**Model Deployment Using Nvidia Triton Inference Server**

**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](https://azure.microsoft.com/free/?WT.mc_id=A261C142F) 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. (I’m using a VM with the SKU of Standard B4ms, which is the default for Ubuntu Server 18.04)

Text

Description automatically generated

1. 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’.
2. 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.
3. Under ‘Public Inbound Ports’ allow selected ports should be already selected with SSH(22) as the selected port.
4. 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.
5. 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.
6. 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](https://www.microsoft.com/en-us/p/windows-terminal/9n0dx20hk701?activetab=pivot:overviewtab).

**Log into the VM:**

1. Open your terminal emulator of choice. For illustration, we’ll be using [Windows Terminal](https://www.microsoft.com/en-us/p/windows-terminal/9n0dx20hk701?activetab=pivot:overviewtab), 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.

A picture containing background pattern

Description automatically generated

1. 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:

Text

Description automatically generated

Type ‘yes’ and press enter to log into the VM.

1. 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.

1. 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*

1. Now that we have everything set up in terms of pre-requisites, let’s look at the structure of the model repo, and in particular the config.pbtxt. To put this into better context, we’ll use a model visualization utility called [Netron](https://netron.app/). You can either download this as an application, or review the model online. After you open the model, click on the Input box for the model, and the Model Properties will pop up to the right on the screen. While we’ve pre-populated the values for this workshop, these are the values you’ll need when setting up the .pbtxt if you’re building other models. If you would like, you can also scan through the entire model in Netron to look at all of the convolutional layers.

Graphical user interface

Description automatically generated with low confidence

1. 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.

Text

Description automatically generated

1. 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:

Text

Description automatically generated

1. 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*

1. 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.

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*

Text

Description automatically generated

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*

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:

Text

Description automatically generated

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

ls demo/annotated\_images

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.

A picture containing text

Description automatically generated

**Thank you for joining Microsoft at GTC!**

**We sincerely hope you enjoyed this workshop!**