Quick-Start & Admin-Safe Guide

(Assistant Lecturer: Eng. Ahmed Métwalli) (Last updated: July 24, 2025)

1. How to Connect to the Cluster

A. FortiClient SSL-VPN (Windows Linux macOS)

1. **Download FortiClient VPN-only edition**
   Grab the installer for your OS from Fortinet's official page (Windows / Mac DMG / Linux RPM/DEB).[1]

2. **Create a new connection:**
   Remote Access → Configure VPN → SSL-VPN.[2]

   Connection Name Any mnemonic label (e.g. "Mito").

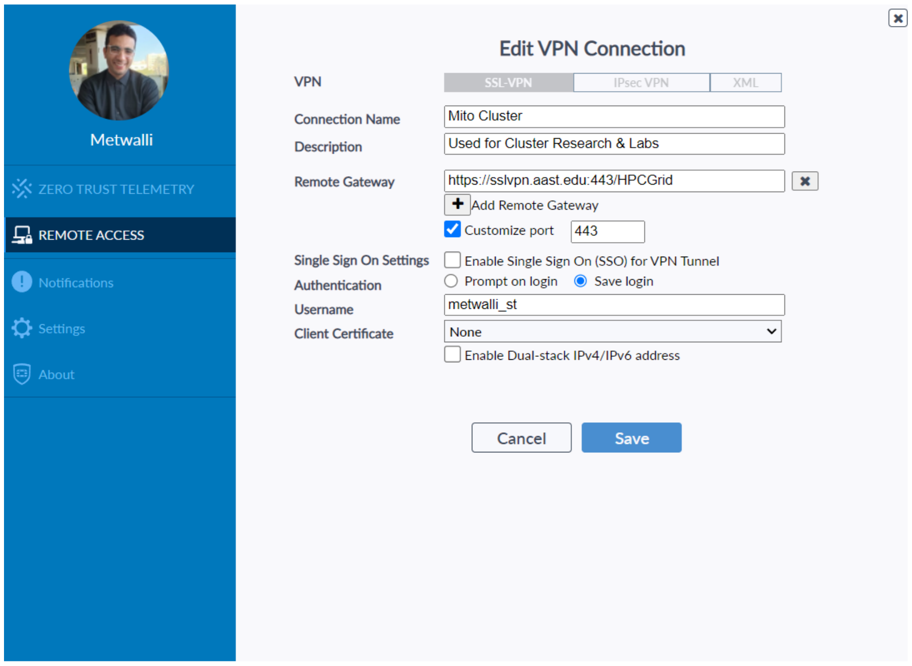
   Remote Gateway `https://sslvpn.aast.edu:443/HPCGrid`
   (URI routes you straight to the grid portal; `:443` is the default HTTPS port, no need to open extra firewall holes.)

   Customized Port 443 — keeps traffic on standard HTTPS.

   Authentication *Save Login* to avoid re-typing every session.

   Username Campus ID *without* "@student.aast.edu".

3. Click **Save**, then **Connect**. Enter your password, wait for "Status: Connected", then leave FortiClient running in the tray.

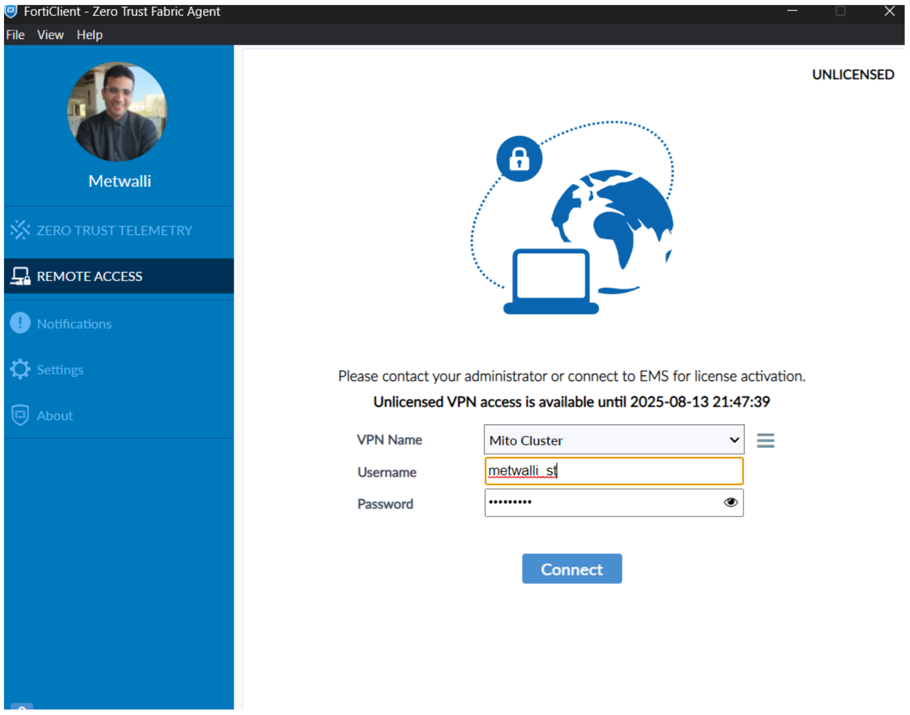


Edit VPN Connection

VPN: SSL-VPN | IPsec VPN | XML
Connection Name: Mito Cluster
Description: Used for Cluster Research & Labs
Remote Gateway: https://sslvpn.aast.edu:443/HPCGrid
Add Remote Gateway
☑ Customize port 443
Single Sign On Settings: ☐ Enable Single Sign On (SSO) for VPN Tunnel
Authentication: ○ Prompt on login [✓ Save login]
Username: metwalli_st
Client Certificate: None
☐ Enable Dual-stack IPv4/IPv6 address

Figure 1: Forticlient A

[1] See `https://www.fortinet.com/support/product-downloads`.
[2] Full field list in FortiClient Admin Guide §"Configuring an SSL VPN connection".



FortiClient - Zero Trust Fabric Agent

File View Help

Metwalli

ZERO TRUST TELEMETRY

REMOTE ACCESS

Notifications

Settings

About

UNLICENSED

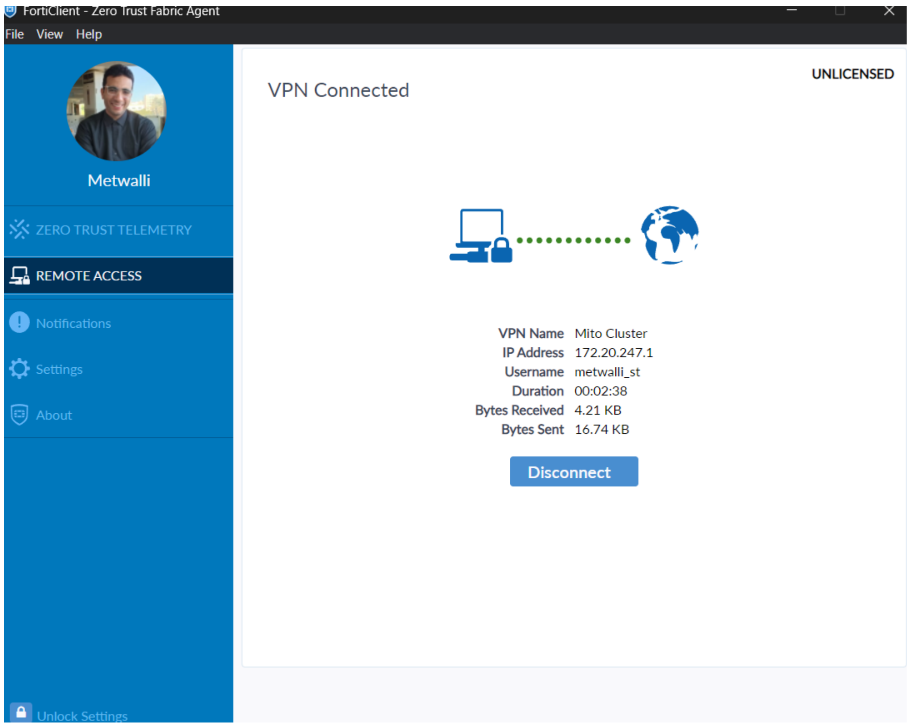
Please contact your administrator or connect to EMS for license activation.

Unlicensed VPN access is available until 2025-08-13 21:47:39

VPN Name	Mito Cluster
Username	metwalli_st
Password	•••••••••

Connect

Figure 2: Forticlient B



FortiClient - Zero Trust Fabric Agent

File View Help

Metwalli

ZERO TRUST TELEMETRY

REMOTE ACCESS

Notifications

Settings

About

Unlock Settings

UNLICENSED

VPN Connected

VPN Name	Mito Cluster
IP Address	172.20.247.1
Username	metwalli_st
Duration	00:02:38
Bytes Received	4.21 KB
Bytes Sent	16.74 KB

Disconnect

Figure 3: Forticlient C

B. VS Code SFTP Workflow

1. Install the extension Open the VS Code Marketplace, search for *"SFTP" by liximomo* (publisher ID `liximomo.sftp`) and click **Install**. The plugin enables one-click upload, download and remote terminal access over SSH.

2. Create project workspace In your local development folder, make sure there is a `.vscode/` directory:

```
mkdir -p ~/projects/mito-demo/.vscode
cd ~/projects/mito-demo
```

3. Add `sftp.json` Place the following content in `.vscode/sftp.json`. Replace `<password>` as needed; you can switch to key-based auth later via the `"privateKey"` field.

```
{
  ”name”: ”Mito–EntryPoint”,
  ”protocol”: ”sftp”,
  ”host”: ”10.1.8.4”,
  ”port”: 22,
  ”username”: ”metwalli_st@student.aast.edu”,
  ”password”: ”<your password>”,
  ”remotePath”: ”/home/metwalli_st@student.aast.edu/project”,
  ”uploadOnSave”: false,
  ”syncMode”: ”update”,
  ”watcher”: {
    ”files”: ”/*”,
    ”autoUpload”: false,
    ”autoDelete”: false
  }
}
```

Listing 1: Minimal SFTP config

Key fields:

- `remotePath` – root directory on the cluster where your coursework lives. The extension mirrors local edits there instantly when you choose "Upload"
- `uploadOnSave=false` – avoids accidental uploads; you push manually or per-file.
- `syncMode="update"` – only newer files are transferred, cutting network time
- `watcher` – disables background polling that could waste login-node I/O.

4. Refresh & verify Click the icon in VS Code's SFTP sidebar to reload the profile. A new entry named "`Mito-EntryPoint`" appears.

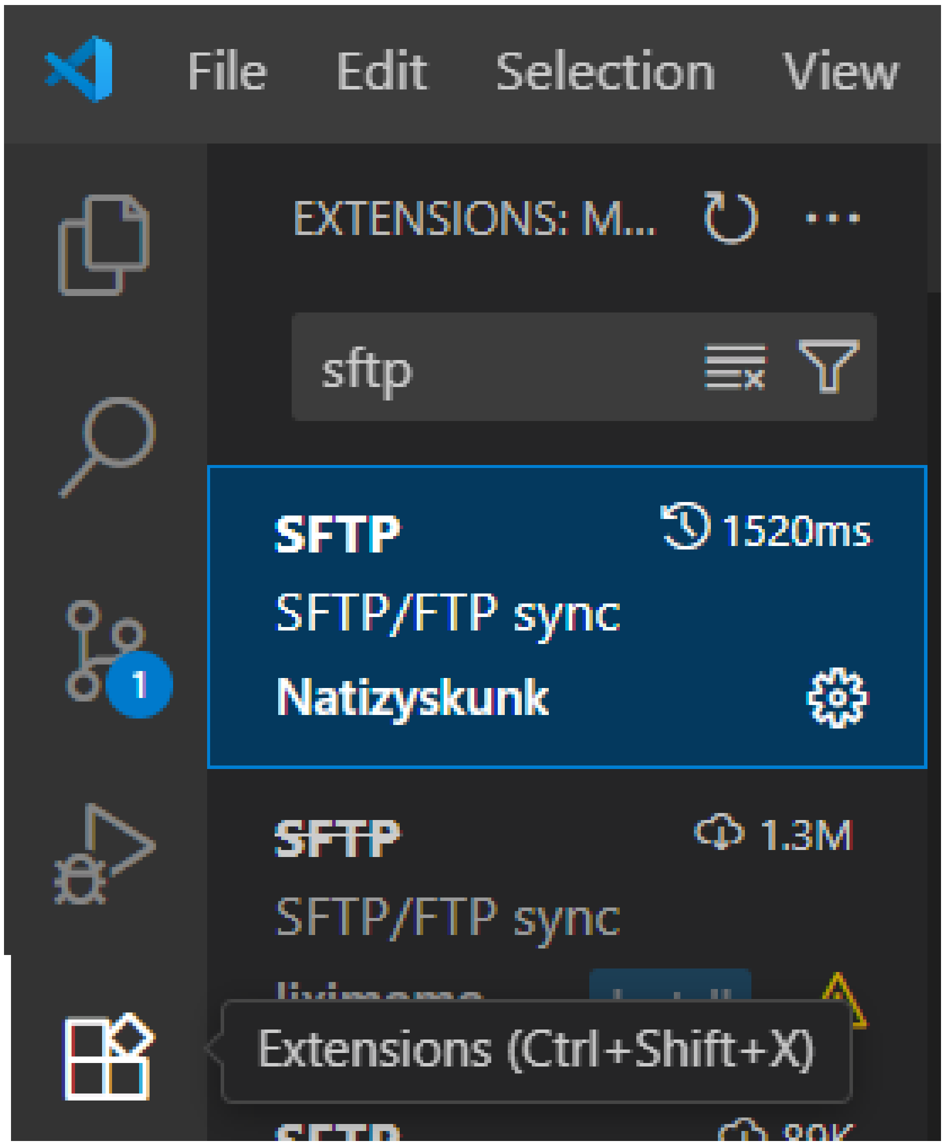
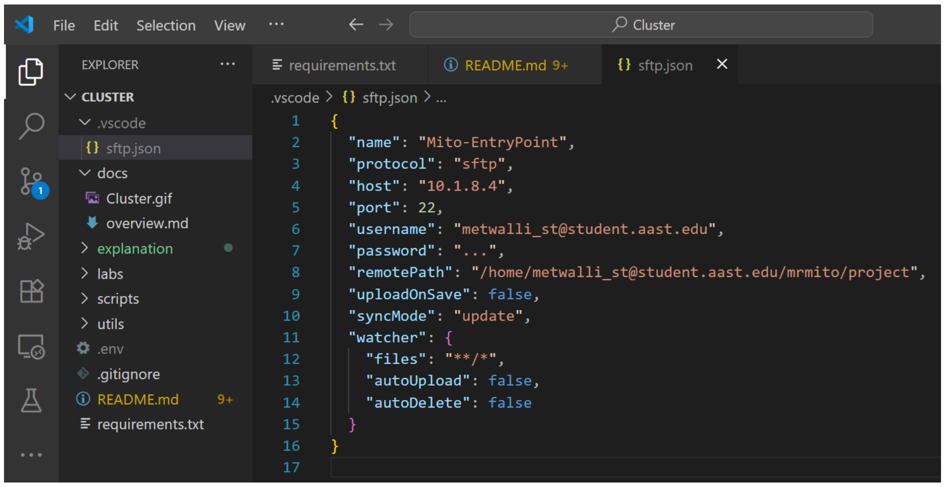


Figure 4: SFTP 1: The VS code extension



```
{
  "name": "Mito-EntryPoint",
  "protocol": "sftp",
  "host": "10.1.8.4",
  "port": 22,
  "username": "metwalli_st@student.aast.edu",
  "password": "...",
  "remotePath": "/home/metwalli_st@student.aast.edu/mrmito/project",
  "uploadOnSave": false,
  "syncMode": "update",
  "watcher": {
    "files": "**/*",
    "autoUpload": false,
    "autoDelete": false
  }
}
```

Figure 5: SFTP 2: .vscode/sftp.json file

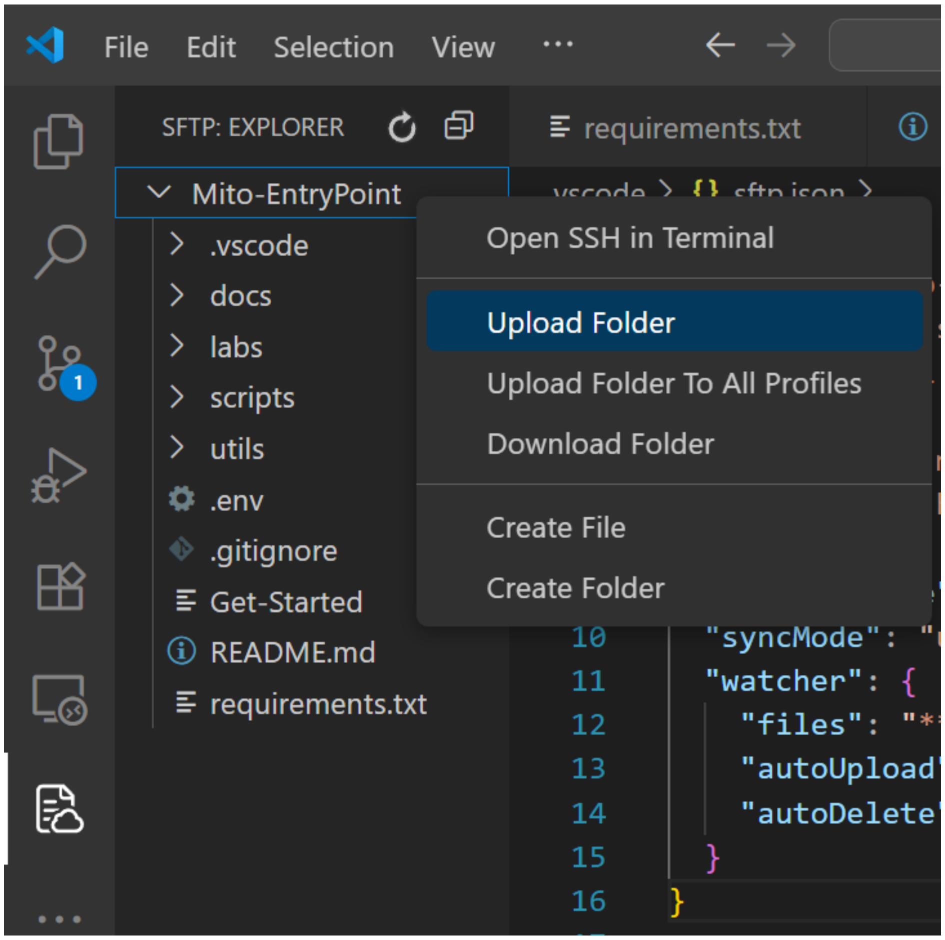
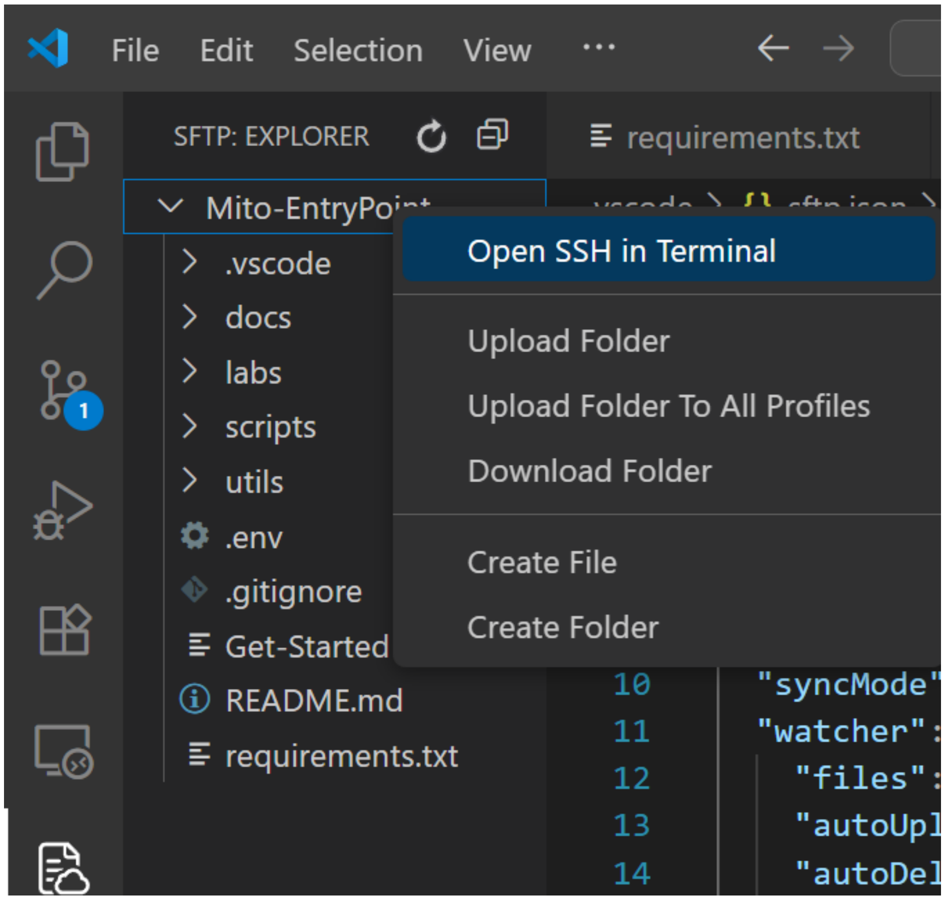
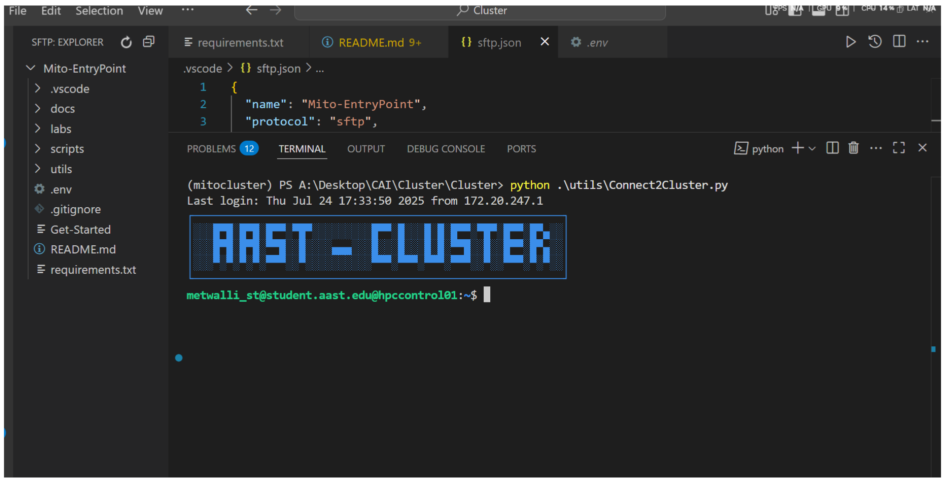


Figure 6: SFTP 3: Navigate to extension on the left and click refresh to see the cluster directory on your local or upload/download folders/files. Note: you can upload or download individual files



```
"syncMode"
"watcher":
  "files":
  "autoUpl
  "autoDel
```

Figure 7: SFTP 4: You can either open an SSH in Terminal or go to the next figure open the custom terminal provided by Mito Cluster repo.



```
{
  "name": "Mito-EntryPoint",
  "protocol": "sftp",
```

```
(mitocluster) PS A:\Desktop\CAI\Cluster\Cluster> python .\utils\Connect2Cluster.py
Last login: Thu Jul 24 17:33:50 2025 from 172.20.247.1
AAST - CLUSTER
metwalli_st@student.aast.edu@hpccontrol01:~$
```

Figure 8: SFTP 5: Logging to Cluster through terminal by Connect2Cluster.py which provides a custom shell and prompted without any password since it is already in sftp.json

5. Clone the course helper repo (optional)

```
cd ~/projects
```

```
git clone https://github.com/AIBabyTeaching/Cluster
```

This repo includes `Connect2Cluster.py`, a helper that spawns an SSH terminal using the saved credentials

6. Create a `.env` file *before* running Connect2Cluster.py The helper script reads connection parameters from environment variables, so place a hidden file named `.env` at your project root:

```
HOST=10.1.8.4
USER=metwalli_st@student.aast.edu
PASS=<your password>
```

Listing 2: Environment variables consumed by Connect2Cluster.py

Why this file?

- The **python-dotenv** package loads key–value pairs from `.env` into `os.environ` at runtime, sparing you from hard-coding secrets in the script
- Storing credentials outside source control is a widely recommended security practice
- `Connect2Cluster.py` (see the course repo) expects `HOST`, `USER`, `PASS` variables; without them it exits with an authentication error

After saving the file, continue with the next step to launch the helper script or use "Open SSH in Terminal" from the SFTP side-bar.

6. Open a terminal on the cluster Either run

```
python Cluster/Connect2Cluster.py
```

or right-click the profile in the SFTP panel and choose **"Open SSH in Terminal"**. VS Code attaches an interactive shell without prompting for the password again (the extension forwards the token)

7. Upload your project folder Inside the SFTP view, right-click the connection and select *Upload Files*. All local files are copied into `/home/$USER/project`. You can now submit `sbatch` jobs or edit remote scripts with normal VS Code autocomplete—changes sync on demand

Getting Started (Lab 0) — Environment Setup

A. Activate the shared toolchain

```
# run on new login shell
source /opt/llamaenv/bin/activate
python --version # should print "Python 3.11.x"
```

Listing 3: Enable cluster-wide Python tools

B. Create your own virtual environment

1. Navigate to $HOME and spin up a clean venv:

```
cd ~
python3.11 -m venv llamaenv_local
```

   This keeps course work isolated from the global site packages.

2. Activate and install dependencies (after uploading `requirements.txt`):

```
source ~/llamaenv_local/bin/activate
pip install --upgrade pip
pip install -r requirements.txt
```

   Installing inside the venv avoids permission issues and guarantees reproducibility.

C. Make activation automatic (optional)

- Alias in `~/.bashrc`

```
echo 'alias actllama="source ~/llamaenv_local/bin/activate"' >> ~/.bashrc
# now run:
source ~/.bashrc && actllama
```

  Simple and portable.

- Dir-based auto-activation (direnv, pyenv, etc.) — not required but handy if you juggle many projects; see cluster repo README for an example `.envrc`.

Reactivate with source ~/llamaenv_local/bin/activate whenever you open a new SSH tab.

0 Ground Rules

Do	Don't
Submit work via sbatch or srun	Run heavy code on the **login node**
Query resources with sinfo before launching	Over-request CPUs / RAM / time
Clean jobs using scancel	Leave zombie processes after logout

1 Core Command Reference

`sinfo -l`	Detailed partition/node list incl. reasons (`-R`)
`squeue -u $USER`	Show your queued/running jobs
`sbatch <script>`	Submit batch script
`salloc --pty bash`	Grab interactive allocation + open shell
`srun <cmd>`	Launch task inside allocation
`scancel <jobID>`	Cancel or signal a job
`sacct -j <jobID>`	Post-mortem accounting (elapsed, exit code)
`scontrol show job <jobID>`	Full job meta inc. failure *Reason*
`sstat -j <jobID>`	Live CPU/memory I/O stats for running job
`sprio --all`	View job scheduling priority factors
`watch -n10 "<cmd>"`	Auto-refresh any Slurm or Unix command

2 Smoke-Test Batch Script (30 s)

```
#!/bin/bash
#SBATCH -J smoke_test
#SBATCH -o smoke_%j.out
#SBATCH -e smoke_%j.err
#SBATCH -N 1 -n 1
#SBATCH -t 00:00:30 # wall-time

echo "Node: $(hostname)"
echo "Start: $(date)"
sleep 5
echo "End : $(date)"
```

Listing 4: Save as `smoke.sbatch`

3 Typical Workflow

1. **Check capacity**
   sinfo –o "%P %.6t %.6D %.9m"
2. **Submit**
   sbatch smoke.sbatch
3. **Watch queue**
   watch –n2 "squeue –u $USER"
4. **Cancel if needed**
   scancel <jobID>

3.1 What those four lines actually do

The four-step workflow is the minimal loop every Slurm user repeats all day: check capacity, launch work, watch it, and abort if needed. Here is what each command does under the hood:

`sbatch smoke.sbatch`
: Hands the file to the scheduler, which enqueues it and returns a JobID. Slurm then runs the script on the first node that fits your #SBATCH resource headers. Unlike `srun`, `sbatch` exits immediately, so your terminal is free for other work.

`watch -n2 "squeue -u $USER"`
: 'squeue' shows every pending (PD), running (R) or completing (CG) job you own; wrapping it with `watch` refreshes the output every two seconds until you hit `q`. This gives you a real-time dashboard without writing a loop yourself.

`scancel <JobID>`
: Sends SIGTERM (then, after `KillWait`, a SIGKILL) to all steps in the job and frees the allocation. Use this when you spot a typo in your script or the job is stuck in an infinite loop. You can also cancel everything you own with `scancel -u $USER`.

For deeper post-mortem data, chain `sacct -j <JobID>` to see wall time, exit code and node list once the job leaves the queue. If you need even more granularity while the job is still running, `sstat -j <JobID>` streams live CPU, memory and I/O counters per step.

4 Interactive Debug Session

```
salloc --time=01:00:00 -c 2 --mem=4G --pty bash
hostname
squeue -j $SLURM_JOB_ID
exit # releases resources
```

Listing 5: 60-minute allocation

5 Live Dashboards (watch)

```
# Partition status every 10 s
watch -n10 'sinfo -R -o "%P %.6t %.6D %.15f"'

# Your jobs with elapsed time + reason
watch -n10 'squeue -u $USER -o "%.9i %.2t %.10M %.15R"'
```

Installing pdsh (parallel shell)

pdsh lets you issue the *same* command on dozens of nodes concurrently via SSH, which is invaluable for quick health checks (pdsh –a uptime), pushing config tweaks, or harvesting logs from a whole partition in seconds. Because it speaks ordinary SSH, no extra daemons are required on the compute side, making it a lightweight companion to Slurm's batch engine.

Git-only, no sudo build

```
# 1) Grab the source
mkdir -p ~/tools && cd ~/tools
git clone https://github.com/chaos/pdsh.git # 3 MB
cd pdsh

# 2) Configure for SSH backend, install under $HOME/pdsh
./configure --with-ssh --without-rsh \
           --prefix=$HOME/pdsh # puts files in ~/pdsh/{bin,lib}
make -j4 # compile
make install # writes only to $HOME/pdsh
```

Listing 6: One-time install into $HOME/pdsh

Add to your $PATH (once)

```
echo 'export PATH=$HOME/pdsh/bin:$PATH' >> ~/.bashrc
source ~/.bashrc
pdsh -V # should print version string if install succeeded
```

What the `--prefix` flag does The −−prefix option tells Autotools to install everything under the given directory rather than the system default `/usr/local`: Pointing it to `$HOME/pdsh` keeps the cluster OS untouched while giving you a self-contained tree you can delete or move at will

Once these steps are complete you can, for example, pdsh −w hpc[11−14] 'df −h /scratch' to scan four nodes at once or pdcp −w ^hpc16 my.cfg /tmp to copy a file across them (see `man pdsh/pdcp` for full syntax).

6 Quick Exit Checklist

- squeue −u $USER → empty
- ps −u $USER on login node → no stray processes
- Sync results from /scratch back to external drive