**DevOps Code Challenge**

**for**

**DevOps Position at Neonomics**

**V1.0**

|  |  |
| --- | --- |
| Date | **11-02-2021** |
| Number of Pages | **5** |
| Author | **Osman Elamin** |

**Introduction**

This document is to document the solution for DevOps Code Challenge as part of Neonomics’s interview process , it supposed to give full insight of how the solution has been design and work with explanation for how to operate it and how different parts works together .

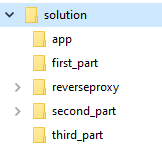
Assumptions:

Solution is design and worked with the following assumption:

* Multiple ready computing machines (VMs/EC2…etc).
* SSH key authenticated on all machines.
* Ubuntu 18.04+ operating System.
* Python installed on all machines
* Not all parts are mandatory.
* Same Containers and Apps to be used on all Tasks .

**Solution Folder Structure:**

The solution is produced on Task structural way with consideration of shared files to be on root directory, as showd:



**App Folder** : contains Container B documents.

**Reverseproxy** Folder: contains Container B documents.

**First\_part**: contains documents related to First Part.

**Second\_part**: contains documents related to Second Part.

**Third\_part**: contains documents related to Third Part.

Solution is published on my personal repository on github with the following link:

<https://github.com/osmansays/no_challenge>

*First Part Task*

*Docker File*

Task:

Create a Docker image:

* Use your favorite base image but keep it as light as possible.
* Install a web server in container A and another app/service (of your
* Choice) in container B.
* Add https support.
* Export some basic Prometheus metrics for both containers.
* Make container A to send requests to container B and receive and output.
* Once you wrote the Dockerfile and created an image, run the container and test the application.
* Describe how did you test it and provide output.
* Describe one or more weaknesses of your Dockerfile. Is it ready to be

used in production?.

**Solution:**

Docker compose is used to resolve solve this task to resolve services connectivity and dependency , with the following:

Container A (App/Service):

* Used custom developed python web application with Flask to generate random number with each request , and built the Docker file based on alpine base image for smallest size possible, files located on app folder ,with the following consideration:
  + config Folder: contains nginx configuration that will be copied to image during the build process.
  + Ssl Folder: contains self-signed SSL certificates that will be copied to image during the build process to fulfill https support request.
  + Dockerfile .
  + Configured to run and exposes on Port 5000

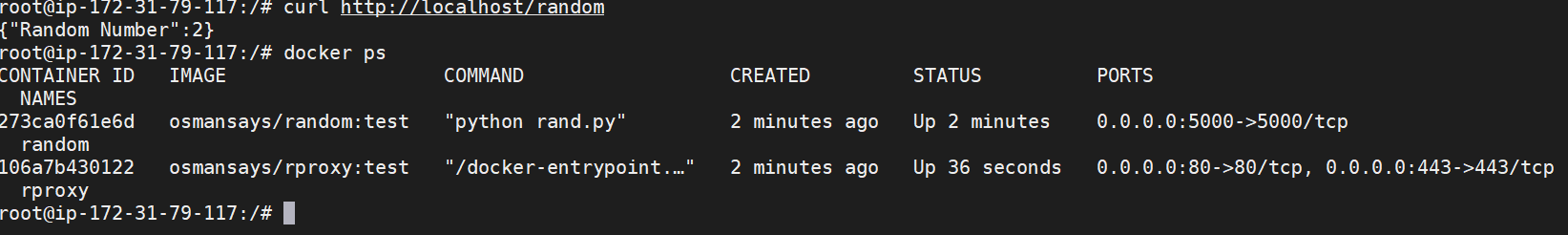
Container B (Proxy Server/Web Server):

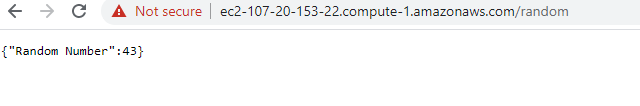
* Used nginx , and built the Docker file based on alpine base image for smallest size possible, files located on reverseproxy folder , with the following consideration::
  + rand.py: contains python source code to run the service and application.
  + rand.test.py: contains App’s unit test case.
  + requirements.txt : contains application dependency package .
  + Dockerfile .
  + Configured to run and exposes on Port 80

To operate and deploy the solution , just run deploy.sh located on first\_part folder witch will use docker-compose file located on same place to deploy the solution , images are pushed to my personal public repository on Ducker Hub taged with test.

**How to test:**

To test the solution , simple http/https request should be done on web server container through host IP/DNS on port 80 with sub directoy /random,(i.g <http://localhost/random>) .The request can be through browser or simple curl command.





* **Describe one or more weaknesses of your Dockerfile. Is it ready to be**

**used in production?.**

This solution is not meant for production , for the following reasons:

All containers are running on single server , with no orchestration or availability/performance guaranteed.

Use of self-signed SSL certificate.

All services and ports are accessible to users , any user can reach the app solution directly with going through reverse proxy .

*First Part Task*

*Automation, Kubernetes and Docker*

Task:

Use the previous images for this part Create the YAML files required for deploying the pods in a kubernetes cluster. You can use Helm or Kubernetes manifests directly.

Write and run a script using your favourite tool (Terraform, Ansible, CloudFormation...) which will install Docker, a Kubernetes cluster and kubectl and then create a deployment in your cluster with your app running.

**Solution:**

Ansible is used here for the automation , where all the tasks for the playbook resides on second\_part\k8sDeployment.yml file , The playbook installs latest version of Docker , then Single Node for Kubernetes using Kubeadm and enable the scheduling for this node as single node cluster.

fullDeploymentOn.sh is used for this deployment and it has to get Server IP address /DNS as passed argument in the following format ./fullDeploymentOn.sh {servername} (i.e . fullDeploymentOn.sh k8server ).

Kubernetes deployment files located on second\_part\solutions-deployment-yml directory, one for service deployment and another for deployment where it’s deploy mulitple containers Pod for the web server/proxy to work as an ambassador with the external word.

Kubectl is install on remote server and on local server and configuration file is copied through ssh to ensure kubectl is working on operator machine to deploy yaml files representing the deployement and services.

*First Part Task*

*Pipeline, Jenkins & CI/CD*

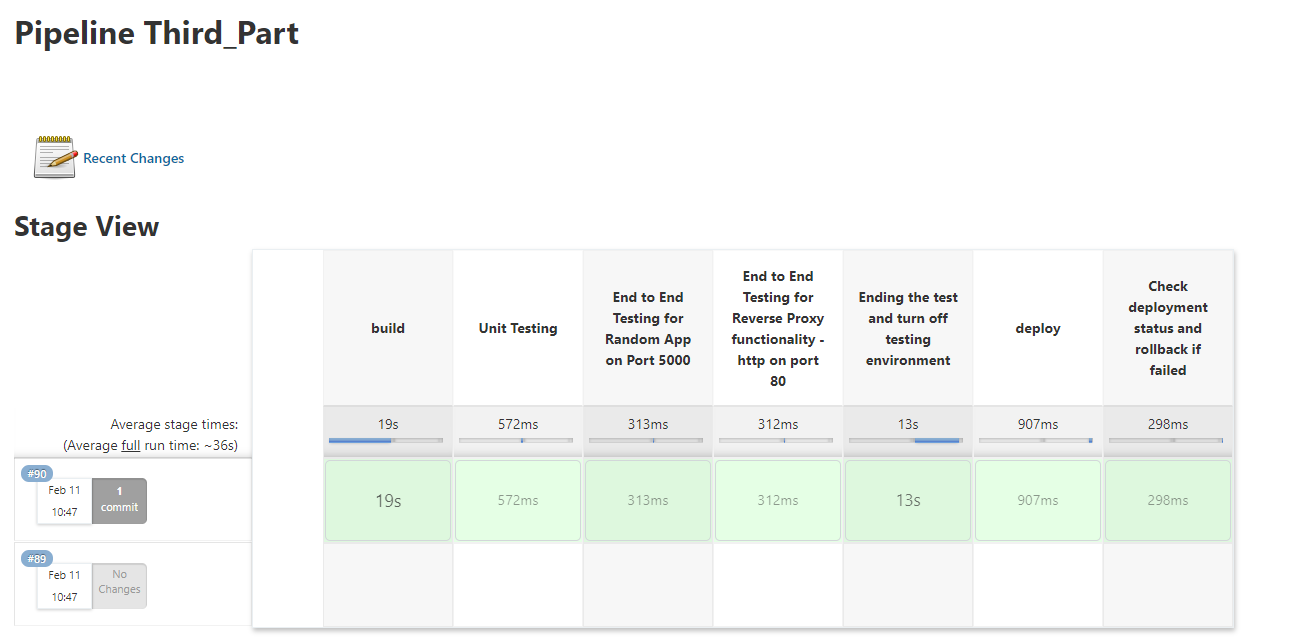
Task:

A pipeline that will test & deploy your previous containers Runs a security scan to both containers Rolls back the deployment if the pod is not in good health

**Solution:**

Jenkins is used to perform this task and the Jenkinsfile is located on third\_part\Jenkinsfile ,

The result are as showed below:

****