**Terraform plan** — this command shows you what applying do by checking the template, state file, and actual state of the resource. It is recommended to use this before running apply command to ensure accidental deletion of any resources

**terraform validate**— to check the syntax of the file

**terraform fmt**– to do the formatting of the file

**terraform taint** -> marks a single resource for recreation. The resource will be destroyed and then created again.

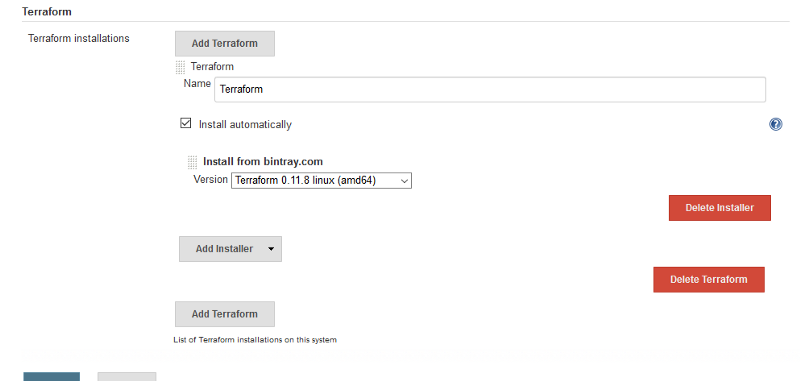
Terraform Jenkins:

First you need to install plugin

Go to Manage Jenkins — Install Plugin — Terraform

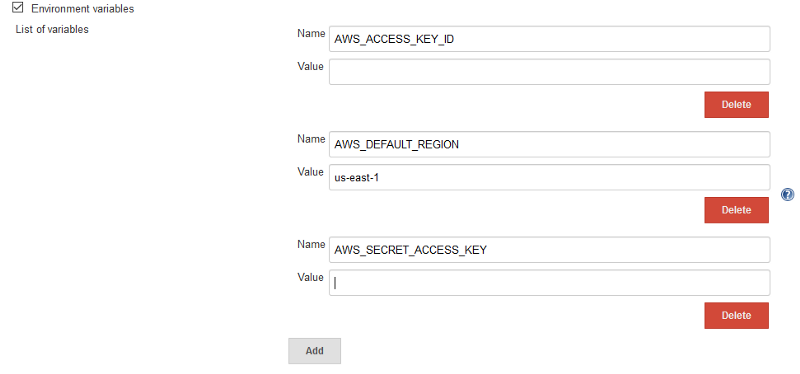
Now configure Terraform

Manage Jenkins ->Global Tool Configuration



To store AWS Secret Key

Manage Jenkins -> Configure System -> Set environment variable



Now create a pipeline job

pipeline {

agent any

stages {

stage(‘checkout’) {

steps {

git branch: ‘develop’, url: ‘[git@](mailto:git@git.kpd-i.com" \t "_blank)your url’

}

}

stage(‘Set Terraform path’) {

steps {

script {

def tfHome = tool name: ‘Terraform’

env.PATH = “${tfHome}:${env.PATH}”

}

sh ‘terraform — version’

}

}

stage(‘Provision infrastructure’) {

steps {

dir(‘dev’)

{

sh ‘terraform init’

sh ‘terraform plan -out=plan’

// sh ‘terraform destroy -auto-approve’

sh ‘terraform apply plan’

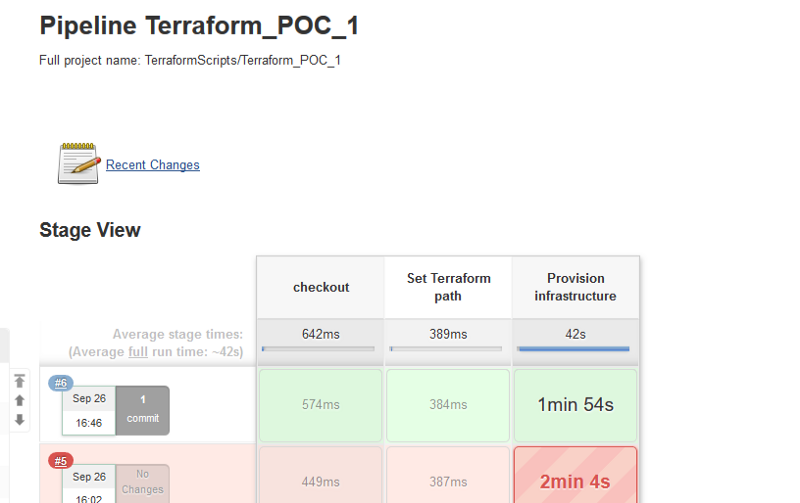
}

}

}

}

}



Terraform variables uses name as default to get the values or uses the Terraform.tfvars

Edit variables.tf:

vi variables.tf

variables.tf contents:

#Define variables

variable "container\_name" {

description = "Name of the container."

default = "blog"

}

variable "image\_name" {

description = "Image for container."

default = "ghost:latest"

}

variable "int\_port" {

description = "Internal port for container."

default = "2368"

}

variable "ext\_port" {

description = "External port for container."

default = "80"

}

Edit main.tf:

vi main.tf

main.tf contents:

# Download the latest Ghost Image

resource "docker\_image" "image\_id" {

name = "${var.image\_name}"

}

# Start the Container

resource "docker\_container" "container\_id" {

name = "${var.container\_name}"

image = "${docker\_image.image\_id.latest}"

ports {

internal = "${var.int\_port}"

external = "${var.ext\_port}"

}

}

Edit outputs.tf:

vi outputs.tf

outputs.tf contents:

#Output the IP Address of the Container

output "ip\_address" {

value = "${docker\_container.container\_id.ip\_address}"

description = "The IP for the container."

}

output "container\_name" {

value = "${docker\_container.container\_id.name}"

description = "The name of the container."

}

Validate the changes:

terraform validate

Plan the changes:

terraform plan -out=tfplan -var container\_name=ghost\_blog

MAPS AND LOOKUPS:

MAPS – keyvalue and pairs, uses type in variables

LOOKUP – Is to search in main.tf to attach the variables

Edit variables.tf:

vi variables.tf

variables.tf contents:

#Define variables

variable "env" {

description = "env: dev or prod"

}

variable "image\_name" {

type = "map"

description = "Image for container."

default = {

dev = "ghost:latest"

prod = "ghost:alpine"

}

}

variable "container\_name" {

type = "map"

description = "Name of the container."

default = {

dev = "blog\_dev"

prod = "blog\_prod"

}

}

variable "int\_port" {

description = "Internal port for container."

default = "2368"

}

variable "ext\_port" {

type = "map"

description = "External port for container."

default = {

dev = "8081"

prod = "80"

}

}

Validate the change:

terraform validate

Edit main.tf:

vi main.tf

main.tf contents:

# Download the latest Ghost Image

resource "docker\_image" "image\_id" {

name = "${lookup(var.image\_name, var.env)}"

}

# Start the Container

resource "docker\_container" "container\_id" {

name = "${lookup(var.container\_name, var.env)}"

image = "${docker\_image.image\_id.latest}"

ports {

internal = "${var.int\_port}"

external = "${lookup(var.ext\_port, var.env)}"

}

}

Plan the dev deploy:

terraform plan -out=tfdev\_plan -var env=dev

Apply the dev plan:

terraform apply tfdev\_plan

Plan the prod deploy:

terraform plan -out=tfprod\_plan -var env=prod

Apply the prod plan:

terraform apply tfprod\_plan

Destroy prod deployment:

terraform destroy -var env=prod -auto-approve

Use environment variables:

export TF\_VAR\_env=prod

Open the Terraform console:

terraform console

Execute a lookup:

lookup(var.ext\_port, var.env)

Exit the console:

unset TF\_VAR\_env

Terraform workspaces:

**Creating a workspace**

Terraform commands:

workspace: New, list, select and delete Terraform workspaces

Workspace subcommands:

delete: Delete a workspace list: List Workspaces new: Create a new workspace select: Select a workspace show: Show the name of the current workspace

Setup the environment:

cd terraform/basics

Create a dev workspace:

terraform workspace new dev

Plan the dev deployment:

terraform plan -out=tfdev\_plan -var env=dev

Apply the dev deployment:

terraform apply tfdev\_plan

Change workspaces:

terraform workspace new prod

Plan the prod deployment:

terraform plan -out=tfprod\_plan -var env=prod

Apply the prod deployment:

terraform apply tfprod\_plan

Select the default workspace:

terraform workspace select default

Find what workspace we are using:

terraform workspace show

Select the dev workspace:

terraform workspace select dev

Destroy the dev deployment:

terraform destroy -var env=dev

Select the prod workspace:

terraform workspace select prod

Destroy the prod deployment:

terraform destroy -var env=prod

Null resources: to perform local operations

In this lesson, we will utilize a Null Resource in order to perform local commands on our machine without having to deploy extra resources.

Setup the environment:

cd terraform/basics

main.tf contents:

# Download the latest Ghost Image

resource "docker\_image" "image\_id" {

name = "${lookup(var.image\_name, var.env)}"

}

# Start the Container

resource "docker\_container" "container\_id" {

name = "${lookup(var.container\_name, var.env)}"

image = "${docker\_image.image\_id.latest}"

ports {

internal = "${var.int\_port}"

external = "${lookup(var.ext\_port, var.env)}"

}

}

resource "null\_resource" "null\_id" {

provisioner "local-exec" {

command = "echo ${docker\_container.container\_id.name}:${docker\_container.container\_id.ip\_address} >> container.txt"

}

}

Reinitialize Terraform:

terraform init

Validate the changes:

terraform validate

Plan the changes:

terraform plan -out=tfplan -var env=dev

Apply the changes:

terraform apply tfplan

View the contents of container.txt:

cat container.txt

Destroy the deployment:

terraform destroy -auto-approve -var env=dev

Modules:

Reuse code like modules from python either internal or external using the source flag…

