A simple Sinatra Application deployed with Terraform on AWS

# User-Guide

Requirements

1. An AWS account with sufficient privileges to deploy an EC2 instance.
2. A Centos7 base image on AWS – preferably a baked image with appropriate SSH keys to connect to from a jumphost (if needed).
3. Preferably an SSH client on any OS of your choice with the ability to run Terraform. If you’re using Windows, some choices would be Cygwin, Mobaxterm or Putty if you’re connecting to a remote host.
4. Ensure your ssh client has git installed.
5. Terraform setup – please follow these steps to get your local/remote terraform environment up:
   1. Follow the getting started guide - <https://www.terraform.io/intro/getting-started/install.html>
   2. Download terraform for the OS of your choice - <https://www.terraform.io/downloads.html>
   3. Ensure the terraform binary/executable is on the PATH. The following guides will help you:
      1. Linux - <https://stackoverflow.com/questions/14637979/how-to-permanently-set-path-on-linux-unix>
      2. Windows - <https://stackoverflow.com/questions/1618280/where-can-i-set-path-to-make-exe-on-windows>

What can you expect to see?

1. A fully configured Centos7 EC2 instance with nginx, passenger and the deployed code.
2. The EC2 instance can only be SSH’ed from a jumphost.
3. A security group that’ll allow port 80 access to your ec2 instance from your location.

Step-by-Step Guide to deploying the App

Now that your SSH and Terraform environment has been setup, lets clone the git repository with the terraform code.

1. **Getting the code**

$ git clone <https://github.com/chaosome/iProperty.git>

Alternatively, you can download a zip file with the contents from the github repository

You should see the following files in your newly cloned directory:

$ ls

main.tf README.md terraform.tfvars userdata.tpl vars.tf User\_Guide-Terraform\_Sinatra\_App.docx

The following are descriptions of these files:

Main.tf – the main terraform code that does infrastructure and code deployment.

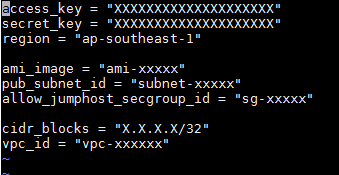
Terraform.tfvars – variables used and its values

Userdata.tpl – userdata to bootstrap the AWS instance with the app deployment prerequisites & code deployment itself.

Vars.tf – variable declaration file.

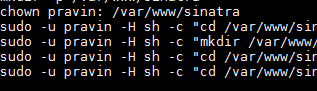
1. **Preparing the environment**

Edit terraform.tfvars and provide all the required IDs, keys etc. for AWS. It is required that your AWS environment is setup with a working VPC, public subnet, a jumphost sec group and a baked image *(this will be covered further in assumptions and design choices)*:



**Note: cidr\_blocks should be the public IP of your location so that you’re able to access the app from your browser.**

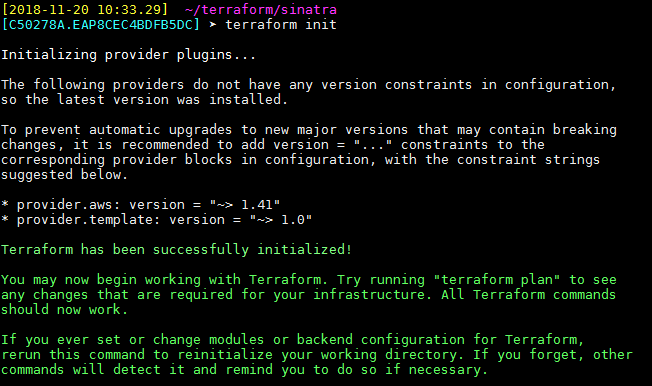
Also ensure that the appropriate local user account is reflected accurately in the userdata file. Here, my account ‘pravin’ is used. Change that to your desired local account:



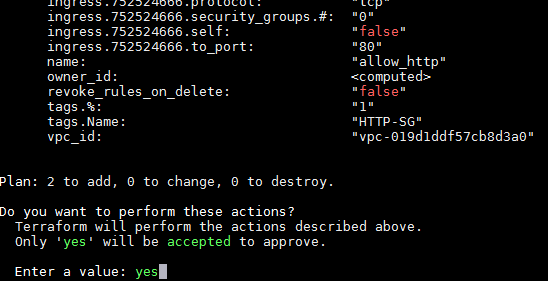
1. **Running Terraform**

Once the above is done, we are ready to run our terraform scripts.

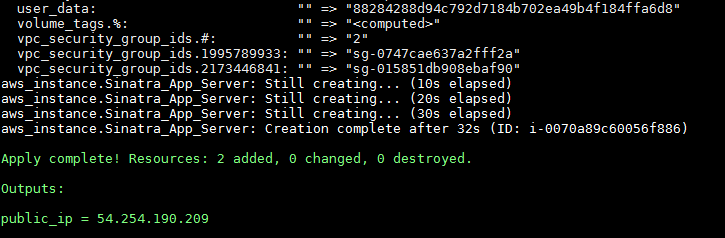
Start by running terraform init in the same directory of the scripts to initialize local settings and gather required plugins for the run. Your desired output should be as follows:



Next, run terraform apply. It will provide you with a plan of what will be deployed and a prompt for you to proceed or abort. To proceed, just type **yes**:



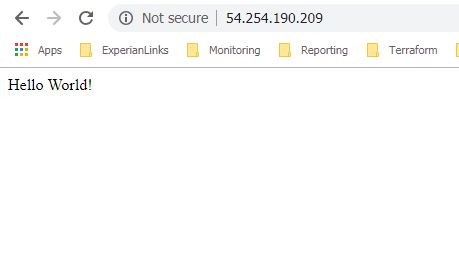
At this point, terraform is on its way to deploying your infrastructure. Once it is done, you will see the following message:



Take note of the public IP. This will be used in the next step to verify if the app is running.

1. **Test your app**

Open your favourite browser, and put in the Public IP that you’ve saved earlier. Your desired output should be as follows:



Your app works!

Assumptions

1. The terraform code expects an AWS environment with required access privileges and a fully working VPC, public subnet & security group for jumphost access.
2. The preparation of a pre-baked Centos 7 image is expected with an initial yum update, creation of a local account and ssh keys for ssh access from a jumphost (only) & disabled SELINUX.
3. A fully setup jumphost is expected to be present for this setup. This is to help connect to the new application server if needed.
4. No requirement of a DNS Name was stated as such, access to the web application will be via public IP.

Design Choices

1. Due to the simplicity of the ask, terraform coupled with AWS UserData were the only two methods chosen to deploy infrastructure and application code. The Terraform Code & UserData written are extremely legible, easy to debug and update if needed. Furthermore, the ‘push-button’ style of deployment employed here allows easy, stress-free deployment for an application of this size.
2. A jumphost was used to not allow direct ssh access to the instance. Being on the public subnet, it makes the instance more vulnerable to attacks.
3. The application root directory was created with a local user account and owned by that same account as a method to increase security of the pages served.
4. Phusion Passenger was used due to its simplicity in deployment and its ability to integrate with the light-weighted Nginx. The deployment could have been presented with Passenger in standalone mode but a better solution is to have a persistent connection to the app, which can only be provided by a web server.

It could be better

1. This method of deployment only gets your application up and running but does not maintain its state. Plus if the complexity of the setup increases, using just terraform and userdata is not preferred. The desired (and my preferred) method would be to provision the infrastructure with terraform and then hand it off to a configuration management tool e.g. puppet to setup the web environment and deploy code. AWS Userdata can be used to install the puppet agent and its config, and from that point on, its puppet’s job to set up the server and maintain its state.
2. Running this setup from scratch (new VPC, subnets, security groups etc.) would have been a great demonstration of a complete push-button solution but not ideal if you’re already running existing infrastructure. A setup from scratch on terraform would radically reduce the need to pre-configure items like VPC-ID, Subnet-ID as they’re computed on the fly.
3. Utilizing the auto-assigned public IP is a great way to save on EIPs but on every restart of your instance, your IP changes. Using something like cli53 allows your instance to dynamically set your latest public IP to your domain name on route53.