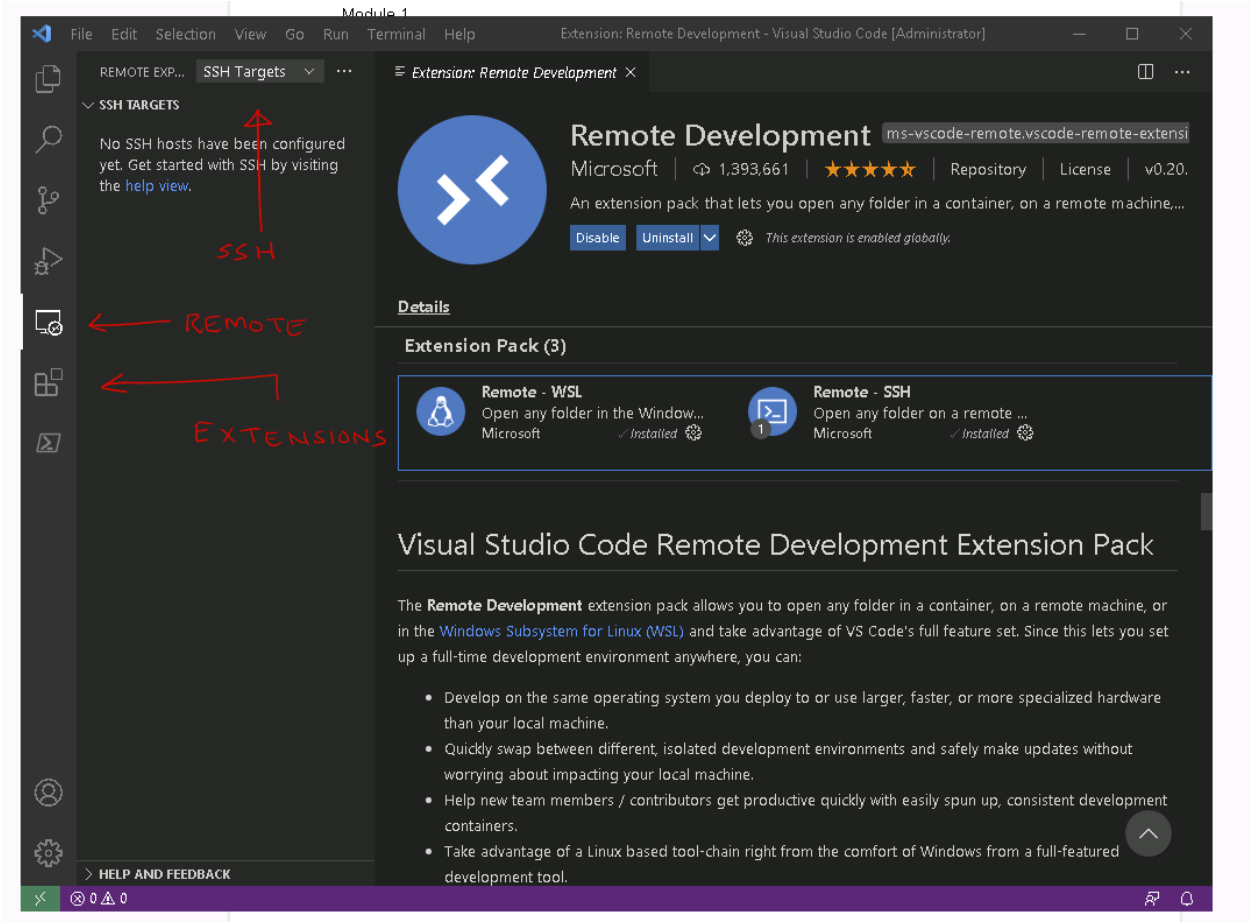
# Module 0

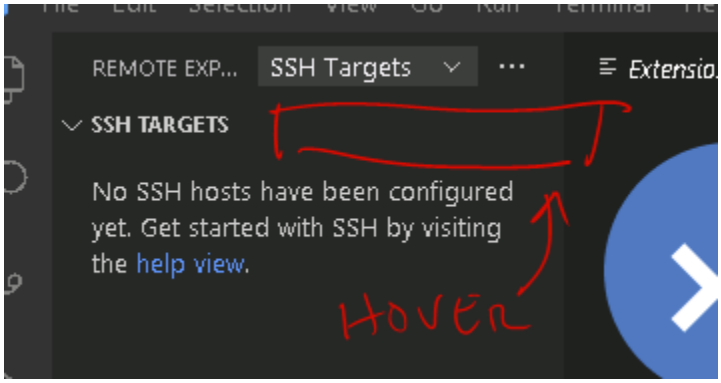
Using the linux box for development is a crappy experience - lets install chocolaty to setup the windows machine with vscode, python, and pip

1. Run the following command as administrator in powershell:
   `Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))`
2. When it's finished installing, close the powershell window and re-open it again as administrator
3. Run the following command to install vscode: `choco install -y vscode vscode-python vscode-powershell openssh python3 git chromium postman`
4. Run the following commands:
   a. `ssh-keygen` (accept all defaults and leave password blank)
   b. `cp ~\.ssh\id_rsa.pub ~\.ssh\authorized_keys`
   c. `scp -r ~\.ssh root@rhel1:`
   d. `ssh root@rhel1 "chmod -R 600 .ssh"`
   e. `ssh root@rhel1 "echo worked"`

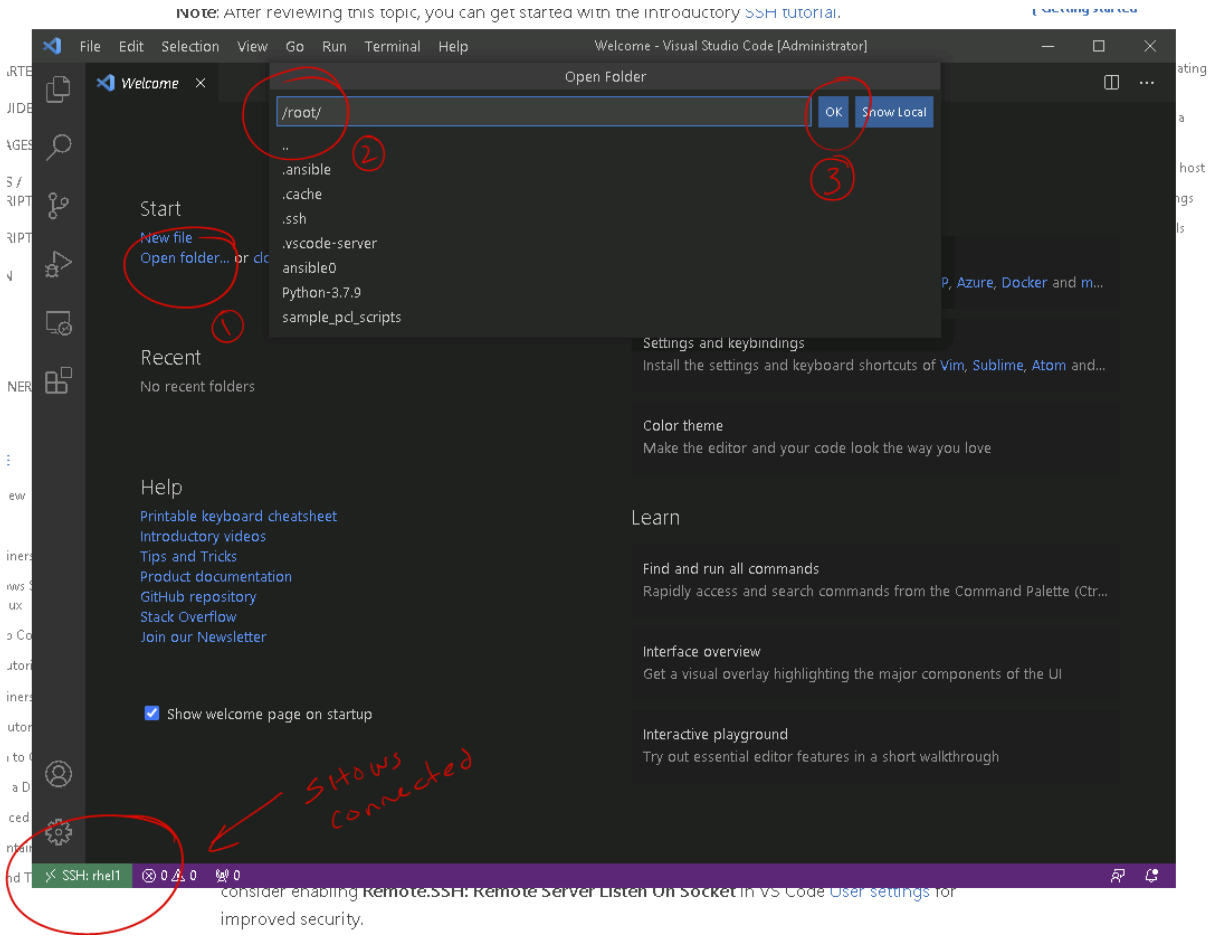
5. Launch Visual Studio Code, click on the side bar icon that looks like a stack of blocks (extensions), and then search for and install the “remote development” extension.



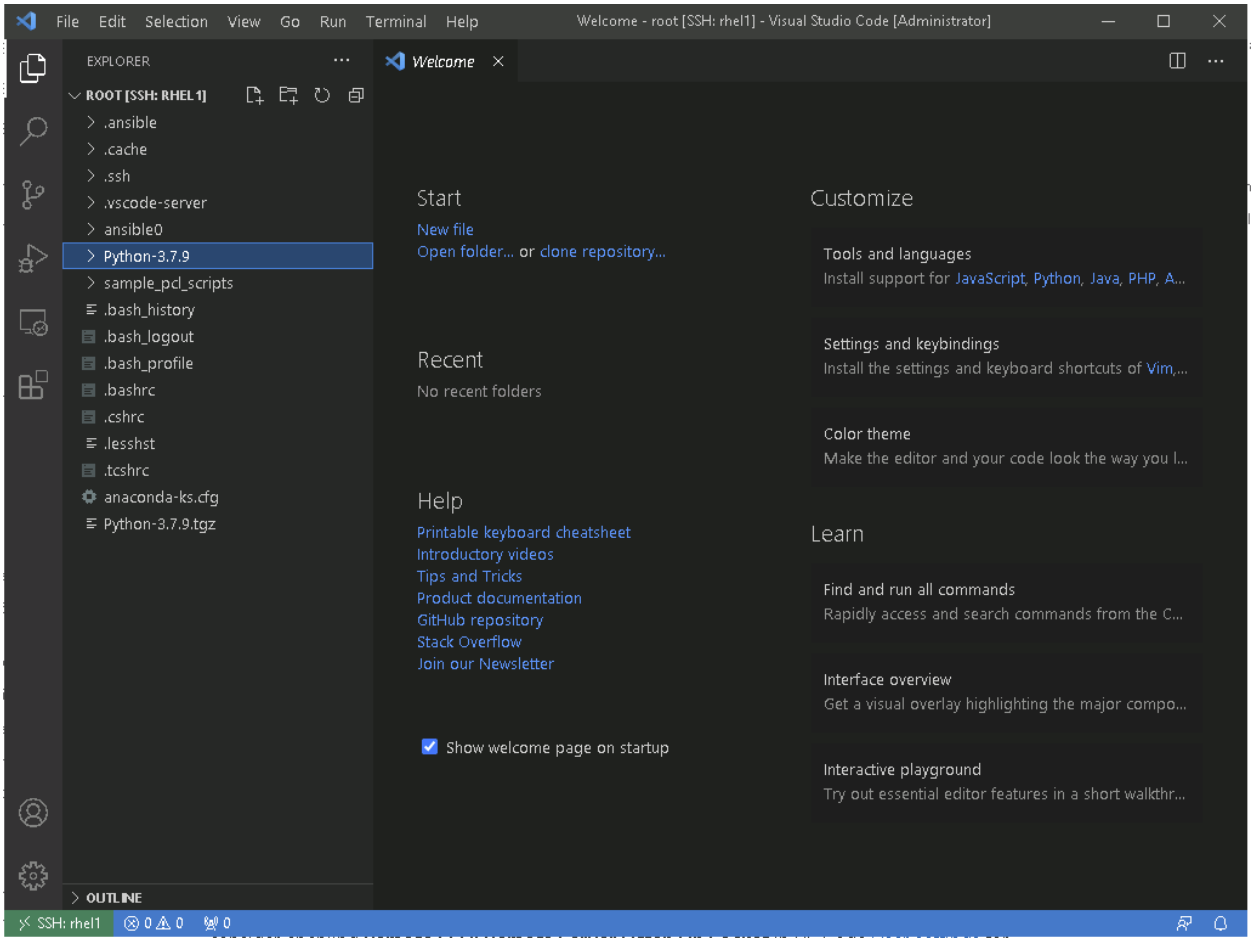

6. Hover your mouse under ssh targets and click the + icon when it shows up



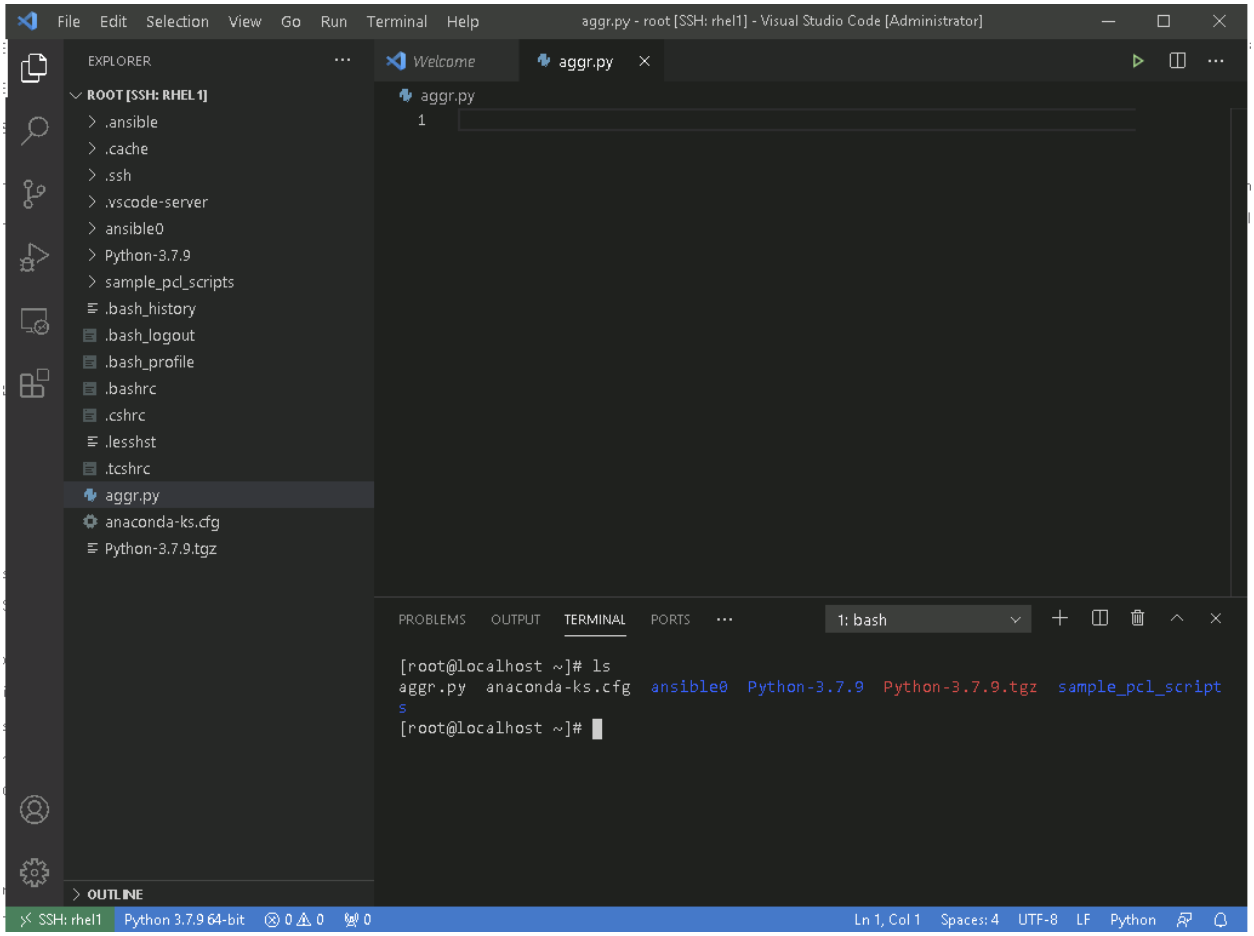

7. You will be prompted to enter a ssh target, add “ssh root@rhel1”
8. Accept the default location for the ssh config file
9. Click connect.




10. You should now see all of the folders in root's home directory on rhel1 listed in the sidebar:




11. Go to the terminal menu, and open a new terminal, this will open on the bottom of the screen, and you can now run your scripts, and see the output in the terminal




# Module 1:

On your redhat linux system, ping cluster1. If you get an error message, cluster1 may not have a dns entry in dc1's dns. You can either use the ip address of the cluster (192.168.0.101 for cluster1), or create a dns entry, or an entry in the /etc/hosts file.

**For the aggregate script, it's useful to add:**

```
requests.packages.urllib3.disable_warnings()
```

**For the qtree script:**

- Using a name like qt-$(date +%s) will create a qtree with a name like: qt-1619202807, and where the # is the seconds since 1970. This allows you to not have to delete the qtree or change the name if you iterate testing. Same for the Qos policy
- In order to set a user quota, we need a user. **pcuser** exists by default, so use pcuser, or create a new user. Quotas do not apply to root.

- Unless you manually enable quotas, you will get a quota resize error. This is fine and safe to ignore for the labs. In reality, you would also need to enable quotas on the volume.

```
python3 qtree.py -c 192.168.0.101 -v Vol1 -q qt-$(date +%s) -vs
VServer1 -sh 1000000000 -fh 1000 -un pcuser -qos qos-$(date +%s) -u
admin -p Netapp1!
Creating QTree...
[2021-04-23 14:33:28,745] [ INFO] [utils:183] Job (success): success.
Timeout remaining: 30.
qtree qt-1619202807 created successfully
Creating Quota Rule...
[2021-04-23 14:33:28,847] [ INFO] [utils:183] Job (running): Quota
State is off, Quota resize will not be attempted. Timeout remaining:
30.
[2021-04-23 14:33:33,878] [ INFO] [utils:183] Job (success): success.
Timeout remaining: 25.
quota rule for qt-1619202807 created successfully
Creating QOS Policy...
qtree qos-1619202807 created successfully
```

**Example json to create a qos policy:**

```
{
  "adaptive": {
     "absolute_min_iops": 100,
     "expected_iops": 100,
     "peak_iops": 100
  },
  "name": "pancakes",
  "svm": {
     "name": "VServer1"
  }
}
```

**Example json to create a qtree:**

```
 {
      "name": "test1",
      "svm": {"name": "VServer1"},
      "volume": {"name": "Vol1"}
 }
```

**Example json for creating a quota rule:**

```
{
  "files": {
```

```
        "hard_limit": 100
    },
    "qtree": {

        "name": "test1"
    },
    "space": {
        "hard_limit": 1222800
    },
    "svm": {
        "name": "VServer1"
    },
    "type": "user",
    "user_mapping": "off",
    "users": [
        {
        "name": "pcuser"
        }
    ],
    "volume": {
        "name": "Vol1"
    }
}
```

# Module 2

The /etc/resov.conf has a typo, It should say:

```
search demo.netapp.com
nameserver 192.168.0.253
```

# Module 4

Task1, step 1-2:

- `yum install epel-release ansible`
- `yum install ansible ansible-doc ansible-lint vim-ansible ansible-python3 ansible-review git`
- pip3 is probably already installed, so you can skip the sudo easy_install line if it is
- Change the pip install line to include oslo-log:
    - `pip3 install ansible netapp-lib requests oslo-log`
- The rest of the command should be fine

Task1, step 1-8:

- The ping command may fail when connecting to rhel2.
- You can either:
    - Add the flag “--ask-pass” to be prompted for a password
    - Do s`sh-copy-id 192.168.0.62` to copy your ssh public key to rhel2.
        - If you did not generate a ssh key earlier, `ssh-keygen` will create one

Task 2, step 2-5:
- This fails
- Comment out the line for the svm21_root volume in the vars section
- Move the aggregate delete to the very end, and this should fix it.

Task 3, step 3-6:
- The first task setting up licenses fails; This works:

```
- name: Install Licenses
    na_ontap_license:
    state: present
    license_codes: "{{ licenses }}"
    <<: *login
```

- The task called “Create NFS Life Node1” fails
- The var_cluster1.yml calls for a subnet called “Demo” but it was already created called “subnet”
- Change the subnet_name: {{ subnetname }} to subnet_name: “subnet”

Task 3, 3-10:
- Same issue as 3-6 with the licenses
- There is a task to create a subnet called Demo - comment this out
- Change the subnet used by the subnet_name to “subnet” just like in 3-6

Module 5:
- 1-2: you can just type the commands “vserver create…” instead of breaking it up onto two lines like they do.
- You may get an error about the CN when doing the cifs create command, you can leave the “-ou CN=Computers” off, and it works fine.