MP-3 Guide

Acknowledgements

This guide has been developed with the help of the students contribution through their Piazza posts notably RKR, Pradeep Sharma and Eric Ellwanger.

https://piazza.com/class/k5k5ta49mzz4v5?cid=320

3.1 AWS EKS / Kubernetes Setup

You need to use your personal AWS account to setup the EKS as educate accounts do not support this. Here are some of the steps:

1. Create a new EC2 instance (t2.micro / t3.medium) - master node. Some students have faced issues with t2.micro due to resource constraints and thus to be on the safer side, t3.medium will be preferable for this step.
2. Follow the instructions and setup eksctl from this EC2 command line. You need to select 2 t3.medium for the worker nodes and "Cluster with Linux-only workloads": https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html
3. Verify if your EKS / Kubernetes cluster is up and running. You can use the following commands as reference:
    a. https://eksctl.io/
    b. https://kubernetes.io/docs/reference/kubectl/cheatsheet/

3.2 Containerization

All of the following steps need to be performed from the master node:

1. Move the python files (classify.py & requirements.txt) over to the master node.
2. Install docker on the master EC2 node:

```
$ sudo yum install -y docker
```

$ sudo service docker start

3. Create your Dockerfile. The dockerfile should include the environment variables. The following link should help: https://runnable.com/docker/python/dockerize-your-python-application

   If you get errors related to pytorch during the build phase, try to use the cpu version. Change this inside requirements.txt:

   torchvision=0.5.0+cpu

4. Try to build and run the image: https://docs.docker.com/get-started/part2/
   There is no need to deploy the image to the docker hub. Once you build, it gets deployed to your local docker repository which is accessible by both Docker and Kubernetes.

3.3 Deploying to cluster

1. You need to first write the job.yaml file in order to be able to deploy the application to kubernetes. The following link should help: https://kubernetes.io/docs/concepts/workloads/controllers/jobs-run-to-completion/
2. If the yaml file is correctly created, you can deploy it to kubernetes and verify using the following command: $ kubectl get pods --all-namespaces
3. In order to use dynamic names for your pods, you can change your yaml file to generate name and use kubectl create instead of apply to deploy this:

```
kind: Job

metadata:

      generateName: img-classification-
```

3.4 Resource Provisioning

While the provided link covers pretty much all that is for resource provisioning, please make sure of the following:

1. Your job yaml file should also have the limits specified. Without this, the pod deployment will fail.
2. At a given time, only 2 job pods will be running and the others queued - you can try this by deploying 3 pods in quick succession and then printing all the pods within the free-service namespace.

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

3.5 Exposing Image Classification

You should implement the server and ensure that it works by using a REST client.

1. Add rule to your master EC2 instance similar to MP-1 to allow http / tcp traffic on the specific port where your server is running.
2. For python clients, the following examples will work: https://github.com/kubernetes-client/python/tree/master/examples
3. For retrieving the pod information as desired by our MP, the following are the object mappings:

```
{
"name": pod.metadata.name,
"ip": pod.status.pod_ip,
"namespace":pod.metadata.namespace,
"node":pod.spec.node_name,
"status": pod.status.phase
}
```