Custom Docker Image for Azure Machine Learning Project

Prerequisites

Azure Data Science VM – Ubuntu 18, which already has Python, Docker and Visual Studio Code installed. Please verify and if not present install below –

* Python version 3.8 or later. [Download Python](https://www.python.org/downloads/)
* Docker running locally. Follow the instructions to [download and install Docker](https://docs.docker.com/desktop/)
* An IDE or a text editor to edit files. We recommend using [Visual Studio Code](https://code.visualstudio.com/Download).

If you want to login to the Ubuntu Desktop, download and install [X2GoClient](https://wiki.x2go.org/doku.php/doc:installation:x2goclient) in your laptop to connect to the Azure DSVM. Alternatively, you can ssh to the Ubuntu DSVM for the tasks below, as well.

Create my custom Docker Image

Create the Dockerfile and requirements.txt

We will create a docker image that includes Conda version 4.9.2 with Python 3.8.5.final.0 from the *continuumio/miniconda3*base image.

Login to the Azure DSVM created above in Prerequisite section, to carry out this project. And open a ‘Terminal’ / bash shell to run the following.

First create a folder *docker-python-aml* in your home folder and cd to that folder.

Create a file named ***requirements.txt*** and add the following lines there. This will contain the list of python packages we plan to put in the docker image.

Flask==1.1.2

json5==0.9.5

jsonschema==3.2.0

matplotlib==3.3.4

numpy==1.20.1

oauthlib==3.1.0

pandas==1.2.3

pip==21.0.1

scikit-image==0.18.1

scikit-learn==0.23.2

xgboost==0.90

joblib==1.0.0

azureml-sdk==1.24.0

Now, create the file named *Dockerfile* with the following content.

# Start with the Conda 4.9

FROM continuumio/miniconda3

# Create working directory and instruct Docker to use this path

# for all subsequent commands

WORKDIR /app

# Copy requirements.txt from local folder to WORKDIR

COPY requirements.txt requirements.txt

# Upgrade pip first to 20.1.1

# Otherwise azureml-sdk will complain about

# not finding ruamel-yaml package

# (a weird bug in ruamel package possibly)

RUN python -m pip install --upgrade pip==20.1.1

# Execute pip command to install from requirements.txt

RUN pip3 install -r requirements.txt

# In order to avoid the error - "....Failed to load entrypoint

# automl = azureml.train.automl.run:AutoMLRun.\_from\_run\_dto.."

# Install this package

RUN pip3 install azureml-sdk[notebooks]

# Add source code into the image using the COPY command

COPY . .

ENTRYPOINT ["/bin/bash", "-l", "-c"]

CMD ["/bin/bash"]

Create a python program to test image has all packages

Create the file named *testPythonModulesNeeded.py* with the following content.

from azureml.core import Run

import pandas as pd

import numpy as np

import joblib

import os

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import roc\_auc\_score

from sklearn.metrics import roc\_curve

run = Run.get\_context()

print("Hello", run)

Create docker image

**# Name your docker image**

> export MY\_DOCKER\_IMAGE=tr-linux64-conda-python3.8-docker

**# Build the docker image MY\_DOCKER\_IMAGE**

# Make sure you have created the requirements.txt and the Dockerfile

# in the current directory and then run -

>sudo docker build –tag $MY\_DOCKER\_IMAGE .

**# Run the image and do quick test of python location, version, conda version etc**

>sudo docker run -t -i $MY\_DOCKER\_IMAGE

This above command should return a shell in the container.

**# Check the conda and python versions (expected 4.9.2 and 3.8.5.final.0**

(base) root@33396fc07e2:/app# conda info

**# Check if all ML packages are available in the image, by running this test python code**

(base) root@33396fc07e2:/app# python testPythonModulesNeeded.py

Hello <azureml.core.run. OfflineRun object at …>

# Exit the container shell

(base) root@33396fc07e2:/app# exit

Setup Azure Container Registry

Create an Azure Container Registry instance

Create a Service Principal to access images in Azure Container Registry

To access the docker images from the Azure Container Registry, you need to have the credentials to login first. There are multiple ways to achieve this. We will use the service principal option, which is common pattern for access by script.

Run the following Azure CLI code to create the service principal. You can run it in the Cloud Shell from the Azure Portal or from your local machine provided it has the Azure CL installed.

#!/bin/bash

**# Modify for your environment.**

**# ACR\_NAME: The name of your Azure Container Registry**

**# SERVICE\_PRINCIPAL\_NAME: Must be unique within your AD tenant**

ACR\_NAME=<container-registry-name>

SERVICE\_PRINCIPAL\_NAME=acr-service-principal

**# Obtain the full registry ID for subsequent command args**

ACR\_REGISTRY\_ID=$(az acr show --name $ACR\_NAME --query id --output tsv)

**# Create the service principal with rights scoped to the registry.**

**# Default permissions are for docker pull access. Modify the '--role'**

**# argument value as desired:**

**# acrpull: pull only**

**# acrpush: push and pull**

**# owner: push, pull, and assign roles**

SP\_PASSWD=$(az ad sp create-for-rbac --name http://$SERVICE\_PRINCIPAL\_NAME --scopes $ACR\_REGISTRY\_ID --role acrpull --query password --output tsv)

SP\_APP\_ID=$(az ad sp show --id http://$SERVICE\_PRINCIPAL\_NAME --query appId --output tsv)

**# Output the service principal's credentials; use these in your services and**

**# applications to authenticate to the container registry.**

echo "Service principal ID: $SP\_APP\_ID"

echo "Service principal password: $SP\_PASSWD"

Check the [documentation](https://docs.microsoft.com/en-us/azure/container-registry/container-registry-auth-service-principal#create-a-service-principal) on details on creating service principal.

Register image in Azure Container Registry

>sudo az login

>sudo az acr login –name tbdemoacr

>sudo docker tag $MY\_DOCKER\_IMAGE tbdemoacr.azurecr.io/demo-aml/$MY\_DOCKER\_IMAGE

>sudo docker push tbdemoacr.azurecr.io/demo-aml/$MY\_DOCKER\_IMAGE

**# Check if you can pull the image from your registry now**

>sudo docker pull tbdemoacr.azurecr.io/demo-aml/$MY\_DOCKER\_IMAGE

**# Test the image**

>sudo docker run -t -i $MY\_DOCKER\_IMAGE

Setup Azure Machine Learning Environment

Create Azure Machine Service

Create AML Environment in notebook

Open Azure Machine Learning Studio and create your notebook or import the notebook ***CreateCustomEnvFromADockerImage.ipynb*** from my repository [docker-python-aml](https://github.com/tirtho/docker-python-aml) in github,

Create a folder ***diabetes-training*** in the above notebook at the same level where you have created the .ipynb file above. Then, copy the ***diabetes\_training.py*** and the ***diabetes.csv*** files from my repository [docker-python-aml](https://github.com/tirtho/docker-python-aml) in github.

In ***CreateCustomEnvFromADockerImage.ipynb*** edit the following variables to enter your Azure Container Registry address and Service Principal details.

env\_name.docker.base\_image = "<your azure container registry name>.azurecr.io/demo-aml/<name of the docker image>:latest"

env\_name.docker.base\_image\_registry.username = "<azure container registry service principal id>"

env\_name.docker.base\_image\_registry.password = "<service principal password>"

Create a Compute Target named CustomAMLCompute or attach to your existing Compute Target by setting the following parameter, cluster\_name in the ***CreateCustomEnvFromADockerImage.ipynb*** notebook

cluster\_name = "CustomAMLCompute"

Now run your notebook ***CreateCustomEnvFromADockerImage.ipynb***.

Once it completes go to the Run history and find that the output folder contains the pickled file containing your trained model.

That’s it!!

Miscellaneous

Useful Commands

# Check if image is running in a container

>sudo docker ps

# Stop if it is running

>sudo docker stop $MY\_DOCKER\_IMAGE

# Check if MY\_DOCKER\_IMAGE is in local docker images list

>sudo docker images -a

# List dangling images

>sudo docker images -f dangling=true

# Remove dangling images

>sudo docker image prune

# Remove existing container named MY\_DOCKER\_IMAGE

>sudo docker rm $MY\_DOCKER\_IMAGE

# Remove existing MY\_DOCKER\_IMAGE image

>sudo docker rmi $MY\_DOCKER\_IMAGE

# Remove all docker images

>sudo docker rmi -f $(sudo docker images -a -q)