MP12 Concerns and FAQs:

1. How to access a worker node in EKS?

-> Type "kubectl get svc" and get the IP addresses of the worker nodes and then use ssh
-> kubectl get nodes

2. Do we have to create a new VPC?
-> NO

3. **Some useful commands:**

kubectl get deployments -A --all-namespaces
kubectl get jobs -A --all-namespaces
kubectl describe <...>
kubectl get pods -A --all-namespaces
kubectl describe pod <pod_name> --all-namespaces
kubectl logs <...>
kubectl get nodes -A --all-namespaces
docker ps -a
docker container prune
docker images

4. **TIP**: In the job yaml file(s) you can use **generateName** instead of **name** to get a unique name and be able to create new job using the same yaml file

5. When specifying an image within a yaml file for kubectl, how do I specify a local docker file?
-> When you create a docker image, it gets deployed to your local docker image repo. You just have to specify that image name within the yaml file under image.

6. Also a few tips to save you some money in AWS. you can work in this order:

6a. Containerize the classify.py first (you may need a t2.medium to build the image and test container, with ~20GB volume. You can change it back to t2.micro)

6b. Develop the web application to expose image classification. (you can't test at this point just yet)

6c. Configure kubernetes and launch AWS EKS (you can delete cluster or detach node group and recreate if you have to)

6d. Test job deployment manually, and then Test the web application (for every test you should delete jobs from both default and free-service namespaces)

7. Students suggested MP guide:

https://docs.google.com/document/d/1cPkPejiiuqjq0czuG2WsEsASueYTWpl5aPpBr1a1KUM/edit?usp=sharing

8. How can I minimize the cost?

EKS along with 2 t3 medium nodes charge about ~ $0.2 / hour which you can start and stop as you work on the MP. If you want to minimize this cost further, you can do everything from section 3.2 to 3.5 locally and then deploy it to EKS. In order to setup kubernetes on your local machine, the following tutorial will help:

https://kubernetes.io/docs/setup/learning-environment/minikube/

While this will start a single node server, most of the configurations will still stay the same.

9. I have set up Docker, Flask server etc. on my EC2 instance of educate starter account. Do I have to set it up again on my personal EC2 to access EKS?

-> No. You can access EKS through any system as far as you have configured aws credentials properly.

https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html

https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-configure.html

10. I am spending too much time on this MP. What should I do?

-> It is extremely important to have a conceptual understanding first and then try to approach this MP.

11. When I run my jobs, the status goes to ContainerCantRun.

-> Use your path variables value and add it to the dockerfile. When you build the docker it will now include an updated path.

12. The Docker image created to run the provided script lives on Docker Hub. Where exactly do Kubernetes live in our system? It can't be either of the 2 EC2 instances, right?

-> K8s acts as a container management tool.

13. From what I understand, Kubernetes just executes our Docker containers in a specific state, such as setting the Environment variables, correct?

-> K8s acts as a container management tool.

14. When we invoke Kubernetes in AWS, how will it know how to reach our EC2 instances?

-> You need to specify which image to deploy on k8s.

15. What is the purpose of running 2 EC2 instances?

-> You need containers to run your applications.

16. How to respond to MP's grader GET and POST?

-> There is no code for the GET request. You are just hitting the endpoint to get the fields from each of your jobs as listed in the project description.

17. The autogravder test.py is returning with a 502 bad gateway every time

-> In test.py I was adding http:// before my IP, the solution was to only use IP:PORT as specified.

18. ModuleNotFoundError: No module named 'torch'

-> RUN pip --no-cache-dir install torch torchvision in Dockerfile

19. How to generate Free Service vs Default namespace?

-> You have to use generate-name or change the name dynamically.You cannot deploy the pod with the same name

20. To delete all pods

-> kubectl delete --all all --namespace=free-service

21. MP context diagram: (Designed by students)

https://docs.google.com/drawings/d/16n-ll084RyDPXsXzRvz0RI1N02CT1VxtslrslhIqssk/edit

22. TIPS:

AWS EKS/Kubernetes setup

** Create EC2 (t2.micro) (aka node 0) -> ssh into the EC2 instance and then setup the EKS so that the credentials are managed through that.
** Add rule to allow http traffic (Used port 8080)
** Followed the instructions https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html
** Use "Cluster with Linux-only workloads" option and passed ssh public key from EC2 (node 0) which is required if you want to ssh to worker nodes (node 1 & node 2 both are t3.medium)
** When I created the cluster I used only specific zones (3 zones)

*** Containerization**

** The classify.py file that I used had hard-coded DATASET & TYPE which I fixed manually
** I modified classify.py to flush stdout to see the output
** Added ENV to Dockerfile
** To keep it simple, I manually downloaded the raw/processed mnist data from internet and built it into the Docker image and turned off the download in classify.py
** HINT: DATASET value mnist vs kmnist ;** Tested the Docker image on node 0 and then did the same on node 1 and node 2 (so I have image locally available on node 1 & node 2)
** Used tag when building docker image (did not use ":latest")

*** Deploying to cluster**
** Created two job specs (one for free service and another one for premium)
** Tested job specs locally on node 0
** Defined ENV in job spec

*** Resource Provisioning**
** Followed the instruction at https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/quota-memory-cpu-namespace/ to create & apply resource quota
** Locally tested the resource quota on node 0
** Ensured that atmost there can be only two free-service jobs and there can be any number of premium jobs

*** Exposing Image Classification**
** Created GET "/config" endpoint to return pod details
** Created two POST endpoints: one for free and another one for premium (HINT: DATASET value mnist vs kmnist ;-))
** Log debug messages from your RESTful endpoints so that you can find out what's going when you run test.py
** You don't have to necessarily use client libraries ;-)
* Created a simple shell script on node 0 to delete jobs. Required before running test.py
* Finally delete all the services

23 More tips:
For MP12, autograder creates 3 free POSTs, then call a GET config to check if there are only 2 free pods running, then it repeats the same process with 3 premium POST and check config again to check if 3 pods were created, finally it call a premium POST and a config GET again.
One common **mistake iss: inside free and premium yaml files I was using "name: <job_name>" (inside metadata). This means that the free and premium jobs were not creating new pods every time I request a POST. Instead, by using "generateName: <job_name>-" (inside metadata), I solved this issue, so now every POST creates a new pod, and ResourceQuota in the free-service namespace limits to a max of 2 free pods.**
Also, students can access kubectl commands directly inside the FLASK app to parse the information necessary to create the config GET. However, autograder calls many of the GET and POST in sequence, and accessing directly kubectl commands in Python is a slow solution (takes 3s to return the config GET), making autograder to generate a timeout error. The solution is to use the python API directly through load_kube_config() and ApiClient().
A good way to increase the productivity to create and test your FLASK code, clean your pods/jobs, create/delete your aws clusters and quickly run the autograder is to use your local machine and two jupyter notebooks. You can safely transform your local PC in a web server using a free app called: ngrox. You use one notebook to run the flask app and one notebook to track pods and jobs created, delete them easily, without to repeat many commands in an Ubuntu

VM and run autograder. This saved me a lot of time in configurating an EC2 and testing my code.