

<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

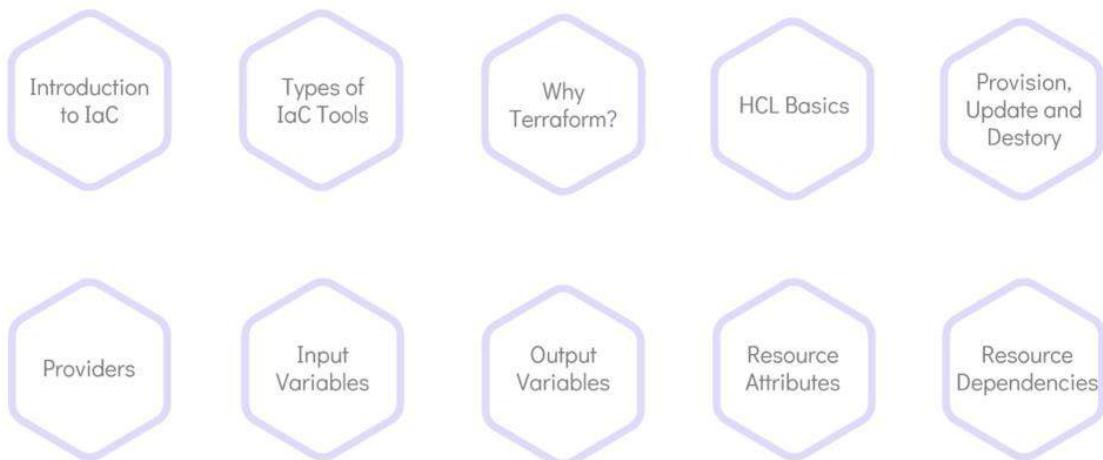
Section 10: terraform import tainting resources and debugging

Section 11: terraform modules

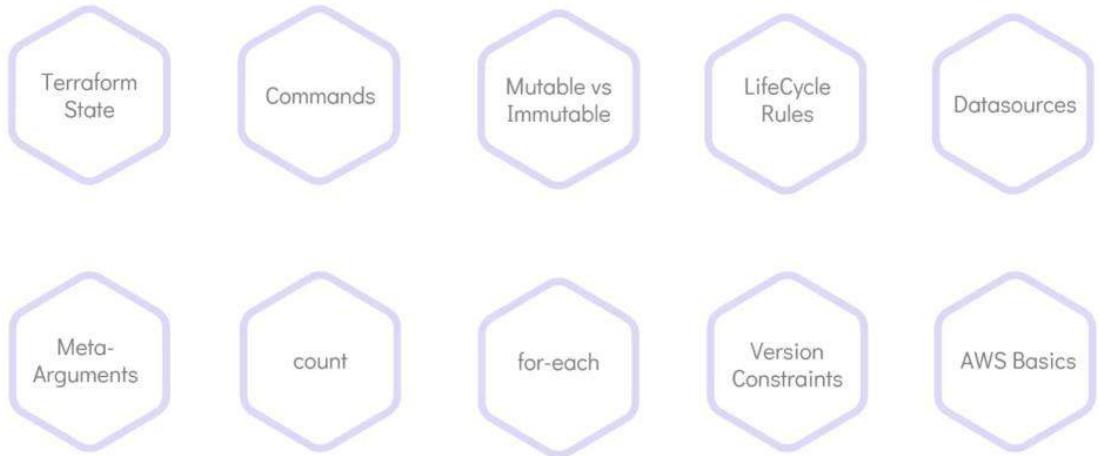
Section 12: terraform functions and conditional expression

Section 1: Introduction

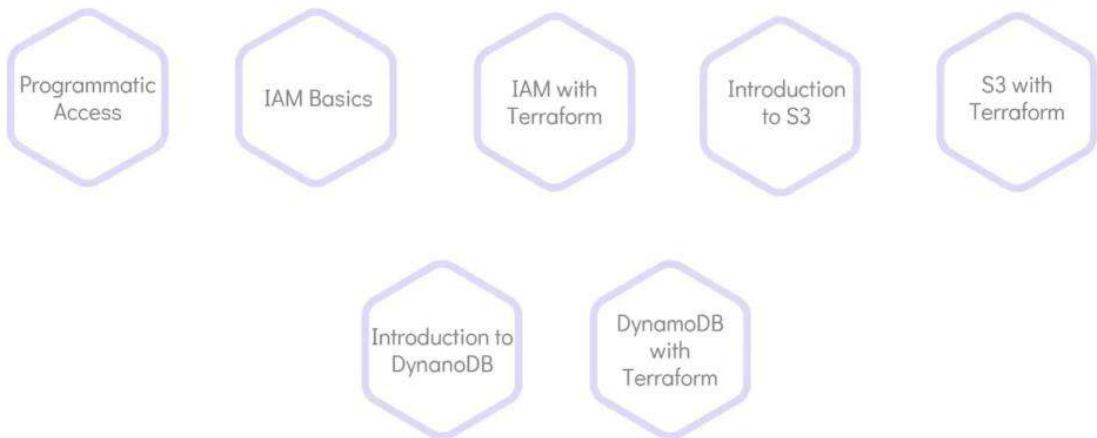
Objectives



Objectives



Objectives



Objectives



Objectives



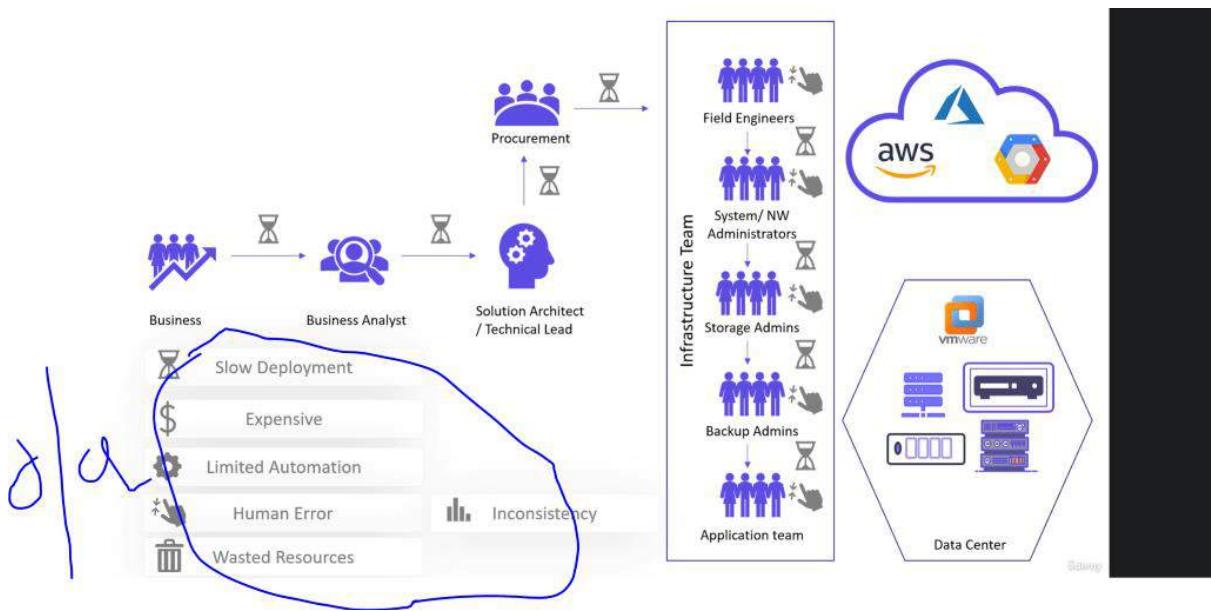
Join Our Community

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-iccmKnUM7pUXpqUdKUNIgQ

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 :

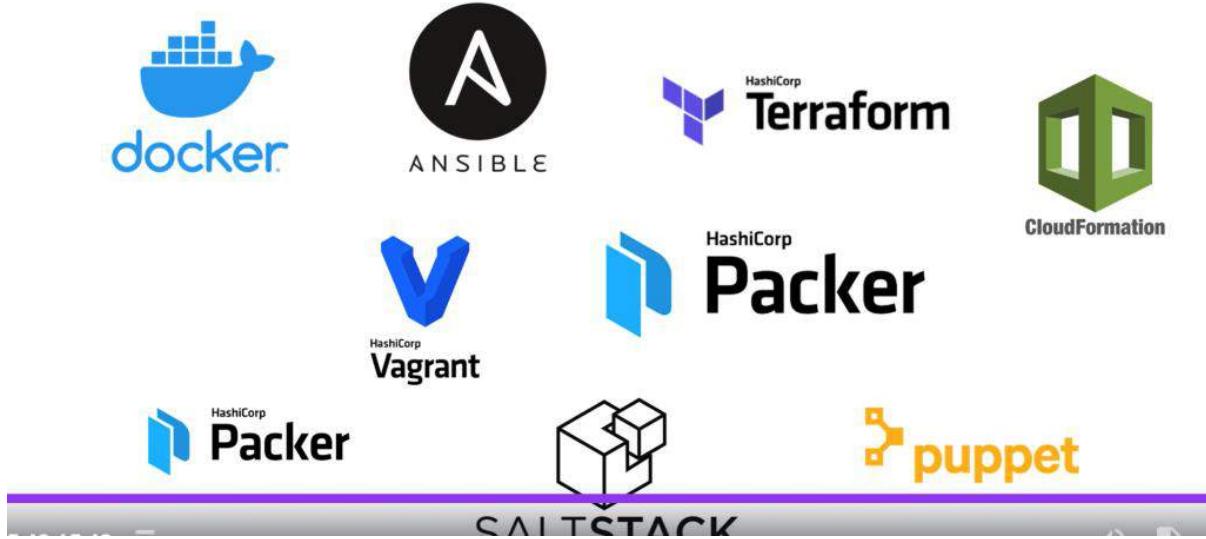
By moving to cloud the time spin up infrastructure are reduced

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

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 screenshot displays two examples of Infrastructure as Code (IaC) tools:

Left Side: A terminal window titled "ec2.sh" showing a shell script. The script provisions an AWS Lambda function named "lambdaFunction" and creates an AWS Lambda function named "lambdaFunction". It uses AWS CloudFormation to define the Lambda functions and their dependencies.

```
#!/bin/bash
# Create Lambda function
IP_ADDRESS="10.2.2.1"
EC2_INSTANCE=$(aws lambda create-function --function-name lambdaFunction --runtime nodejs12.x --role arn:aws:iam::123456789012:role/lambdaBasicExecutionRole --handler index.handler --code S3Bucket=aws-quickstart,S3Key=lambda-functions/lambdaFunction/index.js)
INSTANCE=$(echo ${EC2_INSTANCE} | sed 's/*INSTANCE//')
sed 's/.*/$INSTANCE/' > $INSTANCE
# Wait for instance to be ready
while ! aws lambda get-function-configuration --function-name $INSTANCE | grep -q "running"
do
    echo Waiting for $INSTANCE is to be ready...
done
# Check if instance is not provisioned and exit
if ! $(aws lambda get-function-configuration --function-name $INSTANCE | grep -q "running"); then
    echo Instance $INSTANCE is stopped.
    exit
fi
# Associate address
aws ec2 associate-address $IP_ADDRESS -i $INSTANCE
echo Instance $INSTANCE was created successfully!!!
```

Right Side: An AWS CloudFormation console view for a stack named "Step 7: Review Instance Launch". The stack summary shows:

- Stack Status: CREATE_IN_PROGRESS
- Stack ID: arn:aws:cloudformation:us-east-1:123456789012:stack/Step 7: Review Instance Launch/123456789012
- Created At: 2020-07-09T15:48:36.426Z
- Last Updated At: 2020-07-09T15:48:36.426Z

The "Instance Type" section shows:

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)
t2.micro	Variable	1	1	EBS only

The "Security Groups" section shows:

Type	Protocol	Port Range
launch-wizard-1	tcp	22

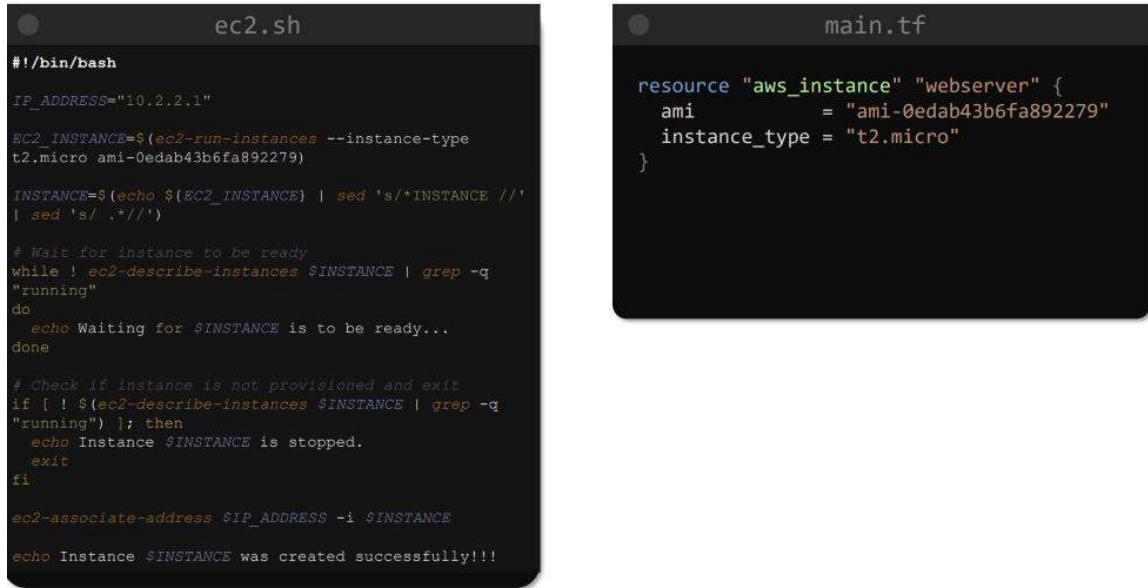
The "Instance Details" section shows:

Number of instances	1
Network	vpc-fe3baa86

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

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

Infrastructure as Code



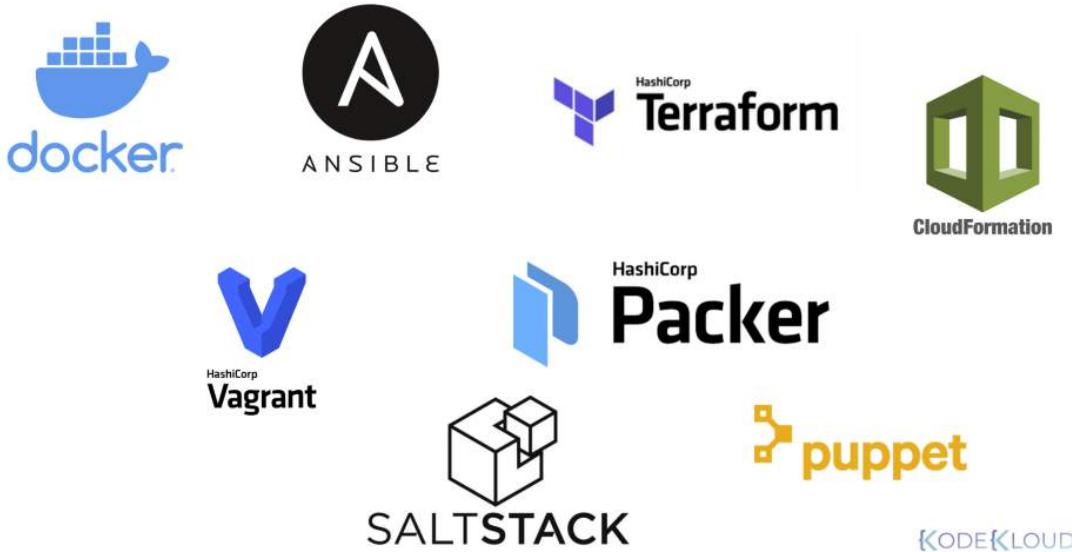
```
ec2.sh
#!/bin/bash
IP_ADDRESS="10.2.2.1"
EC2_INSTANCE=$(ec2-run-instances --instance-type t2.micro ami-0edab43b6fa892279)
INSTANCE=$(echo ${EC2_INSTANCE} | sed 's/*INSTANCE //'
| sed 's/ .*//')
# Wait for instance to be ready
while ! ec2-describe-instances $INSTANCE | grep -q "running"
do
    echo Waiting for $INSTANCE is to be ready...
done
# Check if instance is not provisioned and exit
if [ ! $(ec2-describe-instances $INSTANCE | grep -q "running") ]; then
    echo Instance $INSTANCE is stopped.
    exit
fi
ec2-associate-address $IP_ADDRESS -i $INSTANCE
echo Instance $INSTANCE was created successfully!!!
```

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

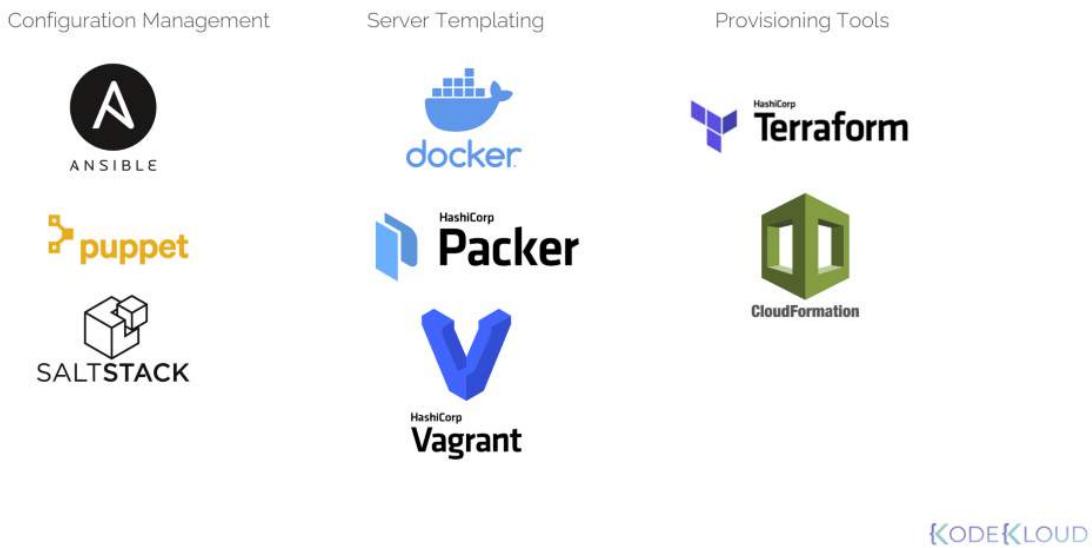
Note :

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

Types of IAC Tools



Types of IAC Tools



Configuration management:

These are usually used to install and manage software on existing infrastructure resources such as servers, databases, etc.

Idempotent: every time 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**.

Types of IAC Tools

Configuration Management



Designed to Install and Manage Software

Maintains Standard Structure

Version Control

Idempotent

Server templating tools

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

Server Templating Tools

Pre Installed Software and Dependencies

Virtual Machine or Docker Images

Immutable Infrastructure



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:

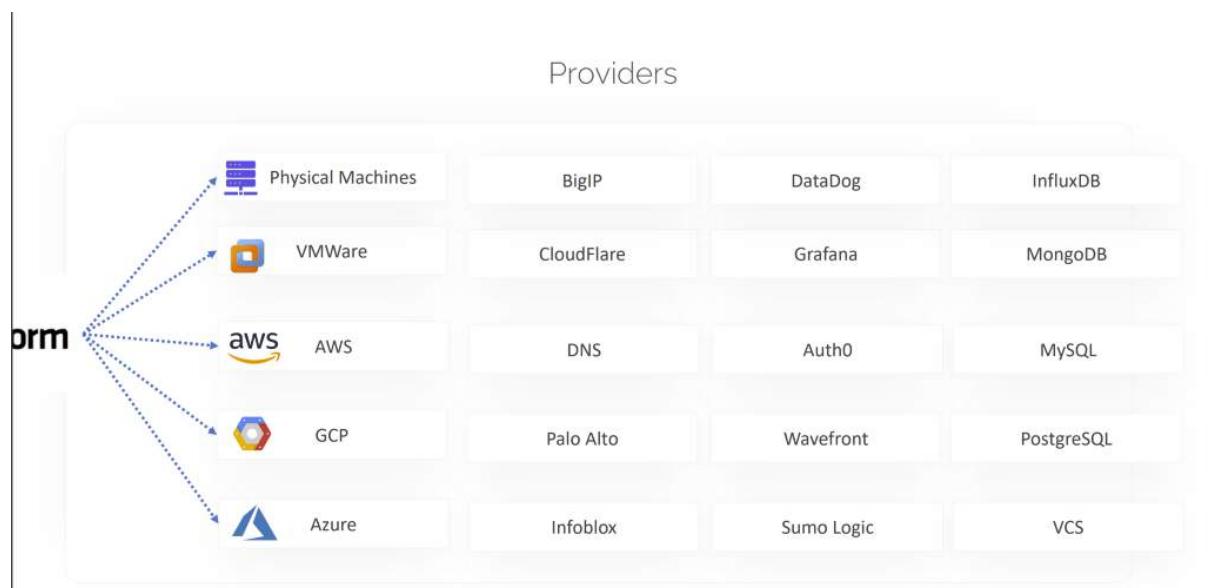
It can deploy infrastructure across multiple platform

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 controlled by 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"
    }
}

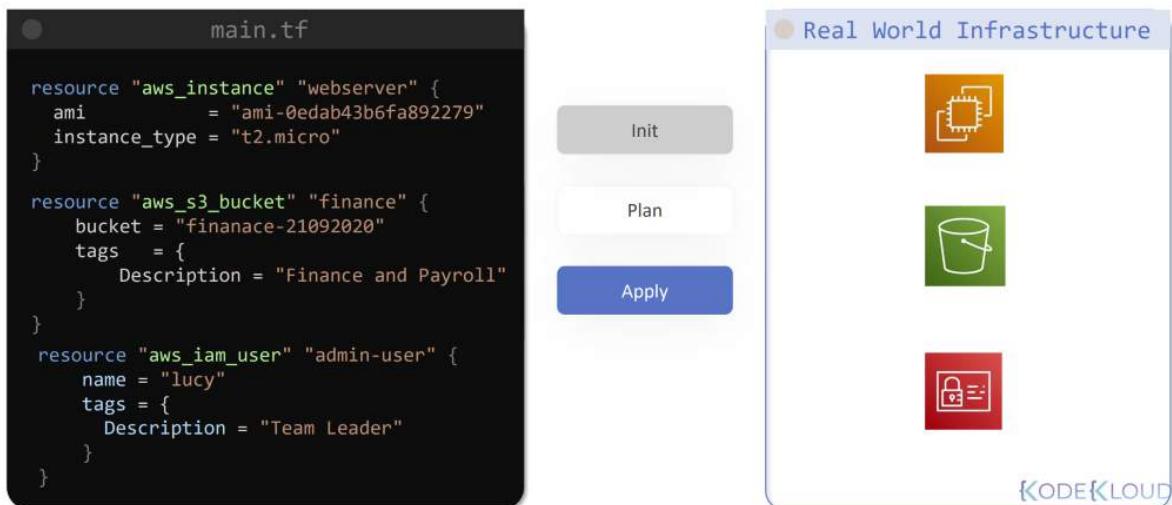
resource "aws_iam_user" "admin-user" {
    name = "lucy"
    tags = {
        Description = "Team Leader"
    }
}
```

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

Declarative



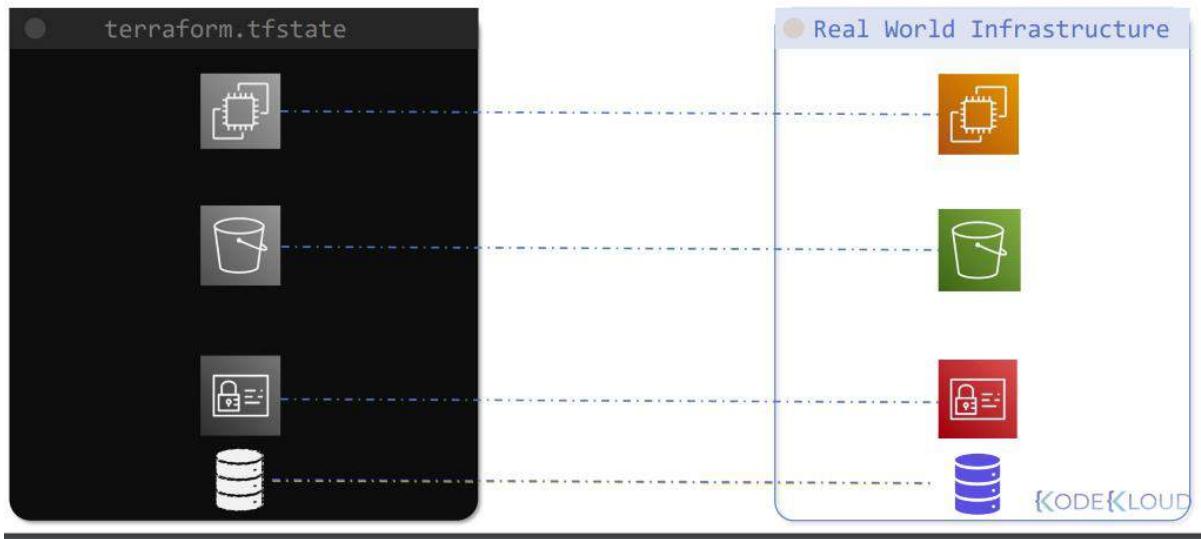
Note :

Every object that terraform manages is called resource

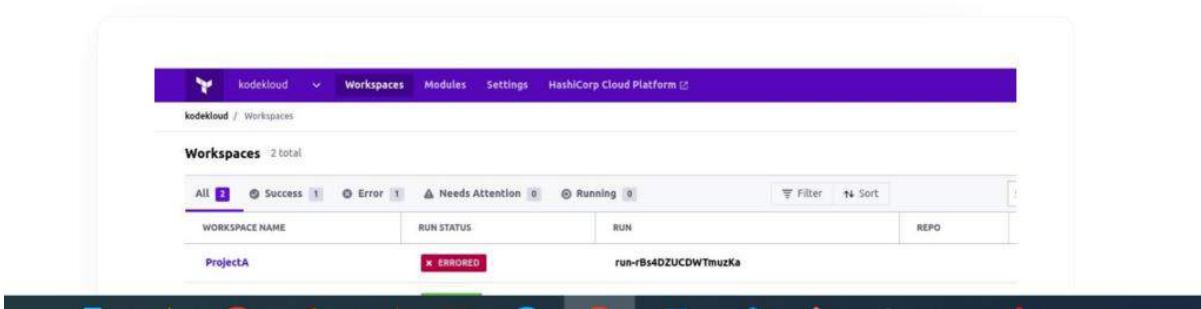
Terraform can read existing attribute of infrastructure by configuring datasource

Terraform can also import other resources outside terraform either manually or by means of other iac tool

Terraform Import



Terraform Cloud and Terraform Enterprise



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



HCL – Declarative Language

```
aws.tf
```

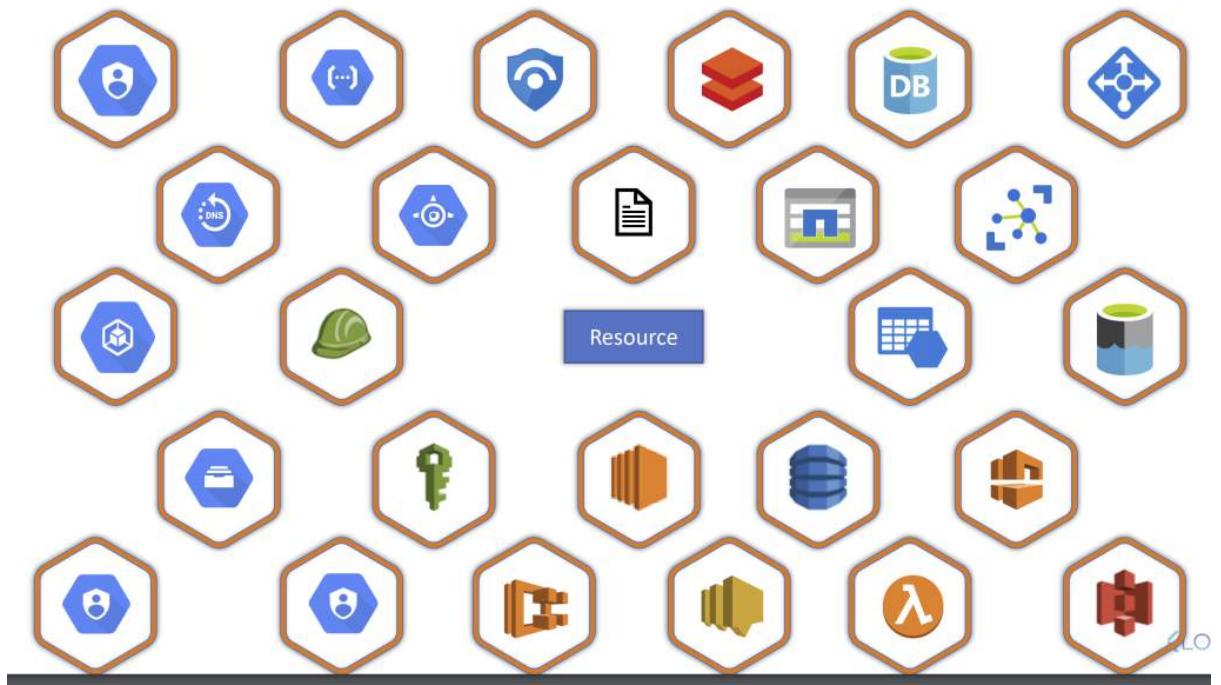
```
resource "aws_instance" "webserver" {
    ami = "ami-0c2f25c1f66a1ff4d"
    instance_type = "t2.micro"
}
```

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 initialize 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 command 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>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>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>terraform apply
```

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

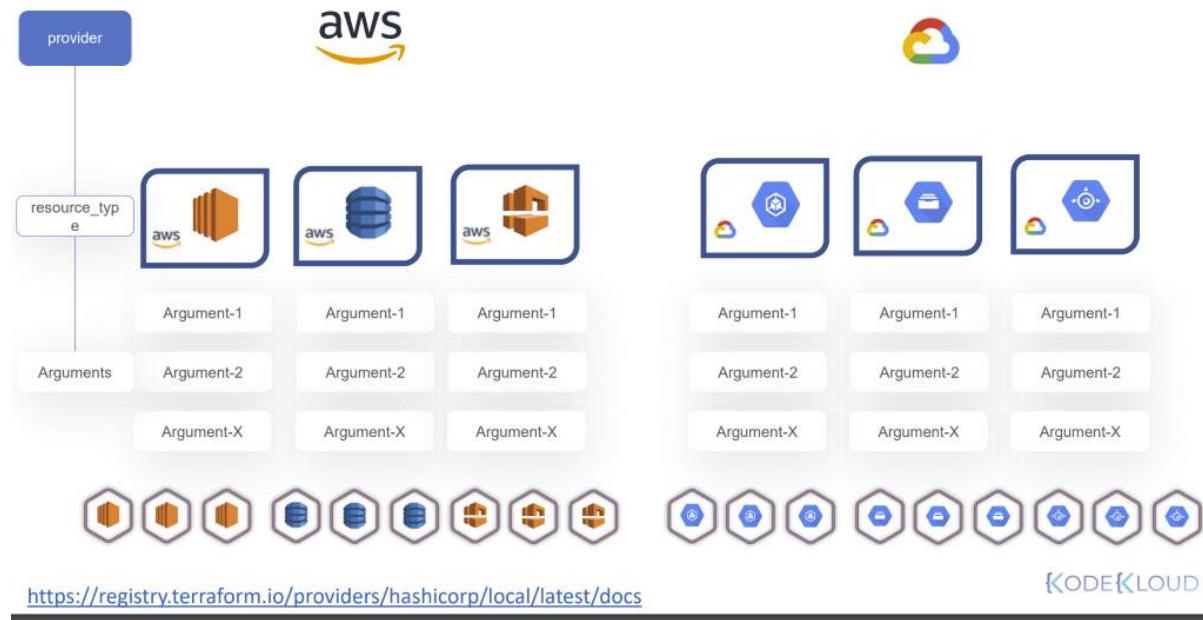
Enter a value: yes

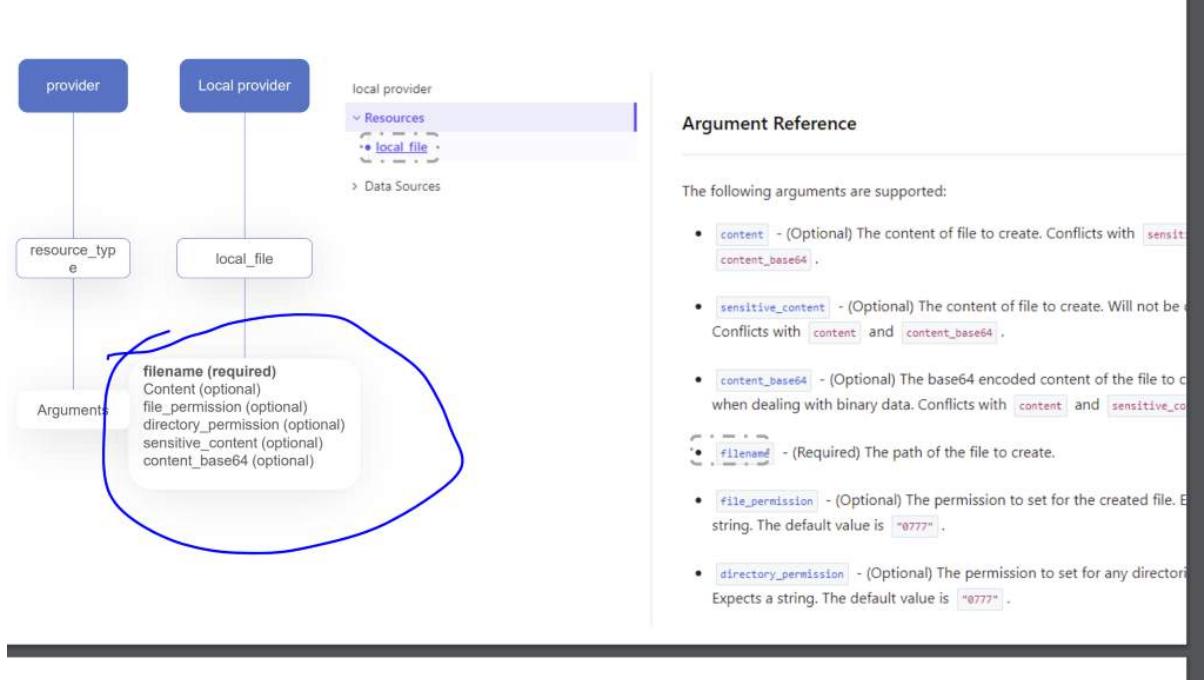
local_file.test: Creating...
local_file.test: Creation complete after 0s [id=d87a5018d4e1c1e1644d11eacf53a95c8968ef]

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

C:\Users\saurakes\OneDrive - Cisco\Desktop\terraform>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               = "d87a5018d4e1c1e1644d11eacf53a95c8968ef"
}

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





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\terraform>terraform plan
local_file.test: Refreshing state... [id=d87a5018d4e1c1e1644d1leafcdf53a95c8968ef]

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                  = "d87a5018d4e1c1e1644d1leafcdf53a95c8968ef" -> (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\terraform>
```

```

C:\Users\saurakes\OneDrive - Cisco\Desktop>terraform apply
local_file.test: Refreshing state... [id=d87a5018d4e1c1e1644d1leafcdf53a95c8968ef]

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                   = "d87a5018d4e1c1e1644d1leafcdf53a95c8968ef" -> (known after apply)
    # (3 unchanged attributes hidden)
}

Plan: 1 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

local_file.test: Destroying... [id=d87a5018d4e1c1e1644d1leafcdf53a95c8968ef]
local_file.test: Destruction complete after 0s
local_file.test: Creating...
local_file.test: Creation complete after 0s [id=d87a5018d4e1c1e1644d1leafcdf53a95c8968ef]

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

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

```

```

C:\Users\saurakes\OneDrive - Cisco\Desktop>terraform destroy
local_file.test: Refreshing state... [id=d87a5018d4e1c1e1644d1leafcdf53a95c8968ef]

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

Terraform will perform the following actions:

  # local_file.test will be destroyed
- resource "local_file" "test" {
    - content          = "we love pets" -> null
    - directory_permission = "0777" -> null
    - file_permission   = "0700" -> null
    - filename          = "C://Users//saurakes//test//pet.txt" -> null
    - id                = "d87a5018d4e1c1e1644d1leafcdf53a95c8968ef" -> null
}

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

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

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

```

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

```
>_
```

```
$ terraform init
```



KODEKLOUD

Official



Verified



bigip

by: FSNetworks



heroku



digitalocean

by: digitalocean

Community



activedirectory



ucloud



netapp-gcp

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

To prevent automatic upgrades to new major versions from containing breaking changes, we recommend adding version constraints to the required_providers block in your configuration, with the constraint below.

```
* registry.terraform.io/[hashicorp]/local
```

Hostname

Organizational Namespace

Terraform has been successfully initialized

Configuration Directory

```
>_
[terraform-local-file]$ ls /root/terraform-local-file
local.tf
```

```
local.tf
```

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

```
cat.tf
```

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

KODEKLOUD

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

```
main.tf
```

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

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

random provider
▼

Resources

- random_id
- random_integer
- random_password
- random_pet
- random_shuffle
- random_string
- random_uuid

Argument Reference

The following arguments are supported:

- `keepers` - (Optional) Arbitrary map of values that will be generated. See [the main provider documentation](#).
- `length` - (Optional) The length (in words) of the pet name.
- `prefix` - (Optional) A string to prefix the name with.
- `separator` - (Optional) The character to separate words in the pet name.

```
>_
```

```
$ 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

```

>_ $ terraform apply
local_file.new_file: Refreshing state...
[id=d1a31467f206d6ea8ab1cad382bc106bf46df69e]

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

Terraform will perform the following actions:

# 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.

random_pet.my-pet: Creating...
random_pet.my-pet: Creation complete after 0s [id=Mrs.hen]
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

```



Mrs.hen

KODEKLOUD

Using Input Variables

```

main.tf

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

}

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

```

Argument	Value
filename	"/root/pets.txt"
content	"We love pets!"
prefix	"Mrs"
separator	".."
length	"1"

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

The image shows four terminal windows illustrating lists as types in Terraform.

- variables.tf (Top Left):**

```
variable "prefix" {  
    default = ["Mr", "Mrs", "Sir"]  
    type = list(string)  
}
```
- variables.tf (Top Right):**

```
variable "prefix" {  
    default = ["Mr", "Mrs", "Sir"]  
    type = list(number)  
}
```
- variables.tf (Bottom Left):**

```
variable "prefix" {  
    default = [1, 2, 3]  
    type = list(number)  
}
```
- Terminal Output (Bottom Right):**

```
>_  
$ 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

The image shows two terminal windows illustrating maps as types in Terraform.

- variables.tf (Left):**

```
variable "cats" {  
    default = {  
        "color" = "brown"  
        "name" = "bella"  
    }  
    type = map(string)  
}
```
- variables.tf (Right):**

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

Set

```
variables.tf
variable "prefix" {
  default = ["Mr", "Mrs", "Sir"]
  type = set(string)
}
```

```
variables.tf
variable "prefix" {
  default = ["Mr", "Mrs", "Sir", "Sir"]
  type = set(string)
}
```

```
variables.tf
variable "fruit" {
  default = ["apple", "banana"]
  type = set(string)
}
```

```
variables.tf
variable "fruit" {
  default = ["apple", "banana", "banana"]
  type = set(string)
}
```

```
variables.tf
variable "age" {
  default = ["10", "12", "15"]
  type = set(number)
}
```

```
variables.tf
variable "age" {
  default = ["10", "12", "15", "10"]
  type = set(number)
}
```

Objects

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

Objects

Key	Example	Type
name	bella	string
color	brown	string
age	7	number
food	["fish", "chicken", "turkey"]	list
favorite_pet	true	bool

```
variables.tf
variable "bella" {
  type = object({
    name = string
    color = string
    age = number
    food = list(string)
    favorite_pet = bool
  })

  default = {
    name = "bella"
    color = "brown"
    age = 7
    food = ["fish", "chicken", "turkey"]
    favorite_pet = true
  }
}
```

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"]
}

}

```

```

1   resource "local_file" "jedi" {
2     filename = var.jedi["filename"]
3     content = var.jedi["content"]
4   }
5

```

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
}
```

KODEKLOUD

```
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" {
}
```

KODEKLOUD

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
```

terraform.tfvars | terraform.tfvars.json

*.auto.tfvars | *.auto.tfvars.json

Automatically Loaded

KODEKLOUD

Variable Definition Precedence

The terminal shows the following sequence of events:

- File `main.tf`:

```
resource local_file pet {  
  filename = var.filename  
}
```
- File `variables.tf`:

```
variable filename {  
  type  = string  
}
```
- Environment variable:

```
$ export TF_VAR_filename="/root/cats.txt"
```
- File `terraform.tfvars`:

```
filename = "/root/pets.txt"
```
- File `variable.auto.tfvars`:

```
filename = "/root/mypet.txt"
```
- Command:

```
$ terraform apply -var "filename=/root/best-pet.txt"
```

KODEKLOUD

Variable Definition Precedence

A table defines the precedence order:

Order	Option
1	Environment Variables
2	<code>terraform.tfvars</code>
3	<code>*.auto.tfvars</code> (alphabetical order)
4	<code>-var</code> or <code>-var-file</code> (command-line flags)

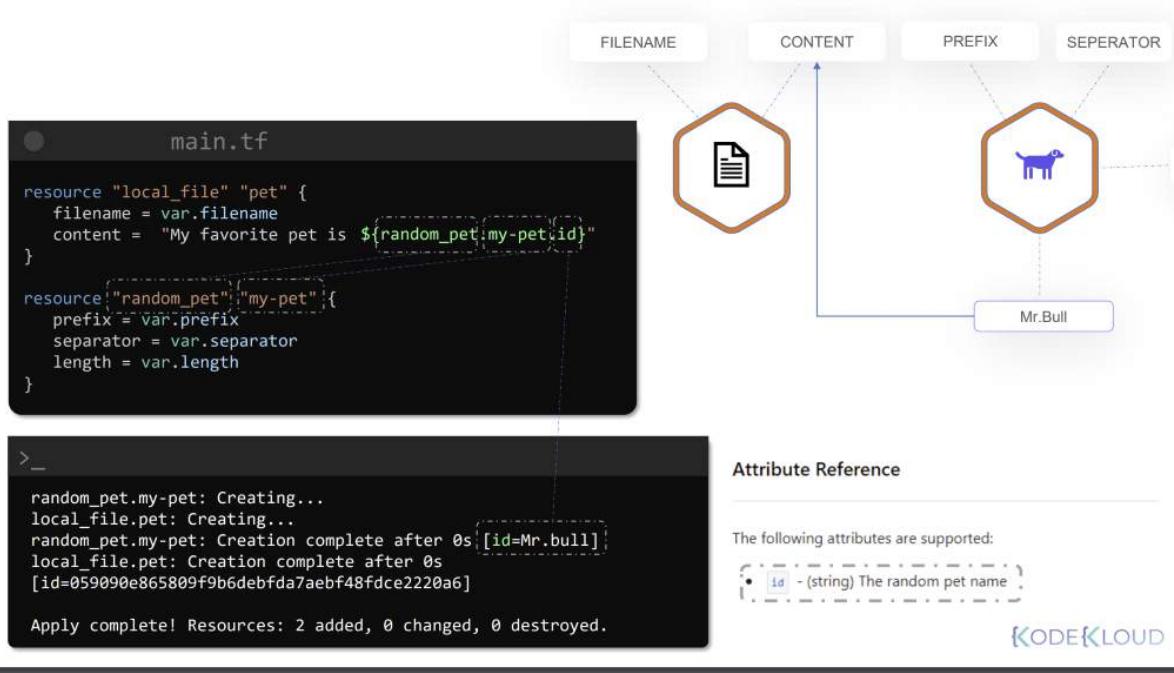
The terminal shows the same sequence of events as the first screenshot, with the command-line flag option highlighted.

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

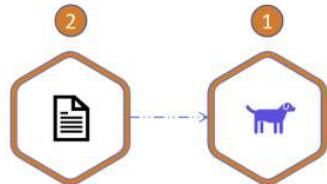
In previous lecture we saw how to link one resource to other resource using reference

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
}
```



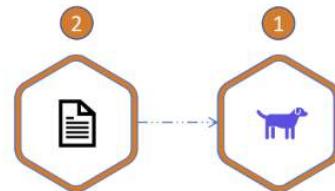
Here we are not telling which resource is depend on which . This type of dependency is called implicit

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



```
>_
$ terraform apply
.
.
Outputs:
pet-name = Mrs.gibbon
```

KODEKLOUD

```
>_
$ terraform output
pet-name = Mrs.gibbon
```

```
>_
$ terraform output pet-name
Mrs.gibbon
```



Output Variable



ANSIBLE



SHELL SCRIPTS

KODEKLOUD

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

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

variables.tf

```
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

```
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 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 symbols:
+ 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 destroyed.
```

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



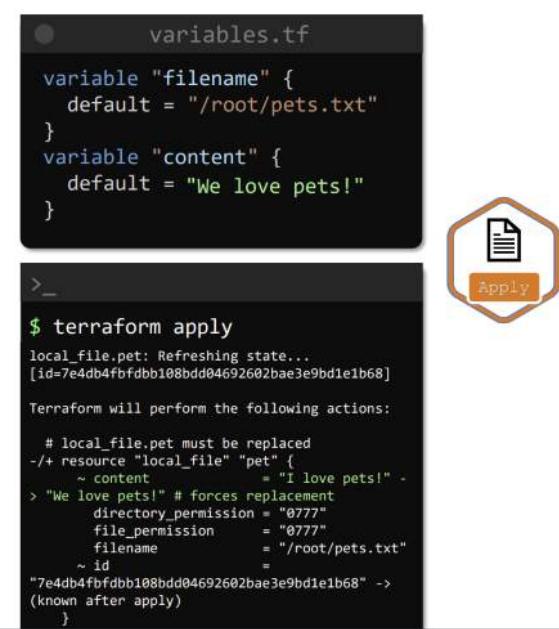
```
[terraform-local-file]$ cat terraform.tfstate  
{  
  "version": 4,  
  "terraform_version": "0.13.0",  
  "serial": 1,  
  "lineage": "e35dde72-a943-de50-3c8b-1df8986e5a31",  
  "outputs": {},  
  "resources": [  
    {  
      "mode": "managed",  
      "type": "local_file",  
      "name": "pet",  
      "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": "7e4db4fbfd8b108bdd04692602bae3e9bd1e1b68",  
            "sensitive_content": null  
          },  
          "private": "bnVsBAA=="  
        }  
      ]  
    }  
  ]  
}
```

```
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=7e4db4fbfd8b108bdd04692602bae3e9bd1e1b68]  
.  
.  
[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": "7e4db4fbfd8b108bdd04692602bae3e9bd1e1b68",  
            "sensitive_content": null  
          },  
          "private": "bnVsBAA=="  
        }  
      ]  
    }  
  ]  
}
```



```

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

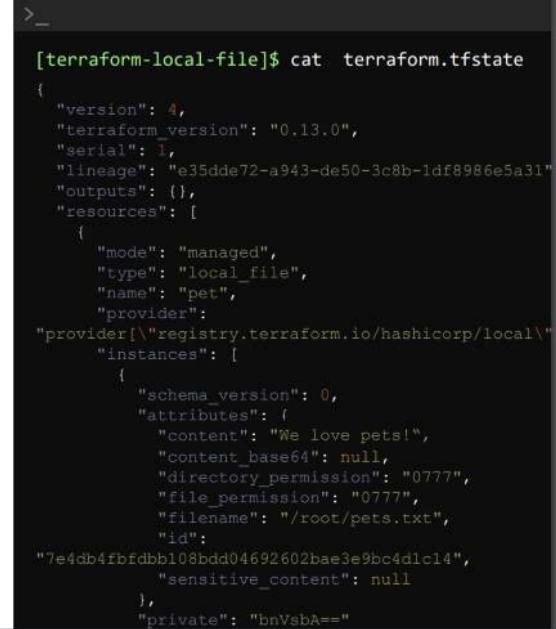
>_
$ terraform apply
local_file.pet: Refreshing state...
[+id=7e4db4fbfdbb108bdd04692602bae3e9bd1e1b68]

Terraform will perform the following actions:

  # local_file.pet must be replaced
  -/+ resource "local_file" "pet" {
      ~ content          = "I love pets!" -
      > "We love pets!" # forces replacement
      directory_permission = "0777"
      file_permission     = "0777"
      filename            = "/root/pets.txt"
      ~ id                =
    "7e4db4fbfdbb108bdd04692602bae3e9bd1e1b68" ->
    (known after apply)
  }

>_

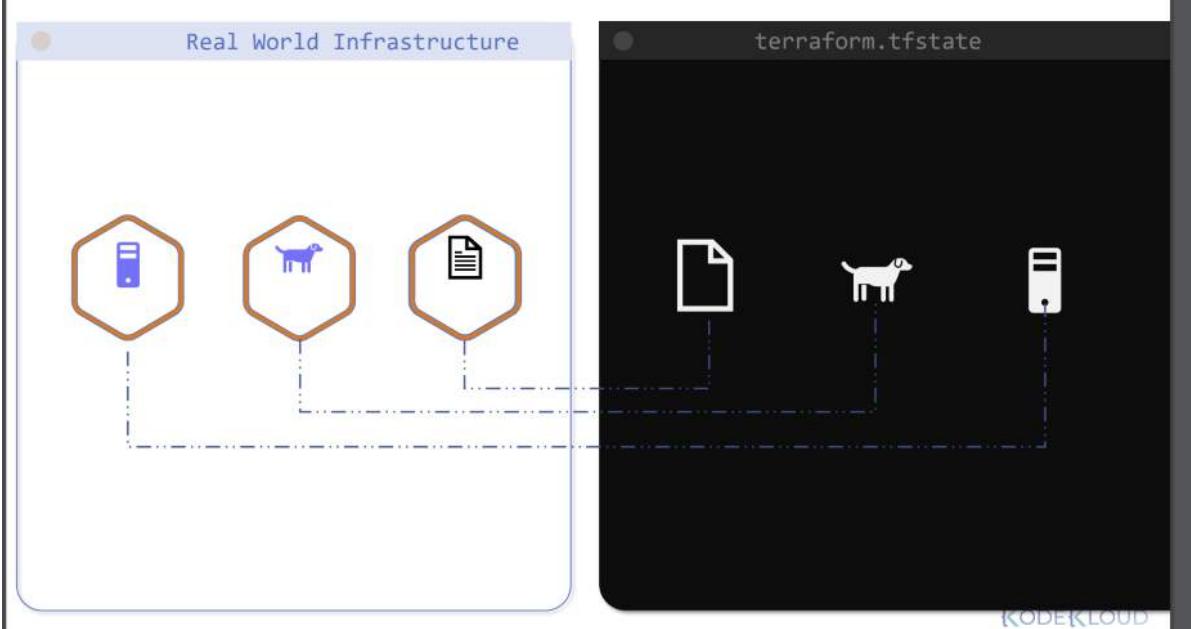
```



```

>_
[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": "registry.terraform.io/hashicorp/local",
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "content": "We love pets!",
            "content_base64": null,
            "directory_permission": "0777",
            "file_permission": "0777",
            "filename": "/root/pets.txt",
            "id": "7e4db4fbfdbb108bdd04692602bae3e9bd1e1b68",
            "sensitive_content": null
          },
          "private": "bnVsbA=="
        }
      ]
    }
  ]
}

```



KODEKLOUD

Purpose of state

It map resource configuration to the real world infrastructure

State file can be considered as for all the resources that are managed in the real world

When Terraform creates a resource, it records the identity in the state file.

Each resource created by Terraform will have a 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!"
}
```

KODEKLOUD

Tracking Metadata

```
$ terraform apply
```

```
.
```

```
.
```

```
.
```

```
Plan: 3 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.cat: Creating...
```

```
random_pet.my-pet: Creating...
```

```
local_file.cat: Creation complete after 0s
```

```
[id=fe44888891fc40342313bc44a1f1a8986520c89]
```

```
random_pet.my-pet: Creation complete after 0s [id=yaw]
```

```
local_file.pet: Creating...
```

```
local_file.pet: Creation complete after 0s
```

```
[id=28b373c6c1fa3fce132a518eadd0175c98f37f20]
```

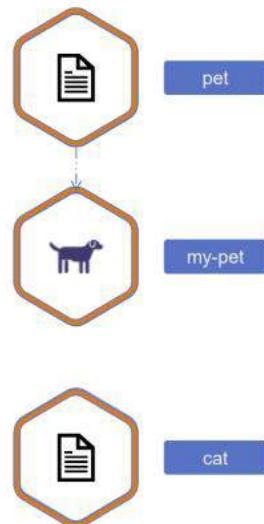
```
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
```

KODEKLOUD

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!"
}
```



KODEKLO

Tracking Metadata

```
main.tf
```

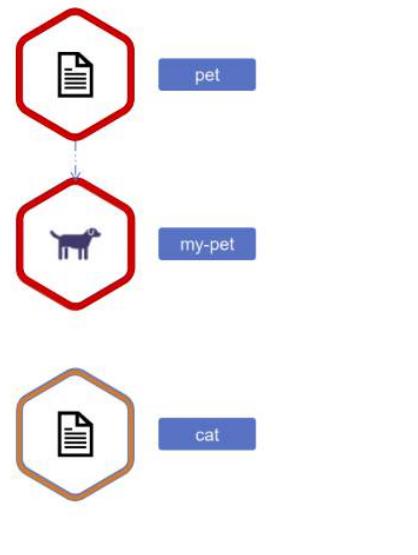
```
resource "local_file" "cat" {
  filename = "/root/cat.txt"
  content  = "I like cats too!"
}
```



Tracking Metadata

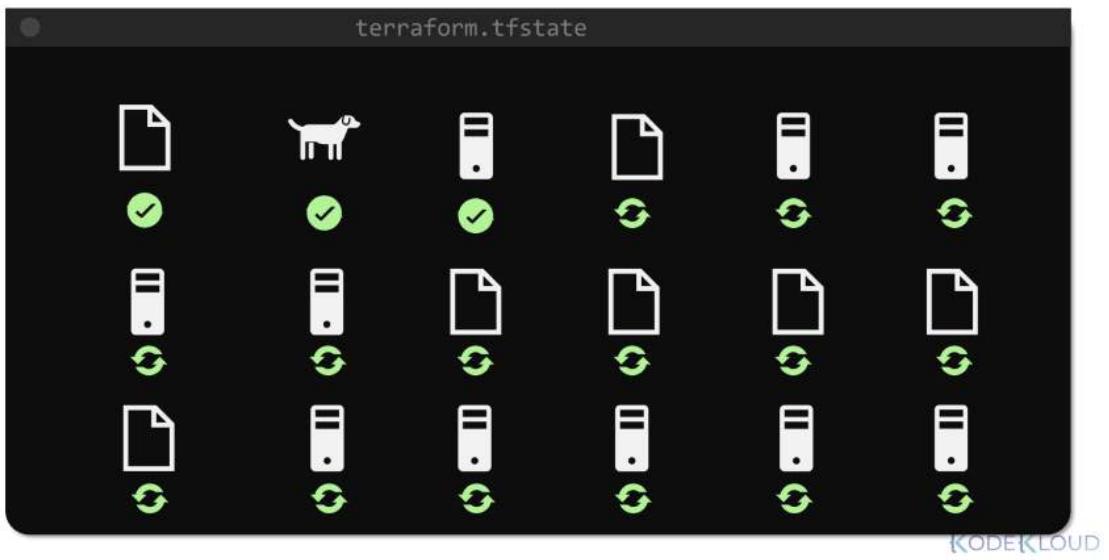
```
main.tf
resource "local_file" "cat" {
  filename = "/root/cat.txt"
  content  = "I like cats too!"
}

>_
$ 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



Performance

```
$ 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)
  }

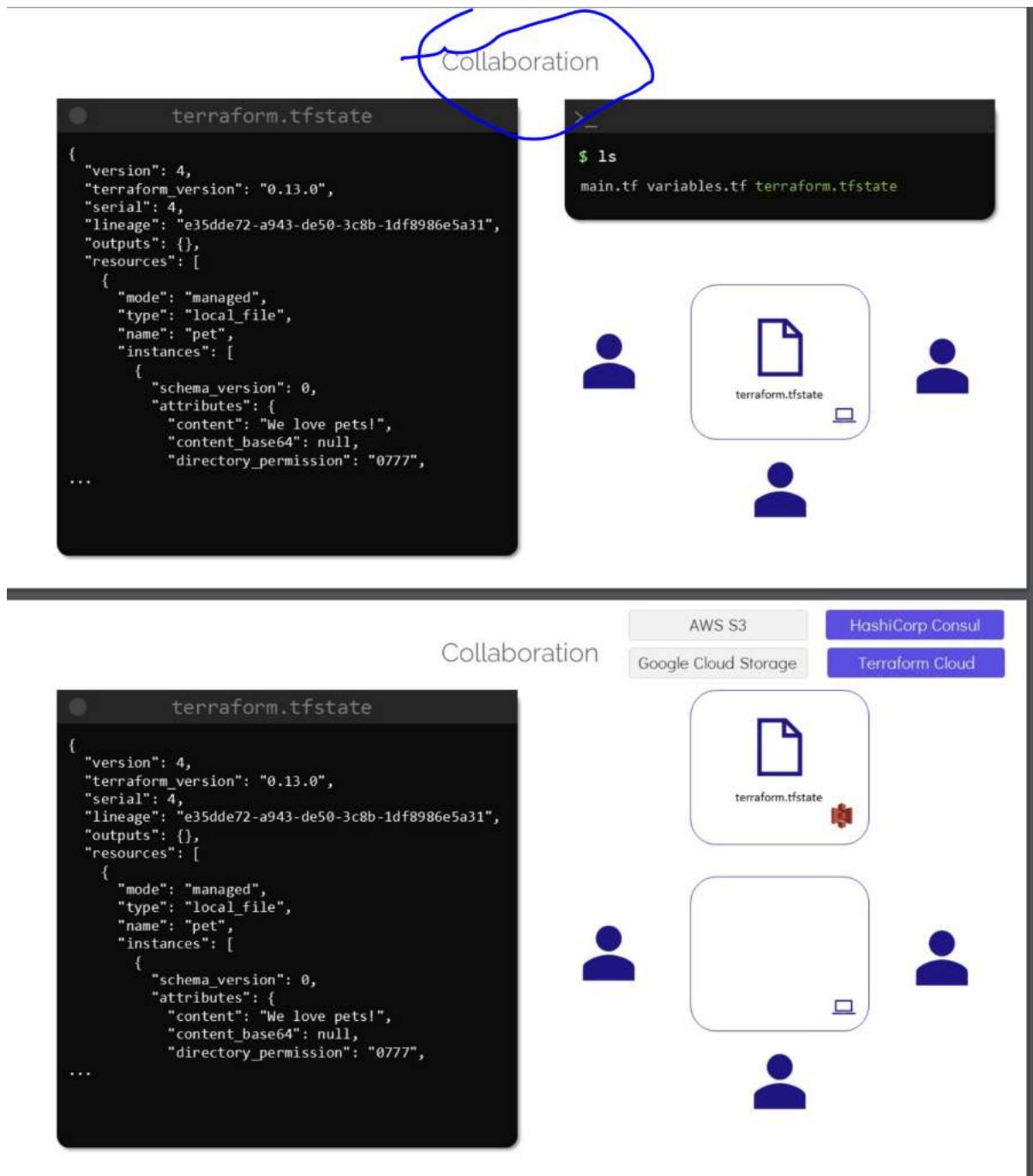
Plan: 1 to add, 0 to change, 1 to destroy.
```

A terminal window showing the output of "terraform plan --refresh=false". It displays an execution plan with resource actions and a summary of changes. The output includes:
\$ 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.

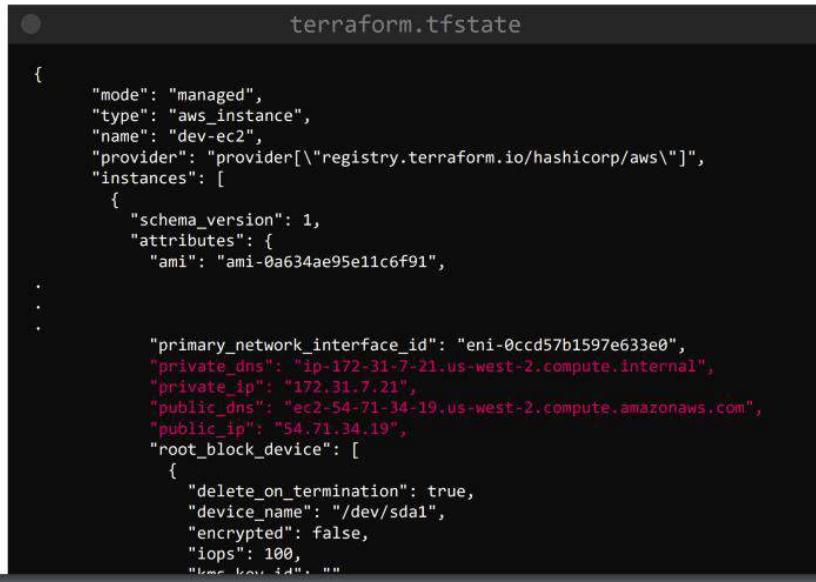


Terraform state consideration

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

The state file should always be stored in secure storage.

Sensitive Data



```
terraform.tfstate

{
  "mode": "managed",
  "type": "aws_instance",
  "name": "dev-ec2",
  "provider": "provider[\\"registry.terraform.io/hashicorp/aws\\"]",
  "instances": [
    {
      "schema_version": 1,
      "attributes": {
        "ami": "ami-0a634ae95e11c6f91",
        ...
        "primary_network_interface_id": "eni-0ccd57b1597e633e0",
        "private_dns": "ip-172-31-7-21.us-west-2.compute.internal",
        "private_ip": "172.31.7.21",
        "public_dns": "ec2-54-71-34-19.us-west-2.compute.amazonaws.com",
        "public_ip": "54.71.34.19",
        "root_block_device": [
          {
            "delete_on_termination": true,
            "device_name": "/dev/sda1",
            "encrypted": false,
            "iops": 100,
            "kms_key_id": "",
            "volume_id": "vol-070720a3636979c22",
            "volume_size": 8
          }
        ]
      }
    }
  ]
}
```

KODEKLOUD

Terraform State Considerations

Remote State Backends



Version Control



```
terraform.tfstate

{
  "mode": "managed",
  "type": "aws_instance",
  "name": "dev-ec2",
  "provider": "provider[\\"registry.terraform.io/hashicorp/aws\\"]",
  "instances": [
    {
      "schema_version": 1,
      "attributes": {
        "ami": "ami-0a634ae95e11c6f91",
        ...
        "primary_network_interface_id": "eni-0ccd57b1597e633e0",
        "private_dns": "ip-172-31-7-21.us-west-2.compute.internal",
        "private_ip": "172.31.7.21",
        "public_dns": "ec2-54-71-34-19.us-west-2.compute.amazonaws.com",
        "public_ip": "54.71.34.19",
        "root_block_device": [
          {
            "delete_on_termination": true,
            "device_name": "/dev/sda1",
            "encrypted": false,
            "iops": 100,
            "kms_key_id": "",
            "volume_id": "vol-070720a3636979c22",
            "volume_size": 8
          }
        ]
      }
    }
  ]
}
```



```
main.tf

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

KODEKLOUD

Section 6: working with terraform

Terraform commands

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

Terraform fmt

terraform fmt



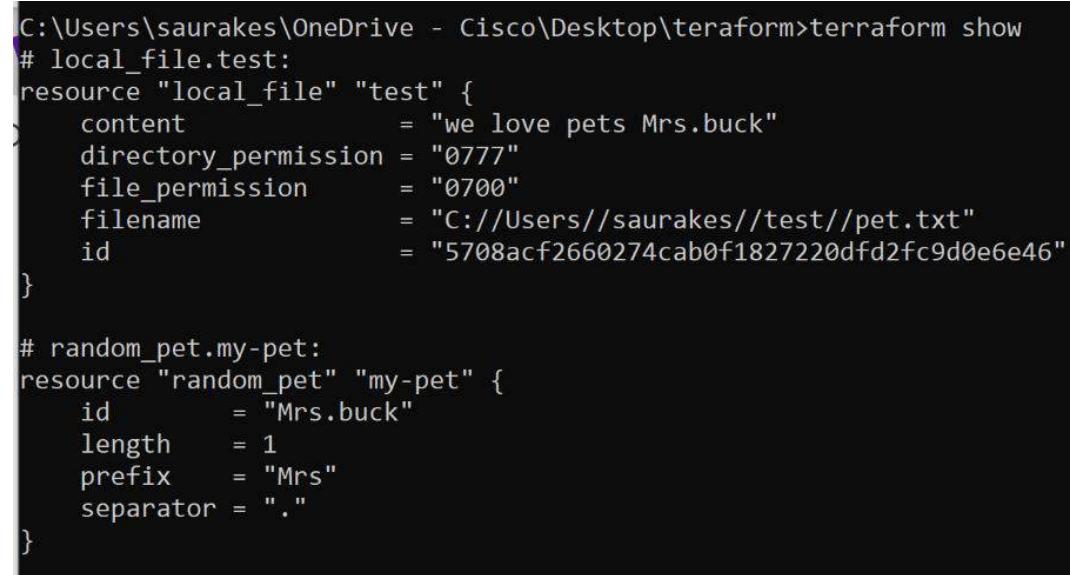
The screenshot shows two terminal windows side-by-side. The left window displays the contents of a file named 'main.tf' with the following code:

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

The right window shows the command being run: '\$ terraform fmt' followed by the file name 'main.tf'.

Terraform show :

Print out current state of infrastructure



The screenshot shows the output of the 'terraform show' command. It lists resources and their current state, including local_file.test and random_pet.my-pet.

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

The image shows two terminal windows side-by-side. The left window displays the output of the command `terraform show`, which prints a Terraform configuration block for a `local_file` resource named `pet`. The right window displays the output of the command `terraform show -json`, which prints the same configuration block in JSON format.

```
$ terraform show
# local_file.pet:
resource "local_file" "pet" {
  content          = "We love pets!"
  directory_permission = "0777"
  file_permission    = "0777"
  filename          = "/root/pets.txt"
  id                =
}

$cba595b7d9f94ba1107a46f3f731912d95fb3d2c"
```

```
$ terraform show -json
{"format_version": "0.1", "terraform_version": "0.13.0", "values": {"root_module": {"resources": [{"address": "local_file.pet", "mode": "managed", "type": "local_file", "name": "pet", "provider_name": "registry.terraform.io/hashicorp/local", "schema_version": 0, "values": {"content": "We love pets!", "content_base64": null, "directory_permission": "0777", "file_permission": "0777", "filename": "/root/pets.txt", "id": "cba595b7d9f94ba1107a46f3f731912d95fb3d2c", "sensitive_content": null}}}]}}
```

Terraform providers

To see list of all providers use in the configuration directory

The image shows a terminal window displaying the output of the command `terraform providers`. It lists the providers required by configuration and state.

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

The terminal shows the execution of the command `$ terraform providers`. The output indicates that the `hashicorp/local` provider is required by configuration and state, and it is being mirrored from `registry.terraform.io`.

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

>_
$ terraform providers
Providers required by configuration:
└── provider[registry.terraform.io/hashicorp/local]

Providers required by state:
provider[registry.terraform.io/hashicorp/local]

$ terraform providers mirror /root/terraform/new_local_file
- Mirroring hashicorp/local...
- Selected v1.4.0 with no constraints
- Downloading package for windows_amd64...
- Package authenticated: signed by HashiCorp
```

KODEKLOUD

Terraform output

The terminal shows the execution of the command `$ terraform output`, which displays two outputs: `content` and `pet-name`.

```
main.tf
resource "local_file" "pet" {
    filename      = "/root/pets.txt"
    content       = "We love pets!"
    file_permission = "0777"
}
resource "random_pet" "cat" {
    length      = "2"
    separator   = "-"
}
output content {
    value      = local_file.pet.content
    sensitive  = false
    description = "Print the content of the file"
}
output pet-name {
    value      = random_pet.cat.id
    sensitive  = false
    description = "Print the name of the pet"
}

>_
$ terraform output
content = We love pets!
pet-name = huge-owl

$ terraform output pet-name
pet-name = huge-owl
```

KODEKLOUD

Terraform graph :

Will show visual representation of dependency

terraform graph

```
main.tf
```

```

resource "local_file" "pet" {
  filename = "/root/pets.txt"
  content = "My favorite pet is ${random_pet.my-pet.id}"
}
resource "random_pet" "my-pet" {
  prefix = "Mr"
  separator = "."
  length = "1"
}

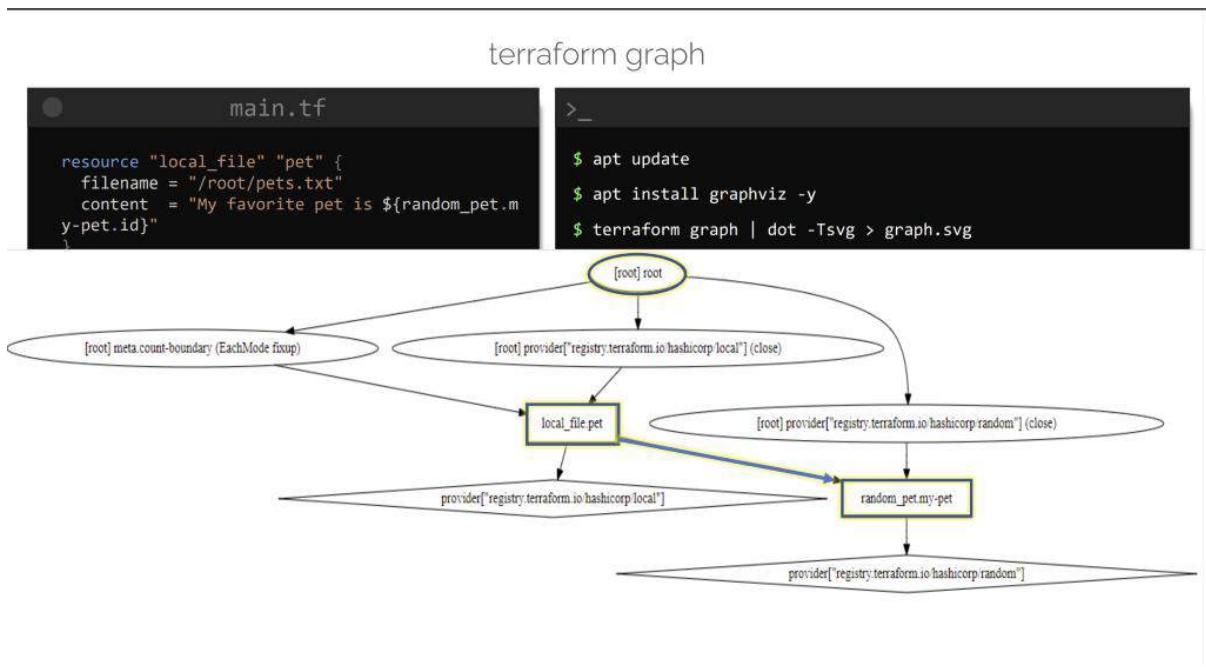
```

```
>_
```

```

$ terraform graph
graph TD
    compound = "true"
    newrank = "true"
    subgraph "root" {
        "[root] local_file.pet (expand)" [label = "local_file.pet", shape = "box"]
        "[root] provider[\"registry.terraform.io/hashicorp/local\"]" [label = "provider[\"registry.terraform.io/hashicorp/local\"]", shape = "diamond"]
        "[root] provider[\"registry.terraform.io/hashicorp/random\"]" [label = "provider[\"registry.terraform.io/hashicorp/random\"]", shape = "diamond"]
        "[root] random_pet.my-pet (expand)" [label = "random_pet.my-pet", shape = "box"]
        "[root] local_file.pet (expand)" --> "[root] provider[\"registry.terraform.io/hashicorp/local\"]"
        "[root] local_file.pet (expand)" --> "[root] provider[\"registry.terraform.io/hashicorp/random\"]"
        "[root] random_pet.my-pet (expand)" --> "[root] meta.count-boundary (EachMode fixup)"
        "[root] meta.count-boundary (EachMode fixup)" --> "[root] local_file.pet (expand)"
        "[root] provider[\"registry.terraform.io/hashicorp/local\"] (close)" --> "[root] provider[\"registry.terraform.io/hashicorp/random\"]"
        "[root] provider[\"registry.terraform.io/hashicorp/random\"] (close)" --> "[root] random_pet.my-pet (expand)"
    }

```



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 **immutable 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

Immutable Infrastructure



KODEKLOUD

Immutable Infrastructure

```
main.tf
```

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

```
$ terraform apply
```

```
# local_file.pet must be replaced
-/+ resource "local_file" "pet" {
    content          = "We love pets!"
    directory_permission = "0777"
    ~ file_permission      = "0777" -> "0700" # forces
replacement
    filename         = "/root/pet.txt"
    ~ id              =
"5f8fb950ac60f7f23ef968097cda0a1fd3c11bdf" -> (known after
apply)
}

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

local_file.pet: Destroying...
[id=5f8fb950ac60f7f23ef968097cda0a1fd3c11bdf]
local_file.pet: Destruction complete after 0s
local_file.pet: Creating...
local_file.pet: Creation complete after 0s
[id=5f8fb950ac60f7f23ef968097cda0a1fd3c11bdf]

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



KODEKLOUD

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



The screenshot illustrates the immutable infrastructure approach. On the left, a file named `main.tf` contains the following Terraform code:

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

On the right, a terminal window shows the output of the `terraform apply` command:

```
>_
$ terraform apply
local_file.pet must be replaced
  -/+ resource "local_file" "pet" {
      content          = "We love pets!"
      directory_permission = "0777"
      ~ file_permission    = "0777" -> "0700" # forces replacement
      filename         = "/root/pet.txt"
      ~ id              =
"5f8fb950ac60f7f23ef968097cda0a1fd3c11bdf" -> (known after apply)
}

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

local_file.pet: Destroying...
[id=5f8fb950ac60f7f23ef968097cda0a1fd3c11bdf]
local_file.pet: Destruction complete after 0s
local_file.pet: Creating...
local_file.pet: Creation complete after 0s
[id=5f8fb950ac60f7f23ef968097cda0a1fd3c11bdf]

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

Below the terminal window, there are two orange hexagonal icons, each containing a document symbol.

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

The diagram illustrates the `create_before_destroy` lifecycle configuration. On the left, the `main.tf` file contains a `local_file` resource with `create_before_destroy = true`. Two orange hexagonal icons below represent the state of the resources: one with a lock icon indicating it's locked, and another with a document icon indicating it's unlocked.

On the right, the terminal output shows the execution of `terraform apply`. It highlights the replacement of the local file resource, showing the original content ("We love pets!") and the modified content ("We love pets!"). The terminal also displays the plan and apply results, confirming the creation and destruction of the resource.

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

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

This will especially use to resource accidentally deleted

The diagram illustrates the `prevent_destroy` lifecycle configuration. On the left, the `main.tf` file contains a `local_file` resource with `prevent_destroy = true`. An orange hexagonal icon below represents the state of the resource, showing a lock icon indicating it's locked.

On the right, the terminal output shows the execution of `terraform apply`. It highlights the refresh of the state for the `local_file.my-pet` resource. An error message is displayed: "Error: Instance cannot be destroyed" followed by the plan details. The terminal also displays the apply results.

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

ignore_changes



```
main.tf
resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"
  tags = {
    Name = "ProjectA-Webserver"
  }
}

>_
$ terraform apply
aws_instance.webserver: Refreshing state... [id=i-05cd83b221911acd5]

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
~ update in-place

Terraform will perform the following actions:

# aws_instance.webserver will be updated in-place
~ resource "aws_instance" "webserver" {

  ~ tags
    ~ "Name" = "ProjectB-WebServer" -> "ProjectA-WebServer"

}

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

ProjectB-WebServer i-05cd83b221911acd5 Running t2.micro 2/2 checks ... EKLOUD
```

ignore_changes



```
main.tf
resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"
  tags = {
    Name = "ProjectA-Webserver"
  }
  lifecycle {
    ignore_changes = [
      tags
    ]
  }
}

>_
$ terraform apply
aws_instance.webserver: Refreshing state... [id=i-05cd83b221911acd5]

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

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.

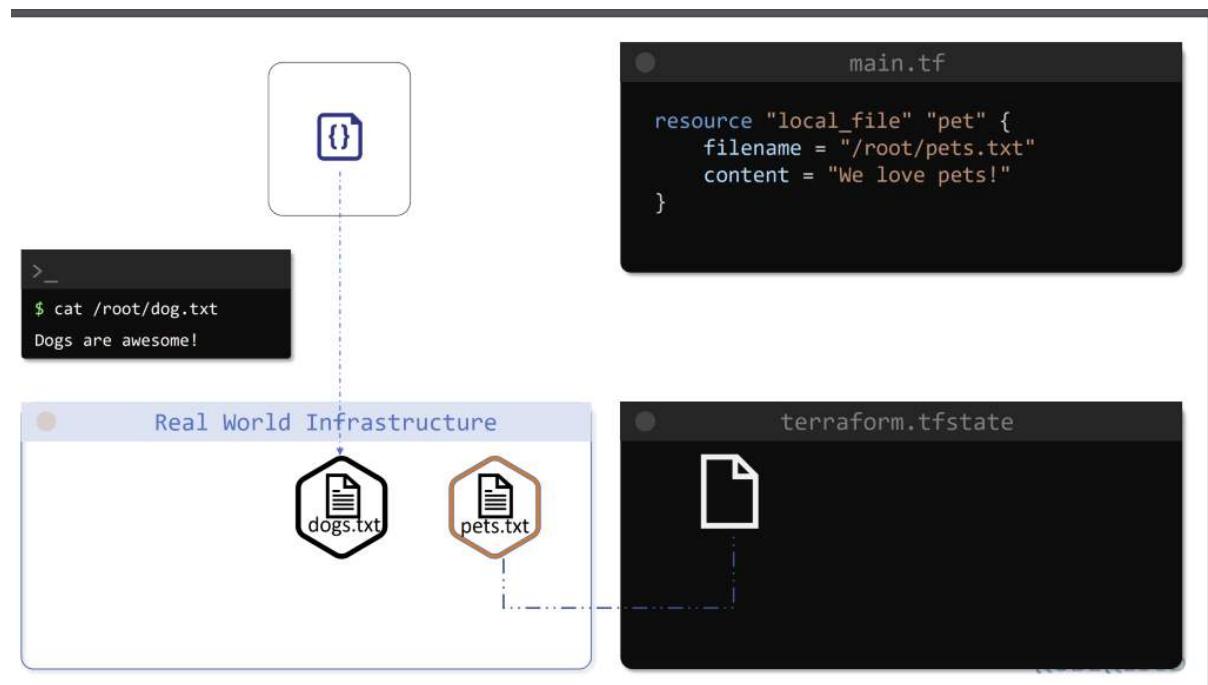
We know that terraform make use of configuration files along with the state file to provision infrastructure resources

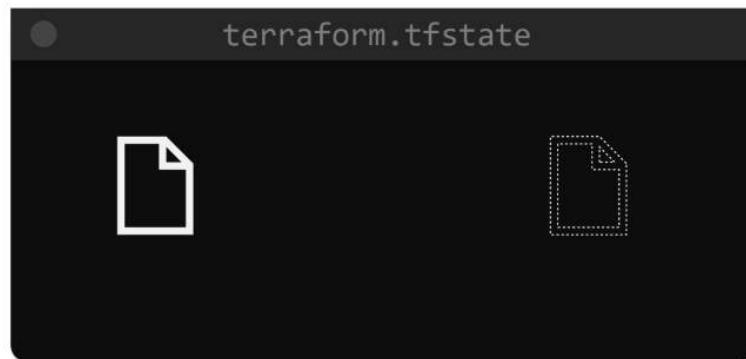
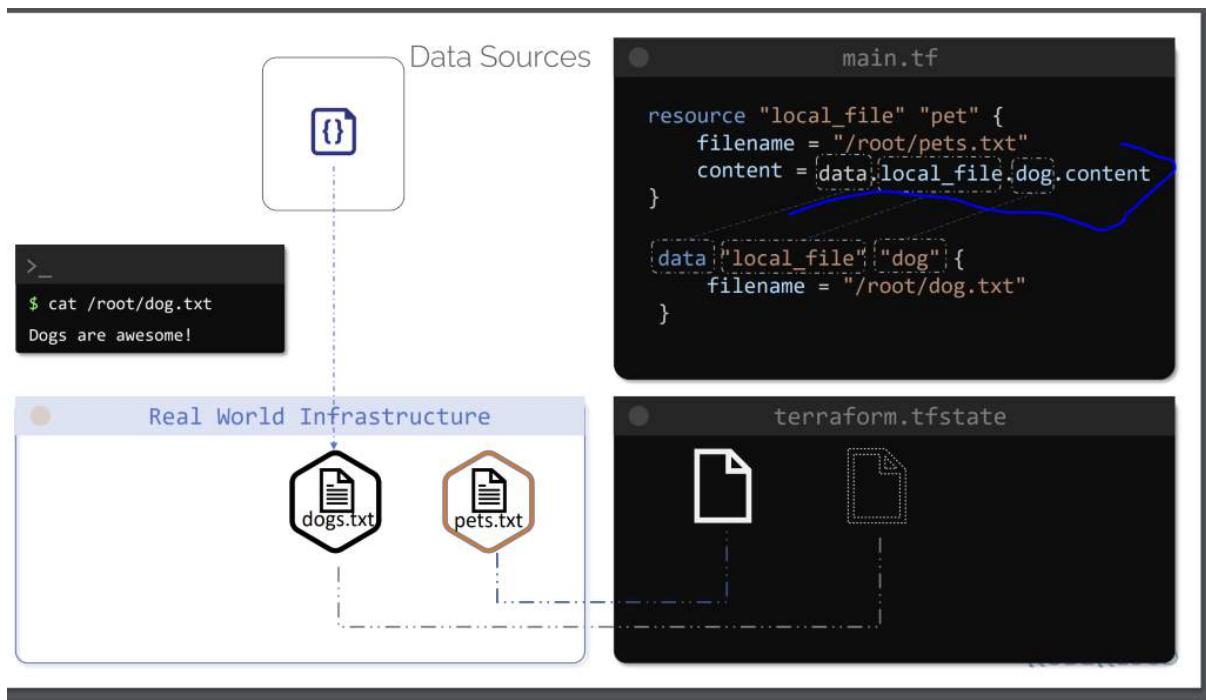
Example :

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



```
count
```

```
main.tf
```

```
resource "local_file" "pet" {  
    filename = var.filename  
    count   = 3  
}
```

```
variables.tf
```

```
variable "filename" {  
    default = "/root/pets.txt"  
}  
>_  
$ terraform plan  
[Output Truncated]  
Terraform will perform the following actions:  
...  
# local_file.pet[2] will be created  
+ resource "local_file" "pet" {  
    + directory_permission = "0777"  
    + file_permission     = "0777"  
    + filename            = "/root/pets.txt"  
    + id                  = (known after apply)  
}  
Plan: 3 to add, 0 to change, 0 to destroy.
```

KODEKLOUD

Length Function

The image shows a terminal window with two tabs: `main.tf` and `variables.tf`, and a separate terminal window below.

`main.tf` contains:

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

`variables.tf` contains:

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

Below the code, there are three icons representing files labeled `pet[0]`, `pet[1]`, and `pet[2]`.

The terminal window below shows the command `$ ls /root` and its output:

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

Length Function

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

LengthFunction

The screenshot shows a Terraform workspace with two files and their corresponding terminal outputs.

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"
  ]
}
```

Terminal Outputs:

- main.tf:** Shows three hexagonal icons labeled `pet[0]`, `pet[1]`, and `pet[2]`.
- variables.tf:** Shows the command `$ ls /root` outputting `pets.txt`, `dogs.txt`, and `cats.txt`.
- Terminal 1 (Left):** Shows the command `$ terraform apply` followed by a list of actions:
 - # local_file.pet[0] will be created
 - + resource "local_file" "pet" {
 - + directory_permission = "0777"
 - + file_permission = "0777"
 - + filename = "/root/pets.txt"
 - + id = (known after apply)
- Terminal 2 (Right):** Shows the command `$ ls /root` outputting `pet.txt`, `dogs.txt`, and `cats.txt`.

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

This screenshot shows two terminal windows. The left window contains the `main.tf` file:

```
resource "local_file" "pet" {
  filename = each.value
  for_each = var.filename
}
```

The right window contains the `variables.tf` file:

```
variable "filename" {
  type=set(string)
  default = [
    "/root/pets.txt",
    "/root/dogs.txt",
    "/root/cats.txt"
  ]
}
```

Below the terminals, there are three hexagonal icons labeled `pet[0]`, `pet[1]`, and `pet[2]`, each containing a document icon.

Output of `terraform plan`:

```
$ terraform plan
Terraform will perform the following actions:
# local_file.pet["/root/cats.txt"] will be created
+ resource "local_file" "pet" {
  + directory_permission = "0777"
  + file_permission      = "0777"
  + filename              = "/root/cats.txt"
}
... <output trimmed>
Plan: 3 to add, 0 to change, 0 to destroy.
```

This screenshot shows two terminal windows. The left window contains the `main.tf` file:

```
resource "local_file" "pet" {
  filename = each.value
  for_each = toset(var.filename)
}
```

The right window contains the `variables.tf` file:

```
variable "filename" {
  type=set(string)
  default = [
    "/root/pets.txt",
    "/root/dogs.txt",
    "/root/cats.txt"
  ]
}
```

Below the terminals, there are three hexagonal icons labeled `pet[0]`, `pet[1]`, and `pet[2]`, each containing a document icon.

Output of `terraform plan`:

```
$ terraform plan
Terraform will perform the following actions:
# local_file.pet["/root/cats.txt"] will be created
+ resource "local_file" "pet" {
  + directory_permission = "0777"
  + file_permission      = "0777"
  + filename              = "/root/cats.txt"
}
... <output trimmed>
Plan: 3 to add, 0 to change, 0 to destroy.
```

for_each

The terminal shows two files: `main.tf` and `variables.tf`.

`main.tf` contains:

```
resource "local_file" "pet" {
    filename = each.value
    for_each = toset(var.filename)
}
output "pets" {
    value = local_file.pet
}
```

`variables.tf` contains:

```
variable "filename" {
    type = list(string)
    default = [
        "/root/dogs.txt",
        "/root/cats.txt"
    ]
}
```

Below the code, there are three hexagonal icons labeled `pet[0]`, `pet[1]`, and `pet[2]`, each containing a document icon.

On the right, the terminal output shows the results of `terraform plan`:

```
$ terraform plan
Terraform will perform the following actions:
# local_file.pet["/root/pets.txt"] will be destroyed
+ resource "local_file" "pet" {
    + directory_permission = "0777"
    + file_permission      = "0777"
    + filename              = "/root/pets.txt"
}
... <output trimmed>
Plan: 0 to add, 0 to change, 1 to destroy.
```

for_each

The terminal shows two files: `main.tf` and `variables.tf`.

`main.tf` contains:

```
resource "local_file" "pet" {
    filename = each.value
    for_each = toset(var.filename)
}
output "pets" {
    value = local_file.pet
}
```

`variables.tf` contains:

```
variable "filename" {
    type = list(string)
    default = [
        "/root/cats.txt",
        "/root/dogs.txt"
    ]
}
```

Below the code, there are three hexagonal icons labeled `pet[0]`, `pet[1]`, and `pet[2]`, each containing a document icon.

On the right, the terminal output shows the results of `terraform output`:

```
* terraform output
pets = {
    "/root/cats.txt" = {
        "directory_permission" = "0777"
        "file_permission"      = "0777"
        "filename"             = "/root/cats.txt"
        "id"                  = "da39a3ee5e6b4b0d3255bfef95601890af80709"
    }
    "/root/dogs.txt" = {
        "directory_permission" = "0777"
        "file_permission"      = "0777"
        "filename"             = "/root/dogs.txt"
        "id"                  = "da39a3ee5e6b4b0d3255bfef95601890af80709"
    }
}
```

A blue curved arrow points from the `"/root/cats.txt"` entry in the output back to the `for_each` line in the `main.tf` code.

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" = "da39a3ee5e6b4b0d3255bfef95601890af80709"
  },
  {
    "directory_permission" = "0777"
    "file_permission" = "0777"
    "filename" = "/root/dogs.txt"
    "id" = "da39a3ee5e6b4b0d3255bfef95601890af80709"
  },
  {
    "directory_permission" = "0777"
    "file_permission" = "0777"
    "filename" = "/root/cats.txt"
    "id" = "da39a3ee5e6b4b0d3255bfef95601890af80709"
  },
]
```

```
for_each
```

```
>_
$ terraform output
pets = {
  "/root/cats.txt" = {
    "directory_permission" = "0777"
    "file_permission" = "0777"
    "filename" = "/root/cats.txt"
    "id" = "da39a3ee5e6b4b0d3255bfef95601890af80709"
  }

  "/root/dogs.txt" = {
    "directory_permission" = "0777"
    "file_permission" = "0777"
    "filename" = "/root/dogs.txt"
    "id" = "da39a3ee5e6b4b0d3255bfef95601890af80709"
  }
}
```

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

The screenshot shows the Terraform Registry interface. On the left, there is a terminal window displaying the following Terraform configuration:

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

On the right, the Terraform Registry page for the `local` provider is shown. The URL is `https://registry.terraform.io/providers/hashicorp/local`. The page includes the HashiCorp logo, a search bar, and navigation links for Providers, hashicorp, local, Version 2.0.0, and Latest Version. The provider details section for `local` shows:

- local** (Official) by HashiCorp
- Utility**
- Description: Used to manage local resources, such as creating files
- Version: 2.0.0 (Published 9 days ago)
- Installs: 15.8M
- Source Code: [hashicorp/terraform-provider-local](#)

This screenshot is similar to the one above, but a dropdown menu is open over the version list. The menu is titled "LATEST VERSION" and lists several versions of the provider:

- Version 2.0.0 (selected)
- Version 1.4.0
- Version 1.3.0
- Version 1.2.2
- Version 1.2.1

The rest of the page content is identical to the first screenshot, including the terminal output and provider details.

The screenshot shows the HashiCorp Local provider documentation page. On the left, there is a code editor window titled "main.tf" containing the following Terraform code:

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

On the right, there is a "How to use this provider" section with the following content:

To install this provider, copy and paste this code into your Terraform configuration. Then, run `terraform init`.

Terraform 0.13 Latest

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

Below this, there is another code editor window titled "main.tf" containing the following Terraform code, with a blue oval highlighting the "required_providers" block:

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

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

On the right, there is a "How to use this provider" section with the same content as the top one.

The terminal window shows the contents of a file named `main.tf` on the left and the output of the `terraform init` command on the right.

`main.tf` content:

```
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 init` output:

```
$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/local versions matching "1.4.0"...
- Installing hashicorp/local v1.4.0...
- Installed hashicorp/local v1.4.0 (signed by HashiCorp)

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.
```

KODEKLOUD

The terminal window shows the contents of a file named `main.tf` on the left and the output of the `terraform init` command on the right. A blue arrow points from the specific version range in the configuration to the corresponding output line in the terminal.

`main.tf` content:

```
main.tf
terraform {
  required_providers {
    local = {
      source = "hashicorp/local"
      version = "> 1.2.0, < 2.0.0, != 1.4.0"
    }
  }
}

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

`terraform init` output:

```
$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/local versions matching "> 1.2.0, < 2.0.0, != 1.4.0"...
- Installing hashicorp/local v1.3.0...
- Installed hashicorp/local v1.3.0 (signed by HashiCorp)

Terraform has been successfully initialized!
```

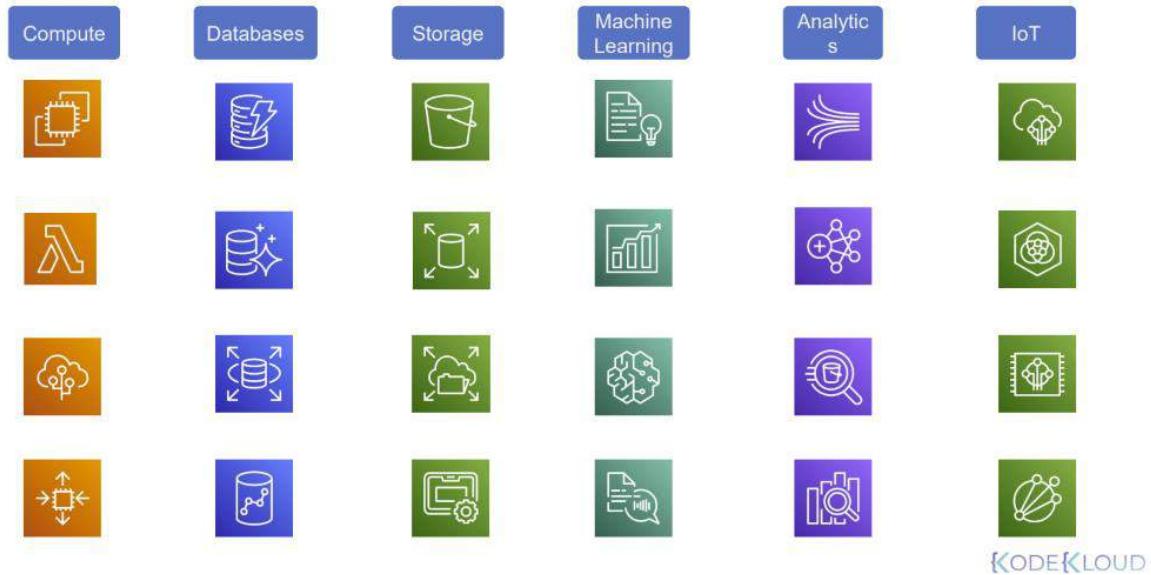
Terraform will download either 1.2 or incremental version such as 1.3 , 1.4 etc

Section 7: terraform with aws

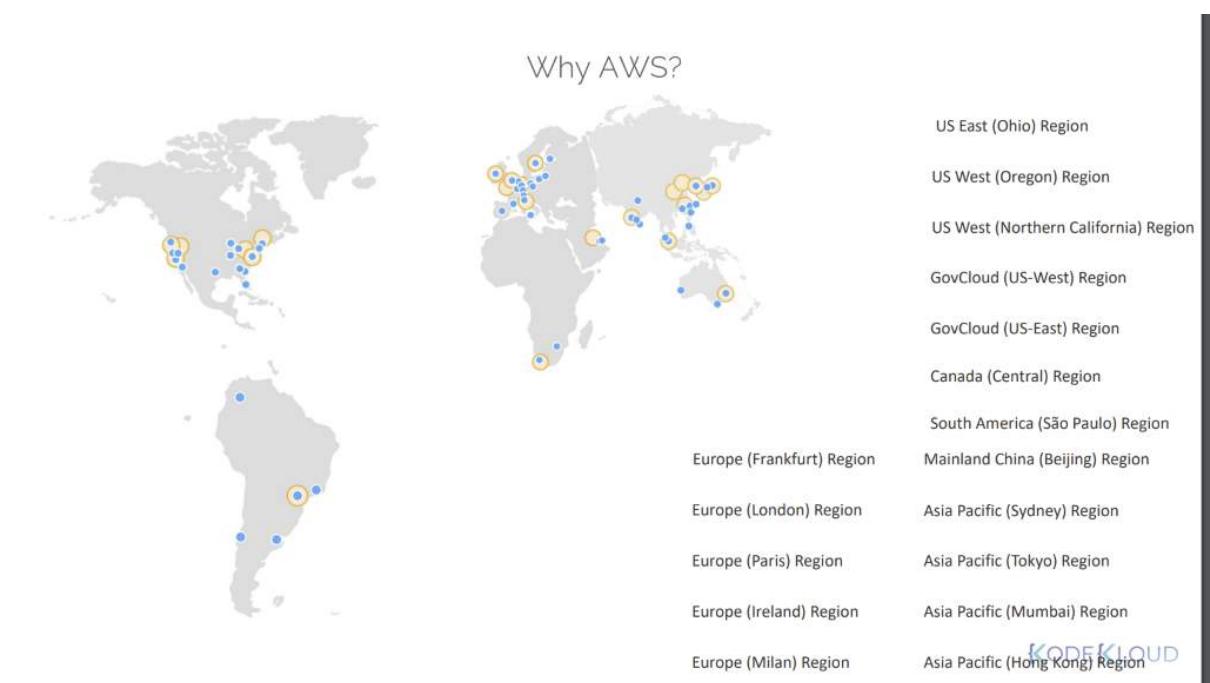
Getting started with aws

Aws: cloud computing platform. It offers 100 of services

Why AWS?

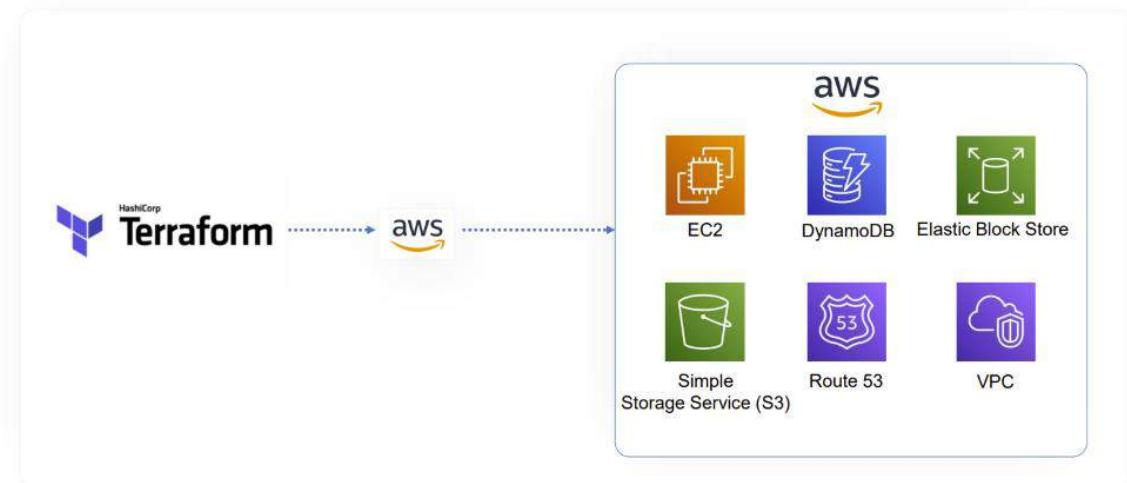


Aws allow global cloud infrastructure. This allow to deploy services in number of globally distributed region



Terraform manage aws cloud based service

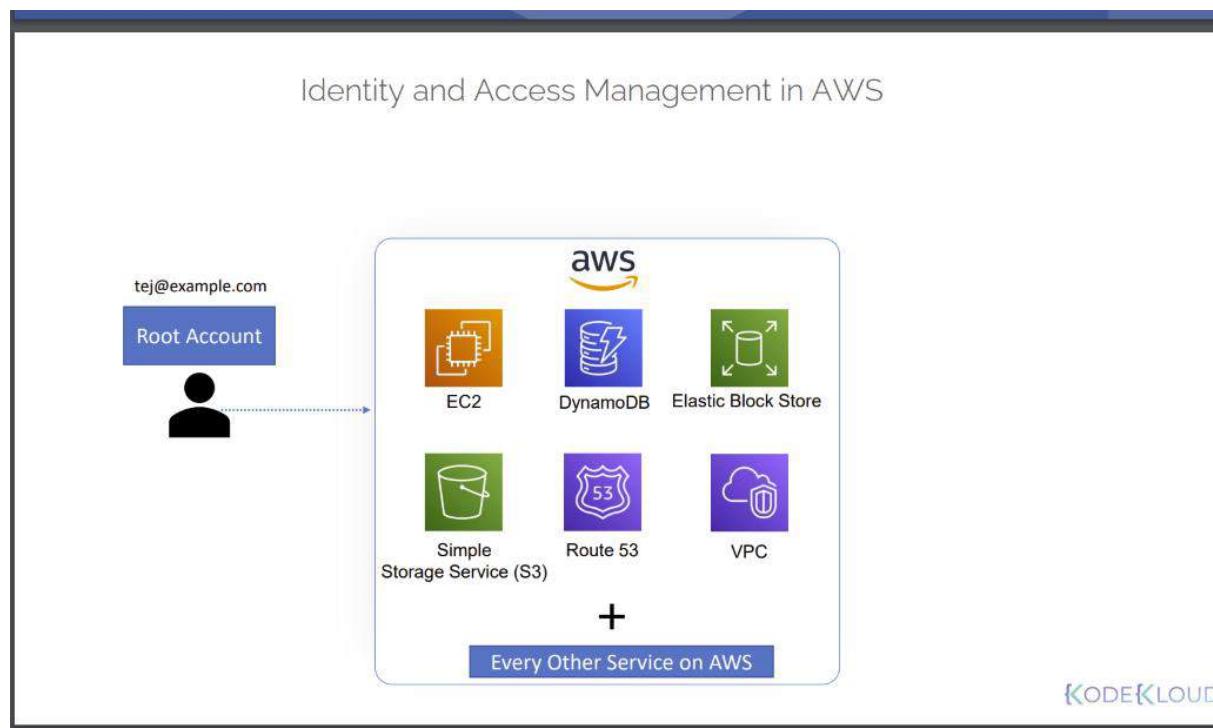
Why AWS with Terraform?



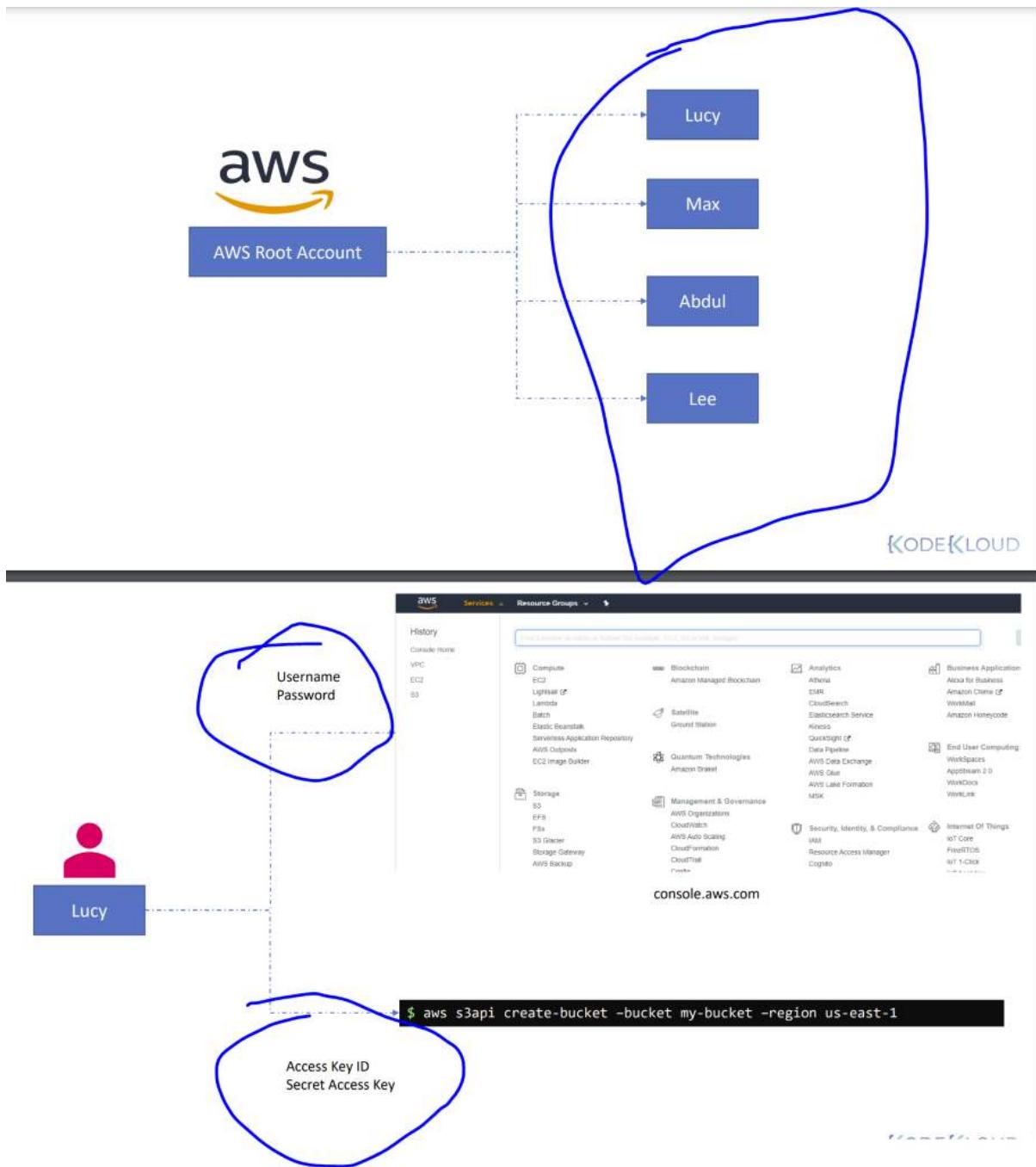
Introduction to IAM

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

The screenshot shows the AWS IAM console. On the left, there is a list of users with Lucy selected. In the center, the details for the AdministratorAccess policy are shown. A preview of the policy document is displayed:

```
{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Action": "*", "Resource": "*" } ] }
```

A blue button labeled "AdministratorAccess Policy" is visible. Below the policy details, there is a link "AdministratorAccess" next to an icon of a key and a document.

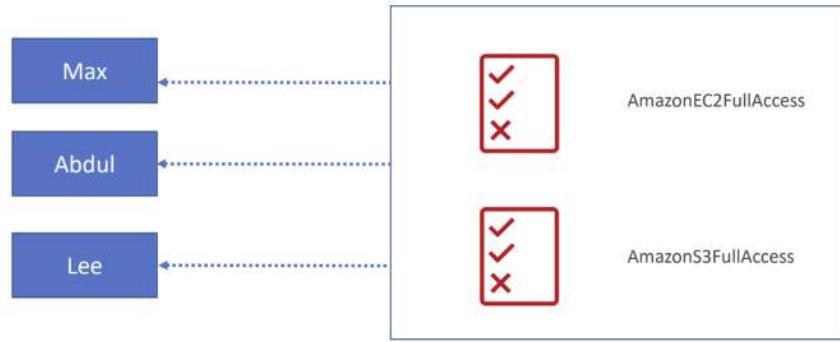
The screenshot shows the AWS IAM console with the "Filter policies" dropdown set to "Policy name". The search bar contains "AdministratorAccess". The results table shows the following data:

Policy name	Type	Used as
AdministratorAccess	Job function	Permissions policy (2)
AlexaForBusinessDeviceSetup	AWS managed	None
AlexaForBusinessFullAccess	AWS managed	None
AlexaForBusinessGatewayExecution	AWS managed	None
AlexaForBusinessLifesizeDelegatedAccessPolicy	AWS managed	None
AlexaForBusinessPolyDelegatedAccessPolicy	AWS managed	None
AlexaForBusinessReadOnlyAccess	AWS managed	None
AmazonAPIGatewayAdministrator	AWS managed	None

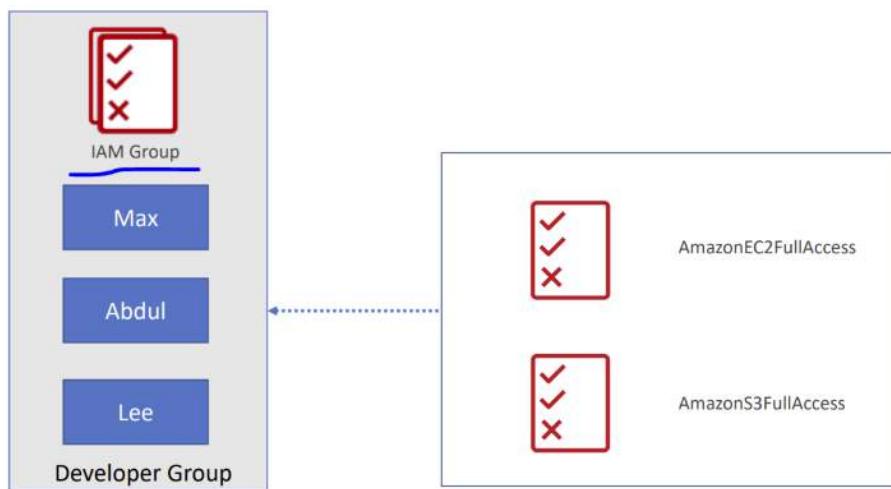
At the bottom, a link to the AWS documentation is provided: https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_job-functions.html.

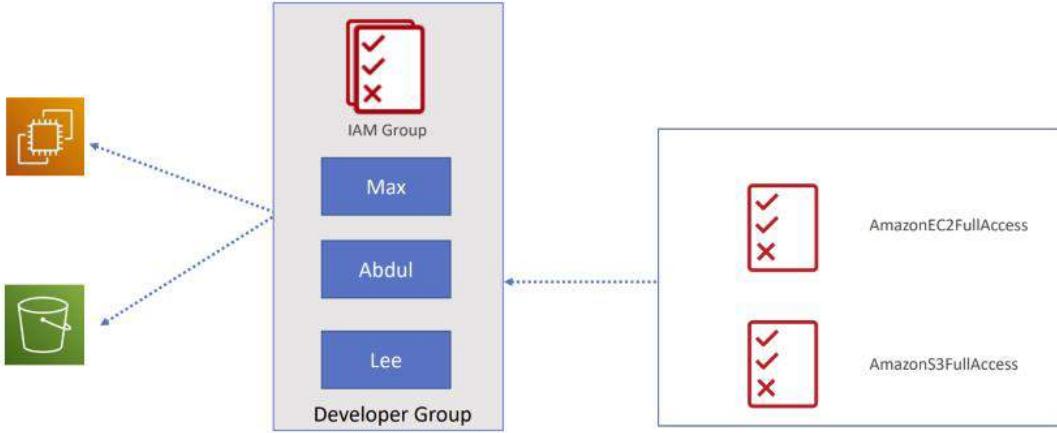
Job Function	Policy Name
Administrator	AdministratorAccess
Billing	Billing
Database Administrator	DatabaseAdministrator
Network Administrator	NetworkAdministrator
View-Only User	ViewOnlyAccess

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



KO





KODEKLOUD



Note : imp concept

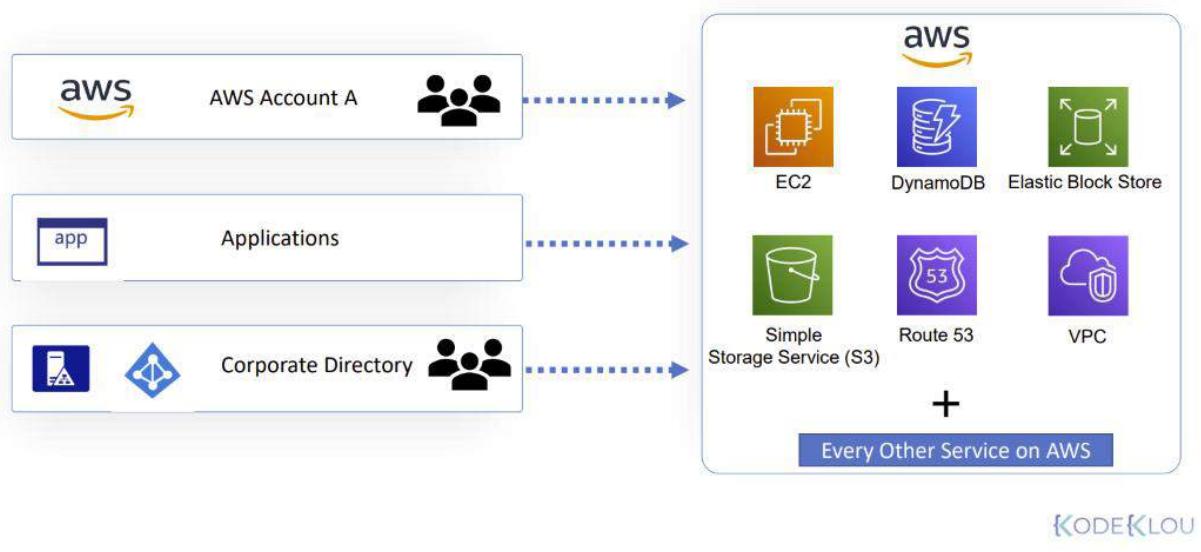
Imp point : just like regular user resources in aws(ec2 , dynamo db) also do not have permission to interact with each other by default. They need to be granted access specifically.

*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 used to access another AWS account



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

A screenshot of the AWS IAM Policy creation interface. The top part shows a preview of the policy named 'CreateEC2TagsPolicy' with a red checkmark icon. The JSON code for the policy is displayed below:

```
{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Action": [ "ec2:DeleteTags", "ec2:CreateTags" ], "Resource": "*" } ] }
```

The bottom part shows the policy details in the AWS console:

Policy name	Type	Used as	Description
CreateEC2TagsPolicy	Customer managed	None	Permission to create and delete EC2 Tags

The KODEKLOUD logo is visible at the bottom right.

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 a global service. Once we create user or group , it will be available in all regions of account

The screenshot shows the AWS Services page. On the left, there's a sidebar with 'History' and links to 'Console Home', 'VPC', 'EC2', and 'S3'. The main area is a grid of service icons and names. Categories include:

- Compute:** EC2, Lightsail, Lambda, Batch, Elastic Beanstalk, Serverless Application Repository, AWS Outposts, EC2 Image Builder.
- Storage:** S3, EFS, FSx, S3 Glacier, Storage Gateway, AWS Backup.
- Blockchain:** Amazon Managed Blockchain.
- Satellite:** Ground Station.
- Quantum Technologies:** Amazon Braket.
- Management & Governance:** AWS Organizations, CloudWatch, AWS Auto Scaling, CloudFormation, CloudTrail, Config.
- Analytics:** Athena, EMR, CloudSearch, Elasticsearch Service, Kinesis, QuickSight, Data Pipeline, AWS Data Exchange, AWS Glue, AWS Lake Formation, MSK.
- Business Application:** Alexa for Business, Amazon Chime, WorkMail, Amazon Honeycode.
- End User Computing:** WorkSpaces, AppStream 2.0, WorkDocs, WorkLink.
- Internet Of Things:** IoT Core, FreeRTOS, IoT 1-Click.
- Security, Identity, & Compliance:** IAM, Resource Access Manager, Cognito.

At the bottom right, the KODEKLoud logo is visible.

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

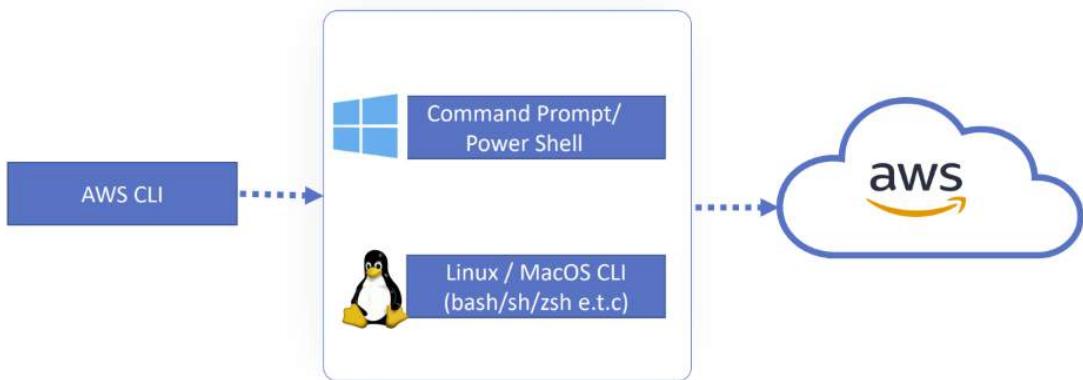
Create another policy read and list for s3 service ----- s3-readonly-policy

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

 \$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64-2.0.30.zip" -o "awscliv2.zip"
\$ unzip awscliv2.zip
\$ sudo ./aws/install

\$ aws --version
aws-cli/2.0.47 Python/3.7.4 Linux/4.14.133-113.105.amzn2.x86_64 botocore/2.0.0

 Download and Install <https://awscli.amazonaws.com/AWSCLIV2.msi>

C:\> aws --version
aws-cli/2.0.47 Python/3.7.4 Windows/10 botocore/2.0.0

 Download and Install
<https://awscli.amazonaws.com/AWSCLIV2.pkg>

\$ aws --version
aws-cli/2.0.47 Python/3.7.4 Darwin/18.7.0 botocore/2.0.0

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

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

```
$ 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"
    }
}
```

A screenshot of a terminal window showing the AWS CLI help output for the 'aws' command. The output is as follows:

```
$ aws help
AWS()

NAME
    aws -

DESCRIPTION
    The AWS Command Line Interface is a unified tool to
    manage your AWS
    services.

SYNOPSIS
    aws [options] <command> <subcommand> [parameters]
        Use aws command help for information on a specific
        command. Use aws
        help topics to view a list of available help topics. The
        synopsis for
        each command shows its parameters and their usage.
Optional parameters
    are shown in square brackets.
.
.
[Output Truncated]
```

The entire terminal window is highlighted with a blue oval.

KODEKLOUD

A screenshot of a terminal window showing the AWS CLI help output for the 'aws iam' command. The output is as follows:

```
$ aws iam help
IAM()

NAME
    iam -

DESCRIPTION
    AWS Identity and Access Management (IAM) is a web service
    for securely
    controlling access to AWS services. With IAM, you can
    centrally manage
    users, security credentials such as access keys, and
    permissions that
    control which AWS resources users and applications can
    access. For more
    information about IAM, see AWS Identity and Access
    Management (IAM) and
    the AWS Identity and Access Management User Guide .

AVAILABLE COMMANDS
    o add-client-id-to-open-id-connect-provider
    o add-role-to-instance-profile
.
.
[Output Truncated]
```

The entire terminal window is highlighted with a blue oval.

```
$ aws <command> help
```

KODEKLOUD

We can get help about sub command also

<https://docs.aws.amazon.com/cli/latest/reference>

```
$ aws iam create-user help

NAME
  create-user -

DESCRIPTION
  Creates a new IAM user for your AWS account.

  The number and size of IAM resources in an AWS account are
  limited. For
    more information, see IAM and STS Quotas in the IAM User
  Guide .

  See also: AWS API Documentation

  See 'aws help' for descriptions of global parameters.

SYNOPSIS
  create-user
  [--path <value>]
  --user-name <value>
  [--permissions-boundary <value>]
  [--tags <value>]
  [--cli-input-json <value>]
  [-generate-cli-skeleton <value>]
```

```
$ aws <command> <subcommand> help
```

KODEKLOUD

AWS IAM with Terraform

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

Filter

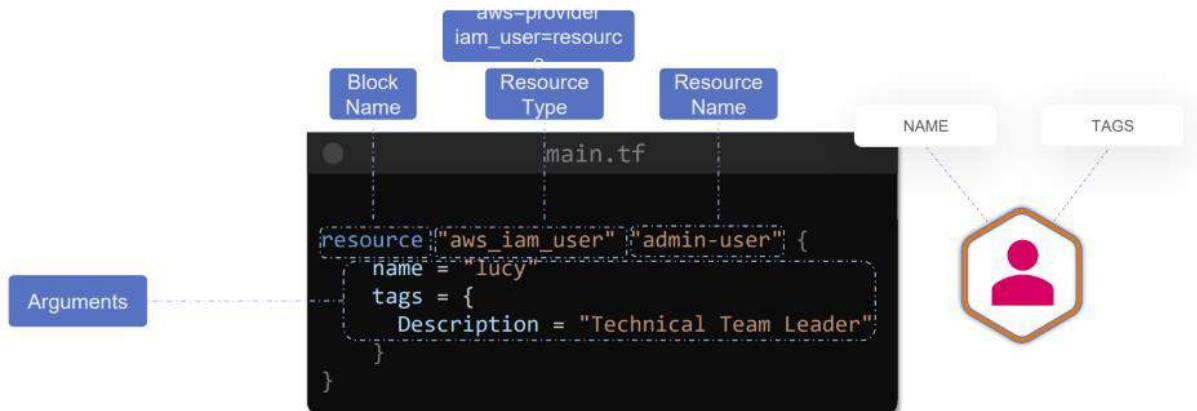
- > GuardDuty
- < IAM
- < Resources
- aws_iam_access_key
- aws_iam_account_alias
- aws_iam_account_password_policy
- aws_iam_group
- aws_iam_role
- aws_iam_role_policy
- aws_iam_role_policy_attachment
- aws_iam_saml_provider
- aws_iam_server_certificate
- aws_iam_service_linked_role
- aws_iam_user
- aws_iam_user_group_membership
- aws_iam_user_login_profile
- aws_iam_user_policy

Argument Reference

The following arguments are supported:

- `name` - (Required) The user's name. The name must consist of upper and lowercase alphanumeric characters with no spaces. You can also include any of the following characters: `-, ., @, _`. User names are not distinguished by case. For example, you cannot create users named both "TESTUSER" and "testuser".
- `path` - (Optional, default "/") Path in which to create the user.
- `permissions_boundary` - (Optional) The ARN of the policy that is used to set the permissions boundary for the user.
- `force_destroy` - (Optional, default false) When destroying this user, destroy even if it has non-Terraform-managed IAM access keys, login profile or MFA devices. Without `force_destroy` a user with non-Terraform-managed access keys and login profile will fail to be destroyed.
- `tags` - Key-value map of tags for the IAM user

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



KODEKLOUD

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



```
main.tf
```

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

```
$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v3.6.0...
- Installed hashicorp/aws v3.6.0 (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 = "~> 3.6.0"

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
```



```
main.tf
```

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

```
$ terraform plan
provider.aws.region
  The region where AWS operations will take place. Examples
  are us-east-1, us-west-2, etc.

  Enter a value: us-west-1
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.

Error: error configuring Terraform AWS Provider: no valid
credential sources for Terraform AWS Provider found.

Please see
https://registry.terraform.io/providers/hashicorp/aws
for more information about providing credentials.

Error: NoCredentialProviders: no valid providers in chain.
  Deprecated.
    For verbose messaging see
      aws.Config.CredentialsChainVerboseErrors
```

We can configure credential in provider:-

main.tf

```
provider "aws" {  
    region = "us-west-2"  
    access_key = "AKIAI44QH8DHBEEXAMPLE"  
    secret_key = "je7MtGbClwBF/2tk/h3yCo8n..."  
}  
resource "aws_iam_user" "admin-user" {  
    name = "lucy"  
    tags = {  
        Description = "Technical Team Leader"  
    }  
}
```

main.tf

```
provider "aws" {  
    region = "us-west-2"  
    access_key = "AKIAI44QH8DHBEEXAMPLE"  
    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
```

\$ 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 = "AKIAI44QH8DHBEEXAMPLE"
  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 = AKIAI44QH8DHBEEXAMPLE
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=AKIAI44QH8DHBEEXAMPLE
$ 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.



The image shows a terminal window with a dark background and light-colored text. The title bar of the window says "main.tf". The content of the file is a Terraform configuration for creating an IAM user named "lucy". The code includes a "tags" block with a "Description" key set to "Technical Team Leader".

```
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

The screenshot shows a Mac desktop environment. In the foreground, a terminal window titled "main.tf" displays Terraform configuration code:

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

Below the terminal are two icons: a user icon and a key icon labeled "IAM Policy".

In the background, a web browser window is open to the "AdministratorAccess Policy" page of the AWS IAM Argument Reference. The JSON policy document shown is:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}
```

A blue box highlights the "AdministratorAccess Policy" link at the bottom of the page.

The browser also displays the "Argument Reference" section, which includes a note about supported arguments and a bulleted list:

- `description` - (Optional, Forces new resource) Description of the IAM policy.
- `name` - (Optional, Forces new resource) The name of the policy. If omitted, Terraform will assign a random, unique name.
- `name_prefix` - (Optional, Forces new resource) Creates a unique name beginning with the specified prefix. Conflicts with `name`.
- `path` - (Optional, default "/") Path in which to create the policy. See [IAM Identifiers](#) for more information.
- `policy` - (Required) The policy document. This is a JSON formatted string. For more information about building AWS IAM policy documents with Terraform, see the [AWS IAM Policy Document Guide](#)

The screenshot shows a Mac desktop environment. In the foreground, a terminal window titled "main.tf" displays Terraform configuration code:

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

resource "aws_iam_policy" "adminUser" {
  name   = "AdminUsers"
  policy = ?
}
```

Below the terminal are two icons: a user icon and a key icon labeled "IAM Policy".

In the background, a web browser window is open to the "Heredoc Syntax" page of the AWS Documentation. The JSON policy document shown is identical to the one in the previous screenshot:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}
```

A blue box highlights the "Heredoc Syntax" link at the bottom of the page.

The browser also displays the "COMMAND" section, which includes a code block:

```
[COMMAND] <<DELIMITER
Line1
Line2
Line3
DELIMITER
```

How to attach policy document here. There are multiple ways

We can pass entire json file between the delimiter

The screenshot shows the KodeKloud Terraform interface. On the left, a code editor displays `main.tf` with the following content:

```

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

resource "aws_iam_policy" "adminUser" {
  name = "AdminUsers"
  policy = <<EOF
EOF
}

```

On the right, two boxes explain the code elements:

- AdministratorAccess Policy**: A JSON object representing a policy statement:


```
{
        "Version": "2012-10-17",
        "Statement": [
          {
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
          }
        ]
      }
```
- Heredoc Syntax**: A box showing the syntax for multi-line strings:


```
[COMMAND] <<DELIMITER
Line1
Line2
Line3
DELIMITER
```

Below the code editor are icons for a user and a key.

The bottom right corner features the KodeKloud logo.

The screenshot shows the same `main.tf` file with additional code added to attach the policy to the user:

```

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

resource "aws_iam_policy" "adminUser" {
  name = "AdminUsers"
  policy = <<EOF
EOF
}

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}

resource "aws_iam_user_policy_attachment" "lucy-admin-access" {
  user = aws_iam_user.admin-user.name
  policy_arn = aws_iam_policy.adminUser.arn
}

```

A blue oval highlights the policy attachment section. Below the code editor, a dashed arrow points from the user icon to the key icon, indicating the relationship between them.

The bottom right corner features the KodeKloud logo.

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

```
main.tf
```

```
resource "aws_iam_user" "admin-user" {
  name = "lucy"
  tags = {
    Description = "Technical Team Leader"
  }
}
resource "aws_iam_policy" "adminUser" {
  name   = "AdminUsers"
  policy = file("admin-policy.json")
}
EOF
}

resource "aws_iam_user_policy_attachment" "lucy-admin-access" {
  user = aws_iam_user.admin-user.name
  policy_arn = aws_iam_policy.adminUser.arn
}
```

```
admin-policy.json
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    }
  ]
}
```

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.

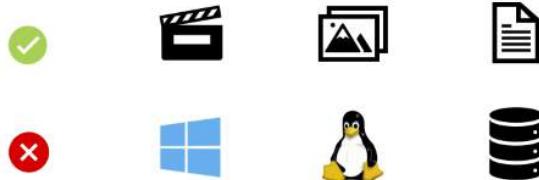
Everything within a bucket is an object

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.

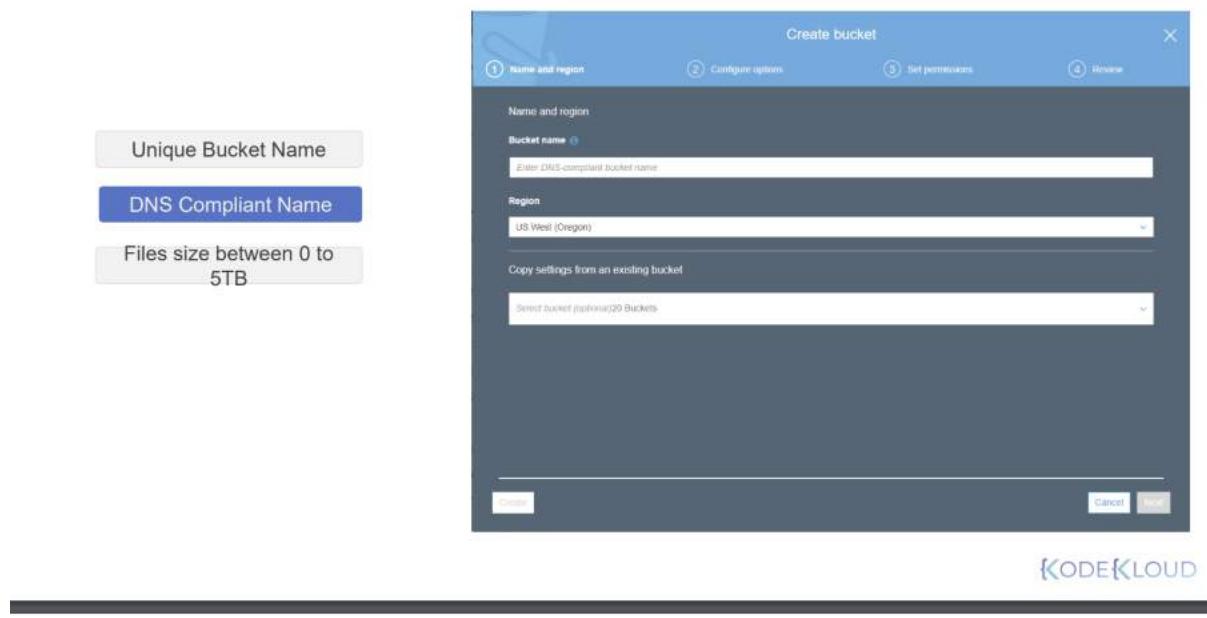


KODEKLOUD



KODEKLOUD

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



KODEKLOUD

https://<bucket_name>.us-west-1.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

KODEKLOUD

Any object in S3 consists of **object data and metadata**

Metadata consists of owner, size of object

Note :

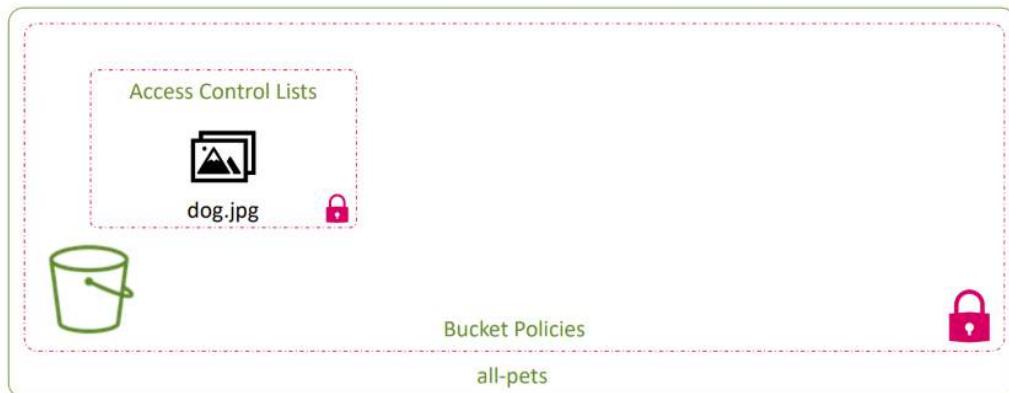
When data is uploaded into bucket it has least privilege means no one can access bucket except owner.

Access to bucket and its objects are governed by bucket policies and access controller lists.

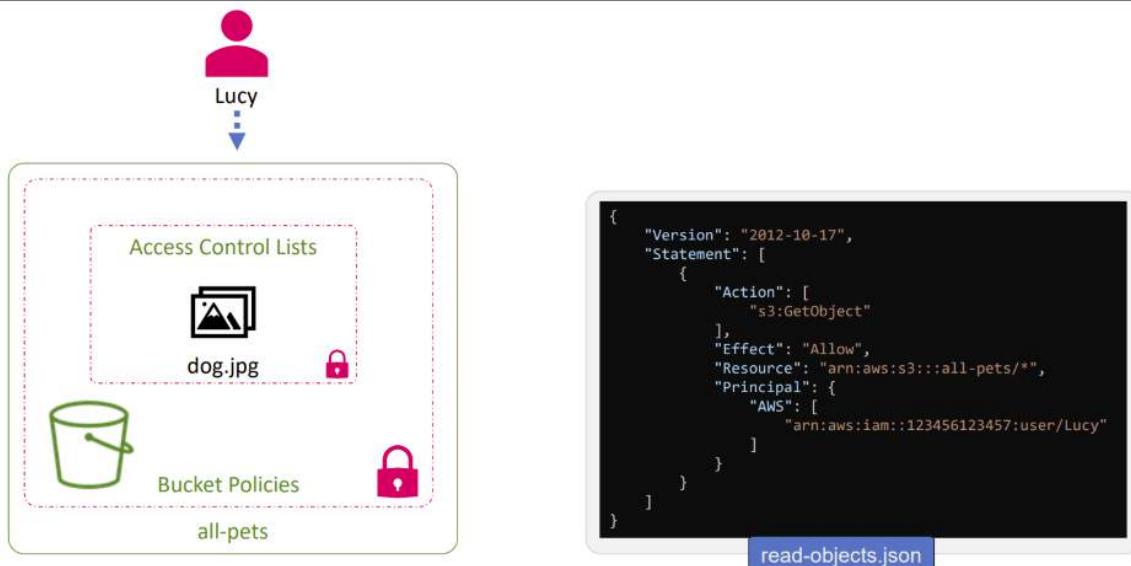
Bucket policies are permissions that are granted at bucket level

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"
}

```

AWS

finance-21092020 finance-2020.doc finance-analysts

terraformer, tfstate

finance-data

KODEKLOUD

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": "s3",
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::<<bucket-name>>/*",
      "Principal": {
        "AWS": [
          "<< arn >>"
        ]
      }
    }
  ]
}

```

EOF

EOF

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

aws

KODEKLOUD

read-objects.json

```

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.a[*]}"
        ]
      }
    }
  ]
}
EOF
}

```



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.



DynamoDB

Highly Scalable

Fully Managed by AWS

NoSQL Database

Single-Digit Millisecond Latency

Data Replicated across Regions

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

The left window shows the Terraform configuration file `main.tf` with the following code:

```

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": "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"
}
EOF
}

```

The right window shows the AWS Lambda function configuration with the following JSON data:

```

{
  "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"
}

```

The left window shows the Terraform configuration file `main.tf` with the following code:

```

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
}

```

The right window shows the terminal output of `terraform apply`:

```

$ 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

Advantages of `terraform.tfState`

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

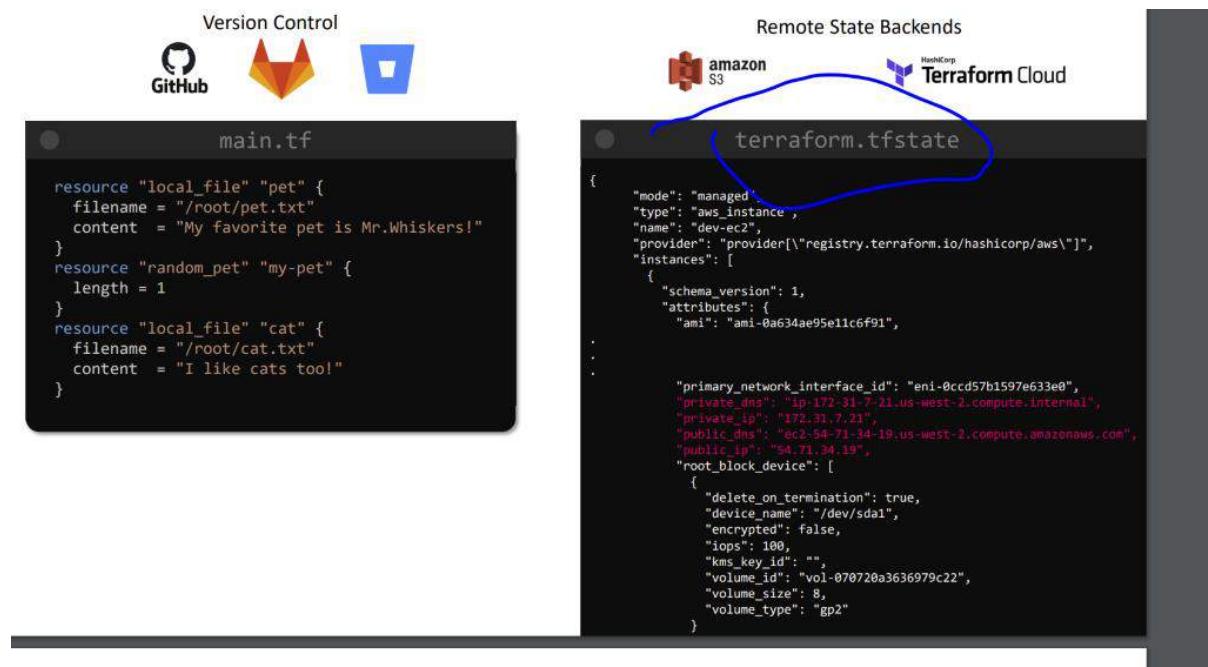
Improving **performance** of terraform operation while working with large terraform file

It allows to **collaborate** team

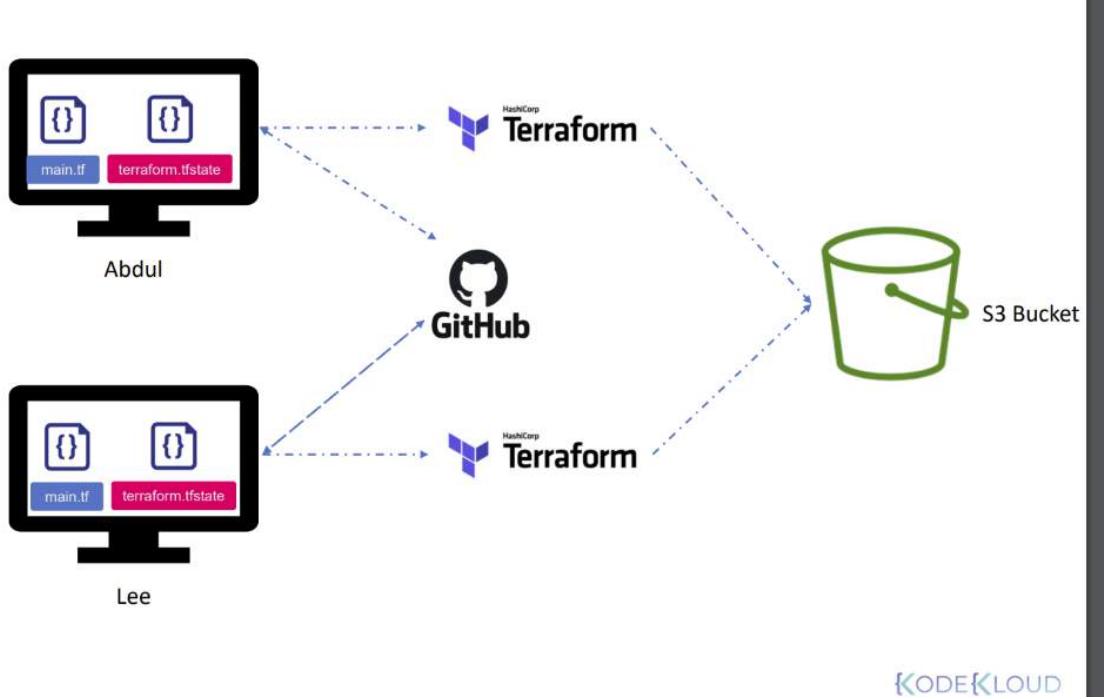
Tf.state file is created locally



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

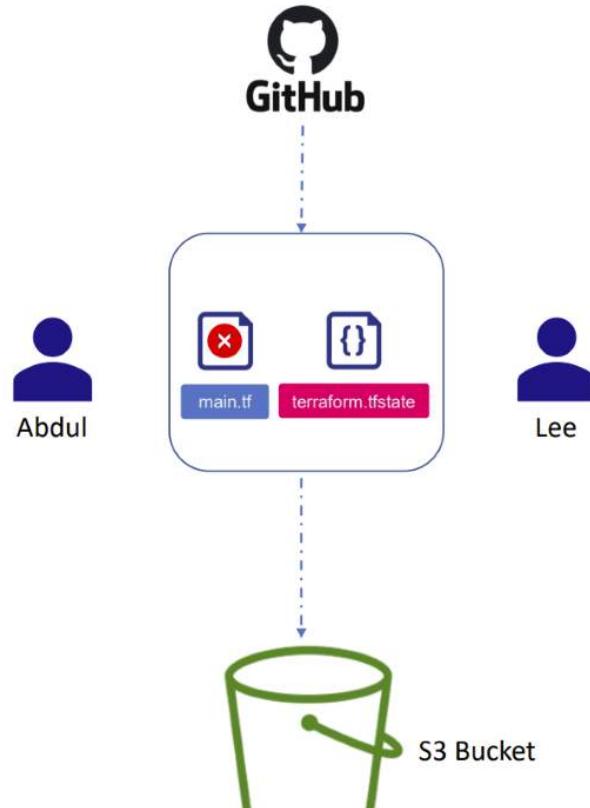


It is **not good** to store the state file in **version control system** since it store **sensitive data**



KODEKLOUD

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.



```

>_ Terminal 1
$ terraform apply
.
.
+ server_side_encryption = (known after apply)
+ storage_class           = (known after apply)
+ version_id               = (known after apply)
}

Plan: 2 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.finance: Creating...
aws_s3_bucket_object.finance-2020: Still creating...
[10s elapsed]
aws_s3_bucket.finance: Still creating... [10s elapsed]
aws_s3_bucket_object.finance-2020: Still creating...
[20s elapsed]
aws_s3_bucket.finance: Still creating... [20s elapsed]

```



```

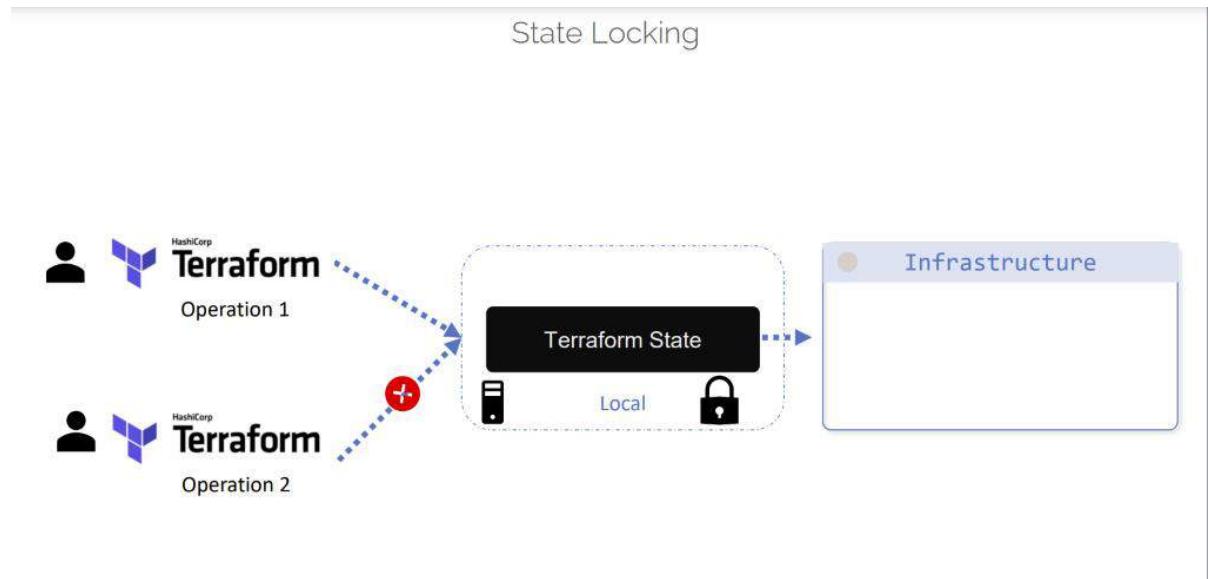
>_ Terminal 2
$ terraform apply
Error: Error locking state: Error acquiring the state
lock: resource temporarily unavailable
Lock Info:
  ID:      fefe3806-007c-084b-be61-cef4cdc77dee
  Path:    terraform.tfstate
  Operation: OperationTypeApply
  Who:     root@iac-server
  Version: 0.13.3
  Created: 2020-09-22 20:35:27.051330492 +0000 UTC
  Info:

Terraform acquires a state lock to protect the state from
being written
by multiple users at the same time. Please resolve the
issue above and try
again. For most commands, you can disable locking with
the "-lock=false"
flag, but this is not recommended.

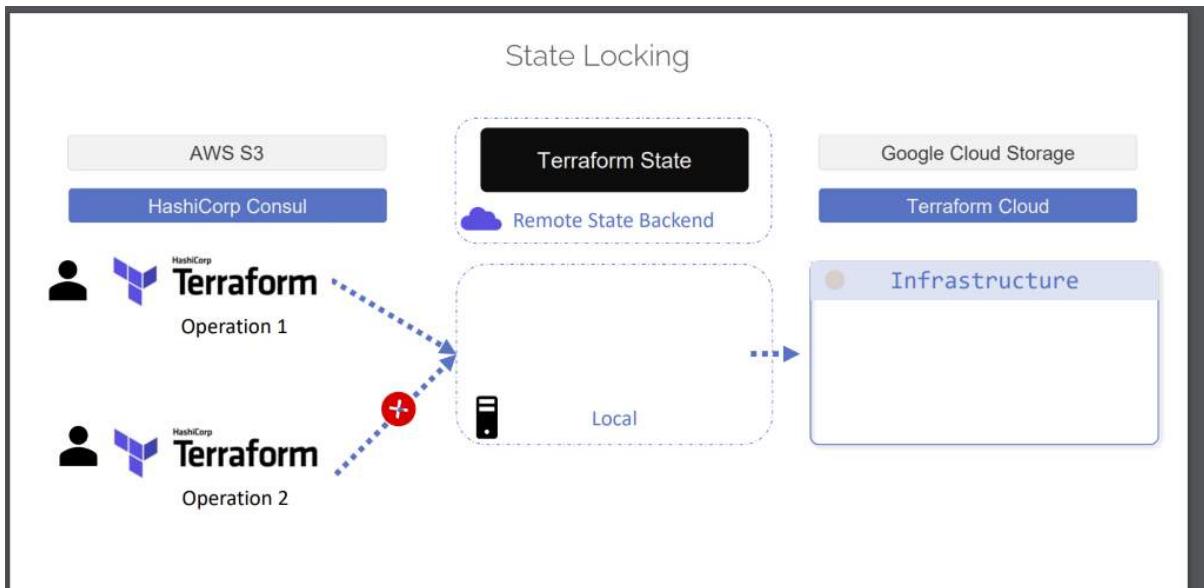
```

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

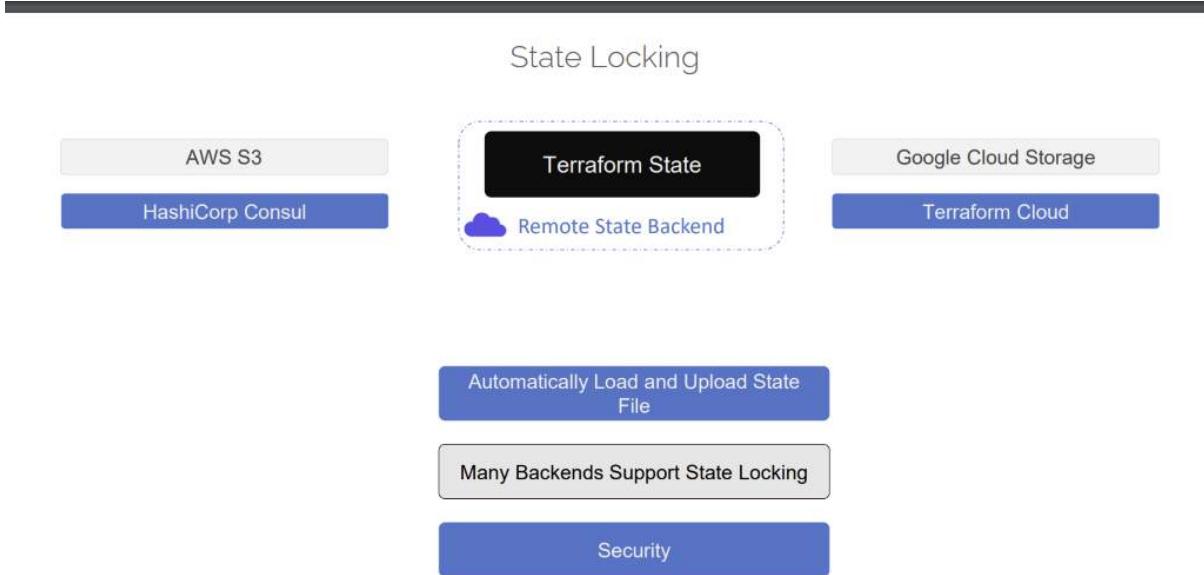
Imp : version control system such as github do not support the state locking and as a result Multiple user can use same state file simultaneously which can result data loss and conflicts in state file.



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



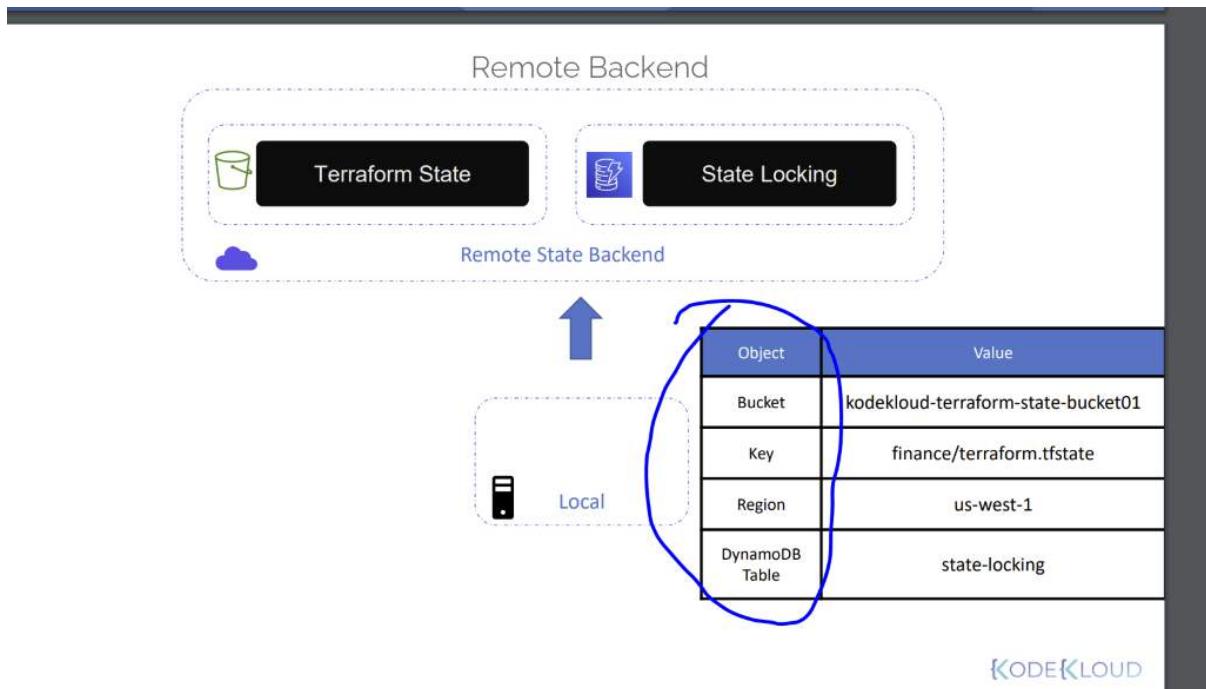
Remote backends with s3

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

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

terraform {
  backend "s3" {
    bucket      = "kodekloud-terraform-state-bucket01"
    key         = "finance/terraform.tfstate"
    region     = "us-west-1"
    dynamodb_table = "state-locking"
  }
}
```

>_

```
$ ls
main.tf  terraform.tfstate
```

Object	Value
Bucket	kodekloud-terraform-state-bucket01
Key	finance/terraform.tfstate
Region	us-west-1
DynamoDB Table	state-locking

```

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

terraform.tf
terraform {
  backend "s3" {
    bucket      = "kodekloud-terraform-state-bucket01"
    key         = "finance/terraform.tfstate"
    region     = "us-west-1"
    dynamodb_table = "state-locking"
  }
}

```

```

>_
$ terraform apply
Backend reinitialization required. Please run "terraform init". Reason: Initial configuration of the requested backend "s3"
The "backend" is the interface that Terraform uses to store state, perform operations, etc. If this message is showing up, it means that the Terraform configuration you're using is using a custom configuration for the Terraform backend.
Changes to backend configurations require reinitialization. This allows Terraform to setup the new configuration, copy existing state, etc. This is only done during "terraform init". Please run that command now then try again.
Error: Initialization required. Please see the error message above.

```

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

```

>_
$ terraform init
Initializing the backend...
Do you want to copy existing state to the new backend?
Pre-existing state was found while migrating the previous "local" backend to the newly configured "s3" backend. No existing state was found in the newly configured "s3" backend. Do you want to copy this state to the new "s3" backend? Enter "yes" to copy and "no" to start with an empty state.

Enter a value: yes
Successfully configured the backend "s3"! Terraform will automatically use this backend unless the backend configuration changes.

Initializing provider plugins...
- Using previously-installed hashicorp/aws v3.7.0
.
.[Output Truncated]

>_
$ rm -rf .terraform

```

KODEKLOUD

We can delete the local state file configuration

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

```
>_

$ terraform apply
Acquiring state lock. This may take a few moments...
aws_s3_bucket.terraform-state: Refreshing state... [id=kodekloud-terraform-state-bucket01]
aws_dynamodb_table.state-locking: Refreshing state... [id=state-locking]

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
Releasing state lock. This may take a few moments.
```

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

The terminal window shows the following commands:

```
>_
$ vi terraform.tfstate
$ terraform state show aws_s3_bucket.finance
# terraform state <subcommand> [options] [args]
```

A modal window titled "terraform.tfstate" displays the JSON content of the state file:

```
{
  "mode": "managed",
  "type": "aws_instance",
  "name": "dev-ec2",
  "provider": "provider[\\"registry.terraform.io/hashicorp/aws\\"]",
  "instances": [
    {
      "schema_version": 1,
      "attributes": {
        "ami": "ami-0a634ae95e11c6f91",
        "primary_network_interface_id": "eni-0ccd57b1597e633e0",
        "private_dns": "ip-172-31-7-21.us-west-2.compute.internal",
        "private_ip": "172.31.7.21",
        "public_dns": "ec2-54-71-34-19.us-west-2.compute.amazonaws.com",
        "public_ip": "54.71.34.19",
        "root_block_device": [
          {
            "delete_on_termination": true,
            "device_name": "/dev/sda1",
            "encrypted": false,
            "iops": 100,
            "kms_key_id": "",
            "volume_id": "vol-070720a3636979c22",
            "volume_size": 8,
            "volume_type": "gp2"
          }
        ],
        "volumes": [
          {
            "id": "vol-070720a3636979c22",
            "size": 8
          }
        ]
      }
    }
  ]
}
```

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

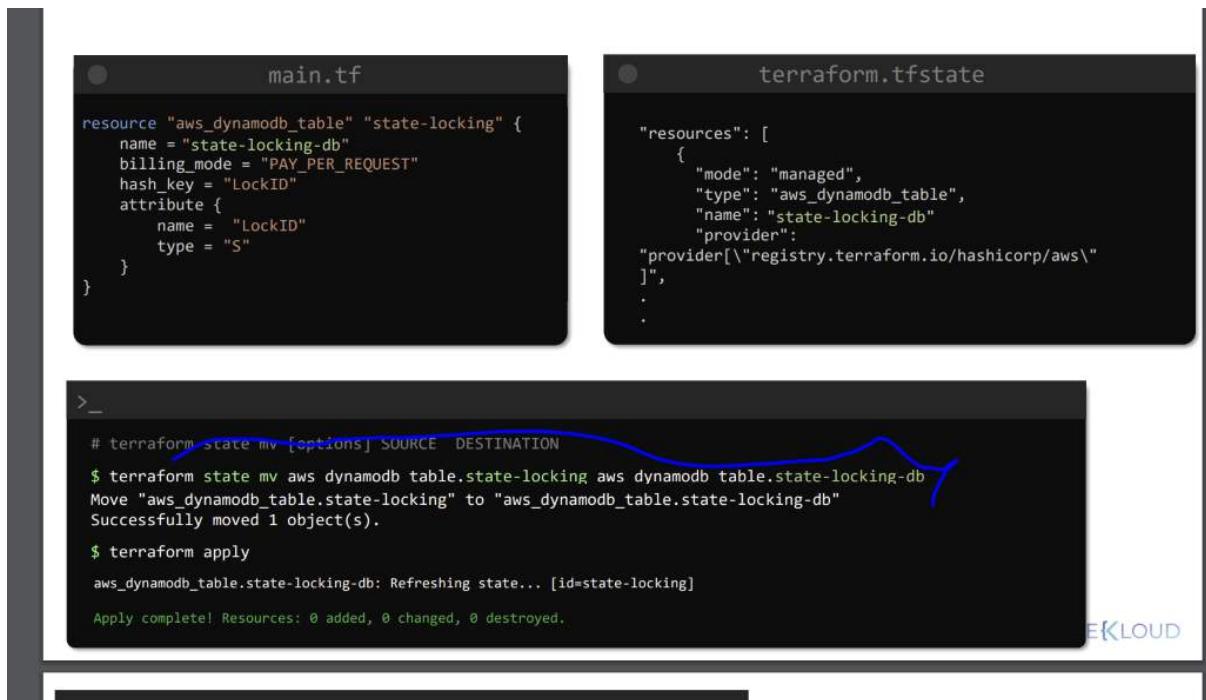
```
>_  
  
# terraform state list [options] [address]  
  
$ terraform state list  
aws_dynamodb_table.cars  
aws_s3_bucket.finance-2020922  
  
$ terraform state list aws_s3_bucket.finance-2020922  
aws_s3_bucket.finance-2020922
```

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"  
    bucketRegionalDomainName = " finance-2020922.s3.us-west-1.amazonaws.com"  
    force_destroy = false  
    hostedZoneId = "Z2F5ABCDE1ACD"  
    id           = "finance-2020922 "  
    region       = "us-west-1"  
    requestPayer = "BucketOwner"  
    tags          = {  
        "Descriptipon" = "Bucket to store Finance and Payroll Information"  
    }  
  
    versioning {  
        enabled    = false  
        mfaDelete  = false  
    }  
}
```

Terraform mv command



The terminal shows the following commands:

```

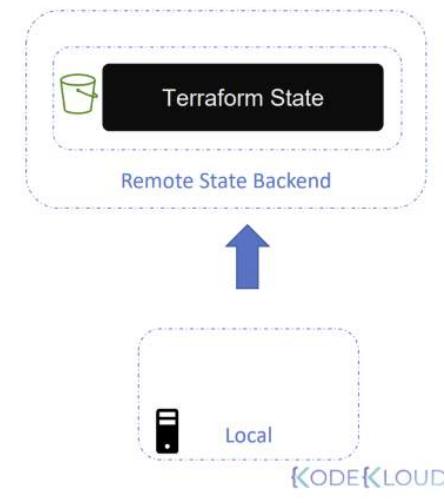
>_
# terraform state mv [options] SOURCE DESTINATION
$ terraform state mv aws dynamodb table.state-locking aws dynamodb table.state-locking-db
Move "aws_dynamodb_table.state-locking" to "aws_dynamodb_table.state-locking-db"
Successfully moved 1 object(s).

$ terraform apply
aws_dynamodb_table.state-locking-db: Refreshing state... [id=state-locking]
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

```

KODEKLOUD

Terraform state pull command to view state file from remote



The diagram illustrates the flow of Terraform State:

- Local:** Represented by a smartphone icon.
- Remote State Backend:** Represented by a bucket icon.
- Terraform State:** Represented by a black rectangular box labeled "Terraform State".

A blue arrow points upwards from the Local box to the Remote State Backend box, and another blue arrow points upwards from the Remote State Backend box to the Terraform State box.

```

>_
$ ls
main.tf provider.tf

# terraform state pull [options] SOURCE DESTINATION
$ terraform state pull

{
  "version": 4,
  "terraform_version": "0.13.0",
  "serial": 0,
  "lineage": "b6e2cf0e-ef8d-3c59-1e11-c6520dcd745c",
  "resources": [
    {
      "mode": "managed",
      "type": "aws_dynamodb_table",
      "name": "state-locking-db",
      "provider": "provider[\"registry.terraform.io/hashicorp/aws\"]",
      "instances": [
        {
          "schema_version": 1,
          "attributes": {
            ...
          }
        }
      ]
    }
  ]
}

$ terraform state pull | jq '.resources[] | select(.name == "state-locking-db")|.instances[].attributes.hash_key'
"LockID"

```

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...
```

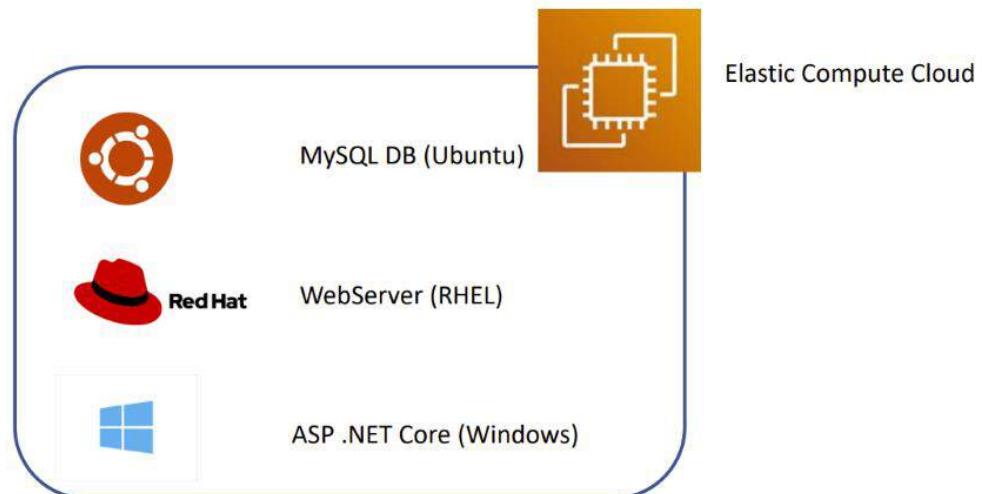
It is only removed from terraform management

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



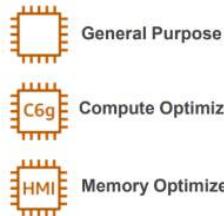
Amazon Machine Image (AMI's)

 Amazon Linux 2 AMI ami-0c2f25c1f66a1ff4d

 Red Hat Enterprise Linux 8 ami-04312317b9c8c4b51

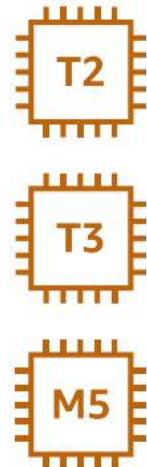
 Ubuntu Server 20.04 LTS ami-0edab43b6fa892279

Instance Types



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

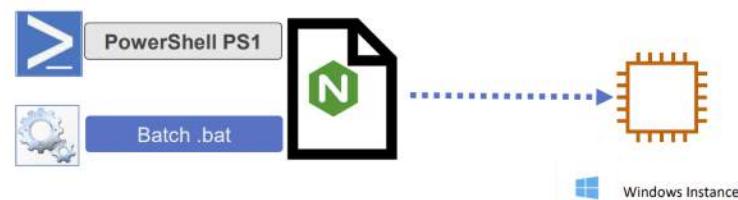
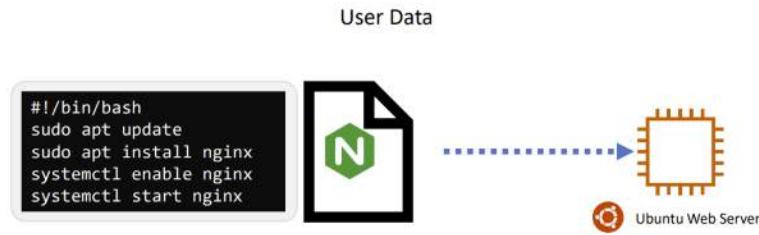


EBS : elastic block storage

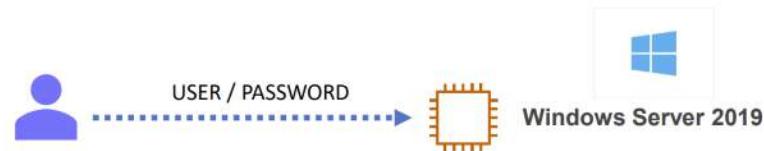
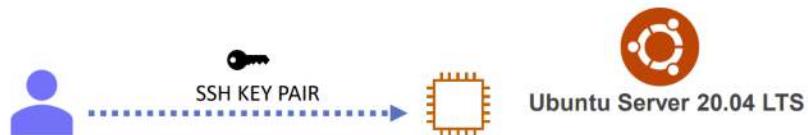


EBS Volume

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



KODEKLOU



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"
  }
}
```

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 trigger a stop/start of the EC2 instance.
- `tags` - (Optional) A map of tags to assign to the resource.

```
provider.tf

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

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
```

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 trigger 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 not 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 AAAAB3NzaC1yc2EAAAQABAAQDicpU+kT9isaZy7cHYa
+oCTUo1S6Tg6vCEq+ufucIMrA7RLTng1+YfTfvgrY2UiHGxuuJ11E
yT0x2UrGexVx4G2TzX/am2WFzNbcGSg2bCXTkVQY93KOhbW9y851a
+g1wITODC0oxEMFr/CVsrgJ4bfbp8S896VKBxC1WpSU9GscPP28GV
uDgm2ATBuL78AF root@iac-server"
}
```

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 "icmpv6")
- `from_port` - (Required) The start port (or ICMP type number if protocol is "icmpv6")
- `protocol` - (Required) The protocol. If you select a protocol of "-1" (semantically 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

Provisioners

main.tf

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

resource "aws_security_group" "ssh-access" {
    << code hidden >>
}
```

Advanced Details

Metadata accessible	<input checked="" type="checkbox"/> Enabled
Metadata version	V1 and V2 (token optional)
Metadata token response hop limit	1
User data	<input checked="" type="radio"/> As text <input type="radio"/> As file <input type="checkbox"/> Input is already base64 encoded
#!/bin/bash sudo apt update sudo apt install nginx -y systemctl enable nginx systemctl start nginx	

KODEKLOUD

Remote Exec

main.tf

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

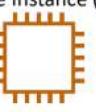
resource "aws_security_group" "ssh-access" {
    << code hidden >>
}
```


SSH


WINRM


Terraform


Local Machine


Remote Instance (EC2)

```
apt update
apt install nginx -y
systemctl enable nginx
systemctl start nginx
```

✓ Network Connectivity (Security Group)
✓ Authentication (SSH Key Pair)

KODEKLOUD

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

Remote Exec

```
main.tf
>_
$ terraform apply
aws_key_pair.web: Creating...
aws_security_group.ssh-access: Creating...
aws_key_pair.web: Creation complete after 0s [id=terraform-20201015013048569100000001]
aws_security_group.ssh-access: Creation complete after 1s [id=sg-e
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 'remote-exec'...
aws_instance.webserver (remote-exec): Connecting to remote host vi
aws_instance.webserver (remote-exec): Host: 3.96.136.157
aws_instance.webserver (remote-exec): User: ubuntu
aws_instance.webserver (remote-exec): Password: false
aws_instance.webserver (remote-exec): Private key: true
aws_instance.webserver (remote-exec): Certificate: false
aws_instance.webserver (remote-exec): SSH Agent: false
aws_instance.webserver (remote-exec): Checking Host Key: false
aws_instance.webserver: Still creating... [40s elapsed]
aws_instance.webserver (remote-exec): Connecting to remote host vi
aws_instance.webserver (remote-exec): Host: 3.96.136.157
aws_instance.webserver (remote-exec): User: ubuntu
aws_instance.webserver (remote-exec): Password: false
aws_instance.webserver (remote-exec): Private key: true
aws_instance.webserver (remote-exec): Certificate: false
aws_instance.webserver (remote-exec): SSH Agent: false
aws_instance.webserver (remote-exec): Checking Host Key: false
aws_instance.webserver (remote-exec): Connected!
aws_instance.webserver: Still creating... [50s elapsed]
aws_instance.webserver: Creation complete after 50s [id=i-068fad3e]
```

Local Exec

Local exec

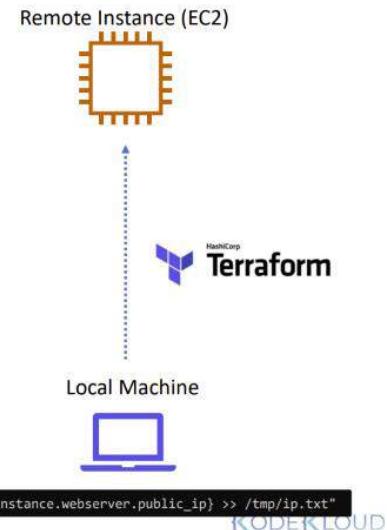
Local Exec

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

  provisioner "local-exec" {
    command = "echo ${aws_instance.webserver2.public_ip} >> /tmp/ips.txt"
  }
}
```

```
>_
$ cat /tmp/ips.txt
54.214.68.27
```

- **command** - (Required) This is the command to execute, it can be provided as a relative path to the current working directory or as an absolute path. It is evaluated in a shell, and can use environment variables or Terraform variables.



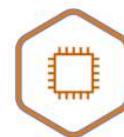
KODEKLOUD

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

Creation Time Provisioner

```
main.tf
resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"
  provisioner "local-exec" {
    command = "echo Instance ${aws_instance.webserver.public_ip} Created! > /tmp/instance_state.txt"
  }
}
```

```
>_
$ cat /tmp/instance_state.txt
Instance 3.96.136.157 Created!
```



KODEKLOUD

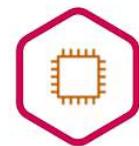
We can also run provisioner before the resource is destroyed

Destroy Time Provisioner

```
main.tf
```

```
resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"
  provisioner "local-exec" {
    command = "echo Instance ${aws_instance.webserver.public_ip} Created! > /tmp/instance_state.txt"
  }
  provisioner "local-exec" {
    when      = destroy
    command = "echo Instance ${aws_instance.webserver.public_ip} Destroyed! > /tmp/instance_state.txt"
  }
}
```

```
>_
$ cat /tmp/instance_state.txt
Instance 3.96.136.157 Deleted!
```



KODEKL

If the command within the provisioner fails the terraform apply command will also show error

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

OnFailure= continue

Failure Behavior

```
main.tf

resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"
  provisioner "local-exec" {
    on_failure = fail
    command = "echo Instance ${aws_instance.webserver.public_ip} Created! > /tmp/instance_state.txt"
  }
  provisioner "local-exec" {
    when      = destroy
    command = "echo Instance ${aws_instance.webserver.public_ip} Destroyed! > /tmp/instance_state.txt"
  }
}
```

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

KODEKLOUD

Failure Behavior

```
main.tf

resource "aws_instance" "webserver" {
  ami           = "ami-0edab43b6fa892279"
  instance_type = "t2.micro"
  provisioner "local-exec" {
    on_failure = continue
    command = "echo Instance ${aws_instance.webserver.public_ip} Created! > /tmp/instance_state.txt"
  }
  provisioner "local-exec" {
    when      = destroy
    command = "echo Instance ${aws_instance.webserver.public_ip} Destroyed! > /tmp/instance_state.txt"
  }
}
```

```
>_
$ terraform apply
aws_instance.webserver (local-exec) The system cannot find the path specified.
aws_instance.project: Creation complete after 22s [id=i-01585c2b9dbc445db]

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

KODEKLOUD

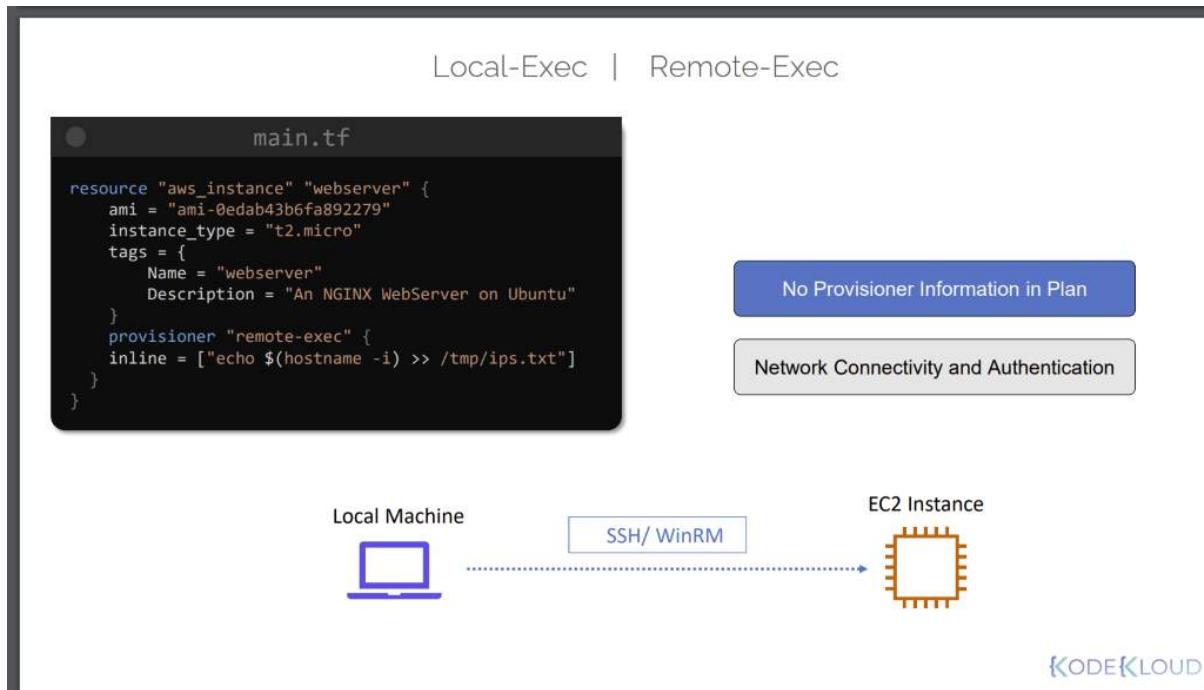
Provisioner behavior

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



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 are 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.
```

KODEKLO

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" {
```

KODEKLO

Taint

```
$ terraform taint aws_instance.webserver
Resource instance aws_instance.webserver has been marked as tainted.

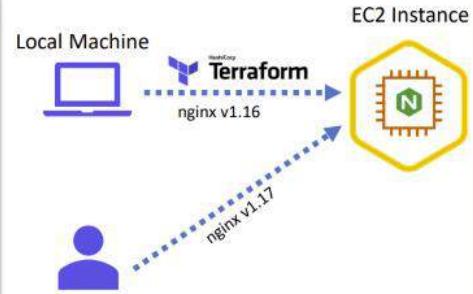
$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

aws_instance.webserver: Refreshing state... [id=i-0fd3946f5b3ab8af8]

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" {
```



KODEKLOUD

Taint

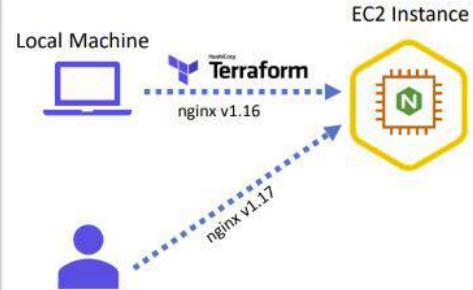
```
$ terraform untaint aws_instance.webserver
Resource instance aws_instance.webserver has been successfully
untainted.

$ 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-0fd3946f5b3ab8af8]

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.
```



Debugging

How to enable and debugging in terraform

Log Levels

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



Terraform log

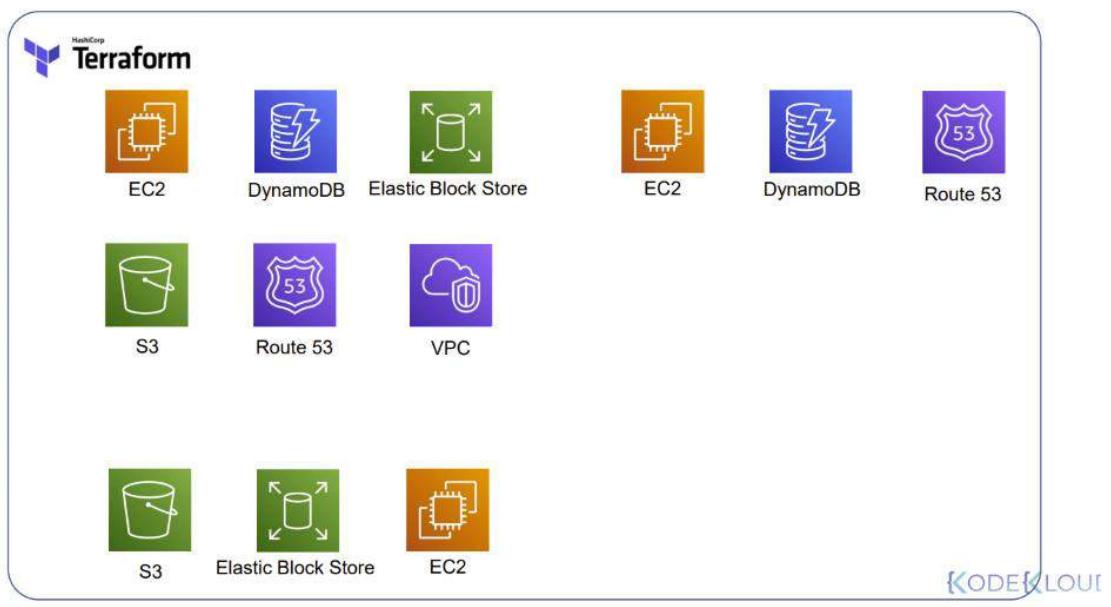
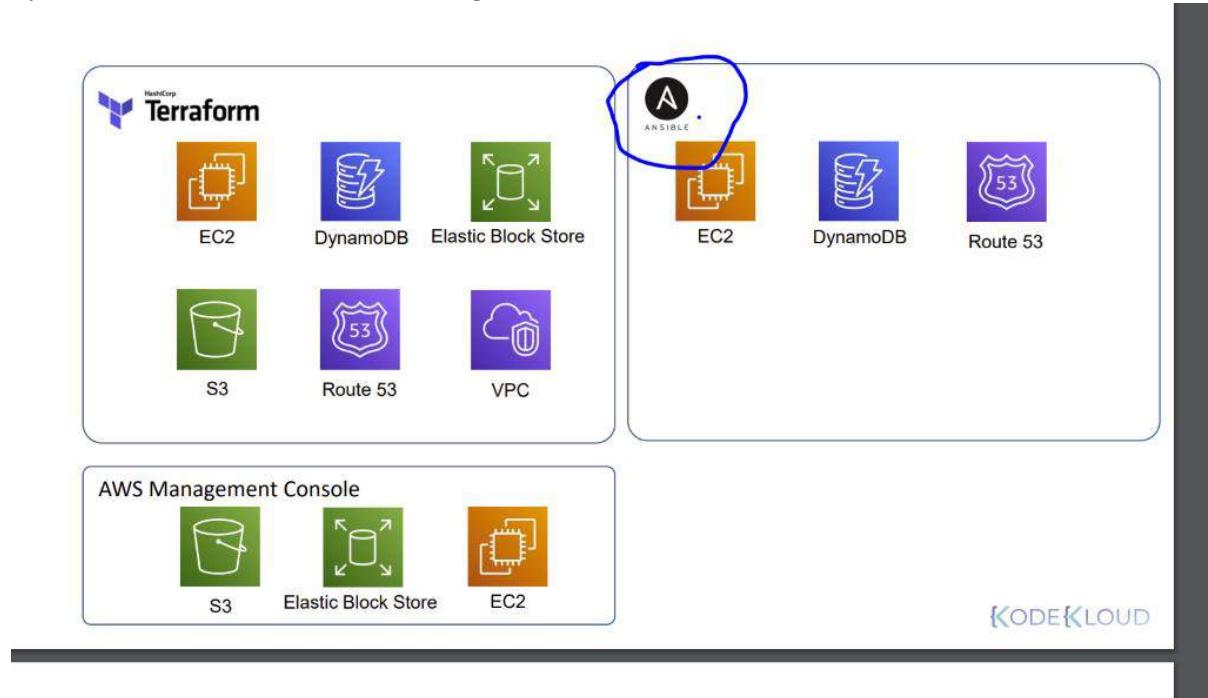
```
$ export TF_LOG_PATH=/tmp/terraform.log
$ head -10 /tmp/terraform.logs
---
2020/10/18 22:08:30 [INFO] Terraform version: 0.13.0
2020/10/18 22:08:30 [INFO] Go runtime version: go1.14.2
2020/10/18 22:08:30 [INFO] CLI args: []string{"C:\\Windows\\system32\\terraform.exe",
"plan"}
2020/10/18 22:08:30 [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 are created by other resource like ansible but what if the aws resource is created by other resource and we want to bring it 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)
}
```

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

```

main.tf

resource "aws_instance" "weberver" {
    # configuration here
}
resource "aws_key_pair" "key" {
    # configuration here
}
resource "aws_security_group" "ssh-access" {
    # configuration here
}
resource "aws_s3_bucket" "data-bucket" {
    # configuration here
}
resource "aws_dynamodb_table" "user-data" {
    # configuration here
}
resource "aws_instance" "web-server-2" {
    # configuration here
}
```

	aws_instance
	aws_key_pair
	aws_iam_policy
	aws_s3_bucket
	aws_dynamodb_table
	aws_instance

NodeCloud

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

```
main.tf
resource "aws_instance" "webserver" {
  # configuration here
}

key_pair.tf
resource "aws_key_pair" "web" {
  # configuration here
}

dynamodb_table.tf
resource "aws_dynamodb_table" "state-locking" {
  # configuration here
}

security_group.tf
resource "aws_security_group" "ssh-access" {
  # configuration here
}

ec2_instance.tf
resource "aws_instance" "webserver-2" {
  # configuration here
}

s3_bucket.tf
resource "aws_s3_bucket" "terraform-state" {
  # configuration here
}
```

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

```
>_
$ ls
provider.tf
id_rsa
id_rsa.pub
main.tf
pub_ip.txt
terraform.tfstate.backup
terraform.tfstate
iam_roles.tf
iam_users.tf
security_groups.tf
variables.tf
outputs.tf
s3_buckets.tf
dynamo_db.tf
local.tf
```

Complex Configuration Files

Duplicate Code

Increased Risk

Limits Reusability

KODEKLOUD

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
```



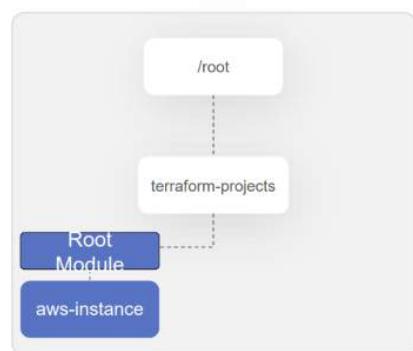
```
main.tf
```

```
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"
}
```



KODEKLOL

Root Module

```
>_
$ mkdir /root/terraform-projects/development
main.tf
```

```
main.tf
```

```
module "dev-webserver" {
  source = "../aws-instance"
}
```

KODEKLOUD

The diagram illustrates the directory structure of a Terraform project. At the top level is a folder named 'terraform-projects'. Inside it, there are two subfolders: 'Root Module' and 'development'. The 'Root Module' folder contains a file named 'main.tf'. Below the 'Root Module' is a 'Child Module' folder, which also contains a 'main.tf' file. This child module defines a single module named 'dev-webserver' that sources from another module located in the 'aws-instance' folder. A blue circle highlights the 'aws-instance' folder, indicating its role as a child module.

```
>_
$ mkdir /root/terraform-projects/development
main.tf
```

```
main.tf
```

```
module "dev-webserver" {
  source = "../aws-instance"
}
```

The diagram illustrates the directory structure of a Terraform project. At the top level is a folder named 'terraform-projects'. Inside it, there are two subfolders: 'Root Module' and 'development'. The 'Root Module' folder contains a file named 'main.tf'. Below the 'Root Module' is a 'Child Module' folder, which also contains a 'main.tf' file. This child module defines a single module named 'dev-webserver' that sources from another module located in the 'aws-instance' folder. A blue circle highlights the 'aws-instance' folder, indicating its role as a child module.

Creating and using a module

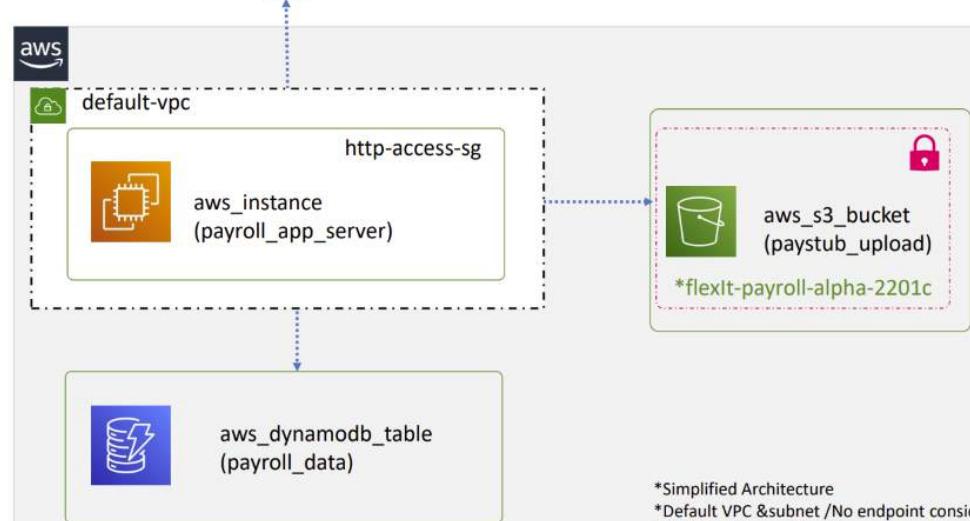
Requirement : to redeploy the same stack in different countries using terraform module

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



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}"  
}
```

```
/root
```

```
terraform-projects
```

```
modules
```

```
payroll-ap
```

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  
}
```

```
/root
```

```
terraform-projects
```

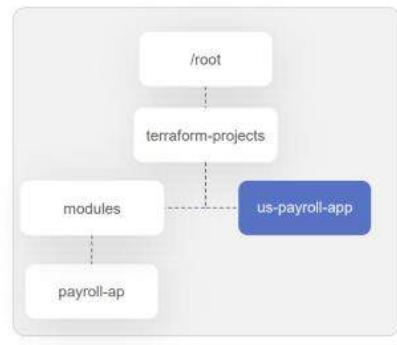
```
modules
```

```
payroll-ap
```

```
>_
$ 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... LOUD
```

```
>_
$ 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 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"
.
.
```



[Simpler Configuration Files](#)

[Lower Risk](#)

[Re-Usability](#)

[Standardized Configuration](#)

KODEKLOU

Same module can be used in different regions 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 used 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

Local Module



```
main.tf
module "dev-webserver" {
  source = "../aws-instance/"
  key    = "webserver"
}
```

Provider dropdown:

- alicloud
- aws
- azurerm
- google
- oci

Modules

Modules are self-contained packages of Terraform configurations that a

Module	Description	Last Published	Size
terraform-aws-modules / vpc	Terraform module which creates VPC resources on AWS	2 hours ago	5.5M
terraform-aws-modules / security-group	Terraform module which creates EC2-VPC security groups on AWS	2 months ago	5.4M
terraform-aws-modules / eks	Terraform module to create an Elastic Kubernetes (EKS) cluster and associa	11 days ago	2.0M

<https://registry.terraform.io/browse/modules>

Terraform Registry



Search bar: security-group

- [terraform-aws-modules/security-group](#)
Terraform module which creates EC2-VPC security groups on AWS
- [dcos-terraform/security-groups](#)
Create DC/OS related security groups
- [Azure/network-security-group](#)
Terraform module to create a network security group and assign it to the specified subnet
- [devops-workflow/security-group](#)
Terraform module which creates EC2-VPC security groups on AWS
- [claranet/nsg](#)
Terraform module for Azure Network Security Group



aws security-group

AWS

Terraform module which creates EC2-VPC security groups on AWS

Published August 20, 2020 by [terraform-aws-modules](#)
Module managed by [antsiblanks](#)
Total provisions: 5.4M
Source Code: [github.com/terraform-aws-modules/terraform-aws-security-group](#) (report an iss

Submodules Examples

The screenshot shows the AWS CloudFormation console page for the "security-group" module. At the top, there's a large "aws" logo. Below it, the module name "security-group" is displayed with a blue square icon. To the right, a dropdown menu shows "Version 3.16.0 (latest)". The main content area contains a brief description: "Terraform module which creates EC2-VPC security groups on AWS". Below this, there's a list of module details: "Published August 20, 2020 by terraform-aws-modules", "Module managed by antonbabenko", "Total provisions: 54M", and "Source Code: [github.com/terraform-aws-modules/terraform-aws-security-group](#) (report an issue)". At the bottom of this section are two buttons: "Submodules" and "Examples". On the right side, there's a "Provision Instructions" box with the following Terraform code:

```
module "security-group" {  
  source = "terraform-aws-modules/security-group/  
  version = "3.16.0"  
  # insert the 2 required variables here  
}
```

KODEKLOUD

Terraform Module

The screenshot shows the Kodekloud Terraform Module interface for the "security-group" module. It has a similar layout to the AWS CloudFormation console. At the top left is the "aws" logo, followed by the module name "security-group" with a blue square icon. A dropdown menu shows "Version 3.16.0 (latest)". The main content area includes the same module details and a "Provision Instructions" box with the same Terraform code as the AWS screenshot.

On the left side, there's a sidebar with a tree view of other modules available in this category. The tree structure starts with "active-mq", followed by "alertmanager", "carbon-relay-ng", "cassandra", "consul", "docker-swarm", "elasticsearch", "grafana", "graphite-statsd", and then three collapsed items under "http": "http-80", "http-8080", and "https-443".

At the bottom right of the interface, the "KODEKLOUD" logo is visible.

main.tf

```

module "security-group_ssh" {
  source  = "terraform-aws-modules/security-group/aws/modules/ssh"
  version = "3.16.0"
  # insert the 2 required variables here
  vpc_id = "vpc-7d8d215"
  ingress_cidr_blocks = [ "10.10.0.0/16"]
  name = "ssh-access"
}

```

Provision Instructions
Copy and paste into your Terraform configuration, insert the variables, and run `terraform init`:

```

module "security-group" {
  source  = "terraform-aws-modules/security-group/aws"
  version = "3.16.0"
  # insert the 2 required variables here
}

```

>_ \$ terraform get
Downloading terraform-aws-modules/security-group/aws 3.16.0 for security-group_ssh...
- security-group_ssh in .terraform\modules\security-group\ssh\modules\ssh

<https://registry.terraform.io/modules/terraform-aws-modules/security-group/aws/latest/submodules/ssh>

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**

Functions

main.tf

```

resource "aws_iam_policy" "adminUser" {
  name   = "AdminUsers"
  policy = file("admin-policy.json")
}

resource "local_file" "pet" {
  filename = var.filename
  count   = length(var.filename)
}

```

main.tf

```

resource "local_file" "pet" {
  filename = var.filename
  for_each = toset(var.region)
}

variable region {
  type      = list
  default   = ["us-east-1",
              "us-east-1",
              "ca-central-1"]
  description = "A list of AWS Regions"
}

```

>_ \$ terraform console
>file("/root/terraform-projects/main.tf")
resource "aws_instance" "development" {
 ami = "ami-0edab43b6fa892279"
 instance_type = "t2.micro"
}
> length(var.region)
3
>toset(var.region)
[
 "ca-central-1",
 "us-east-1",
]
>

Functions

Numeric Functions

String Functions

Collection Functions

Type Conversion
Functions

Numeric Function

Numeric Functions

The terminal window shows the output of a Terraform console session. It starts with a variable definition in variables.tf, followed by several numeric operations:

```
variables.tf
variable "num" {
  type = set(number)
  default = [ 250, 10, 11, 5 ]
  description = "A set of numbers"
}

$ terraform console
>_
> max (-1, 2, -10, 200, -250)
200
> min (-1, 2, -10, 200, -250)
-250
> max(var.num...)
250
> ceil(10.1)
11
> ceil(10.9)
11
> floor(10.1)
10
> floor(10.9)
10
```

String function

The terminal window shows the output of a Terraform console session. It starts with a variable definition in variables.tf, followed by various string manipulation operations:

```
variables.tf
variable "ami" {
  type = string
  default = "ami-xyz,AMI-ABC,ami-efg"
  description = "A string containing ami ids"
}

$ terraform console
>_
> split("", "ami-xyz,AMI-ABC,ami-efg")
[ "ami-xyz", "AMI-ABC", "ami-efg" ]
> split("", var.ami)
[ "ami-xyz", "AMI-ABC", "ami-efg" ]
> lower(var.ami)
ami-xyz,ami-abc,ami-efg
> upper(var.ami)
AMI-XYZ,AMI-ABC,AMI-EFG
> title(var.ami)
Ami-Xyz,AmI-ABC,Ami-Efg
> substr(var.ami, 0, 7)
ami-xyz
> substr(var.ami, 8, 7)
AMI-ABC
> substr(var.ami, 16, 7)
ami-efg
```

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

String Functions

The terminal window shows the contents of a file named `variables.tf` and a session in the `terraform console`.

`variables.tf` content:

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

`terraform console` session output:

```
>_
$ 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

The terminal window shows the contents of a file named `variables.tf` and a session in the `terraform console`.

`variables.tf` content:

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

`terraform console` session output:

```
>_
$ 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

Map Functions

The terminal window shows the contents of `variables.tf` and the output of running `terraform console`. The `variables.tf` file defines a variable `ami` as a map type with three entries: "us-east-1" = "ami-xyz", "ca-central-1" = "ami-efg", and "ap-south-1" = "ami-ABC". The `description` field is "A map of AMI ID's for specific regions". The `terraform console` output shows the keys and values of the map, and a successful lookup for the key "ca-central-1".

```
variables.tf
variable "ami" {
  type = map
  default = {
    "us-east-1" = "ami-xyz",
    "ca-central-1" = "ami-efg",
    "ap-south-1" = "ami-ABC"
  }
  description = "A map of AMI ID's for specific regions"
}

>_
$ terraform console
> keys(var.ami)
[
  "ap-south-1",
  "ca-central-1",
  "us-east-1",
]
> values(var.ami)
[
  "ami-ABC",
  "ami-efg",
  "ami-xyz",
]
> lookup(var.ami, "ca-central-1")
ami-efg
```

KODEKLOUD

The terminal window shows the contents of `variables.tf` and the output of running `terraform console`. The `variables.tf` file is identical to the one in the previous screenshot. The `terraform console` output shows a successful lookup for the key "ca-central-1". However, when attempting a lookup for the key "us-west-2", it fails with an error message: "Error: Error in function call on <console-input> line 1: (source code not available)". The error also states that `var.ami` is a map of string with 3 elements. A subsequent lookup for the key "us-west-2" with the value "ami-pqr" is successful.

```
variables.tf
variable "ami" {
  type = map
  default = {
    "us-east-1" = "ami-xyz",
    "ca-central-1" = "ami-efg",
    "ap-south-1" = "ami-ABC"
  }
  description = "A map of AMI ID's for specific regions"
}

>_
$ terraform console
> lookup(var.ami, "us-west-2")
Error: Error in function call
on <console-input> line 1:
(source code not available)
|-----
| var.ami is map of string with 3 elements
Call to function "lookup" failed: lookup failed
to find 'us-west-2'.

> lookup (var.ami, "us-west-2", "ami-pqr")
ami-pqr
```

KODEKLOUD

Conditional Expression

We can also test arithmetic and logical operator using terraform console

Numeric operator

Numeric Operators

```
>_
$ terraform console
> 1 + 2
3

> 5 - 3
2

> 2 * 2
4

> 8 / 2
4
```

Equality operator

Equality Operators

```
>_
$ terraform console

> 8 == 8
true

8 == 7
false

> 8 != "8"
true
```

Comparision operator

Comparison Operators

```
>_  
$ terraform console  
  
> 5 > 7  
false  
  
> 5 > 4  
true  
  
> 5 > 5  
False  
  
> 5 >= 5  
true  
  
> 4 < 5  
true  
  
> 3 <= 4  
true
```

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]
```

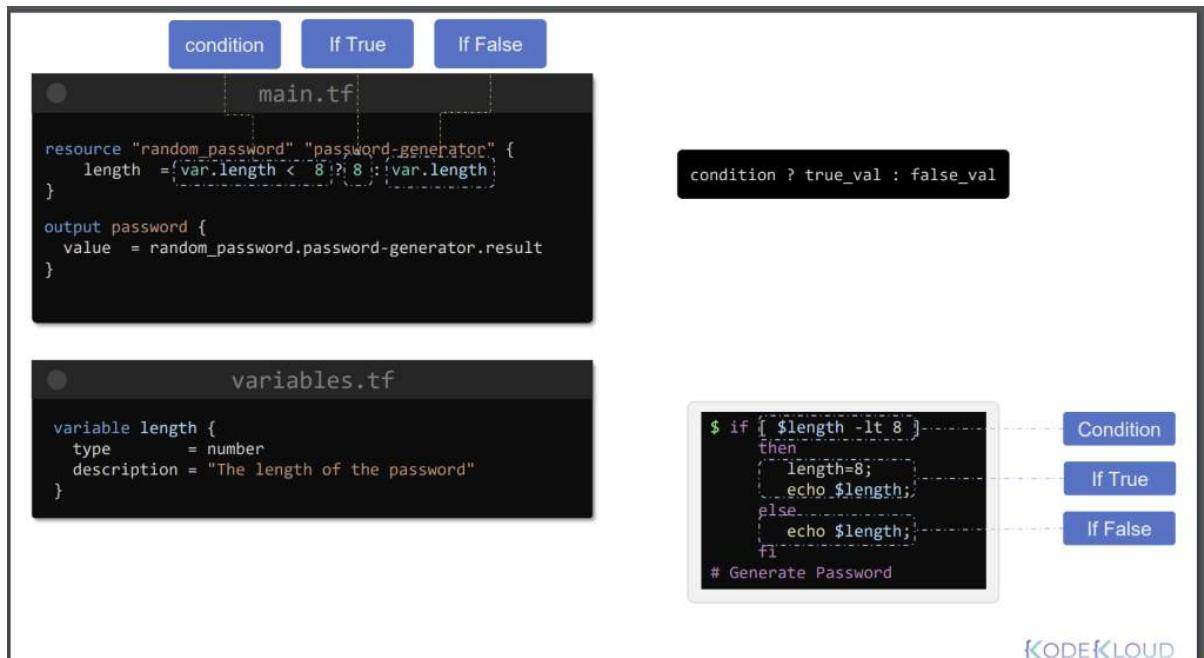
```
$ if [ $length -lt 8 ]
then
  length=8;
  echo $length;
else
  echo $length;
fi
# Generate Password
```

Condition

If True

If False

KODEKLOUD



KODEKLOUD

```
>_
$ terraform apply -var=length=5
Terraform will perform the following actions:

# random_password.password-generator will be created
+ resource "random_password" "password-generator" {
    + id          = (known after apply)
    + length      = 8
}

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

Outputs:
password = &(1Beiaq

$ terraform apply -var=length=12
Terraform will perform the following actions:

# random_password.password-generator must be replaced
-/+ resource "random_password" "password-generator" {
    ~ id          = "none" -> (known after apply)
    ~ length      = 8 -> 12 # forces replacement.

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

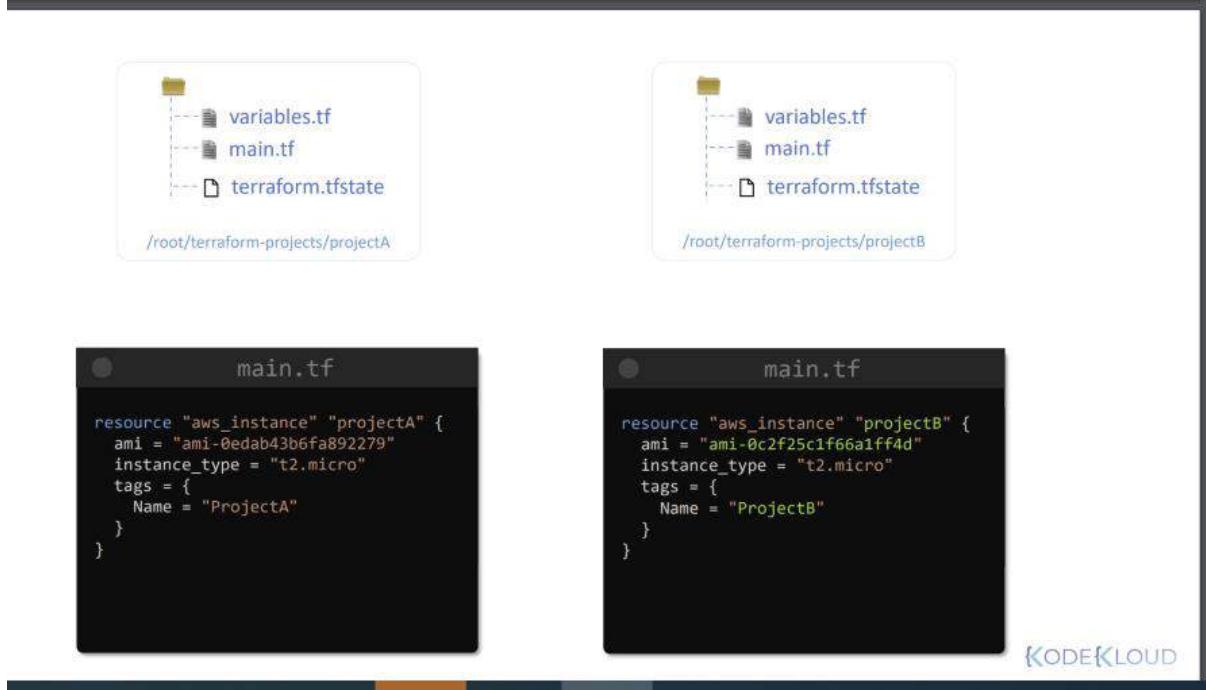
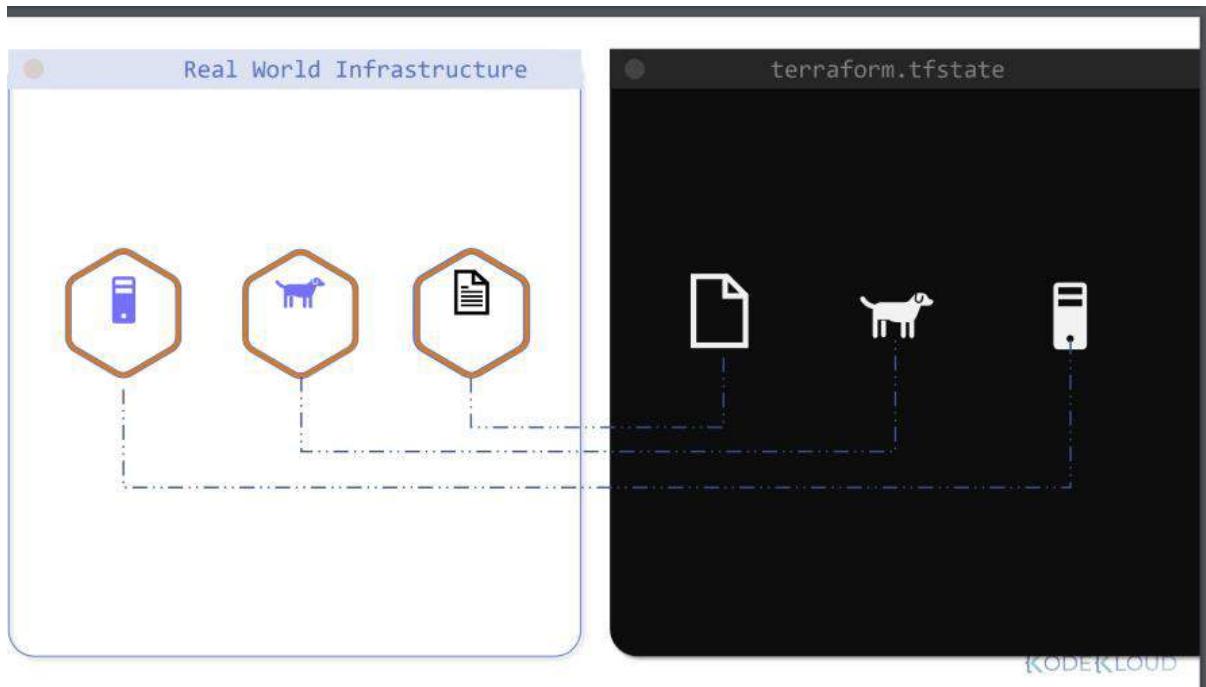
Outputs:
password = 8B@o}{cUzrZ7
```

KODEKLOUD

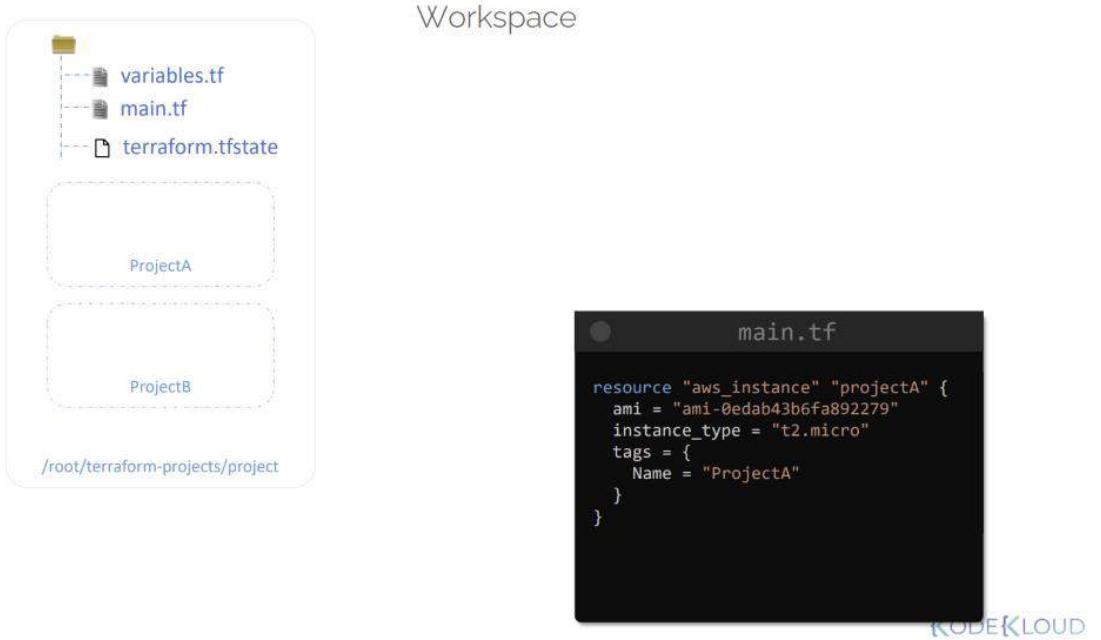
Terraform workspace oss

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)

Workspace

```

>_
$ terraform workspace new ProjectA
Created and switched to workspace "ProjectA"!

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.

$ terraform workspace list
  default
* ProjectA

```

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

```

variables.tf
variable region {
  default = "ca-central-1"
}
variable instance_type {
  default = "t2.micro"
}
variable ami {
  type     = map
  default  = {
    "ProjectA" = "ami-0edab43b6fa892279",
    "ProjectB" = "ami-0c2f25c1f66a1ff4d"
  }
}

main.tf
resource "aws_instance" "projectA" {
  ami           =
  instance_type =
  tags          = {
    Name =
  }
}

```

```

variables.tf
variable region {
  default = "ca-central-1"
}
variable instance_type {
  default = "t2.micro"
}
variable ami {
  type     = map
  default  = {
    "ProjectA" = "ami-0edab43b6fa892279",
    "ProjectB" = "ami-0c2f25c1f66a1ff4d"
  }
}

main.tf
resource "aws_instance" "projectA" {
  ami           =
  instance_type =
  tags          = {
    Name =
  }
}

```

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.
```

KODEKLUB

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
```