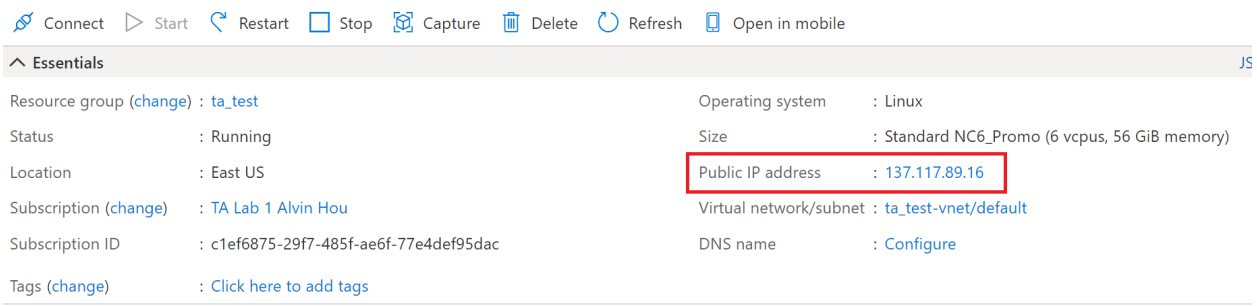
CS224n: Practical Tips for using Virtual Machines

Contents

Connecting to a VM



Access VM through Password

If you had chosen password authentication when setting up the VM, you could access the VM with the following ssh command

```
$ ssh myuser@137.117.89.16
```

where `137.117.89.16` is the Public IP address. You should be able to login to the VM after entering your password.

Access VM through SSH key

If you had chosen ssh key authentication, you could access the VM with the following command

```
$ ssh -i /path/to/mykey.pub myuser@137.117.89.16
```

Managing Code Deployment to a VM

Transfer files through scp

You are welcome to use scp/rsync to manage your code deployments to the VM. The following are some example commands on how to use scp.

Copy a single file from local (your mac/pc) to the VM

```
$ scp /path/to/local/code.py ange@13.90.46.179:~
```

Copy a directory from local to the VM

```
$ scp -r /path/to/local/assignment4/ ange@13.90.46.179:~
```

Copy a single file from the VM back to local

```
$ scp ange@13.90.46.179:/path/at/vm/code.py /path/at/local
```

Copy a directory from the VM back to local

```
$ scp -r ange@13.90.46.179:/path/at/vm/hw4 /path/at/local/hw4
```

If you are using ssh-key authentication to access your VM, simply add `-i ~/.ssh/mykey` to the scp command. A simple example:

```
$ scp -i ~/.ssh/mykey /path/to/local/code.py  ange@13.90.46.179:~
```

Transfer and Manage files through Git

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. Upload the assignments to the repo from your local machine. If you are not sure how to do this, check out the Git tutorial below or online

```
$ git add .
$ git commit -m "Init commit"
$ git remote add origin https://github.com/GithubUser/CS224n.git
$ git push -u origin master
```

3. SSH to your Azure VM

```
$ ssh username@123.117.89.17
```

4. Git clone the repo from your Github

```
$ git clone https://github.com/GithubUser/CS224n.git
```

5. Start running the assignment on the VM
6. Git push any changes to your Github (repeat step 2)

If you prefer to use Github SSH key, see the following

1. Create a Git repo on Github, Bitbucket or whatever hosted service you prefer.
2. Create a 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 always keep the repository **private**. Github offers free private repositories to students.*

Remote Development on a VM

If you are not a fan of using git or scp to update files every time you made a change locally, there are some tools for remote development on a VM.

VSCode Remote Development

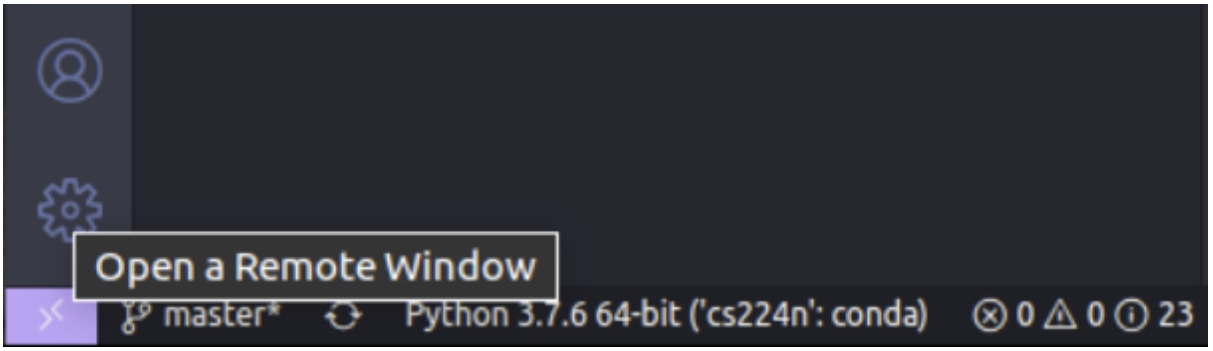
VSCode provides an easy way for developers to connect their IDEs to the remote host through an ssh tunnel. As a result, we can write and run code the same way as usual but on the Azure VM.

Before you start, you should first clone or upload the assignment to your Azure VM.

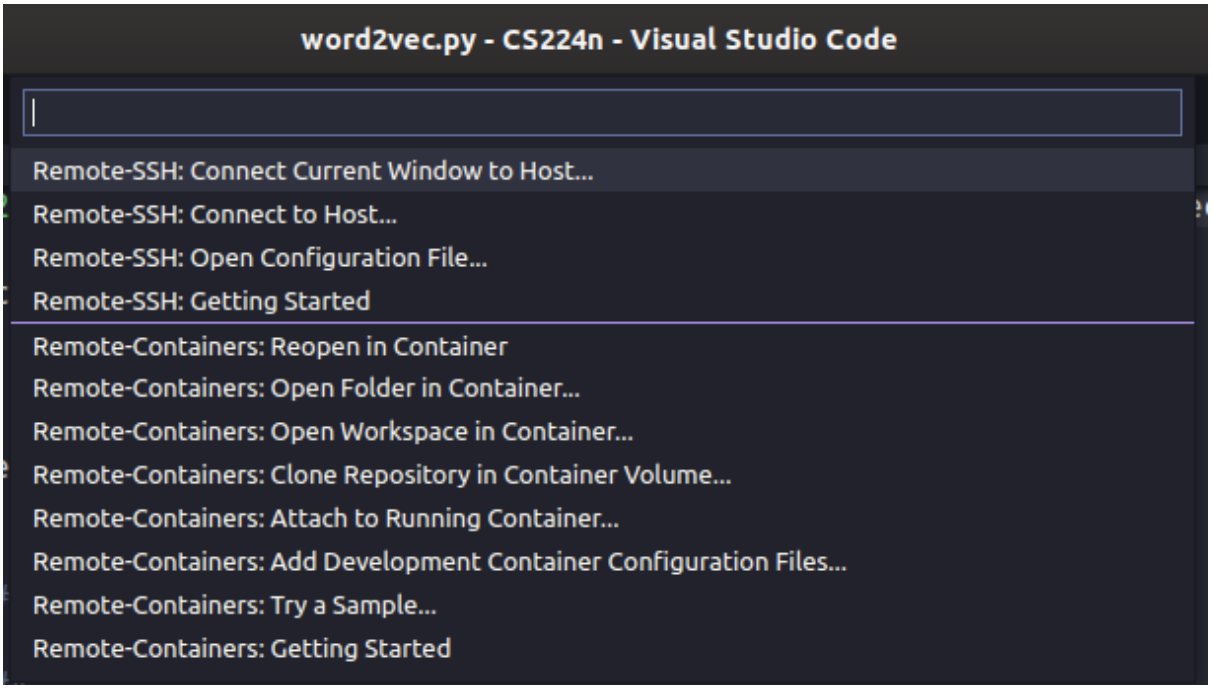
1. Install VScode (https://code.visualstudio.com/)

2. Install VScode Remote Development Extension (https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.vscode-remote-extensionpack)

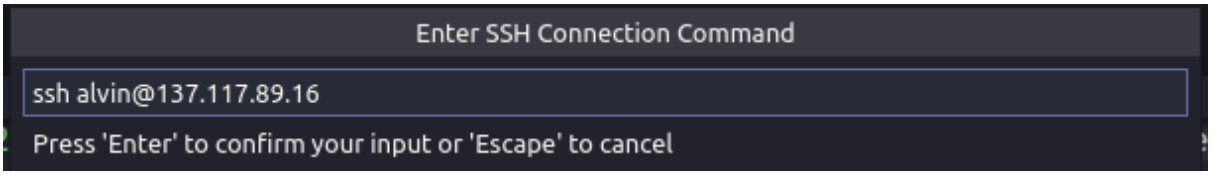
3. Click on "Open a Remote Window" on the bottom left corner



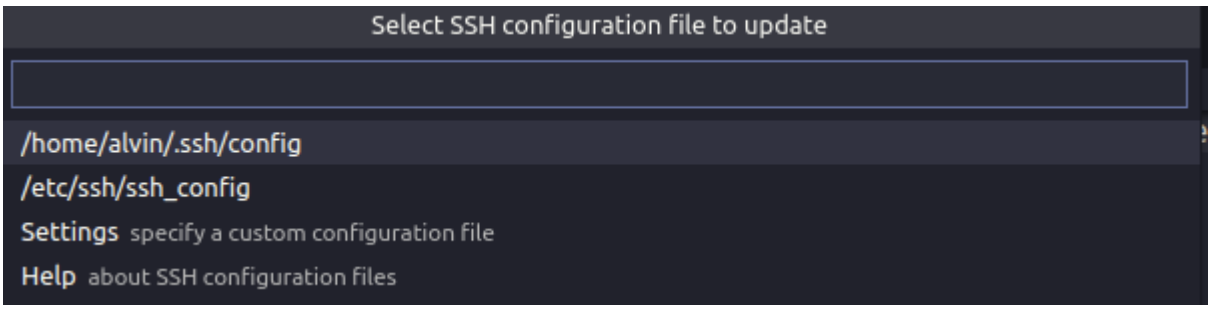
4. Select "Remote-SSH: Connect Current Window to Host"



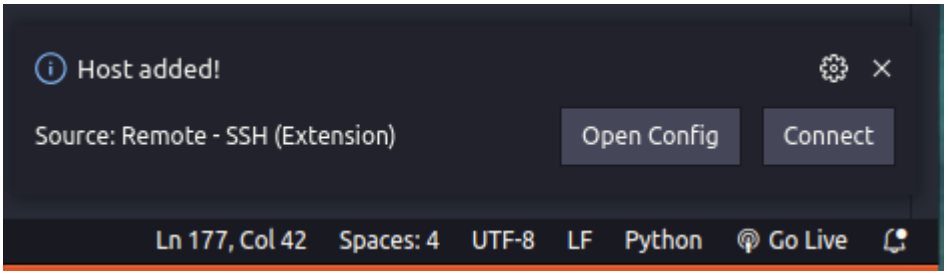
5. Click "Add New SSH Host" and enter the ssh command for your Azure VM



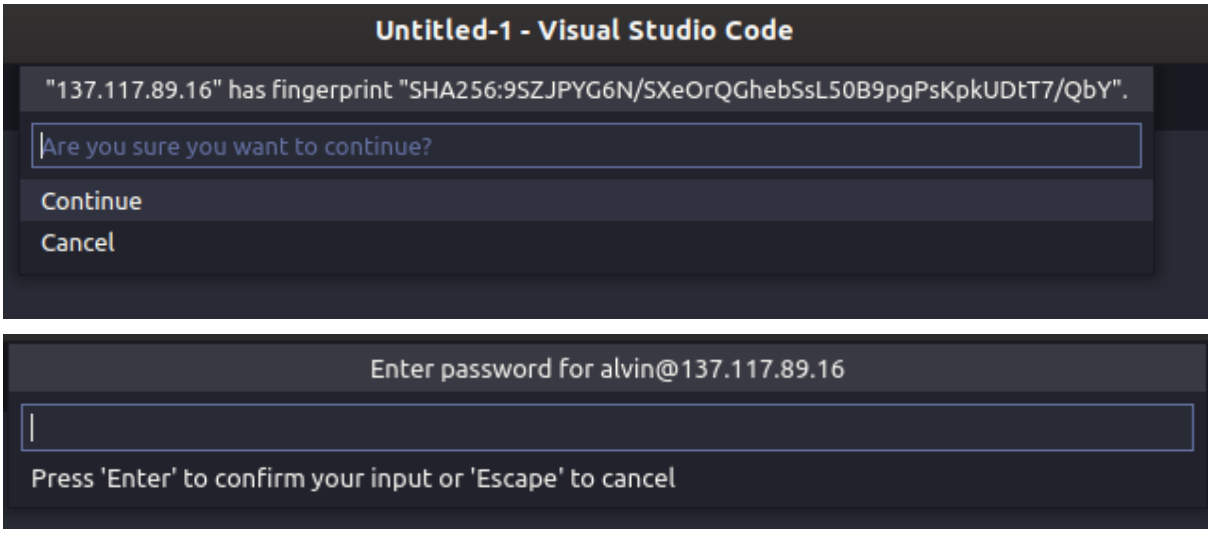
6. Select an SSH config file to update. Either option should be fine.



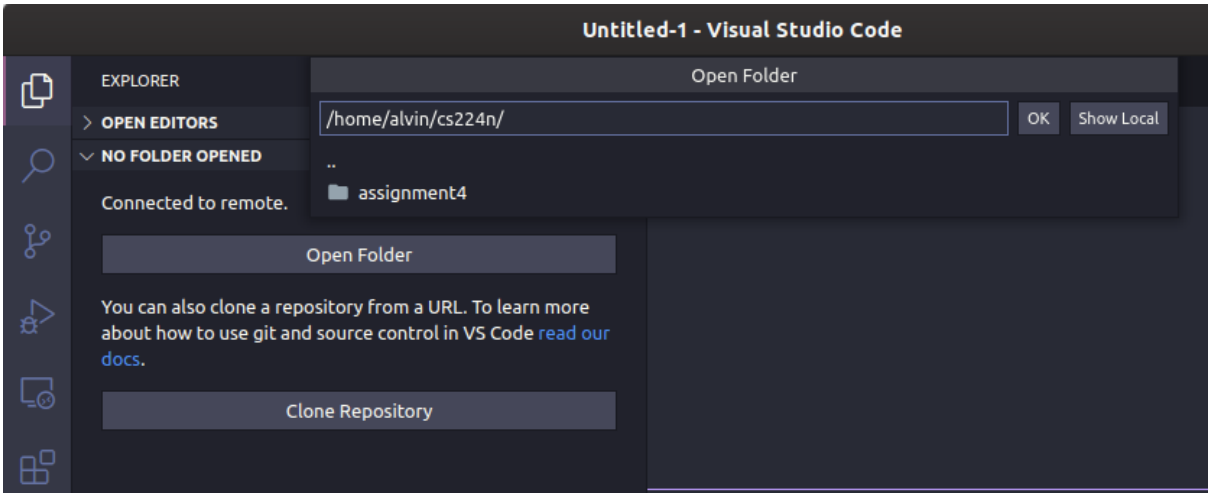
7. Click on "Connect" after the host is added



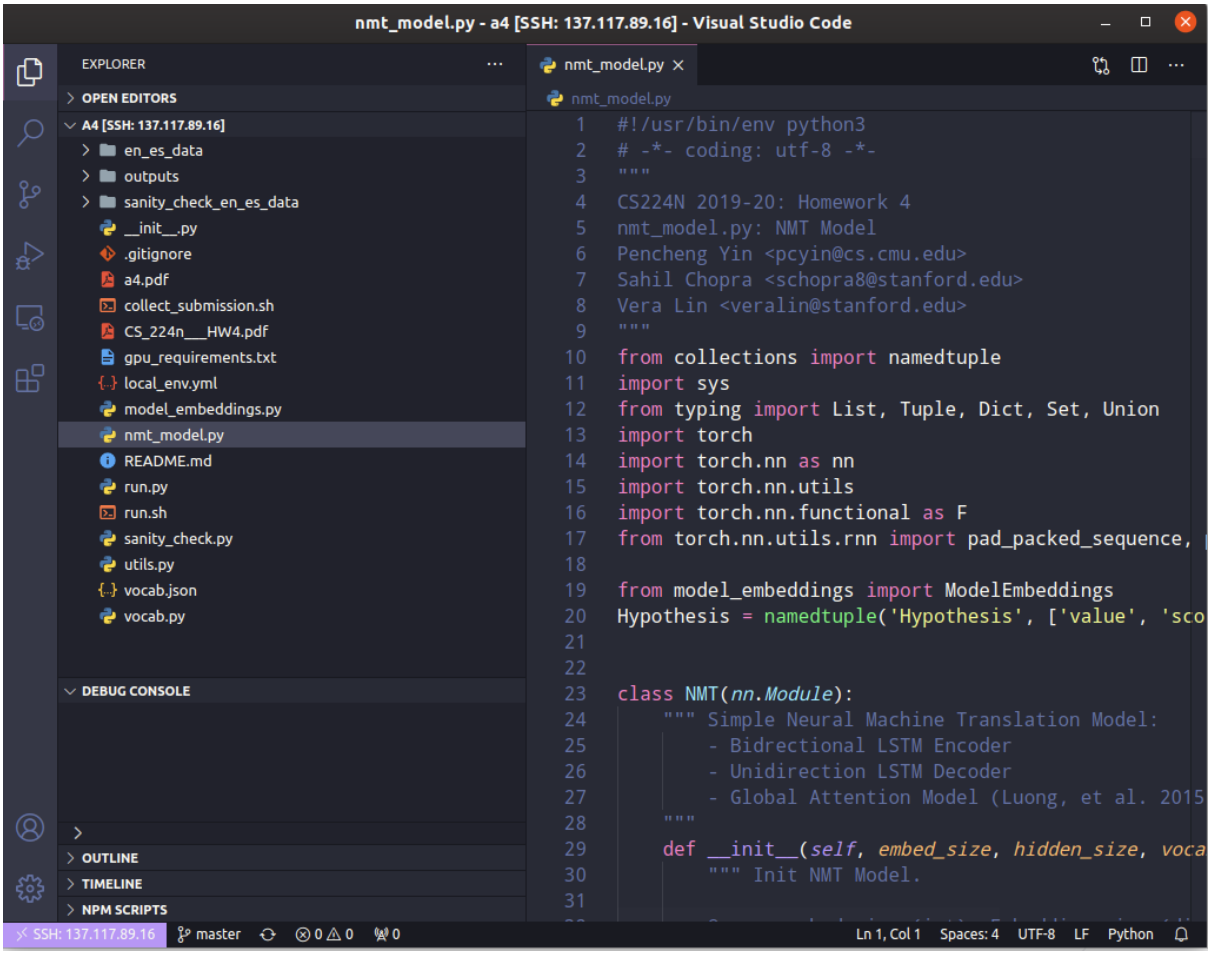
8. Enter "Continue" and fill in your password.



9. You've now successfully connected to your Azure VM! Click "Open folder" and find the folder where your assignment is located



10. Happy coding with VScode on your Azure VM!



See https://code.visualstudio.com/docs/remote/remote-overview for more tutorials and details.

Disclaimer: We still advise you to **develop your code on your local machine** to save Azure credits. You could use this for quick debugging.

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 sessions `ctrl - b - x`
8. Split a pane horizontally `ctrl - b - "`

Split a pane vertically `ctrl - b - %`

9. Move to pane `ctrl - b - [arrow_key]`

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).
4. Use techniques like gradient accumulation (also additional work). You can find a discussion of some of these techniques in this article: https://medium.com/huggingface/training-larger-batches-practical-tips-on-1-gpu-multi-gpu-distributed-setups-ec88c3e51255 . They're all also implemented in the huggingface transformers library: https://github.com/huggingface/transformers