**Process flow of the Cloud Provision IaC**

**Prerequisites of setting up a new Cloud Provider to deploy the application.**

* Need to have Terraform installed on the client desktop.
* Need to have ssh installed for remote login to the EC2 instance.
* Create a free-tier AWS account for the deployment using a new email ID and credit card number.
* Could use windows or MacBook desktop for provision of the AWS cloud.

**Infrastructure-as-Code to be deployed for the github application**

In order for Terraform to be able to make changes in your AWS account, you will need to configure the AWS credentials for the user you created earlier. There are several ways to do this (see [A Comprehensive Guide to Authenticating to AWS on the Command Line](https://blog.gruntwork.io/a-comprehensive-guide-to-authenticating-to-aws-on-the-command-line-63656a686799))

Update the Access code and Secret key into the the terraform code inside the variable.tf file

So that new credentials of the root are updated from the new aws account.

The Github repository link for the terraform IaC.

/terraform/

**Network.tf** 🡪 This script helps to create the VPC and access code.

**Routing.tf** 🡪 This script helps to create the route table and IGW

**Subnets.tf** 🡪 this script helps to create the subnet and AZ of network

**Securitygroups.tf** 🡪 This setups the firewall rules and security policy

**Dns-and-dhcp.tf** 🡪 This script setup the DNS zones and network access.

**Ec2-machine.tf** 🡪 This script create the Key pair and EC2 instance for deploying the application packages and codes.

**Variables.tf** 🡪 This are the variable file of the parameters in the terraform files.

**Deployment of the application from the github repository**

# **Run the app**

The REST API application need to be provided with the source code . I am not developer experience, I never do coding for sometime. I have infrastructure & devops experience in CI/CD framework. If you not clear on devops roles we can have a discussion.

**Testing**

PyTest

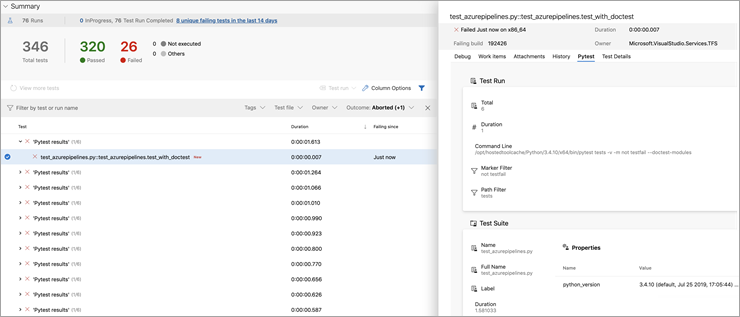
* PyTest is an open-source Python-based testing framework that is generally all-purpose but especially for **Functional and API testing.**
* **Pip**(Package Installer for Python) is required for PyTest installation.
* It supports simple or complex text code to test API, databases, and UIs.
* Simple syntax is helpful for easy test execution.
* Rich plugins and is able to run tests in parallel.
* Can run any specific subset of tests.

**Sample:**

|  |
| --- |
| import pytest                                //Import unittest module//  def test\_file1\_method():               //Function inside class//        x=5        y=6        assert x+1 == y,"test failed" |
|  |

To run the test use the ***py.test***command.

**Screenshot for Reference:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2019/12/PyTest-1.png)

*[image*[*source*](http://github.com/)*]*

**Packages/Methods:**

| **Function** | **Parameters** | **Working** |
| --- | --- | --- |
| **pytest.approx()** | expected, rel=None, abs=None, nan\_ok=False | Assert that two numbers or two sets of numbers are approximately equal to some differences. |
| **pytest.fail()** | msg (str) pytrace(bool) | If the executing test fails explicitly the message is shown. |
| **pytest.skip()** | allow\_module\_level(bool) | Skip the executing test with the message shown. |
| **pytest.exit()** | msg (str) returncode (int) | Exit testing process. |
| **pytest.main()** | args=None plugins=None | Return exit code once in-process test execution is done. |
| **pytest.raises()** | expected\_exception: Expectation[, match] | Assert that a code block call raises expected\_exception or to raise a failure exception |
| **pytest.warns()** | expected\_warning: Expectation[, match] | Asserting warning with the functions |

If you want to access a test written in a specific file we use the below command.

|  |
| --- |
| py.test <filename> |

**Pytest Fixture:**Pytest Fixture is used to run code before executing the test method to avoid code repetition. This is basically used to initialize database connection.

You can define PyTest fixture as shown below.

|  |
| --- |
| @pytest.fixture |

**Assertion:**Assertion is the condition that returns true or false. Test execution stops when the assertion fails.

**Given below is an test:**

|  |
| --- |
| def test\_string\_equal():  assert double(55) == 62  assert 25 == 62  +  where 25 = double(55) |

Download Link**:**[Pytest](https://pypi.org/project/pytest/" \t "_blank)

**Containers**

I have created the dockfile. The dockerfile have been created for the posgresql and python application in the alpine image. Pls review it

**CI**

I have uploaded the gitlab-ci.yml for the CI to be run in the gitlab website.

**Execution Process of Terraform.**

There are three commands which are pretty much required to provision the infrastructure using Terraform.

* $ terraform init
* $ terraform plan
* $ terraform apply

**Note : AWS RDS services is not created in the terraform scripts since Free tier Account charges for the RDS services and testing. Since charges are involve, loaded PostgreSQL in the EC2 instance itself to do the testing and demo**.