[Terraform-Tutorial]

https://github.com/Patelvijaykumar/Terraform-Tutorial

Introduction of Terraform


```
resource "aws_instance" "my-web" {
  ami = "ami-9ad37hhf2"
  instance_type = "t2.micro"
}
```

Terraform

- Infrastructure as code
- Automation of Infrastructure
- Maintain the state of infrastructure (state mgmt)
- Maintain infrastructure change history using version control system like git

Terraform

- Ansible,chef ,puppet have focusing automating installation and configuration of software
  - Maintain system in compliance in certain mode
- Terraform can automate any cloud Provider
  - AWS,AZURE,GCP,Digital Ocean

Fundamental of Terraform

Terraform Goal

- Unify the view of resources using infrastructure as code
- Support the modern data center (IaaS,PaaS,SaaS)
- Expose a way to satisfy and predictably change infrastructure
- Manage anything with API

Terraform vs other tool

- Provides a high-level description of infrastructure
- Support parallel management of resources
- Separate planning from execution (dry-run)
- Detailed Document
- Support All Major Cloud Provider

Terraform state

- Open Source
- First Release July 28, 2014
- Over 700 Contributor
- Over 6200 github stars
- One release every 2 week

Sample Template create aws instance

```
Resource "aws_instance" "my-web-server"{
    ami="ami-9adtedj65"
    instance_type="t2.micro"
}
```

Infrastructure as Code

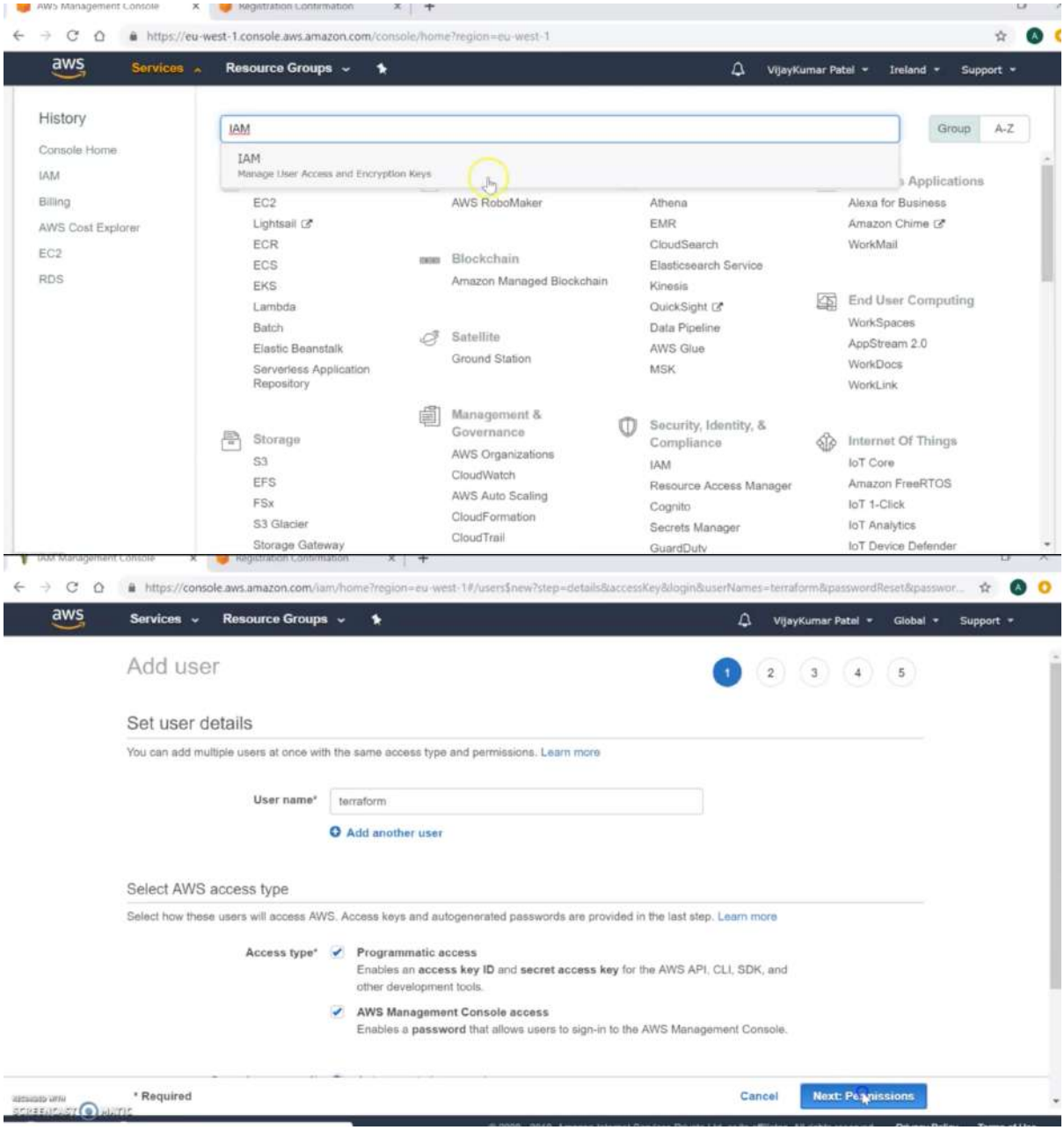
- Provide a coding workflow to create infrastructure
- Expose a workflow for managing updates to existing infrastructure
- Integrate with application code workflow (git,code review)
- Provide modular,shareable component for separation of concern
- Human-readable configuration is designed for human consumption so user can quickly interpret and understand their infrastructure configuration
- HCL includes a full json parser for machine-generated configuration

Terraform Installation

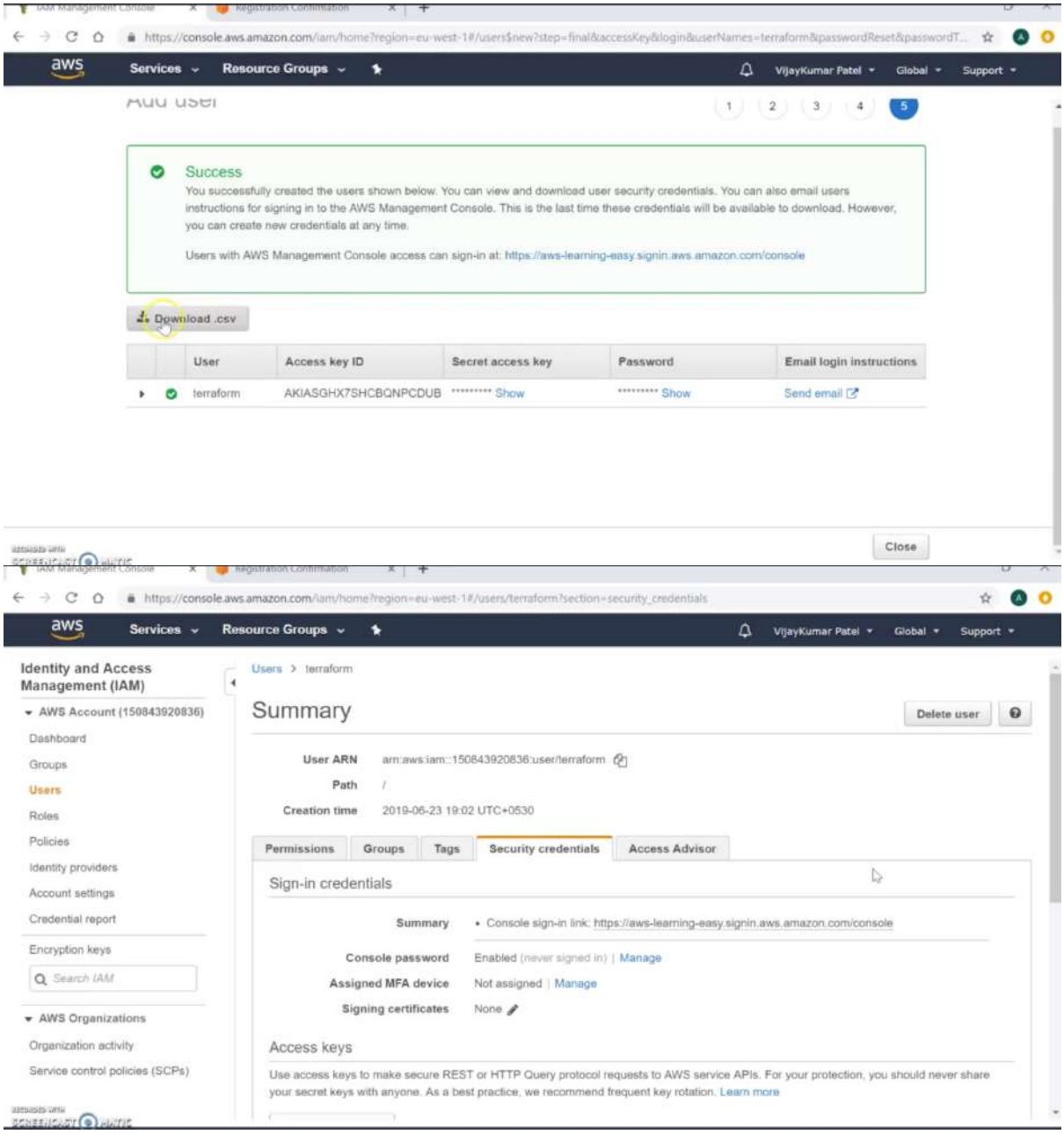
Terraform Basics Setup

Terraform

- Virtual Machine (EC2) On AWS
  - Open AWS Account
  - Create Admin User
  - Terraform Script for creating Instance
  - Terraform Apply


https://eu-west-1.console.aws.amazon.com/console/home?region=eu-west-1
Services
Resource Groups
VijayKumar Patel
Ireland
Support
History
Console Home
IAM
Billing
AWS Cost Explorer
EC2
RDS
IAM
Manage User Access and Encryption Keys
Group
A-Z
EC2
Lightsail
ECR
ECS
EKS
Lambda
Batch
Elastic Beanstalk
Serverless Application Repository
AWS RoboMaker
Blockchain
Amazon Managed Blockchain
Satellite
Ground Station
Athena
EMR
CloudSearch
Elasticsearch Service
Kinesis
QuickSight
Data Pipeline
AWS Glue
MSK
Alexa for Business
Amazon Chime
WorkMail
End User Computing
WorkSpaces
AppStream 2.0
WorkDocs
WorkLink
Storage
S3
EFS
FSx
S3 Glacier
Storage Gateway
Management & Governance
AWS Organizations
CloudWatch
AWS Auto Scaling
CloudFormation
CloudTrail
Security, Identity, & Compliance
IAM
Resource Access Manager
Cognito
Secrets Manager
GuardDuty
Internet Of Things
IoT Core
Amazon FreeRTOS
IoT 1-Click
IoT Analytics
IoT Device Defender

https://console.aws.amazon.com/iam/home?region=eu-west-1#/users$new?step=details&accessKey&login&userNames=terraform&passwordReset&passwor...
Services
Resource Groups
VijayKumar Patel
Global
Support
Add user
1
2
3
4
5
Set user details
You can add multiple users at once with the same access type and permissions. Learn more
User name*
terraform
Add another user
Select AWS access type
Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more
Access type*
Programmatic access
Enables an access key ID and secret access key for the AWS API, CLI, SDK, and other development tools.
AWS Management Console access
Enables a password that allows users to sign-in to the AWS Management Console.
* Required
Cancel
Next: Permissions



Terraform First Script

Aws-instance-example.tf

```
provider "aws"{
 region     = "us-east-1"
 access_key = "AKIASGHX7SHCAXOJTTXU"
 secret_key = "M2eRRCodTbRn60jiQvjib/3+wz3SK/6B8atMJAL0"
 version = "~> 2.0"
}

resource "aws_instance" "web1" {
  ami           = "ami-035b3c7efe6d061d5"
  instance_type = "t2.micro"
}

```

Terraform Commands

Command | Terraform plan

- The plan show you what will happen
- You can save plans to guarantee what will happen
- Plan show reason for certain action (such as recreate)
- Prior to Terraform,User had to guess change ordering,parallelization, and rollout effect

Command | Terraform plan

- + indicates a resource will be created
- - indicates a resource will be destroyed
- ~ indicates a resource will be update in plan
- -/+ indicates a resource will be destroyed and re-created

Terraform Plan

```
# aws_instance.web1 will be created
  + resource "aws_instance" "web1" {
      + ami                          = "ami-035b3c7efe6d061d5"
      + arn                          = (known after apply)
      + associate_public_ip_address  = (known after apply)
      + availability_zone            = (known after apply)

Plan: 1 to add, 0 to change, 0 to destroy.
------------------------------------------------------------------------
Note: You didn't specify an "-out" parameter to save this plan, so
Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.
```

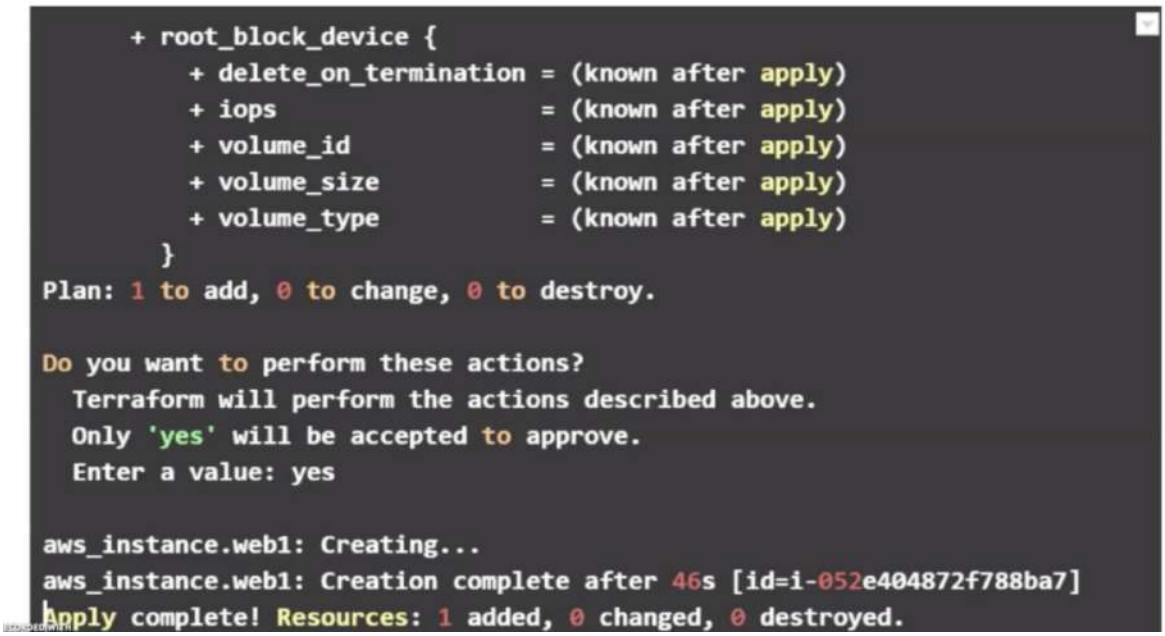
Command | terraform apply

- Execute changes to reach desired state
- Parallelism changes when possible
- Handles and recovers transient errors safely

Command | terraform apply

- Current state to target state
- Update Existing resource when update are allowed
- Re-creates existing resources when updates are not allowed

Terraform Apply



```
resource "aws_instance" "web" {
  ami           = "${data.aws_ami.ubuntu.id}"
  instance_type = "t2.micro"
  key_name      = "Bastion"
  tags {
    Name = "HelloWorld"
  }
}
```

Changes in code

Terraform plan changes

```
availability_zone:              "us-west-2a" => "<computed>"
ebs_block_device.#:             "0" => "<computed>"
ephemeral_block_device.#:       "0" => "<computed>"
instance_state:                 "running" => "<computed>"
instance_type:                  "t2.micro" => "t2.micro"
ipv6_address_count:             "" => "<computed>"
ipv6_addresses.#:               "0" => "<computed>"
key_name:                       "" => "Bastion" (forces new resource)
network_interface.#:            "0" => "<computed>"
network_interface_id:           "eni-c61346e5" => "<computed>"
```

Terraform Apply changes

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

The state of your infrastructure has been saved to the path
below. This state is required to modify and destroy your
```

Command: terraform destroy

- Destroys running infrastructure
- Does not touch infrastructure not managed by terraform

terraform destroy

```
      - root_block_device {
          - delete_on_termination = true -> null
          - iops                  = 100 -> null
          - volume_id             = "vol-0bc8ddac28de52548" -> null
          - volume_size           = 8 -> null
          - volume_type           = "gp2" -> 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.

  Enter a value: yes
aws_instance.web1: Destroying... [id=i-052e404872f788ba7]
Destroy complete! Resources: 1 destroyed.
```

Terraform Commands Demo
https://github.com/Patelvijaykumar/Terraform-Tutorial/tree/master/aws-instance-first-script

Terraform Plan out file

```
terraform plan -out confing.terraform
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.web1: Refreshing state... [id=i-018993dc3dde1edc1]
------------------------------------------------------------------------
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create
Terraform will perform the following actions:
  # aws_instance.web1 will be created
  + resource "aws_instance" "web1"
      + ami                          = "ami-035b3c7efe6d061d5"
      + arn                          = (known after apply)
Plan: 1 to add, 0 to change, 0 to destroy.
------------------------------------------------------------------------
This plan was saved to: confing.terraform
To perform exactly these actions, run the following command to apply:
    terraform apply "confing.terraform"
```

Terraform apply with plan out file

```
terraform apply confing.terraform
aws_instance.web1: Creating...
Please wait for Terraform to exit or data loss may occur.

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

Terraform Basic

Terraform Variable

Terraform Variable

- Terraform Script Basic structure (recommended)
  - Main script
  - Variable script
  - Output script
- Use variable to hide secret
- Variable value change as per region or account specific
  - Instance type value can be differ in dev and prod account

Terraform main.tf script without Variable

```
provider "aws"{
 region     = "us-east-1"
 access_key = "AKIASGHX[redacted]"
 secret_key = "M2eRRCodTbRn60jiQvjib/3+wz3S[redacted]"
 version = "~> 2.0"
}
```

Terraform script with variable

```
provider "aws"{
 region     = "${var.region}"
 access_key = "${var.access_key = ""}"
 secret_key = "${var.secret_key}"
 version = "~> 2.0"
}
```

Terraform Variable.tf

```
variable "access_key" { }
variable "secret_key" { }
variable "region" {
default="us-east-1"
}
```

Terraform terraform.tfvars

```
access_key=""
secret_key =""
region="us-east-1"
```

Variable Demo

https://github.com/Patelvijaykumar/Terraform-Tutorial/tree/master/aws-instance-first-script

Software Provisioning

File upload

```
resource "aws_instance" "web-server" {
  ami           = "${lookup(var.ami_id, var.region)}"
  instance_type = "t2.micro"
  key_name      = "terraform"

  provisioner "file" {
    source      = "index.html"
    destination = "/tmp/index.html"
  }
```

File upload Host Connection

```
resource "aws_instance" "web-server" {
  ami           = "${lookup(var.ami_id, var.region)}"
  instance_type = "t2.micro"
  key_name      = "terraform"

  provisioner "file" {
    source      = "index.html"
    destination = "/tmp/index.html"
  }

  connection {
    user        = "ec2-user"
    private_key = "${file("${var.private_key_path}")}"
    host        = "${aws_instance.web-server.public_ip}"
```

Remote Execution Commands

```
provisioner "remote-exec" {
    inline = [
      "sudo yum install -y httpd;sudo cp /tmp/index.html /var/www/html/",
      "sudo service httpd restart",
      "sudo service httpd status"
    ]
  }
```

Software Provisioning Demo

https://github.com/Patelvijaykumar/Terraform-Tutorial/tree/master/Software-provision

Terraform Output

Terraform output

- Many Resources dependent with other resource attribute
  - Resource Name
  - Resource arn
  - EC2 IP (Public IP)
  - EC2 DNS name
- Terraform keeps attribute of all resources which terraform creates
- To get attribute can use as a output

Output Define

```
output "public_ip"{
  value="${aws_instance.web-server.public_ip}"
}
```

- Resource Type. Resource Name.Attribute
- Aws_instance attributes example are (id, arn,public_dns,public_ip)
- https://www.terraform.io/docs/providers/aws/r/instance.html (See Attributes Reference)

Terraform attribute in script

```
  provisioner "local-exec" {
    command = "echo ${aws_instance.web-server.private_ip} >> ip_list.txt"
  }

  provisioner "local-exec" {
    command = "echo ${aws_instance.web-server.arn} >> arn.txt"
  }
}
```

Terraform Output Demo

https://github.com/Patelvijaykumar/Terraform-Tutorial/tree/master/terraform-output

Terraform Remote State

Terraform State

- Terraform keep remote state of infrastructure
- State will be stores on terraform.tfstate file
- Terraform.tfstate.backup for previous state file

Terraform state

- Terraform state file can be in remote , using backend functionality
- A "backend" in Terraform determines how state is loaded and how operation such as apply is executed. This abstraction enables non-local file state storage, remote execution and State Locking.
- Default backend is terraform state file
- Other backend are
  - S3
  - Consul

Terraform backend Supports

1. Non-local file state storage
2. Remote operations
3. State Locking

Terraform backend Config

1. Add the backend in main.tf file
2. Run initialize process

Terraform backend Consule

```
terraform {
  backend "consule"{
    address = "mybackup.cusule.io"
    path ="terraform/backup/"
  }
}
```

Terraform Backend s3 Setup

```
terraform {
  backend "s3" {
    bucket = "mybucket"
    key    = "path/to/my/key"
    region = "us-east-1"
  }
}
```

terraform backend Remote Setup

```
data "terraform_remote_state" "network" {
  backend = "s3"
  config {
    bucket = "terraform-state-prod"
    key    = "network/terraform.tfstate"
    region = "us-east-1"
  }
}
```

Terraform Script State Lock

dynamodb_table - (Optional) The name of a DynamoDB table to use for state locking and consistency. The table must have a primary key named LockID. If not present, locking will be disabled.

- Create LockId Field in dynamodb Table

Terraform Remote State Demo

https://github.com/Patelvijaykumar/Terraform-Tutorial/tree/master/terraform-remote-state

Terraform Data Sources

Terraform data sources

- Data Source provides a dynamic Information
- Many Information of AWS are needed dynamically without create new resources
- Terraform exposed this information using data sources
- Example:
  - VPC ID
  - Security ID
  - AMI List
  - Availability Zone

Terraform Data Sources Demo

https://github.com/Patelvijaykumar/Terraform-Tutorial/tree/master/terraform-data-source

Terraform Modules

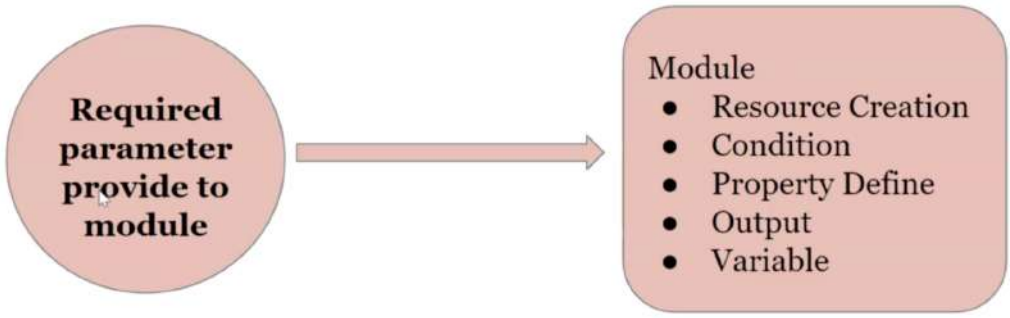
Terraform Modules

- Terraform provides modules which allow us to abstract away reusable parts, which we can configure once, and use everywhere
- You can use module to make terraform more organized
- Modules allow us to group resources together, define input variables which are used to change needed resource configuration parameters, define output variables that other resources or modules can use.

Terraform Modules

- Module reuse from
  - Third party modules like git
  - Reuse part of your code example instance create

How Terraform Modules works



Definition
source = "git::https://github.com/Patelvijaykumar/module-terraform.git"
source = "git::ssh://username@example.com/storage.git"

Terraform Modules

Use Module from git

```
module "module-example" {
  source = "github.com/Patelvijaykumar/terraform-aws-instance-template.git"

}
```

Use Module local

```
module "module-example" {
  source = "./terraform-aws-instance-template"

}
```

Terraform Modules

Example of Passing Arguments to module

```
# Demostration of pass agruments in module
module "module-example" {
  source = "github.com/Patelvijaykumar/terraform-aws-instance-template.git"

    region                  = "us-east-1"
    ami_id                  = "ami-035b3c7efe6d061d5"
    instance_type           = "t2.micro"
    tag                     = "module example"

}
```

Terraform Modules

Main.tf

```
provider "aws" {
  region = "${var.region}"
}

resource "aws_instance" "web" {
  ami           = "${var.ami_id}"
  instance_type = "${var.instance_type}"

  tags = {
    Name = "${var.tag}"
  }
}
```

Variables.tf

```
variable "ami_id" {}
variable "region" {}
variable "instance_type" {}
variable "tag" {
default="Testing"
}
```

Output.tf

```
output "instance_ip" {
  value = ["${aws_instance.web.public_ip}"]
}
```

Terraform Modules

Use the output from the module in your code

```
output "instance_public_ip_address"{
  value="${module.module-example.instance_ip}"
}
```

Terraform Module-Demo

https://github.com/Patelvijaykumar/Terraform-Tutorial/tree/master/terraform-module