Specification Document

For

Automation of infra using terraform

Version 0.0.1

Prepared by Soumyajit Basu

6th July 2020

Table of contents

Introduction **4**

Requirement **4**

Deliverables **4**

Technology **4**

Terraform **4**

AWS (Amazon Web Services) **5**

Terraform workflow **5**

- Terraform init 5
  - Working directory contents 5
- Terraform plan 5
- Terraform apply 6
- Terraform destroy 6

Introduction

The document aims to elaborate the technical specifications for infrastructure automation. The specifications documented here are based on the prototype code to provision infrastructure on **Amazon Web Services.**

The document will mainly consist of the problem statement, the technology and its overview and the desired workflow.

Requirement

The organization is adopting the DevOps methodology and in order to automate provisioning of infrastructure there's a need to set up a centralised server for Jenkins. In order to do it dynamically we would need to dynamically provision an ec2 instance on AWS and install Jenkins and its dependencies accordingly.

Deliverables

- Launch an EC2 instance using Terraform.
- Connect to the instance.
- Install Jenkins, Java and Python in the instance.

Technology

- AWS (using EC2 instance)
- Terraform

Terraform

Terraform is a configuration based language that helps to provision, change and destroy infrastructure on any cloud services. Here we are going to use AWS as our cloud platform. In short we can define terraform as a platform which would deliver infrastructure as code.

There are three steps to working with terraform. The first step is to

1. Write the infrastructure as code using declarative configuration files. HashiCorp Configuration Language (HCL) allows for concise descriptions of resources using blocks, arguments and expressions.
2. Running `terraform plan` to check whether the execution plan for a configuration matches the expectation before provisioning or changing the infrastructure.
3. Apply changes to respective cloud providers with the command `terraform apply` to reach the desired state of configuration.

AWS (Amazon Web Services)

Amazon web services is an on demand cloud computing platform which helps in maintaining infrastructure and services for software development on cloud.

Terraform workflow

Terraform mainly includes 4 respective flows that includes init, plan, apply and destroy.

Terraform init

The terraform init command is used to initialize a working directory containing terraform configuration files. This is the first command that should be run after writing a new terraform configuration file.

Working directory contents

A terraform working directory typically contains:

- A Terraform configuration describing resources Terraform should manage. This configuration is expected to change over time.
- A hidden `.terraform` directory, which Terraform uses to manage cached provider plugins and modules, record which workspace is currently active, and record the last known backend configuration in case it needs to migrate state on the next run. This directory is automatically managed by Terraform, and is created during initialization.
- State data, if the configuration uses the default local backend. This is managed by Terraform in a terraform.tfstate file (if the directory only uses the default workspace) or a `terraform.tfstate.d` directory (if the directory uses multiple workspaces).

Terraform plan

The `terraform plan` command creates an execution plan. By default, creating a plan consists of:

- Reading the current state of any already-existing remote objects to make sure that the Terraform state is up-to-date.
- Comparing the current configuration to the prior state and noting any differences.
- Proposing a set of change actions that should, if applied, make the remote objects match the configuration.

The plan command alone will not actually carry out the proposed changes, and so we can use this command to check whether the proposed changes match what you expected before you apply the changes or share your changes with your team for broader review. If Terraform detects that no changes are needed to resource instances or to root module output values, terraform plan will report that no actions need to be taken.

Terraform apply

The `terraform apply` command executes the actions proposed in a terraform plan. The most straightforward way to use terraform apply is to run it without any arguments at all, in which case it will automatically create a new execution plan (as if you had run terraform plan) and

then prompt you to approve that plan, before taking the indicated actions.
Another way to use terraform apply is to pass it the filename of a saved plan file you created earlier with `terraform plan -out=<file_name>`, in which case Terraform will apply the changes in the plan without any confirmation prompt. This two-step workflow is primarily intended for when running terraform in automation.

Note: The terraform plan and terraform apply can be executed multiple times to configure the right resource objects required.

Once terraform apply is executed this would basically:

1. Provision an ec2 instance based on the `ami`
2. Download dependencies to set up the ec2 machine workable.
3. Install java.
4. Install jenkins.
5. Configure jenkins.
6. Install python.

Terraform destroy

The `terraform destroy` command is a convenient way to destroy all remote objects managed by a particular Terraform configuration.
While you will typically not want to destroy long-lived objects in a production environment, Terraform is sometimes used to manage ephemeral infrastructure for development purposes, in which case you can use `terraform destroy` to conveniently clean up all of those temporary objects once you are finished with your work.

The workflow for terraform to provision and install the required packages is present here.