Assignment-1

Building the Application

Assignment Scope and definitions:

Write configuration-as-code recipes (using your preferred orchestration software) to:

* Create the server (can be local VM or AWS based)
* Configure an OS image (your choice) appropriately.
* Deploy the provided application.
* Make the application available on port 80.
* Ensure that the server is locked down and secure.

Prerequisites:

* This application is built on Ruby platform so ruby and gem has to be installed as prerequisites.
* Docker is our orchestration tool and hence docker has to be installed, commands provided in the document.
* The prerequisites will be installed as part of ***Dockerfile*** so no additional softwares are required.

Environment setup and Assumptions:

* AWS EC2 Instance has to be configured with AMI as Ubuntu 16.04 or 18.04 LTS version.
* EC2 Type: t2.micro :: 1 vCPU and 1 GB of RAM
* Docker Version: 18.0 or 19.0 either will work.
* Create locked down and secure Instance:
  + By configure in-bound rule in security group and add in-bound port number 80 and do not put all-all so it must not be 0.0.0.0/0
  + Configure security groups to permit the minimum required network traffic for the EC2 instance.
  + Create a VPC and IAM role and define the policy to restrict IAM user.
  + This policy restricts an IAM user or group access to only Start/Stop/Reboot EC2 instances in that particular region.
  + Note: Replace the Owner, Bob, and [AWS Region](https://docs.aws.amazon.com/general/latest/gr/rande.html#endpoint-tables) with parameters from your environment.
  + Finally, create similar policies for each group of IAM users, using a different Region for each one.

Solution Approach:

* We write configuration-as-code recipes by using Docker as preferred orchestration software.
* *Dockerfile* is used and with just few instructions in *Dockerfile* will install Nginx, Ruby and other dependencies and at the end it will create image for sinatra-app.
* This *Dockerfile* will be deployed further by Kubernetes using kubectl commands and steps are mentioned in Deployment Document (ie assignment-2-doc.)

Brief Introduction about Containers:

**Containerized development environments** are easier to set up than traditional development environments, once you learn how to build images as discussed below. This is because a containerized development environment will isolate all the dependencies your app needs inside your Docker image; there’s no need to install anything other than Docker on your development machine. In this way, you can easily develop applications for different stacks without changing anything on your development machine.

Steps to complete:

* Login AWS EC2 instance by using putty or mobaxterm client
* After login change the default user to root user for docker installation, use command as:

**“sudo su -“**

* Start docker services by command: **“service docker start”**
* Install docker and here below are commands for docker installation

**“sudo apt-get update && apt-get install docker.io -y”**

* Verify the installation by running docker command:

“docker run hello-world” # You will display – ***‘Hello from Docker!’***

Git: by using Git, we can clone the given example project from GitHub:

git clone <https://github.com/rea-cruitment/simple-sinatra-app>

Define a container with Dockerfile

Take a look at the file called Dockerfile defined below. Dockerfile describe how to assemble a private filesystem for a container, and can also contain some metadata describing how to run a container based on this image.

The *Dockerfile* looks like this:

# Use the official image as a parent image

FROM ubuntu:latest

# Set the working directory

WORKDIR /usr/src/app

# Copy the file from your host to your current location of container

COPY gem /usr/src/ruby/

# Run the command to install Ruby and Nginx both by this single command.

RUN apt-get update && apt-get install ruby -y && apt-get install nginx -y

# Run the gem command to install Sinatra App

RUN gem install sinatra

# Inform Docker that the container is listening on the specified port at runtime.

EXPOSE 80

# Copy the rest of your app's source code from your host to your image filesystem.

COPY . .

Build and test your image

Now that we have a Dockerfile, it’s time to build our image

docker build --tag my-sinatra-app . # The dot represents current path

docker images # This will list the image

You’ll see Docker step through each instruction in your Dockerfile, building up your image as it goes. If successful, the build process should end with a message Successfully tagged my-sinatra:1.0.

Run your image as a container

1. Start a container based on your new image:
2. docker run --publish 80:80 --detach --name sinatra-app my-sinatra-app

There are a couple of common flags here:

* + --publish asks Docker to forward traffic incoming on the host’s port 80, to the container’s port 80. Containers have their own private set of ports, so if you want to reach one from the network, you have to forward traffic to it in this way. Otherwise, firewall rules will prevent all network traffic from reaching your container, as a default security posture.
  + --detach asks Docker to run this container in the background.
  + --name specifies a name with which you can refer to your container in subsequent commands, in this case sinatra-app.

1. docker ps # To check that container is running