AI 510 Artificial Intelligence of Cloud Computing

HOS08A Kubernetes with Containers

07/02/2024 Reviewed by Anh Nguyen

07/03/2024 Reviewed by Naveena Moddu

School of Technology and Computing (STC) @City University of Seattle (CityU)

Before You Start

- The directory path shown in the screenshots may be different from yours.
- Some steps are not explained in the tutorial. If you are not sure what to do:
    - Consult the resources listed below.
    - If you cannot solve the problem after a few tries, please contact the student worker through the MS Teams course channel.

Learning Outcomes

- Students will be able to learn:
    - Introduction to Kubernetes on a local machine
    - Kubernetes configuration with Docker container images

Resource

- Kubernetes, (n.d), Production-Grade Container Orchestration https://kubernetes.io/
- minikube, (n.d), minikube start, https://minikube.sigs.k8s.io/docs/start/
- Kubernetes, (n.d), Concept of Container Images, https://kubernetes.io/docs/concepts/containers/images/

Docker Images to Kubernetes

From our previous module 07, we have seen how to configure Kubernetes and deploy already existing Nginx container images. In this module, we will learn how to apply custom Docker images to Kubernetes.

Here are a few reminder explanations of the components of Kubernetes:

- Pod
    - The smallest unit of Kubernetes
    - An abstraction over the container
    - Usually, one application is deployed per pod
    - Each Pod gets its own IP address
    - New IP address re-creation if the Pod dies
- Service
    - It provides a permanent IP address
    - Lifecycle of Pod and Service NOT connected

- Ingress
  - Route traffic into the cluster
- ConfigMap
  - Stores the external configuration of your application.
- Secret
  - Stores the base64 encoded secret data such as ID and password.

HOS08A Setup

We will reuse the HOS02A final result and HOS07A Kubernetes on the Minikube setup.

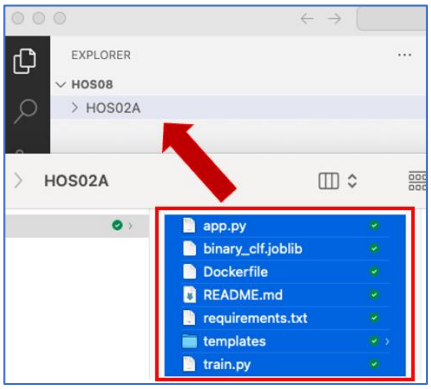
If you have uninstalled Docker or Kubernetes on Minikube, please follow previous HOS02A to install Docker and HOS07A to configure Kubernetes to the dashboard along with testing an example Docker image in the configured Kubernetes cluster.

Prediction Service Test

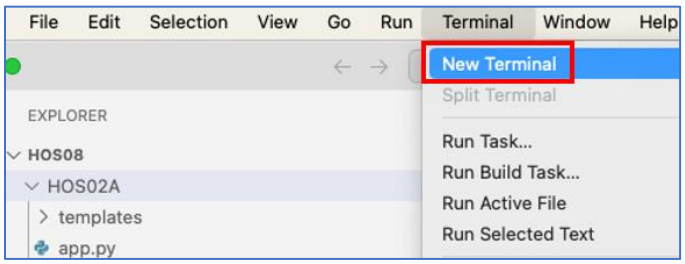
Most of this section's steps were done in HOS02A. However, to practice general MLOps deploy flow in chronological order, we will start by ensuring prediction service functionality in the host OS.

Step 1) Open VSCode, Open or create a folder for the HOS08 task

Step 2) Paste the provided HOS02A code to a HOS08A folder.



Step 3) Open a terminal



Step 4) In the terminal, change the directory into HOS02A and install service-related Python packages to your host environment with the following commands:

```
cd HOS02A
```

Optional – If you have already installed the necessary packages in HOS02A, skip this.

```
pip install -r requirements.txt
```

Step 5) We will train and start the server in your host OS by running the following commands:

Train model: `python3 train.py`

1) **Start** our prediction service:

```
python app.py
```

2) **Open** server URL with /predict and data: `http://127.0.0.1:3456/predict/10`
3) **Change** server URL with /predict and data: `http://127.0.0.1:3456/predict/100`

Take **Screenshots** of your Step 5 - 2 & 3 to be submitted after HOS08A. After taking screenshots, press **ctrl + c** to stop the server in the terminal.

Prediction Service in a Docker Container

Our HOS02A already had a Dockerfile as well. The host OS-tested prediction service from the previous section will be packaged into a Container image and tested before the image is pushed to the public Docker registry.

Step 1) We are still in the HOS02A folder in the terminal. In the terminal, run the following command to build a docker image and start a prediction server using the newly built image. Before running the command, start Docker Desktop.

1) **Build** image(include "**.**" char):

```
docker image build -t flask_docker_image .
```

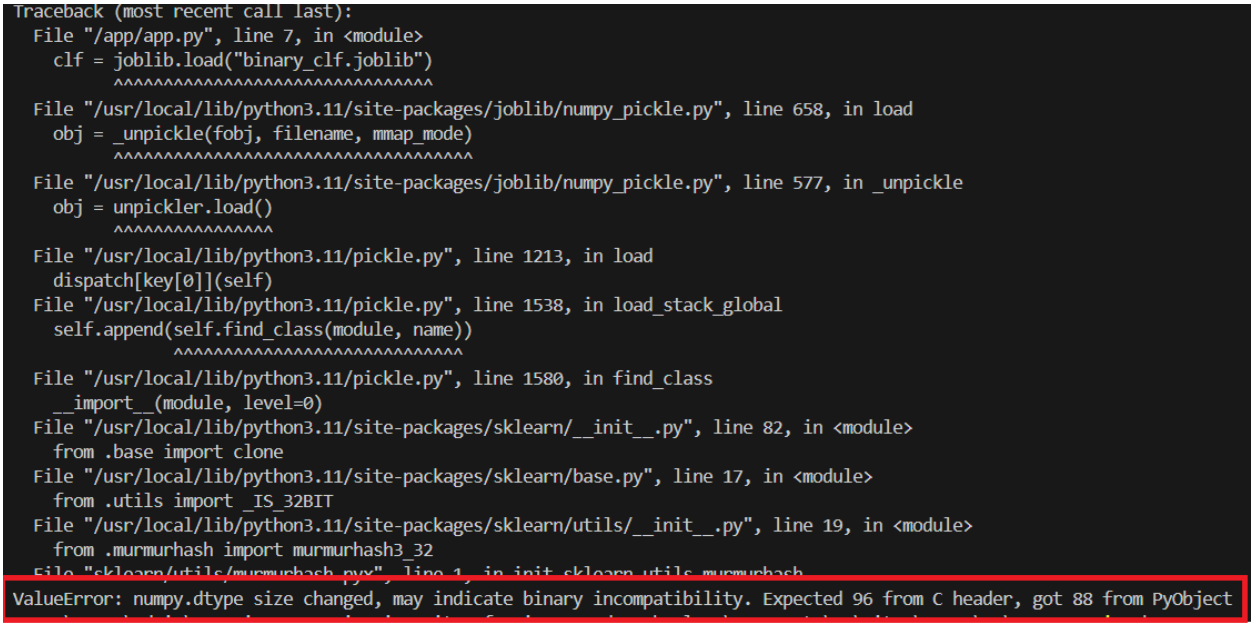
2) **Run** the container:

```
docker run --rm -p 3456:3456 flask_docker_image
```

3) **Open** server URL with predict and data: http://127.0.0.1:3456/**predict/10**
4) **Change** server URL with predict and data: http://127.0.0.1:3456/predict/100

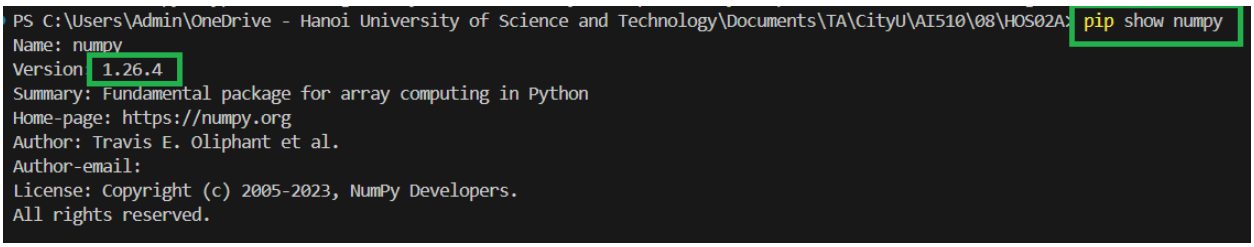
Take **Screenshots** of your Step 1 - 1 ~ 3 to be submitted after the HOS08A. After taking screenshots, press **ctrl + c** to stop the server in the terminal.

Note: If you encounter the following error, it might be caused by a mismatch between your local's numpy version and the docker container's numpy version.

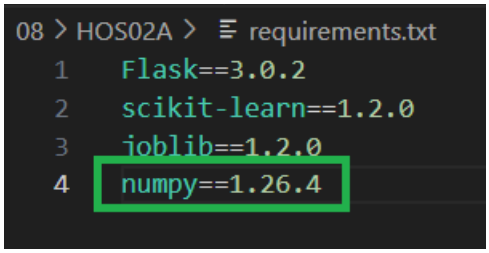


How to fix?

1. Get your local numpy version: `pip show numpy`



2. Put the numpy version into requirements.txt



3. Rebuild Docker image: `docker image build -t flask_docker_image .`
4. Re-run the container: `docker run --rm -p 3456:3456 flask_docker_image`

Step 2) Starting here, we will take different steps from what we did in the HOS02A. From HOS05, we have used the public Docker registry – Docker hub to push the Jenkins-built Docker image *through the Jenkins pipeline*. Log in to the Docker Hub page(https://hub.docker.com/) to double-check your account ID.

Step 3) We built the image named "flask_docker_image" in Step 1. We will tag the built image with the "latest" tag to the built image with the following command in the terminal.

```
docker tag flask_docker_image {YOUR_DOCKER_HUB_ID}/flask_docker_image:latest
```

Example(ex id = abc_example_id):

```
> docker tag flask_docker_image abc_example_id/flask_docker_image:latest
```

Explanation : `docker tag {image name} {repo owner name}/{image name}:{tag name}`

Step 4) Login to your Docker Hub with your credentials

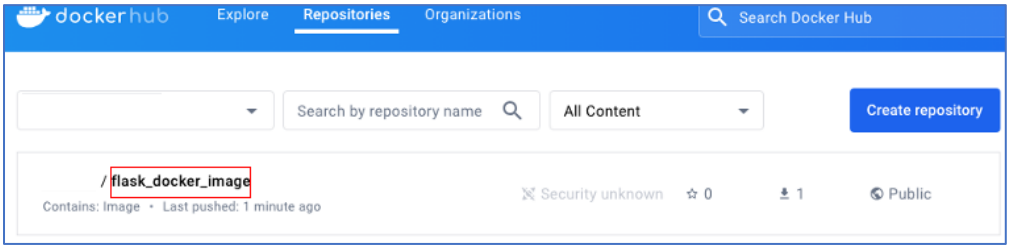
```
docker login docker.io
```

```
PS C:\Users\Admin\OneDrive - Hanoi University of Science and Technology\Documents\TA\CityU\AI510> docker login docker.io
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: [redacted]
Password:
Login Succeeded
```

Step 5) Push tagged Docker image using the following command in the terminal. Push speed may vary depending on your internet connection but it will take about 3 ~ 5 min to push the image to the Docker registry.

```
docker push {YOUR_DOCKER_HUB_ID}/flask_docker_image:latest
```

Check if the image is uploaded in the Docker hub (https://hub.docker.com/)



Take a **Screenshot** of this Docker Hub page to be submitted after the HOS08A.

Kubernetes - Use Docker Hub Uploaded Image

Step 1) If you uninstalled the minikube installed in HOS07A, please view "HOS07A - **Setup Minikube and Kubectl**" to complete the necessary installations. If you didn't uninstall minikube, run the "minikube start" command in the terminal to start the cluster.

```
minikube start
```

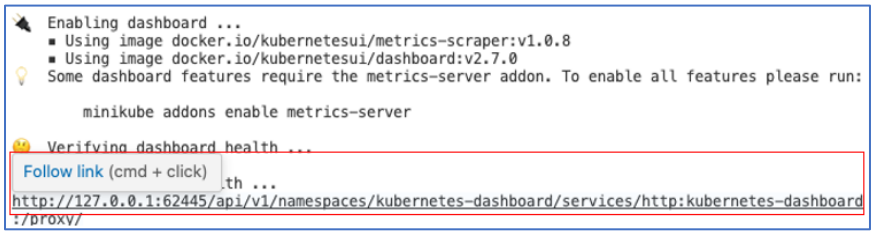
Output example:

```
😄  minikube v1.33.0 on Darwin 14.4.1 (arm64)
✨  Automatically selected the docker driver
📌  Using Docker Desktop driver with root privileges
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.43 ...
🔥  Creating docker container (CPUs=2, Memory=7792MB) ...
🐳  Preparing Kubernetes v1.30.0 on Docker 26.0.1 ...
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: storage-provisioner, default-storageclass
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Step 2) In the terminal, run the following commands to enable metrics and view the Kubernetes dashboard.

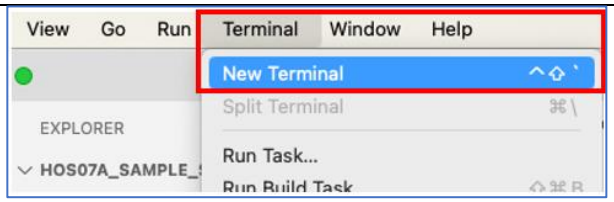
```
minikube addons enable metrics-server

minikube dashboard –url
```

When the URL shows, cmd/ctrl + click URL > “Follow link” to open the Kubernetes dashboard in the web browser.



Step 3) As briefly addressed in the introduction, the Ingress manages traffic routes. We will go back to the terminal and enable Ingress. Open a new terminal and run:

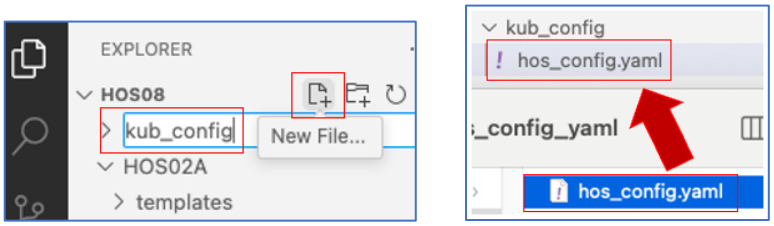
```
minikube addons enable ingress
```



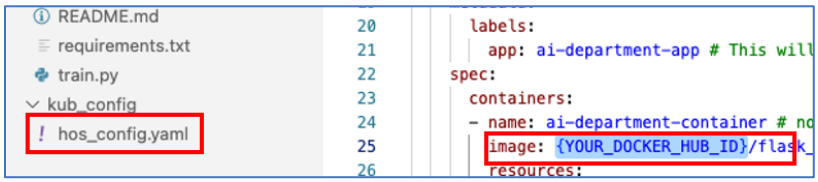

```
> minikube addons enable ingress
    ingress is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/mas
    After the addon is enabled, please run "minikube tunnel" and your ingress resources would be
    ▪ Using image registry.k8s.io/ingress-nginx/controller:v1.10.0
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.0
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.0
🔎  Verifying ingress addon...
🌟  The 'ingress' addon is enabled
```

Step 4) We configured the necessary add-ons to deploy our container image to the minikube Kubernetes cluster and trigger the prediction from our machine. In VSCode, create a new folder called “kub_config” under HOS08, and copy and paste the provided “hos_config.yaml” file under the folder. We will edit the Kubernetes configuration yaml file which reflects our container image.

The config file contains more than a deployment configuration document. It contains, namespace, deployment, service, and ingress configuration documents.



Step 5) In the file edit the "`image:{YOUR_DOCKER_HUB_ID}/flask_docker_image:latest`" under the "`kind: Deployment`" document. Replace "`{YOUR_DOCKER_HUB_ID}`" with your Docker ID. If your Docker ID is "ABC" it would be "`image: ABC/flask_docker_image:latest`". The `containerPort` is already configured to match our container image prediction application port(ex. `containerPort: 3456`). Note that "---" means separation between Kubernetes config yaml for Kubernetes.



Step 6) In the terminal where we ran enable ingress, execute the following command to apply our config file to the Kubernetes cluster.

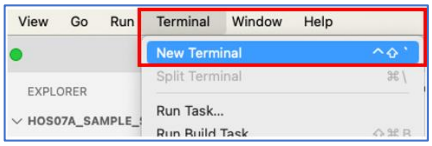
```
cd kub_config

kubectl apply -f hos_config.yaml
```

```
% minikube addons enable ingress
  ingress is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master/OWNERS
  After the addon is enabled, please run "minikube tunnel" and your ingress resources would be available at "127.0.0.1
"
    ▪ Using image registry.k8s.io/ingress-nginx/controller:v1.10.0
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.0
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.0
  Verifying ingress addon...
  The 'ingress' addon is enabled
% cd kub_config
% kubectl apply -f hos_config.yaml
```

Step 7) As we said in HOS07A, managed services such as "Google Kubernetes Engine", users usually visit the provisioned endpoint by Ingress. However, due to the limitation of minikube, we need to do an extra step to visit the Ingress endpoint. Open a new terminal and run:

```
minikube tunnel
```

Note that we need to have this terminal keep it open in parallel to the dashboard running terminal. In the new terminal, execute the following command to open a tunnel connection to the cluster from the host OS.

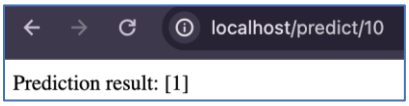



```
minikube tunnel
Tunnel successfully started
NOTE: Please do not close this terminal as this process must stay alive for the tunnel to be accessible ...
```

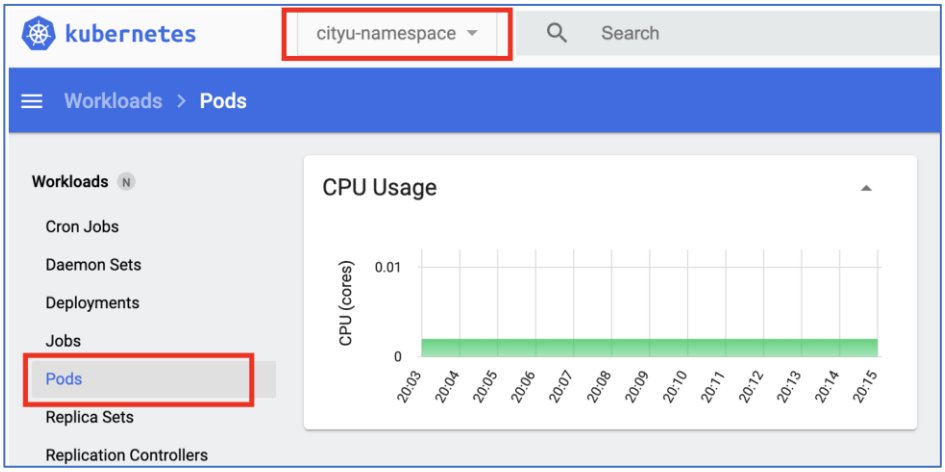
Step 8) Now that prediction Deployment, Service, and Ingress are configured and accessible through the minikube tunnel. Open a Web Browser and run prediction. The root URL should be "localhost"

1) **Open** server URL with **predict** and **data**: http://localhost/**predict/10**
2) **Change** server URL with **predict** and **data**: http://localhost/**predict/100**

Note that Ingress routes traffic to the right service port (ex. 6000) and container port (ex. 3456) while Ingress also captures all of the request traffic that comes to the root of your hostname (ex. http://localhost/).



Step 9) Go back to the Kubernetes Dashboard we opened in Step 2 of this section and take a **screenshot** of your "Pods" view from namespace "cityu-namespace" to be submitted after the HOS08A.



HOS submission instructions:

1. Please install the GitHub Desktop: https://cityuseattle.github.io/docs/git/github_desktop/
2. Clone, organize, and submit your work through GitHub Desktop: https://cityuseattle.github.io/docs/hoporhos