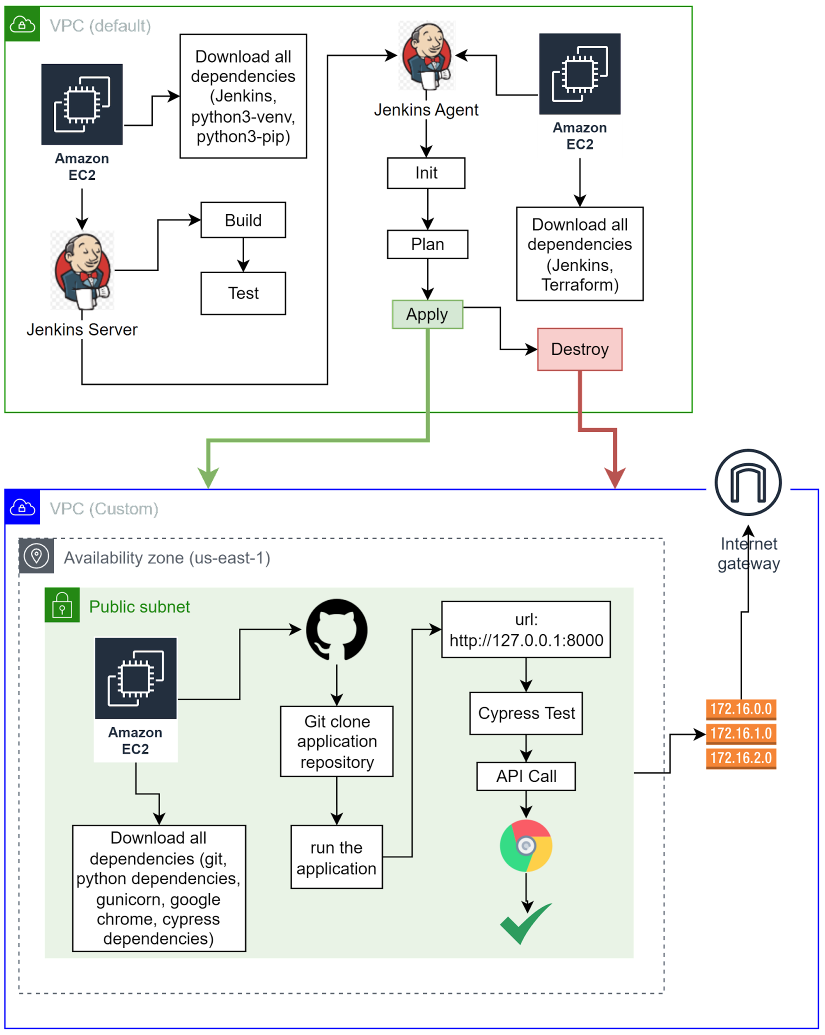
[Deployment 4 Documentation]

Author: Suborna Nath
Date: November 6, 2022

Description: In this deployment, we will be configuring a VPC with Terraform and deploying a flask application with the configured files.

Diagram



VPC (default)
Amazon EC2
Download all dependencies (Jenkins, python3-venv, python3-pip)
Jenkins Server
Build
Test
Jenkins Agent
Init
Plan
Apply
Amazon EC2
Download all dependencies (Jenkins, Terraform)
Destroy
VPC (Custom)
Availability zone (us-east-1)
Public subnet
Amazon EC2
Download all dependencies (git, python dependencies, gunicorn, google chrome, cypress dependencies)
Git clone application repository
run the application
url: http://127.0.0.1:8000
Cypress Test
API Call
Internet gateway
172.16.0.0
172.16.1.0
172.16.2.0

Initial Setup: Download dependencies

At first, we create an EC2 instance and ssh into it with the proper .pem file. In the EC2, we download all the dependencies such as Jenkins, python3-pip and python3-venv. Next, we open up the Jenkins server on the browser. In the server, we configure the AWS keys (secret key, access key) needed to run the terraform files successfully. We also configure a Jenkins agent which we will use to run terraform, stages 4-7.

Troubleshoot: We needed the Jenkins agent because the pipeline was getting stuck. So, we divided the stages between two EC2 so that we are not maxing out the computing power of our instances and causing the system to slow down.

Pipeline Walkthrough

1. Declarative: Checkout SCM

- In this initial stage, Jenkins will access the GitHub repo and pull the latest version of the
  source code. It will add the project files to the Jenkins workspace.

2. Build

- Python environment is getting created and activated
- Latest version of pip is installed
- Installing all the required packages from requirements.txt
- Setting the application name "application.py" as an environment variable
- Running the Flask application

3. Test

- Activating the python environment
- Running the pytests written in ./test_app.py file and saving the results as results.xml
    - ➔ Testing the homepage to make sure we can access it
    - ➔ Testing an invalid URL to make sure it gives a 404 error
    - ➔ Testing a saved URL to make sure it redirects the request with a 302 status code

4. Init

In the terraform directory (intTerraform), we are initializing the terraform environment and downloading all the dependencies. We are creating a terraform environment.

5. Plan

In this stage, Terraform is going over the steps that are planned out in the terraform files and saves them. In this case, we are creating a custom VPC with CIDR block 172.19.0.0/16. Inside the VPC, we will create a public subnet that will run an EC2 instance. In the EC2 instance, we will be deploying our Flask application "url-shortner" and performing a cypress test. The test will check if we have the correct title for the website. The website will be hosted with Gunicorn and can be accessed with the public IP of the EC2 at port 8000.

When creating the EC2 with Terraform, it will be reading a .sh file and running the commands inside it.

6. Apply

This stage will be creating all the resources mentioned in the terraform files as described in the previous stage. When creating the EC2, it will be running a few bash commands to set up the environment for the Flask application. It will be downloading some dependencies including Gunicorn, git, python3-pip, and google chrome. Inside the EC2, we will clone a GitHub repository that has all the files for the flask app to run. From there, we will also download cypress test dependencies and dependencies from requirements.txt.

7. Destroy

In this stage, we will destroy all the resources created with Terraform which includes the VPC, EC2 instance, public subnet, internet gateway and the route table.

What can be improved?
The pipeline can have a separate stage for the cypress test. I can also set up a notification system to go off if any stage of the pipeline fails and also if the pipeline runs successfully, the email can send a confirmation for that as well. I can also add a backend for the app and set it up in a private subnet.