

XCS224U Azure Guide

This guide will help you set up and use Azure Virtual Machines for any work in XCS224U that you'd like (homework, bake-offs, or projects). Before you start, it cannot be stressed enough: **do not leave your machine running when you are not using it!**

[Access and Setup](#)

[Azure Labs Subscription for this Class](#)

[Best Practices for Managing Credit](#)

[Registration](#)

[Connecting to a VM](#)

[Practical Guide for Using the VM](#)

[Managing Processes on a VM](#)

[TMUX Cheatsheet](#)

[Managing Code Deployment to a VM](#)

[Managing Memory, CPU and GPU Usage on a VM](#)

Access and Setup

Azure Labs Subscription for this Class

We are using [Azure Lab Services](#) to manage VMs for the XCS224U course. Every student will be allocated 75 hours total to use however you'd like. **It's important for students to manage credit wisely in order to make the most efficient use of it (see next section).**

You will receive an invitation to register for the lab. Credit has been assigned per student and everyone's instances are preconfigured with Linux DSVM (Data Science Virtual Machine) images so you can expect some packages/tools to be installed.

Best Practices for Managing Credit

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

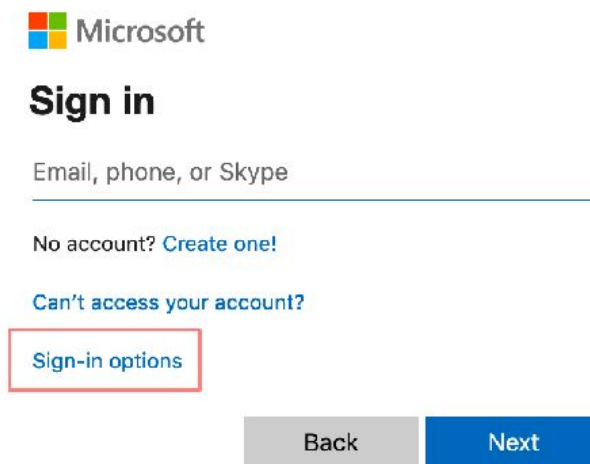
- whether you are ssh'd to the machine at that time
- whether you are running any processes on the machine at that time
- the computational intensity of the processes you're running
- whether you're using GPUs

Therefore, the most important thing for managing credit wisely is to carefully turn your VM on and off only when you need it.

We advise you to **develop your code on your local machine** (for example your laptop with the CPU version of Pytorch installed) for debugging (i.e., work on your new code until you are able to complete several training iterations without errors), then run your code on your Azure VM when it's time to train on a GPU.

Registration

1. Go to this link: <https://labs.azure.com/register/18m3lr11>
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
or
 - B. Logging in with a Skype account
or
 - C. If you click 'Sign-in Options' you will also be presented with the option to sign in using your GitHub credentials



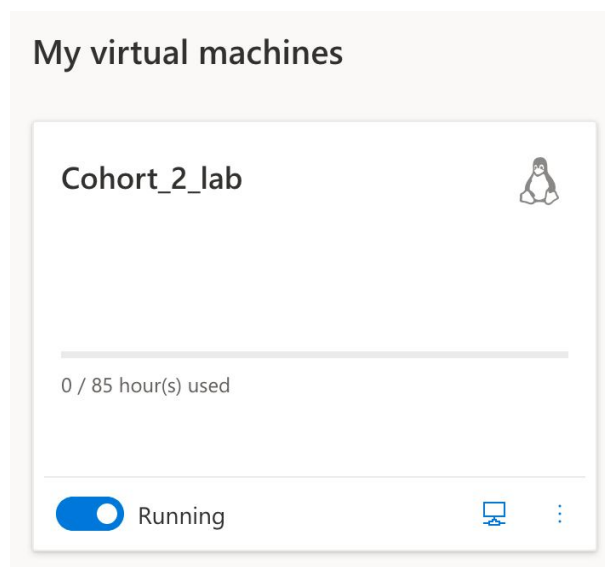
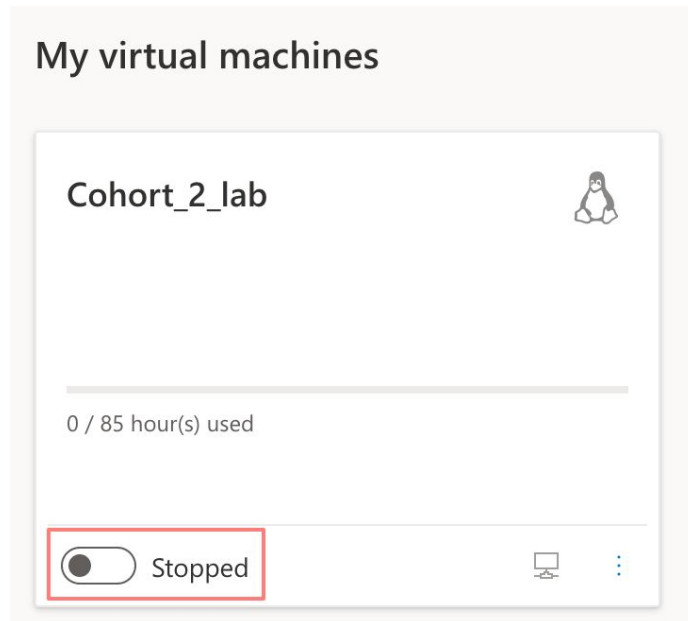
The image shows a Microsoft sign-in interface. At the top is the Microsoft logo. Below it is the heading "Sign in". Under the heading is a text input field labeled "Email, phone, or Skype". Below the input field are two links: "No account? Create one!" and "Can't access your account?". Below these links is a button labeled "Sign-in options", which is highlighted with a red rectangular border. At the bottom of the form are two buttons: "Back" (grey) and "Next" (blue).

Once you've done A, B, or C - follow any additional prompt instructions (depends on which way you chose) - and you will be registered for the lab!

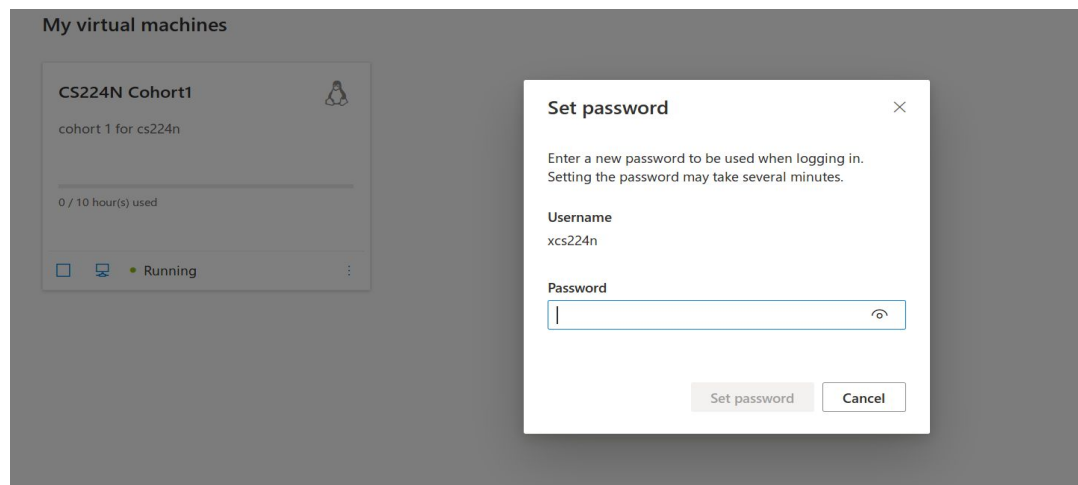
Connecting to a VM

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.

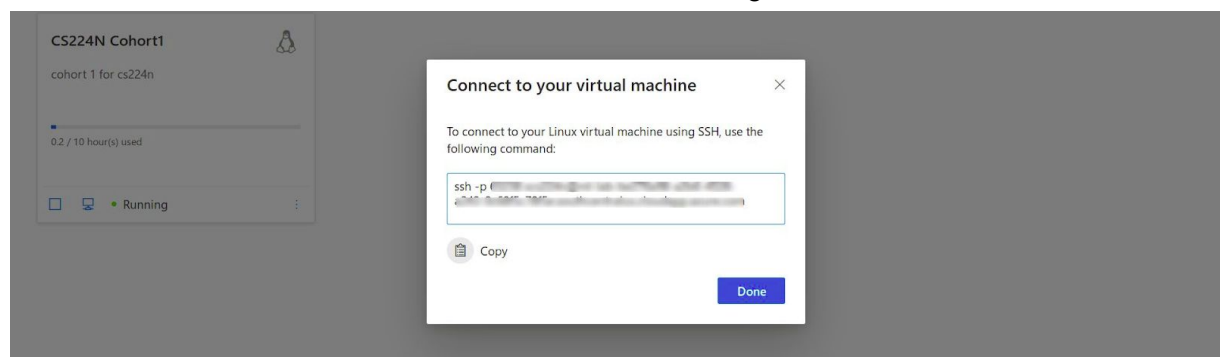
Click on the 'Stopped' button to start the instance (this will take a few minutes). When it is up and running, it will look like this and say "Running" in the bottom status bar:



2. Click the monitor icon in the window above 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).



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



4. Copy the link and paste it into your terminal (Windows users can use [PuTTY](#))

```
xcs224n@ML-EnvVm-00000:~$ ssh -p 49521 xcs224n@ml-lab-be276a98-a2b6-4528-a243-0c68f5c78f5e.southcentralus.cloudapp.azure.com
The authenticity of host '[ml-lab-be276a98-a2b6-4528-a243-0c68f5c78f5e.southcentralus.cloudapp.azure.com]:49521 ([52.249.56.254]:49521)' can't be es
ECDSA key fingerprint is SHA256:XFGUWgt5qk9teCFYNUWE9Qd76lMGkeBD2g8Ktr0f+Y.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[ml-lab-be276a98-a2b6-4528-a243-0c68f5c78f5e.southcentralus.cloudapp.azure.com]:49521,[52.249.56.254]:49521' (ECDSA) to
xcs224n@ml-lab-be276a98-a2b6-4528-a243-0c68f5c78f5e.southcentralus.cloudapp.azure.com's password:
>Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.15.0-1055-azure x86_64)

96 packages can be updated.
3 updates are security updates.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

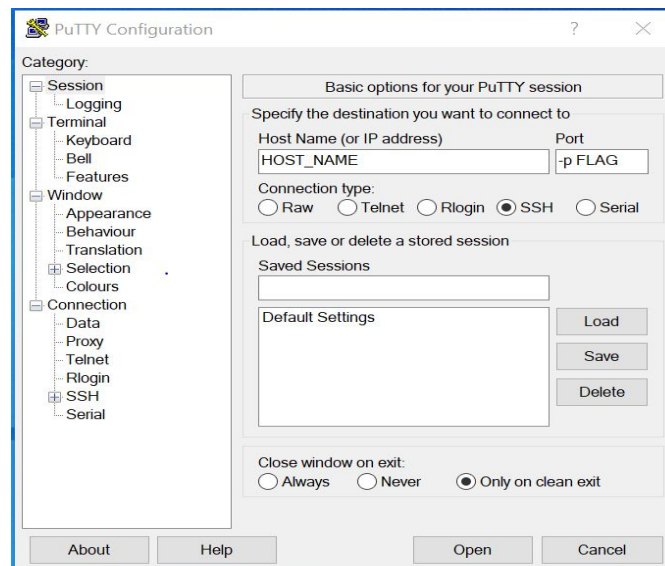
*****
* Welcome to the Linux Data Science Virtual Machine on Azure! *
* For more information on available tools and features, *
* visit http://aka.ms/dsvm/discover. *
*****

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

xcs224n@ML-EnvVm-00000:~$
```

Windows users using [PuTTY](#):

Make sure to copy into “Host Name” the url from the Azure ssh command and into “Port” whatever appears after the `-p` flag in the ssh command.



5. Update PyTorch Version

The VM template is initialized with PyTorch 1.2.0 (not ≥ 1.3 as suggested in the XCS224U course setup guide).

To update PyTorch to 1.3 or above on your VM:

- Run **conda install pytorch==1.4.0 torchvision==0.5.0 -c pytorch**

- You may have to run the above command twice (we have found in testing that in the first run dependencies are updated and in the second run pytorch and torchvision are updated).

6. Check that Pytorch can access the GPUs by opening Python and typing the following:

```
import torch
torch.cuda.current_device()
torch.cuda.device(0)
torch.cuda.device_count()
.
```

You should see something like this:

```
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

*****
* Welcome to the Linux Data Science Virtual Machine on Azure!          *
*                                                                       *
* For more information on available tools and features,                *
* visit http://aka.ms/dsvm/discover.                                   *
*****

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

xcs224n@ML-EnvVm-00000:~$ ipython
Python 3.5.5 |Anaconda custom (64-bit)| (default, May 13 2018, 21:12:35)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import torch

In [2]: torch.cuda.current_device()
Out[2]: 0

In [3]: torch.cuda.device(0)
Out[3]: <torch.cuda.device at 0x7f3e1023bc50>

In [4]: torch.cuda.device_count()
Out[4]: 1

In [5]: torch.cuda.get_device_name()
Out[5]: 'Tesla K80'
```

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

Practical Guide for Using the VM

Managing Processes on a VM

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 leave a script running on a VM and log-off, your process will likely be disrupted. Furthermore, it is often quite nice to be able to have multiple terminal windows open with different processes all visible at the same time, without having to SSH into the same machine multiple different times.

TMUX or "Terminal Multiplexer" is a very simple solution to all the problems above.

Essentially, 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 actually 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.

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]`

Managing Code Deployment to a VM

You are welcome to use scp/rsync to manage your code deployments to the VM. However, a better 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)

*Note: If you use Github to manage your code, you must keep the repository **private** until the class is over.*

Managing Memory, CPU and GPU Usage on a 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 (either on the GPU or just normal RAM).

First of all, it's important to check that you not running multiple memory hungry processes that maybe have slipped into the background (or a stray TMUX session).

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 (this is troublesome to implement, and costs much more on Azure)
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).