Importing Resources

Importing existing resources

```
[root@techlanders]# cat resource.tf
resource "aws_instance" "gagan-ec2" {
  ami           = "ami-0c94855ba95c71c99"
  instance_type = "t2.micro"
  availability_zone = "us-east-1e"
}
[root@techlanders]#

#use terraform import command to import state in terraform statefile
[root@techlanders]# terraform import aws_instance.gagan-ec2 i-073cd0c68788f5c57
[root@techlanders]# terraform show
[root@techlanders]# terraform plan
[root@techlanders]# terraform apply
```

Importing existing resources

- You can import existing resources – which are not created using terraform command, into terraform state using terraform import command.
- The current implementation of Terraform import can only import resources into the state. It does not generate configuration. A future version of Terraform will also generate configuration.
- Because of this, prior to running terraform import it is necessary to write a resource configuration block for the resource manually, to which the imported object will be attached.
- This command will not modify your infrastructure, but it will make network requests to inspect parts of your infrastructure relevant to the resource being imported.

Terraform Workspace

Workspaces

- Workspace is to create multiple isolated environments in same directory.
- Terraform starts with a single workspace named "default” and same can’t be deleted.
- Named workspaces allow conveniently switching between multiple instances of a single configuration within its single backend.
- A common use for multiple workspaces is to create a parallel, distinct copy of a set of infrastructure in order to test a set of changes before modifying the main production infrastructure. For example, a developer working on a complex set of infrastructure changes might create a new temporary workspace in order to freely experiment with changes without affecting the default workspace.
- It’ll create terraform.tfstate.d directory with internal workspace-name subdirectories to handle state files.

Workspaces

- terraform workspace list
- terraform workspace new {new-workspace-name}
- terraform workspace show
- terraform workspace select {workspace-name}
- terraform workspace delete {workspace-name}

Data Sources

Data Sources

- Data sources allow data to be fetched or computed for use elsewhere in Terraform configuration. Use of data sources allows a Terraform configuration to make use of information defined outside of Terraform or defined by another separate Terraform configuration.
- Each provider may offer data sources alongside its set of resource types.

Data Sources

```
#aws ec2 describe-images    --region us-east-2    --image-ids ami-00399ec92321828f5
#
data "aws_ami" "web" {
 owners          = ["099720109477"]
 filter {
    name = "virtualization-type"
    values = ["hvm"]
  }
most_recent = true
}
```

NOTE: Identify Image, Owners, Details from AMI section of AWS GUI

Data Sources

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

data "aws_ami" "amazon_linux" {
  most_recent = true
  owners      = ["amazon"]

  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*-x86_64-gp2"]
  }
}
data "aws_region" "current" { }
resource "aws_instance" "myawsserver" {
  ami = data.aws_ami.amazon_linux.id
  instance_type = "t2.micro"

  tags = {
    Name = "Techlanders-aws-ec2-instance"
    Env = "Dev"
  }
}
output "myawsserver-ip" {
  value = aws_instance.myawsserver.public_ip
}
output "region" {
 value = data.aws_region.current.name
}
```

Data Sources

```
data "aws_ami" "amazon_linux" {
  most_recent = true
  owners      = ["309956199498"]

  filter {
    name   = "name"
    values = ["RHEL-8.*-x86_64-2-Hourly2-GP2"]
  }
}
```

Terraform Functions

Functions

The Terraform language includes a number of built-in functions that you can call from within expressions to transform and combine values.

Functions help us to perform few specific tasks (i.e. sort, search, reads, dates etc) easily with pre-written programs.

The Terraform language has a number of built-in functions that can be used in expressions to transform and combine values. Functions follow a common syntax:

<FUNCTION NAME>(<ARGUMENT 1>, <ARGUMENT 2>)

For e.g.

min(55, 3453, 2)

file

```
provisioner "remote-exec" {
   inline = [
    "touch /tmp/gagandeep",
     "sudo mkdir /root/gagan"
    ]
connection {
   type     = "ssh"
   user     = "ec2-user"
   insecure = "true"
   private_key = "${file("test.pem")}"
   host     =  aws_instance.myawsserver.public_ip
 }
```

Terraform Modules

Terraform Modules

A module is a container for multiple resources that are used together. Modules can be used to create lightweight abstractions, so that you can describe your infrastructure in terms of its architecture, rather than directly in terms of physical objects.

The .tf files in your working directory when you run terraform plan or terraform apply together form the root module. That module may call other modules and connect them together by passing output values from one to input values of another.

Usual Structure:

$ tree minimal-module/

.

├── README.md

├── main.tf

├── variables.tf

├── outputs.tf

Terraform Modules

Define variables under one resource file under /root/app1 directory and then use same as module in separate folder/resource file:

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

module "s1" {
source = "/root/app1"
bucket = “my-module-test"
instance_type = "t2.micro"
ami = "ami-00399ec92321828f5"
}
```

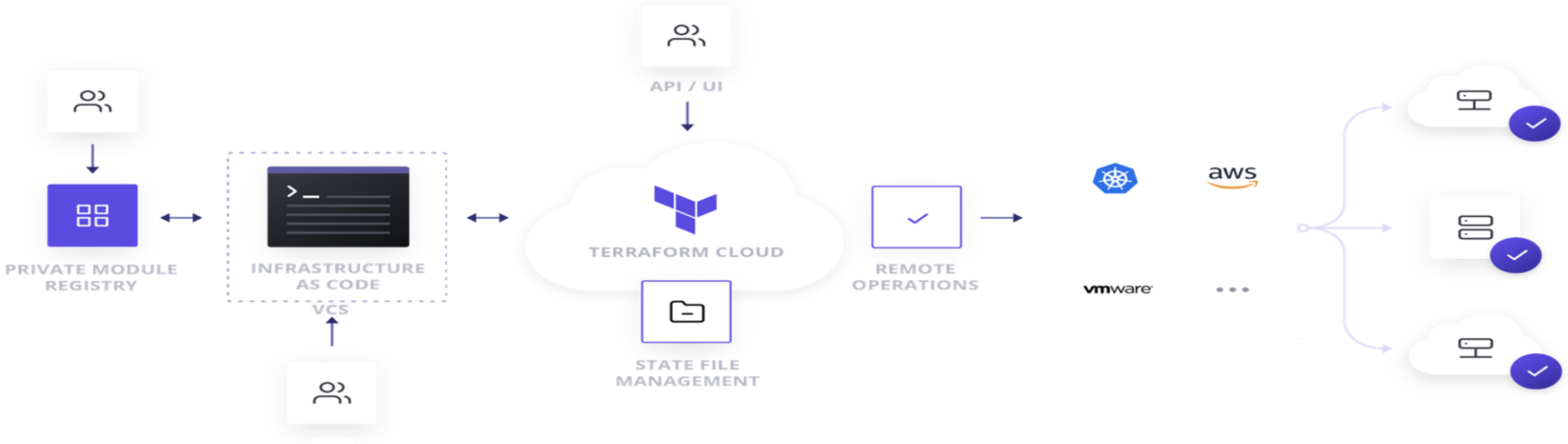
OEM Modules usage

```
provider "aws" {
  region = "us-east-2"
}
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"
  name = "gds-vpc"
  cidr = "10.0.0.0/16"
  azs             = ["us-east-2a", "us-east-2b"]
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24"]
  public_subnets  = ["10.0.101.0/24", "10.0.102.0/24"]
  enable_nat_gateway = false
  enable_vpn_gateway = false
  tags = {
    Terraform = "true"
    Environment = "dev"
  }
}
```

Terraform Cloud

Terraform Cloud

Terraform Cloud is an application that helps teams use Terraform together. It manages Terraform runs in a consistent and reliable environment and includes easy access to shared state and secret data, access controls for approving changes to infrastructure, a private registry for sharing Terraform modules, detailed policy controls for governing the contents of Terraform configurations, and more.



Terraform Cloud

It is a platform that performs Terraform runs to provision infrastructure, either on demand or in response to various events

Terraform Cloud offers a **team-oriented remote Terraform workflow**, designed to be comfortable for existing Terraform users and easily learned by new users. The foundations of this workflow are remote Terraform execution, a **workspace-based organizational model**, **version control integration**, **command-line integration, remote state management** with cross-workspace data sharing, and a **private Terraform module registry.**

Terraform Cloud runs Terraform on **disposable virtual machines in its own cloud infrastructure**. **Remote Terraform execution** is sometimes referred to as "remote operations."

Terraform Cloud

- Terraform cloud is a GUI based Cloud SaaS solution. It is offered as a multi-tenant SaaS platform and is designed to suit the needs of smaller teams and organizations.
- Benefits of Terraform Cloud:
    - It manages Terraform runs in a consistent and reliable environment.
    - Best for bigger teams, as it provides secure and easy access to shared state and secret data.
    - It offers Remote State Management, Data Sharing, Run Triggers, and Private registry for Terraform modules.
    - Role Based Access Controls (RBAC) for approving changes to infrastructure.
    - Version Control Integration with Major VCS providers like Github, Gitlab, Bitbucket, Azure DevOps
    - Full APIs support for all operations to integrate this with other tools and environments.

Terraform Cloud

- Notifications can be configured with services which support webhooks
- You can run the configuration from existing environment or from terraform cloud-based server.
- **Sentinel Policies:** Terraform Cloud embeds the Sentinel policy-as-code framework, which lets you define and enforce granular policies for how your organization provisions infrastructure. You can limit the size of compute VMs, confine major updates to defined maintenance windows, and much more. Policies can act as firm requirements, advisory warnings, or soft requirements that can be bypassed with explicit approval from your compliance team.
- **Cost Estimation**: Before making changes to infrastructure in the major cloud providers, Terraform Cloud can display an estimate of its total cost, as well as any change in cost caused by the proposed updates.

Terraform Enterprise

- Terraform Enterprise is a self-hosted distribution of Terraform Cloud Application.
- Provides additional security as everything is on-prem.
- It offers enterprises a private instance of the Terraform Cloud application, with no resource limits and with additional enterprise-grade architectural features like audit logging and SAML single sign-on.

Lab: Cloud

1) Open webpage https://app.terraform.io and create an account there

2) Create an Organization by providing name and Email address

3) Create a workspace

Select workflow with VCS (another options like CLI and API driven can also be used)

Integrate your version control system (I have selected Github for this example)

To integrate Github, open link https://github.com/settings/applications/new and provide information been provided by terraform registration page

Setup Oauth authentication as guided by setting up the details

Authorise Terraform cloud to have admin access on your repositories

4) Skip the ssh-keypair as same is not required to setup. SSH-keypair is basically to connect to git repos via ssh, which sometimes required when you have private submodules.

5) Select the repositories where you have your terraform code placed. Don't forget to select the working directory(under advanced section) where you want to work on, especially when you have multiple terraform folders inside the repo. Terraform working directory can be changed lateron from Settings -> General tab.

Lab: Cloud

6) Click on create workspace and finish the creation.

7) Add Terraform Environment Variable and set AWS AK/SK(AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY). Don't forget to select sensitive option on the right side, especially for Secret Access Key.

8) Click on Queue Plan and provide a reason(just any description and better to have though optional) for queuing same.

9) Check the planning logs

10) Apply the changes and look at the output segment of apply logs

11) Cross-check the state under state section post your apply is completed. You can verify your state file outcome, run state and version of the source code etc. You can even download the tfstate file, by clicking on download button.

12) Below add-features can be enabled/disabled/modified:

a) Auto-approve, Remote/Local execution, Terraform version, working directory : Under settings -> General Tab.

b) Manually locking a project. By default Lock is auto-applied during terraform apply. : Under settings -> Locking

c) Email/Webhooks/Slack Notifications - Under Settings -> Notifications

d) Triggering the workspace based on run of another workspace : settings -> Run Triggers

e) Source Code Integration and SSH key setup, can be done via : Settings -> SSH/Version control

Lab: Cloud

Automated pipelining:

13) Change the configuration to auto-approve and do the modification on github under your configuration change and cross-check the terraform apply.