1 Overview

You have developed one of the finest image classification neural network and would like to provide this as an online service to others. In order to do so, you will be setting up and managing the infrastructure using Kubernetes and Docker containers over a cluster of nodes.

2 Requirements

You will need an AWS account with some free credit available and will be working on EKS, EC2, Kubernetes and Docker. Also, you need to be familiar with one of the following programming languages for implementing your server and interacting with kubernetes: Python / Javascript / Java / Go. While we will make every attempt to help out irrespective of your chosen language, we can best assist with python.

3 Procedure

3.1 AWS EKS / Kubernetes Setup

You need to first spawn up an EC-2 (t2.micro) instance through which you will be managing your EKS. While it is possible to setup EKS

your own system, you will be using this EC2 instance later on for integrating a server with EKS. From the EC2 instance, setup the AWS EKS following their official documentation:

https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html

While setting up using eksctl, you will want to select "Cluster with Linux-only workloads" with a total number of 2 nodes of type t3.medium. You have now allocated hardware resources to start your service.

3.2 Containerization

We have provided the image classification neural network implementation with two versions - (classify.py). The first one is a simple feed forward neural network which you will use to market your product providing it as a free service. The second one is your premium service - neural network built using convolution. The provided implementation retrieves the dataset from the internet, train the model and then perform testing / classification on it. The 'DATASET' which could be *mnist* or *kmnist* and the 'TYPE' *ff* or *cnv* is passed using environment variables. You only need to be aware of how to run classify.py and do not need to worry about its implementation.

In order to run this on your cluster, you will first have to dockerize them. Install docker on the EKS EC2 instance and create a docker image. The environment variables will have to be passed through the Dockerfile. We have also provided requirements.txt, which specifies the python packages that the docker needs. Once the images are created, you will want to deploy it to a docker container and test if it works.

https://docs.docker.com/get-started/

The expected output of successfully running a docker container is similar to the following:

```
dataset: mnist

Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz

to ./data/MNIST/raw/train-images-idx3-ubyte.gz

100.1%Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to

./data/MNIST/raw

...

Processing...

Done!

Epoch [1/5], Step [100/600], Loss: 0.2846

Epoch [1/5], Step [200/600], Loss: 0.3859

Epoch [1/5], Step [300/600], Loss: 0.1448

...

Accuracy of the network on the 10K test images: 98 %
```

classify.py

requirements.txt

3.3 Deploying to cluster

Now that the container images are ready, it is time to deploy it to your kubernetes server. First of all, you will need to create a repository for your image on Docker Hub. Then push

your image to your repository so that nodes in kubernetes are able to pull the image. You can follow the folllowing documentation:

https://docs.docker.com/docker-hub/repos/

https://docs.docker.com/engine/reference/commandline/push/

Then you need to create the configuration file and then use the kubectl command line interface to deploy it to one of the cluster nodes. Each container should execute in its own kubernetes pod. You will be using Job as a kubernetes resource type as it runs to completion which is suited for our use-case.

You can follow the official document of Kubernetes to learn how to deploy your image:

https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/

https://kubernetes.io/docs/tasks/inject-data-application/define-environment-variable-container/

You will have to configure the environment variables in your yaml file along with the Dockerfile.

3.4 Resource Provisioning

As you will be exposing both free and premium services, it is important that you provision your cluster appropriately i.e you do not want the free service to take up all of the hardware resources. In order to do so, you will provision atmost 2 CPU out of total 4 (t3.medium has 2 cores) for your free service. This will be done using kubernetes namespace which allows you to create a virtual cluster. Each of the pods should request and have a limit of 0.9 CPU. This will ensure that at a time only two free service pods are running and the other free service pods are queued up however there can be any number of premium pods running. Premium service pods should execute under "default" namespace and the free ones under "free-service".

https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/quota-memory-cpu-namespace/

3.5 Exposing Image Classification

Now that the cluster is ready to be utilized, you will build a server that will accept requests for image classification and for config as needed by the autograder to verify your

implementation. The server will have to be implemented on the EC2 instance that you used to setup the EKS as that specific EC2 instance has the the kubernetes configuration. You can reuse the server code from MP-1 if you'd like however the main part is programmatically interacting with kubernetes which can be done using one of the following client libraries:

https://kubernetes.io/docs/reference/using-api/client-libraries/

You will be implementing the following API's:

1. Free Service - Create a container with the feed forward neural network within the free service namespace

- HTTP POST: /img-classification/free
- BODY: {"dataset":"mnist"}
- Response status code should be 200 if the pod has been successfully created. Based on your namespace configuration, at a time only 2 free service pods will be running.

2. Premium Service - Create a container with the convolutional neural network within the default namespace

- HTTP POST: /img-classification/premium
- BODY: {"dataset":"kmnist"}
- Response status code should be 200. You are hypothetically getting paid for this service.

3. Configuration - Sends a snapshot of the current outlook of your kubernetes cluster

- HTTP GET: /config
- BODY: {"pods": [{"node": "ip-192-168-123-7.ec2.internal", "ip": "192.168.125.2", "namespace": "default", "name": "mnist-deployment-6dc644dd6d-grpfd", "status": "Succeeded"}, {"node": "ip-192-168-133-144.ec2.internal", "ip": "192.168.133.144", "namespace": "kube-system", "name": "aws-node-hjrmj", "status": "Running" }....] }
- Response status code should always be 200. The body contains all of the pods including the ones created by the kubernetes that are currently executing with some of its specifications. The following are the specifications:

{"node" : "node on which the pod is executing", "ip" : "ip address of the pod", "namespace" : "namespace of the node", "name" : "name of the pod", "status":"status of the pod"}

4 Final Result / Submission

Virtualization helped us provision our hardware resources effectively and achieve isolation between different executions. Moreover, we were able to easily deploy our neural network in

any of our nodes without having to worry about the environment, dependencies etc which was simulated to be the same through containers.

Now that your service is ready, it is time to submit it to autograder. Please ensure that you do not have any additional pods running than the kubernetes ones before submitting. Add the necessary info in the payload section in test.py file and execute the program. It takes about ~20s to run all of the test cases on your infrastructureIf all the test cases pass then you will be able to see your grade on coursera.

NOTE: Please make sure that you delete all your old jobs before before running test.py everytime. Please use region -> us-east-1 for your deployment.

test.py