Time Duration: 60 minutes

Before you begin:

* You need an Azure Subscription for Azure IoT Hub
* Azure IaaS VM or a Raspberry PI 3 (preinstalled with Raspbain: <https://www.raspberrypi.org/documentation/installation/noobs.md>)
* You will need as SSH client, like Putty – or use Azure Cloud Shell

1. If not using a Raspberry Pi, In your Azure Subscription, create a “Ubuntu Server 18.04 LTS” VM.

* You can use a Standard HDD and the SKU D3, as we will need >5 GB RAM and 4 CPUs
* Enable SSH inbound

1. Create an Azure IoT Hub (or you can use an existing one).

* Select either S1 or F1 for this lab

1. In your IoT Hub under IoT Edge, click “Add IoT Edge Device” name the device save and copy the connection string for step 5 below.
2. Using your SSH client, connect to the PI or Ubuntu VM using the IP, ID and Password created in step 1
3. Install the Moby Container service and Azure IoT Edge:
   * If on Ubuntu: Run the following commands to install IoT Edge on Ubuntu 18.04 LTS (modified from: <https://docs.microsoft.com/en-us/azure/iot-edge/quickstart-linux>):

sudo curl https://packages.microsoft.com/config/ubuntu/18.04/prod.list > ./microsoft-prod.list

sudo cp ./microsoft-prod.list /etc/apt/sources.list.d/

curl https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > microsoft.gpg

sudo cp ./microsoft.gpg /etc/apt/trusted.gpg.d/

sudo apt-get update && sudo apt-get -y install moby-engine moby-cli && sudo apt-get -y install iotedge

* + If on Raspberry PI: Run the following commands to install IoT Edge on Raspbian (from: <https://docs.microsoft.com/en-us/azure/iot-edge/how-to-install-iot-edge-linux-arm>):

curl -L https://aka.ms/moby-engine-armhf-latest -o moby\_engine.deb && sudo dpkg -i ./moby\_engine.deb

curl -L https://aka.ms/moby-cli-armhf-latest -o moby\_cli.deb && sudo dpkg -i ./moby\_cli.deb

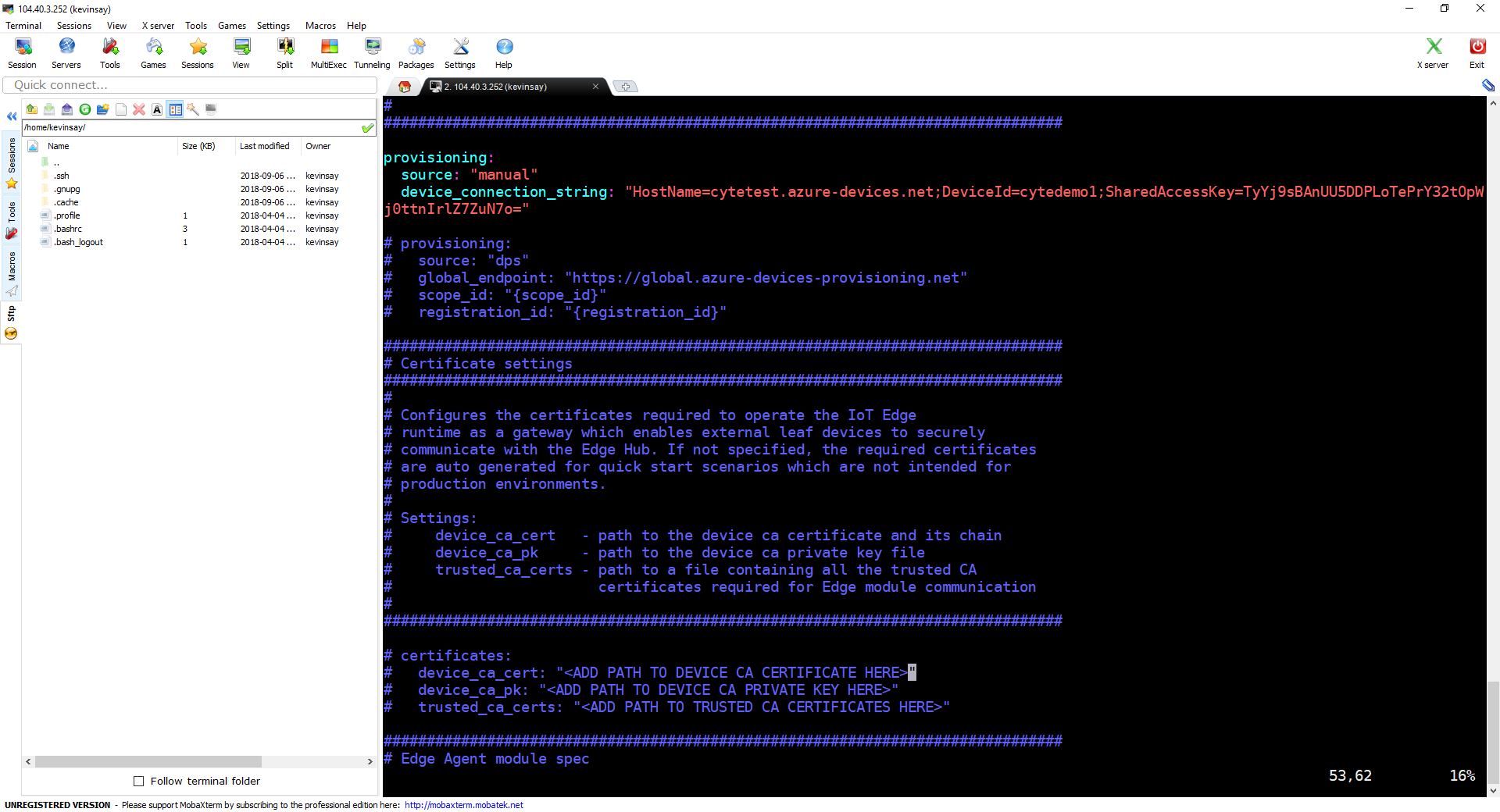
sudo apt-get install -f

curl -L https://aka.ms/libiothsm-std-linux-armhf-latest -o libiothsm-std.deb && sudo dpkg -i ./libiothsm-std.deb

curl -L https://aka.ms/iotedged-linux-armhf-latest -o iotedge.deb && sudo dpkg -i ./iotedge.deb

sudo apt-get install -f

1. Using nano, vi or vim, add the connection string from step 3 to “/etc/iotedge/config.yaml”, as shown below:



1. Run the following command to complete the setup and restart the service.

sudo systemctl restart iotedge

1. To view the “**active (running)**” status of the service, run the following command. You can also view the status in the Azure Portal.

sudo systemctl status iotedge

(informational only) New to Docker/Moby? Here are a few commands to get familiar with:

|  |  |
| --- | --- |
| Description | Command |
| Shows the running containers. Add an ‘-a’ to see all containers, running and not. | sudo docker ps |
| Shows the logs. Add an ‘-f’ after the word logs to continue reading the logs as they are written. | sudo docker logs edgeAgent |
| Show memory, network, disk and cpu of the running containers. Control + C to exit. | sudo docker stats |
| Shows the local cache of images and their size. | sudo docker images |
| Shows the config file for a specific container. | sudo docker inspect edgeAgent |
| Allows you to get inside a container. Great to troubleshooting what the file system looks like. Type exit to exit out of the shell. | sudo docker exec -i -t edgeAgent /bin/ash |
| Copies files from the local system to a directory inside a container.  Copy files from container to local | sudo docker cp run.py edgeAgent:/app/run.py    sudo docker cp edgeAgent:/app/run.py run.py |

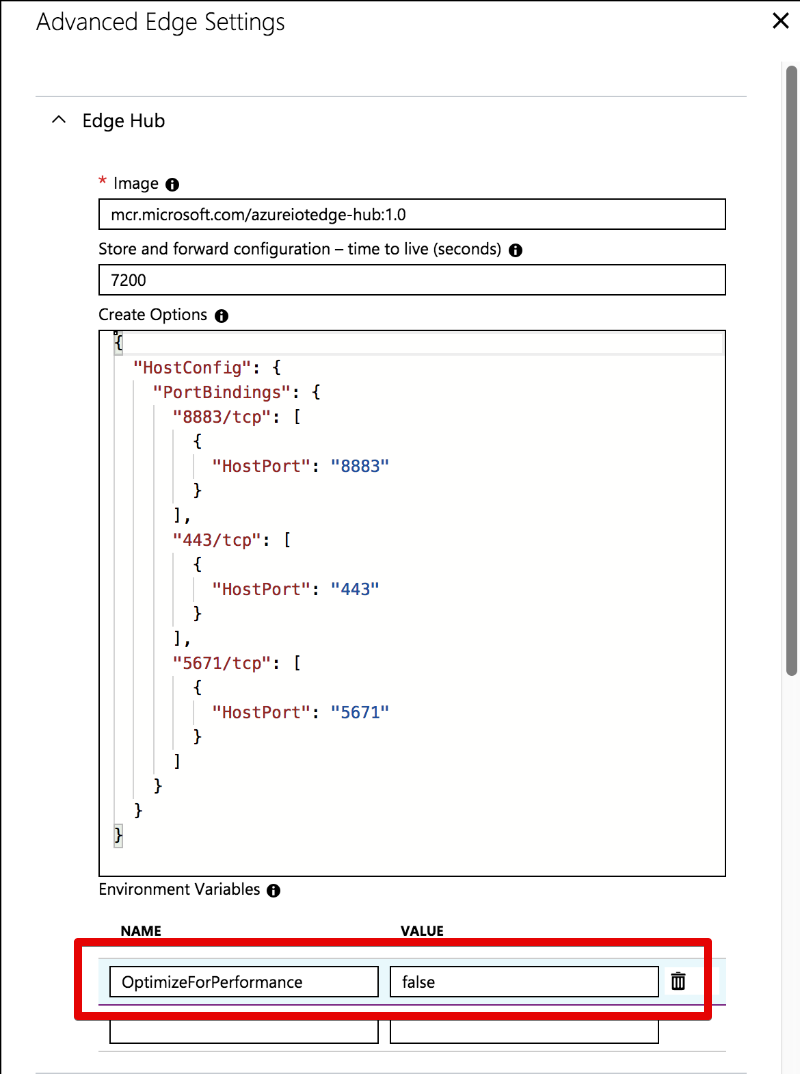
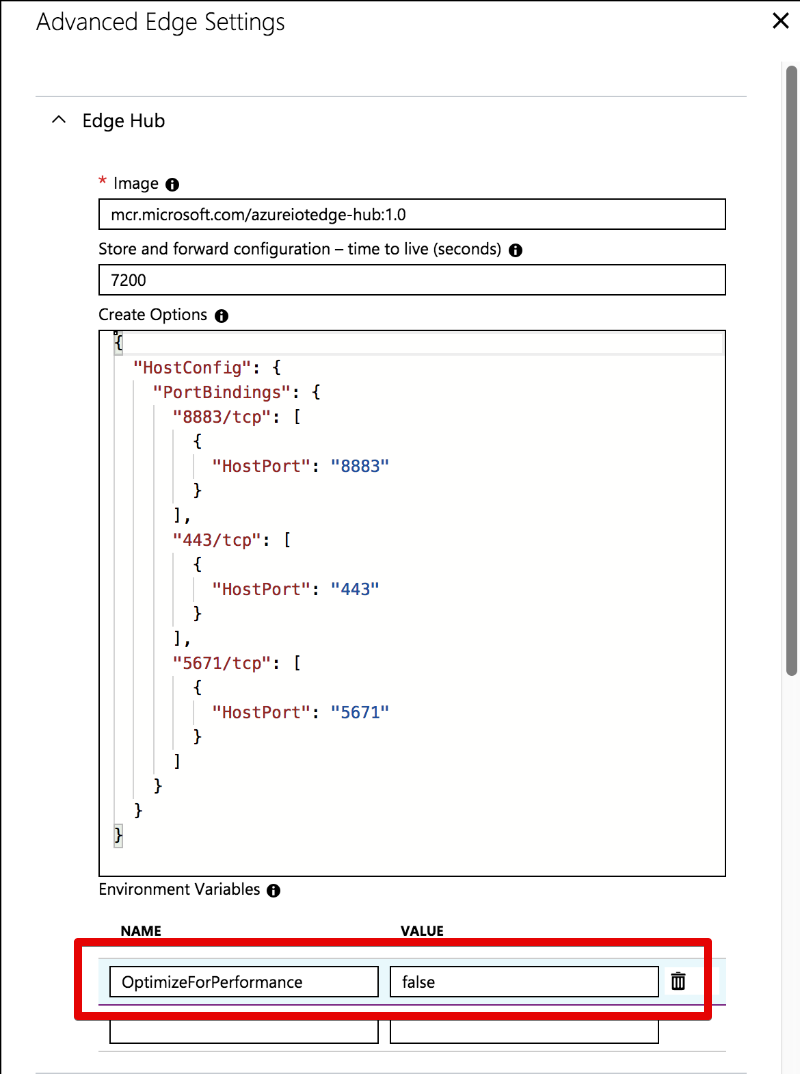
1. In the Azure Portal under your IoT Hub -> **IoT Edge**, select your IoT Edge Device ID. Click **Set Modules** and in the Deployment Modules section, click **Add**. Add a type **IoT Edge Module** for both modules listed below:
   * On Ubuntu:

|  |  |  |
| --- | --- | --- |
| Name | URL | Create Options |
| opencv | ksaye/cytefy19opencv:0.0.1-amd64 | {"ExposedPorts":  {"8080/tcp":{}}, "HostConfig":{"PortBindings":{"8080/tcp":[{"HostPort":"8080"}]}}} |
| tensorFlow | ksaye/cyteft19tensorflow:0.0.1-amd64 | {"ExposedPorts":  {"8181/tcp":{}}, "HostConfig":{"PortBindings":{"8181/tcp":[{"HostPort":"8181"}]}}} |

* + On Raspberry PI:

|  |  |  |
| --- | --- | --- |
| Name | URL | Create Options |
| opencv | ksaye/cytefy19opencv:0.0.1-arm32v7 | {"ExposedPorts":{"8080/tcp":{}},"HostConfig":{"Binds":["/dev/video0:/dev/video0"],"PortBindings":{"8080/tcp":[{"HostPort":"8080"}]},"Privileged":true}} |
| tensorFlow | ksaye/cyteft19tensorflow:0.0.1-arm32v7 | {"ExposedPorts":  {"8181/tcp":{}}, "HostConfig":{"PortBindings":{"8181/tcp":[{"HostPort":"8181"}]}}} |

*Note, because IoT Edge is optimized for performance AND a Raspberry PI has limited hardware, we also need to set the “OptimizedForPerformance” variable to false, as discussed here:* [*https://docs.microsoft.com/en-us/azure/iot-edge/troubleshoot#stability-issues-on-resource-constrained-devices*](https://docs.microsoft.com/en-us/azure/iot-edge/troubleshoot#stability-issues-on-resource-constrained-devices)



1. Click next and type in the following route, then click next and finally Submit.

{

"routes": {

"opencvToTensorFlow": "FROM /messages/modules/opencv/outputs/\* INTO BrokeredEndpoint(\"/modules/tensorFlow/inputs/input1\")",

"tensorFlowToIoTHub": "FROM /messages/modules/tensorFlow/outputs/output1 INTO $upstream"

}

}

1. Back on the Ubuntu or PI host, type the following command to watch the status:

sudo docker logs -f edgeAgent

You should see the following. Note, based on bandwidth, there may be timeout errors that recover. On Raspberry PI, it does take several minutes

2018-09-06 19:58:21.408 +00:00 [INF] - Updated reported properties

2018-09-06 20:53:39.981 +00:00 [INF] - Plan execution started for deployment 2

2018-09-06 20:53:40.010 +00:00 [INF] - Executing command: "Command Group: (

[Create module opencv]

[Start module opencv]

)"

2018-09-06 20:53:40.016 +00:00 [INF] - Executing command: "Create module opencv"

2018-09-06 20:54:41.642 +00:00 [INF] - Executing command: "Start module opencv"

2018-09-06 20:54:42.586 +00:00 [INF] - Executing command: "Command Group: (

[Create module tensorFlow]

[Start module tensorFlow]

)"

2018-09-06 20:54:42.592 +00:00 [INF] - Executing command: "Create module tensorFlow"

2018-09-06 20:55:30.328 +00:00 [INF] - Executing command: "Start module tensorFlow"

2018-09-06 20:55:31.170 +00:00 [INF] - Executing command: "Command Group: (

[Create module edgeHub]

[Start module edgeHub]

)"

2018-09-06 20:55:31.170 +00:00 [INF] - Executing command: "Create module edgeHub"

2018-09-06 20:55:42.139 +00:00 [INF] - Executing command: "Start module edgeHub"

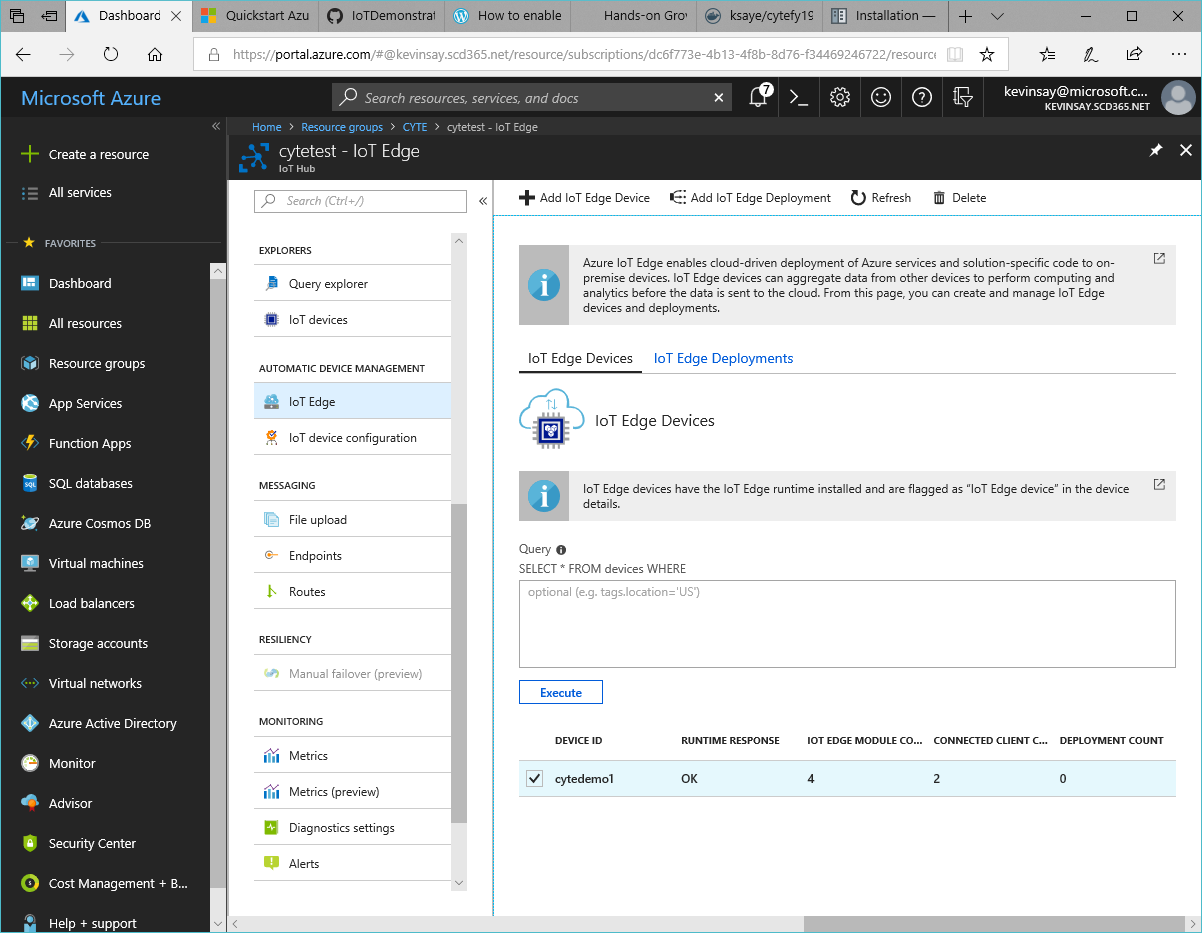
2018-09-06 20:55:42.857 +00:00 [INF] - Plan execution ended for deployment 2

2018-09-06 20:55:48.865 +00:00 [INF] - Updated reported properties

1. To verify the modules are running, run the following command on the Ubuntu or PI host:

sudo docker stats

1. In the Azure Portal, you should see 4 modules deployed and 2 custom modules are connected to Edge.



1. To see what your video modules are detecting:
   * On Ubuntu: on you IaaS VM, select Networking and add inbound ports for tcp/8080 and tcp/8181.
   * On PI: No changes needed.
2. Open a web browser to the IP address of your host, example http://{yourIP}:8080/default.html to see what OpenCV is capturing. Change the port to 8181 to see what TensorFlow is scoring/inferencing.
3. Using Visual Studio Code or Device Explorer (<https://github.com/Azure/azure-iot-sdk-csharp/tree/master/tools/DeviceExplorer>) monitor the messages your TensorFlow module is sending to the Cloud.

{"ObjectDetectionTimeMS": 65681, "ObjectDetectionURL": "http://172.18.0.3:8181/2018-09-07-22-37-34-image.jpg", "Classes": 546, "imageSize": 137303, "ManhattanImageChange": 0.8631416377314814, "ObjectsDetected": 5, "imageURL": "http://172.18.0.2:8080/2018-09-07-22-36-13-image.jpg", "ZeroImageChange": 3.5355794270833334, "dateTime": "2018-09-07T22:36:28", "OpenCVProcessingTimeMS": 14094.215, "Predicted": true, "GraphFile": "faster\_rcnn\_inception\_resnet\_v2\_atrous\_oid\_2018\_01\_28/frozen\_inference\_graph.pb", "imageFileName": "2018-09-07-22-36-13-image.jpg", "cameraURL": "rtsp://home.saye.org:8854/live8.264?user=mobile&passwd=\*\*\*\*\*\*", "ObjectDetections": {"Table": 0.37864378094673157, "Window": 0.5000331997871399, "Couch": 0.3602321147918701, "Sofa bed": 0.4550239145755768, "House": 0.3209814727306366}}

Observer: OpenCV time, TensorFlow time, image size, graph file name, and objects detected. Understand why we have a 90 second frequency in the openCV module.

1. In the Azure Portal, set the desired property of the opencv module. Set “cameraURL” to any RTSP URL or on PI, set the “cameraURL” to 0 (integer not string) to use the local camera.

The rest of the lab is optional.

1. How much compute power was needed? CPU, Memory, etc. What could have been optimized?
2. View the http://{yourIP}:8080/main.py file to fully understand the openCV code running. Port 8181 to see the TensorFlow code. How is the image passed from module to module? Why could I have not just encoded the image and passed it in the message?
3. View the Module Identity Twin of the tensorFlow module. What is the CKPT\_SOURCE URL and the CKPT\_SIZE? How many NUM\_CLASSES are there?
4. Note how we can set the TWIN property cameraURL which changes the URL for openCV. Add the same ability to update the ML model for tensorFlow.

The rest is REALLY optional.

1. Modify the file main.py in the tensorFlow module. Change line 64 to 'ssd\_mobilenet\_v1\_coco\_2017\_11\_17', change line 79 from oid to 'mscoco\_label\_map.pbtxt' and change line 75 to '90'. Restart the tensorFlow container and compare the “ObjectDetectionTimeMS”.
2. Download the source code here: <https://github.com/ksaye/IoTDemonstrations/blob/master/CYTE%20Demo.zip>. Open in Visual Studio Code to review.
3. Build your own Image Classification model.
4. Redeploy the solution on a local device and change the URL to 0 – which is the first USB camera. Be sure change your create container option to include the /dev/video0 port as shown below:

{"ExposedPorts":{"8080/tcp":{}},"HostConfig":{"Binds":["/dev/video0:/dev/video0"],"PortBindings":{"8080/tcp":[{"HostPort":"8080"}]},"Privileged":true}}