kura labs

COHORT 3

Deployment #3

Kerri Smith

15th October 2022

Table of Contents

Overview

This deployment exercise demonstrated the steps for setting up a basic CI/CD pipeline deployment to a custom AWS VPC.

Pipeline deployment

The software application used in this case was a Flask web application called "url shortener"

GitHub was used to manage the code and Jenkins was used to automate the following stages:

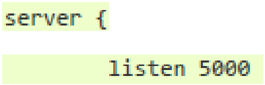
1. Build
2. Test
3. Clean
4. Deploy

Issues:

There were issues encountered in the initial deployment due to:

1. The configuration of the nginx server "/etc/nginx/sites-enabled/default" file.

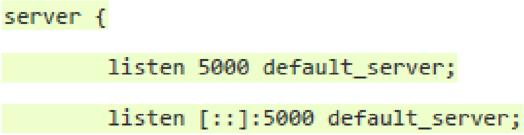
   initial instructions:

   ```
   server {
           listen 5000
   ```

   The errors encountered included requesting browser not being able to access dependencies in subfolder (e.g. css files and js files)

   correction:

   ```
   server {
           listen 5000 default_server;
           listen [::]:5000 default_server;
   ```

   (See Deployment_3-Assignment page 11)

2. The script in the Jenkins file did not successfully allow the application to continue running at the end of a "successful" deployment.
   Correction: download the Jenkins plugin "Pipeline Keep Running Step"
   Use revised script in Jenkinsfile: (See Deployment_3-Assignment page 12)

VPC

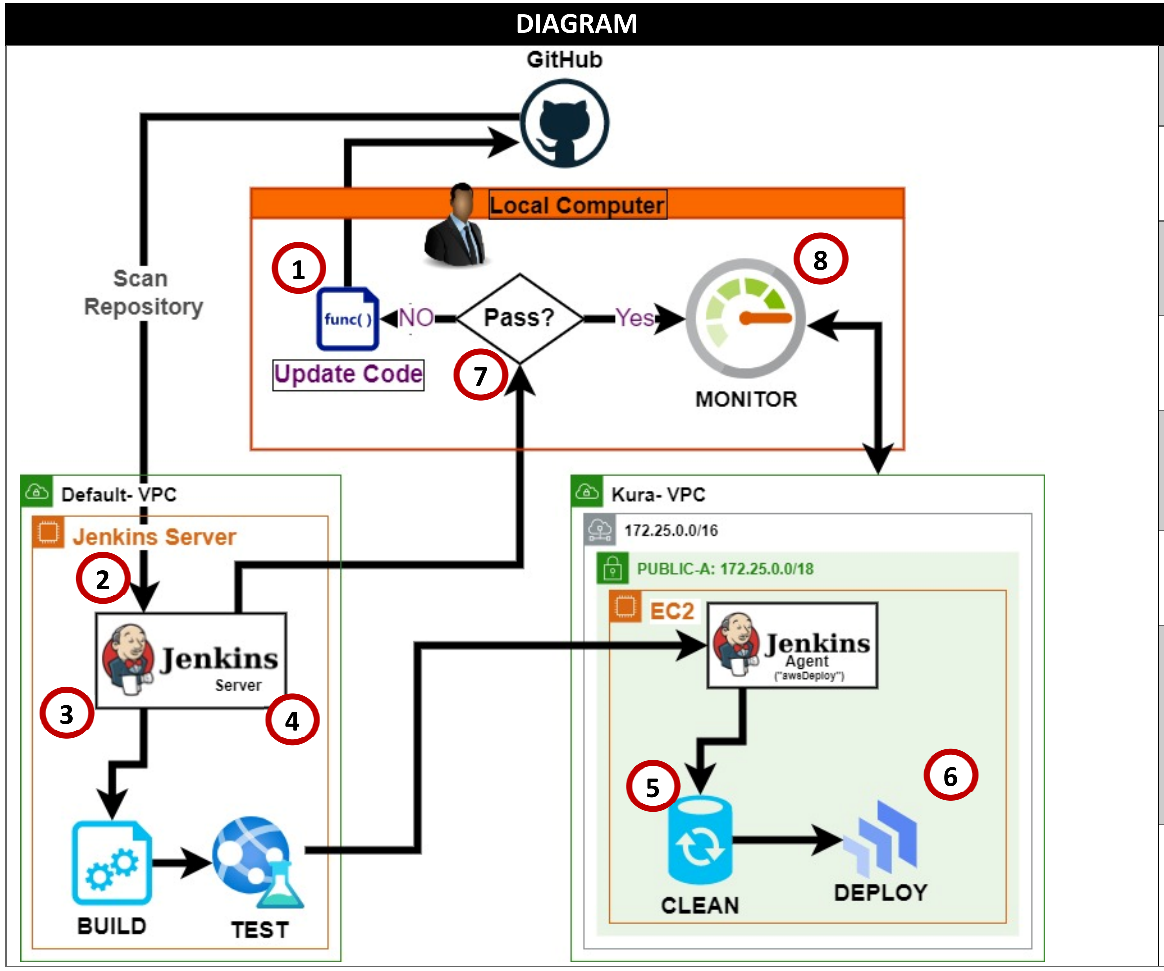
The deployment environment used two Amazon Virtual Private Clouds (Amazon VPCs):

1. The default VPC created with the AWS account
2. A custom VPC called "Kura-VPC". This VPC consisted of
   a. Two availability zones
   b. Two private subnets and one public subnet
   c. An internet gateway
   d. One EC2 which was used to run the Jenkins agent and deploy the application

Issues:

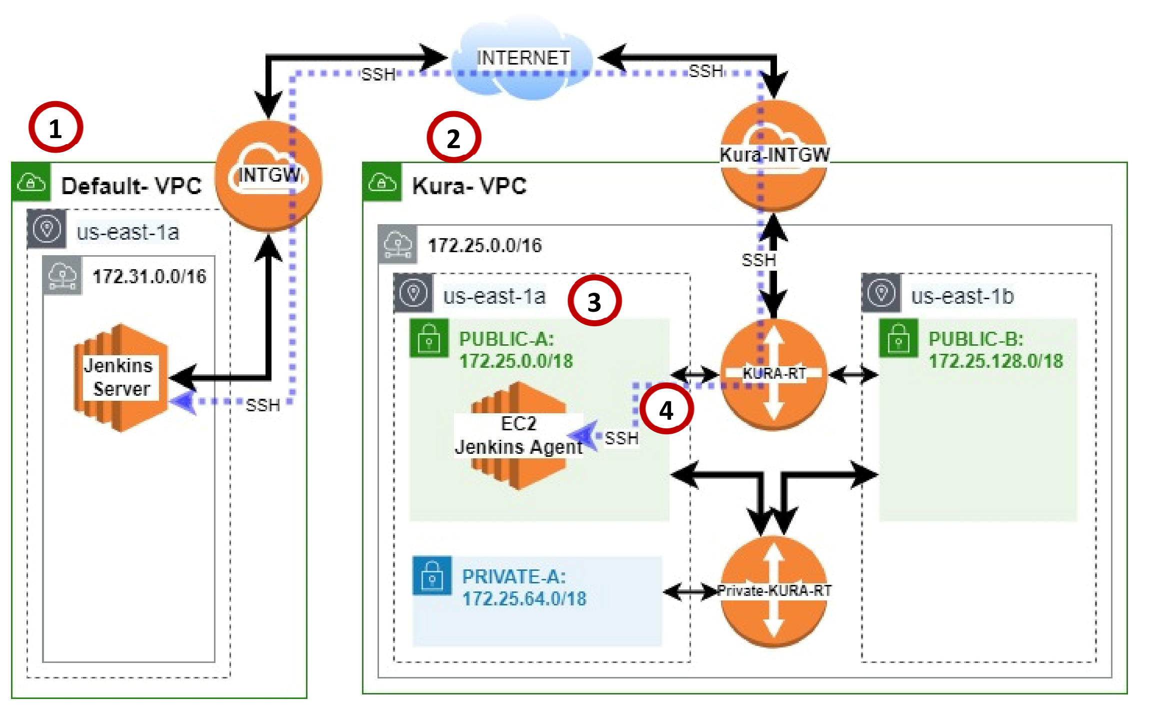
Extended time was used running EC2 to trouble shoot initial issues with deployment. This resulted in the AWS free tier being exceeded. The revised deployment instruction were simulated using Virtual Machines and Virtual Nework on a local computer.

Pipeline

DIAGRAM	STEP	DESCRIPTION
	1	The code is update by the developer and the changes are committed and pushed to GitHub
	2	The Jenkins server scans the repository and retrieves the files
	3	The Jenkins server builds the application. If the build passes, we progress to step **4**. If it fails, we move to step **7**.
	4	The Jenkins server tests the application. If the test passes, we progress to step 5. If it fails, we move to step **7**.
	5	The results of the build and test stages are reported in the Jenkins server. If the clean stage passes, we progress to step **6**. If it fails, we move to step **7**.
	6	The application is deployed into the virtual environment on the EC2 in the custom VPC. We move to step **7**.
	7	If all stages are passed, then we proceed to step 8. If there are errors the error logs and reports are reviewed to identify and isolate the issue. Any code related issued including errors in the Jenkins file are corrected and the updates committed and pushed to GitHub i.e. we proceed to step 1.
	8	The deployed application is monitored

VPC

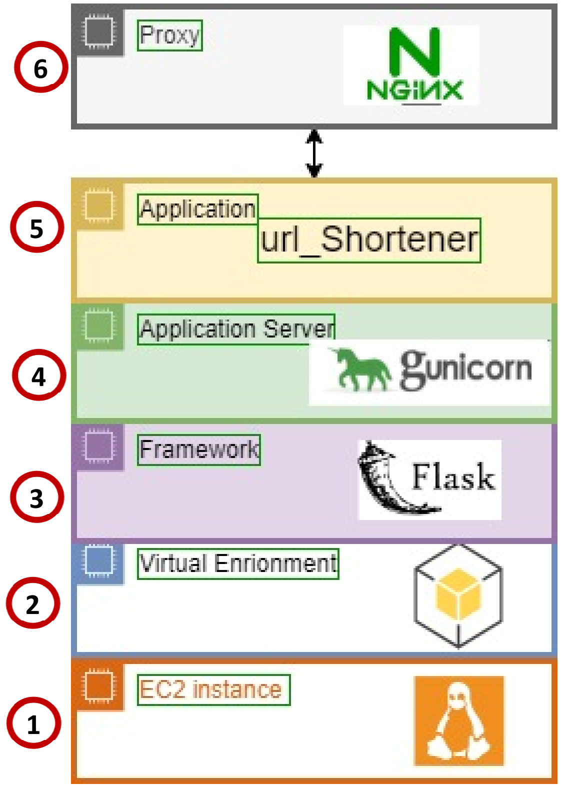
The deployment environment used two Amazon Virtual Private Clouds (Amazon VPCs): The default VPC created with the AWS account and a custom VPC called "Kura-VPC"

DIAGRAM	ITEM	DESCRIPTION
	1	Default- VPC: The default VPC contained an EC2 with a public IP address running the Jenkins server
	2	Kura-VPC The Kura-VPC contained three subnets. One two public subnets (PUBLIC-A, PUBLIC-B) in two availability zones and one private subnet. For this deployment only one EC2 with a public IP address in public Subnet PUBLIC-A was used
	3	Jenkins Agent The Jenkins agent was deployed on the EC2 in the Kura-VPC
	4	SSH connection was used by the Jenkins server to set up the Jenkins Agent on the EC2 in Kura-VPC

STACK

A software stack consists of independent software components that work together to support the execution of an application. These components can include operating system(s), runtime environments, databases, etc. These components tend to function hierarchically with lower-level components enabling or supporting the functionality of higher-level components.

The following diagram describes the software stack used for this deployment

DIAGRAM	ITEM	DESCRIPTION
 6 Proxy NGiNX 5 Application url_Shortener 4 Application Server Gunicorn 3 Framework Flask 2 Virtual Enrionment 1 EC2 instance	1	This is the base operating systems that provides access to processing and memory resources to the all of the other elements of the stack
	2	The PIP virtual environment provides another layer of abstraction from the operating system for the remaining levels of the stack
	3	Flask provided the web framework. This means flask provides the tools and libraries that allow the web application to function.
	4	Gunicorn is the web application server that is used to allow the web application to run
	5	This the code that is running to serve the url shortener web application
	6	Nginx provided a reverse proxy. It shields or masks the internal web application server from interacting directly with the user. Users of the web applications send their requests to the nginx ip address and port, nginx then communicates internally with the web application server and relays the response to the end user.

Proposed Improvements

The pipeline could be improved in the following ways:

1. Include automated monitoring and reporting after deployment
2. Include webhooks to automate deployment of updated code