Azure Guide

XCS234 Course Developement Staff

This guide will help you set up and use Azure Virtual Machines. Before you start, it cannot be stressed enough: **do not leave your machine running when you are not using it! Doing so will result in the unwanted loss of Azure credits.**

Contents

1 Virtual Machine Specifications

1.1 High Level Specifications:

In this guide we will be using Azure Lab Services to manage VMs for the XCS234 course. In particular, each VM instance is preconfigured with Linux DSVM (Data Science Virtual Machine) images so you can expect most packages/tools to be installed.

Each VM instance will provide you with access to accelerated hardware via an Nvidia GPU which will be leverage for model training and inference. Below we provide the full set of hardware specifications for your instance:

Hardware Component	Specification
GPU	K80 (memory 11441MiB)
CPU	6 virtual cores
memory	54 G
disk	146G total

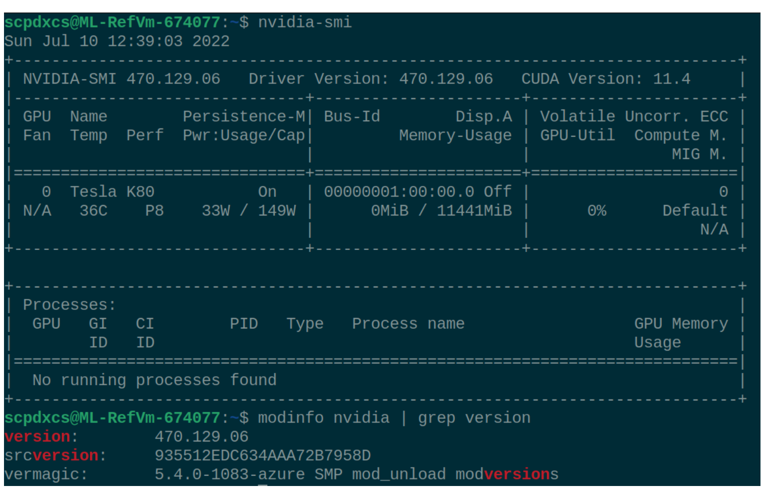
1.2 Detailed Specifications

It is possible to see a more detailed overview of the system architecture in terms of the GPU, CPU, memory and disk space through running the following commands:

GPU: The below command should return information about the Nvidia GPU attached to your instance. For interested readers smi stands for system management interface.

```
$ nvidia-smi
```

The outputs of running this command should look similar to the following:



```
scpdxcs@ML-RefVm-674077:~$ nvidia-smi
Sun Jul 10 12:39:03 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.129.06   Driver Version: 470.129.06   CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           On   | 00000001:00:00.0 Off |                    0 |
| N/A   36C    P8    33W / 149W |      0MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
scpdxcs@ML-RefVm-674077:~$ modinfo nvidia | grep version
version:        470.129.06
srcversion:     935512EDC634AAA72B7958D
vermagic:       5.4.0-1083-azure SMP mod_unload modversions
```

CPU: The below command should return detailed informaton about your system CPU architecture:

```
$ lscpu
```

```
scpdxcs@ML-RefVm-674077:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                6
On-line CPU(s) list: 0-5
Thread(s) per core:    1
Core(s) per socket:    6
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 63
Model name:            Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz
Stepping:              2
CPU MHz:               2596.988
BogoMIPS:              5193.97
Hypervisor vendor:     Microsoft
Virtualization type: full
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              30720K
NUMA node0 CPU(s):     0-5
```

Memory: The below command should return detailed informaton about your system memory usage:

```
$ free -m
```

```
scpdxcs@ML-RefVm-674077:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:          56228         709       53590           1        1928       54933
Swap:             0           0           0
```

Disk Space: The below command should return detailed informaton about your system disk space:

```
$ df -h
```

```
scpdxcs@ML-RefVm-674077:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
udev             28G     0   28G   0% /dev
tmpfs           5.5G  1.1M  5.5G   1% /run
/dev/sdb1       146G   69G   77G  48% /
tmpfs            28G     0   28G   0% /dev/shm
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs            28G     0   28G   0% /sys/fs/cgroup
/dev/loop1      188M  188M     0 100% /snap/storage-explorer/37
/dev/loop2      878M  878M     0 100% /snap/intellij-idea-community/366
/dev/loop0      255M  255M     0 100% /snap/gnome-3-38-2004/106
/dev/loop4       92M   92M     0 100% /snap/gtk-common-themes/1535
/dev/loop3      188M  188M     0 100% /snap/storage-explorer/38
/dev/loop7      128K  128K     0 100% /snap/bare/5
/dev/loop6      566M  566M     0 100% /snap/pycharm-community/281
/dev/loop5      401M  401M     0 100% /snap/gnome-3-38-2004/112
/dev/loop8      566M  566M     0 100% /snap/pycharm-community/286
/dev/loop11      56M   56M     0 100% /snap/core18/2409
/dev/loop12     878M  878M     0 100% /snap/intellij-idea-community/372
/dev/loop10      82M   82M     0 100% /snap/gtk-common-themes/1534
/dev/loop9       47M   47M     0 100% /snap/snapd/16010
/dev/loop13      62M   62M     0 100% /snap/core20/1518
/dev/sdb15      105M  4.4M  100M   5% /boot/efi
/dev/sda1       334G   69M  317G   1% /mnt
/dev/loop14      47M   47M     0 100% /snap/snapd/16292
tmpfs           5.5G     0  5.5G   0% /run/user/1000
```

2 Cloud Credit Management

2.1 Virtual Machine Credits

We are using Azure Lab Services to manage VMs for the XCS234 course. Every student will be allocated 65 hours total for completing Assignments. It's very important for students to manage credit wisely in order to make the most efficient use of it.

2.2 Best Practices for Managing Credits

Azure virtual machines are charged at a flat rate for each minute they are turned on. This is irrespective of:

- whether you are ssh'd to the machine at that time
- whether you are running any processes on the machine at that time
- the computational intensity of the processes you're running
- whether you're using GPUs

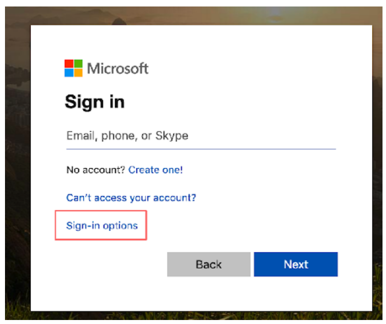
Therefore, the most important thing to consider when managing credit is whether your VM is on or off.

As you will see described in the Assignment 2 handout, we advise you to first develop your code on your local machine until you can complete several training iterations within the test environment without any errors. Once this is achieved we advise running your code on your Azure VM in order to leverage the VM instance's GPU for model training.

3 Access and Setup

3.1 Registration

1. Go to this link: https://labs.azure.com/register/p3jm5scp
2. You'll be presented with a large number of options to register. They are:
   (a) Logging in with an existing Microsoft account using the email/phone associated with it
   (b) Logging in with a Skype account
   (c) If you click 'Sign-in Options' you will also be presented with the option to sign in using your GitHub credentials



Once you've done (a), (b), or (c) - follow any additional prompt instructions (depends on which option you selected) - and you will be registered for the lab!

3.2 Connecting to the VM

1. After signing in you'll be directed to an Azure Lab Services portal where you can view all your virtual machines. Unless you've used Azure Lab Services before, you'll see only one machine along with your remaining hours.

   Click on the 'Stopped' button to start the instance (this will take a few minutes). When it is up and running, it will look similar to the right-hand figure below, importantly the display will show "Running" in the bottom status bar:

2. Click the monitor icon at the bottom right of the tile for your virtual machine instance and you'll be asked to set the instance password (make sure you remember/record this password as you will be asked to enter it when logging into to your VM via SSH).

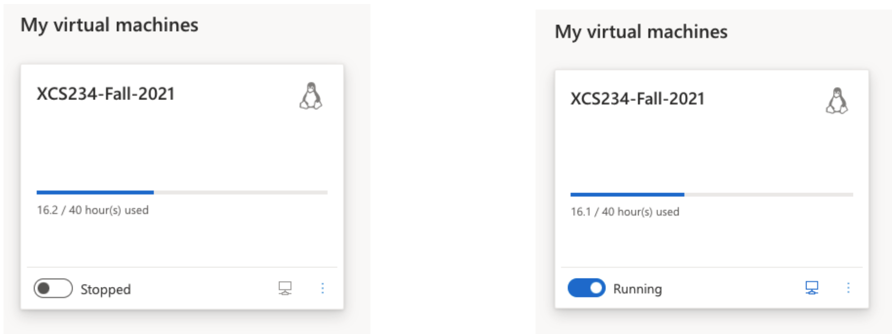
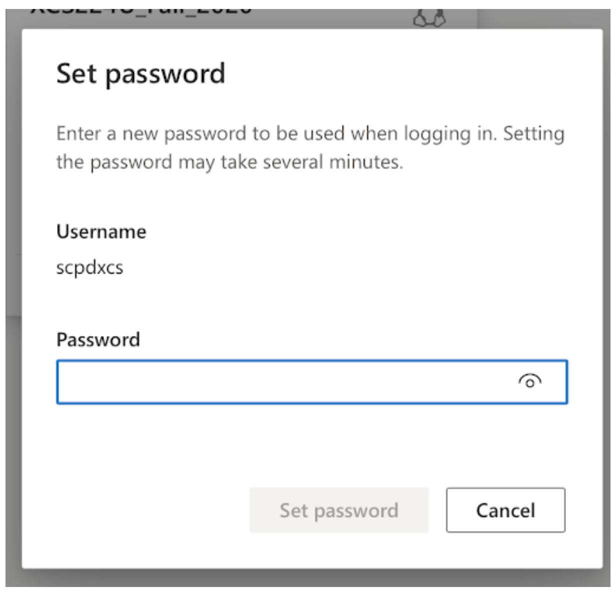
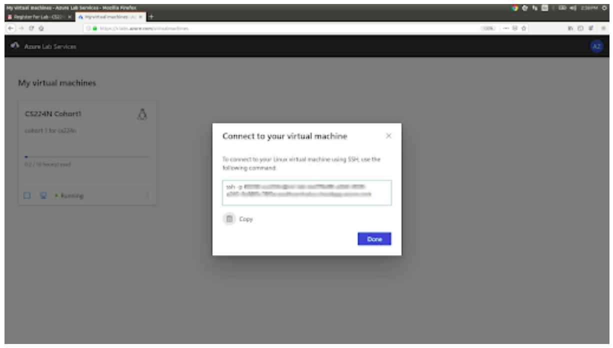


Figure 1: Example of stopped and running virtual machines as seen from the Azure Lab interface

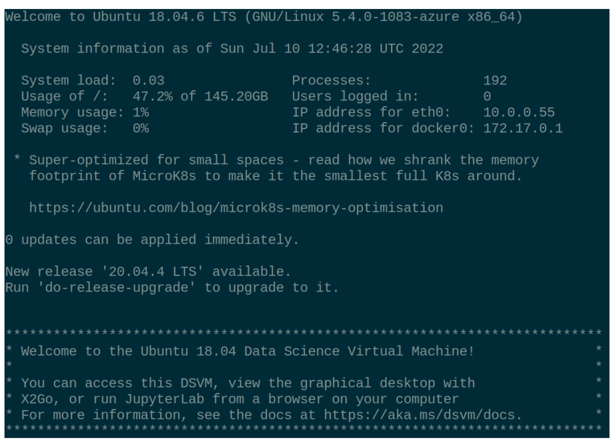


3. Click on the monitor icon and select 'connect via SSH' to get the SSH link.



Note that you could log in from your development machine to the Azure VM without password if you setup the ssh key-based authentication setup, review this link for details.

4. Copy the link and paste it into your terminal (Windows users can use PuTTY). When you have successfully logged in you should see a screen similar to the following:



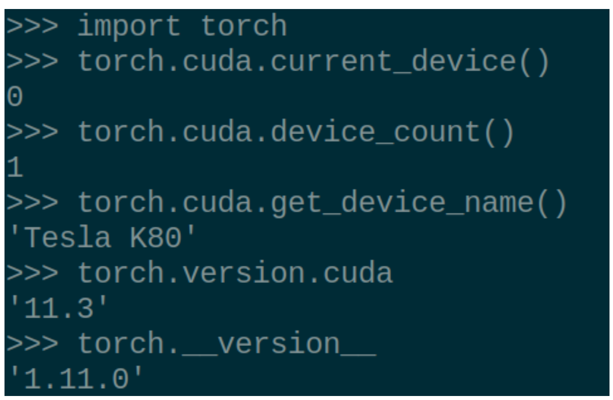
5. Next we wish to ensure that Pytorch can access the GPU available on your machine. To start we will demonstrate this check with one of the preinstalled environments, for each assignment you should perform the same check with the associated assignment conda environment (e.g. `XCS234_A2_GPU` environment). Start by activating the environment being tested:

   ```
   $ conda activate azureml_py38_PT_and_TF
   ```

   Open an interactive Python prompt by typing python into the command line. Python should greet you by letting you know that it's running Python 3.8.5. Into the Python prompt, type each of these lines and then press Enter:

   ```
   import torch
   torch.cuda.current_device()
   torch.cuda.device_count()
   torch.cuda.get_device_name()
   torch.version.cuda
   torch.__version__
   ```

   You should see something like this:

   If you receive error messages or find that this isn't working, post to Slack and/or reach out to your Course Facilitator.

6. Return to the Azure Lab Services portal, toggle off the virtual machine by selecting the button next to the text "Running". After a few minutes, the status should update to "Stopped." Be sure to do this whenever you are not actively running code on the VM.

Setting up a github ssh key could be helpful to pull the code through github, see the details in the Practical Guide for Using the VM section of this document.

As you're working on Assignments 2, instead of activating a preinstalled environment by typing:

```
$ conda activate azureml_py38_PT_and_TF
```

you can instead copy all of the code and files from the assignment repository to your Azure instance, you'll create a new conda environment by using the environment_gpu.yml file that is in the assignment repository through typing (sample: environment_gpu.yml):

```
$ conda env create -f ./environment_gpu.yml
```

This will ensure all of the correct versions of the libraries you'll need for the assignment are installed and create a new environment that you can activate by typing:

```
$ conda activate XCS234_A2_GPU
```

4 Practical Guide for Using the VM

4.1 Managing Processes

In developing your deep learning models, you will likely have to leave certain processes, such as Tensorboard and your training script, running for multiple hours. If you run a process (e.g. model training process) within a given session and you log-off from this session, your process will likely be disrupted. One way around this issue is to detach processes from your current session, in addition to detaching a process it is also often quite nice to be able to be able to manage multiple terminal sessions with different processes at the same time, without having to SSH into the same machine multiple different times. TMUX or "Terminal Multiplexer" is a tool that provides exactly this functionality and can be used to solve the problem we listed above.

TMUX makes it such that in a single SSH session, you can virtually have multiple terminal windows open, all doing completely separate things. Also, you can tile these windows such that you have multiple terminal sessions all visible in the same window. The basic commands are below. Terminal commands are prefaced with a `$` otherwise the command is a keyboard shortcut.

4.2 TMUX Cheatsheet

1. Start a new session with the default name (an integer) `$ tmux`
2. Start a new session with a user-specified name `$ tmux new -s [name]`
3. Attach to a new session `$ tmux a -t [name]`
4. Switch to a session `$ tmux switch -t [name]`
5. Detach from a session `$ tmux detach OR ctrl - b - d`
6. List sessions `$ tmux list-sessions`
7. Kill a session `ctrl - b - x`
8. Split a pane horizontally `ctrl - b - "`
9. Split a pane vertically `ctrl - b - \%`
10. Move to pane `ctrl - b - [arrow_key]`

5 Managing Code Deployment

There are multiple options to transfers files between your VM and your local computer. One option is to use a tool called scp, which stands for "secure copy". scp uses a similar command to ssh for transferring files to and from your VM. Let's say you can access your VM with the following ssh command:

```
$ ssh -p 54003 scpdxcs@ml-lab-XXXXXXXXXXXXX.southcentralus.
cloudapp.azure.com
```

To transfer files to the VM from your local machine, use the following command:

```
$ scp -r -P 54003 path/to/local/file scpdxcs@ml-lab-
XXXXXXXXXXXXX.southcentralus.cloudapp.azure.com:path/to/remote/
destination
```

To transfer files from the VM to your local machine, use the following command:

```
$ scp -r -P 54003 scpdxcs@ml-lab-XXXXXXXXXXXXX.southcentralus.
cloudapp.azure.com:path/to/remote/file path/to/local/
destination
```

The -r option indicates that a recursive copy should be performed, meaning that you can transfer an entire directory structure with just this one command! (note that the -p (lowercase) is now a -P (uppercase))

Note: the scp command copies files regardless of the file changes from the source to the destination; however, rsync will only copy files when the file is updated in the source location. So if you have a large file (such as model), then rsync could be helpful, here is the tutorial on how to use it. Just remember trailing slash (/) is important as in the tutorial.

Here is an example of rsync command with specific port that can be found from Azure web:

```
$ rsync  -arvz -e  ssh -p PORT_NO --progress /Users/name/SCPD/
XCS234/A2/ scpdxcs@ml-lab-xxx:/home/scpdxcs/SCPD/XCS234/A2
```

A different solution is to use a version control system, such as Git. This way, you can easily keep track of the code you have deployed, what state it's in and even create multiple branches on a VM or locally and keep them sync'd.

The simplest way to accomplish this is as follows:

1. Create a Git repo on Github, Bitbucket or whatever hosted service you prefer.
2. Create an SSH key on your VM. (see the link below)
3. Add this SSH key to your Github/service profile.
4. Clone the repo via SSH on your laptop and your VM.

5. When the project is over, delete the VM SSH key from your Github/service account.

Resources:

- Github SSH key tutorial
- Codecademy Git tutorial (great for Git beginners to get started)
- rsync tutorial

Note: If you use Github to manage your code, you must keep the repository private not doing so is a violation of the honor code as your solutions will be visible to other students.

6 Managing Memory, CPU and GPU Usage on the VM

If your processes are suddenly stopping or being killed after you start a new process, it's probably because you're running out of memory (most likely to be GPU memory, but also possible RAM).

First of all, it's important to check that you are not running multiple memory hungry processes that you may have overlooked.

You can see/modify which processes you are running by using the following commands:

1. View all processes `$ ps au`
2. To search among processes for those containing the a query use:

   `$ ps -fA | grep [query]`

   For example, to see all python processes run:

   `ps -fA | grep python.`
3. Kill a process `$ kill -9 [PID]`

You can find the PID (or Process ID) from the output of (1) and (2).

To monitor your normal RAM and CPU usage, you can use the following command: `$ htop` (Hit q on your keyboard to quit.)

To monitor your GPU memory usage, you can use the `$ nvidia-smi` command. If training is running very slowly, it can be useful to see whether you are actually using your GPU fully. (In most cases, when using the GPU for any major task, utilization will be close to 100%, so that number itself doesn't indicate an Out of Memory (OOM) problem.)

However, it may be that your GPU is running out of memory simply because your model is too large (i.e. requires too much memory for a single forward and backward pass) to fit on the GPU. In that case, you need to either:

1. Train using multiple GPUs
2. Reduce the size of your model to fit on one GPU. This means reducing e.g. the number of layers, the size of the hidden layers, or the maximum length of your sequences (if you're training a model that takes sequences as input).
3. Lower the batch size used for the model. Note, however, that this will have other effects as well (as we have discussed previously in class).