Microsoft

Workshop: Developing Custom Object Detection Models with NVIDIA and Azure ML Studio

Using Automated ML for Vision and Triton Server or NVIDIA Deepstream

AI at the edge

Is a growing market be it in manufacturing, energy, retail or smart cities.

In this workshop we will teach how to combine best of both worlds of the Azure and NVIDIA platforms. The content is built up in the following setting:

First: set up. In this part we will guide you through the set up to get started.

Second: label a dataset using Azure Machine Learning Studio.

Third: Use AutoML for vision to train a labeled Dataset and develop a production model.

At the end of the workshop you will have a model suitable for NVIDIA's Triton Server or Deepstream.

Part one: Set up

The set up consists of:

- Creating a Storage Account and loading the image data
- Creating a workspace resource to get started with Azure Machine Learning
  - Create a workspace
  - Create a compute instance
  - Create a datastore

Creating a Storage Account

1. Create a Storage account with Blob store container. If you are unsure how to create the storage account and upload the images use the following details and links to learn how.
2. **Please note:** As you create the storage account make sure you record the ***Storage Account Name, Blob Container Name and Access Key*** as these will be needed later on when we create a Datastore in AML.

    a. **Storage Account overview (Concepts)**: Explore Azure Storage services - Learn | Microsoft Docs

    Storage Account an Blob Store creation:

    To access Azure Storage, you'll need an Azure subscription. If you don't already have a subscription, create a free account before you begin.

    All access to Azure Storage takes place through a storage account. For this quickstart, create a storage account using the Azure portal, Azure PowerShell, or Azure CLI. For help creating a storage account, see Create a storage account.

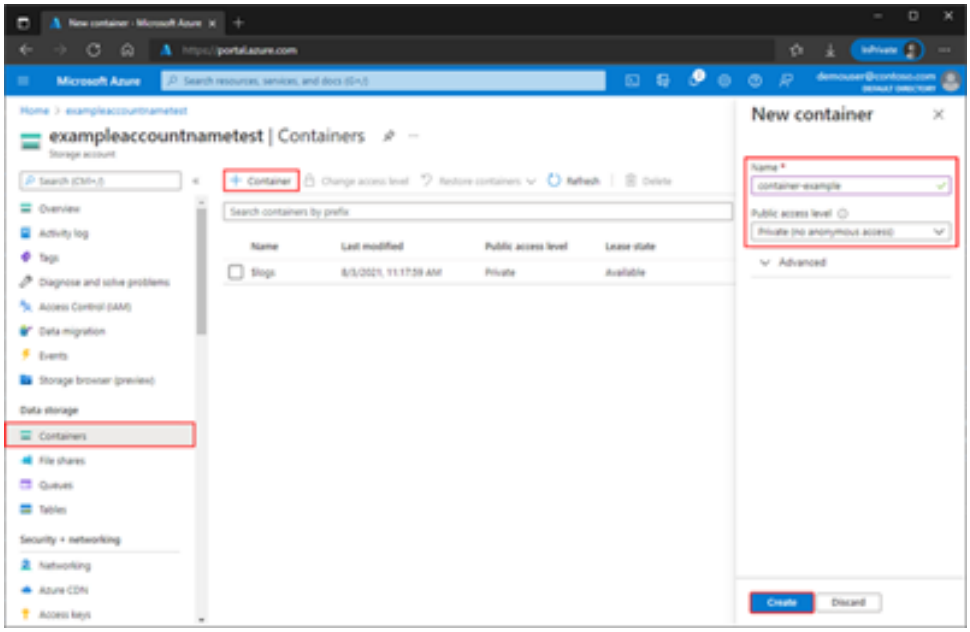
    Name your storage account: **< your unique storage account name>**

    !Record this name as it will be needed later.!

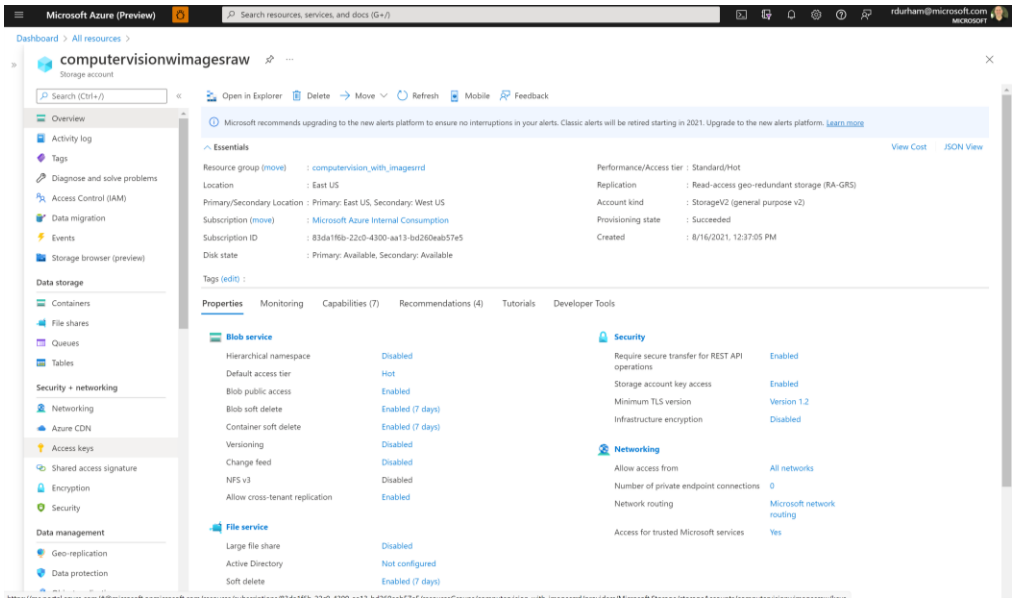
Create a container

To create a container in the Azure portal, follow these steps:

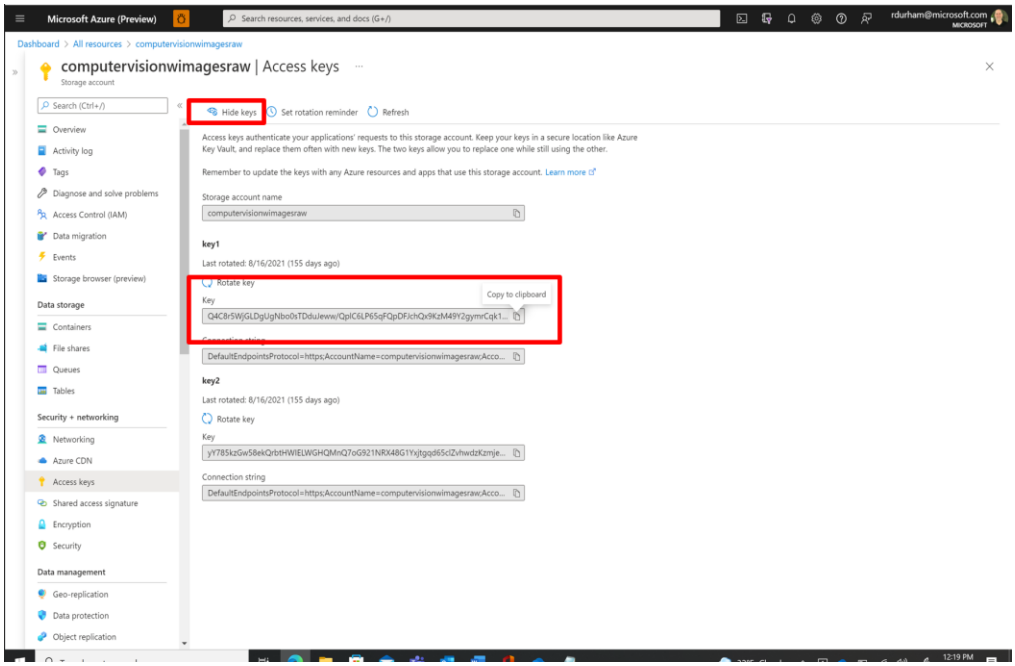
1. Navigate to your new storage account in the Azure portal.
2. In the left menu for the storage account, scroll to the **Data storage** section, then select Containers.
3. Select the + **Container** button.
4. Type a name for your new container. **Name your container**: <your container name> The container name must be lowercase, must start with a letter or number, and can include only letters, numbers, and the dash (-) character. For more information about container and blob names, see Naming and referencing containers, blobs, and metadata.
5. Set the level of public access to the container. The default level is **Private (no anonymous access**).
6. Select Create to create the container.



7. Once the container is created go to the Access keys area under the Security + networking portion of the panel and select the icon.



This will bring up the following screen. Click on the "Show keys" icon (shown below as Hide keys) and copy the key to the clipboard and then record it in notepad. This key will be needed later.



Upload the provided image data. This can be found by downloading and uncompressing the zip file found in **GTC\soda_data.zip.** The image data from this zip

file can be found in the **soda_data\train_img** directory and uploaded into the Storage account container named **soda-raw**. This should be 243 images. These images are numbered **0.jpg – 244.jpg**. ***Please note that at a minimum you will need at least 50 images to train an AutoML for Images model***.

For a reference on how to do this review the following link:

Quickstart: Upload, download, and list blobs - Azure portal - Azure Storage | Microsoft Docs

Create workspace resources you need to get started with Azure Machine Learning

Azure Machine Learning is a cloud service for accelerating and managing the machine learning project lifecycle. Machine learning professionals, data scientists, and engineers can use it in their day-to-day workflows: Train and deploy models, and manage MLOps.

You can create a model in Azure Machine Learning or use a model built from an open-source platform, such as Pytorch, TensorFlow, or scikit-learn. MLOps tools help you monitor, retrain, and redeploy models.

There are a number of advantages of using the Azure AML platform to create computer vision models. These include:

- An Enterprise grade platform service that facilitates the following capabilities when training and deploying CV models:
- A single platform to label, train and deploy models
- Scalability the ability to execute the code for the model training on one compute while the real training of the model happens on another compute that is completely scalable to align with the number of images and modeling tasks.
- Using the hyperdrive functionality of AutoML for images, it is possible to train hundreds of models using different algorithms and hyperparameters and then automatically have AML determine the best (champion) model automatically. This alone with the forementioned capabilities saves time and helps scale the overall model training/deployment process.

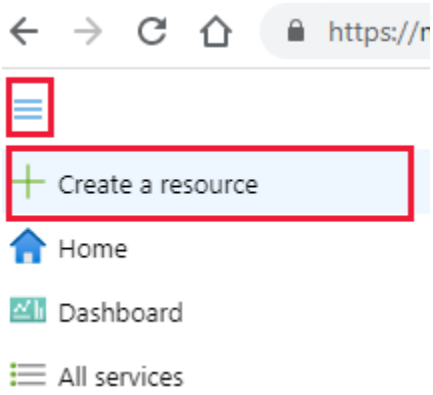
For more information on Azure Machine Learning: Azure Machine Learning - ML as a Service | Microsoft Azure

Create the workspace

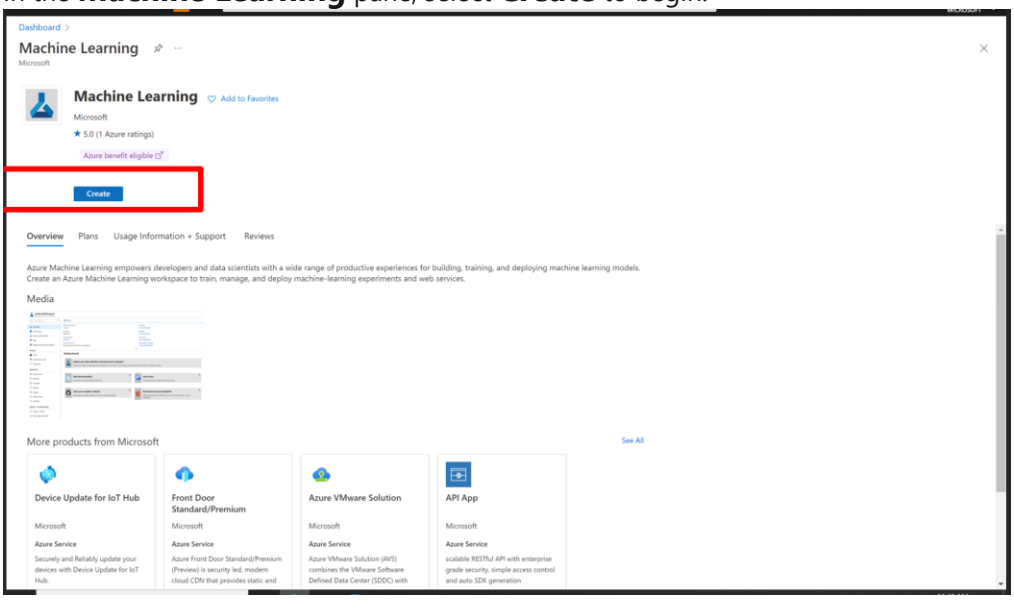
If you already have a workspace, skip this section and continue to Create a compute instance.

If you don't yet have a workspace, create one now:

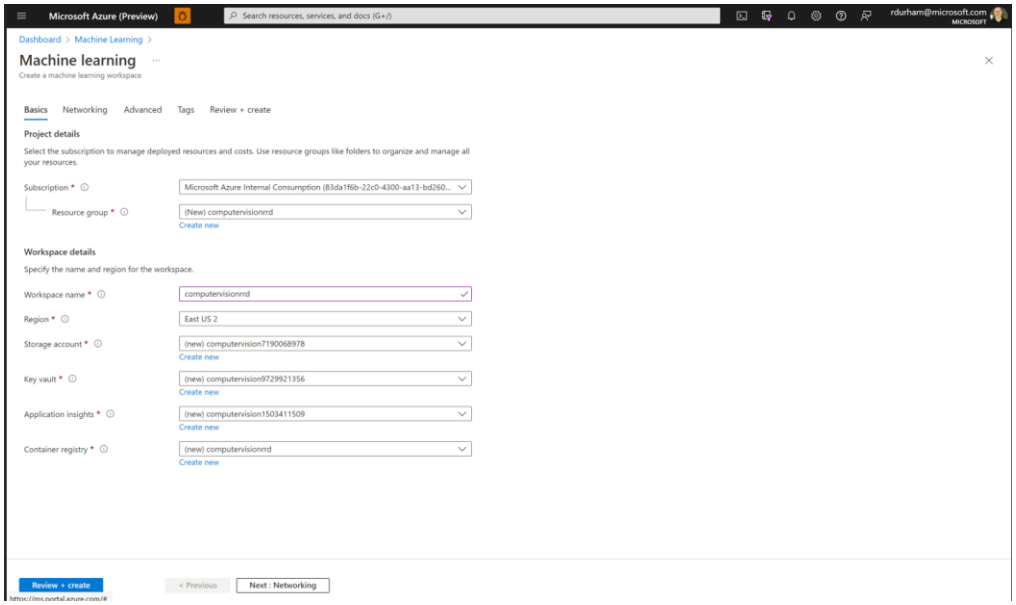
1. Sign in to the Azure portal by using the credentials for your Azure subscription.
2. In the upper-left corner of the Azure portal, select the three bars, then **+ Create a resource**.



3. Use the search bar to find **Machine Learning**.
4. Select **Machine Learning**.
5. In the **Machine Learning** pane, select **Create** to begin.



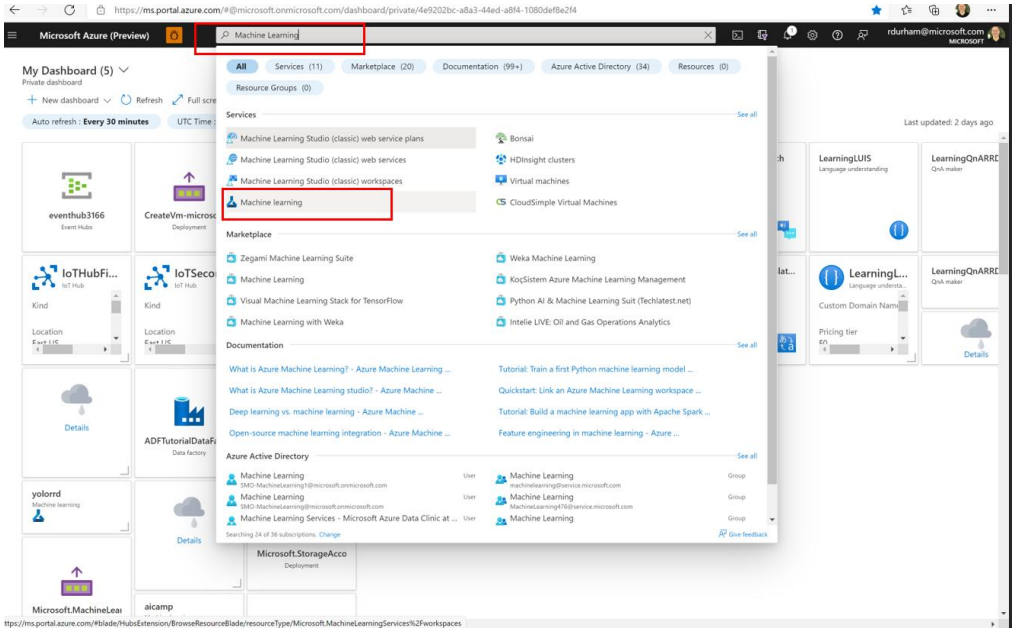
6. Provide the following information to configure your new workspace: **Note you may need to create a new resource group, workspacename, storage account, key vault, application insights and container registry for the workspace**. An example label might be using computervision + <your initials> as the textbox entry.
7. Select **Review and create** to create the workspace.



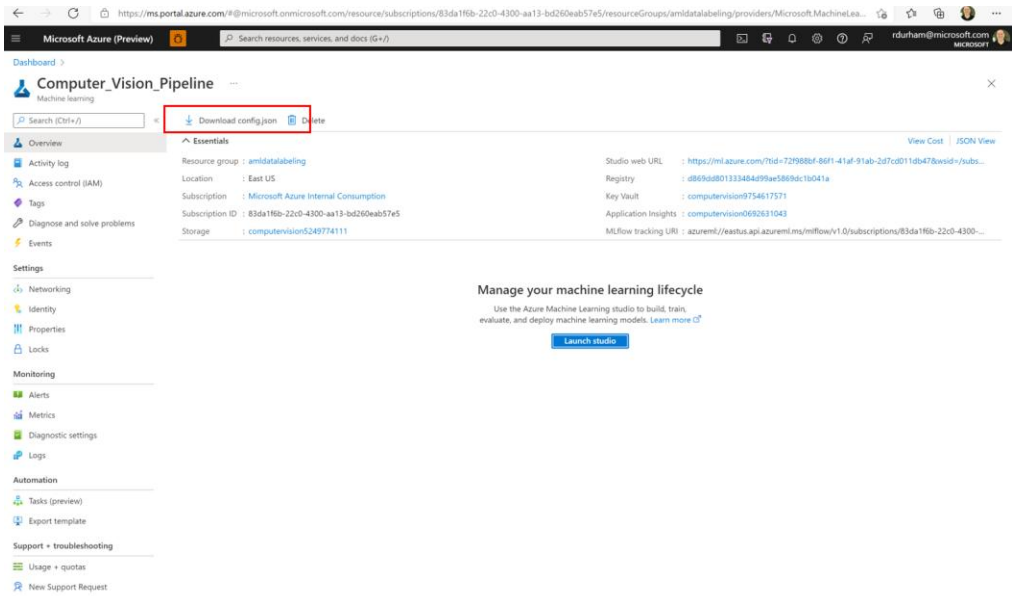
8. To view the new workspace go the Azure Portal and enter "**Machine Learning**" in the search bar:

WARNING: It can take several minutes to create your workspace in the cloud. When the process is finished, a deployment success message appears.

9. To view the new workspace go the Azure Portal and enter "**Machine Learning**" in the search bar:



10. Download the **config.json** file and store it locally for use later in this tutorial (see the image below for download location)



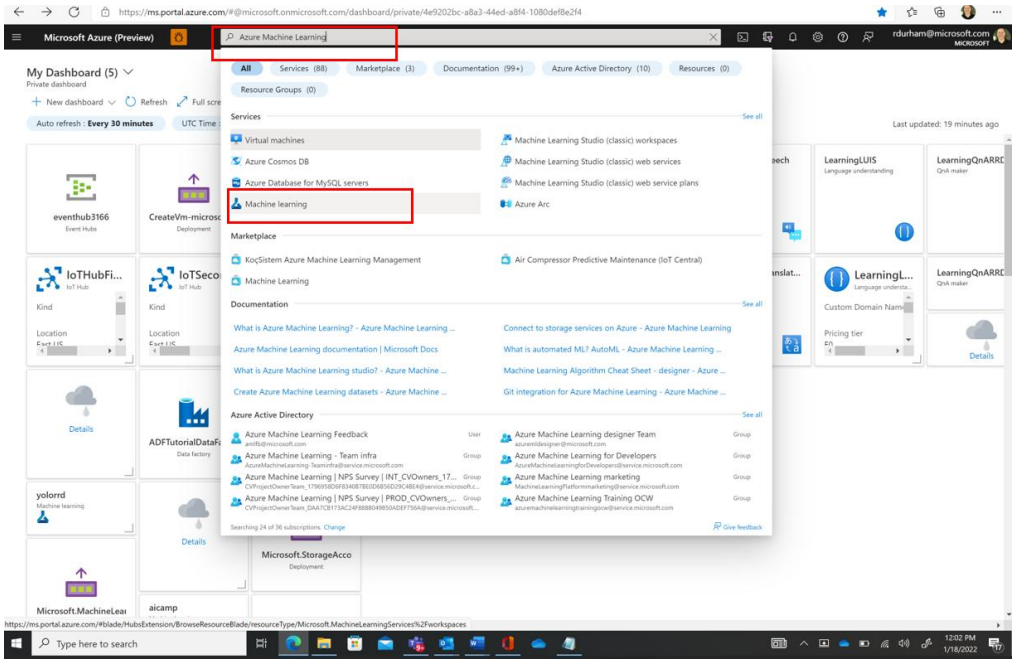
11. From the portal view of your workspace, select **Launch studio** to go to the Azure Machine Learning studio.

Create compute instance

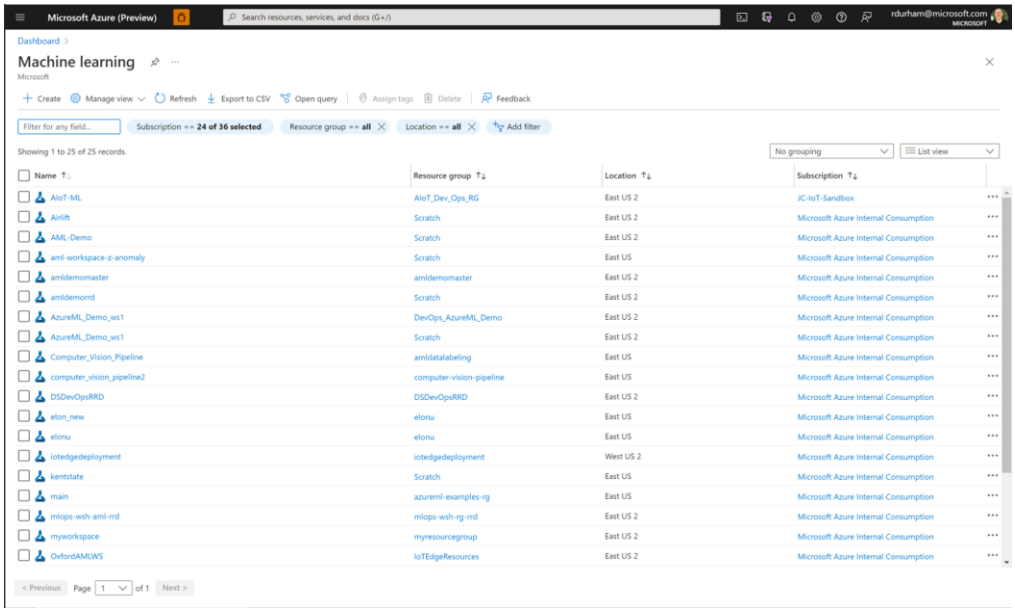
You could install Azure Machine Learning on your own computer. But in this quick start, you'll create an online compute resource that has a development environment already installed and ready to go. You'll use this online machine, a **compute instance,** for your development environment to write and run code in Python scripts and Jupyter notebooks.

Create a **compute instance** to use this development environment for the rest of the tutorials and quickstarts.

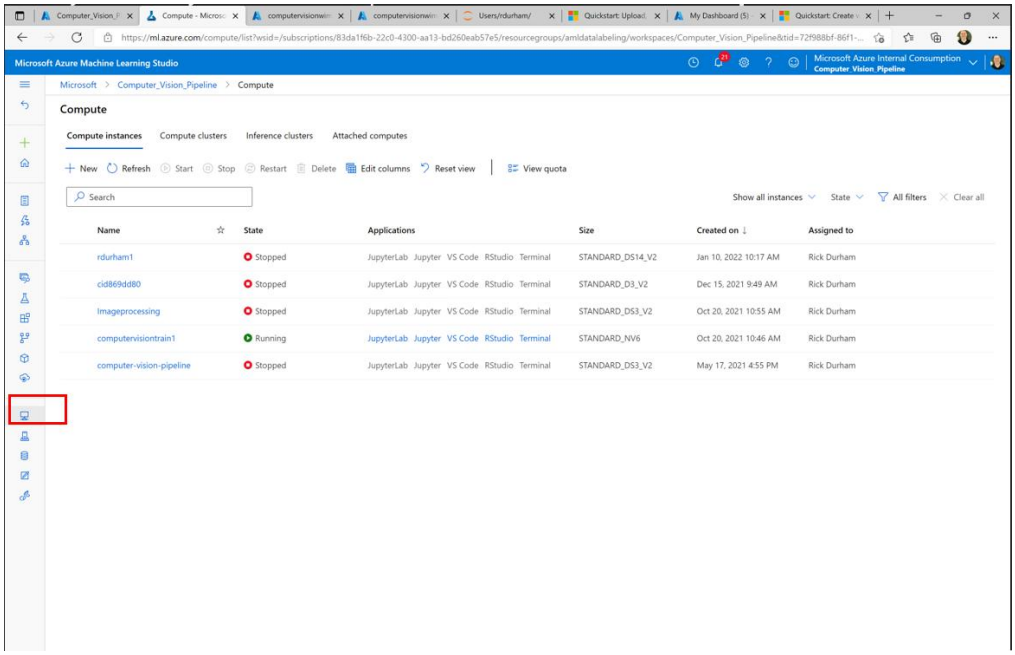
1. If you didn't select **Go to workspace** in the previous section, sign in to Azure Machine Learning studio now, and select your workspace.
2. You can do this by typing in "**Azure Machine Learning**" on the search bar and choosing the Machine Learning icon



3. This will give you a list of AML workspaces to choose from:



4. Once you are in your AML workspace On the left side, select **Compute**.



5. Select **+New** to create a new compute instance.
6. Supply a name like **Imageprocessing**. Please select a NVIDIA GPU instance: **Standard_NC6.** We will use this compute instance a as we execute the notebook and to train our model.
7. Select **Create**.

In about two minutes, you'll see the **State** of the compute instance change from **Creating *to* Running**. It's now ready to go. Now that we have an AML workspace created and a compute instance we are ready to setup our AML Data Store and AML Dataset. This Data Store and Data Set will point back to the image data we uploaded in previous steps.

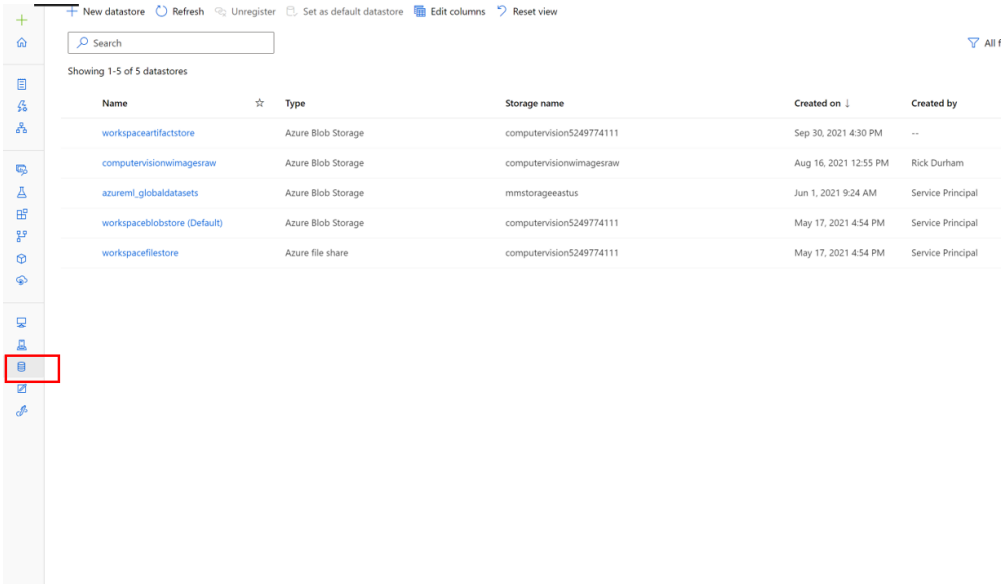
Create a new datastore in a few steps with the Azure Machine Learning studio

Datastores securely connect to your storage service on Azure without putting your authentication credentials and the integrity of your original data source at risk. They store connection information, like your subscription ID and token authorization in your Key Vault that's associated with the workspace, so you can securely access your storage without having to hard code them in your scripts. You can create datastores that connect to these Azure storage solutions.

We are now going to create a new datastore in Azure Machine Learning Studio.

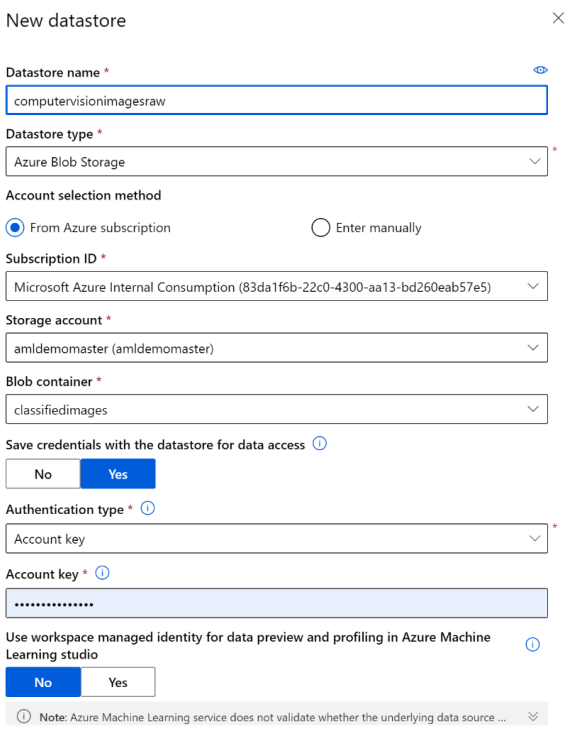
Important

8. Sign in to Azure Machine Learning studio.
9. Select **Datastores** on the left pane under **Manage**.



10. Select **+ New datastore**.
11. Complete the form to create and register a new datastore. The form intelligently updates itself based on your selections for Azure storage type and authentication type. Name the Datastore using a name similar to the one shown below. You will recall we recorded the other form entries as *Storage Account Name, Blob Container Name and Access Key at the beginning of this workshop. You will use them to verify the correct entries for the Storage Account Name, Blob Container Name dropdowns. You will enter the Access key into the text box.*
12. See the storage access and permissions section to understand where to find the authentication credentials you need to populate this form.

The following example demonstrates what the form looks like when you create an **Azure blob datastore**:



Part 2: Create a Labeled Dataset using the Azure Machine Learning Data Labeling Tools

Azure Machine Learning data labeling is a central place to create, manage, and monitor data labeling projects:

Coordinate data, labels, and team members to efficiently manage labeling tasks.

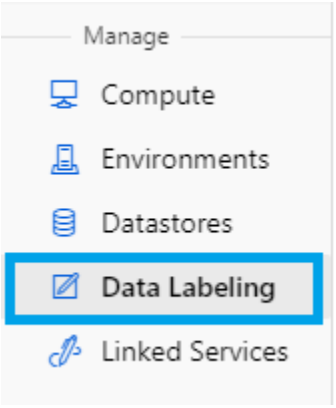
Tracks progress and maintains the queue of incomplete labeling tasks.

Start and stop the project and control the labeling progress.

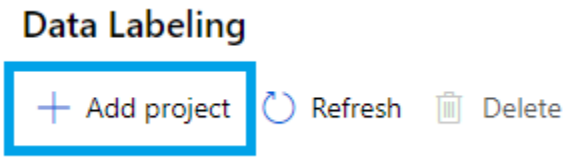
Review the labeled data and export labeled as an Azure Machine Learning dataset.

Here you will use Azure ML to create a dataset of labeled images using images collected on the edge, retrieved from your attached datastore.

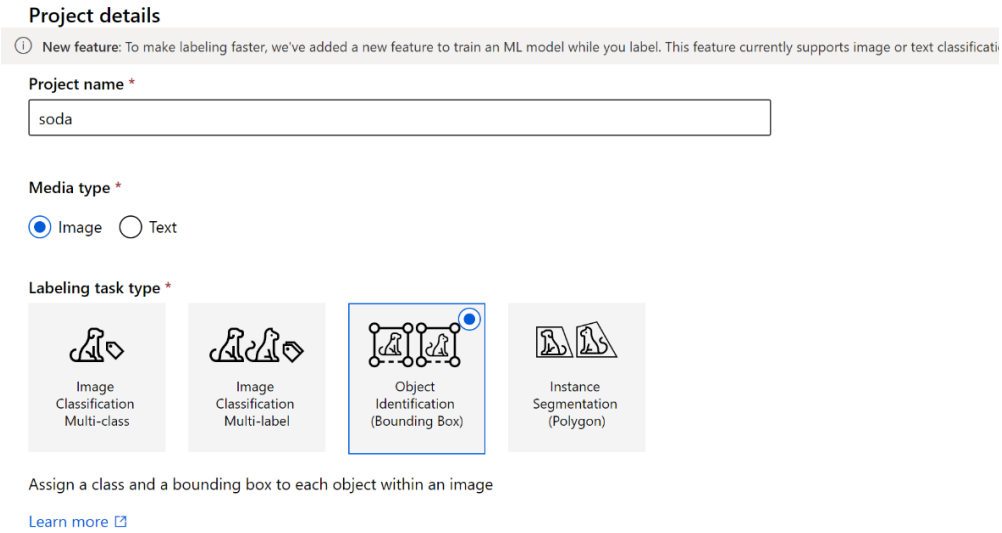
13. Navigate to your Azure Machine Learning workspace and click *Data Labeling*.



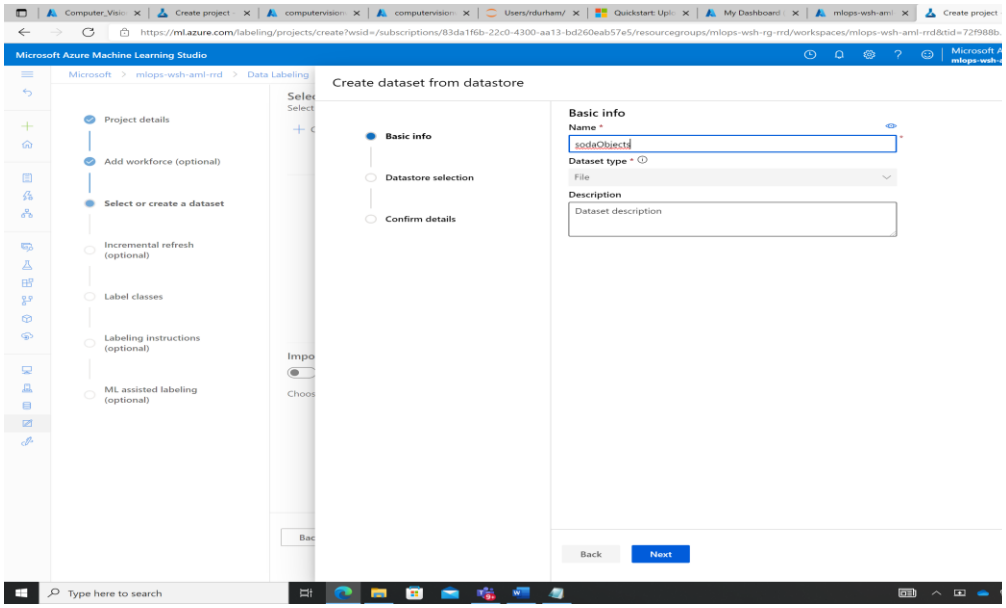
14. From the top menu, click + *Add project*.



15. Under the *Project details* section, give your project a name that is specific to the particular detection task at hand and select *Object Identification (Bounding Box)* from the menu below, then click *Next*.

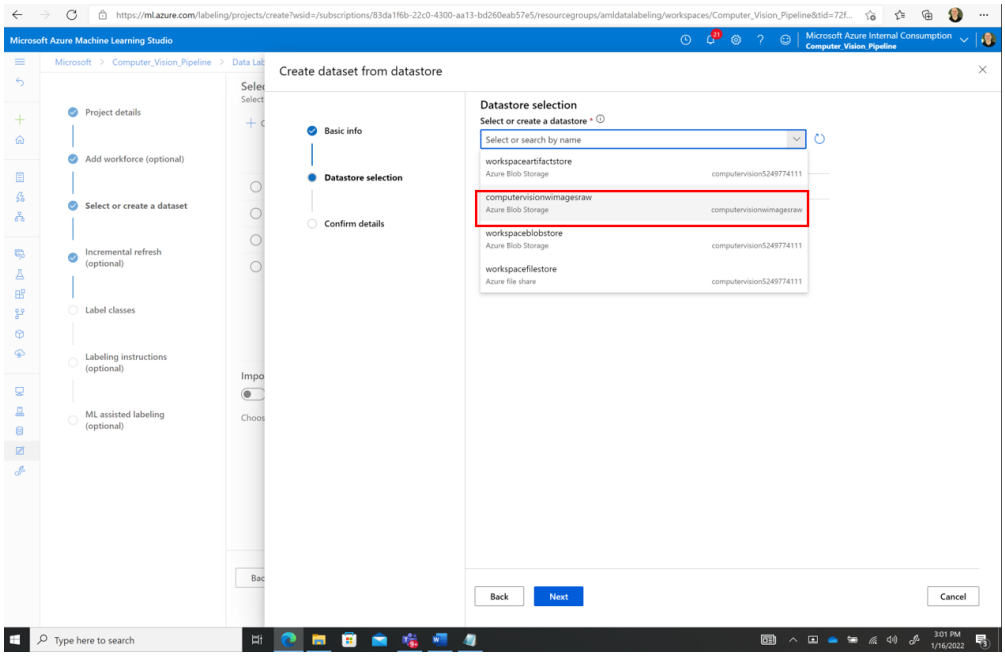


When prompted to “select or create a dataset choose” **+ *Create dataset* and select the *From datastore* option**.



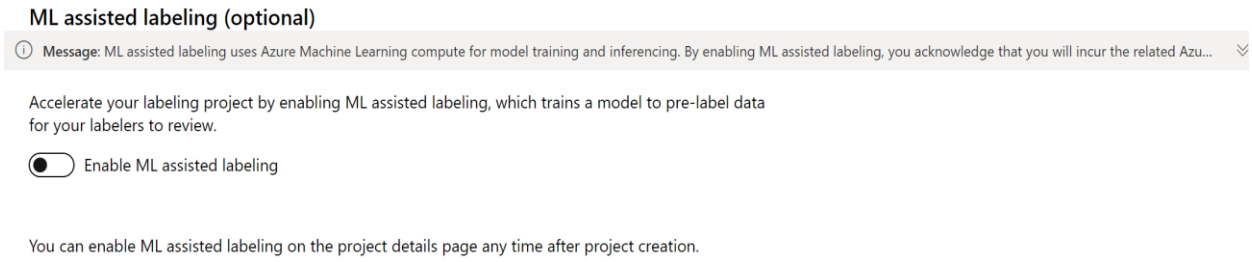
16. Give your new **dataset** the unique name e.g. **sodaObjects** that reflects the images captured in support of the detection task and click *Next*.

17. Under datastore selection choose the **datastore** name that you added previously which contains images captured. Here, you can also provide a wildcarded path if you wish to pull only images from specified partitions. If you wish to pull all images from the container, enter / as the path and click *Next*. Confirm details about your new dataset and click *Create*.

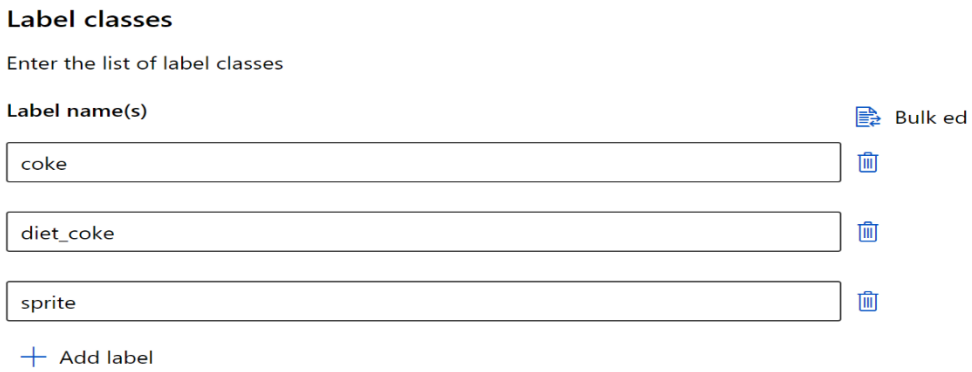


You will be prompted to Enable incremental refresh at regular intervals. This will automatically add newly captured images to your data labeling project.

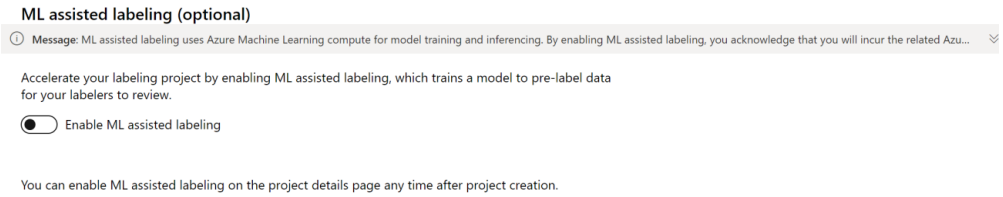
You should **not** use ML-assisted labeling for this lab.



On the next panel, add label classes for all objects or defects you wish to detect - include positive and negative classes here.

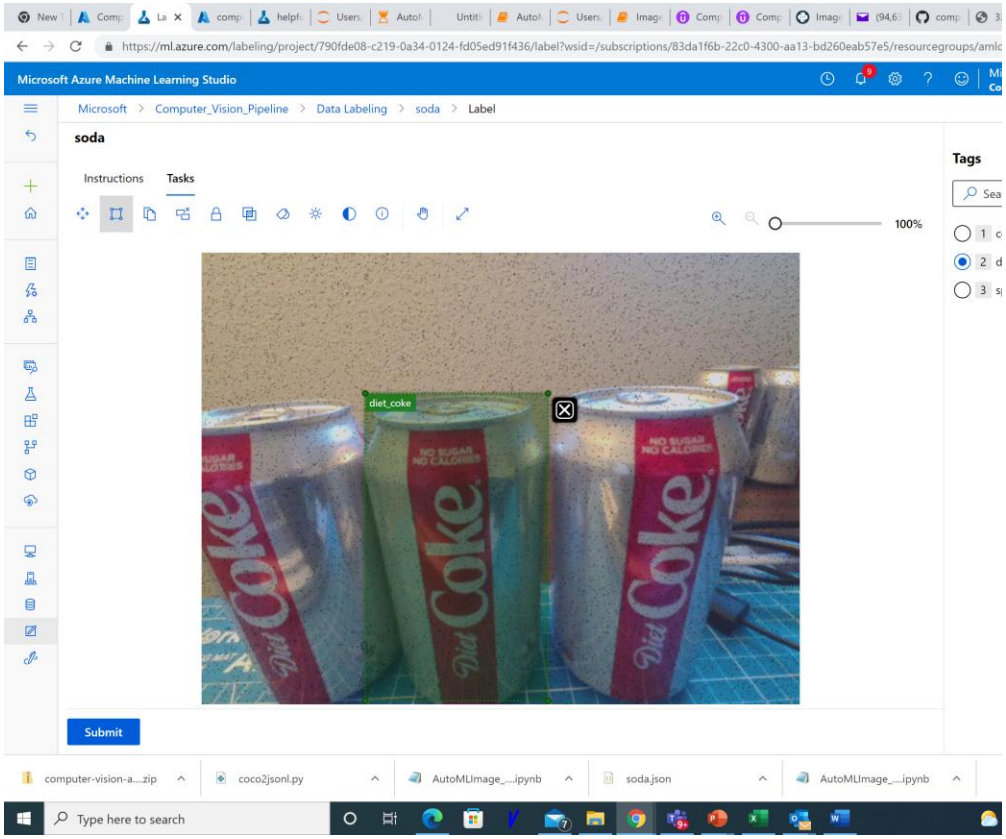


You can **optionally use ML-assisted labeling** which will accelerate your data labeling process, particularly as more data is captured. This option is strongly recommended**. For this course we won't be using this option.**



18. After providing all requested information click *Create project* and wait for the project to initialize.

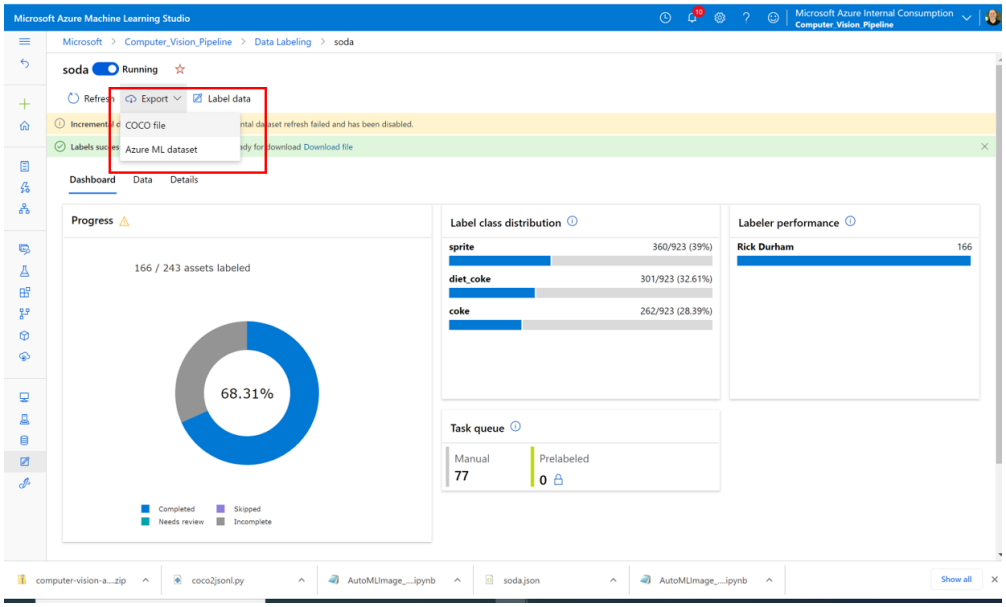
19. Click your newly created project and then select the ***Label*** *data* button. This will open a labeling utility which will allow you to draw bounding boxes and tag objects/defects present in your images. There are multiple keyboard shortcuts available which can be reviewed under the *Shortcut Keys* panel. As you label images hit the submit button and this will save the labelled image.



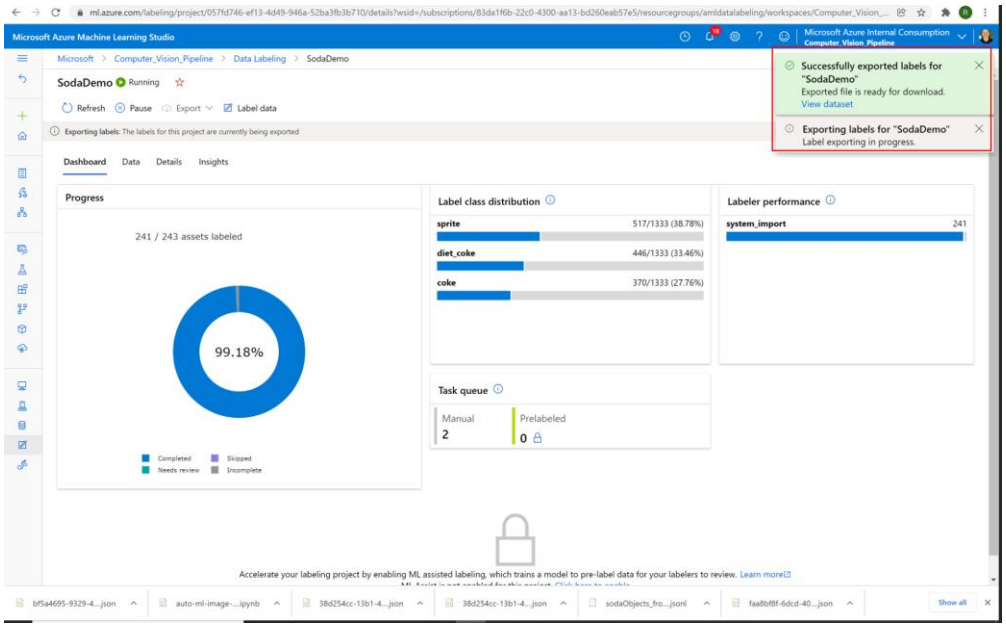
20. The next step after labeling all of your images is to export an annotation file. After labeling a large number of images, navigate to the data labeling project homepage and click *Export* then select *Azure ML Dataset*. This will export your image dataset as an AutoML-compatible Azure ML dataset named according to the format 'NAME_DATE_TIME'.

This is done as follows:

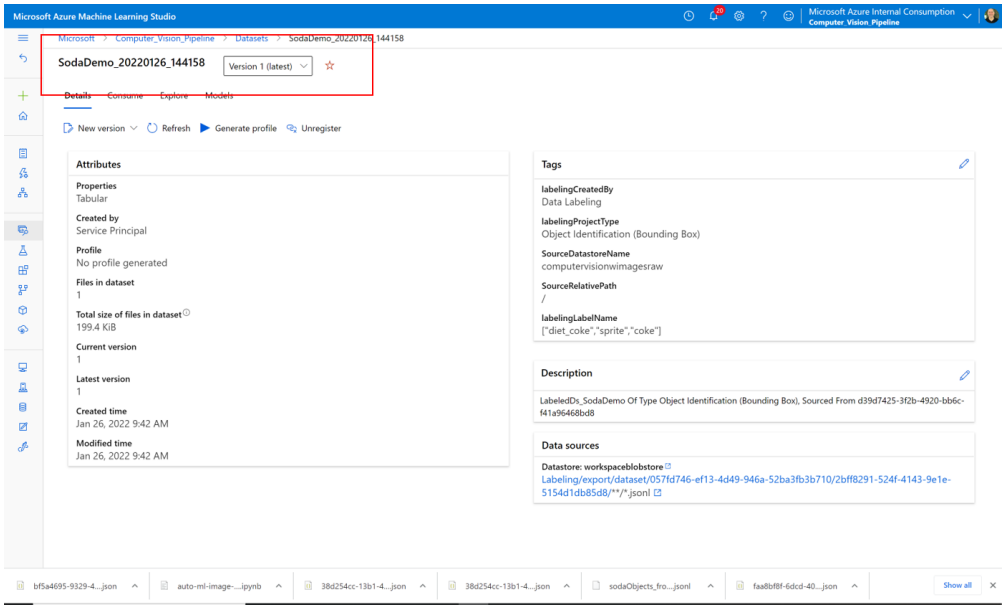
1. Select Export from the Data Labeling Menu
2. Choose Azure ML dataset



3. Click on the **View dataset** link



This will open a view to the newly created dataset. Copy the name of the AML ML dataset in this case it is named " **SodaDemo_20220126_144158"**



Part 3: Use Automated ML to training a computer vision model

Automated ML for images (preview) adds support for computer vision tasks, which allows you to easily generate models trained on image data for scenarios like image classification and object detection.

With this capability you can:

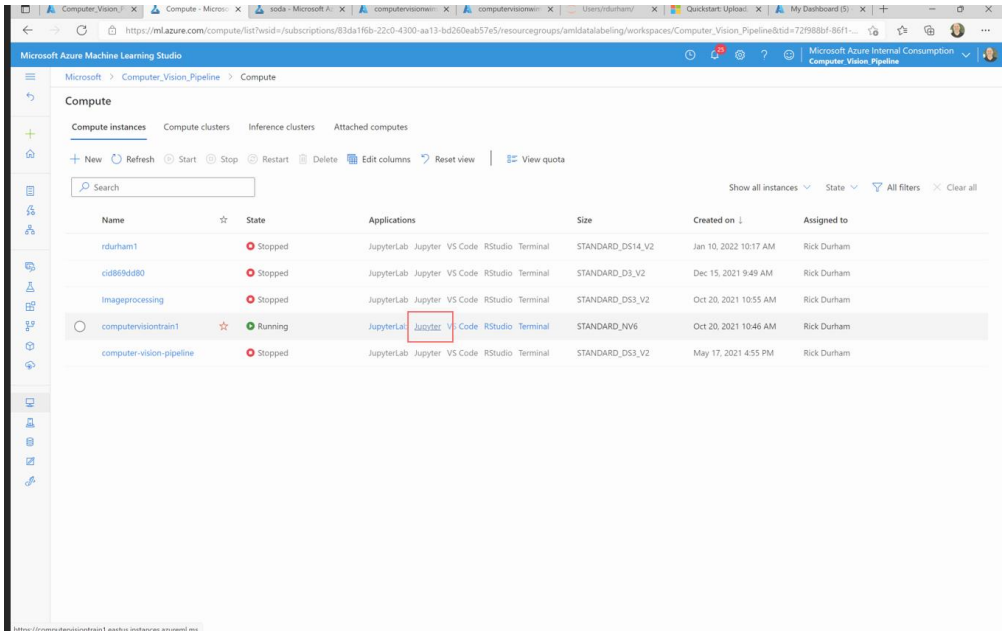
- Seamlessly integrate with the Azure Machine Learning data labeling capability
- Use labeled data for generating image models
- Optimize model performance by specifying the model algorithm and tuning the hyperparameters.
- Download or deploy the resulting model as a web service in Azure Machine Learning.

- Operationalize at scale, leveraging Azure Machine Learning MLOps and ML Pipelines capabilities.
- Authoring AutoML models for vision tasks is supported via the Azure ML Python SDK. The resulting experimentation runs, models, and outputs can be accessed from the Azure Machine Learning studio UI.

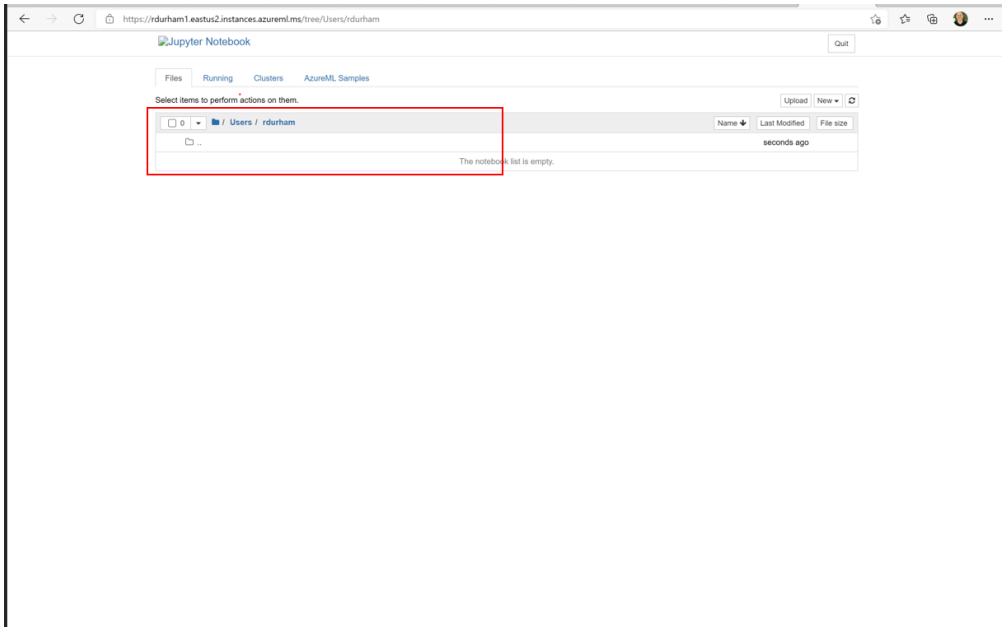
For more information on Automated ML: [What is automated ML? AutoML - Azure Machine Learning | Microsoft Docs]

Open the AML Workspace you created earlier:

4. **Go to workspace** in the previous section, sign in to [Azure Machine Learning studio] now, and select your workspace. Select the compute icon and click on the **Jupyter** link:

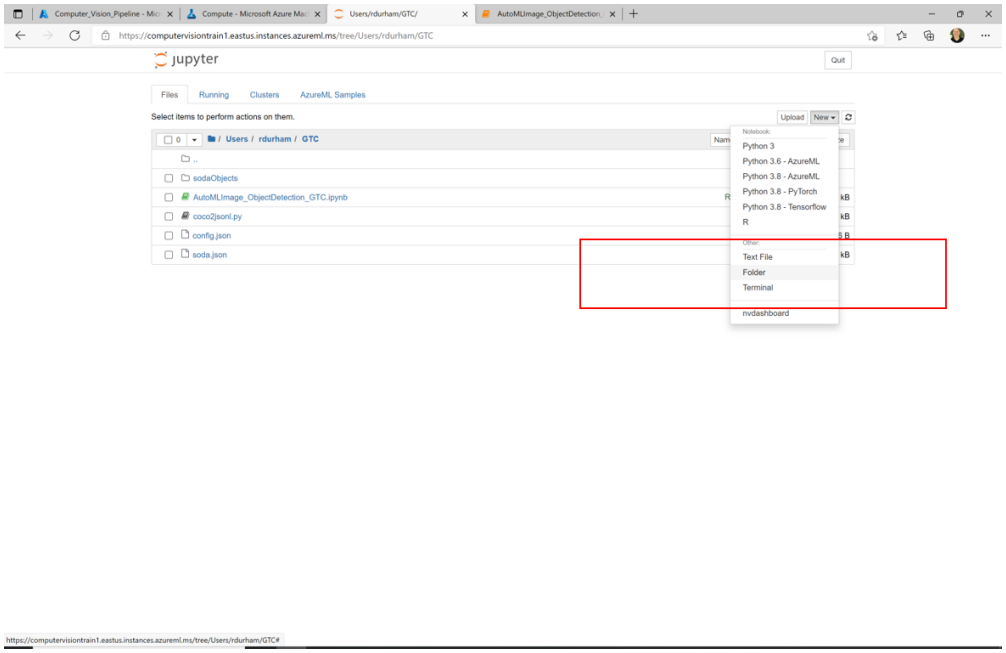


Click on the users directory and select the your user name directory e.g.:

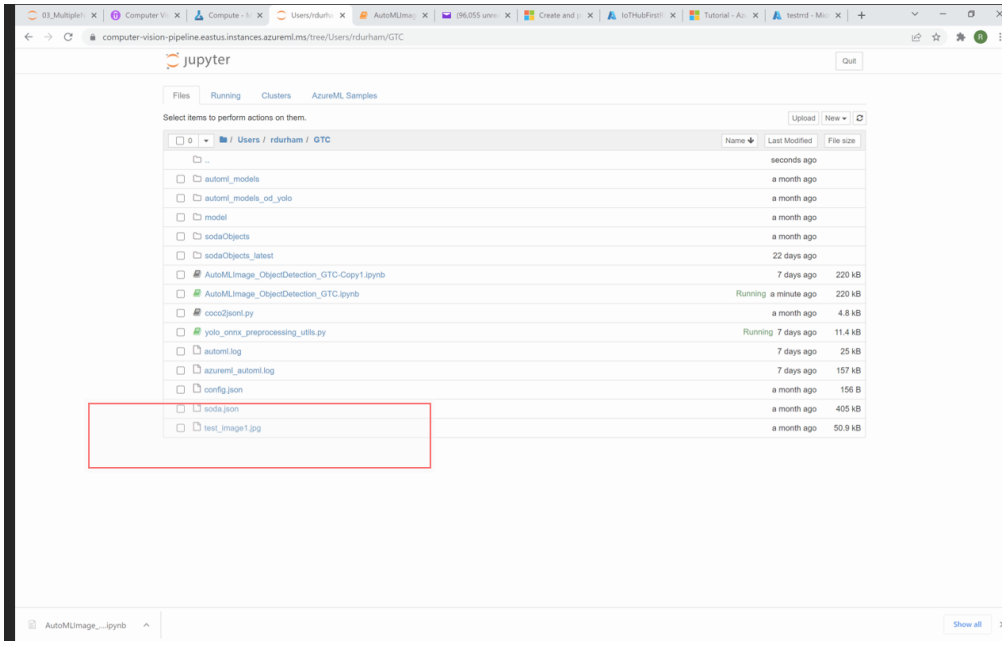


Use the upload buttom (top right) to upload the **auto-ml-image-object-detection_latest-V2.ipynb** Jupyter notebook from the local drive where you stored in the **GTC\ directory**

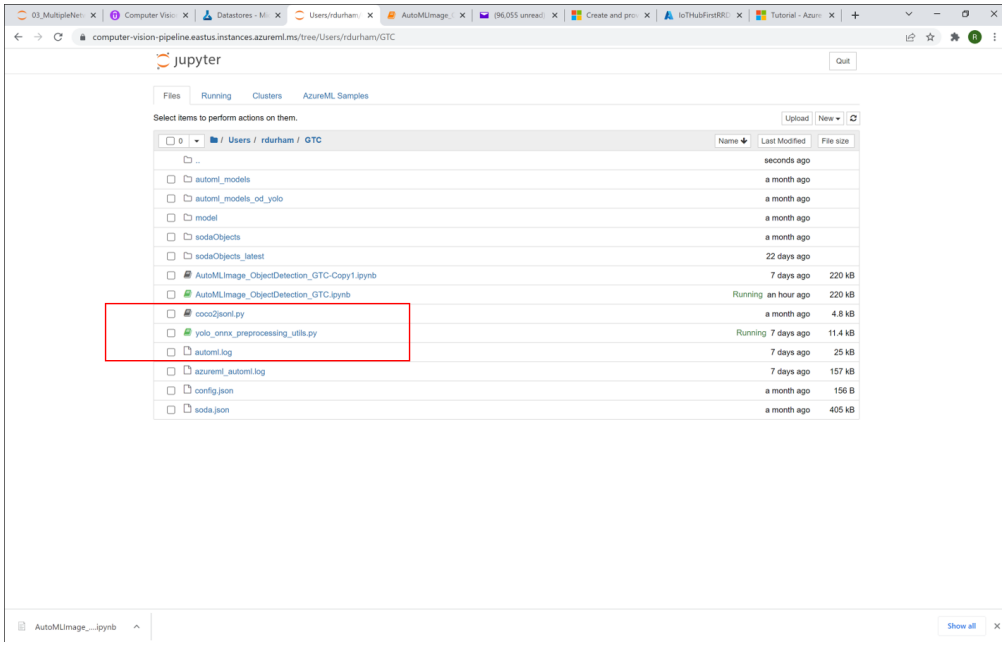
Upload the **config.json** Using the new menu dropdown create a new folder called **sodaObjects**. Once you finished your environment should look like the following:



Upload the file **test_image1.jpg** into the same folder where the notebook resides.

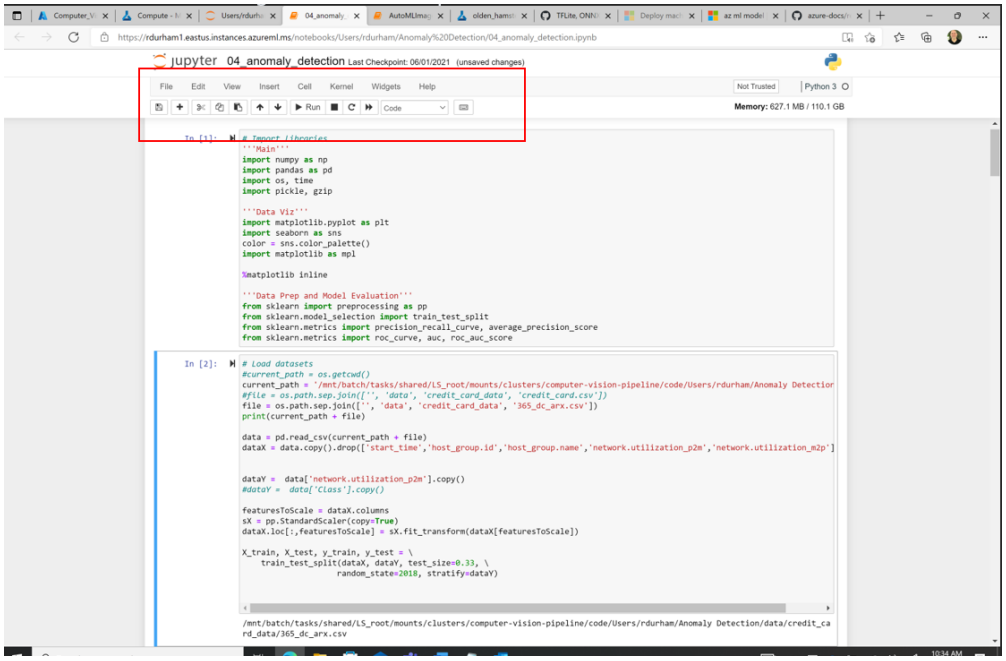


Upload the **yolo_onnx_preprocessing_utils.py** file into the same directory where the notebook is stored:



Execute the AutoMLImage_ObjectDetection_GTC notebook:

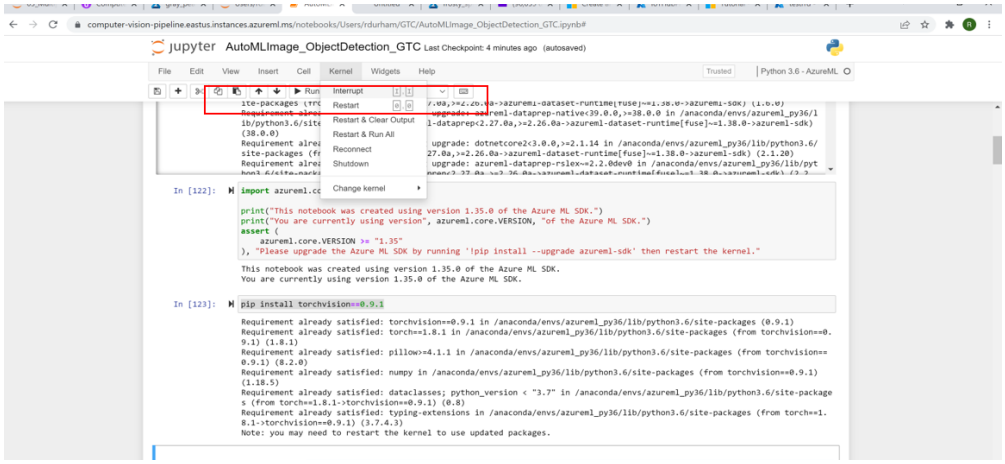
5. Click on the **auto-ml-image-object-detection_latest-V2.ipynb** notebook in the Juptyer environment. You will execute the following cells in order. In order to execute a cell click in the code cell and hold down on the shift key and press enter or execute the run command using the arrow: Example:



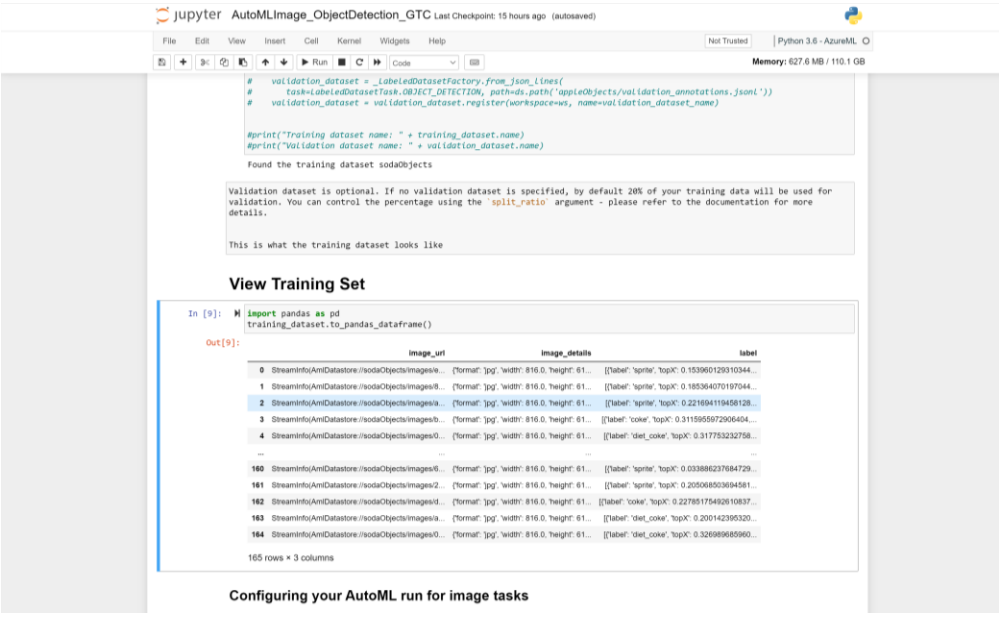
Execute the following Jupyter Notebook cells in order:

Environment Setup

1. Please note after you execute the cell with the code "**pip install torchvision==0.9.1**" you need to restart the Kernel by going to the menu item Kernel and clicking on the "**Restart**" menu item. After this you can execute the next cell pip freeze which will show you all of the python libraries installed.



2. Workspace Setup
3. Compute Target Setup
4. Experiment Setup
5. Dataset with input Training Data (***Please note you will need to replace the name in this step the AML dataset name you recorded earlier** in our example this was called **SodaDemo_20220126_144158**)
6. View Training Set (you should see a dataset as shown below)



Please note that we are using a Compute Instance with a GPU as part of this workshop. As this is an example we didn't create a separate computer cluster.

A compute cluster is an additional compute target that can be used to train larger models with more data. A compute cluster can upscale or downscale the GPU target to align with the workload that could be as few as a few hundred to several thousand images.

Additional you only pay when the nodes are up and not when they are scaled down. For a compute instance you pay per hour even when there is no training happening but is still running.

For more information about compute inside Azure Machine learning:

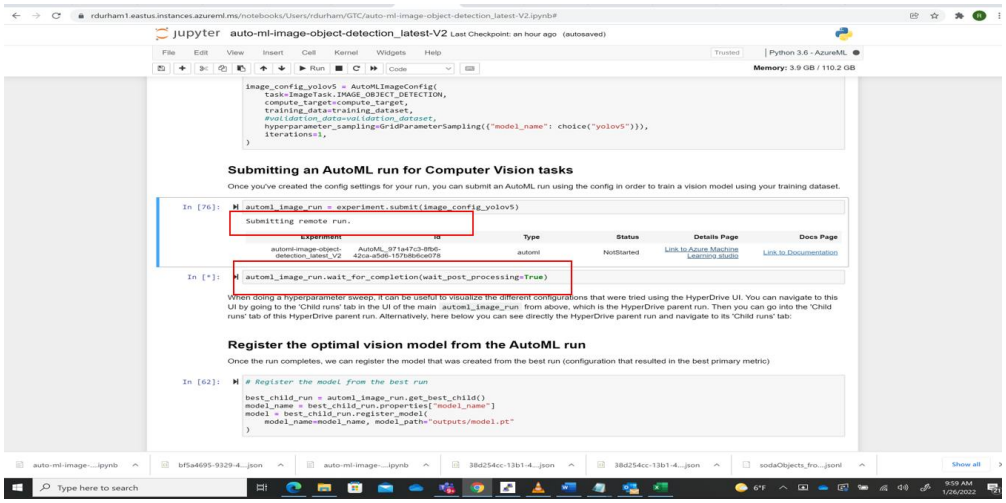
Compute instance: What is an Azure Machine Learning compute instance? - Azure Machine Learning | Microsoft Docs

Compute cluster: Create compute clusters - Azure Machine Learning | Microsoft Docs

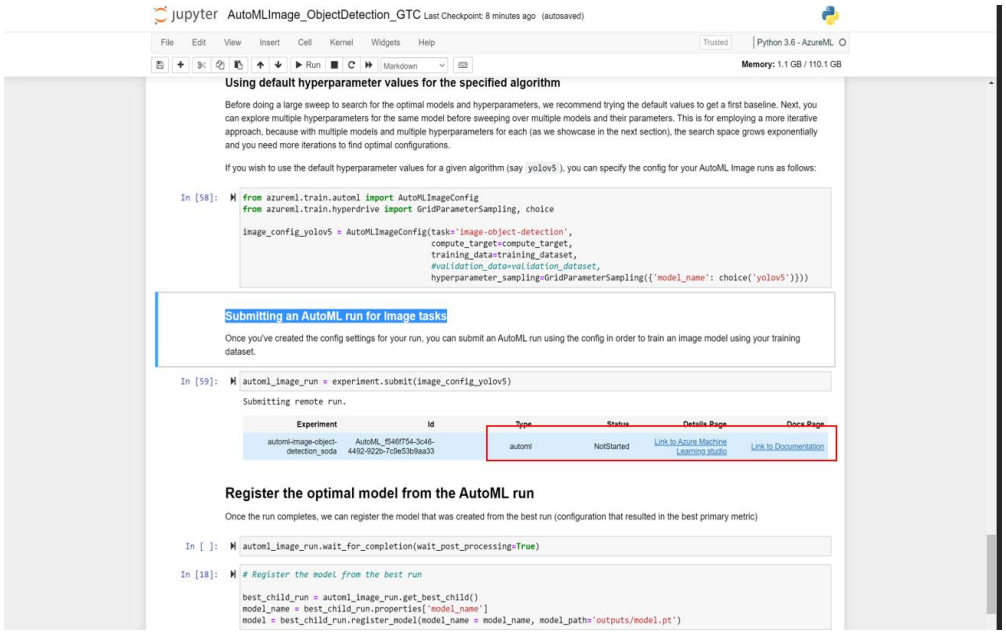
Execute the following Jupyter Notebook cells in order:

7. Using default hyperparameter values for the specified algorithm.

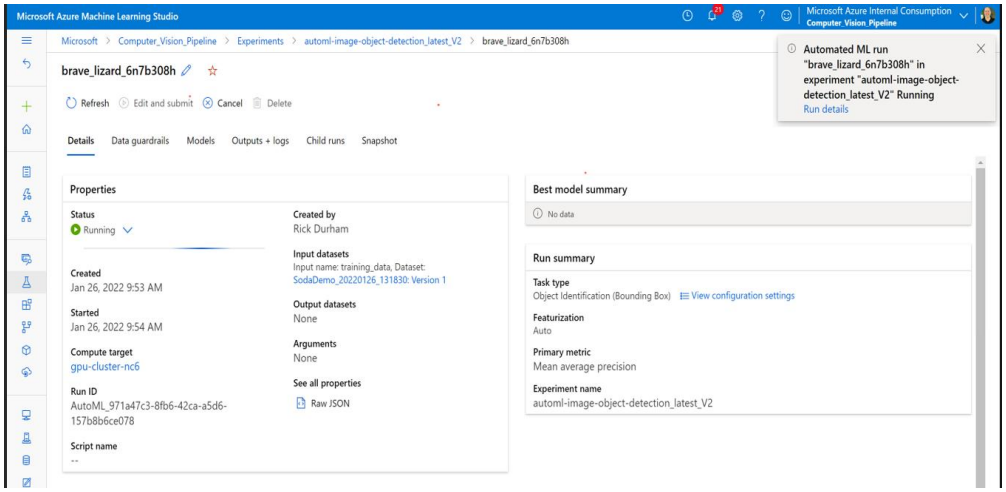
8. **Submitting an AutoML run for Computer Image tasks**. This will start the AML training job and provide a link to the AML workspace to monitor the job. Additionally, execute the next cell which will stay in a holding state until the AML job finishes



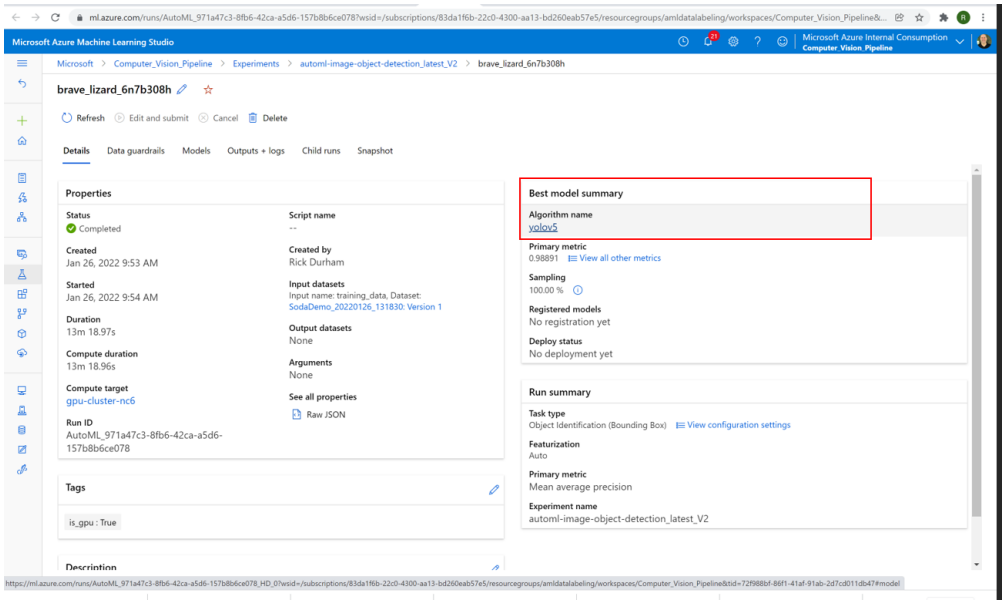
If you want to view the model training in AML click on the Details page shown below



It should be in a "**Running**" state.

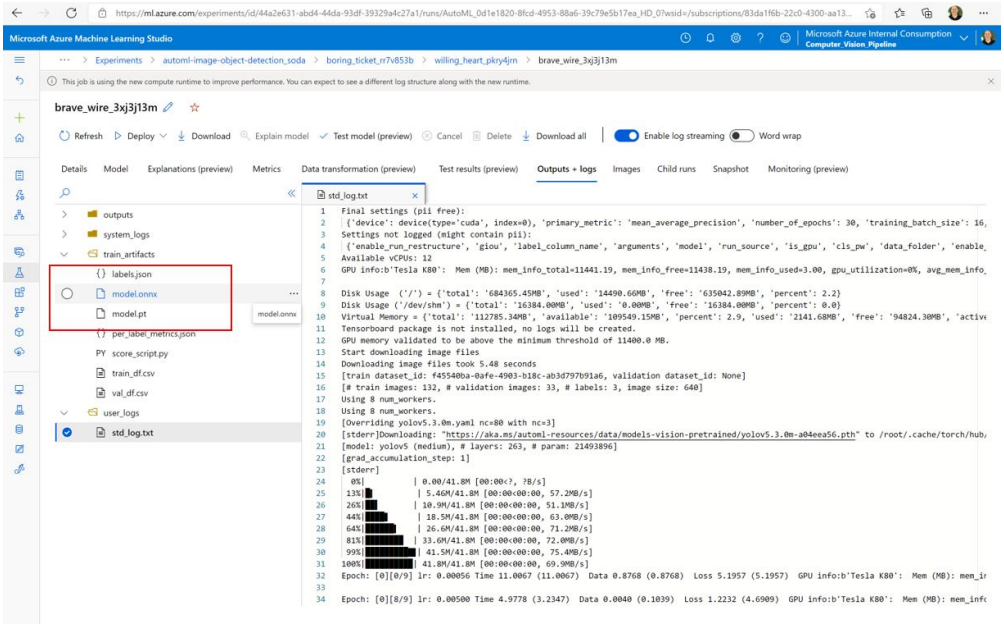


Once the model training job finishes executing you should see the following screen. By clicking on the **Algorithm name** in the **Best model summary** we can see the results of the model training:



Clicking on **Outputs + Logs** and opening the training_artifacts folder will show you the model outputs in terms of the model and the label files:

Please note we see the *label file* (**labels.json**) the *yolo5 weights file* (**model.pt**) and the *ONNX model* (**model.onnx**) in this folder



Switch back to the jupyter notebook

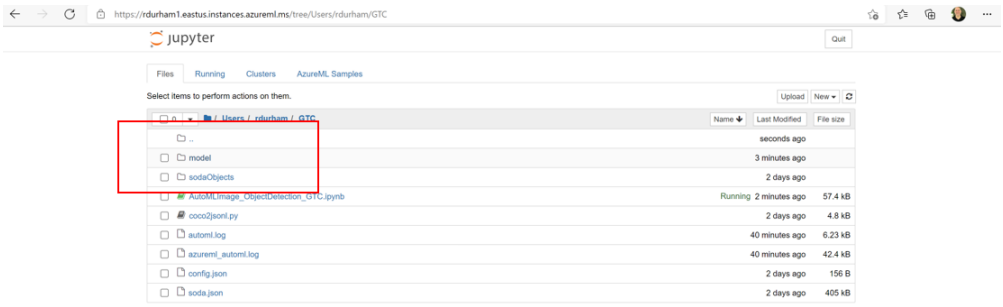
Once we have a trained model in AML we can register the model by executing the following cell in the Jupyter notebook:

Register the optimal model from the AutoML run.

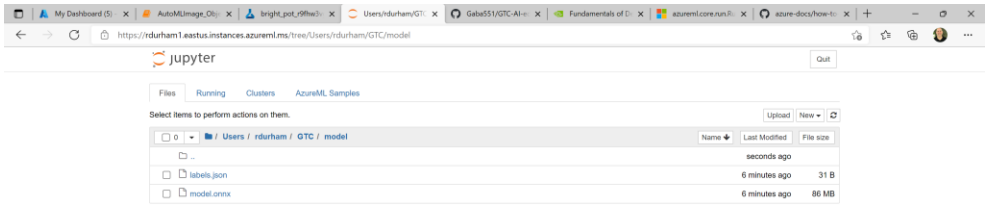
Now that we have the model registered we are ready to download the ONNX model and the associated label file by executing the cell:

Download the model and other associated files e.g. labels

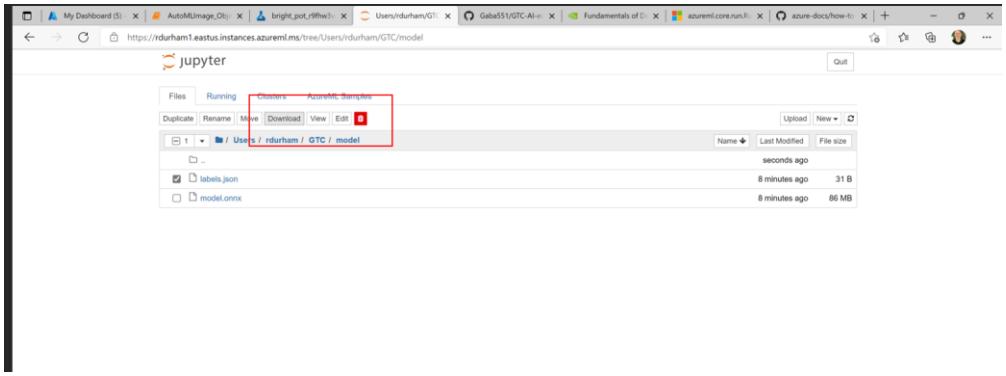
This step will create a directory in Jupyter called model and put the onnx model and label files in that directory:



If we click on the model folder we can see the following two files:

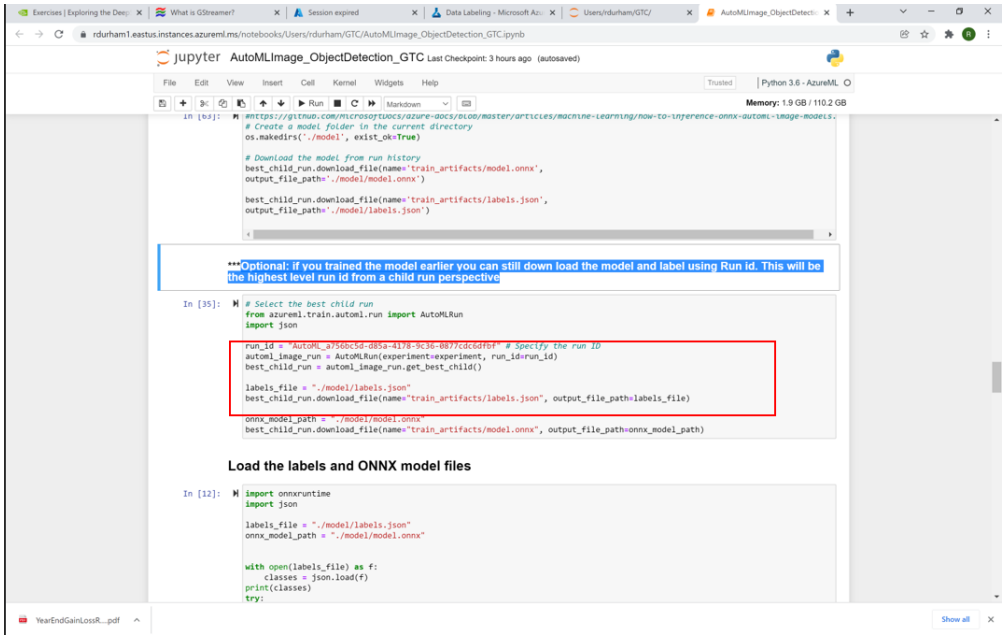


We can click on the text box and download each file to our local workstation.



Now we have the label and model files needed to deploy to our device. We will use these later for our deployment to the target environment.

If for some reason you need to retrieve your model and label files after your training run is over or you reopen the jupyter notebook later you can always execute the "**Optional: if you trained the model earlier you can still download the model and label using Run id. This will be the highest level run id from a child run perspective**" cell. You will need to retrieve the **run_id** for the experiment to get the best model and labels file and replace the one in the sample notebook (see below).



Now let's take a look at how to use the model and the labels file to inference a test image:

Execute the **Load the labels and ONNX model files** cell. This will load the classes and session objects. Review the output results:

```
['coke', 'diet_coke', 'sprite']
ONNX model loaded...
```

Execute the **Get expected input and output details for an ONNX model** cell. This will output show the details of inputs and outputs of the onnx model.

```
No. of inputs : 1, No. of outputs : 4
0 Input name : input, Input shape : ['batch', 3, 640, 640],
Input type  : tensor(float)
```

```
 0 Output name : output, Output shape : ['batch', 25200, 8]
,     Output type  : tensor(float)
 1 Output name : 1397, Output shape : [1, 3, 80, 80, 8],
Output type  : tensor(float)
 2 Output name : 1745, Output shape : [1, 3, 40, 40, 8],
Output type  : tensor(float)
 3 Output name : 2093, Output shape : [1, 3, 20, 20, 8],
Output type  : tensor(float)
```

Execute the cells one-by-one following the **Image Inferencing Preprocessing** cell. These cells will perform the following

inferencing functions:

- Add helper functions and prediction functions
- Set the test image path that points to the test image
- Get predictions from the onnx model
- Invoke the **non_max_suppression** function to remove unwanted lower confidence bounding boxes
- Get the bounding boxes, labels and scores for each object in the image
- Plot the image with its respective objects, bounding boxes, labels

Part 4: Deploy the model onto Triton Server

Triton is multi-framework, open-source software that is optimized for inference. It supports popular machine learning frameworks like TensorFlow, ONNX Runtime, PyTorch, NVIDIA TensorRT, and more. It can be used for your CPU or GPU workloads.

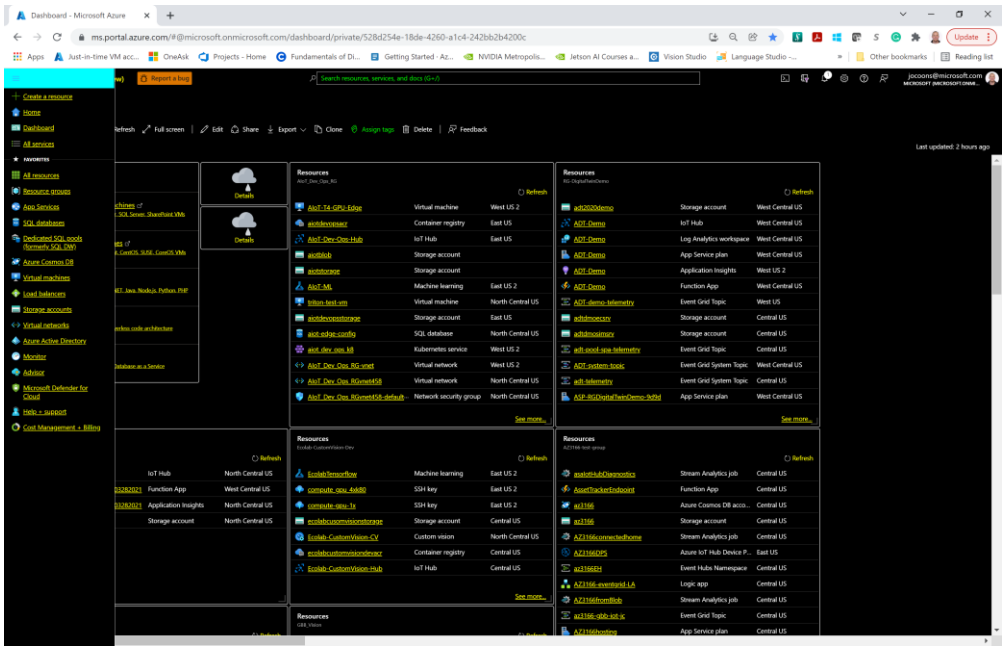
For more information how to use Triton Server in Azure Machine Learning:

[High-performance model serving with Triton (preview) - Azure Machine Learning | Microsoft Docs]

Create a VM:

1. Create a non-GPU VM in Azure (you can also use a dGpu-based machine if you have access to one) To create a VM, you'll need an Azure subscription. If you don't already have a subscription, create a free account before you begin.

2. Log into the Azure Portal, from there you can click I the upper left corner of the screen to bring up the menu, and then click add resource. From there you can either choose Virtual Machine, or the Ubuntu 18.04 Server VM option.



3. In the 'Create a virtual machine' blade, you will select your subscription, select or create your Resource Group, create a friendly name for your VM, choose the Region and availability options. Since this is temporary deployment, chose 'No infrastructure redundancy required'.

4. Since we will be accessing this VM in the workshop remotely, we do want to enable SSH access over Port 22. Under Administrator Account -> Authentication type, choose Password and enter a username and password. We'll use these quite often throughout the workshop, so please make a note of these or create a temporary copy of each in your text editor of choice.

5. Under 'Public Inbound Ports' allow selected ports should be already selected with SSH(22) as the selected port.

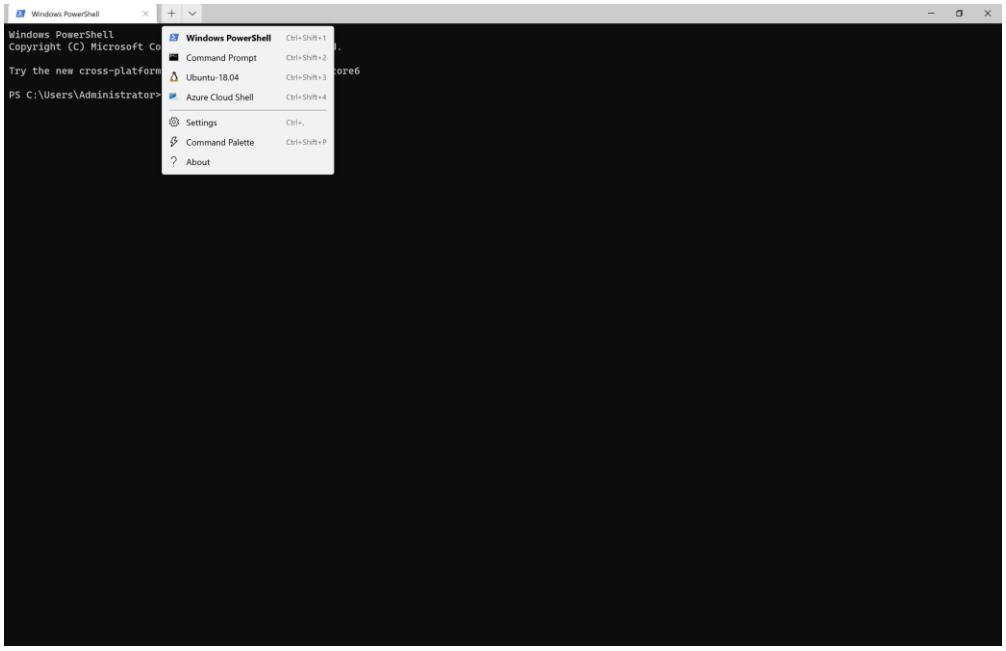
6. Since the aim is to use the simplest VM available, we can skip the additional setup blades you would normally go through, and just select the 'Review & Create' button. Once validated by the system, select 'Create' at the bottom of the screen.

7. This will now create several resources: the virtual machine, a network security group and public IP addresses. When provisioning is complete, select the 'Go to resource' button at the bottom.

8. Copy the Public IP Address in the Overview blade, and save this to your text editor of choice. We'll use this for accessing the VM remotely via a terminal emulator, i.e. TeraTerm or Windows Terminal.

Log into the VM: 1.

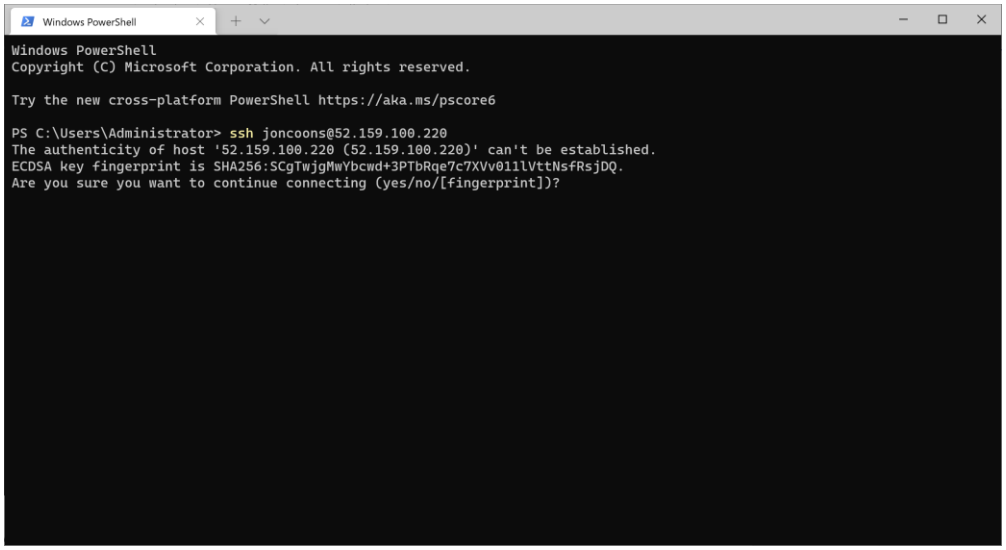
Open your terminal emulator of choice. For illustration, we'll be using Windows Terminal, as it allows for multiple windows to be simultaneously connected concurrently to the VM. We'll be using one window to start the Triton Server, one window to execute a Python script and one to copy images to a directory for processing via the CLI. With Windows Terminal, you also have your choice of CLI experience, PowerShell, Command Prompt, Ubuntu-18.04 (if WSL-2 is installed) or Azure Cloud Shell.



9. Copy the username you used to set up the VM in the previous stage, and in the command line run:

ssh <username>@<your VM IP address>

This will prompt you for the password you saved previously to your text editor. Copy this, and right click in the command line to paste. If logging in for the first time, you'll see the following message:



Type 'yes' and press enter to log into the VM.

10. Now we're going to load in a few packages the Python script needs to execute properly. On the command line, enter:

sudo apt update

sudo apt install -y python3-pip python3-dev nano wget

Prior to installing the Python packages required, we'll want to add '/home/<your username>/.local/bin' to the PATH by taking the following steps in the CLI:

sudo nano ~/.bashrc

Arrow to the bottom of this file in the editor, and add the following line:

export PATH=/home/<uname>/.local/bin:$PATH

Press Ctrl + O and press enter to save the file, then press Ctrl + X to exit. On the command line, run:

source ~/.bashrc

This will reload the configuration for the server to include .local/bin in the PATH.

Now that we've loaded the Ubuntu package requirements and added the directory to PATH, we're going to install the required Python packages. Copy each line individually to run in the terminal window.

```
python3 -m pip install --upgrade pip wheel setuptools

python3 -m pip install numpy>=1.19.0 opencv-contrib-python-headless tritonclient geventhttpclient

python3 -m pip install torch torchvision pandas tqdm PyYAML scipy seaborn requests pybind11 pytest protobuf objdict onnxruntime
```

If you are using a Nvidia GPU-capable VM, you can use onnxruntime-gpu instead of onnxruntime to take advantage of the CUDA/cuDNN acceleration.

11. To run the Triton Server container from Nvidia, we're going to need a container engine If using a Nvidia GPU-based VM, nvidia-docker2 is the correct runtime, and this step can be skipped. For the non-GPU VMs, however, we'll utilize Moby, which is the OSS on which Docker is based. Microsoft has a distribution of this container runtime which can be installed using the following commands:

```
wget https://packages.microsoft.com/config/ubuntu/18.04/multiarch/packages-microsoft-prod.deb -O packages-microsoft-prod.deb

sudo dpkg -i packages-microsoft-prod.deb

rm packages-microsoft-prod.deb
```

Now we can install Moby:

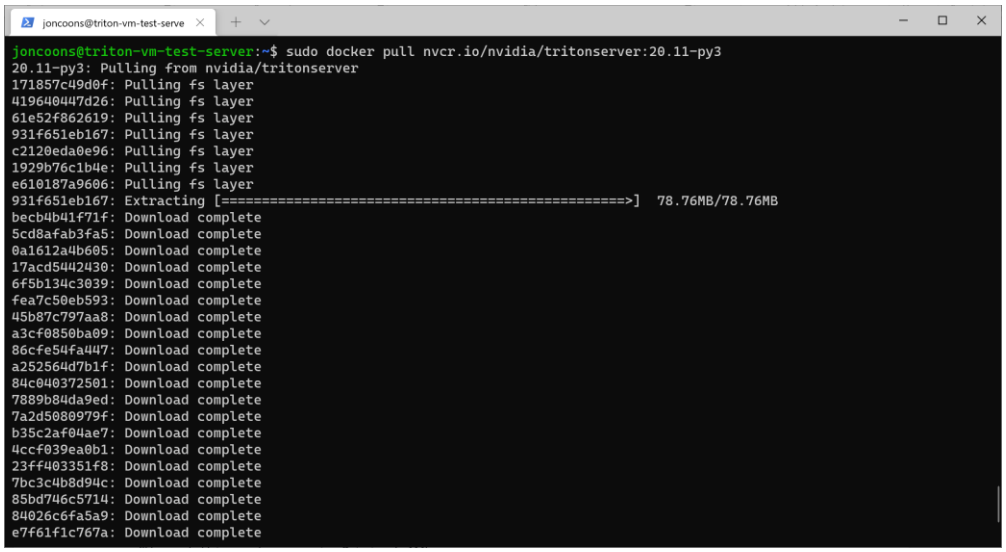
```
sudo apt update

sudo apt install -y moby-engine
```

sudo apt update

12. Now, we're ready to pull the container for the Triton Server from the Nividia NGC repository. You can also pull the container during the 'docker run' statement when we get to that step, but for simplicity, we'll do the pull now. In the terminal emulator, run:

sudo docker pull nvcr.io/nvidia/tritonserver:20.11-py3

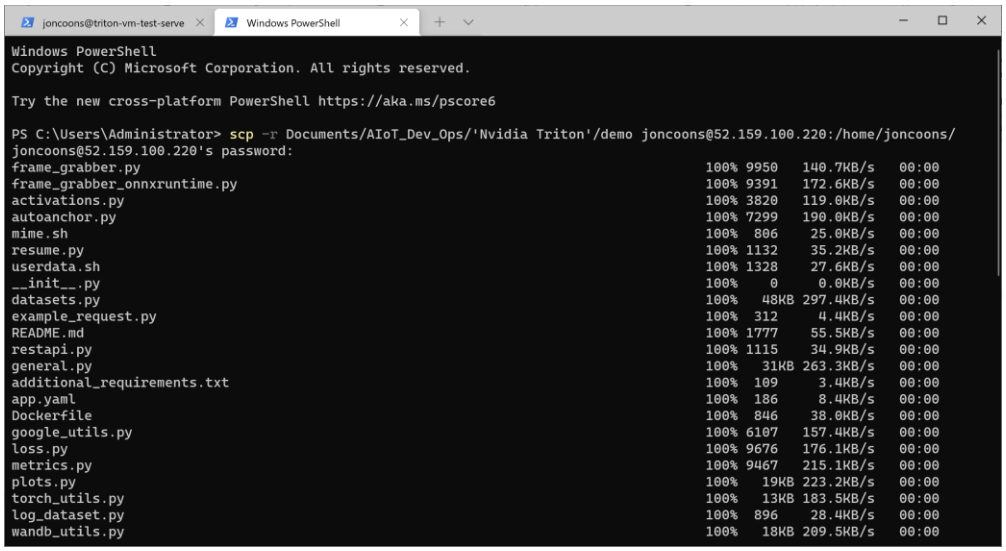
This will take some time to download the container layers and extract them.



13. Now we're ready to copy the 'demo' directory over to the VM. If you download the demo.zip file from the repository, unzip this locally on your PC. Open a command prompt window, either in the utility, or open another window in Windows Terminal. Depending on where you unzipped the files, run the following command in the CLI:

scp -r <path to unzipped>/demo <vm username>@<x.x.x.x vm IP address>:/home/<vm username>/

Here is an example:



14. Once we have this copied over to the VM, let's switch back over to the terminal window connected to the VM and set the permissions for this directory. In that CLI, enter:

sudo chmod -R 777 demo

Now we're ready to run the example Python script on the Triton Server. If you look in the 'demo' directory, you'll see a number of additional folders and files.

In the 'app' folder, there are two Python scripts – frame_grabber.py that uses the Triton Inference Server, and frame_grabber_onnxruntime.py that can be used standalone. The 'utils' folder inside of the 'app' directory contains python scripts to enable the interpretation of the model's output tensor.

Both python scripts are set to *watch* the 'image_sink' directory for any image files that are placed there. In the images-sample, you'll find a number of images we will copy via command line to the 'image_sink' for processing. The python scripts automatically delete the files from the 'image_sink' after the inference has been completed.

In the model folder, you'll find a folder for the name of the model, which holds the model configuration file for the Triton Inference server, as well as the label file. Also included is a folder denoting the version of the model, which contains the ONNX model that the server uses to inference.

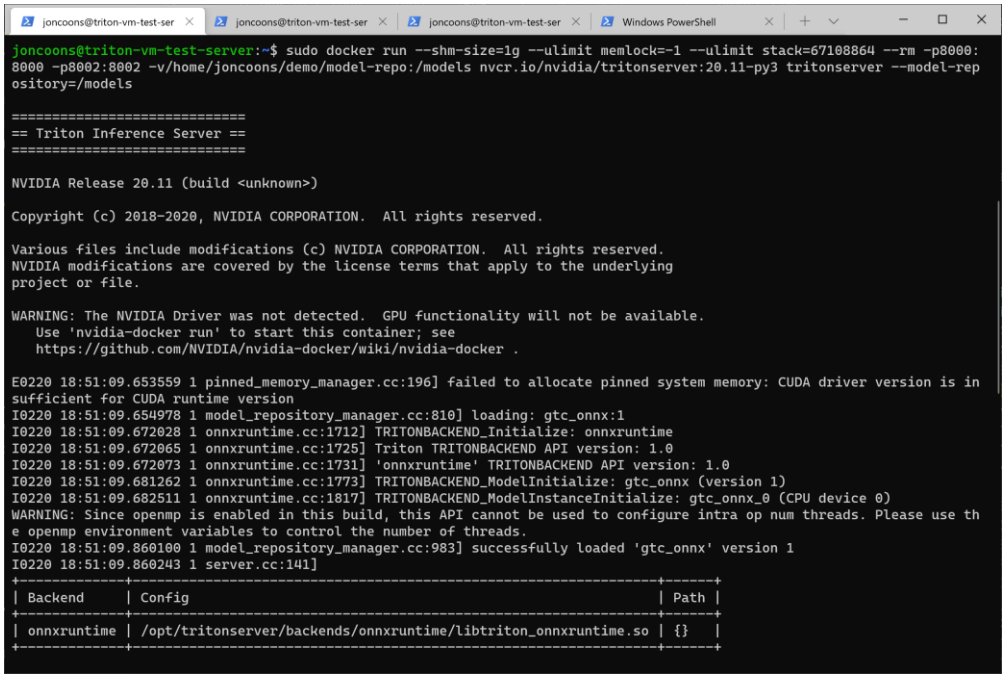
If the model detects the objects it was trained on, the python script will create an annotation of that inference with a bounding box, tag name and confidence sore. The script saves the image into the 'images-annotated' folder, using a unique name using a timestamp, which we can download to view locally. That way, you can copy

the same images over and over again to the 'image_sink' but have new annotated images created each run for illustration purposes.

To get started on the inferencing, we'll want to open two additional windows in the Windows Terminal, and ssh into the VM from each window.

15. In the first window, run the following command, but first change out the username place holder with the username for the VM:

sudo docker run --shm-size=1g --ulimit memlock=-1 --ulimit stack=67108864 --rm -p8000:8000 -p8002:8002 -v/home/<vm username>/demo/model-repo:/models nvcr.io/nvidia/tritonserver:20.11-py3 tritonserver --model-repository=/models



16. In the second window, copy the following command, changing the <vm username> to your value, and set the <probability threshold> to your desired confidence level between 0 and 1 (by default, this is set to .6)

python3 demo/app/frame_grabber.py -u <vm username> -p .07

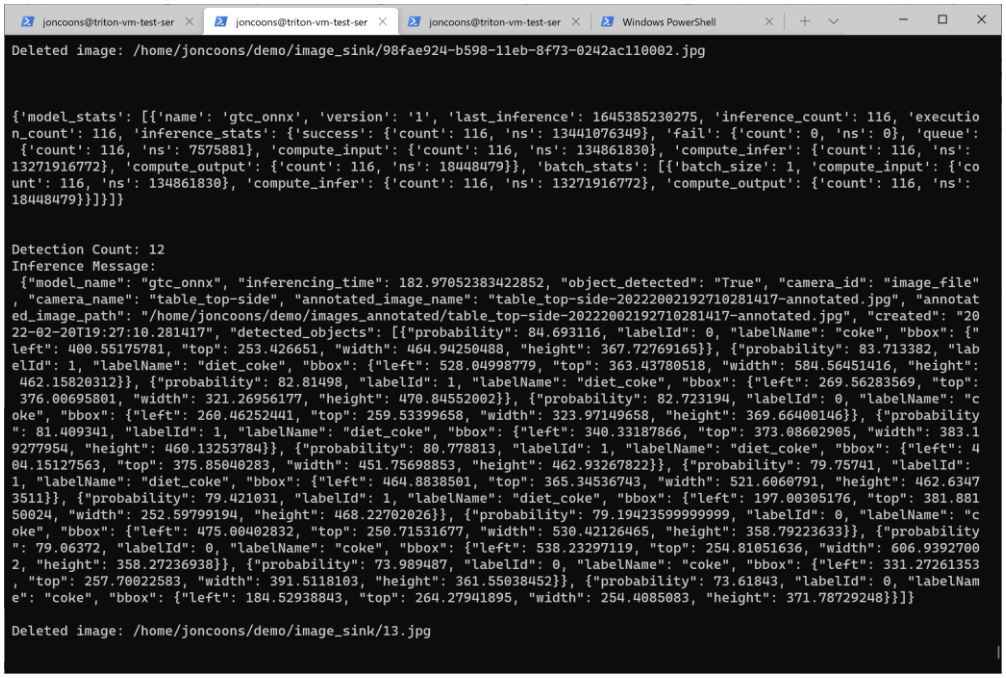
17. In the third window, copy and paste this command to copy the image files from the 'images_sample' folder to the 'image_sink' folder:

cp demo/images_sample/ demo/image_sink/*

If you go back to your second window, you can see the execution of the model, including the model statistics and the returned inference in the form of the following Python dictionary:

```
    {
                            'model_name': self.model_name,
                            'inferencing_time': t_infer,
                            'object_detected': "True",
                            'camera_id': self.camID,
                            'camera_name': f"{self.camLocation}-{self.camPosition}",
                            'annotated_image_name': annotatedName,
                            'annotated_image_path': annotatedPath,
                            'created': created,
                            'detected_objects': result['predictions']
                            }
```

Here is a sample view of what you should see in the second window as the script executes:



18. If you want to see a list of your annotated images, you can run this simple command:

ls demo/annotated_images

19. To download the images to your local machine, we'll first want to create a folder to receive the images. In a command line window, 'cd' to the directory you to place the new folder in, and run:

mkdir annotated_img_download

scp <uname>@x.x.x.x:/home/<uname>/demo/images_annotated/ annotated_img_download/*

This will copy all of the files from the Ubuntu VM to your local device for viewing.

Thank you for joining Microsoft at GTC! We sincerely hope you enjoyed this workshop!