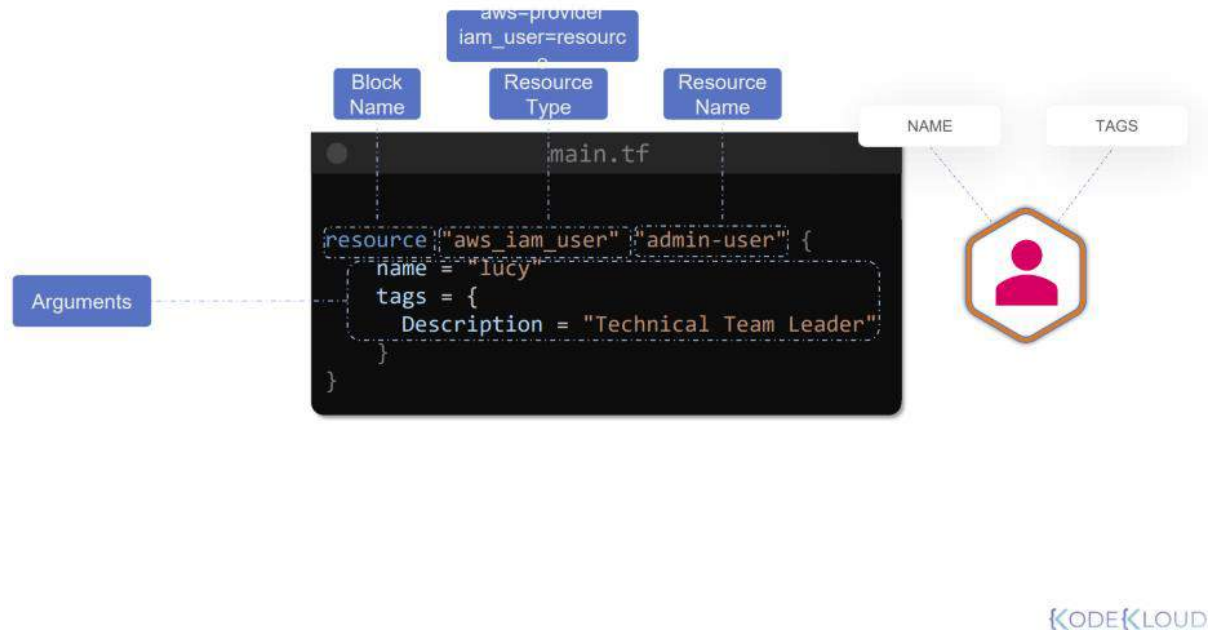
We can get help about sub command also



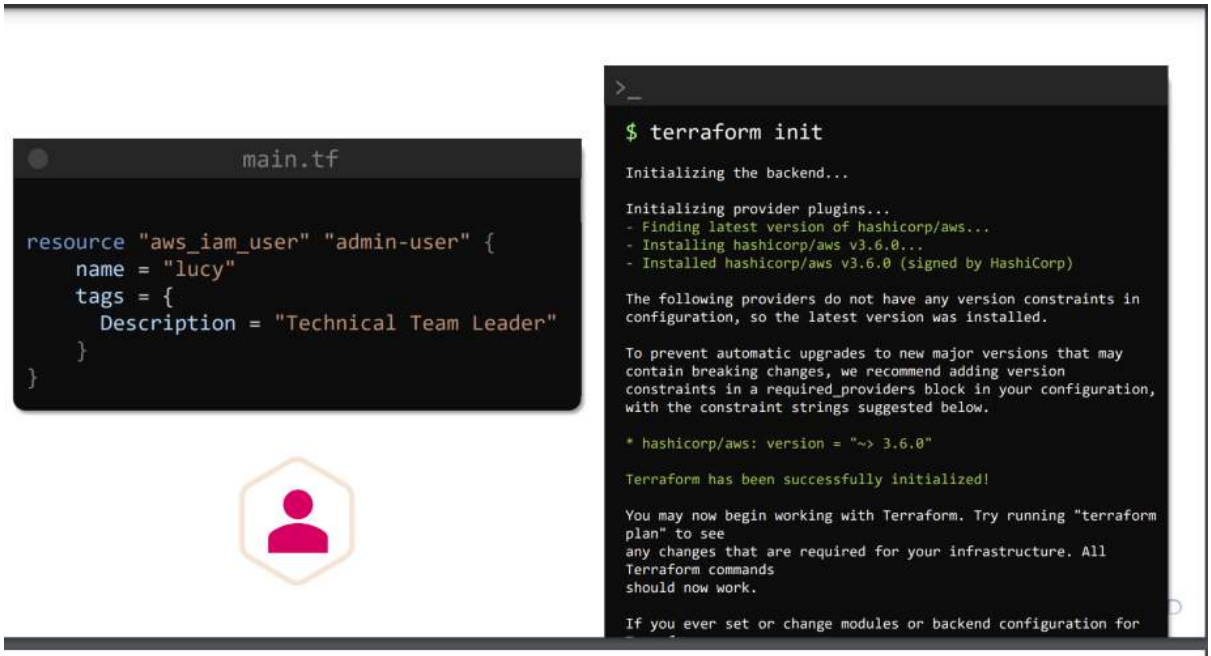
AWS IAM with Terraform

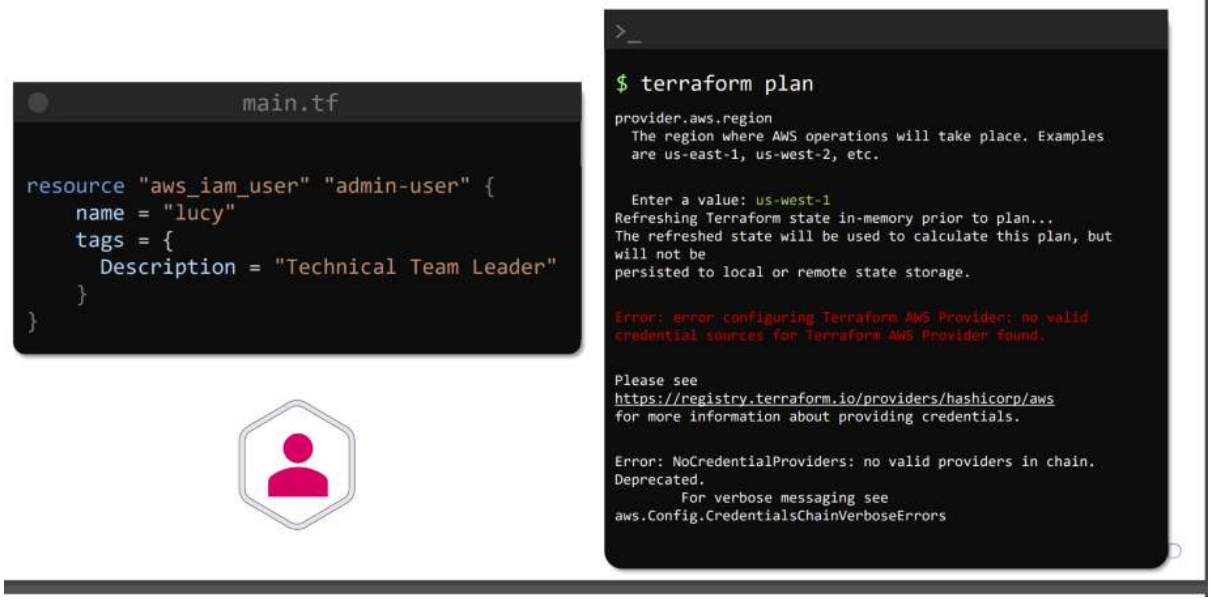
https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/iam_user





Once we run terraform plan two issue will get region and credential





We can configure credential in provider:-

main.tf

```
provider "aws" {
    region = "us-west-2"
    access_key = "AKIAI44QH8DHBEXAMPLE"
    secret_key = "je7MtGbClwBF/2tk/h3yCo8n..."
}
resource "aws_iam_user" "admin-user" {
    name = "lucy"
    tags = {
      Description = "Technical Team Leader"
    }
}
```

```
>_
$ terraform plan
.
.
+ create

Terraform will perform the following actions:

  # aws_iam_user.admin-user will be created
  + resource "aws_iam_user" "admin-user" {
      + arn           = (known after apply)
      + force_destroy = false
      + id            = (known after apply)
      + name          = "Lucy"
      + path          = "/"
      + tags          = {
          + "Description" = "Technical Team Lead"
        }
      + unique_id     = (known after apply)
    }

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

------------------------------------------------------------
-----------

Note: You didn't specify an "-out" parameter to save this
```

main.tf

```
provider "aws" {
    region = "us-west-2"
    access_key = "AKIAI44QH8DHBEXAMPLE"
    secret_key = "je7MtGbClwBF/2tk/h3yCo8n..."
}
resource "aws_iam_user" "admin-user" {
    name = "lucy"
    tags = {
      Description = "Technical Team Leader"
    }
}
```

```
>_
$ terraform apply


# aws_iam_user.admin-user will be created
  + resource "aws_iam_user" "admin-user" {
      + arn           = (known after apply)
      + force_destroy = false
      + id            = (known after apply)
      + name          = "Lucy"
      + path          = "/"
      + tags          = {
          + "Description" = "Technical Team Lead"
        }
      + unique_id     = (known after apply)
    }

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

aws_iam_user.admin-user: Creating...
aws_iam_user.admin-user: Creation complete after 1s
[id=Lucy]

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

main.tf

```
provider "aws" {
    region = "us-west-2"
    access_key = "AKIAI44QH8DHBEXAMPLE"
    secret_key = "je7MtGbClwBF/2tk/h3yCo8n…"
}
resource "aws_iam_user" "admin-user" {
    name = "lucy"
    tags = {
      Description = "Technical Team Leader"
    }
}
```

.aws/config/credentials

```
[default]
aws_access_key_id =
aws_secret_access_key =
```

KODEKLOUD

main.tf

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


}
resource "aws_iam_user" "admin-user" {
    name = "lucy"
    tags = {
      Description = "Technical Team Leader"
    }
}
```

.aws/config/credentials

```
[default]
aws_access_key_id = AKIAI44QH8DHBEXAMPLE
aws_secret_access_key = je7MtGbClwBF/2tk/h3yCo8n…
```

>_

```
$ export AWS_ACCESS_KEY_ID=
$ export AWS_SECRET_ACCESS_KEY_ID=
```

KODEKLOUD

main.tf

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


}
resource "aws_iam_user" "admin-user" {
    name = "lucy"
    tags = {
      Description = "Technical Team Leader"
    }
}
```

.aws/config/credentials

```
[default]
aws_access_key_id =
aws_secret_access_key =
```

>_

```
$ export AWS_ACCESS_KEY_ID=AKIAI44QH8DHBEXAMPLE
$ export AWS_SECRET_ACCESS_KEY_ID=je7MtGbClwBF/2tk/h3yCo8n…
$ export AWS_REGION=us-west-2
```

IAM Policies with Terraform

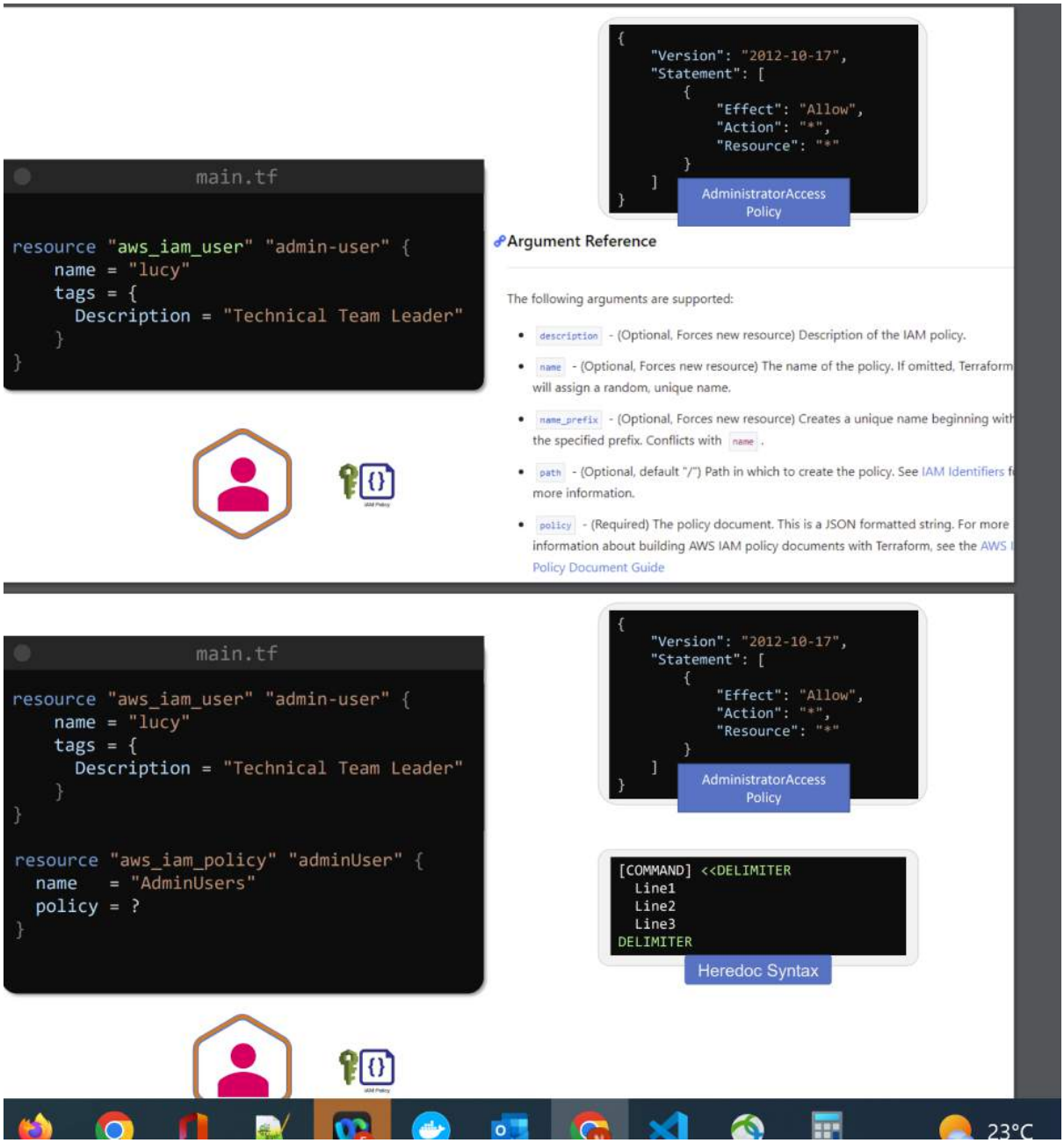
Objective : How to create iam policy with terraform and attach it to user

All users start with least privilege in aws and Lucy does not have any permission at this time.

main.tf

```
resource "aws_iam_user" "admin-user" {
    name = "lucy"
    tags = {
      Description = "Technical Team Leader"
    }
}
```

To add more permission we have to attached IAM policy.
Permission are assigned by policy document which is in json format



How to attach policy document here. There are multiple ways

We can pass entire json file between the delimeter

Third part is : iam policy attached to user

Now we can run terraform init , plan and apply command

There is another way to attached policy

Introduction to aws s3

S3 : simple storage service

It is an object based storage. It allows to store flat file such as document , images and videos but not suitable to store os or databases.

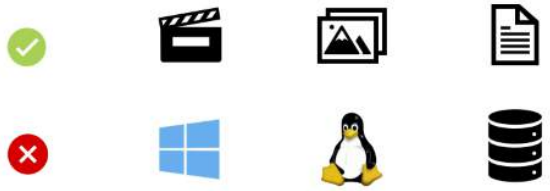
Data in s3 store in the form of s3 bucket. A bucket can be consider a container or directory which will store all file.

Everything within a bucket is an object

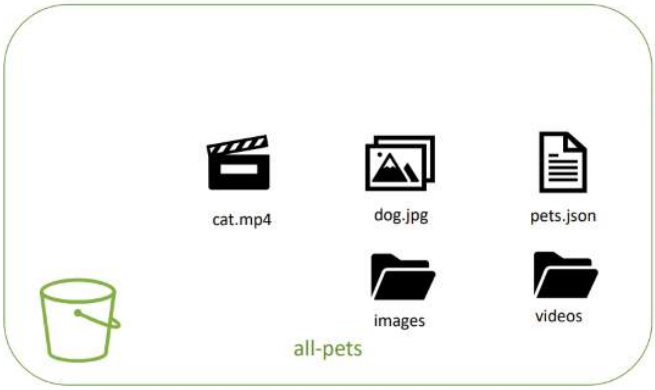
There are couple of consideration should be followed while creating bucket

Bucket name should be unique
While creating a bucket aws also create **dns name for it** and this dns name will be accessible from anywhere in the world.

No two bucket can have same name whether it is created into diff account.



KODEKLOUD



Object #	Name
1	pets.json
2	dog.jpg
3	cat.mp4
4	pictures/cat.jpg
5	videos/dog.mp4

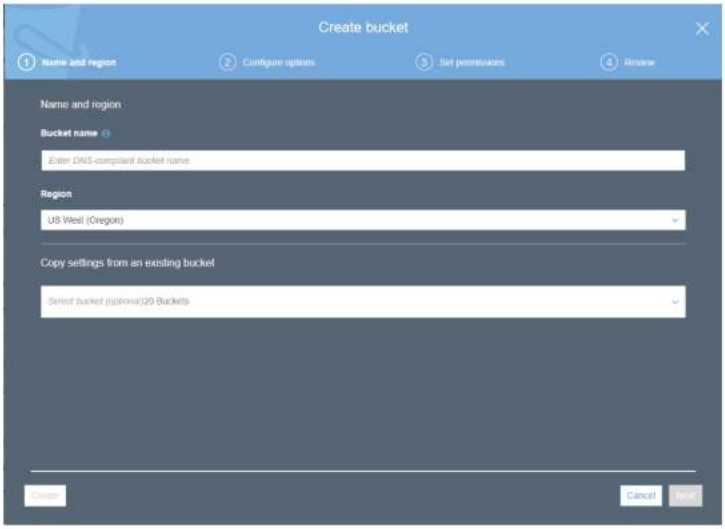
KODEKLOUD

https://docs.aws.amazon.com/AmazonS3/latest/dev/BucketRestrictions.html

Unique Bucket Name

DNS Compliant Name

Files size between 0 to 5TB



KODEKLOUD

https://<bucket_name>.<region>.amazonaws.com

https://all-pets.us-west-1.amazonaws.com

Object #	Name	Address
1	pets.json	https://all-pets.us-west-1.amazonaws.com/pets.json
2	dog.jpg	https://all-pets.us-west-1.amazonaws.com/dog.jpg
3	cat.mp4	https://all-pets.us-west-1.amazonaws.com/cat.mp4
4	pictures/cat.jpg	https://all-pets.us-west-1.amazonaws.com/pictures/cat.jpg
5	videos/dog.mp4	https://all-pets.us-west-1.amazonaws.com/videos/dog.mp4

Any object is s3 consist of **object data and metadata**

Metadata consist of owner, size of object

Note :

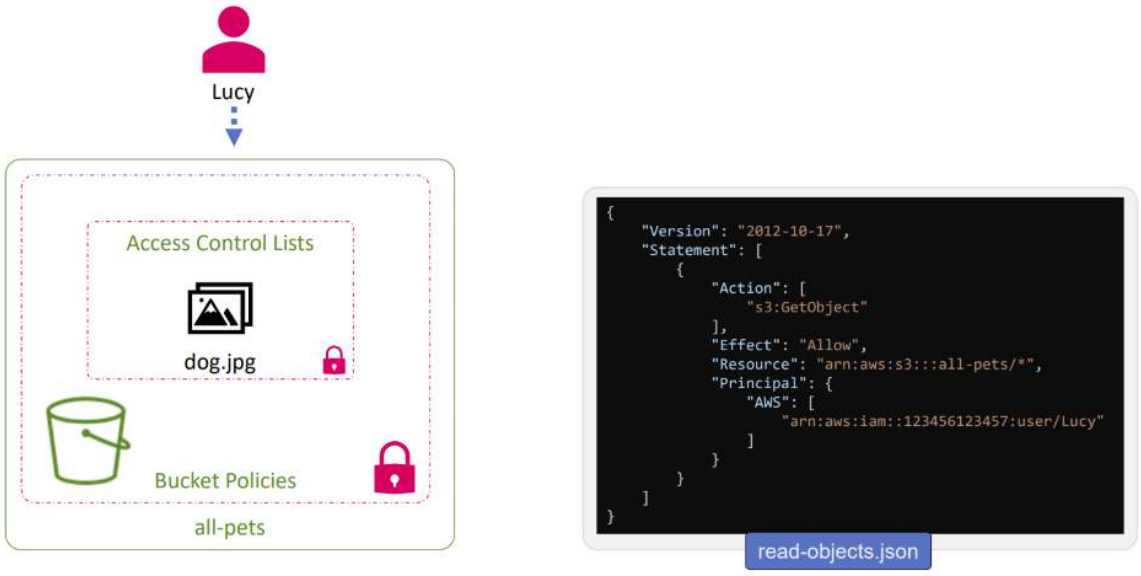
When data is uploaded into bucket it have least privilege means noone can access bucket except owner.
Access to bucket and it object are governed by bucket policies and access controller list.

Bucket polices are permission that are granted at bucket level

Access Control list are permission that are defined at object level

Using **bucket policy** we can grant access to iam user or group or even other aws account user

KODEKLOUD



S3 with Terraform

Note :
If the bucket name is not provided terraform will use random to create a unique bucket

To create bucket : we should use aws_s3_bukcet

main.tf

```
resource "aws_s3_bucket" "finance" {
    bucket = "finanace-21092020"
    tags   = {
        Description = "Finance and Payroll"
    }
}
```

```
$ terraform apply
Terraform will perform the following actions:

  # aws_s3_bucket.finance will be created
  + resource "aws_s3_bucket" "finance" {
      + acceleration_status  = (known after apply)
      + acl                  = "private"
      + arn                  = (known after apply)
      + bucket               = "finanace-21092020"
.
.
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.

  Enter a value: yes

aws_s3_bucket.finance: Creating...
aws_s3_bucket.finance: Creation complete after 0s
[id=finanace-21092020]

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

To upload file :-- use aws_s3_bucket_object

main.tf

```
resource "aws_s3_bucket" "finance" {
    bucket = "finanace-21092020"
    tags   = {
        Description = "Finance and Payroll"
    }
}

resource "aws_s3_bucket_object" "finance-2020" {
    content = "/root/finance/finance-2020.doc"
    key     = "finance-2020.doc"
    bucket = aws_s3_bucket.finance.id
}
```

```
$ terraform apply
.
.
Terraform will perform the following actions:

  # aws_s3_bucket_object.finance-2020 will be created
  + resource "aws_s3_bucket_object" "finance-2020" {
      + acl                    = "private"
      + bucket                 = "finanace-21092020"
      + content                = "/root/finance/finance-
2020.doc"
+ force_destroy          = false
      + id                     = (known after apply)
      + key                    = "finance/finance-
2020.doc"

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.

  Enter a value: yes

aws_s3_bucket_object.finance-2020: Creating...
aws_s3_bucket_object.finance-2020: Creation complete
after 0s [id=finance/finance-2020.doc]
```

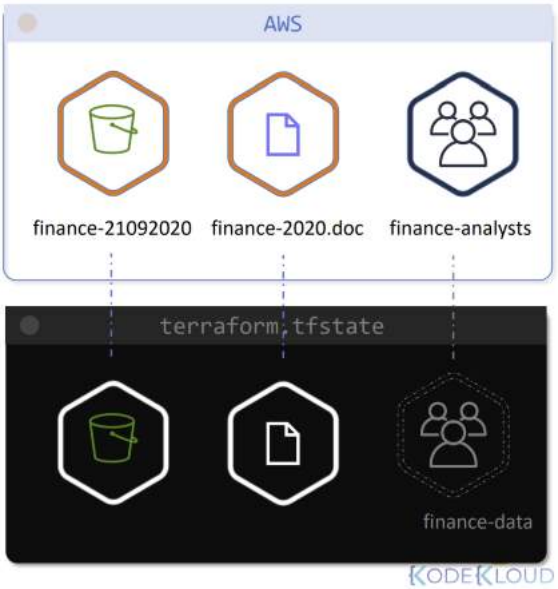
Once we apply the file will be uploaded to s3 bucket

So far we have created two resources a bucket and a bucket object and both of these created in aws infrastructure. The state of these resources recorded in our local terraform.tfstate

Let us now see how to define bucket policy who can access this bucket

main.tf

```
resource "aws_s3_bucket" "finance" {
    bucket = "finanace-21092020"
    tags   = {
        Description = "Finance and Payroll"
    }
}

resource "aws_s3_bucket_object" "finance-2020" {
    content = "/root/finance/finance-2020.doc"
    key     = "finance-2020.doc"
    bucket = aws_s3_bucket.finance.id
}

data "aws_iam_group" "finance-data" {
  group_name = "finance-analysts"

}
```




```
    content = "/root/finance/finance-2020.doc"
    key     = "finance-2020.doc"
    bucket = aws_s3_bucket.finance.id
}

data "aws_iam_group" "finance-data" {
  group_name = "finance-analysts"

}
resource "aws_s3_bucket_policy" "finance-policy" {
  bucket = aws_s3_bucket.finance.id
  policy = <<EOF



EOF
}
```

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "*",
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::<<bucket-name>>/*",
            "Principal": {
                "AWS": [
                    "<<   arn    >>"
                ]
            }
        }
    ]
}
```

read-objects.json


```
    content = "/root/finance/finance-2020.doc"
```

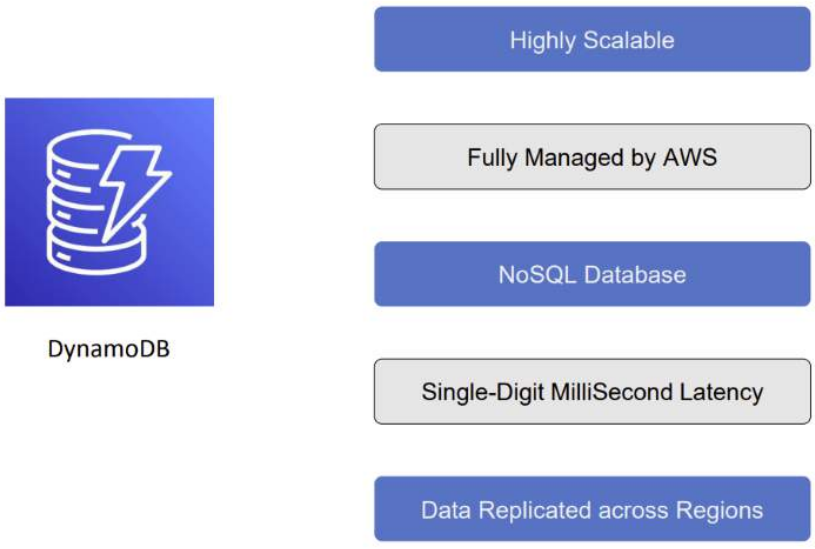
```
    content = "/root/finance/finance-2020.doc"
    key     = "finance-2020.doc"
    bucket = aws_s3_bucket.finance.id
}

data "aws_iam_group" "finance-data" {
  group_name = "finance-analysts"

}
resource "aws_s3_bucket_policy" "finance-policy" {
  bucket = aws_s3_bucket.finance.id
  policy = <<EOF
  {
      "Version": "2012-10-17",
      "Statement": [
          {
              "Action": "*",
              "Effect": "Allow",
              "Resource": "arn:aws:s3:::${aws_s3_bucket.finance.id}/*",
              "Principal": {
                  "AWS": [
                      "${data.aws_iam_group.finance-data.arn}"
                  ]
              }
          }
      ]
  }

EOF
}
```

aws KODEKLOU

Introduction to dynamo db

Dynamo db is no sql database solution provided by AWS

It store data in form of key value pair and document.



Dynamo db with terraform

Use resource type ： aws_dynamodb_table

Create table

main.tf

```
resource "aws_dynamodb_table" "cars" {
  name     = "cars"
  hash_key = "VIN"
  billing_mode = "PAY_PER_REQUEST"
  attribute {
    name = "VIN"
    type = "S"
  }
}
```

- `billing_mode` - (Optional) Controls how you are charged for read and write throughput and how you manage capacity. The valid values are `PROVISIONED` and `PAY_PER_REQUEST`. Defaults to `PROVISIONED`.
- `write_capacity` - (Optional) The number of write units for this table. If the `billing_mode` is `PROVISIONED`, this field is required.
- `read_capacity` - (Optional) The number of read units for this table. If the `billing_mode` is `PROVISIONED`, this field is required.

main.tf

```
resource "aws_dynamodb_table" "cars" {
  name     = "cars"
  hash_key = "VIN"
  billing_mode = "PAY_PER_REQUEST"
  attribute {
    name = "VIN"
    type = "S"
  }
}
```

>_

```
$ terraform apply
+ create

Terraform will perform the following actions:

  # aws_dynamodb_table.cars will be created
  + resource "aws_dynamodb_table" "cars" {
      + arn              = (known after apply)
      + billing_mode     = "PAY_PER_REQUEST"
      + hash_key         = "VIN"
      + id               = (known after apply)
      + name             = "cars"
      + stream_arn       = (known after apply)
      + stream_label     = (known after apply)
      + stream_view_type = (known after apply)

      + attribute {
          + name = "VIN"
          + type = "S"
        }

      + point_in_time_recovery {
          + enabled = (known after apply)
        }
.
.
aws_dynamodb_table.cars: Creating...
aws_dynamodb_table.cars: Creation complete after 0s [id=cars]
```

Create item

main.tf

```
resource "aws_dynamodb_table" "cars" {
  name      = "cars"
  hash_key = "VIN"
  billing_mode = "PAY_PER_REQUEST"
  attribute {
    name = "VIN"
    type = "S"
  }
}
resource "aws_dynamodb_table_item" "car-items" {
  table_name = aws_dynamodb_table.cars.name
  hash_key   = aws_dynamodb_table.cars.hash_key
  item       = <<EOF



EOF
}
```

cars

```
{
    "Manufacturer": "Toyota",
    "Make": "Corolla",
    "Year": 2004,
    "VIN" : "4Y1SL65848Z411439"
}
{
    "Manufacturer": "Honda",
    "Make": "Civic",
    "Year": 2017,
    "VIN" : "DY1SL65848Z411432"
}
{
    "Manufacturer": "Dodge",
    "Make": "Journey",
    "Year": 2014,
    "VIN" : "SD1SL65848Z411443"
}
{
    "Manufacturer": "Ford",
    "Make": "F150",
    "Year": 2020,
    "VIN" : "DH1SL65848Z41100"
}
```

main.tf

```
resource "aws_dynamodb_table" "cars" {
  name      = "cars"
  hash_key = "VIN"
  billing_mode = "PAY_PER_REQUEST"
  attribute {
    name = "VIN"
    type = "S"
  }
}
resource "aws_dynamodb_table_item" "car-items" {
  table_name = aws_dynamodb_table.cars.name
  hash_key   = aws_dynamodb_table.cars.hash_key
  item       = <<EOF
  {
      "Manufacturer": {"S": "Toyota"},
      "Make": {"S": "Corolla"},
      "Year": {"N": "2004"},
      "VIN" : {"S": "4Y1SL65848Z411439"},
  }
EOF
}
```

```
$ terraform apply

  # aws_dynamodb_table_item.car-items will be created
  + resource "aws_dynamodb_table_item" "car-items" {
      + hash_key   = "VIN"
      + id         = (known after apply)
      + item       = jsonencode(
            {
              + Manufacturer = {
                  + S = "Toyota"
                }
              + Model        = {
                  + S = "Corolla"
                }
              + VIN          = {
                  + S = "4Y1SL65848Z411439"
                }
              + Year         = {
                  + N = "2004"
                }
            }
        )
      + table_name = "cars"
    }

Plan: 1 to add, 0 to change, 0 to destroy.
.
.
aws_dynamodb_table_item.car-items: Creating...
```

Section 8: Remote state

What is remote state and state locking

Remote state in terraform

When we run terraform apply first time by default terraform.tfState created inside the configuration directory by default

Adavantages of terraform.tfState

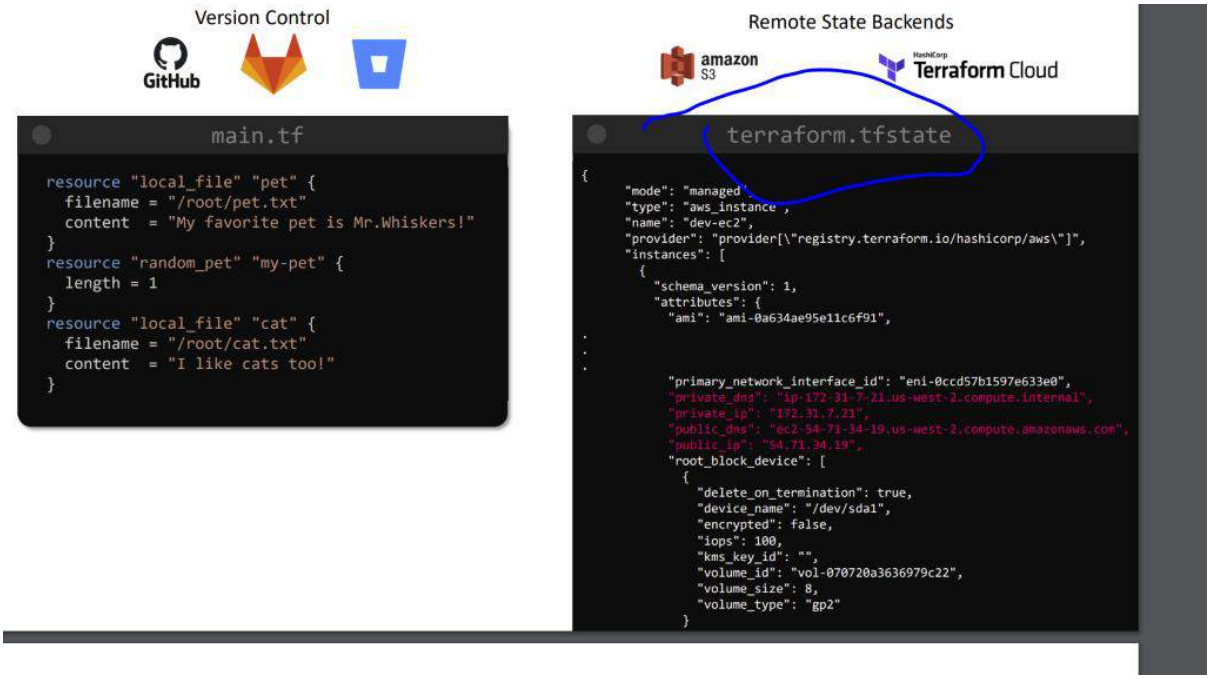
Tracking metadata : such as dependencies which allow to create and delete resources in the correct order

Improving **performance** of terraform operation while working with large terraform file

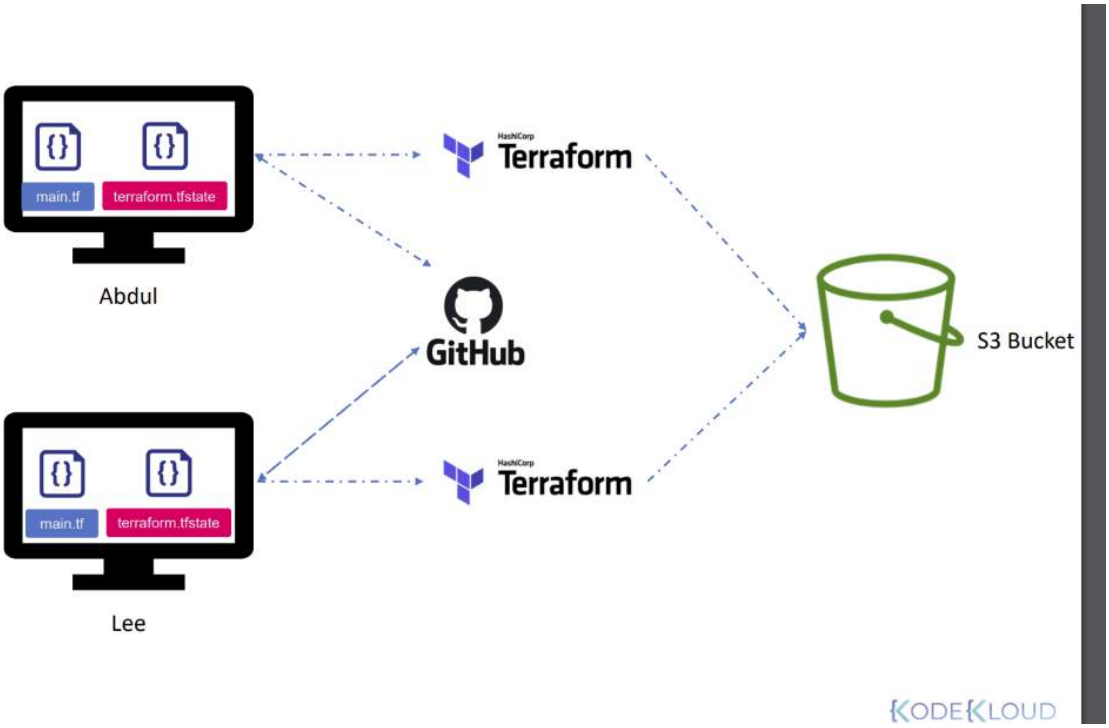
It allows to **collaborate** team

Tf.state file is created locally

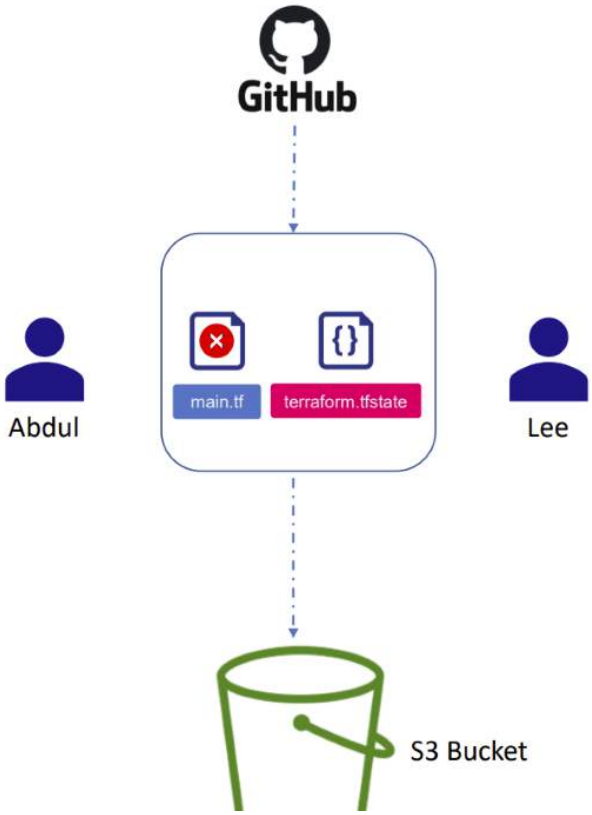
This is the local file of tf.state and it does not provide opportunity to collaborate and this is because the file is only available in client machine such as developer laptops

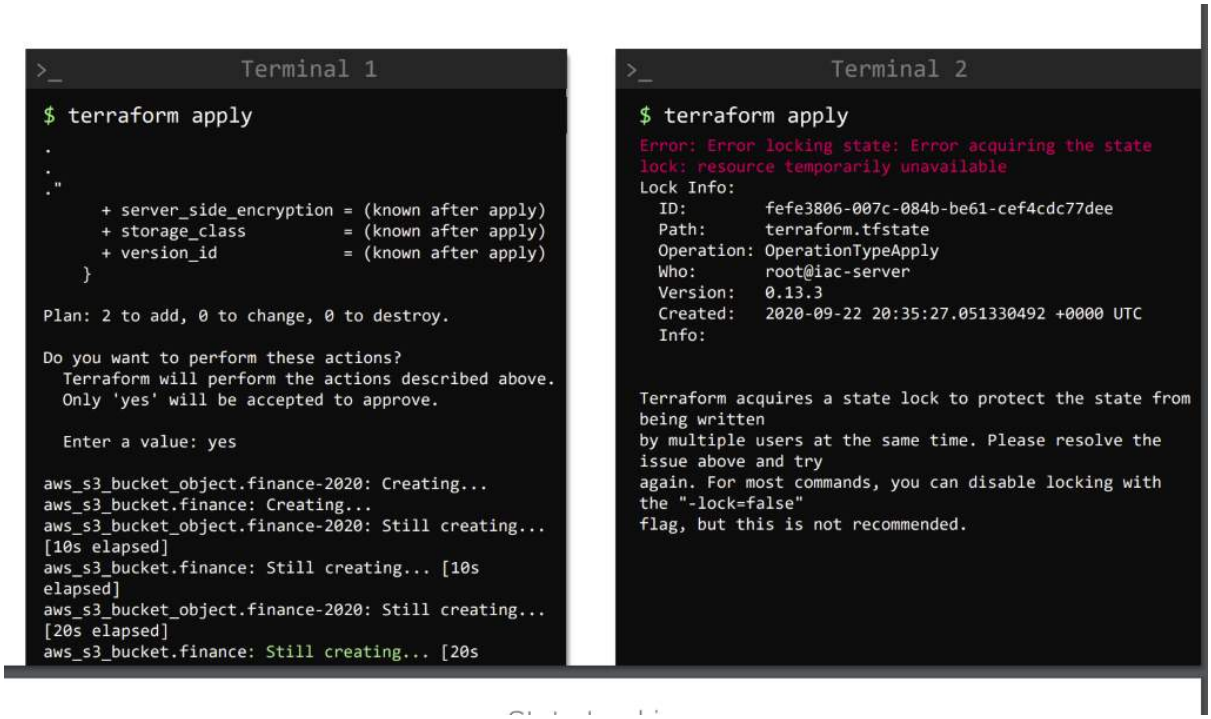


It is **not good** to store the state file in **version control system** since it store **sensitive data**



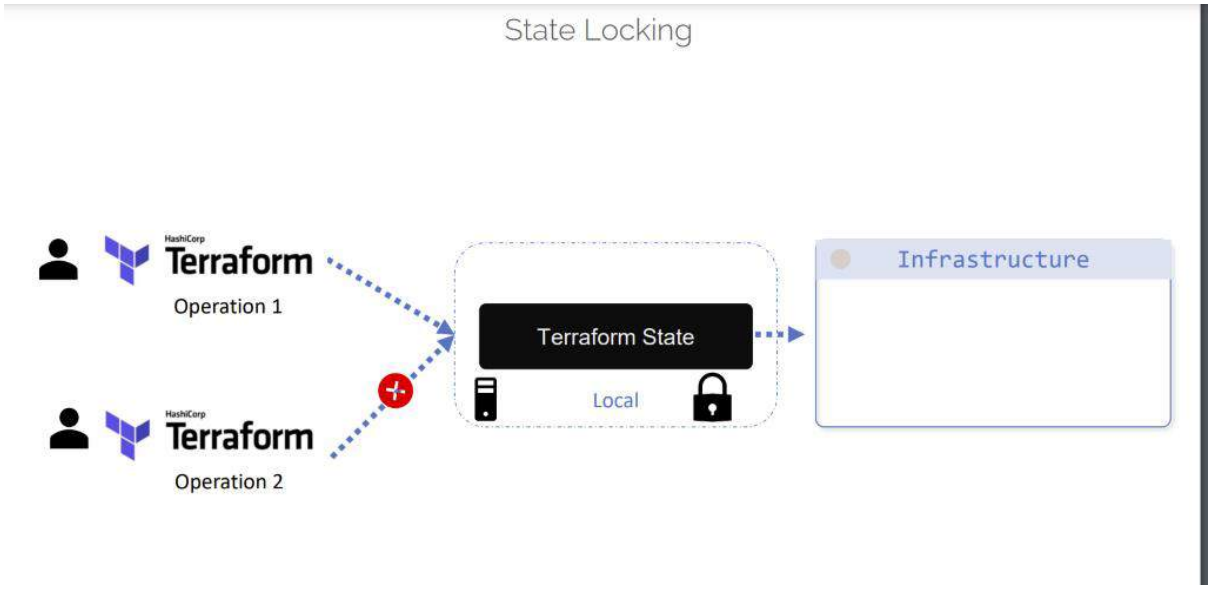
If we are running terraform apply at the same time from two diff machine for same state file then terraform will lock the second one until it is completed in first one.



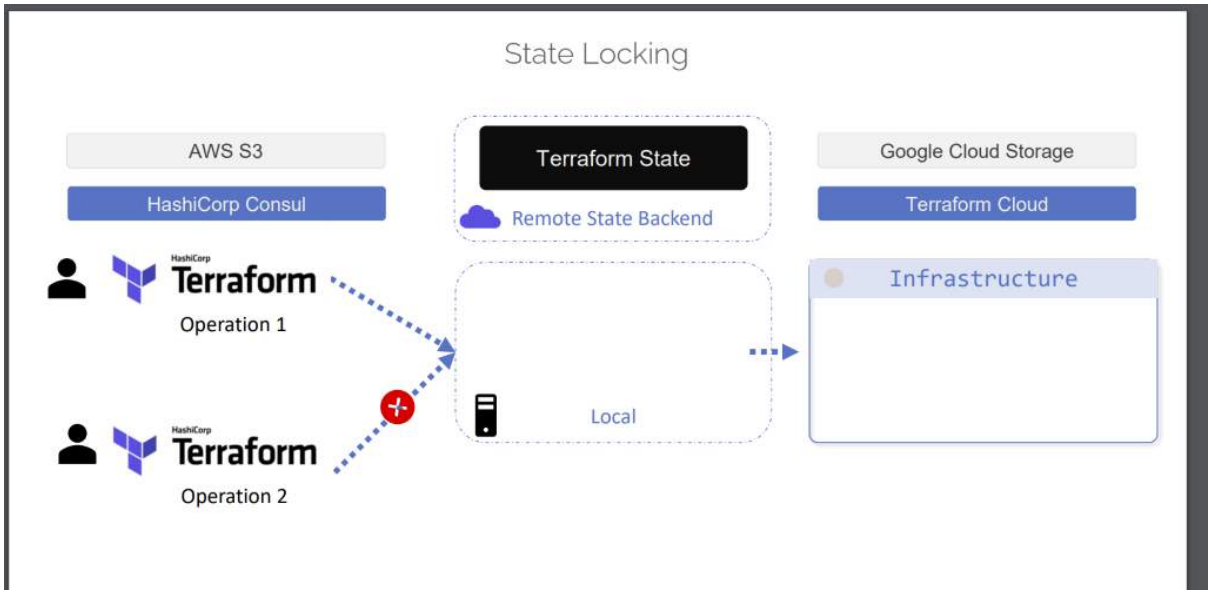


This is very important feature state locking and it ensure multiple user cannot update the file at the same time. It save to corrupt the state file

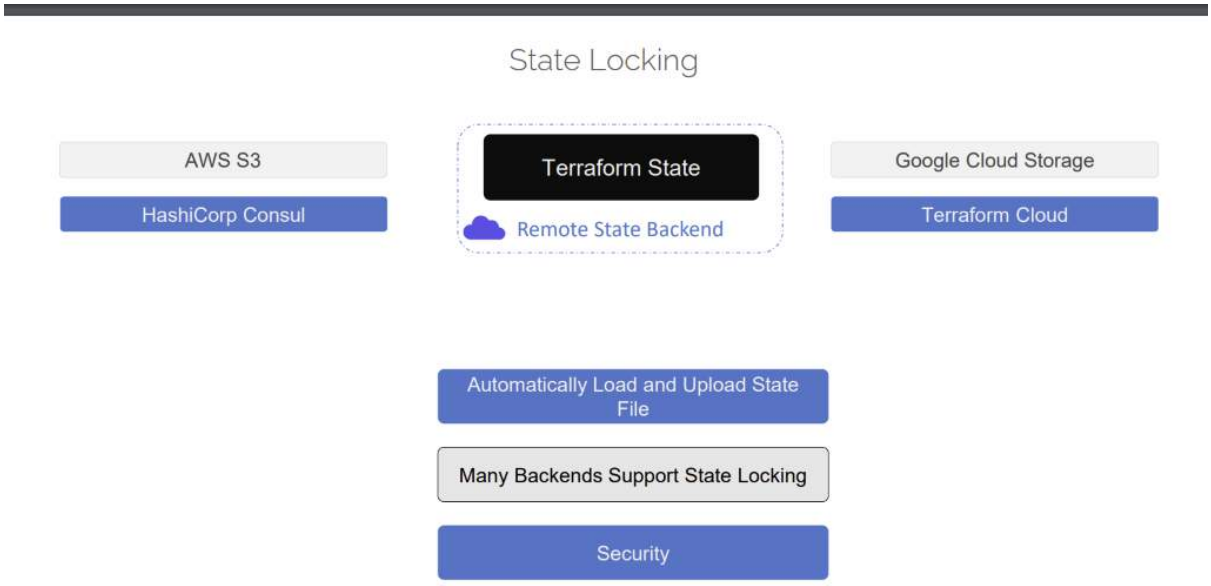
Imp : version control system such as github do not support the state locking and as a result Multiple user can use same state file simultaneously which can result data loss and conflicts in state file.



So we should move state file in remote state such as aws s3, google cloud storage.



In remote state it will update state file after every apply

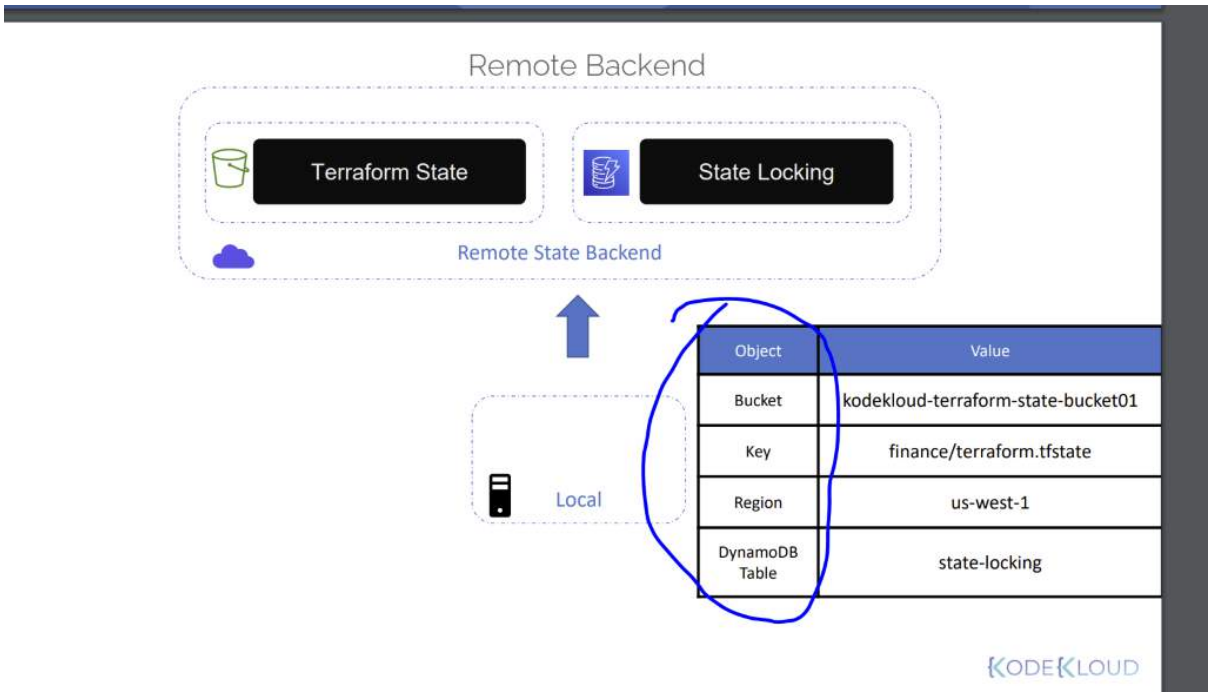


Remote backends with s3

How to use s3 bucket and a **dynamo db table** to configure remote backend for our terraform configuration

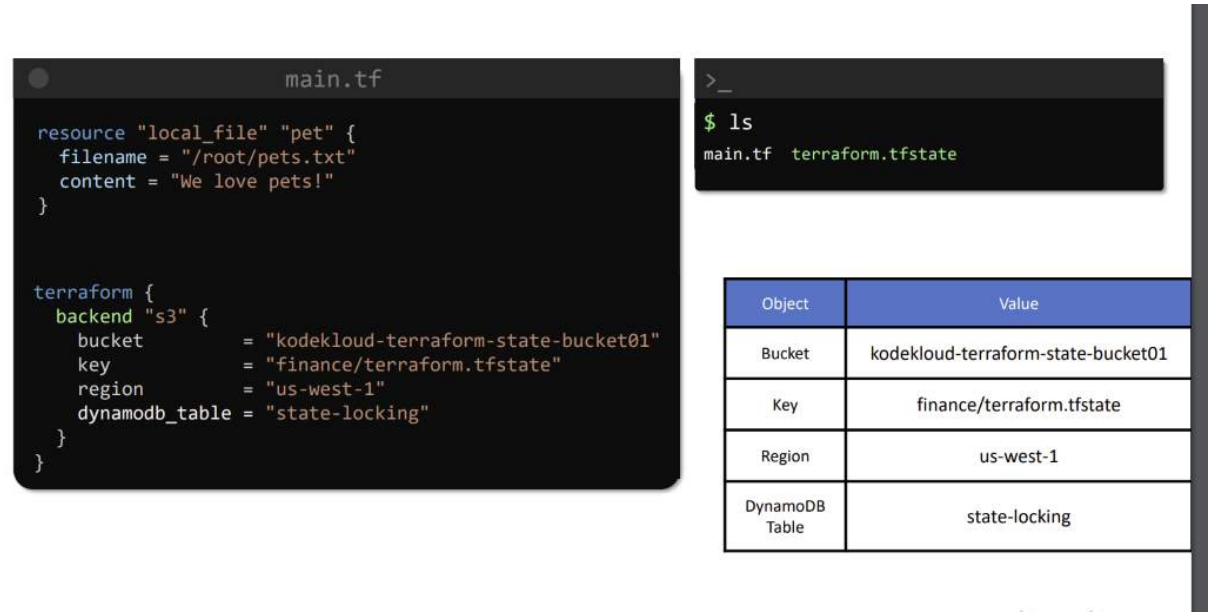
We need two thing
S3 bucket which will use to store terraform state file
and dynamo db table which will use to implement state locking consistency check.

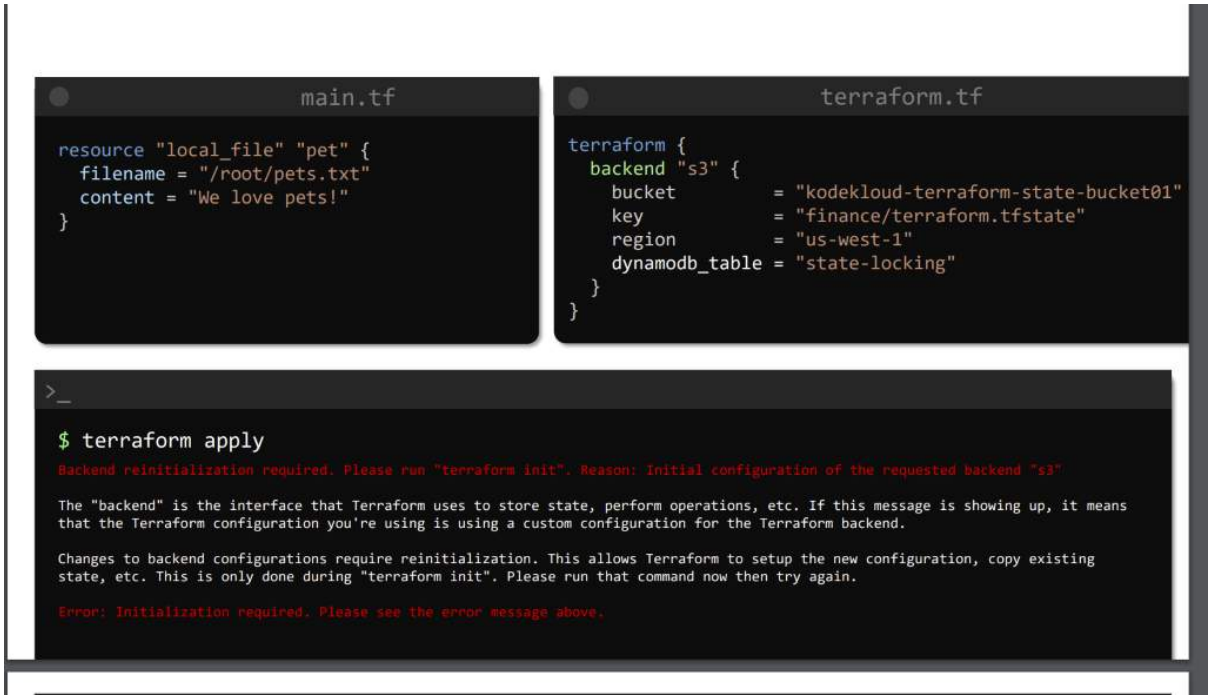
First these two requisite will be completed before beginning to remote backend
And note all the field



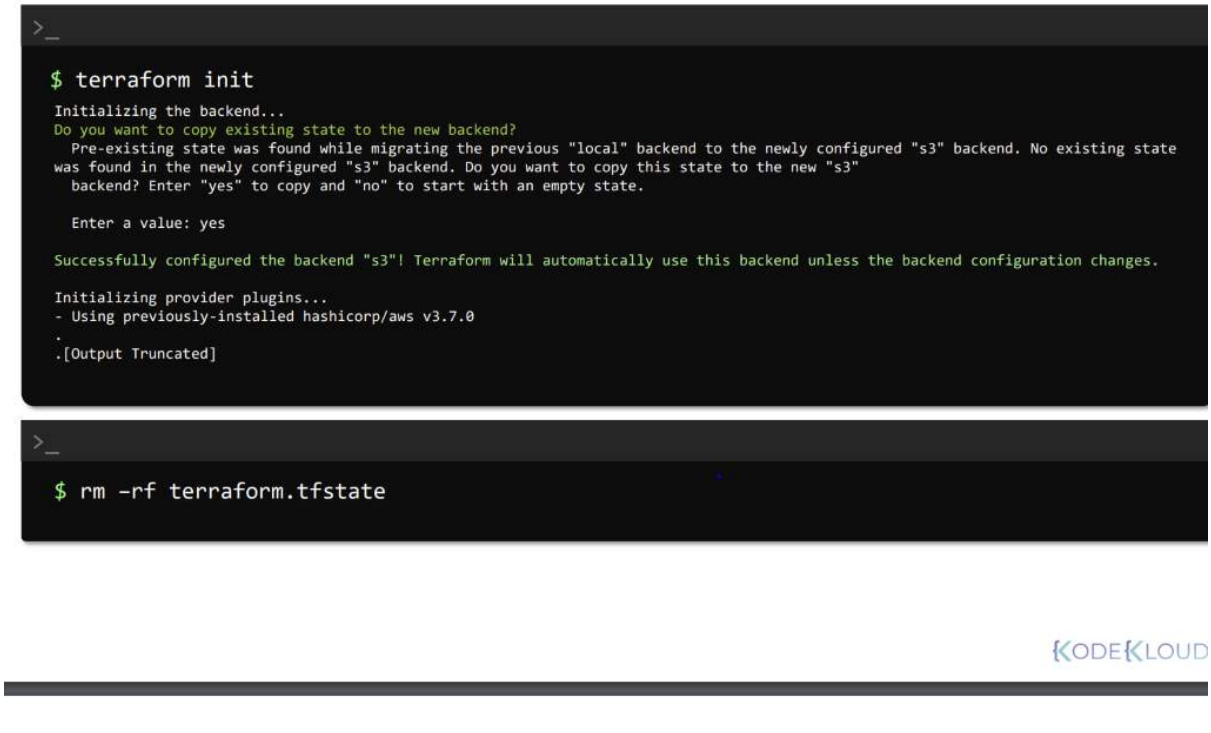
We need remote backend to store the terraform state file

Note : Finance folder should be present in s3 bucket



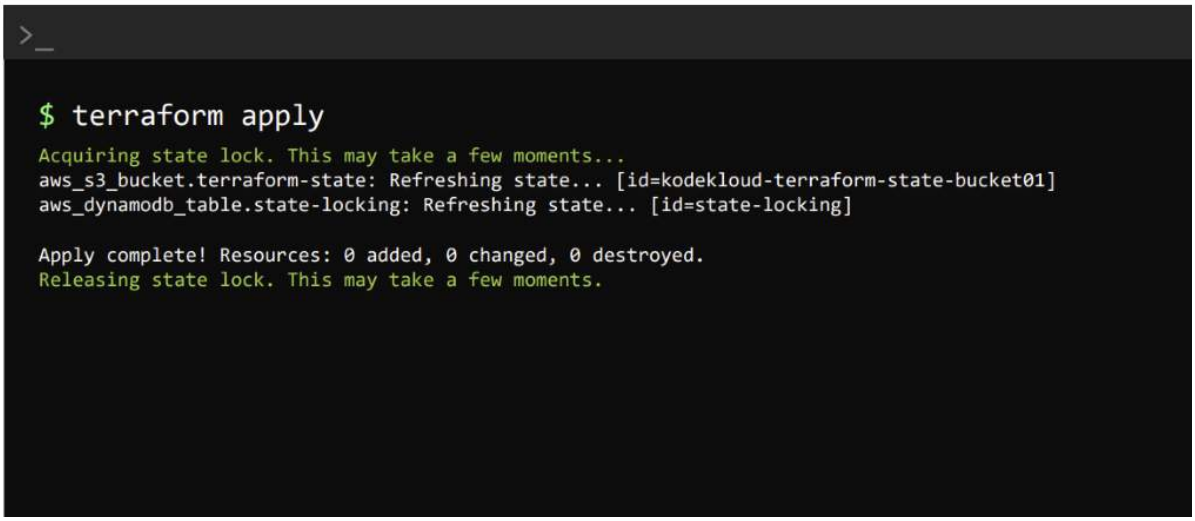


Terraform init : giving the value it will copy state file to s3 bucket



We can delete the local state file configuration

Running terraform apply now ill pull state file from s3 bucket in memory.
The state file will not store in local directory any more



Terraform state commands

Will learn how to list and manipulate state using the terraform state command.

Terraform.tfState file is not edited manually while opening in text editor

Terraform state sub command

First command : terraform state list

This will list all the resources recorded in terraform state file
We can also pass matching address in list . If it is available it will show

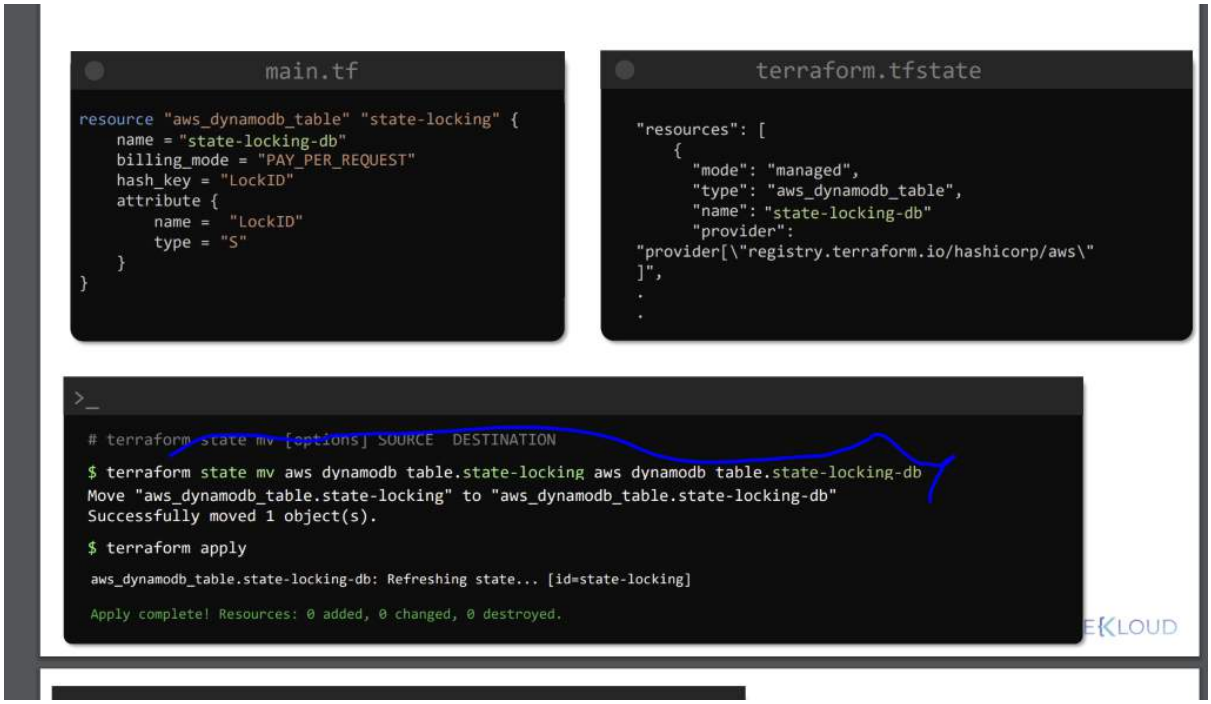
```
# terraform state list [options] [address]

$ terraform state list
aws_dynamodb_table.cars
aws_s3_bucket.finance-2020922

$ terraform state list aws_s3_bucket.finance-2020922
aws_s3_bucket.finance-2020922
```

To get detail information about resource from state file we can make use of **Terraform state show** command

This command will show the attribute of single resource that matching the given address

```
# terraform state show [options] [address]

$ terraform state show aws_s3_bucket.finance-2020922
resource "aws_s3_bucket" "terraform-state" {
    acl                         = "private"
    arn                         = "arn:aws:s3::: finance-2020922 "
    bucket                      = "finance-2020922 "
    bucket_domain_name          = "finance-2020922.s3.amazonaws.com"
    bucket_regional_domain_name = " finance-2020922.s3.us-west-1.amazonaws.com"
    force_destroy               = false
    hosted_zone_id              = "Z2F5ABCDE1ACD"
    id                          = "finance-2020922 "
    region                      = "us-west-1"
    request_payer               = "BucketOwner"
    tags                        = {
        "Descritpion" = "Bucket to store Finance and Payroll Information"
    }

    versioning {
        enabled    = false
        mfa_delete = false
    }
}
```

Terraform mv command



Terraform state pull command to **view state file from remote**

Terraform state rm : it is use to delete item from the state file

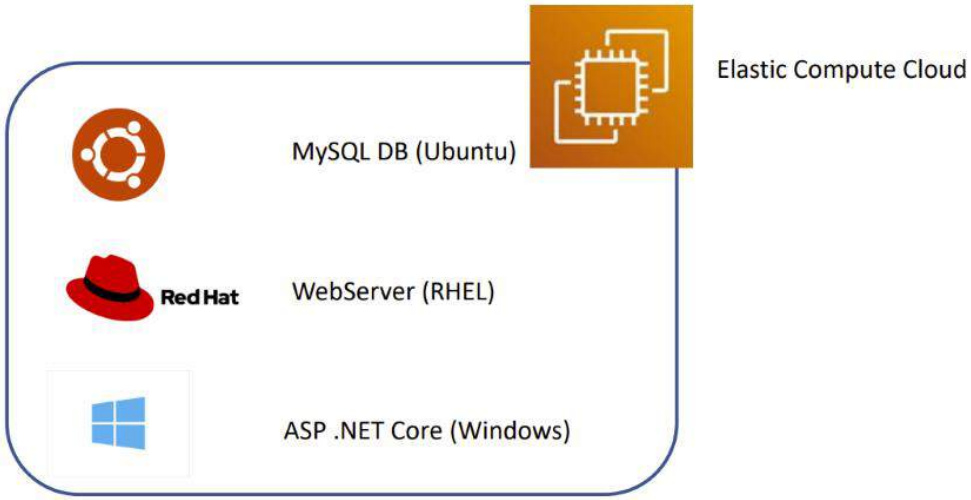
```
>_

#  terraform state rm ADDRESS

$ terraform state rm aws_s3_bucket.finance-2020922
Acquiring state lock. This may take a few moments...
Removed aws_s3_bucket.finance-2020922
Successfully removed 1 resource instance(s).
Releasing state lock. This may take a few moments...
```

It is only removed from terraform management

Section 9: terraform provisioners

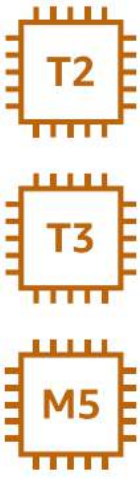
EC2 (elastic compute cloud) instances :
these virtual machine provides scalable compute cloud that can be deployed in a matter of minute

Aws ec2 provides predefined template such AS **AMI**



T2 General Purpose

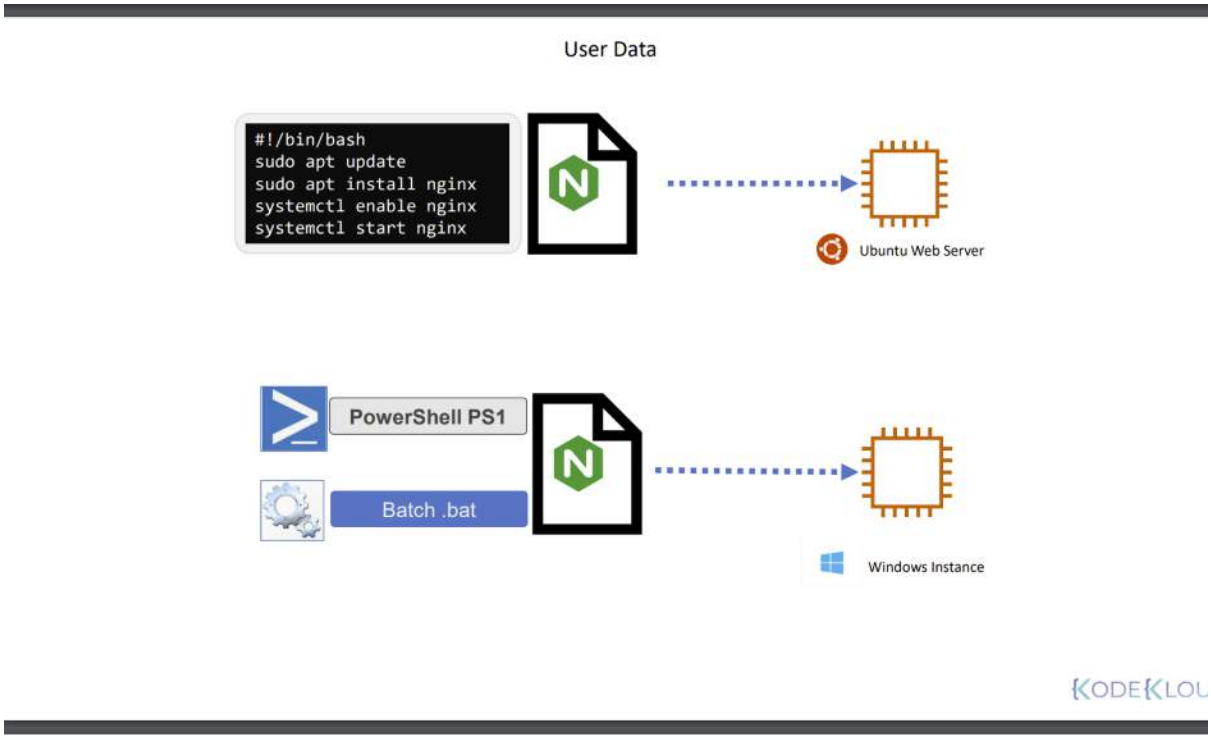
Instance Type	vCPU	Memory (GB)
t2.nano	1	0.5
t2.micro	1	1
t2.small	1	2
t2.medium	2	4
t2.large	2	8
t2.xlarge	4	16
t2.2xlarge	8	32

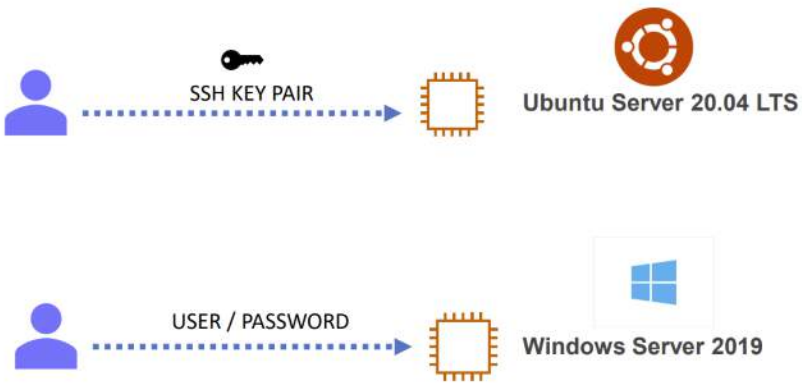


EBS : elastic block storage

EBS Volume

EBS Volume Types		
Name	Type	Description
io1	SSD	For business-critical Apps
io2	SSD	For latency-sensitive transactional workloads
gp2	SSD	General Purpose
st1	HDD	Low Cost HDD frequently accessed, throughput-intensive workloads
sc1	HDD	Lowest cost HDD volume designed for less frequently accessed workloads





Vpc : virtual private network in aws where we can deploy services

AWS EC2 with terraform

main.tf

```
resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"
  tags = {
    Name        = "webserver"
    Description = "An Nginx WebServer on Ubuntu"
  }

}
```

provider.tf

```
provider "aws" " {
  region = “us-west-1"
}
```

Argument Reference

The following arguments are supported:

- `ami` - (Required) The AMI to use for the instance.
- `instance_type` - (Required) The type of instance to start. Updates to this field will trig
  a stop/start of the EC2 instance.
- `tags` - (Optional) A map of tags to assign to the resource.

KODEKLOUD

main.tf

```
resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"
  tags = {
    Name        = "webserver"
    Description = "An Nginx WebServer on Ubuntu"
  }
  user_data = <<-EOF
              #!/bin/bash
              sudo apt update
              sudo apt install nginx -y
              systemctl enable nginx
              systemctl start nginx
              EOF
}
```

provider.tf

```
provider "aws" " {
  region = “us-west-1"
}
```

Argument Reference

The following arguments are supported:

- `ami` - (Required) The AMI to use for the instance.
- `instance_type` - (Required) The type of instance to start. Updates to this field will trig
  a stop/start of the EC2 instance.
- `tags` - (Optional) A map of tags to assign to the resource.
- `user_data` - (Optional) The user data to provide when launching the instance. Do n
  pass gzip-compressed data via this argument; see `user_data_base64` instead.

KODEKLOUD

>_

```
$ terraform apply

  # aws_instance.webserver will be created
  + resource "aws_instance" "webserver" {
      + ami                          = "ami-0edab43b6fa892279"
.
.
      + instance_type                = "t2.micro"
      + ipv6_address_count           = (known after apply)
      + public_ip                    = (known after apply)
      + source_dest_check            = true
      + subnet_id                    = (known after apply)
      + tags                         = {
          + "Description" = "An NGINX WebServer on Ubuntu"
          + "Name"        = "webserver"
        }
      + tenancy                      = (known after apply)
      + user_data                    = "527516162d9d8675a26b6ca97664226e6e2bff82"
      + volume_tags                  = (known after apply)
      + vpc_security_group_ids       = (known after apply)
.
.
aws_instance.webserver: Creating...
aws_instance.webserver: Still creating... [20s elapsed]
aws_instance.webserver: Creation complete after 22s [id=i-0085e5d0f442f7c4f]

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

Name	Instance ID	Instance state	Instance type	Status check	Alarm Status	Availability zone

main.tf

```
resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"
  tags = {
    Name        = "webserver"
    Description = "An Nginx WebServer on Ubuntu"
  }
  user_data = <<-EOF
              #!/bin/bash
              sudo apt update
              sudo apt install nginx -y
              systemctl enable nginx
              systemctl start nginx
              EOF
}
resource "aws_key_pair" "web" {
    public_key = file("/root/.ssh/web.pub")
}
```

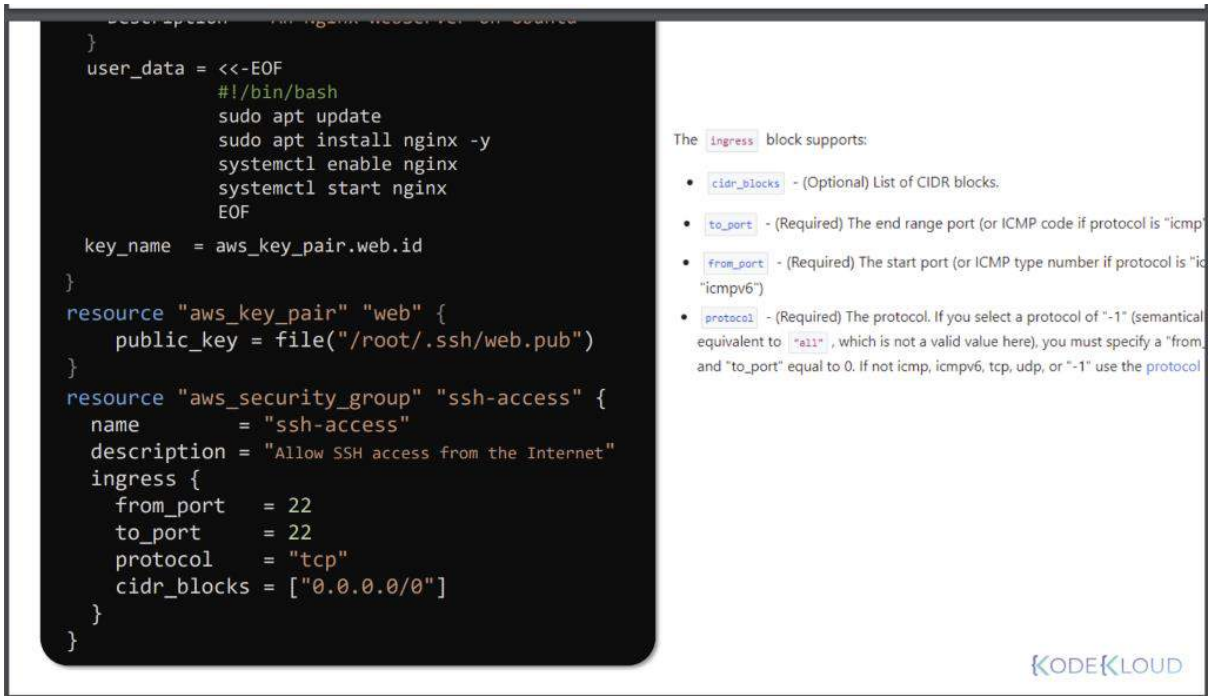
main.tf

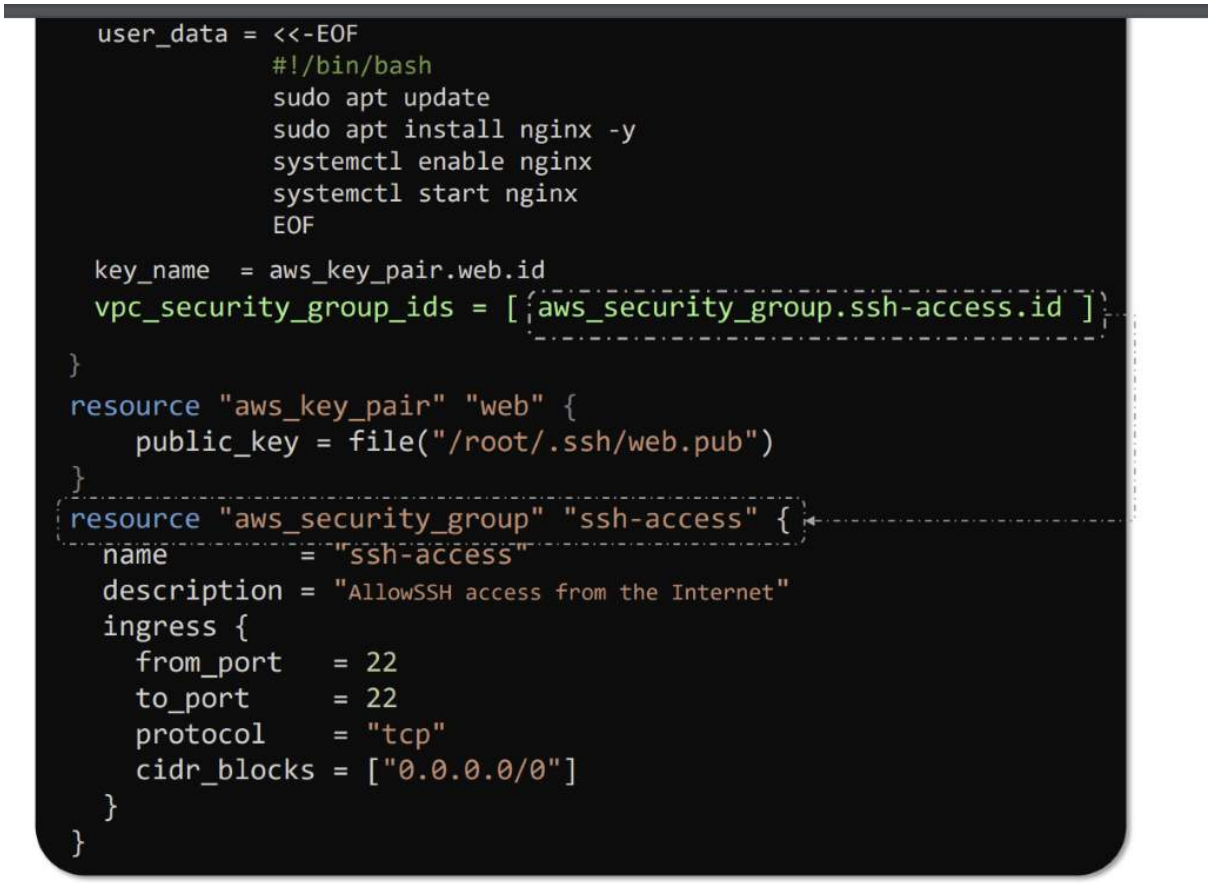
```
resource "aws_instance" "webserver" {
    ami = "ami-0edab43b6fa892279"
    instance_type = "t2.micro"
    tags = {
        Name = "webserver"
        Description = "An NGINX WebServer on Ubuntu"
    }
    user_data =<<-EOF
    #!/bin/bash
    sudo apt update
    sudo apt install nginx -y
    systemctl enable nginx
    systemctl start nginx
    EOF
}
resource "aws_key_pair" "web" {
    public_key = "ssh-
rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDicpU+kT9isaZy7cHYa
+oCTUolS6Tg6vCEq+ufucIMrA7RLTngi+YfTfvgrY2UiHGxuuJ1lE
yT0x2UrGexVx4G2TzX/am2WFzNbcGSg2bCXTkVQY93KOhbW9y851a
+g1wI7TODC0oxEMFr/CVsrJ4bfbp8S896VKBxC1WpSU9GscPP28GV
uDgm2ATBuL78AF root@iac-server"
}
```

KODEKLOUD

main.tf

```
resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"
  tags = {
    Name        = "webserver"
    Description = "An Nginx WebServer on Ubuntu"
  }
  user_data = <<-EOF
              #!/bin/bash
              sudo apt update
              sudo apt install nginx -y
              systemctl enable nginx
              systemctl start nginx
              EOF
  key_name  = aws_key_pair.web.id
}
resource "aws_key_pair" "web" {
    public_key = file("/root/.ssh/web.pub")
}
```


}
user_data = <<-EOF
#!/bin/bash
sudo apt update
sudo apt install nginx -y
systemctl enable nginx
systemctl start nginx
EOF
key_name = aws_key_pair.web.id
}
resource "aws_key_pair" "web" {
public_key = file("/root/.ssh/web.pub")
}
resource "aws_security_group" "ssh-access" {
name = "ssh-access"
description = "Allow SSH access from the Internet"
ingress {
from_port = 22
to_port = 22
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
}
The ingress block supports:
cidr_blocks - (Optional) List of CIDR blocks.
to_port - (Required) The end range port (or ICMP code if protocol is "icmp
from_port - (Required) The start port (or ICMP type number if protocol is "ic
"icmpv6")
protocol - (Required) The protocol. If you select a protocol of "-1" (semantical
equivalent to "all", which is not a valid value here), you must specify a "from_
and "to_port" equal to 0. If not icmp, icmpv6, tcp, udp, or "-1" use the protocol
KODEKLOUD


user_data = <<-EOF
#!/bin/bash
sudo apt update
sudo apt install nginx -y
systemctl enable nginx
systemctl start nginx
EOF
key_name = aws_key_pair.web.id
vpc_security_group_ids = [aws_security_group.ssh-access.id]
}
resource "aws_key_pair" "web" {
public_key = file("/root/.ssh/web.pub")
}
resource "aws_security_group" "ssh-access" {
name = "ssh-access"
description = "AllowSSH access from the Internet"
ingress {
from_port = 22
to_port = 22
protocol = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
}

```
              systemctl start nginx
              EOF

  key_name   = aws_key_pair.web.id
  vpc_security_group_ids = [ aws_security_group.ssh-access.id ]

}
resource "aws_key_pair" "web" {
    public_key = file("/root/.ssh/web.pub")
}
resource "aws_security_group" "ssh-access" {
  name        = "ssh-access"
  description = "AllowSSH access from the Internet"
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
 output publicip {
   value       = aws_instance.webserver.public_ip
 }
```

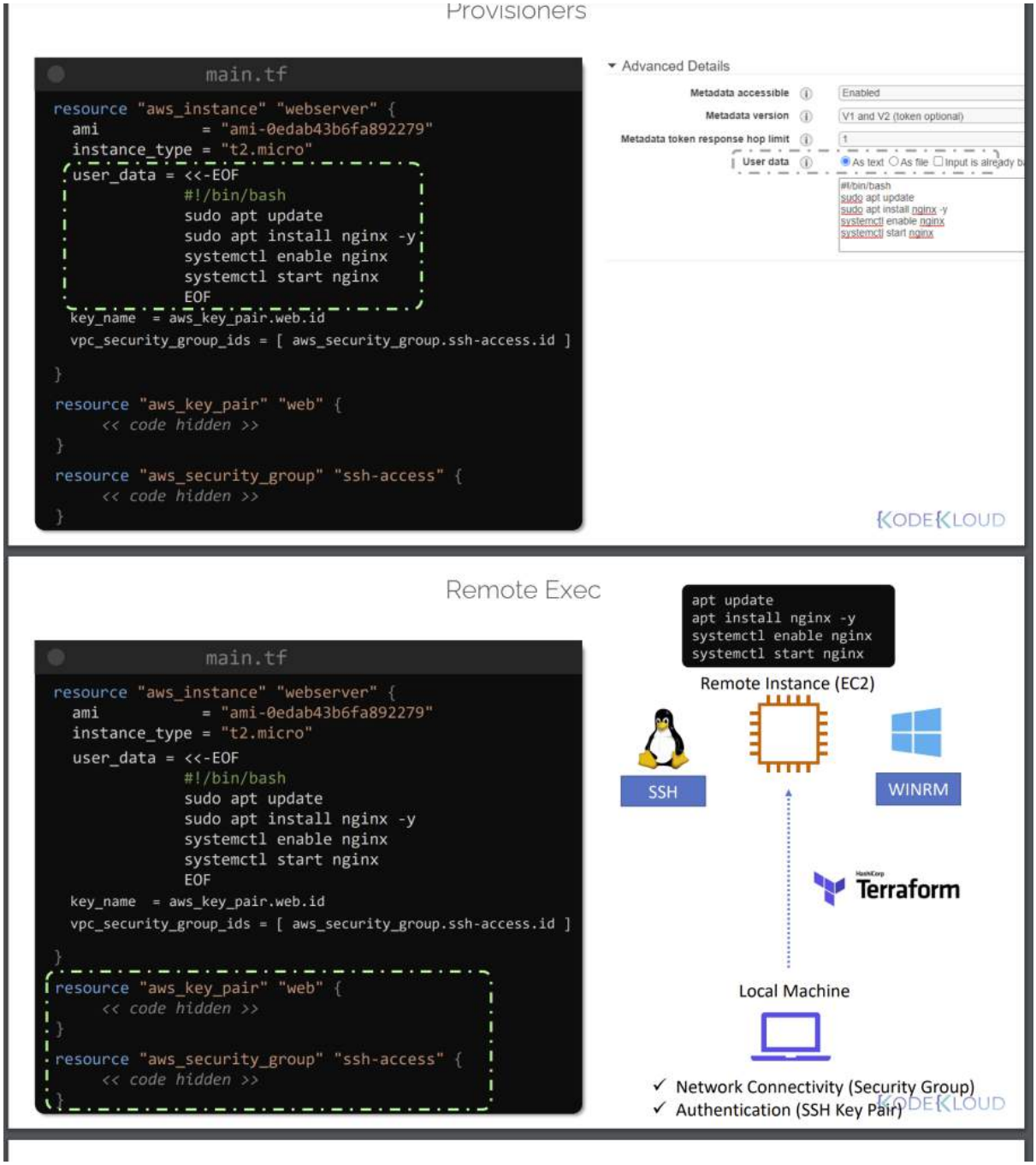
```
>_

$ terraform apply

Plan: 3 to add, 0 to change, 1 to destroy.

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

  Enter a value: yes

aws_instance.webserver: Destroying... [id=i-015b579d0ea84fbb7]
aws_key_pair.web: Creating...
aws_security_group.ssh-access: Creating...
aws_key_pair.web: Creation complete after 1s [id=terraform-
20201014034144926200000001]
aws_security_group.ssh-access: Creation complete after 1s [id=sg-
0f02f3ea92b14bed8]
aws_instance.webserver: Still destroying... [id=i-015b579d0ea84fbb7, 10s elapsed]
aws_instance.webserver: Still destroying... [id=i-015b579d0ea84fbb7, 20s elapsed]
aws_instance.webserver: Destruction complete after 30s
aws_instance.webserver: Creating...
aws_instance.webserver: Still creating... [10s elapsed]
aws_instance.webserver: Still creating... [20s elapsed]
aws_instance.webserver: Still creating... [30s elapsed]
aws_instance.webserver: Creation complete after 32s [id=i-0fd2c1c5eb0762ff5]

Apply complete! Resources: 3 added, 0 changed, 1 destroyed.
Outputs:

publicip = 3.96.203.171
```

Terraform provisioners

Terraform Provisioners are used for executing scripts or shell commands on a local or remote machine as part of resource creation/deletion.

Note :

There should be a network connectivity between local machine and remote for provisioner to work

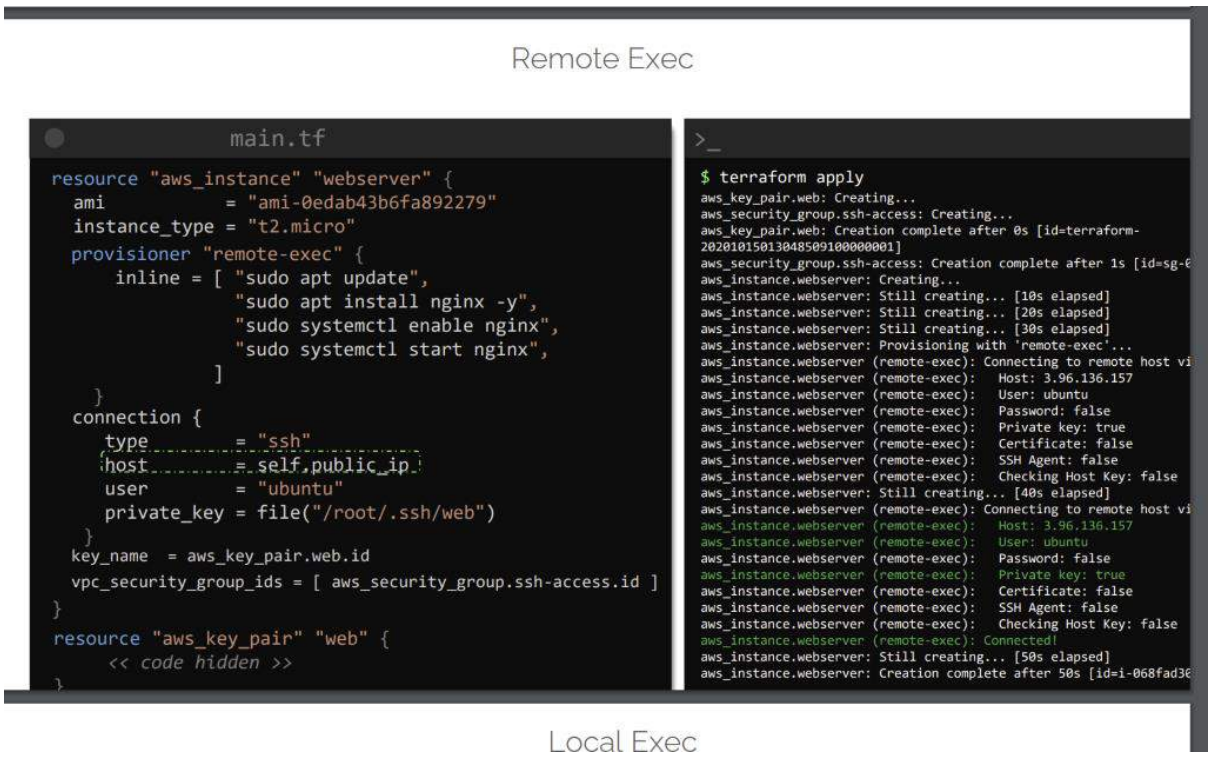


Remote Exec

main.tf

```
resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"
  provisioner "remote-exec" {
      inline = [ "sudo apt update",
                 "sudo apt install nginx -y",
                 "sudo systemctl enable nginx",
                 "sudo systemctl start nginx",
               ]
    }
  key_name  = aws_key_pair.web.id
  vpc_security_group_ids = [ aws_security_group.ssh-access.id ]
}
resource "aws_key_pair" "web" {
     << code hidden >>
}
resource "aws_security_group" "ssh-access" {
     << code hidden >>
}
```

Configuring ssh



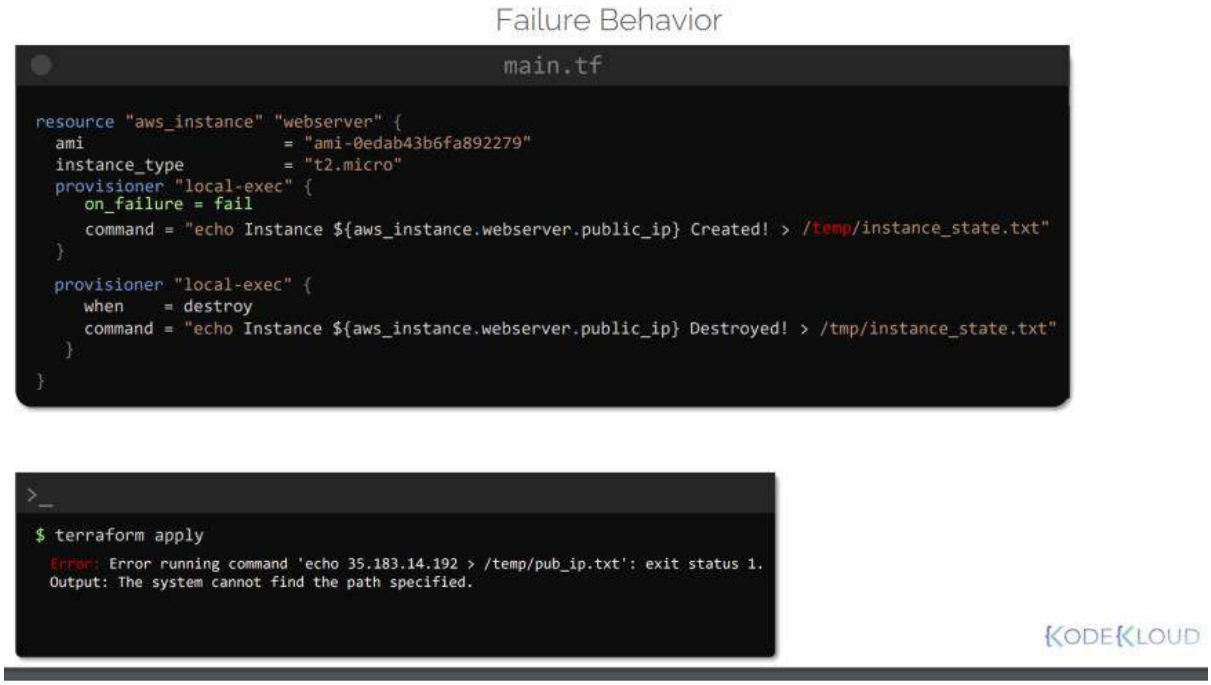
Local exec

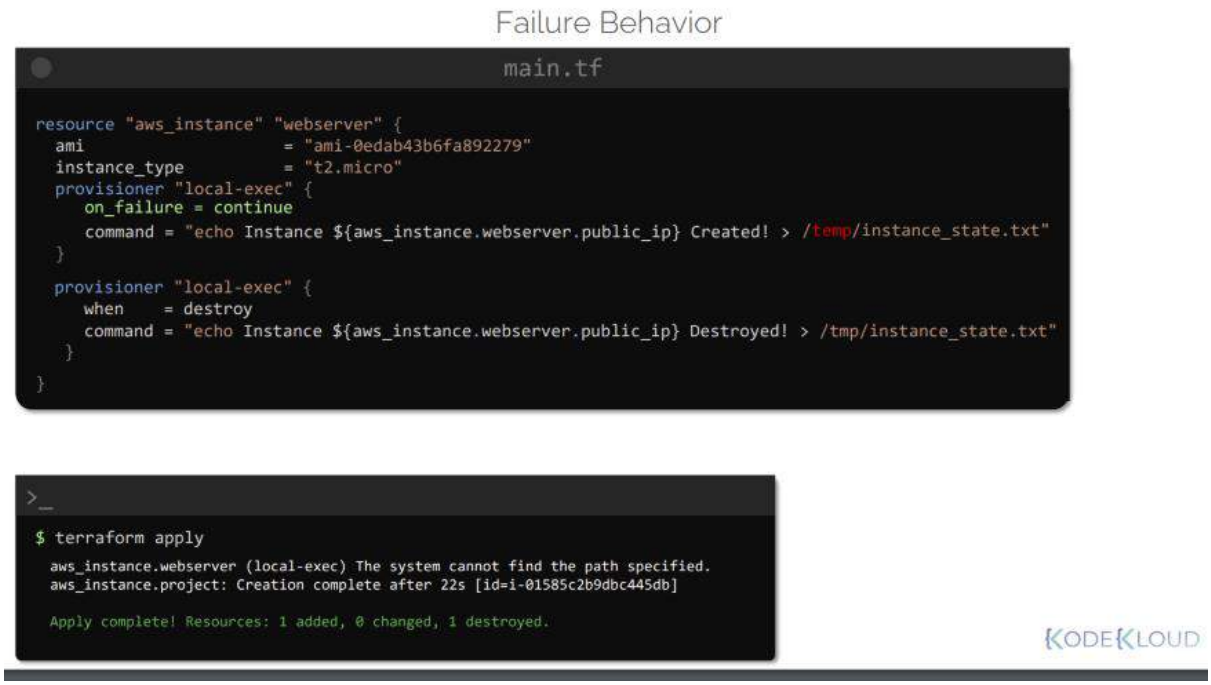
By default provisioners run after the resource is created and it is called **create time provisioner**

We can also run provisioner before the resource is destroyed

If the command within the provisioner fails the terraform apply command will als o show error

If we don't want to resource failed for creating even the provisioner command is failed we can set
OnFailure= continue





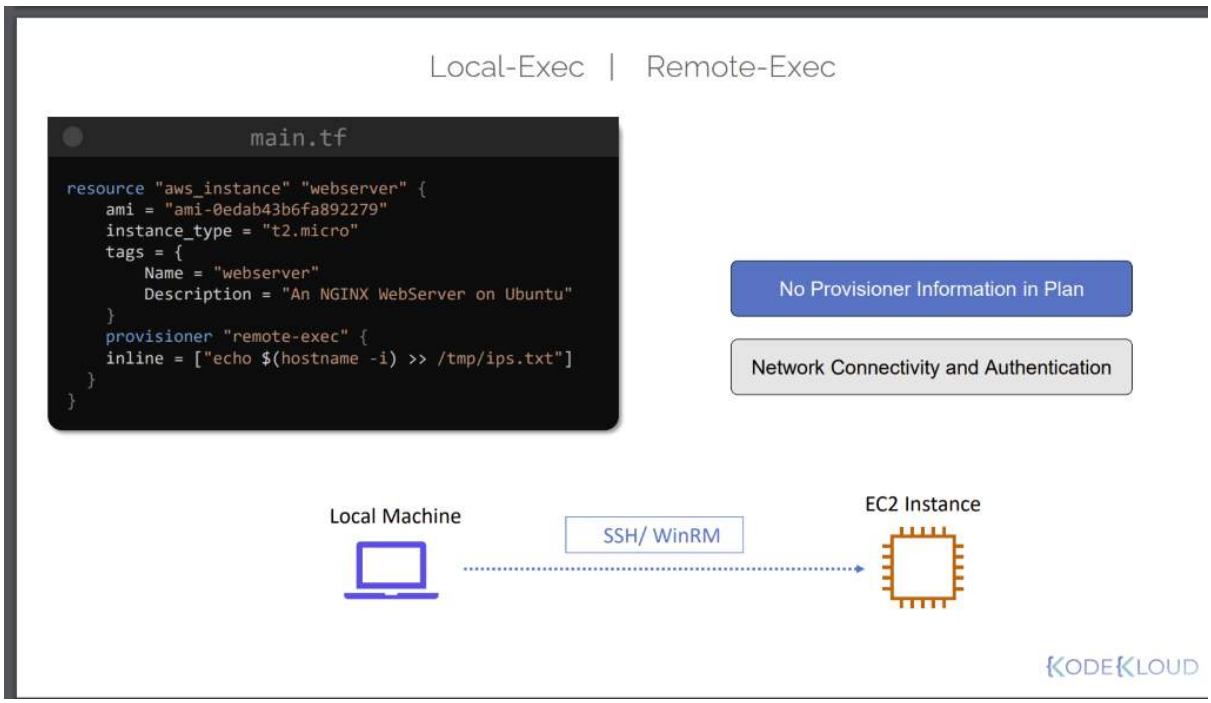
Provisioner behavior

Different behavior of provisioner

Creation time provisioner
Destroy time behaviour
Failure behaviour

Local exec and remote exec provisioner we use when resource is created

Consideration with provisioner

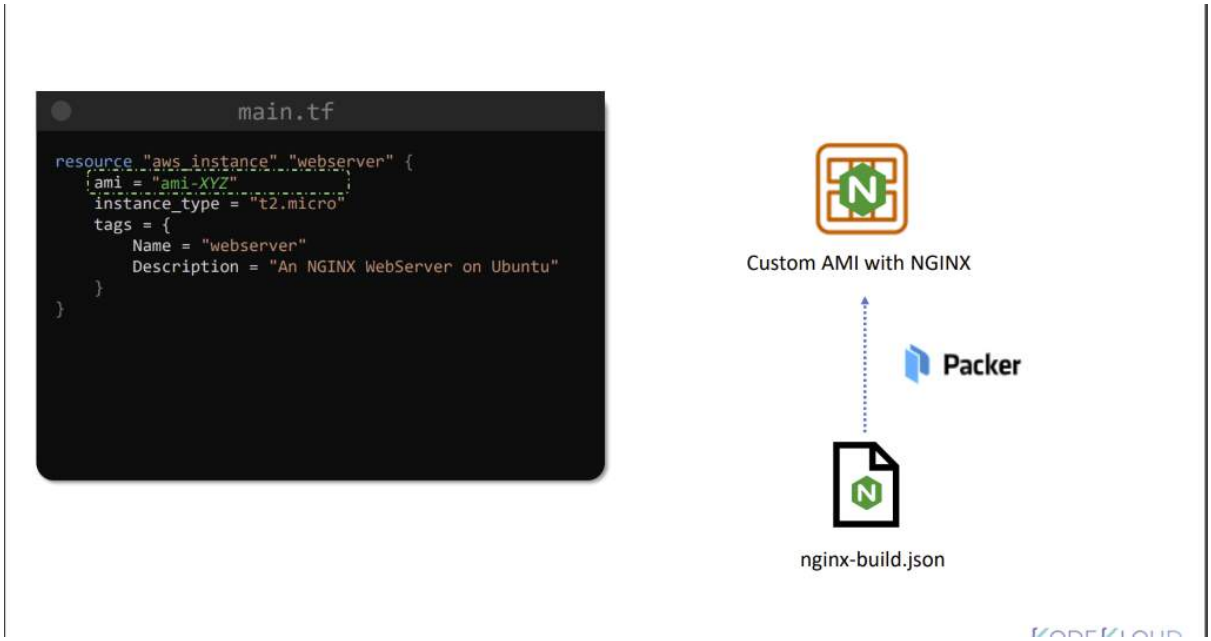


User data run during the instance launch time without having to define the connection block

Provider	Resource	Option
AWS	aws_instance	user_data
Azure	azurrerm_virtual_machine	custom_data
GCP	google_compute_instance	meta_data
Vmware vSphere	vsphere_virtual_machine	user_data.txt

Note :

We can use custom ami with ngx and there is no need of user data



Section 10: terraform import tainting resources and debugging

Terraform Taint
There is a cases when resource creation failed when we run terraform apply

When resource creation failed for any reason terraform mark the resources as **tainted**

Terraform Taint

Terraform maintains a state file that contains the information regarding the real-world resources managed by Terraform IaC. This is a crucial piece of information, as all the task executions of Terraform depend on this file for coordination.
As established earlier, when a resource becomes misconfigured or corrupt, it is desirable to replace them with a new instance. The taint command updates the corresponding resource state as a "tainted" resource so that in the next apply cycle, Terraform replaces that resource.
To improve your Terraform workflow, see the Terraform Best Practices.
Note: The taint command is deprecated since Terraform version 0.15.2. If you are using a version that is lower than this, continue using taint and untaint. Else, it is recommended to use the replace command discussed further

Taint

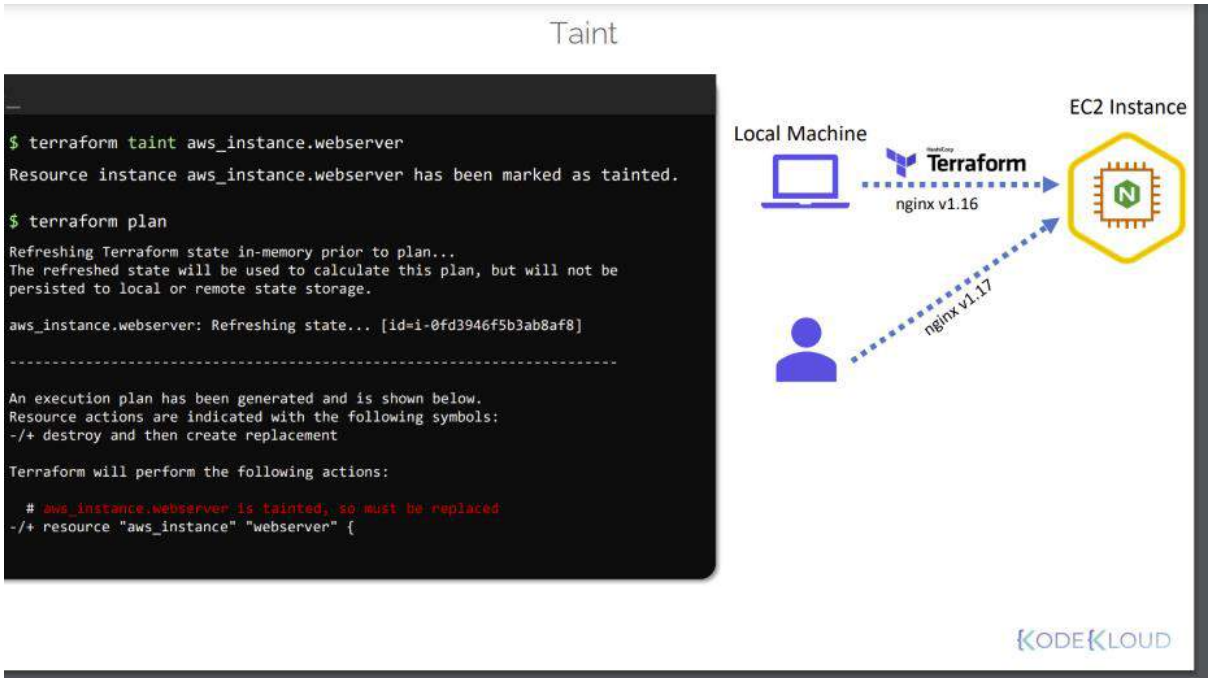
main.tf

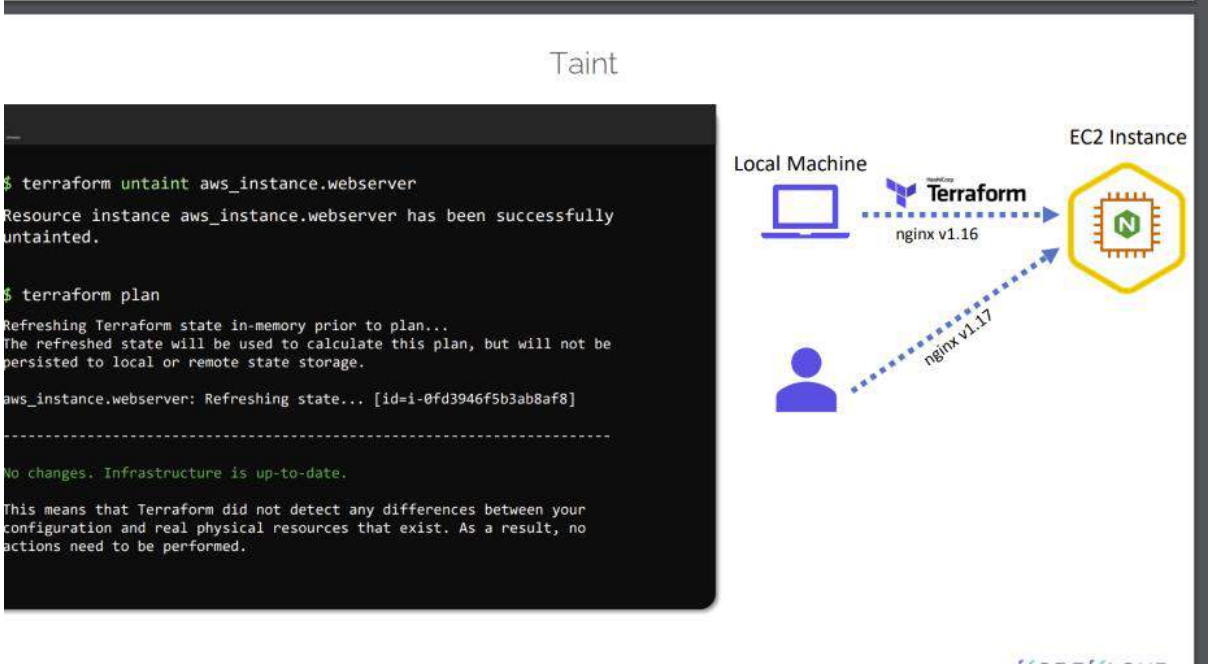
```
resource "aws_instance" "webserver-3" {
  ami                    = "ami-0edab43b6fa892279"
  instance_type          = "t2.micro"
  key_name               = "ws"
  provisioner "local-exec" {
    command = "echo ${aws_instance.webserver-3.public_ip} > /temp/pub_ip.txt"
  }
```

```
>_

$ terraform apply

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


aws_instance.webserver: Creating...
aws_instance.webserver: Still creating... [10s elapsed]
aws_instance.webserver: Still creating... [20s elapsed]
aws_instance.webserver: Still creating... [30s elapsed]
aws_instance.webserver: Provisioning with 'local-exec'...
aws_instance.webserver (local-exec): Executing: ["cmd" "/C" "echo 35.183.14.192 > /temp/pub_ip.txt"]
aws_instance.webserver (local-exec): The system cannot find the path specified.


Error: Error running command 'echo 35.183.14.192 > /temp/pub_ip.txt': exit status 1. Output: The system
cannot find the path specified.
```


Taint

```
>_

$ terraform plan

Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not
be
persisted to local or remote state storage.

aws_instance.webserver: Refreshing state... [id=i-0dba2d5dc22a9a904]

------------------------------------------------------------------------
-

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

Terraform will perform the following actions:

  # aws_instance.webserver is tainted, so must be replaced
-/+ resource "aws_instance" "webserver-3" {
```





Debugging

How to enable and debugging in terraform

```
>_
# export TF_LOG=<log_level>
$ export TF_LOG=TRACE
```

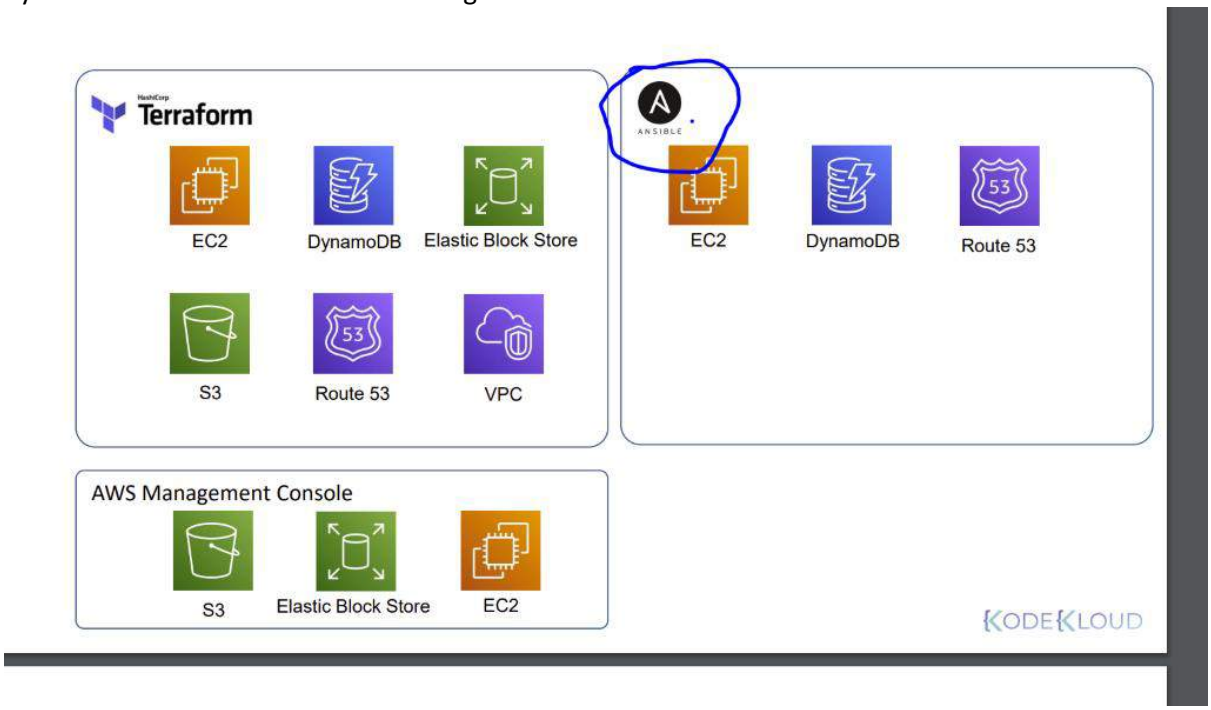
Terraform log

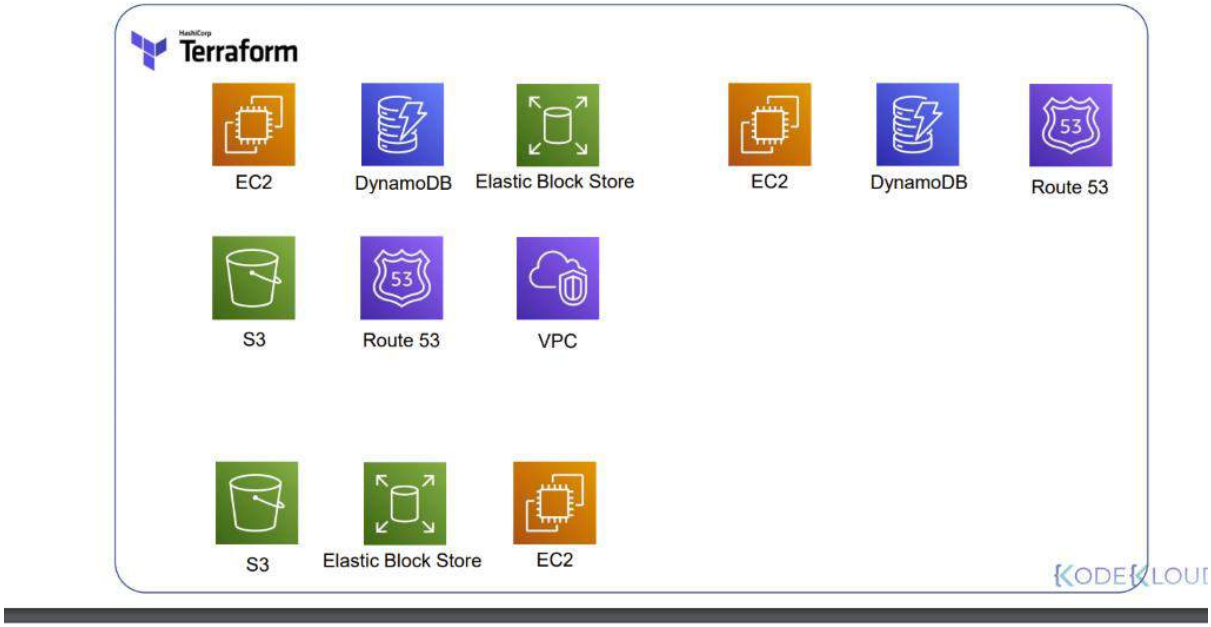
```
$ export TF_LOG_PATH=/tmp/terraform.log

$ head -10 /tmp/terraform.logs
----
2020/10/18 22:08:30 [INFO] Terraform version: 0.13.0
2020/10/18 22:08:30 [INFO] Go runtime version: go1.14.2
2020/10/18 22:08:30 [INFO] CLI args: []string{"C:\\Windows\\system32\\terraform.exe",
"plan"}
2020/10/18 22:08:30 [DEBUG] Attempting to open CLI config file:
C:\Users\vpala\AppData\Roaming\terraform.rc
2020/10/18 22:08:30 [DEBUG] File doesn't exist, but doesn't need to. Ignoring.
2020/10/18 22:08:30 [DEBUG] ignoring non-existing provider search directory
terraform.d/plugins
2020/10/18 22:08:30 [DEBUG] ignoring non-existing provider search directory
C:\Users\vpala\AppData\Roaming\terraform.d\plugins
2020/10/18 22:08:30 [DEBUG] ignoring non-existing provider search directory
C:\Users\vpala\AppData\Roaming\HashiCorp\Terraform\plugins
2020/10/18 22:08:30 [INFO] CLI command args: []string{"plan"}


$ unset TF_LOG_PATH
```

Terraform import

Will learn how to import existing infrastructure in terraform configuration using terraform import command

Sometimes resources is created by other resource like ansible but what if the aws resource created by other resource and we want to bring into the environment of terraform





How do we do that

Note
datasource :- instance_id is not managed by terraform we cannot update or delete using terraform

What is a data in Terraform?

Data sources in Terraform are **used to get information about resources external to Terraform, and use them to set up your Terraform resources**. For example, a list of IP addresses a cloud provider exposes.

Note :

To bring resources completely in terraform we have to import it and for that we can make use of **terraform import command**

Note : terraform import ----It only update the state file

Terraform Import

```
>_
# terraform import  <resource_type>.<resource_name> <attribute>
$ terraform import aws_instance.webserver-2 i-026e13be10d5326f7

Error: resource address "aws_instance.webserver-2" does not exist
in the configuration.

Before importing this resource, please create its configuration in
the root module. For example:

resource "aws_instance" "webserver-2" {
  # (resource arguments)
}
```

KODEKL

Terraform Import

main.tf

```
resource "aws_instance" "webserver-2" {
  # (resource arguments)
}
```

>_

```
$ terraform import aws_instance.webserver-2 i-026e13be10d5326f7

aws_instance.webserver-2: Importing from ID "i-026e13be10d5326f7"...
aws_instance.webserver-2: Import prepared!
  Prepared aws_instance for import
aws_instance.webserver-2: Refreshing state... [id=i-026e13be10d5326f7]

Import successful!

The resources that were imported are shown above. These resources are now in
your Terraform state and will henceforth be managed by Terraform.
```

terraform.tfstate

```
{
    "mode": "managed",
    "type": "aws_instance",
    "name": "webserver-2",
    "provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
    "instances": [
      {
        "schema_version": 1,
        "attributes": {
          "ami": "ami-0edab43b6fa892279",
          "instance_state": "running",
          "instance_type": "t2.micro",
          "key_name": "ws",
          .
          "tags": {
            "Name": "old-ec2"
          },
          .
          .
          "vpc_security_group_ids": [
            "sg-8064fdee"
          ]
        },
          .
          .
      }
    ]
  },
```

main.tf

```
resource "aws_instance" "webserver-2" {
    ami                    = "ami-0edab43b6fa892279"
    instance_type          = "t2.micro"
    key_name               = "ws"
    vpc_security_group_ids = ["sg-8064fdee"]

}
```

```
$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

aws_instance.webserver-2: Refreshing state... [id=i-0d7c0088069819ff8]

------------------------------------------------------------------------

No changes. Infrastructure is up-to-date.

This means that Terraform did not detect any differences between your
configuration and real physical resources that exist. As a result, no
actions need to be performed.
```

Section 11: terraform modules

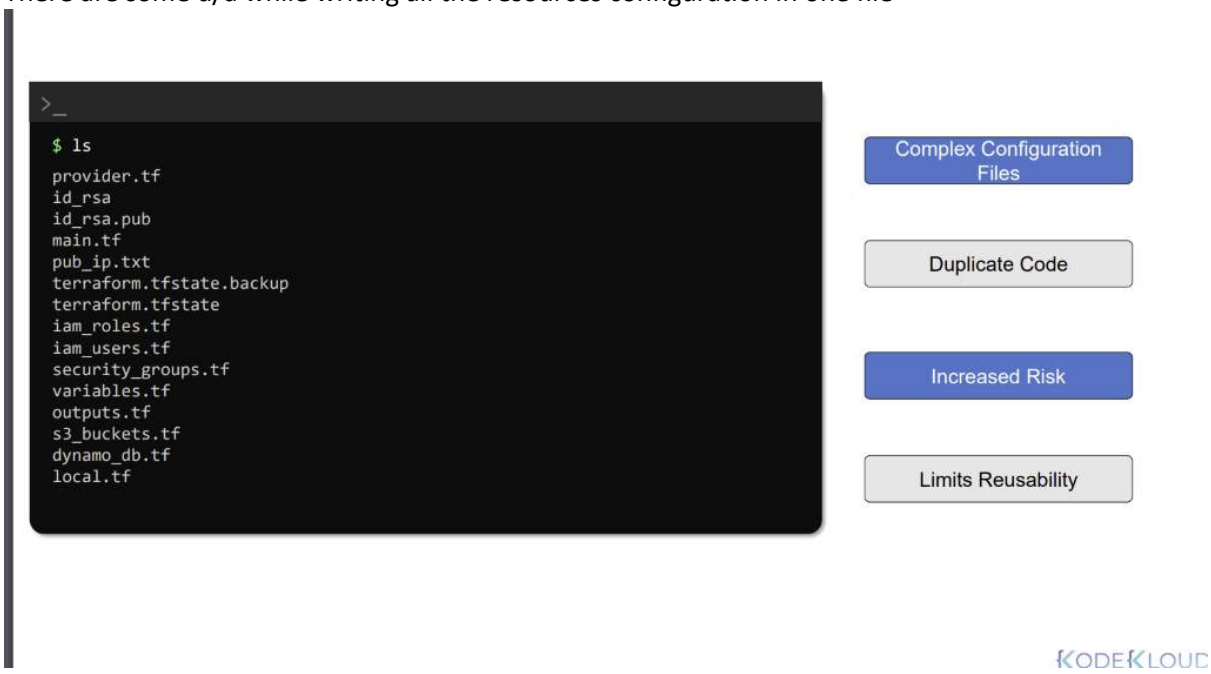
What is the module and how to use in terraform configuration

Note :
In two ec2 instance the code will be much similar
Terraform does not import any limit on number of resources per configuration file

Alternatively we can split configuration file into multiple files and it work into the same way

We saw that terraform will consider every file in directory as long as it has .tf extension

There are some d/a while writing all the resources configuration in one file



These problem can be addressed by making **modules in terraform**
Any configuration directory that contain a set of configuration is called **modules**

Root Module

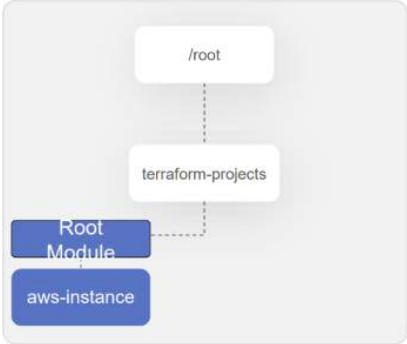
```
>_
$ ls /root/terraform-projects/aws-instance
main.tf      variables.tf
```

main.tf

```
resource "aws_instance" "webserver" {
    ami = var.ami
    instance_type = var.instance_type
    key_name = var.key
}
```

variables.tf

```
variable ami {
  type        = string
  default     = "ami-0edab43b6fa892279"
  description = "Ubuntu AMI ID in the ca-
central-1 region"
}
```



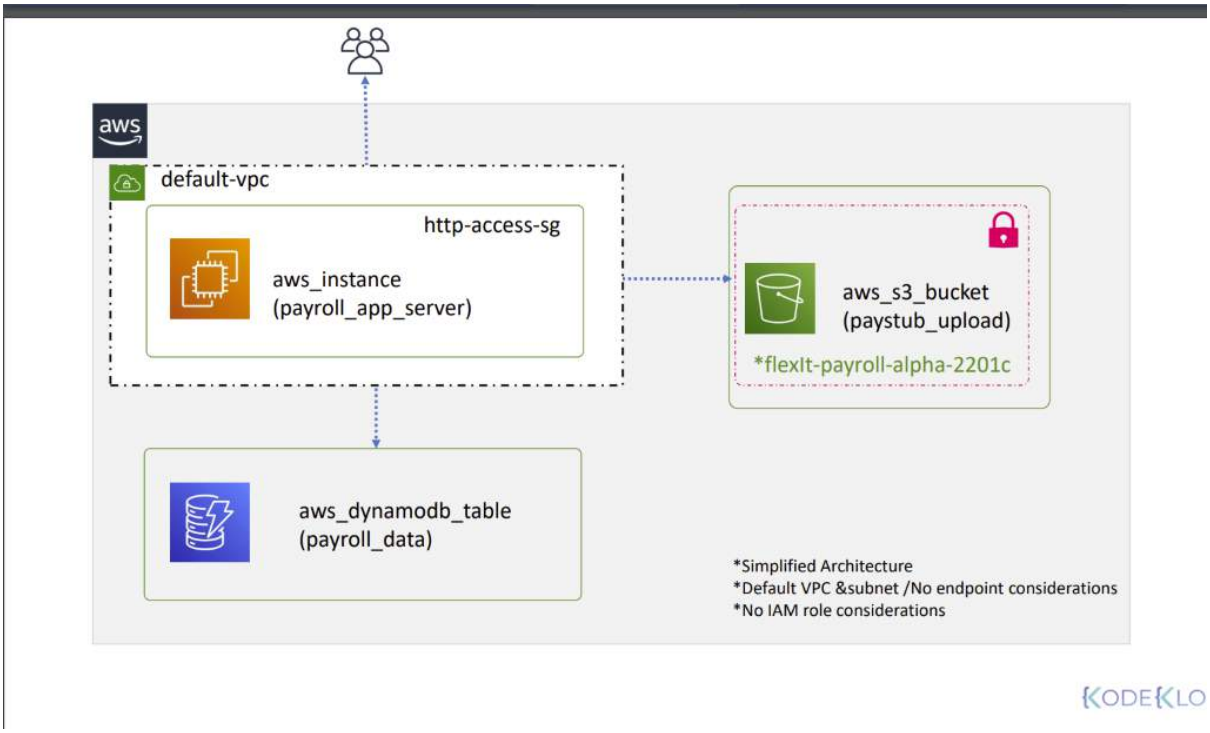
Creating and using a module

Requirement : to redploy the same stack in different countries using terraform module

A module is **a container for multiple resources that are used together**. Every Terraform configuration has at least one module, known as its root module, which consists of the resources defined in the . tf files in the main working directory.

Why modules are used in Terraform?
a module allows you to group resources together and reuse this group later, possibly many times

Architecture

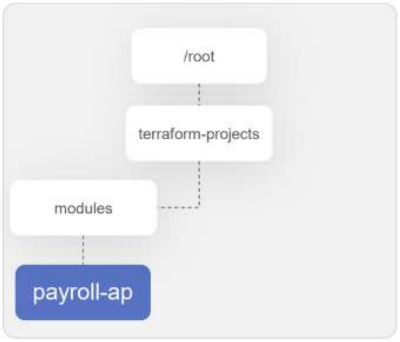

aws
default-vpc
http-access-sg
aws_instance
(payroll_app_server)
aws_s3_bucket
(paystub_upload)
*flexIt-payroll-alpha-2201c
aws_dynamodb_table
(payroll_data)
*Simplified Architecture
*Default VPC &subnet /No endpoint considerations
*No IAM role considerations

KODEKLO

```
>_
$ mkdir /root/terraform-projects/modules/payroll-app
app_server.tf dynamodb_table.tf s3_bucket.tf variables.tf
```

app_server.tf

```
resource "aws_instance" "app_server" {
  ami           = var.ami
  instance_type = "t2.medium"
  tags = {
    Name = "${var.app_region}-app-server"
  }
  depends_on = [ aws_dynamodb_table.payroll_db,
                 aws_s3_bucket.payroll_data
               ]
}
```



s3_bucket.tf

```
resource  "aws_s3_bucket"  "payroll_data" {
  bucket = "${var.app_region}-${var.bucket}"
}
```

dynamodb_table.tf

```
resource "aws_dynamodb_table" "payroll_db" {
  name           = "user_data"
  billing_mode   = "PAY_PER_REQUEST"
  hash_key       = "EmployeeID"

  attribute {
    name = "EmployeeID"
    type = "N"
  }
}
```

variables.tf

```
variable  "app_region" {
    type = string
}
variable "bucket" {
    default = "flexit-payroll-alpha-22001c"
}
variable "ami" {
    type = string
}
```

```
>_
$ mkdir /root/terraform-projects/us-payroll-app
main.tf provider.tf
```

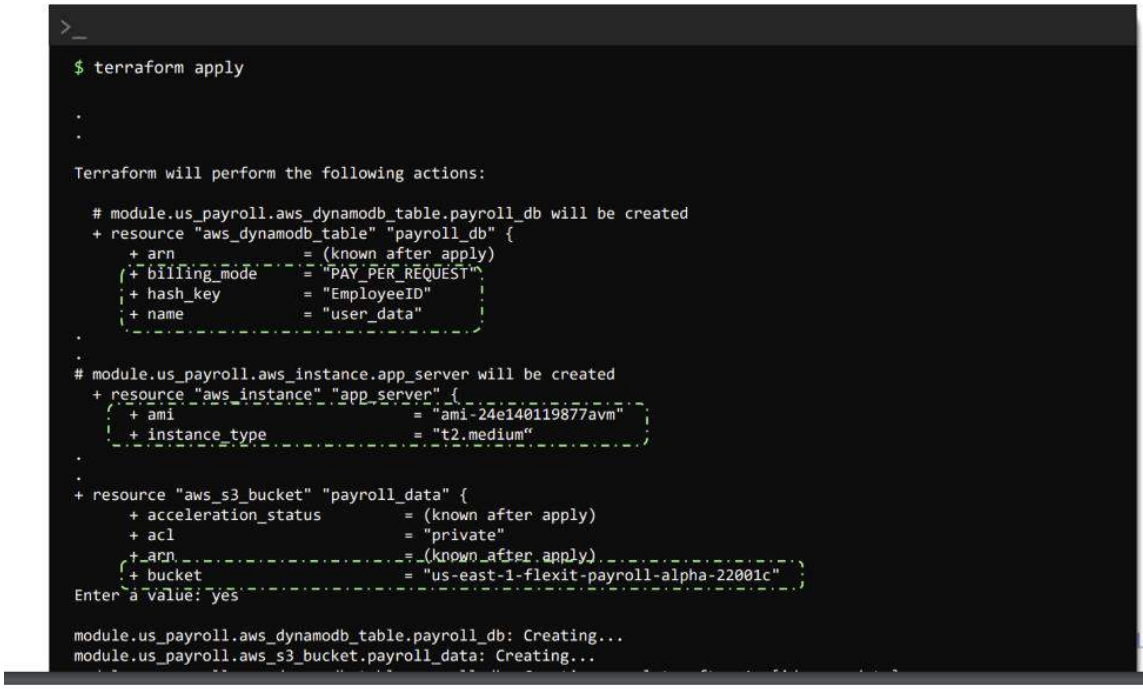
main.tf

```
module  "us_payroll" {
  source = "../modules/payroll-app"
  app_region = "us-east-1"
  ami        = "ami-24e140119877avm"
}
```

KODEKLOUD

```
>_
$ terraform init

Initializing modules...
- us_payroll in .terraform/modules/us_payroll

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v3.11.0...
- Installed hashicorp/aws v3.11.0 (signed by HashiCorp)

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, we recommend adding version constraints in a required_providers
block
in your configuration, with the constraint strings suggested below.

* hashicorp/aws: version = "~> 3.11.0"

Terraform has been successfully initialized!
```

KODEKLOUD


$ terraform apply
Terraform will perform the following actions:
# module.us_payroll.aws_dynamodb_table.payroll_db will be created
+ resource "aws_dynamodb_table" "payroll_db" {
+ arn = (known after apply)
+ billing_mode = "PAY_PER_REQUEST"
+ hash_key = "EmployeeID"
+ name = "user_data"
# module.us_payroll.aws_instance.app_server will be created
+ resource "aws_instance" "app_server" {
+ ami = "ami-24e140119877avm"
+ instance_type = "t2.medium"
+ resource "aws_s3_bucket" "payroll_data" {
+ acceleration_status = (known after apply)
+ acl = "private"
+ arn = (known after apply)
+ bucket = "us-east-1-flexit-payroll-alpha-22001c"
Enter a value: yes
module.us_payroll.aws_dynamodb_table.payroll_db: Creating...
module.us_payroll.aws_s3_bucket.payroll_data: Creating...

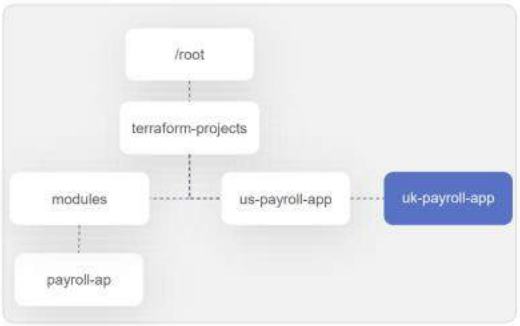
```
$ mkdir /root/terraform-projects/uk-payroll-app
main.tf provider.tf
```

main.tf

```
module  "uk_payroll" {
  source = "../modules/payroll-app"
  app_region = "eu-west-2"
  ami        = "ami-35e140119877avm"
}
```

provider.tf

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



KODEKLOUD

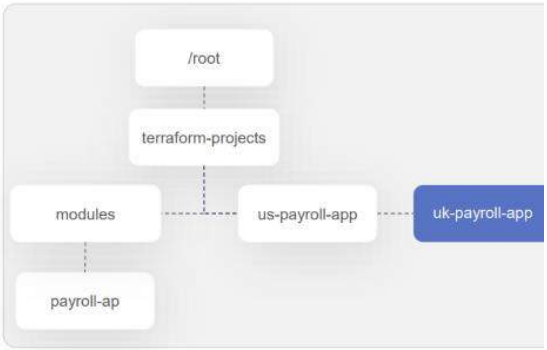
```
$ terraform apply
.
.
Terraform will perform the following actions:

  # module.us_payroll.aws_dynamodb_table.payroll_db will be created
  + resource "aws_dynamodb_table" "payroll_db" {
      + arn              = (known after apply)
      + billing_mode     = "PAY_PER_REQUEST"
      + hash_key         = "EmployeeID"
      + name             = "user_data"
.
.
# module.us_payroll.aws_instance.app_server will be created
  + resource "aws_instance" "app_server" {
      + ami                          = "ami-35e140119877avm"
      + instance_type                = "t2.medium"
.
.
+ resource "aws_s3_bucket" "payroll_data" {
      + acceleration_status         = (known after apply)
      + acl                         = "private"
      + arn                         = (known after apply)
      + bucket                      = "eu-west-2-flexit-payroll-alpha-22001c"
Enter a value: yes

module.us_payroll.aws_dynamodb_table.payroll_db: Creating...
module.us_payroll.aws_s3_bucket.payroll_data: Creating...
module.us_payroll.aws_dynamodb_table.payroll_db: Creation complete after 1s [id=user_data]
```

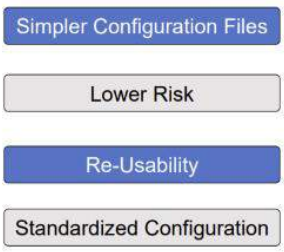
LOUD

```
.
.

Terraform will perform the following actions:

  # module.us_payroll.aws_dynamodb_table.payroll_db will be c
  + resource "aws_dynamodb_table" "payroll_db" {
      + arn              = (known after apply)
      + billing_mode     = "PAY_PER_REQUEST"
      + hash_key         = "EmployeeID"
      + name             = "user_data"
.
.
# module.us_payroll.aws_instance.app_server will be created
  + resource "aws_instance" "app_server" {
      + ami                          = "ami-35e140119877avm"
```

main.tf

```
module  "us_payroll" {
  source = "../modules/payroll-app"
  app_region = "eu-west-2"
  ami        = "ami-35e140119877avm"
}
```





KODEKLOU

Same module can be use in different region which improve the reusability

Using modules from the registry

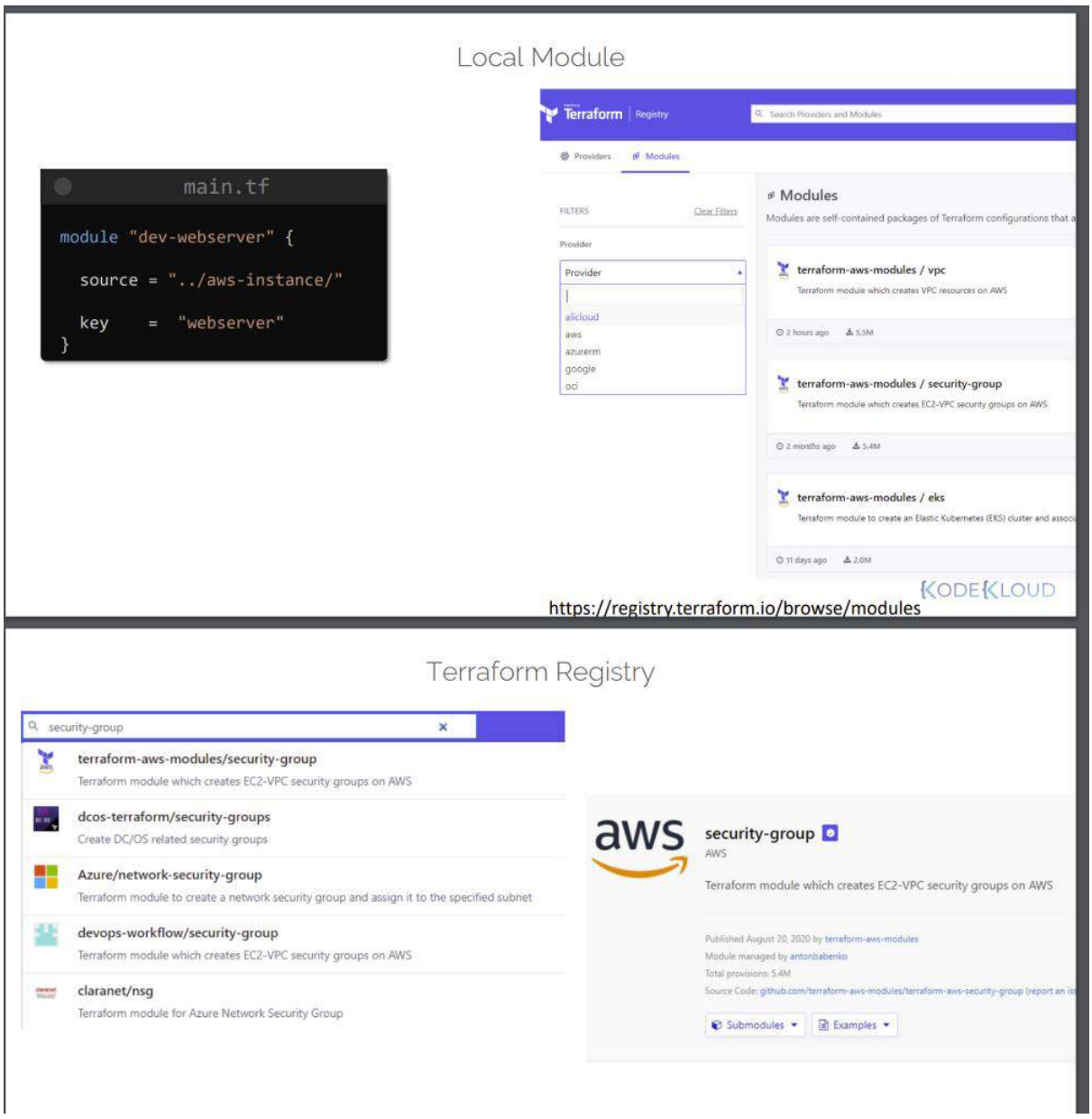
Will learn how to use module from terraform registry

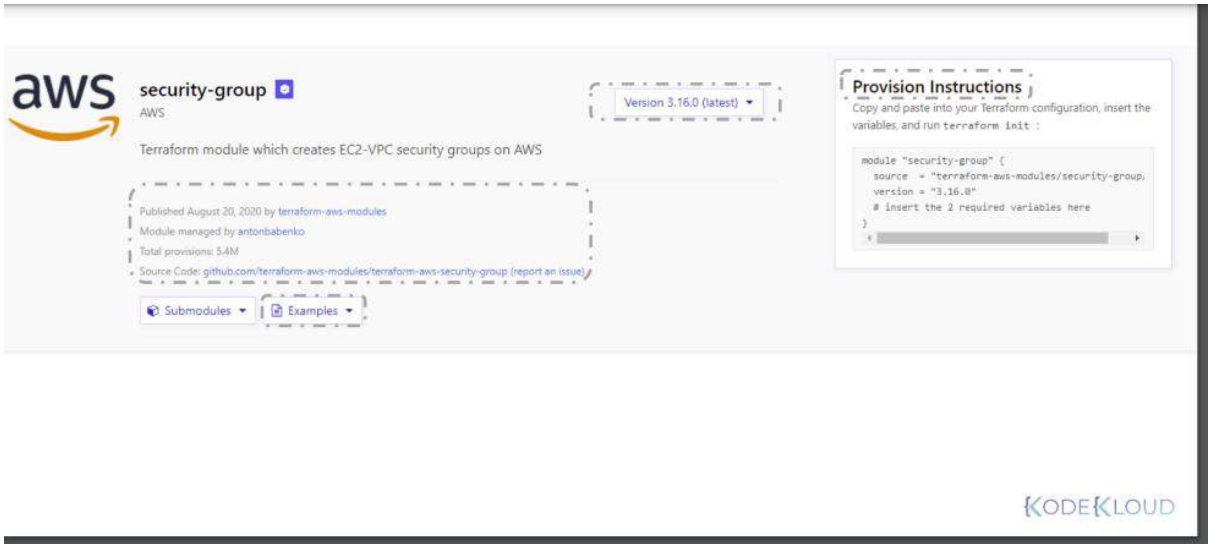
One of the advantages of modules : it can be easily shared with others

Terraform register is use to store provider plugin

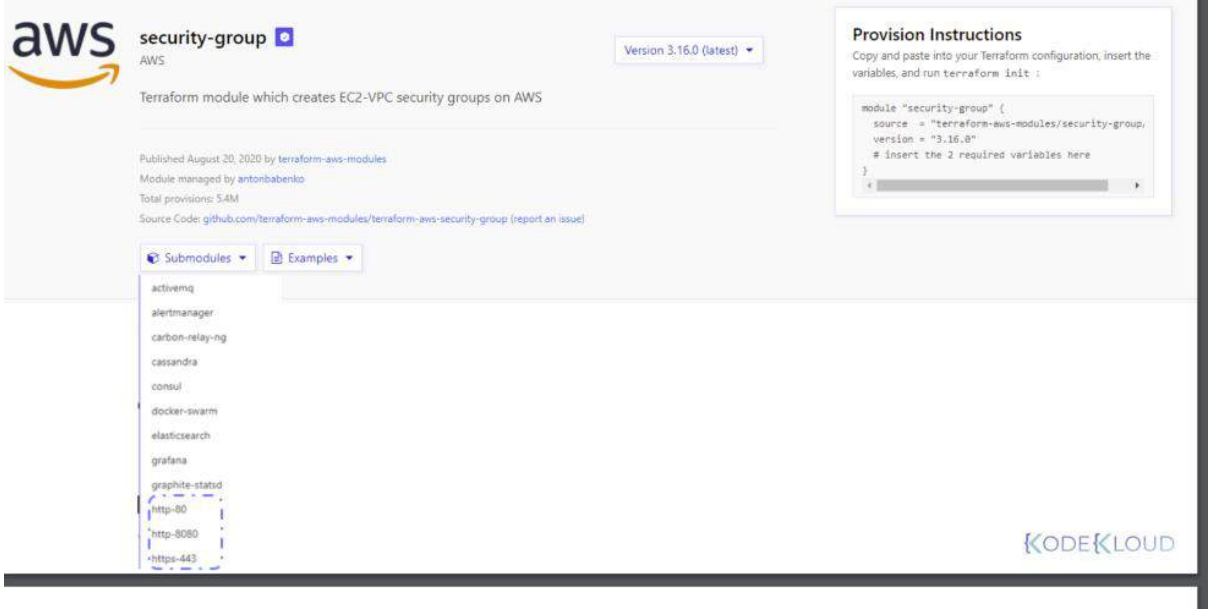
There will be verified module in registry which is tested and verified by hashicorp

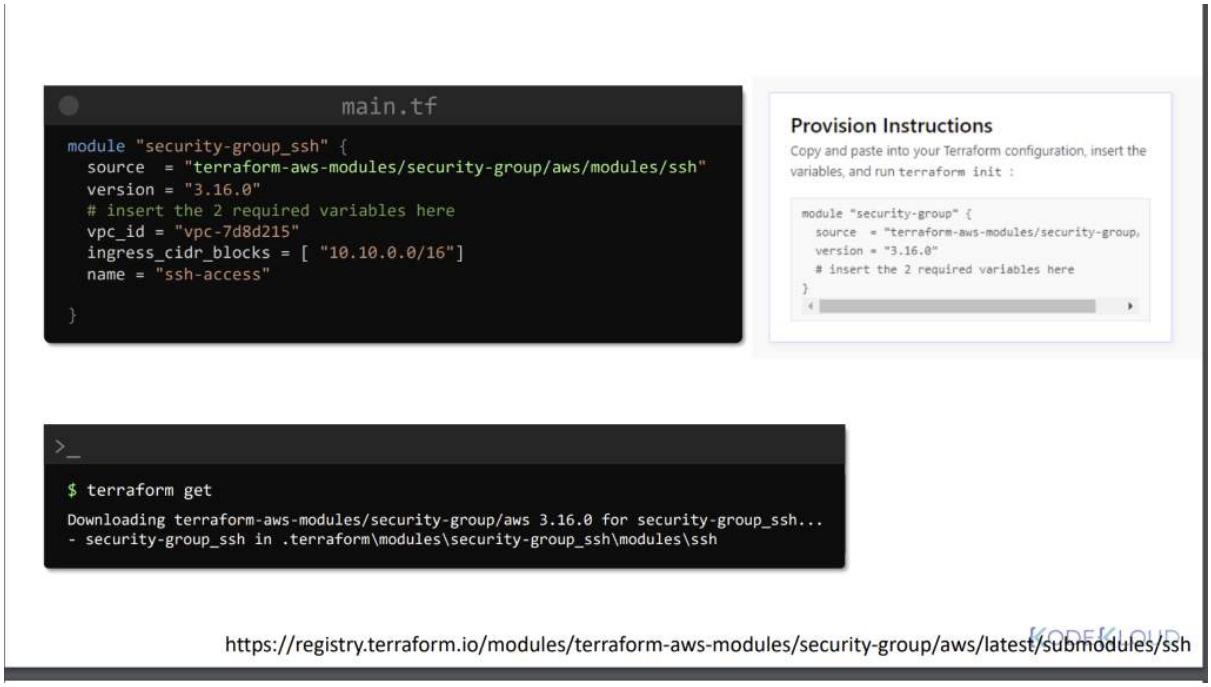
Not verified version will always download latest version from the registry





Terraform Module





Section 12: terraform functions and conditional expression

Will learn some more function in terraform

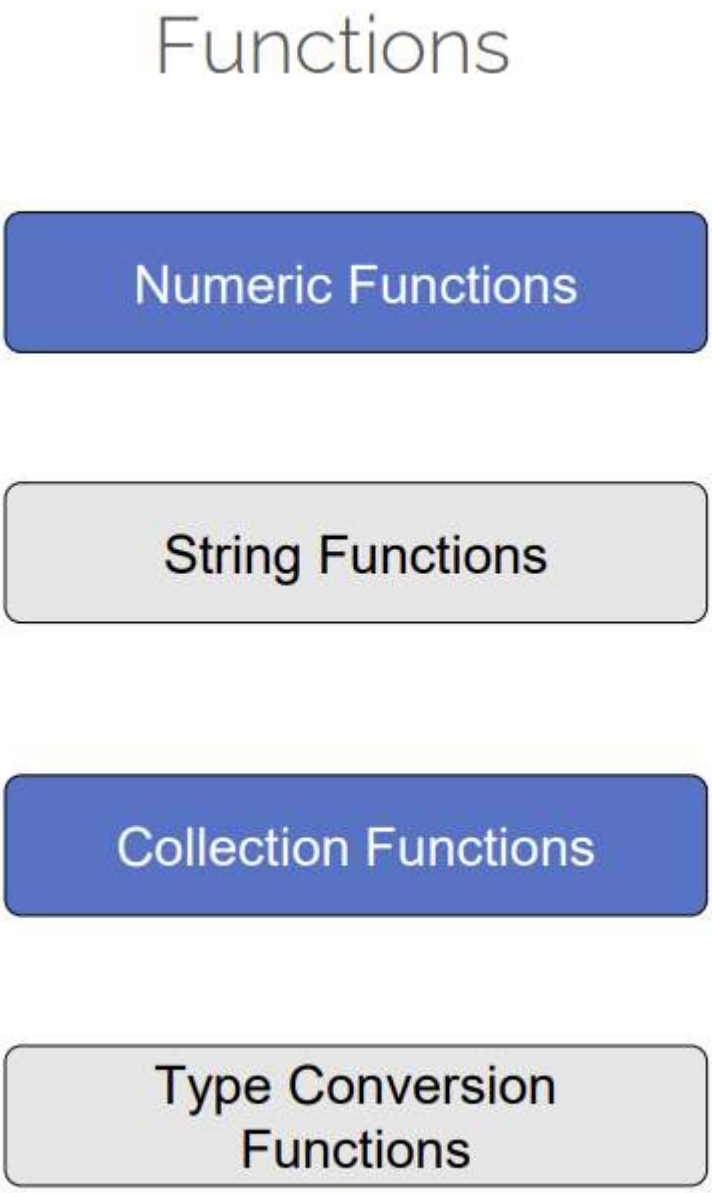
We use **function**

file

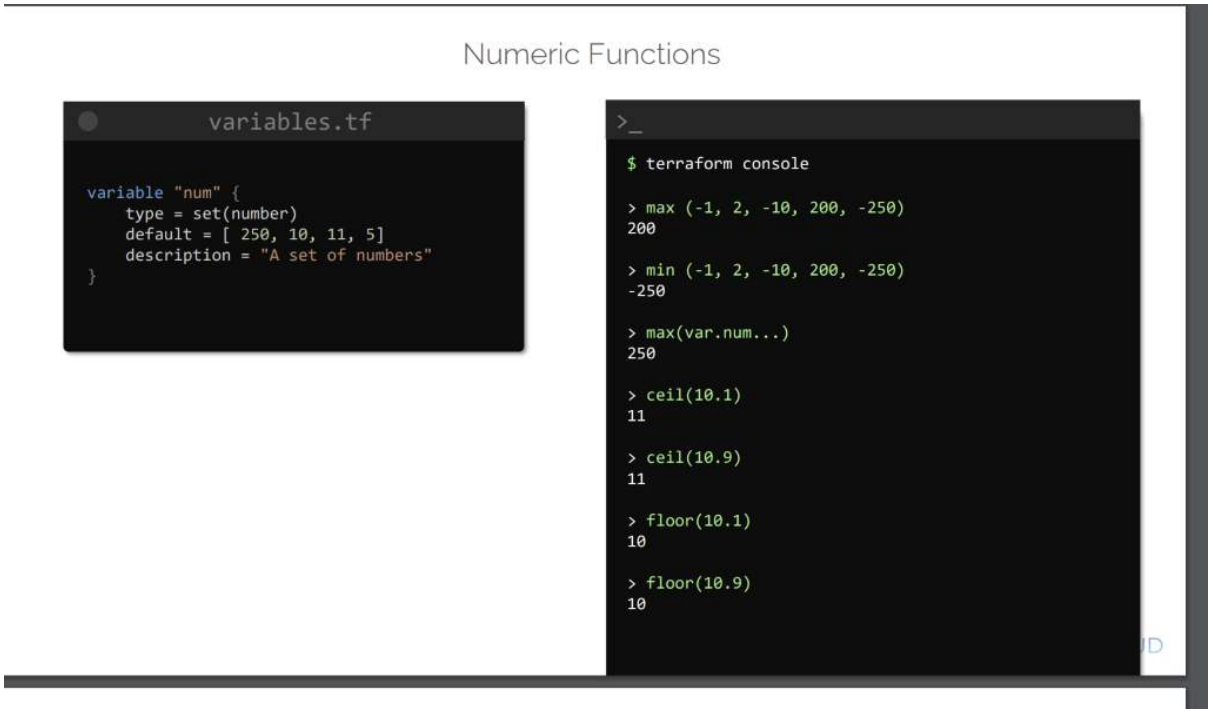
length

ToSet : convert list into set

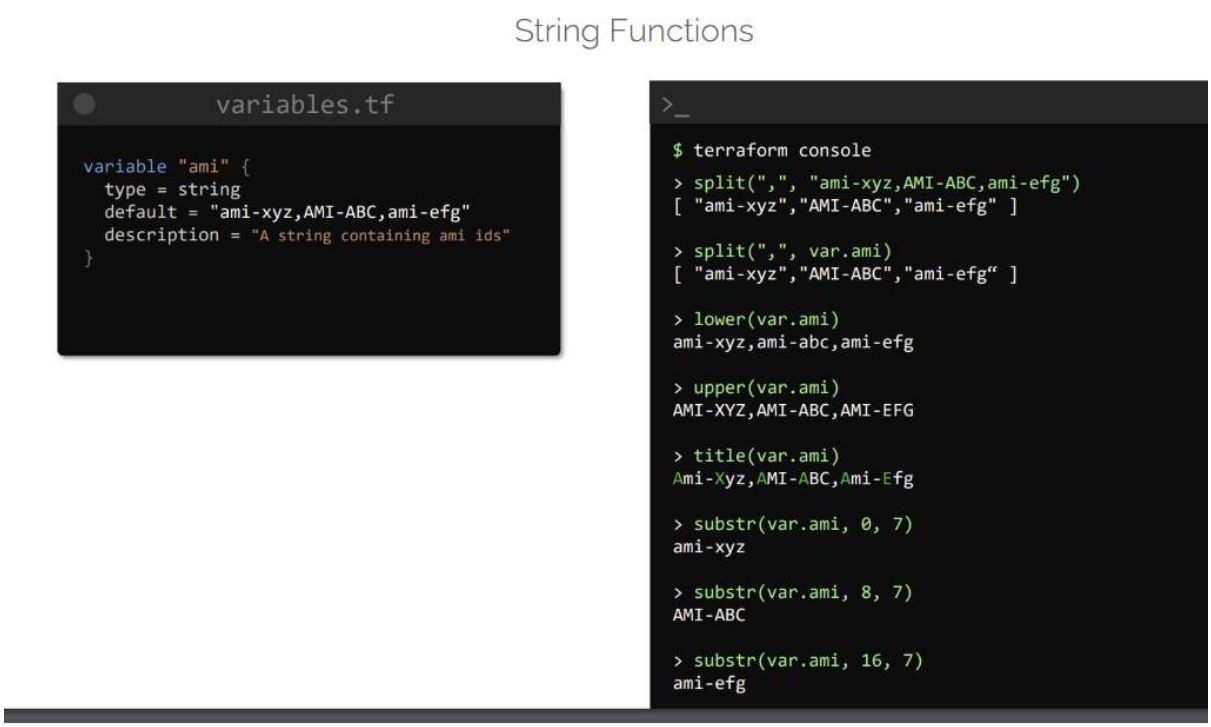
We can test function using **terraform console**



Numeric Function



String function



Join function : opp of split , it convert a list into string

String Functions

variables.tf

```
variable "ami" {
  type = list
  default = ["ami-xyz", "AMI-ABC", "ami-efg"]
    description = "A list of numbers"
}
```

>_

```
$ terraform console
> join(",", ["ami-xyz", "AMI-ABC", "ami-efg"])
ami-xyz,AMI-ABC,ami-efg

> join(",", var.ami)
ami-xyz,AMI-ABC,ami-efg
```

KODEKLOUD

Collection function

Collection Functions

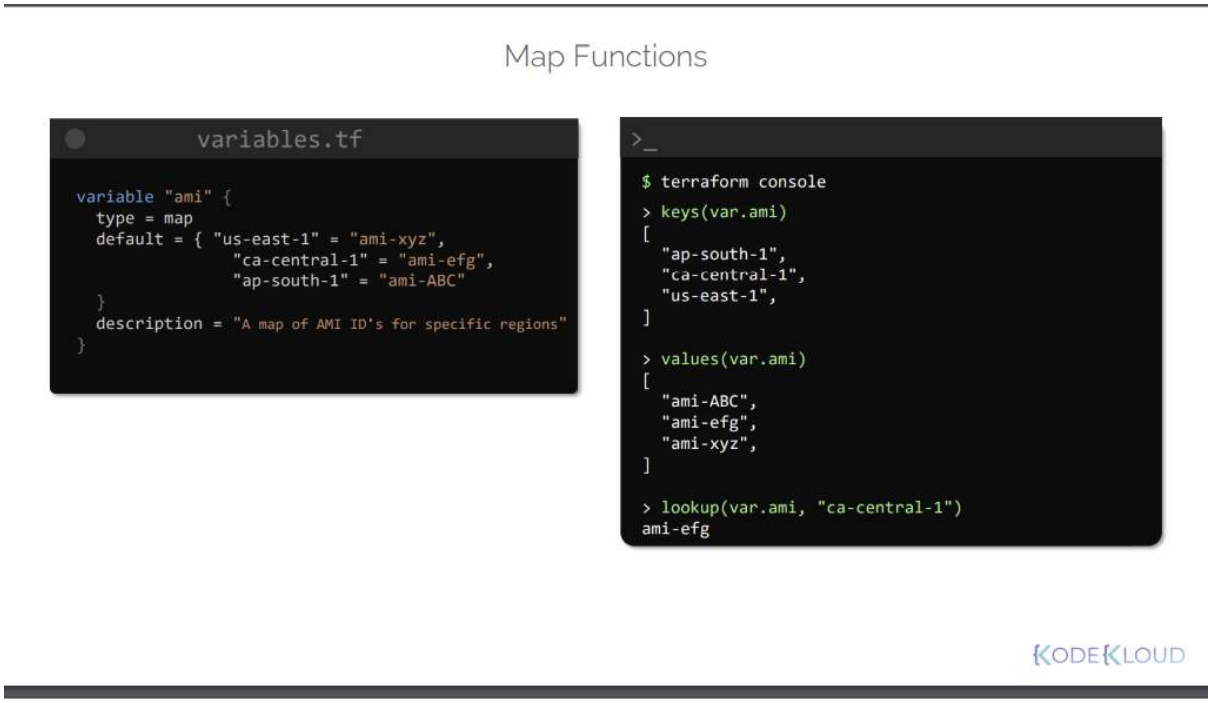
variables.tf

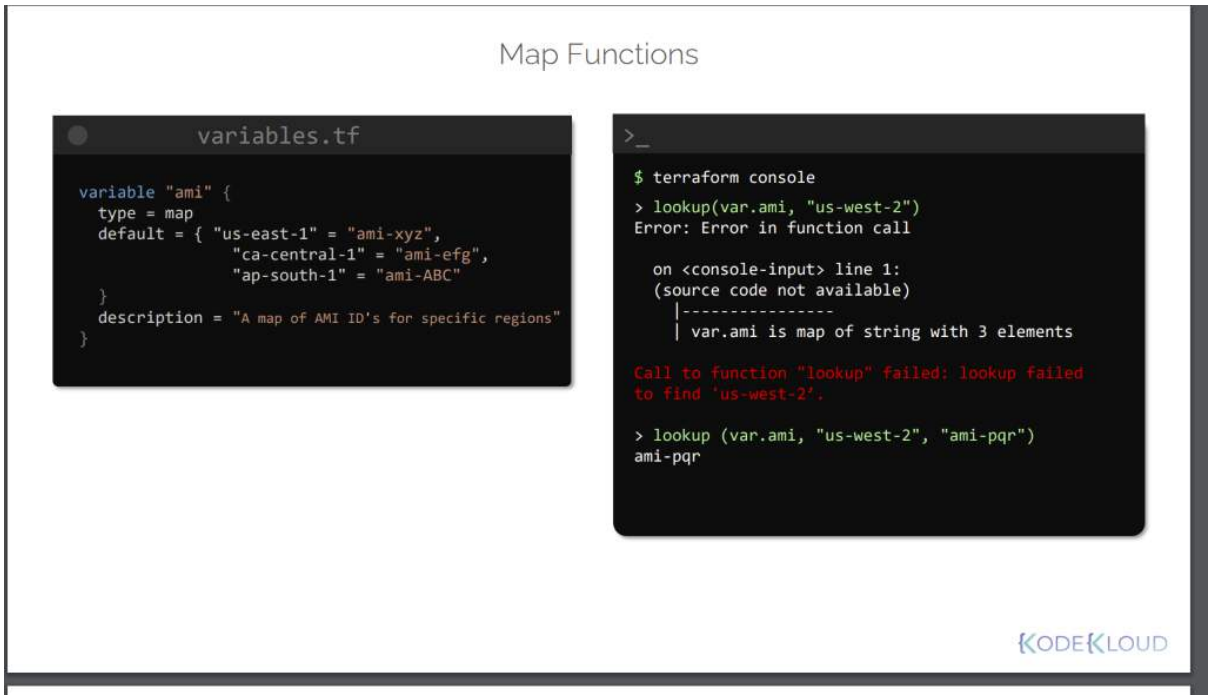
```
variable "ami" {
  type = list
  default = ["ami-xyz", "AMI-ABC", "ami-efg"]
    description = "A list of numbers"
}
```

>_

```
$ terraform console
> length(var.ami)
3

> index(var.ami, "AMI-ABC")
1

> element(var.ami,2)
ami-efg

> contains(var.ami, "AMI-ABC")
true

> contains(var.ami, "AMI-XYZ")
false
```

KODEKLOUD



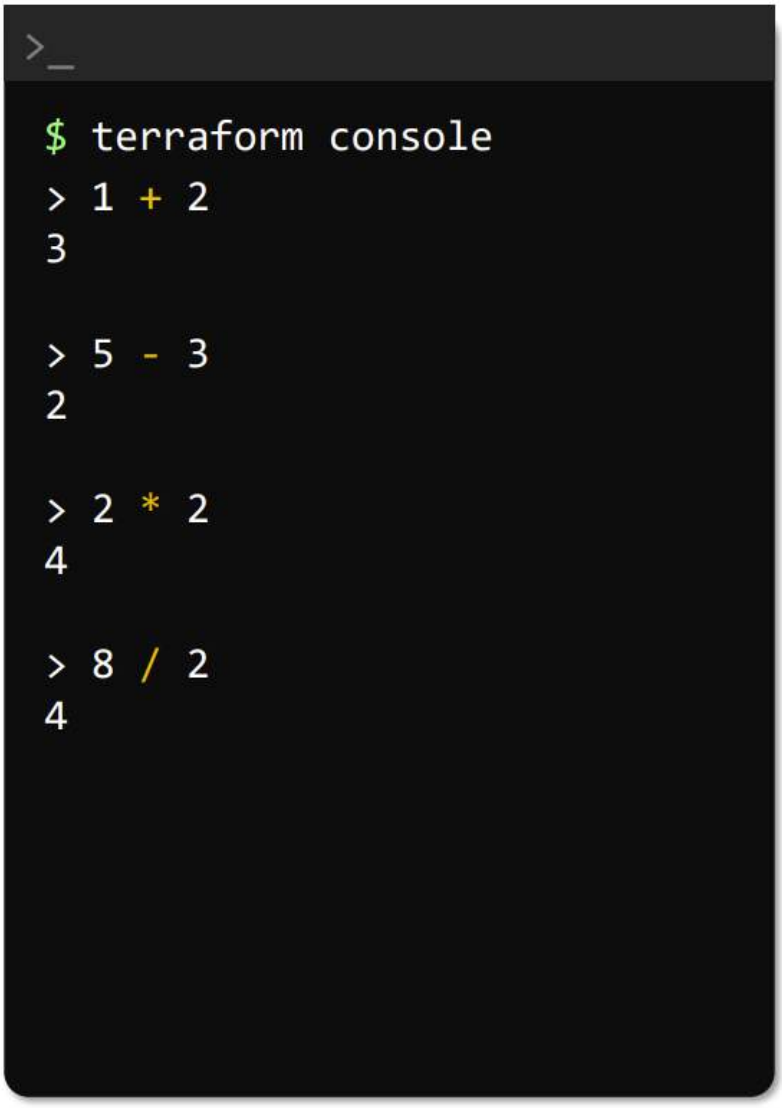


Conditional Expression

We can also test arithmetic and logical operator using terraform console

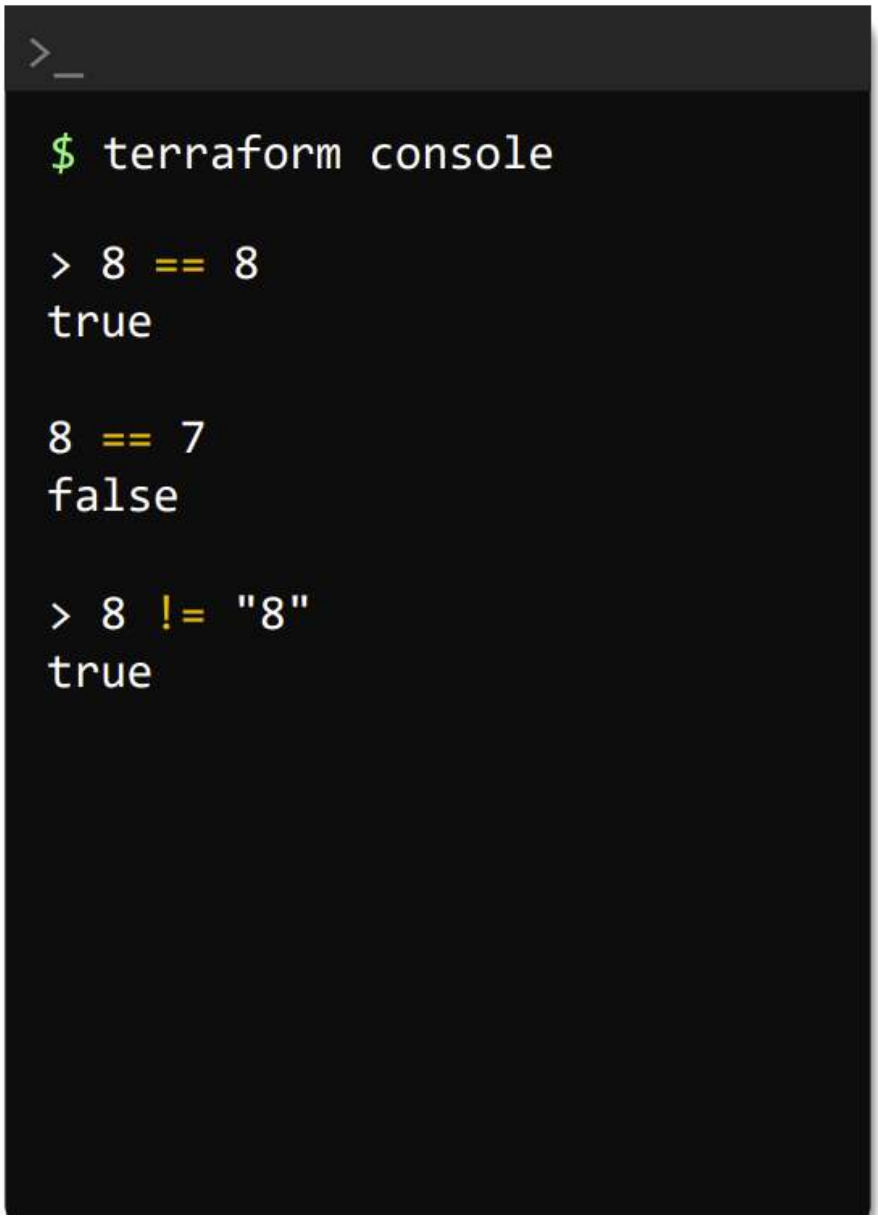
Numeric operator

Numeric Operators

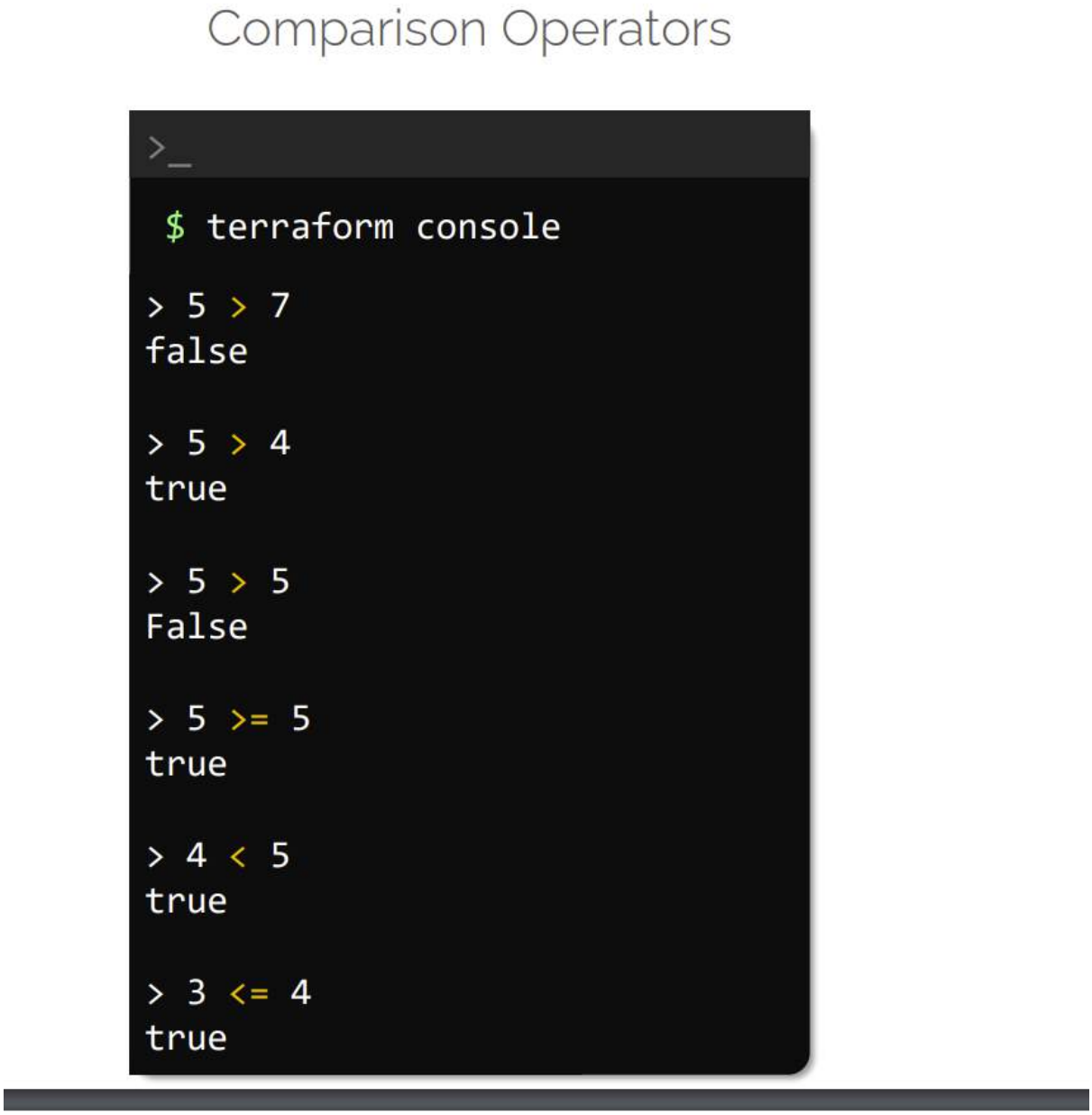


Equality operator

Equality Operators



Comparision operator



Logical operator

Logical Operators

```
>_

$ terraform console

> 8 > 7 && 8 < 10
true

> 8 > 10 && 8 < 10
false

> 8 > 9 || 8 < 10
True

> var.special
true

> ! var.special
false

> ! (var.b > 30)
true
```

variables.tf

```
variable special {
  type        = bool
  default     = true
  description = "Set to true to
      use special characters"
}

variable b {
    type = number
    default = 25
}
```

KODEKLOUD

Logical Operators

```
>_
$ terraform console

> var.a > var.b
true

> var.a < var.b
false

> var.a + var.b
75
```

variables.tf

```
variable a {
    type = number
    default = 50
}
variable b {
    type = number
    default = 25
}
```

KODEKLOUD

main.tf

```
resource "random_password" "password-generator" {
    length  = var.length
}

output password {
  value  = random_password.password-generator.result
}
```

variables.tf

```
variable length {
  type        = number
  description = "The length of the password"
}
```

```
>_
$ terraform apply -var=length=5 -auto-approve
random_password.password-generator: Creating...
random_password.password-generator: Creation
complete after 0s [id=none]

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

Outputs:

password = sjsrW]
```

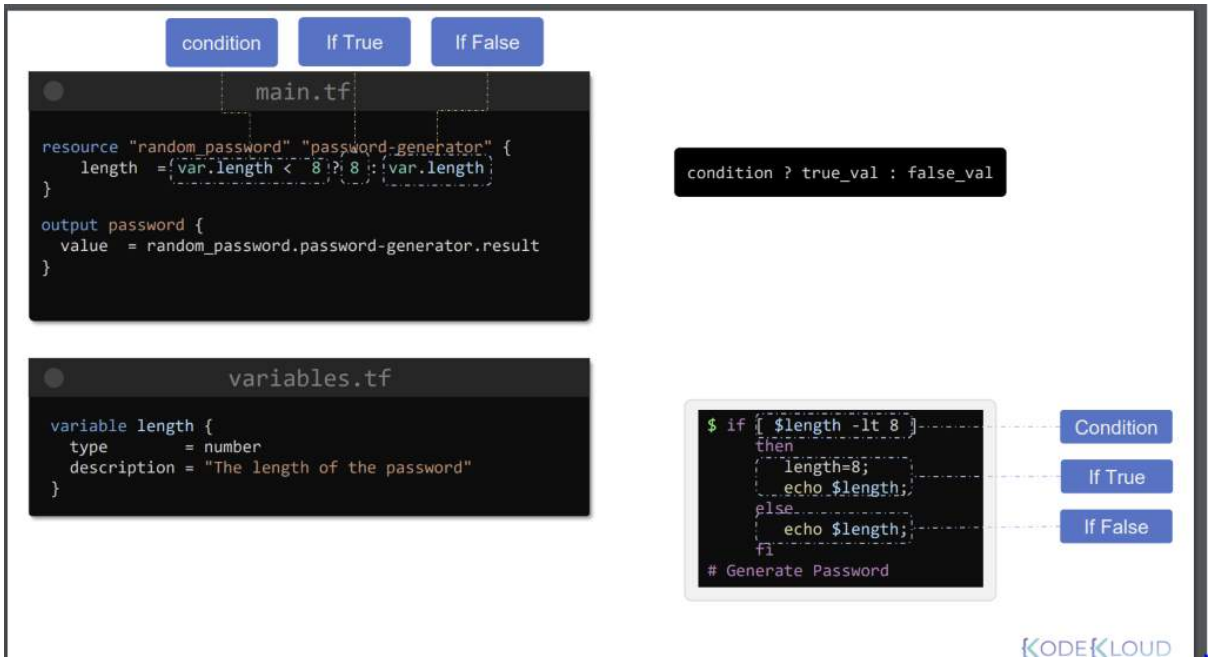
```
$ if [ $length -lt 8 ]
    then
      length=8;
      echo $length;
    else
      echo $length;
    fi
# Generate Password
```

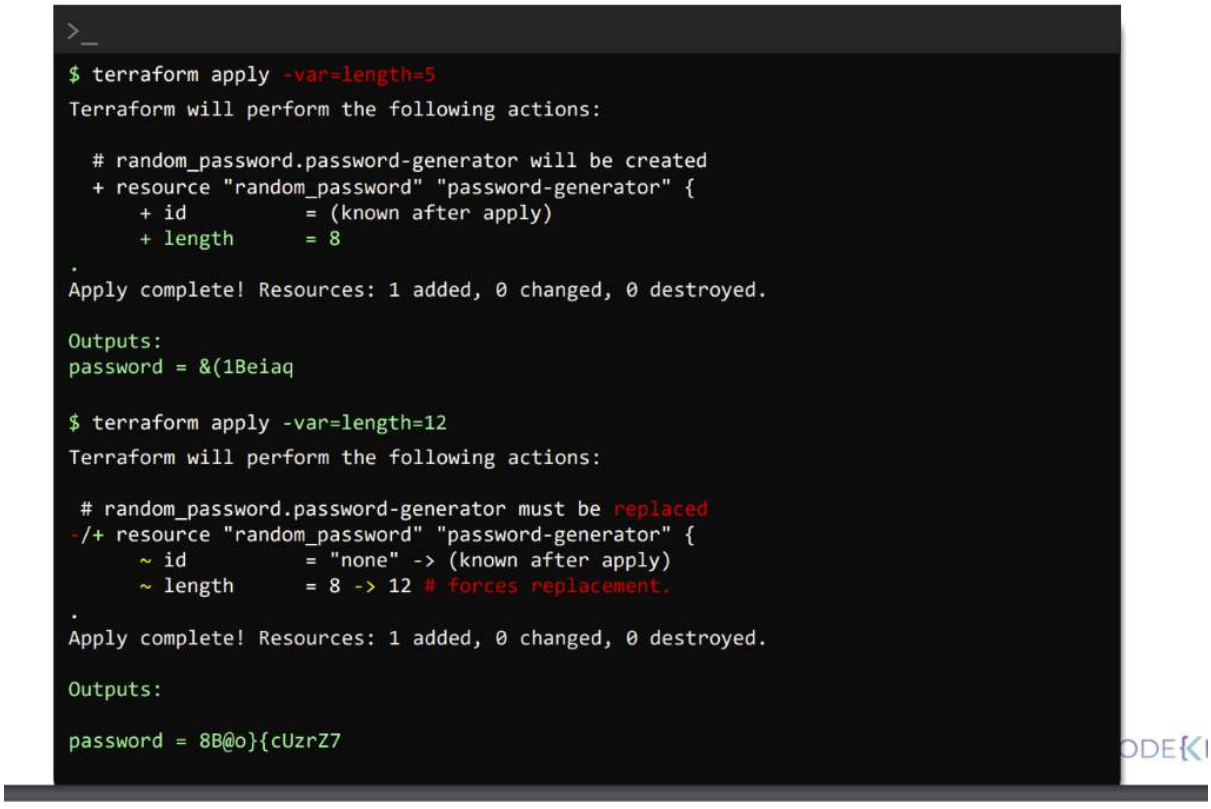
Condition

If True

If False

KODEKLOUD

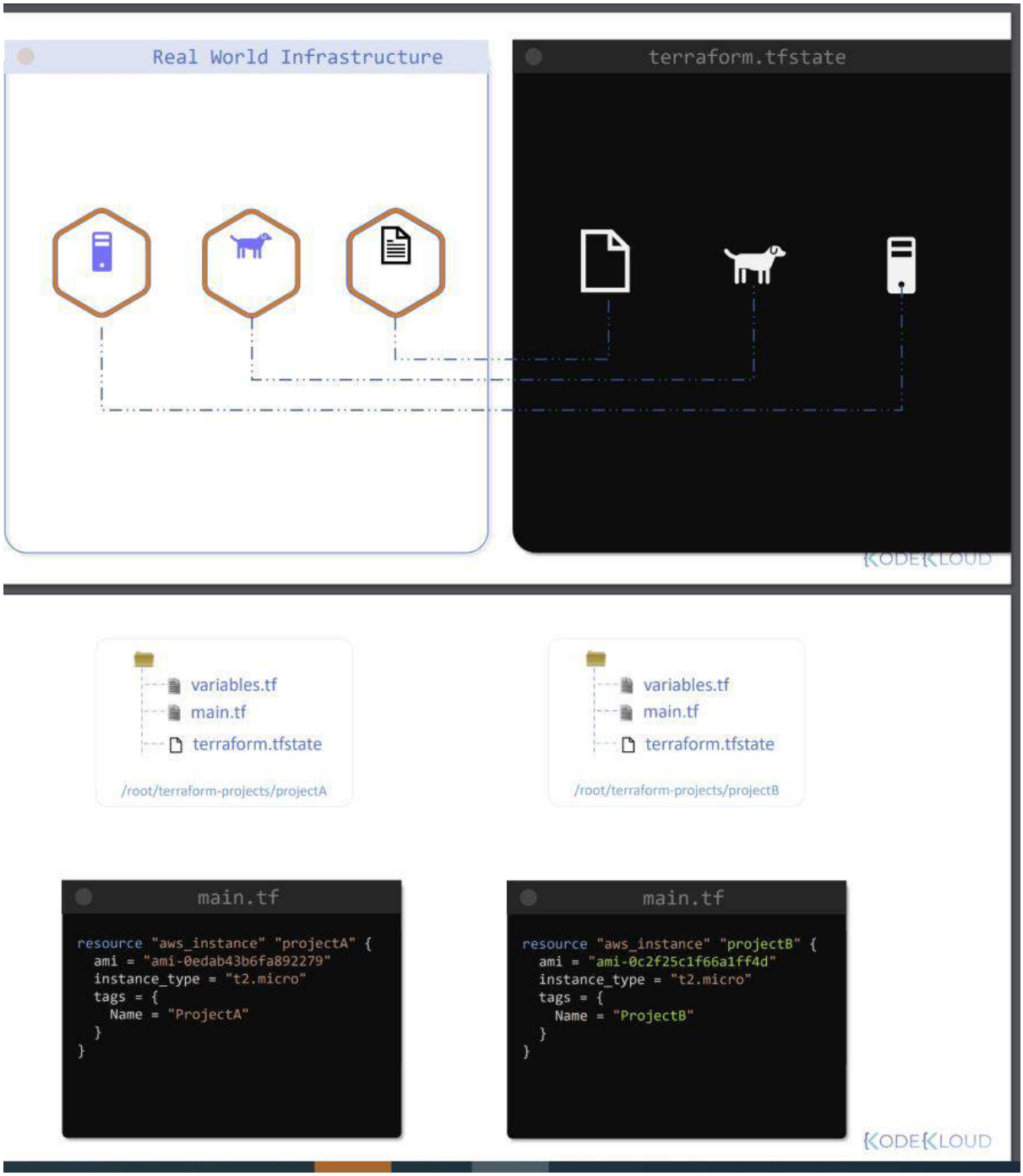




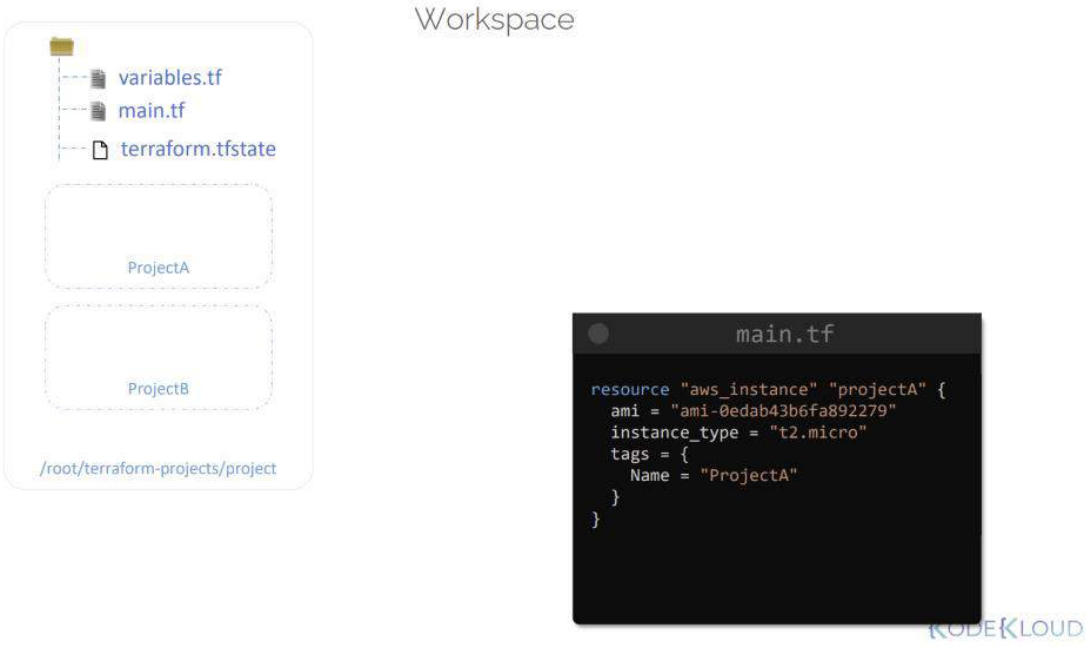
Teraform workspace oss

Will learn workspaces in terraform and how to use it

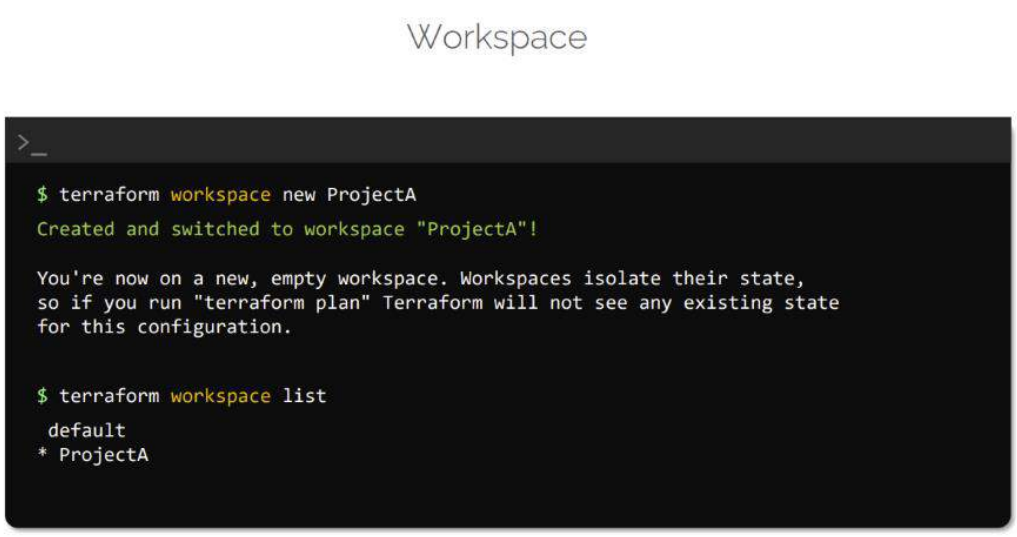
So far we have handle one state file per configuration



Within same workspace we can create multiple project



- (star means current workspace)



Lookup function is use to find a specific value from a map

To create new workspace

```
$ terraform plan
Terraform will perform the following actions:

  # aws_instance.project will be created
  + resource "aws_instance" "project" {
      + ami                          = "ami-0edab43b6fa892279"
      + instance_type                = "t2.micro"
      + tags                         = {
          + "Name" = "ProjectA"
        }
  .
  .

$ terraform workspace new ProjectB
Created and switched to workspace "ProjectB"!

You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
```

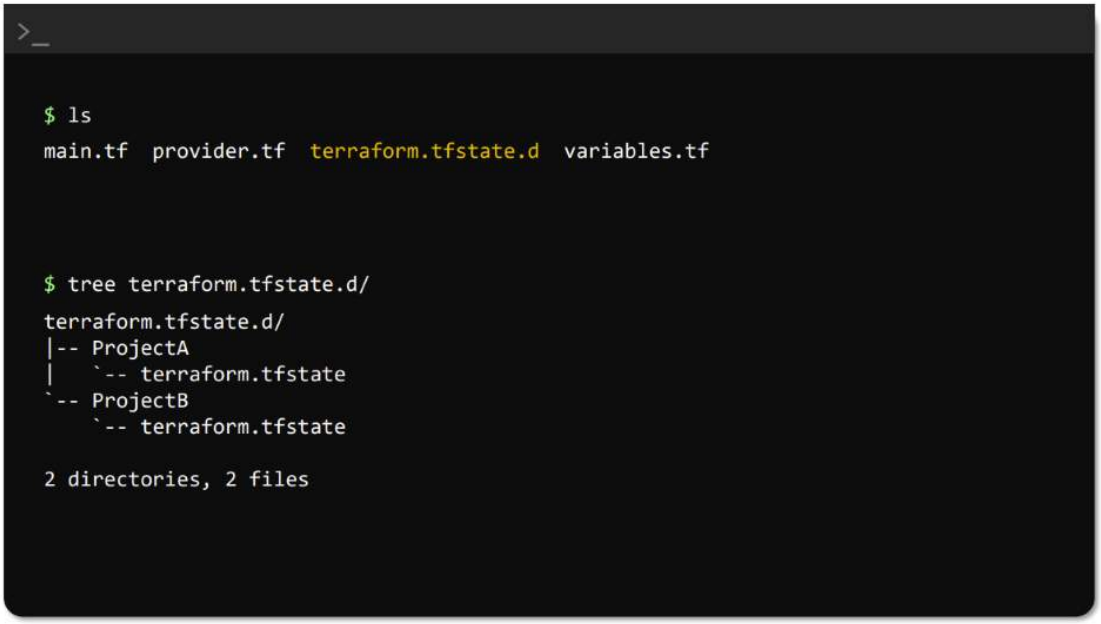
KODEKLO

To switch workspace

```
$ terraform plan
Terraform will perform the following actions:

  # aws_instance.project will be created
  + resource "aws_instance" "project" {
      + ami                          = "ami-0c2f25c1f66a1ff4d"
      + instance_type                = "t2.micro"
      + tags                         = {
          + "Name" = "ProjectB"
        }
  .
  .
  .



$ terraform workspace select ProjectA
Switched to workspace "ProjectA".
```

When we run terraform apply it create two state file for different workspace but where does it store state file

Now it will store file in **terraform.tf.d** directory


>_
$ ls
main.tf provider.tf terraform.tfstate.d variables.tf
$ tree terraform.tfstate.d/
terraform.tfstate.d/
|-- ProjectA
| `-- terraform.tfstate
`-- ProjectB
    `-- terraform.tfstate
2 directories, 2 files