Terraform

Thinknyx®

Terraform Cheatsheet

Terraform

TERRAFORM CHEATSHEET PRESENTED BY THINKNYX TECHNOLOGIES

Thinknyx®

Install Terraform

To Install Terraform in Linux

```
# Get link for the latest 64-bit version of Terraform for Linux from https://www.terraform.io/downloads.html

$ wget https://releases.hashicorp.com/terraform/1.0.2/terraform_1.0.2_linux_amd64.zip
$ unzip terraform_1.0.2_linux_amd64.zip
$ sudo cp terraform /usr/bin

$ terraform --version
Terraform v1.0.2
on linux_amd64
```

To Install Terraform on Window 10, using PowerShell as Administrator

```
# Get link for the latest 64-bit version of Terraform for Windows from https://www.terraform.io/downloads.html

PS C:\thinknyx> Invoke-WebRequest -uri https://releases.hashicorp.com/terraform/1.0.2/terraform_1.0.2_windows_amd64.zip -out terraform.zip
PS C:\thinknyx> Expand-Archive terraform.zip -DestinationPath C:\thinknyx\terraform\ -Force -Verbose
PS C:\thinknyx> $INCLUDE = "C:\thinknyx\terraform"
PS C:\thinknyx> $OLDPATH = [System.Environment]::GetEnvironmentVariable('PATH','machine')
PS C:\thinknyx> $NEWPATH = "$OLDPATH;$INCLUDE"
PS C:\thinknyx> [Environment]::SetEnvironmentVariable("PATH", "$NEWPATH", "Machine")

PS C:\thinknyx> terraform --version
Terraform v1.0.2
on windows_amd64
```

Terraform

TERRAFORM CHEATSHEET PRESENTED BY THINKNYX TECHNOLOGIES

Thinknyx®

Important Blocks in Terraform

- ❑ Providers
- ❑ Resources
- ❑ Data Sources
- ❑ Variables
- ❑ Outputs
- ❑ Modules
- ❑ Provisioners

Providers

Providers are a logical abstraction of an upstream platform APIs

# **Few most commonly used providers**

- ❑ AWS
- ❑ Azurerm
- ❑ Google
- ❑ Kubernetes
- ❑ Null
- ❑ Template
- ❑ Local
- ❑ Random
- ❑ Archive

Examples of Provider Block:

```
❑ provider "aws"{
      region = "us-east-2"
   }
❑ provider "local" {}
```

Resources

Resource Blocks describe individual service or object available at provider using attributes

# Examples

- ❑ aws_instance → EC2 Server in AWS
- ❑ aws_lb → Elastic Load Balancer in AWS
- ❑ kubernetes_deployment → Deployment object in k8s
- ❑ kubernetes_server → Service object in k8s

Example of Resource Block:

```
resource "aws_instance" "ubuntu_server"{
   ami = "ami-00399ec92321828f5"
   instance_type = "t2.micro"
   key_name = "thinknyx"
   tags = {
      Name = "thinknyx"
   }
}
```

Data Sources

Data Source Blocks helps in retrieving the information about existing resource or to compute the information to be used further

Example of Data Source Block:

```
❑ data "aws_availability_zones" "availability_zones"{
      state = "available"
   } # Fetch names of Zones in current AWS Region
❑ data "local_file" "input" {
      filename = "input.txt"
   } # Reads a file from local system
```

Variables

Variables helps in customizing the behavior of terraform configurations without making any change in the actual code

Example of Variable Block:

```
variable "ami"{
  default = "ami-00399ec92321828f5"
}
```

Example of Variable usage:

```
 resource "aws_instance" "ubuntu_server"{
   ami = var.ami #Reference to variable
   instance_type = "t2.micro"
   key_name = "thinknyx"
   tags = {
      Name = "thinknyx"
   }
}
```

Outputs

Outputs in Terraform act as a return value from Resource creation to expose to the User

Example of Variable Block:

```
output "ec2_public_ip" {
  value = aws_instance.ubuntu_server.public_ip
} # Expose Public IP of EC2 Server created
output "application_url" {
  value =
"http://${aws_instance.ubuntu_server.public_ip}:8080"
} # Create a customized output
```

Terraform **TERRAFORM CHEATSHEET PRESENTED BY THINKNYX TECHNOLOGIES** Thinknyx®

Provisioners

Provisioners also known as "THE LAST RESORT" are the blocks which helps in to automate certain behavior or requirements which is not possible in Terraform Declarative model

Types:

- ❑ remote-exec
- ❑ local-exec
- ❑ file

❑ **Remote-exec** - For actions need to be executed on remote-machines

Example:

```
resource "null_resource" "install_packages" {
 provisioner "remote-exec" {
   connection {
     type = "ssh"
     user = "ubuntu"
     private_key = file("c:/thinknyx/thinknyx.pem")
     host =aws_instance.ubuntu_server.public_ip
   }
   inline = [
     "sudo apt-get update –y",
     "sudo apt-get install nodejs npm apache2-y"
   ]
 }
}
```

❑ **Local-exec** - For actions needs to be executed on Local machine where Terraform is running

Example:

```
resource "null_resource" "application_sanity" {
 provisioner "local-exec" {
   command = "curl
http://${aws_instance.ubuntu_server.public_ip"
 }
}
```

❑ **File** - For Copying files from Local machine to remote machines

Example:

```
resource "null_resource" "copy_config" {
 provisioner "file" {
   connection {
     type = "ssh"
     user = "ubuntu"
     private_key = file("c:/thinknyx/thinknyx.pem")
     host =aws_instance.ubuntu_server.public_ip
   }
   source = "apache2.conf"
   destination = "/etc/apache2/apache2.conf"
 }
}
```

Terraform TERRAFORM CHEATSHEET PRESENTED BY THINKNYX TECHNOLOGIES

Modules

Modules are the collection of definitions for multiple resource together to be used together. Also, modules helps in creating reusable resource configurations to create standards across organization or teams while creating same type of resources.
By default every terraform project we create is module also known as Root Module. With in Root Modules we call other modules known as Child Modules which could be available locally or could easily be downloaded from remote locations know Terraform Registry.

❑ Creating module

```
# Create Module Directory
mkdir ./create_ec2
vi main.tf
resource "aws_instance" "ubuntu_server"{
   ami = var.ami
   instance_type = var.instance_type
   key_name = "thinknyx"
   tags = {
      Name = var.tag_name
      Service = var.service_name
   }
}
vi variables.tf
variable "ami"{}
variable "instance_type"{}
variable "tag_name" {}
variable "service_name" {}
vi outputs.tf
output "public_ip"{
  value = aws_instance.ubuntu_server.public_ip
}
output "private_ip"{
  value = aws_instance.ubuntu_server.public_ip
}
```

❑ Calling Child Module in Root Module

```
vi create_application_servers.tf
module "create_frontend_server"{
   source = "./create_ec2"
   ami = "ami-00399ec92321828f5"
   instance_type = "t2.micro"
   tag_name = "thinknyx"
   service_name = "frontend"
}

module "create_backend_server"{
   source = "./create_ec2"
   ami = "ami-ad399ec92657234f5"
   instance_type = "t2.medium"
   tag_name = "thinknyx"
   service_name = "backend"
}

output "frontend_ip"{
   value = module.create_frontend_server.public_ip
}

output "backend_ip"{
   value = module.create_backend_server.private_ip
}
```

TERRAFORM CHEATSHEET PRESENTED BY THINKNYX TECHNOLOGIES

Backend

Backend block is used to define the remote location to be used by the Terraform to store the Terraform State File.

❑ Backend Example Block:

```
# Using AWS S3 bucket as Terraform Backend
# Note: Bucket should exists before Terraform initiation and AWS user used to connect Terraform with AWS should have access on the same

terraform {
 backend "s3" {
  bucket = "thinknyx" # S3 Bucket name
  key = "thinknyx_cheatshaet.tfstate" #State filename
  region = "us-east-2" # Region where bucket exists
 }
}
```

Statefile

Terraform State file maintain the complete information about the Terraform project, like Resource created, Data Source fetched, Outputs defined and all dependencies different resource have

❑ Sample terraform state file

```
{
 "version": 4,
 "terraform_version": "1.0.2",
 "serial": 1,
 "lineage": "151d9bf3-ef7a-11f6-84c0-1fe949089ac3",
 "outputs": {
  "application_url": {
   "value": "http://52.14.107.215:8080",
   "type": "string"
  }
 },
 "resources": [
  {
   "mode": "data",
   "type": "local_file",
   "name": "input",
   "provider": "provider[\"registry.terraform.io/hashicorp/local\"]",
   "instances": [
    {
     "schema_version": 0,
     "attributes": {
      "content": "somethigns.demo",
      "content_base64": "c29tZXRoaWducy5kZW1v",
      "filename": "test.txt",
      "id": "d4e005b708b16564d5df7edb9be0a0661cd24e0b"
     },
     "sensitive_attributes": []
    }
   ]
  }
 ]
}
{"mode":"full","isActive":false}
```

Terraform

TERRAFORM CHEATSHEET PRESENTED BY THINKNYX TECHNOLOGIES

Thinknyx®

Initialize Project

```
# To Initialize terraform workspace
$ terraform init
# To reconfigure backend
$ terraform init -reconfigure
# To reconfigure backend & migrate existing state to same
$ terraform init -migrate-state
```

Provision Infrastructure

```
# To Validate Code
$ terraform validate
# To create execution plan (Dry Run)
$ terraform plan
# To create execution plan (Dry Run) & save it in file
$ terraform plan -out plan.txt
# To implement changes in actual Environment with an approval prompt
$ terraform apply
# To Implement changes without approval
$ terraform apply -auto-approve
# To Implement Specific block changes
$ terraform apply -target aws_instance.ubuntu_server
# To set variable during implementation
$ terraform apply -var instance_type=t2.micro
# To set multiple variable during implementation
$ terraform apply -var instance_type=t2.micro -var key_name=kul
```

Destroy Infrastructure

```
# To destroy entire infrastructure with  an approval prompt
$ terraform destroy
# To destroy entire infrastructure without approval
$ terraform destroy –auto-approve
# To destroy specific block changes
$ terraform destroy -target aws_instance.ubuntu_server
```

Workspaces

Terraform workspaces helps in maintaining multiple state for same configuration. For example same configuration needs to be implemented for different environments or for different regions in case of AWS/Azure/GCP

```
# Workspace can be created post terraform init
# To create & select new workspace
$ terraform workspace new us-east-1
$ terraform workspace new us-east-2
# To select existing workspace
$ terraform workspace select us-east-1
# To List all workspace
$ terraform workspace list
# To display current workspace
$ terraform workspace show
# To delete an empty workspace
$ terraform workspace delete us-east-2
# To delete a non-empty workspace
$ terraform workspace delete us-east-2 -force
```

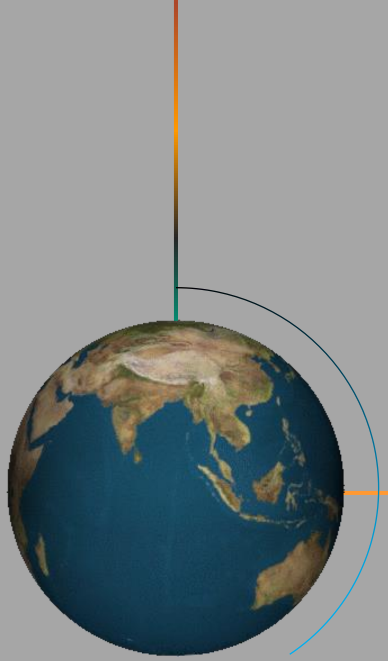
Inspect

```
# To show current state
$ terraform show
# To display list of objects in state file
$ terraform state list
# To display state of specific object
$ terraform state show aws_instance.ubuntu_server
# To display output values
$ terraform output
# To display dependency graph
$ terraform graph
```

Miscellaneous

```
# To remove lock from state file when automatic unlocking failed, Terraform will display the LOCK_ID when unlocking failed
$ terraform force-unlock LOCK_ID
# To forcefully delete & recreate an object on next execution
$ terraform taint aws_instance.ubuntu_server
# To remove taint
$ terraform untaint aws_instance.ubuntu_server
```

Thinknyx®



Thinknyx®

YOU TRUST, WE DELIVER

Reach out to us at:

www.thinknyx.com

support@thinknyx.com

+91 9810344919/9717917973