JLab Computing Resources

Raiqa Rasool
EPSCI Computer Scientist

Computational Science & Technology Division

JLAB ACCOUNT: WHAT DO YOU GET ACCESS TO?

- JLab Web Portal
- Office 365 / Only Outlook(staff or intern only)
- Virtual Private Network (VPN)
- GitLab
- JupyterHub
- Virtual Desktop Infrastructure (VDI)
- ifarm
- slurm

WHAT SETUP IS REQUIRED OTHER THAN JLAB ACCOUNT?

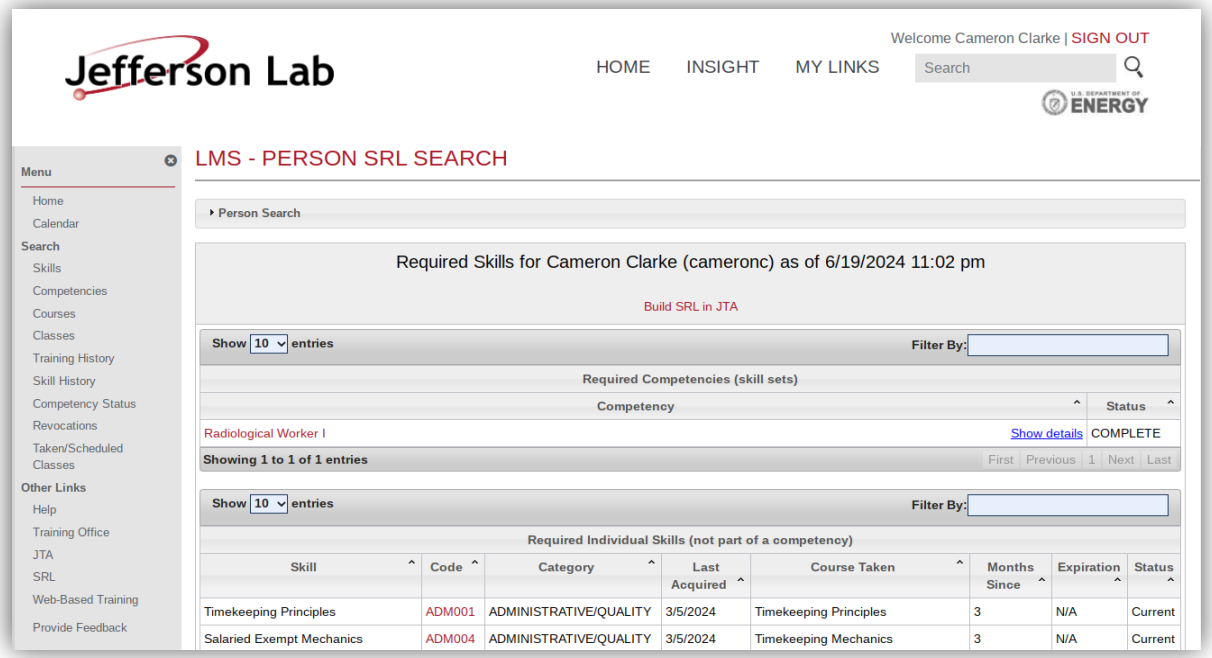
- No MFA (Multifactor Auth):
  1. JLAB Web Portal
  2. GitLab
- Microsoft Authenticator Setup
  1. Office 365/Outlook
  2. JupyterHub*
- MobilePass Phone App/USB Code Generator
  - VPN
  - VDI
  - ifarm*
  - slurm*

- Items marked with * also require approval from your Group/Hall Computing Coordinator:
  - https://jlab.servicenowservices.com/kb?id=kb_article_view&sysparm_article=KB0014686

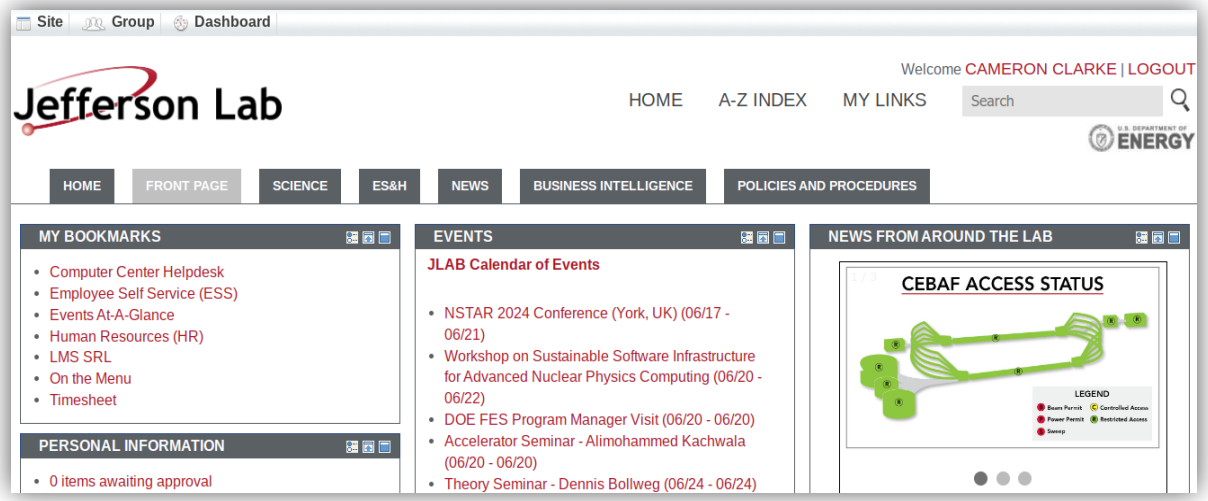
- For more details, refer to "Acquiring a JLab Account Presentation": https://indico.jlab.org/event/1069/attachments/13992/22649/acquiring_jlab_account.pdf

JLAB WEB PORTAL

- Accessing this web portal gives you access to:
  - Training (https://misportal.jlab.org/training/people/srl)
  - JList (staff directory)
  - Help-desk, library, and facilities tickets (https://jlab.servicenowservices.com)
  - Requisition system, etc.



https://misportal.jlab.org/training/people/srl

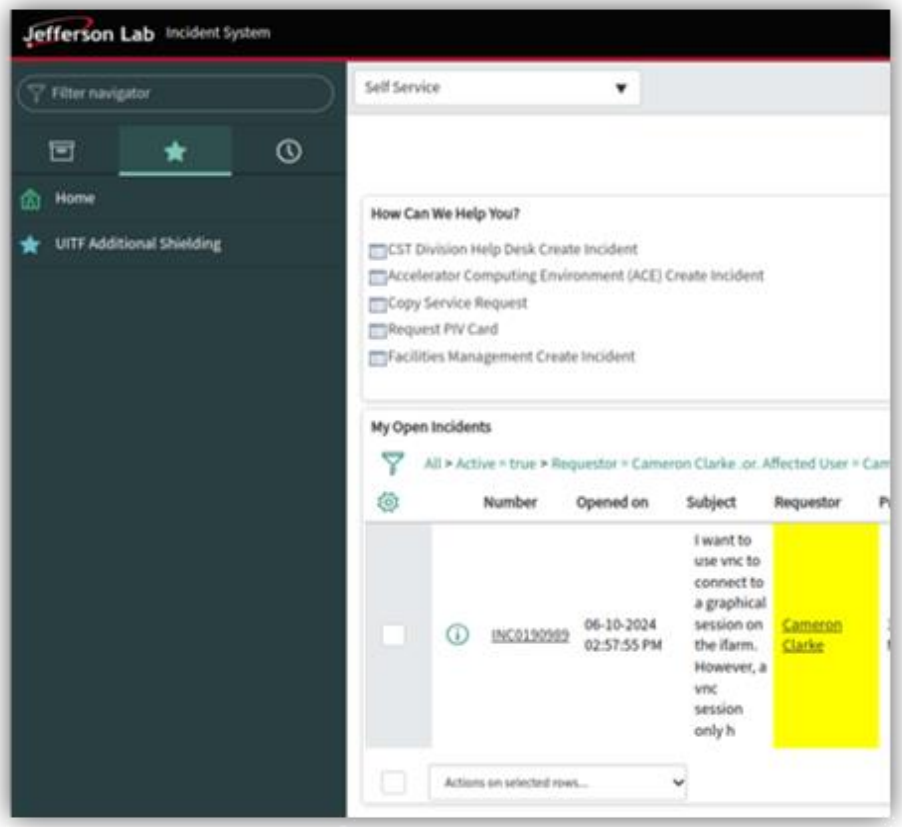


misportal.jlab.org/portal/insight

Jefferson Lab

JLAB WEB PORTAL

- Ask your sponsor or support staff at the lab for guidance
- Email helpdesk@jlab.org for computing assistance
- Email library@jlab.org for journal/book access questions
- Use the *servicenow* portal for facilities and other tickets

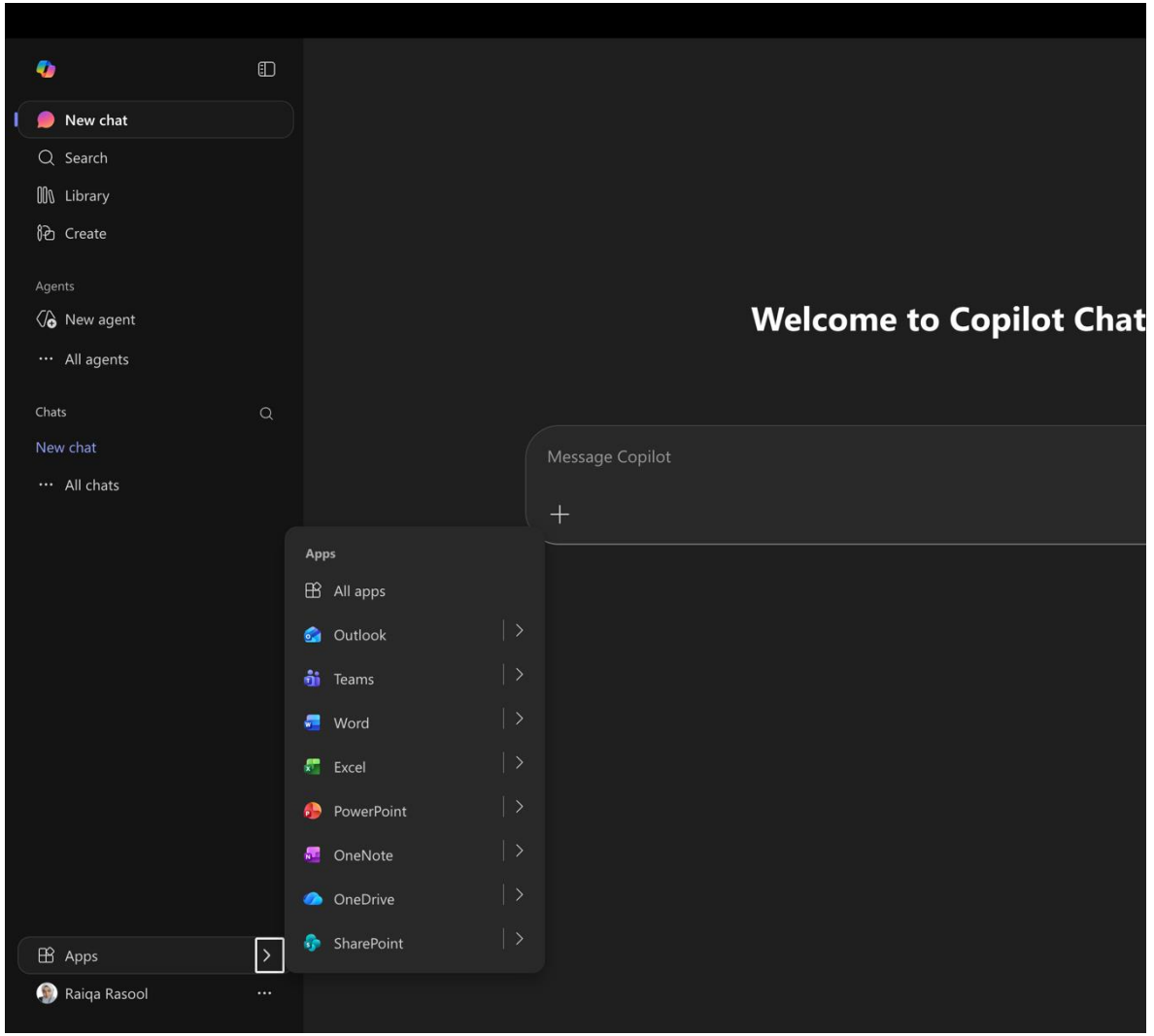


https://jlab.servicenowservices.com

Jefferson Lab

OFFICE 365/ OUTLOOK (ONLY FOR STAFF OR INTERNS)

- Microsoft Authenticator (MFA) setup is required for access to Microsoft Office 365
- There are two types of Office 365 access, that can be assigned by your supervisor or account requester:
  - **Full Office 365 access:**
    - Access to all Office apps (Teams, Outlook, PowerPoint, etc.)
    - Verify at: www.office.com
  - **Email-only access:**
    - Outlook access only
    - Verify at: outlook.office365.com

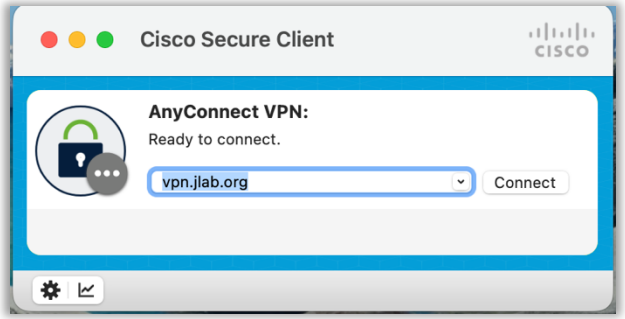


Full Office 365 access

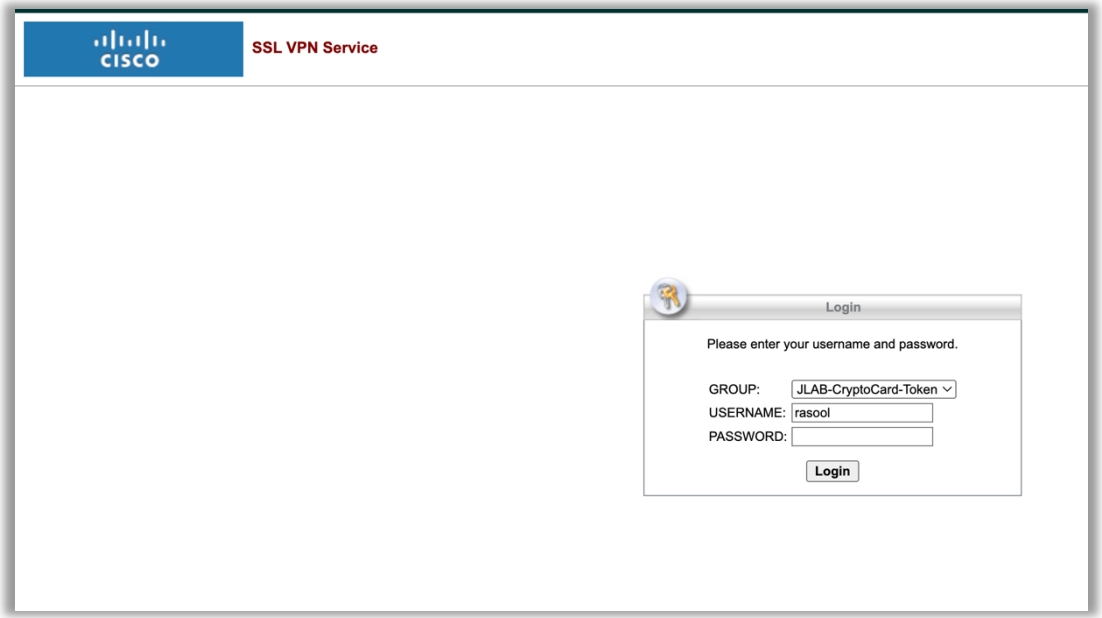
Jefferson Lab

VIRTUAL PRIVATE NETWORK (VPN)

- Provides offsite access to JLab resources as if you are on the internal network
- Enables access to internal-only systems (e.g., Insight protected pages)
- **How to connect:**
  - Web interface: http://vpn.jlab.org/
  - Desktop client (setup support available via Helpdesk)
- **Authentication required:**
  - MobilePASS app or USB code generator
    - Login format: **6-digit PIN + generated code**



Desktop Client

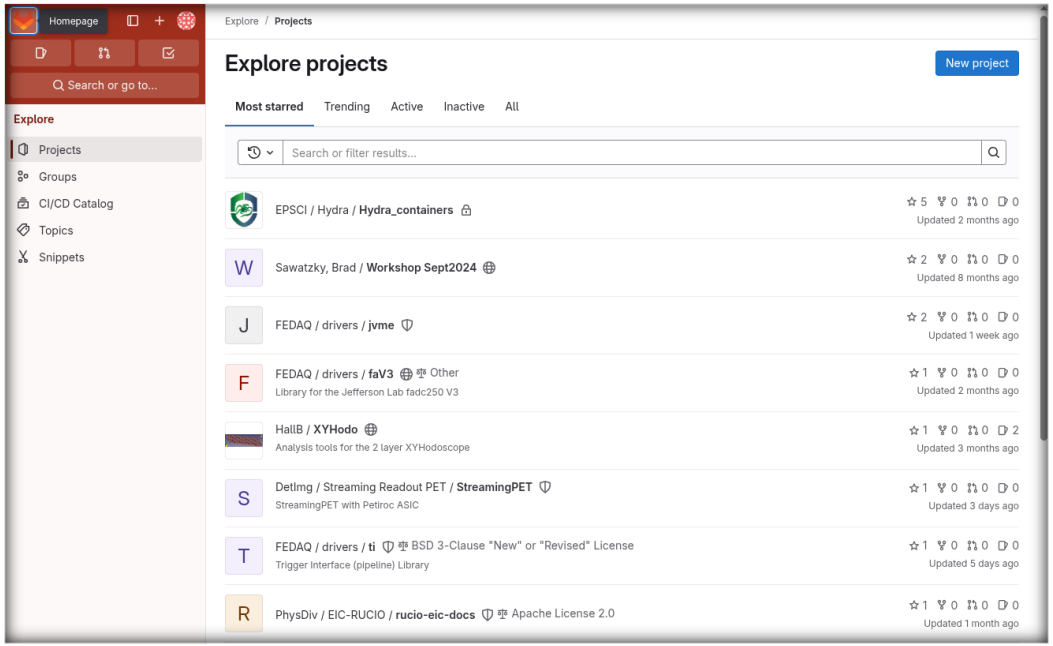


http://vpn.jlab.org/

Jefferson Lab

GITLAB (CODE.JLAB.ORG)

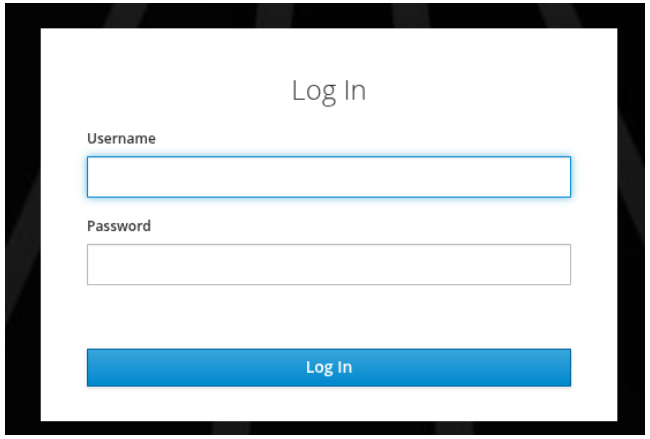
- JLab's hosted Git platform for version control and collaboration
  - https://code.jlab.org
  - Used for managing and sharing code repositories across projects
- **Login** with CUE username & password (no MFA required for web access)
- **Supports:**
  - Repository sharing and collaboration
  - CI/CD pipelines
  - Storage for containers, models, and artifacts
- **Hands-on session (later slides):**
  - Basic Git workflows (clone, commit, push) will be covered.
  - For advanced features & documentation: https://pages.jlab.org/scicomp/software/code-gitlab-docs/



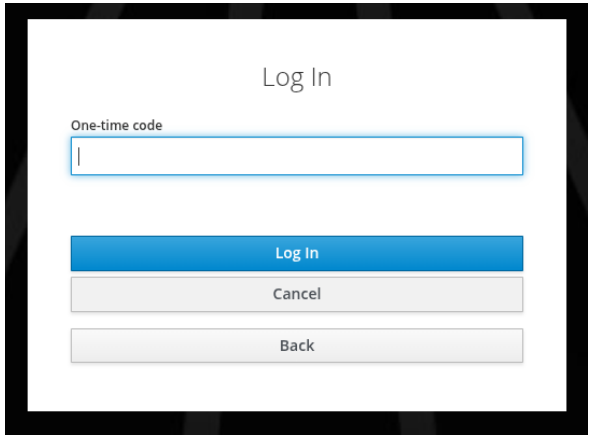
Jefferson Lab

JUPYTERHUB

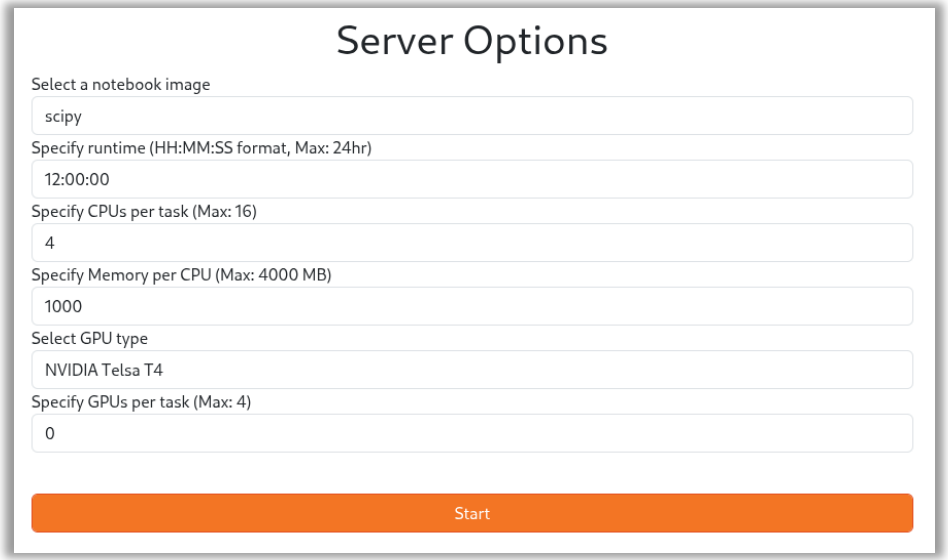
- https://jupyterhub.jlab.org - JLab's hosted Python notebook environment
- Provides access to GPUs and JLab file systems
- Supports interactive development, data analysis, and visualization workflows
- **Authentication:**
  - Requires MFA via Microsoft Authenticator
    - MFA setup occurs during first login
  - Contact Help Desk for setup assistance
- **Getting Started Guide:**
  - https://scicomp.jlab.org/docs/JupyterHub

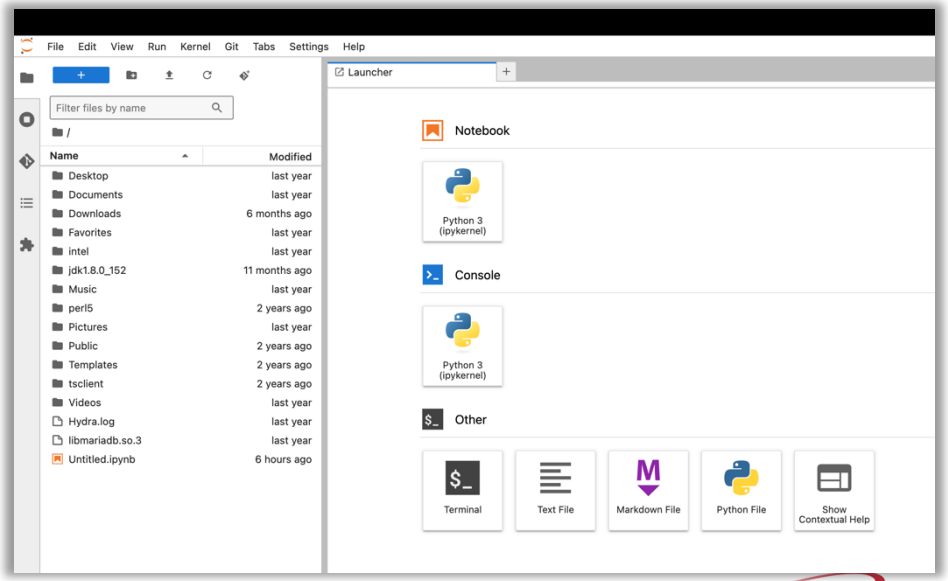


CUE Username & password



Microsoft Authenticator code



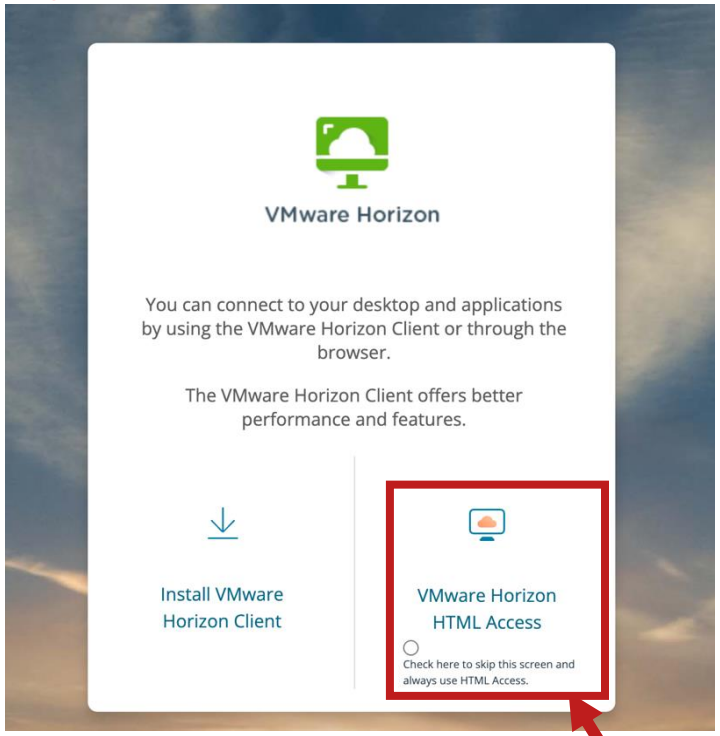


Jefferson Lab

VIRTUAL DESKTOP INFRASTRUCTURE (VDI)

- VDI accessed via VMware Horizon, provides a remote virtual machine linked to your JLab CUE account
- Provides a **RHEL9 Linux desktop** with access to shared resources
- Requires RHEL9 access request via Help Desk
- Two Options: VMware client (desktop) or Web browser (no setup). Both options will be shown on vdi.jlab.org

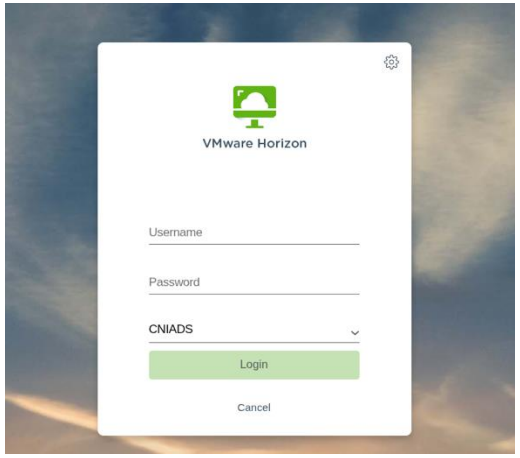
1.



https://vdi.jlab.org

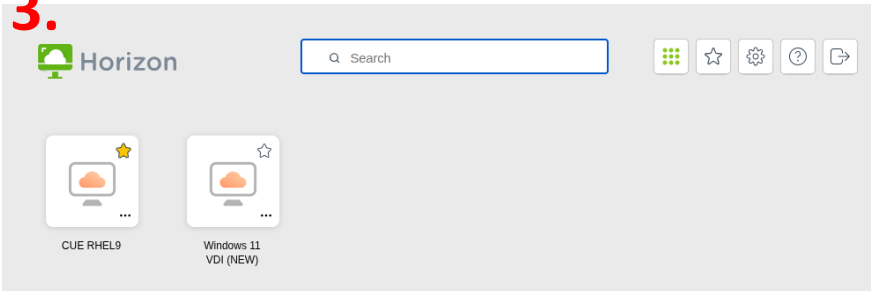
Web access

2.



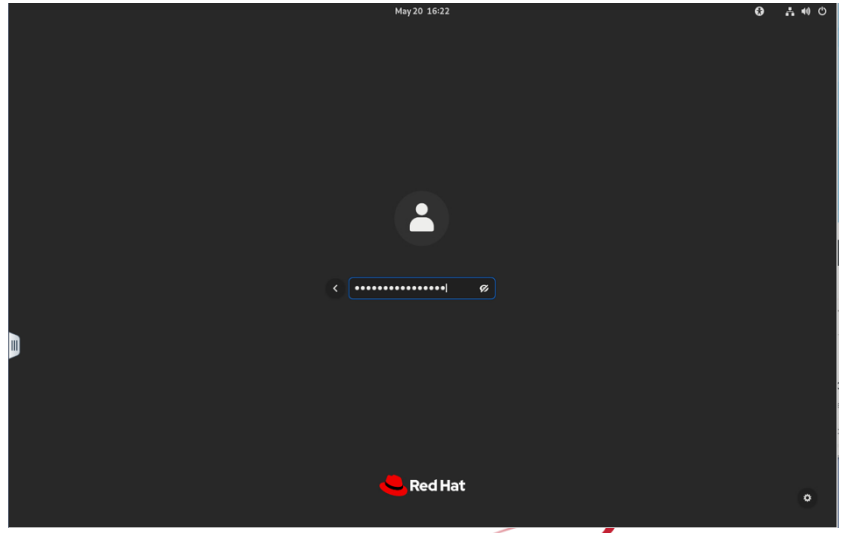
Log in with CUE Username and Password

3.



Select RHEL9 machine (Windows requires SmartCard!!)

4. Enter CUE Username
(Password = **MobilePASS** SAS MFA PIN+Code)



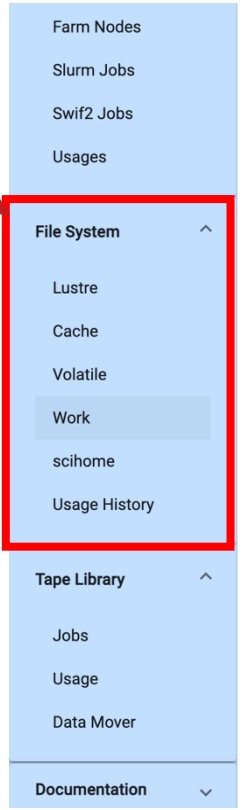
JLAB FILESYSTEMS

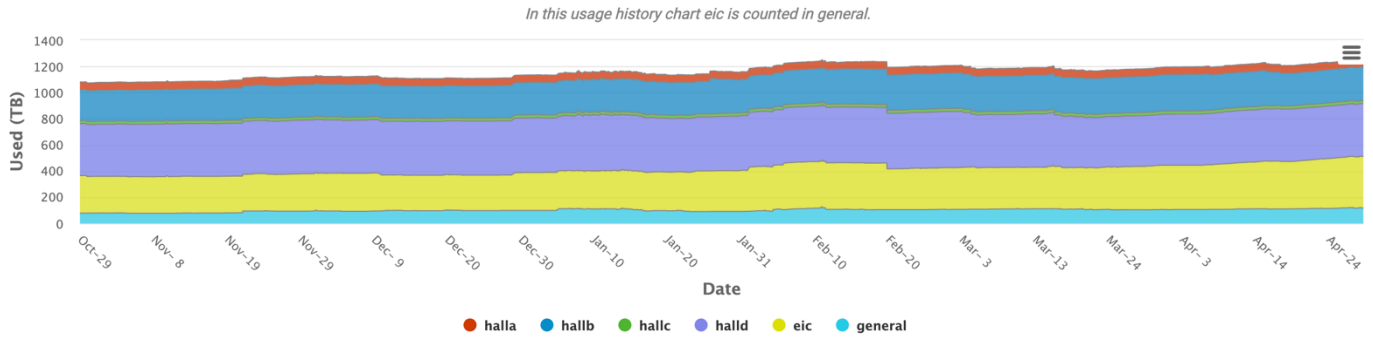
Path	Best Use	FS Type	Deletion	backup
/cache	Bulk I/O, Tape system interaction	Lustre	Once on tape	/mss
/volatile	Bulk I/O Temporary storage	Lustre	auto	NO
/work	Source code, DB files, exe's, etc. User Managed	NFS+ ZFS	manual	NO
/home	Dot files, personal documents, etc	NFS ssd	manual	YES
/farm_out	Farm job stdout/stderr	NFS ssd	auto	NO
/group or /scigroup	Source code Papers, thesis, analysis scripts	NFS ssd	Manual	YES
/scratch/$SLURM_JOB_ID	Farm job I/O to node local disk	ssd	auto	NO
/u/scratch	*CUE scratch. Deprecated (Unavailable on el9)*			
/cvmfs	Software stack, Configuration.			

Manual vs Auto: /home and /work require *manual deletion* when full (hard limits), while /cache and /volatile always allow writes but are *automatically cleaned up* when over quota (soft limits).

JLAB FILESYSTEMS – DISK USAGE

- Web interface for checking used and available quota for JLab filesystems:
  - https://scicomp.jlab.org/scicomp/workDisk





Disk Name/User Name	File System/Server	Quota (GB)	Used (GB)	Used/Quota
clas	zfs@scifs2102	46,075	29,257	63.50%
clas12	zfs@scifs2102	169,670	154,890	91.29%
clas12b	ceph24	153,600	28,822	18.76%
eic3	zfs@scifs2104	498,428	367,786	73.79%
general	ceph24	188,185	107,666	57.21%
halla	zfs@scifs2102	81,972	50,337	61.41%
hallb	zfs@scifs2102	46,057	25,862	56.15%
hallc	zfs@scifs2102	36,427	22,251	61.08%
halld	zfs@scifs2101	409,600	371,458	90.69%
halld4	ceph24	51,200	3,850	7.52%
		1,681,214	1,162,180	

- Hands-on Session (later slides):
  - How to check disk usage status on ifarm
    - Overall filesystem usage
    - User-specific usage
  - Focus: debugging “quota exceeded” issues

JLAB FILESYSTEMS – HOW TO ACCESS

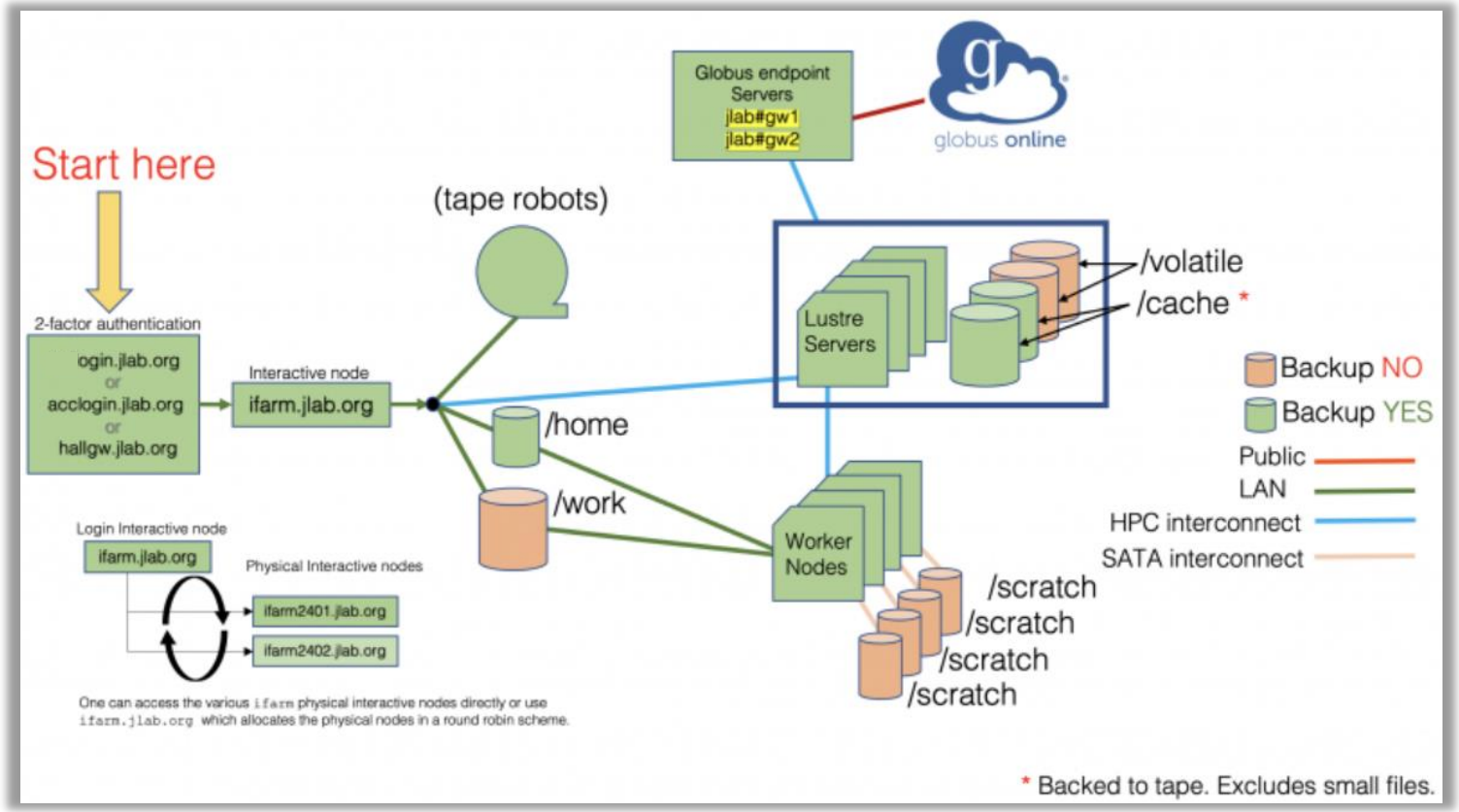
- The following systems provide access to the same shared filesystems **`/home/<username>, /work, /group, /cache, /volatile`**
  - VDI
  - JupyterHub
  - ifarm - *recommended access method*
  - slurm
- Same files, same data, regardless of where you access them
- Work in one system, access it from another
- Important**:**
  - Compute resources does differ across systems, .e.g.
    - ifarm nodes: ~256 physical threads per node
    - VDI (RHEL): ~2 physical threads
    - **Systems share storage, but not compute power**

- VDI & JupyterHub: already covered, VDI will be revisited briefly in hands-on
- ifarm: next slide (also main hands-on focus)
- slurm: May 14 session

JLAB IFARM

JLab's shared HPC:

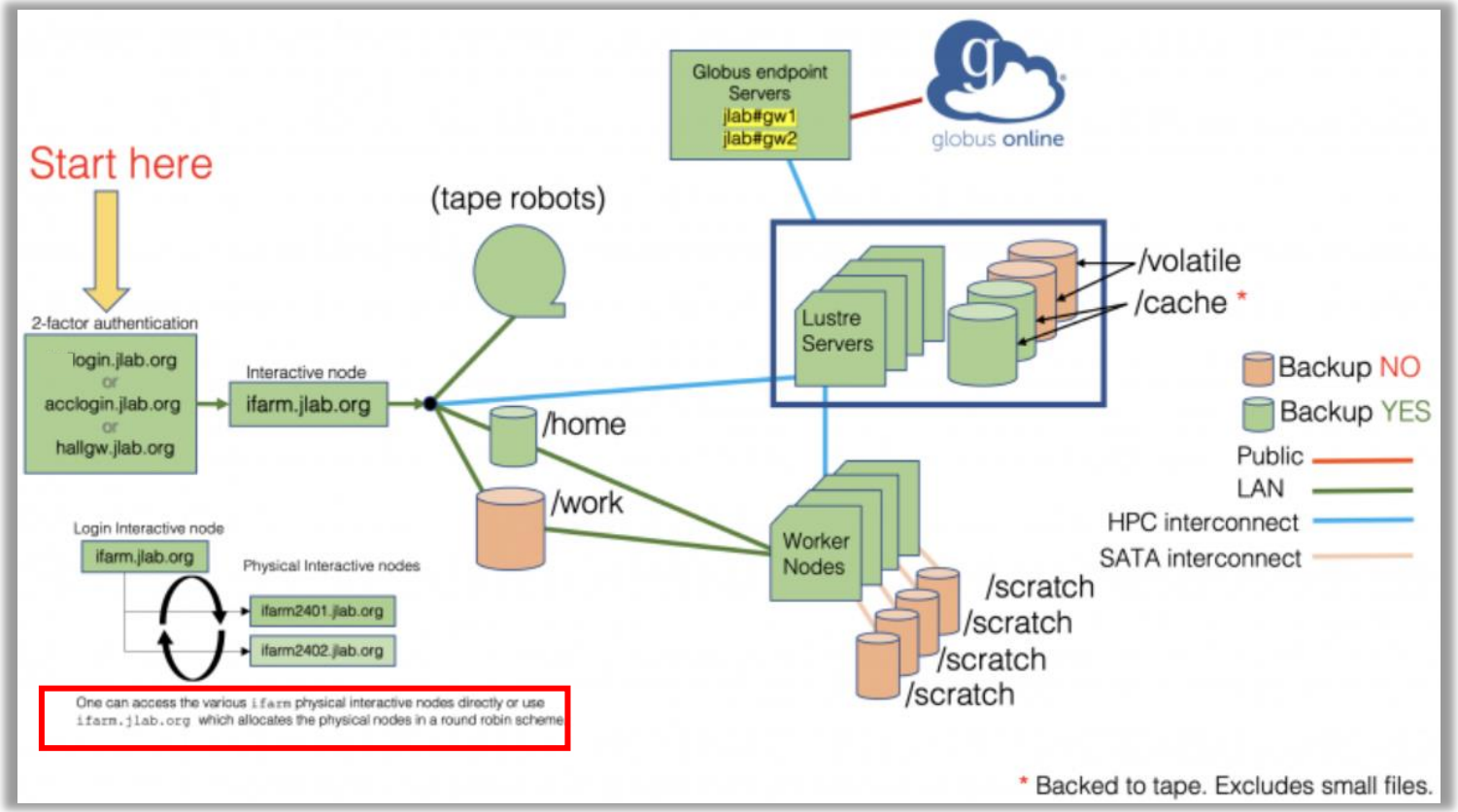
- Computing cluster "farm"
- Has interactive capability through "ifarm"
- Two nodes (ifarm2401, ifarm2402)
- Provides access points for testing, etc. before sending off to production farm
- Interactive nodes:
  - AMD EPYC 9554 (Zen 4 "Genoa")
  - 256 threads (2 sockets × 64 cores × 2 threads/core)
  - 3.1 GHz base / 3.76 GHz max
  - 1.5 TB memory



Source: https://scicomp.jlab.org/docs/getting_started

JLAB IFARM

- These nodes are shared among multiple users
- They are powerful but intended for:
  - Short interactive computation
  - Code testing and development
  - Job submission preparation
- Heavy or long-running workloads should be executed on worker nodes via slurm



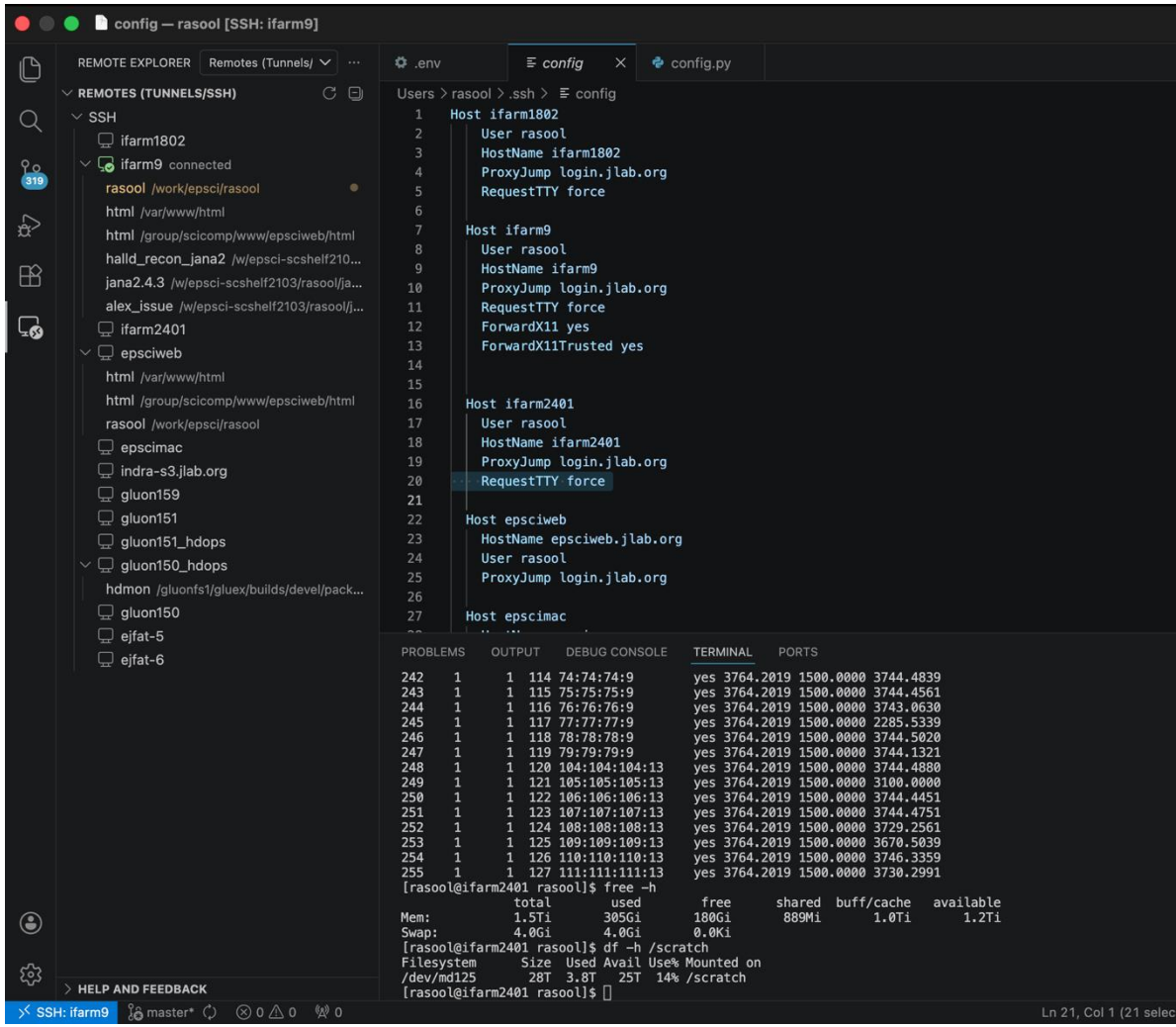
Source: https://scicomp.jlab.org/docs/getting_started

JLAB IFARM - ACCESSING VIA VSCODE (REMOTE SSH)

- One of the easiest ways to *access ifarm* is *using the Remote SSH extension in VS Code*
- We will go through the full setup in the hands-on session

Why VS Code Remote SSH?

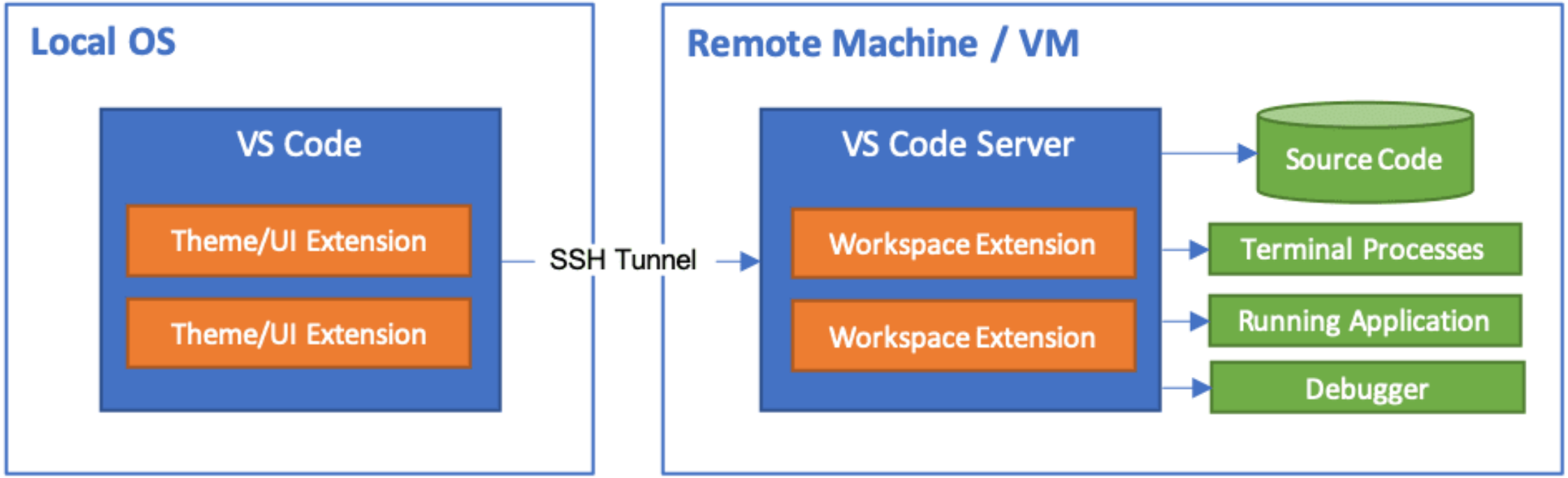
- It provides strong flexibility for remote development
  - Work on remote servers as if they are local
  - Open files, terminals, run code, and debug seamlessly
  - Run and manage multiple terminals/folders without repeatedly reconnecting
  - Same editor experience across local and remote environments
  - Previously opened folders are saved for quick access
  - Multiple hosts can be configured in ~/.ssh/config
    - Connect directly without remembering proxy jumps or complex SSH commands



Jefferson Lab

JLAB IFARM - ACCESSING VIA VSCODE (REMOTE SSH)

Source: https://code.visualstudio.com/docs/remote/ssh



SSH is like a secure, encrypted tunnel over the internet (TCP) that lets you work on a remote computer as if it were right in front of you.

Remote SSH Extension uses SSH to connect securely to a remote machine and starts a small VS Code server there. That VS Code server mainly does all the work—opening files, running code, and handling commands. Your VS Code on your laptop just conveys whatever actions you are taking to this server over SSH and shows you the output of your actions. Everything happens so fast that it feels like you are working directly on the remote machine.

HANDS-ON SESSION

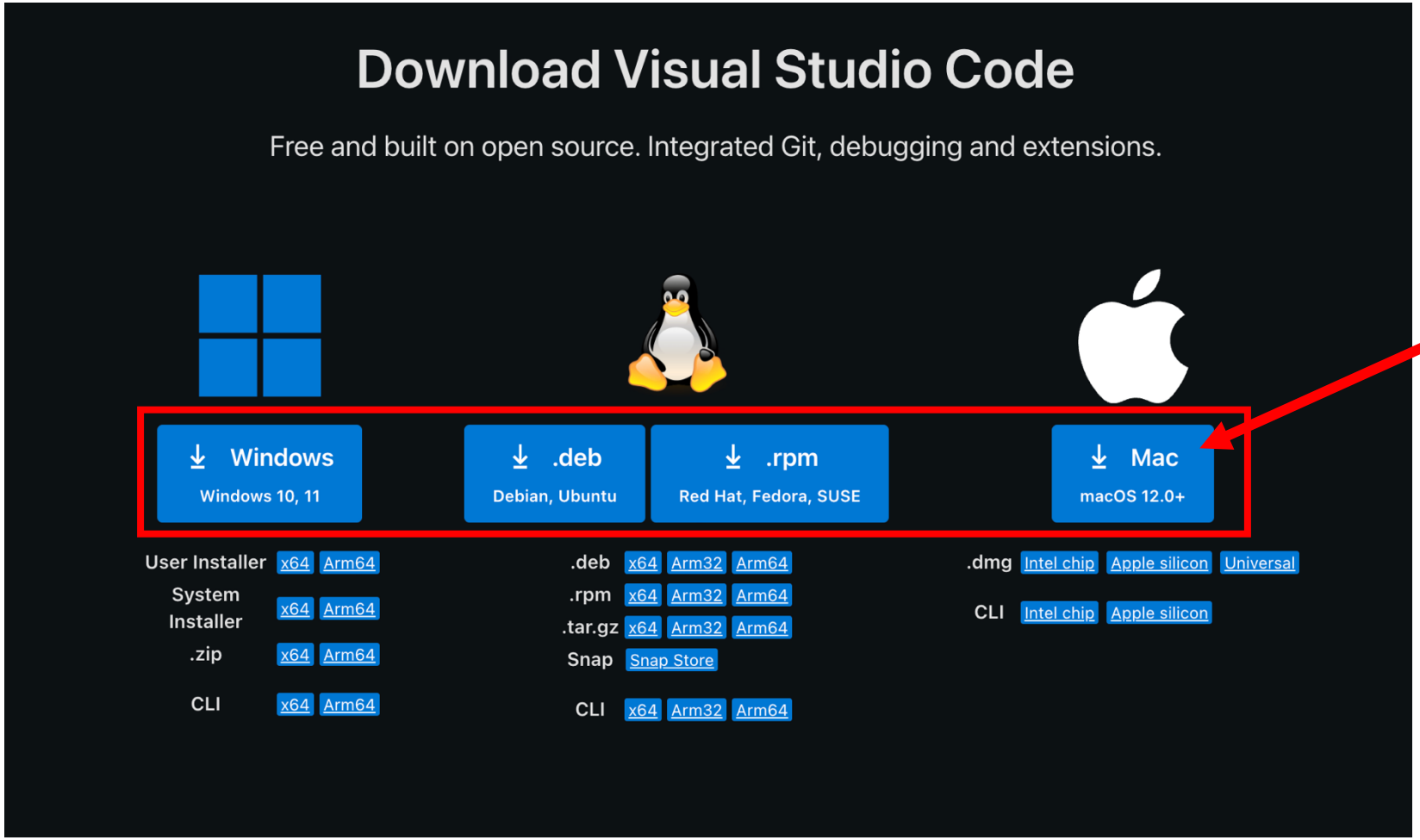
OUTLINE

1. VS Code Setup & Basics
   - Install VS Code, open folders, use terminal.
2. Remote SSH to ifarm
3. Check Filesystems Disk Capacity & Usage
4. Port Forwarding
   - Access remote network apps locally via forwarded ports.
5. X11 Forwarding
   - Access GUI apps running on ifarm from local machine.
6. GitLab First Commit & Push

SECTION 1: VSCODE SETUP & BASICS

INSTALLING VSCODE

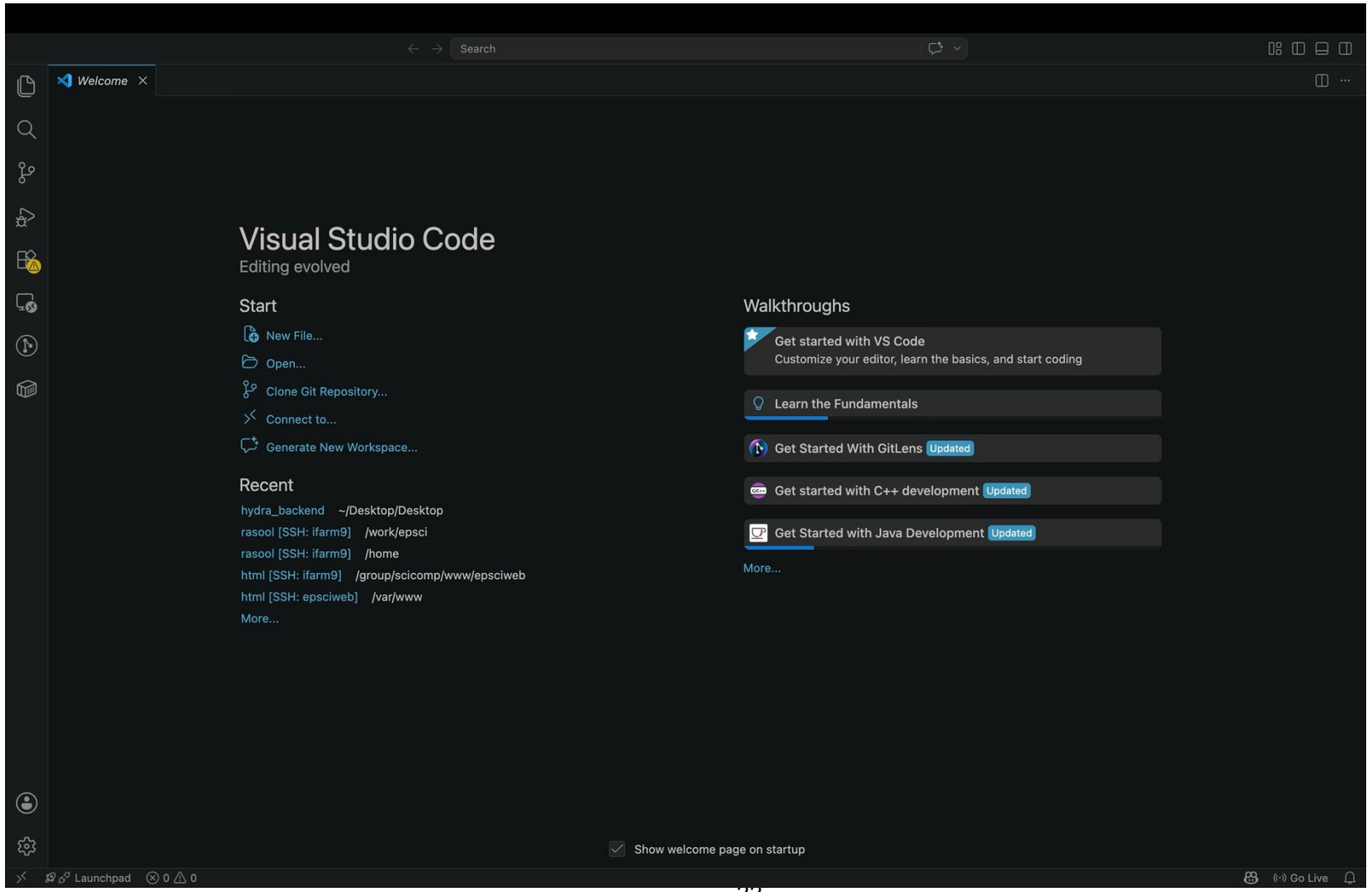
- Go to https://code.visualstudio.com/download and download the installer that matches your operating system.



Click one of these with download icon based on your OS

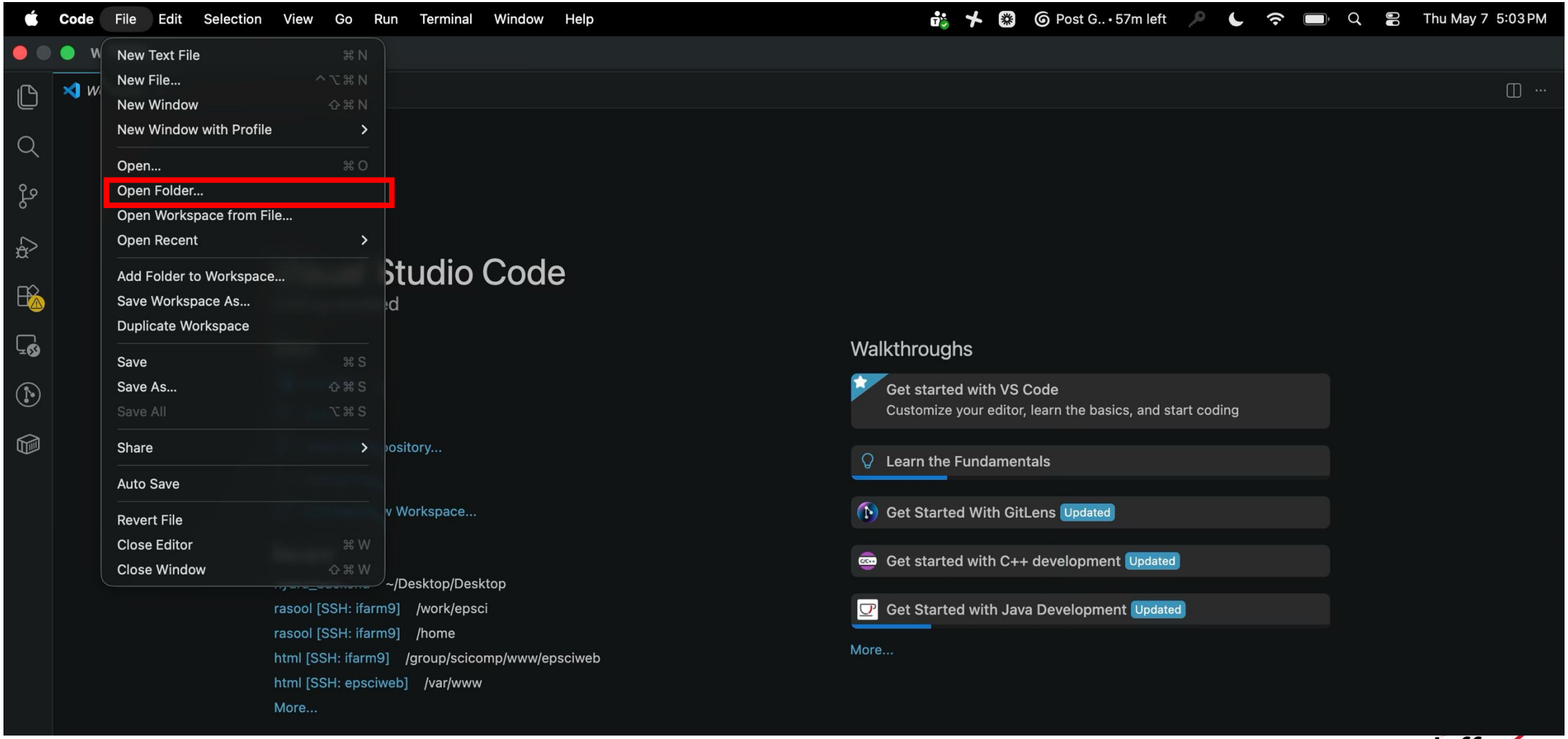
VSCODE – OPENING YOUR FIRST FOLDER

- Once the download is complete, click the downloaded file and follow the installation instructions.
- After installation, launch VS Code by searching it in your operating system's search bar.



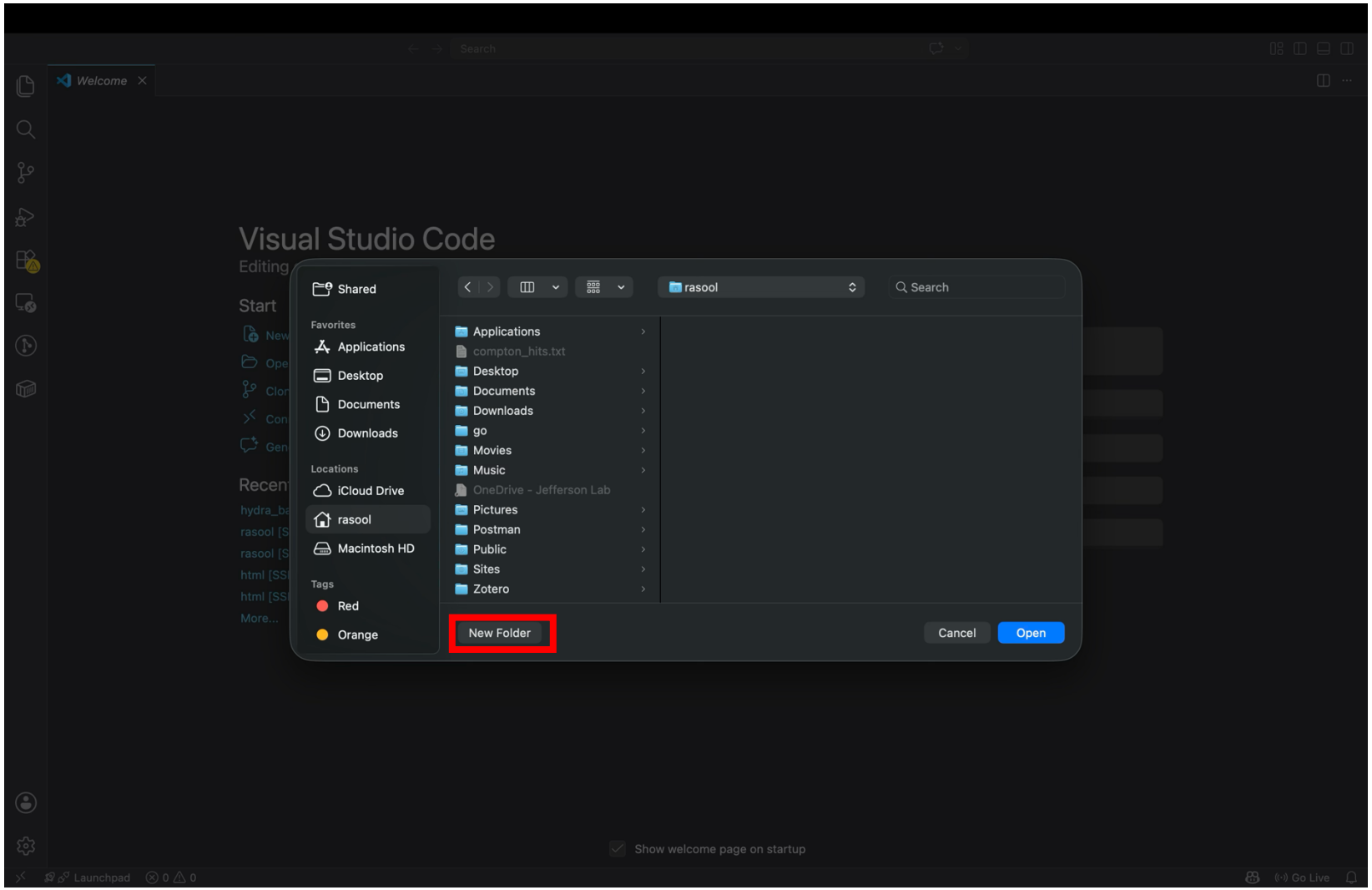
VSCODE – OPENING YOUR FIRST FOLDER

- Once VS Code is open, go to **File > Open Folder…** from the menu.
- Create a new folder named **vscode101**, select it, and it will open inside VS Code.



VSCODE – OPENING YOUR FIRST FOLDER

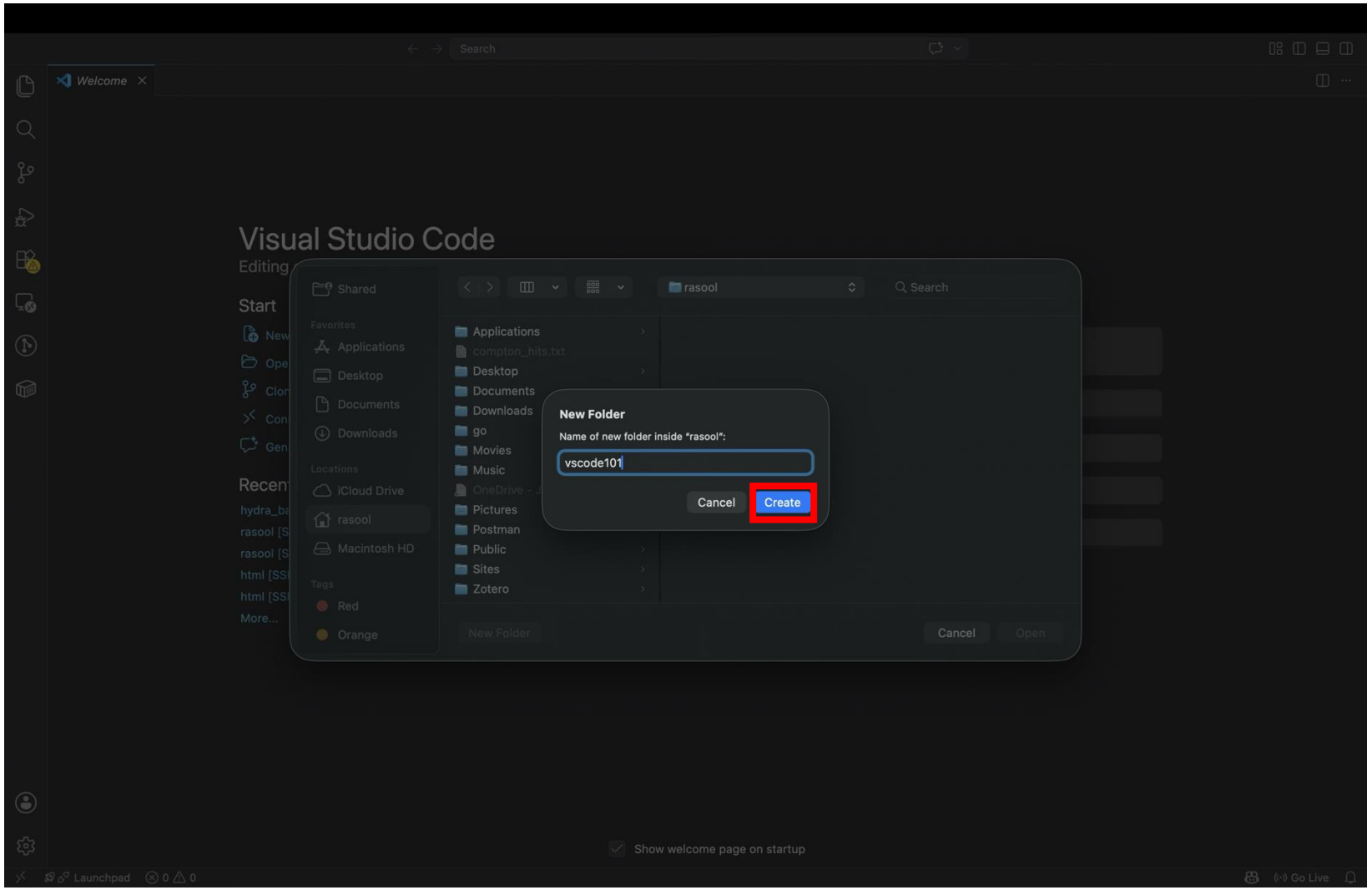
- Once VS Code is open, go to **File > Open Folder…** from the menu.
- Create a new folder named **vscode101**, select it, and it will open inside VS Code.



Jefferson Lab

VSCODE – OPENING YOUR FIRST FOLDER

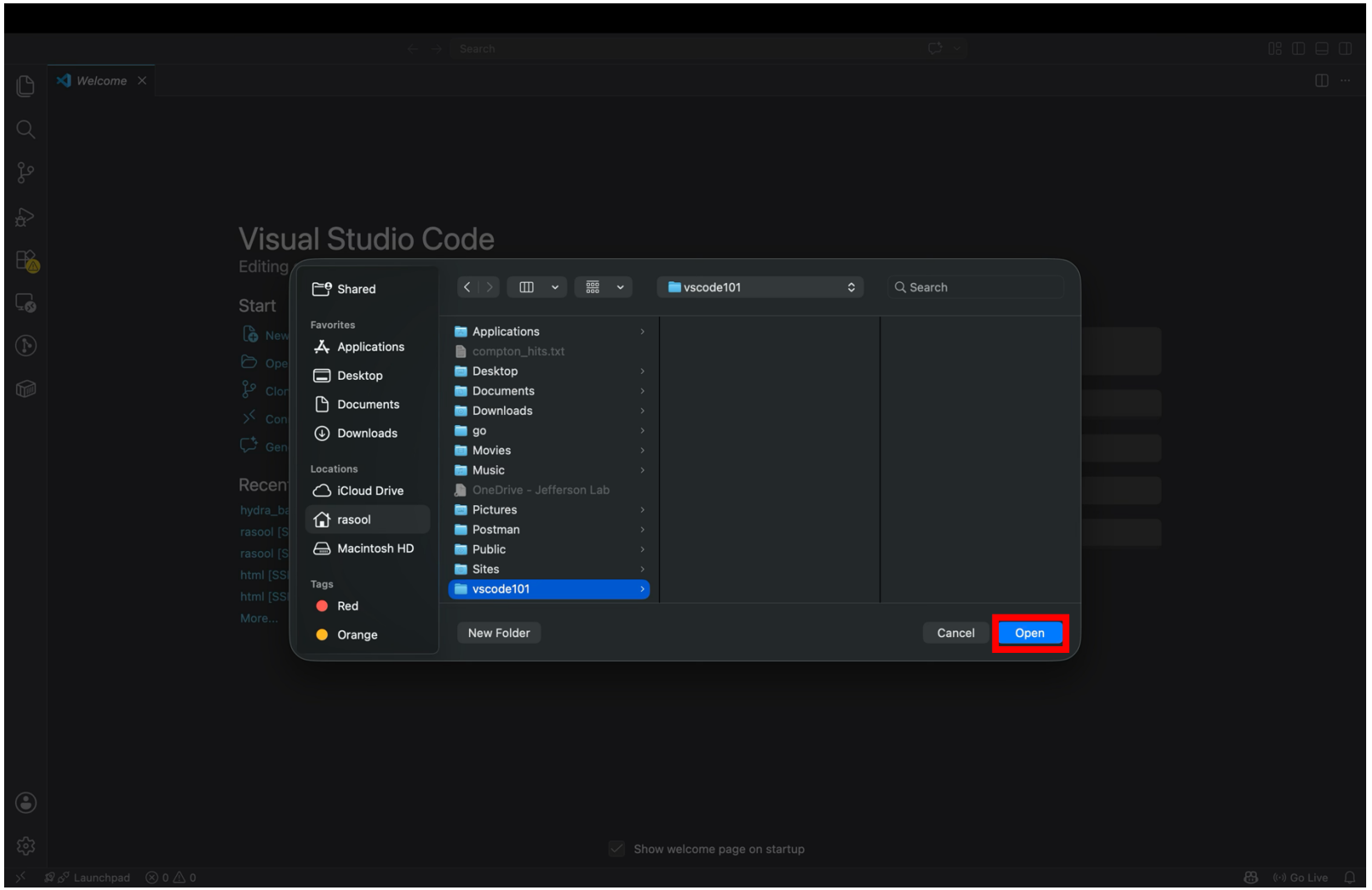
- Once VS Code is open, go to **File > Open Folder…** from the menu.
- Create a new folder named **vscode101**, select it, and it will open inside VS Code.



Jefferson Lab

VSCODE – OPENING YOUR FIRST FOLDER

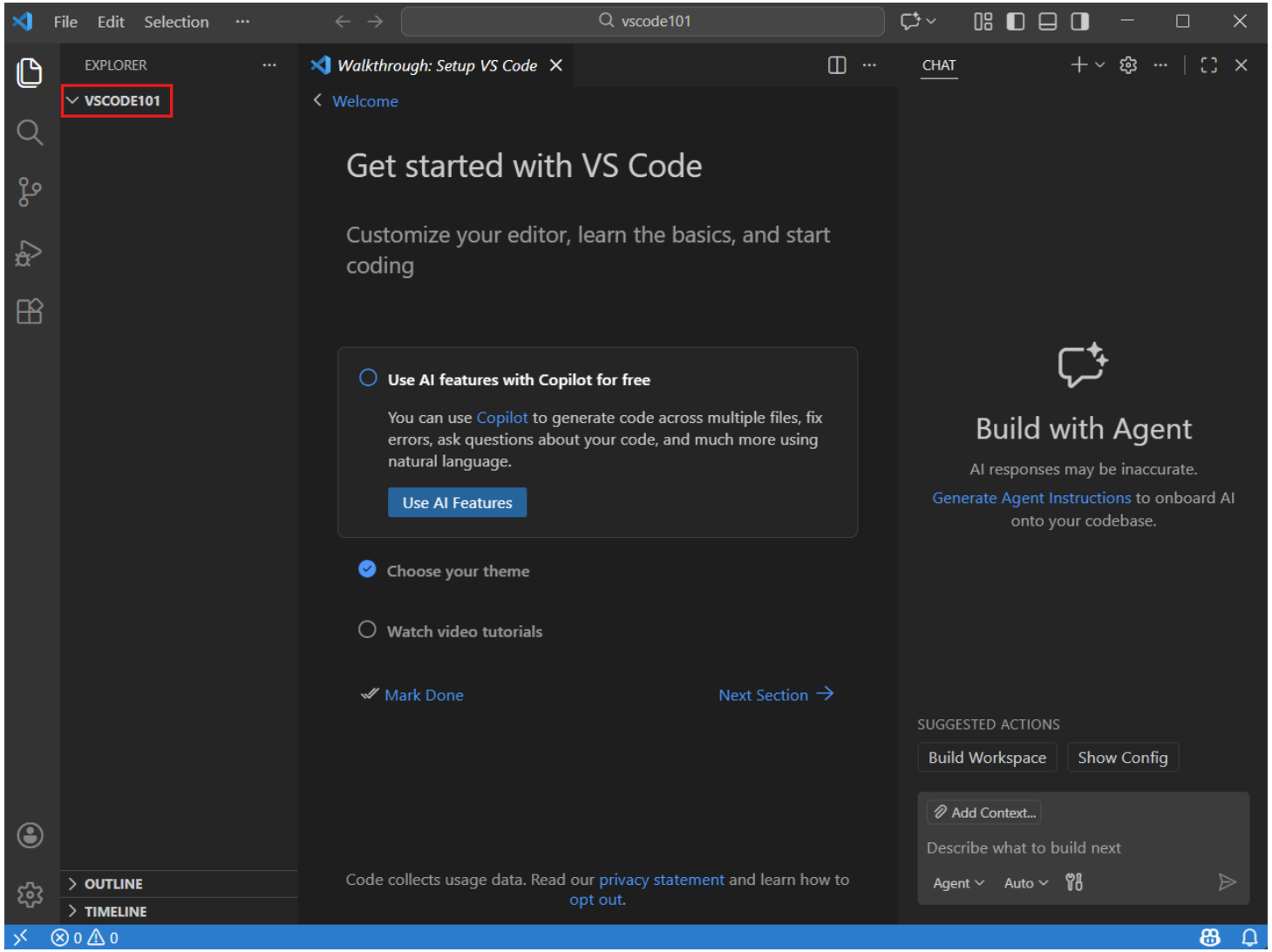
- Once VS Code is open, go to **File > Open Folder** from the menu.
- Create a new folder named **vscode101**, select it, and it will open inside VS Code.



Jefferson Lab

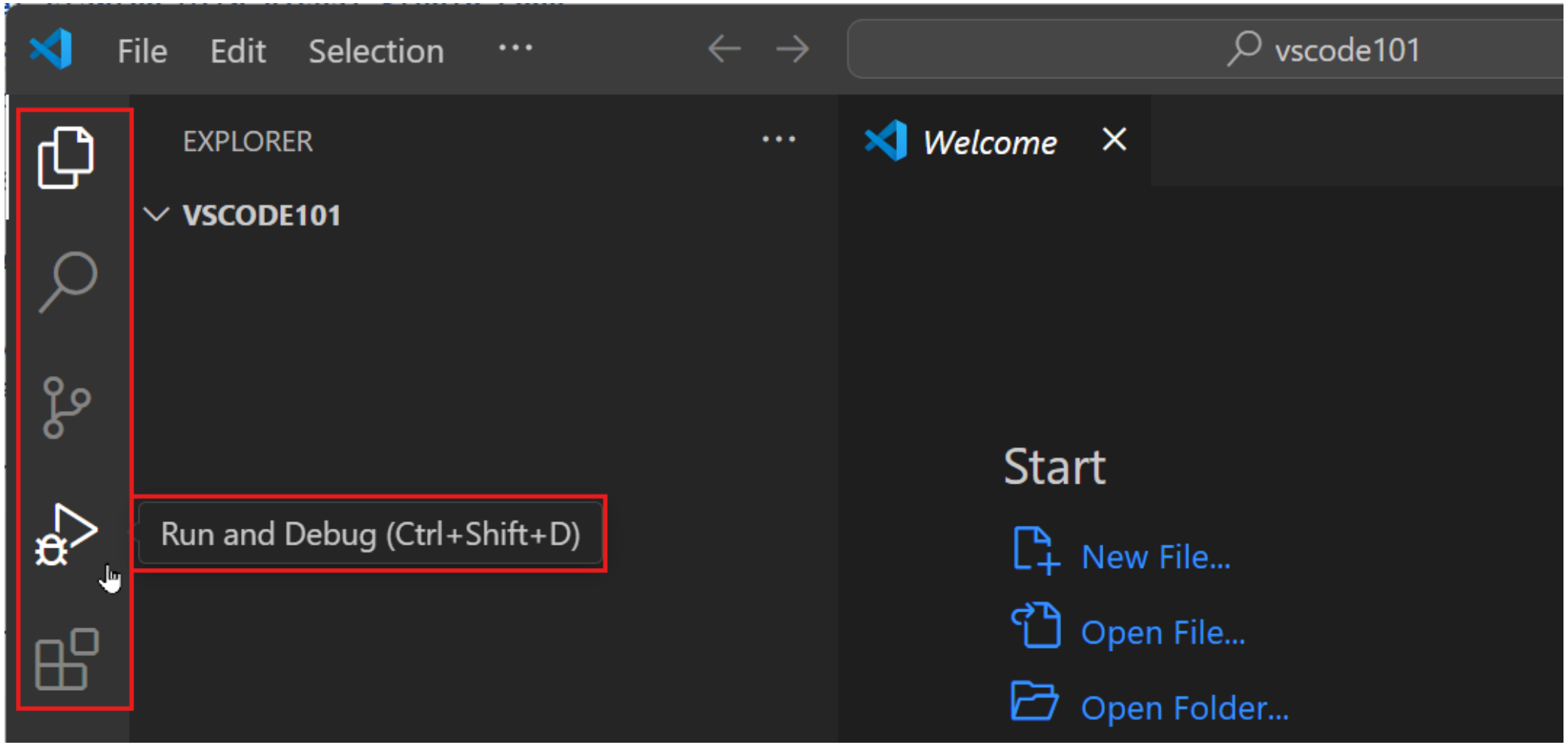
VSCODE – EXPLORER VIEW

- You should now see the Explorer view on the left, showing the name of the folder.
- You'll use the Explorer view to view and manage the files and folders in your workspace.



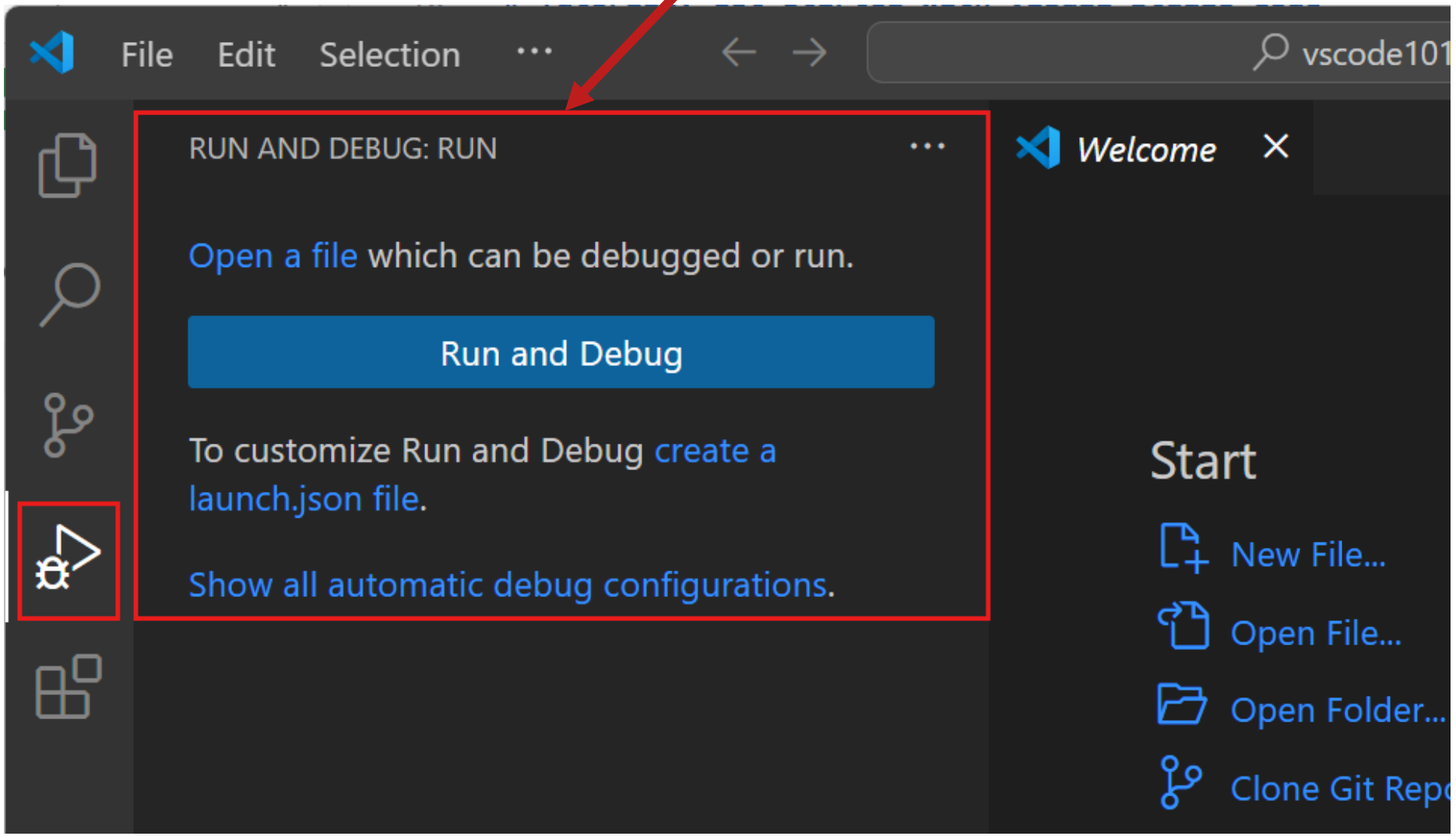
VSCODE – ACTIVITY BAR

- Use the Activity Bar to switch between different views.
- **Tip:** Hover over the Activity Bar to see the name of each view and the corresponding keyboard shortcut. You can toggle a view open and closed by selecting the view again or pressing the keyboard shortcut.



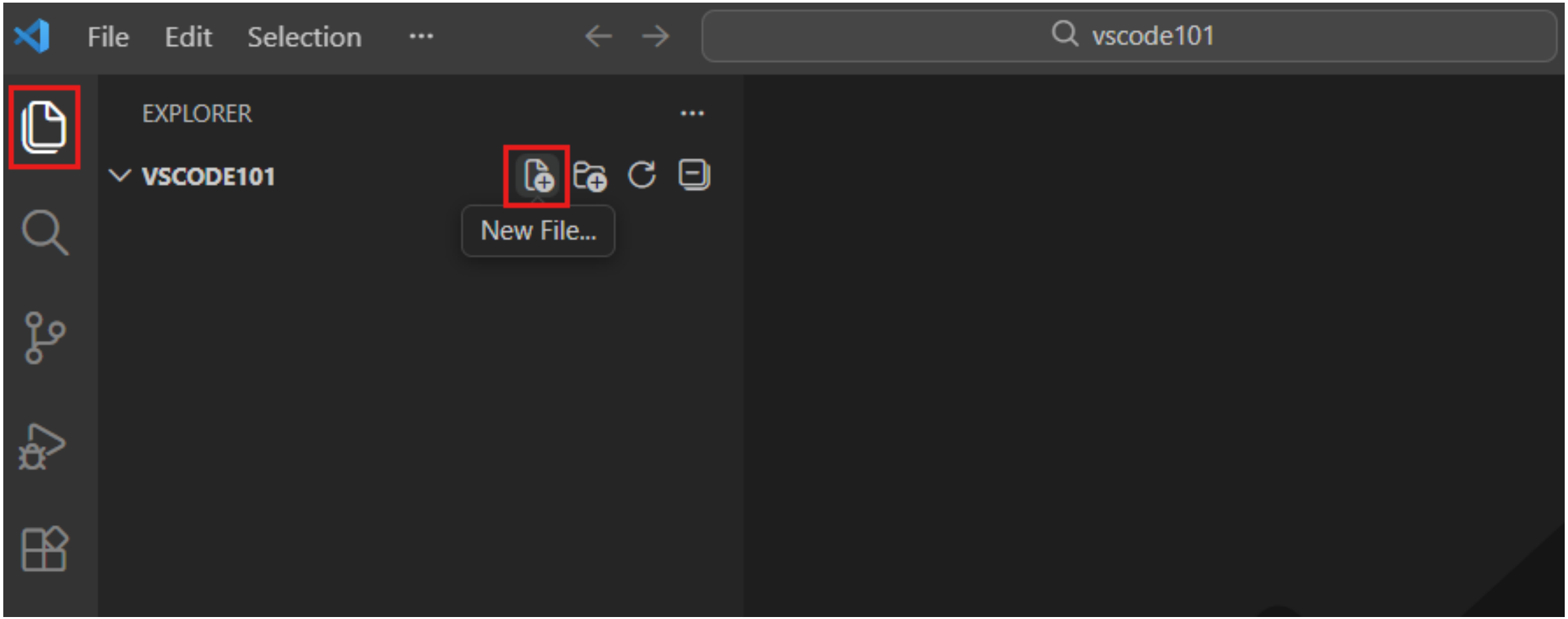
VSCODE – ACTIVITY BAR

- When you select a view in the Activity Bar, the **Primary Side Bar** opens to show view-specific information.



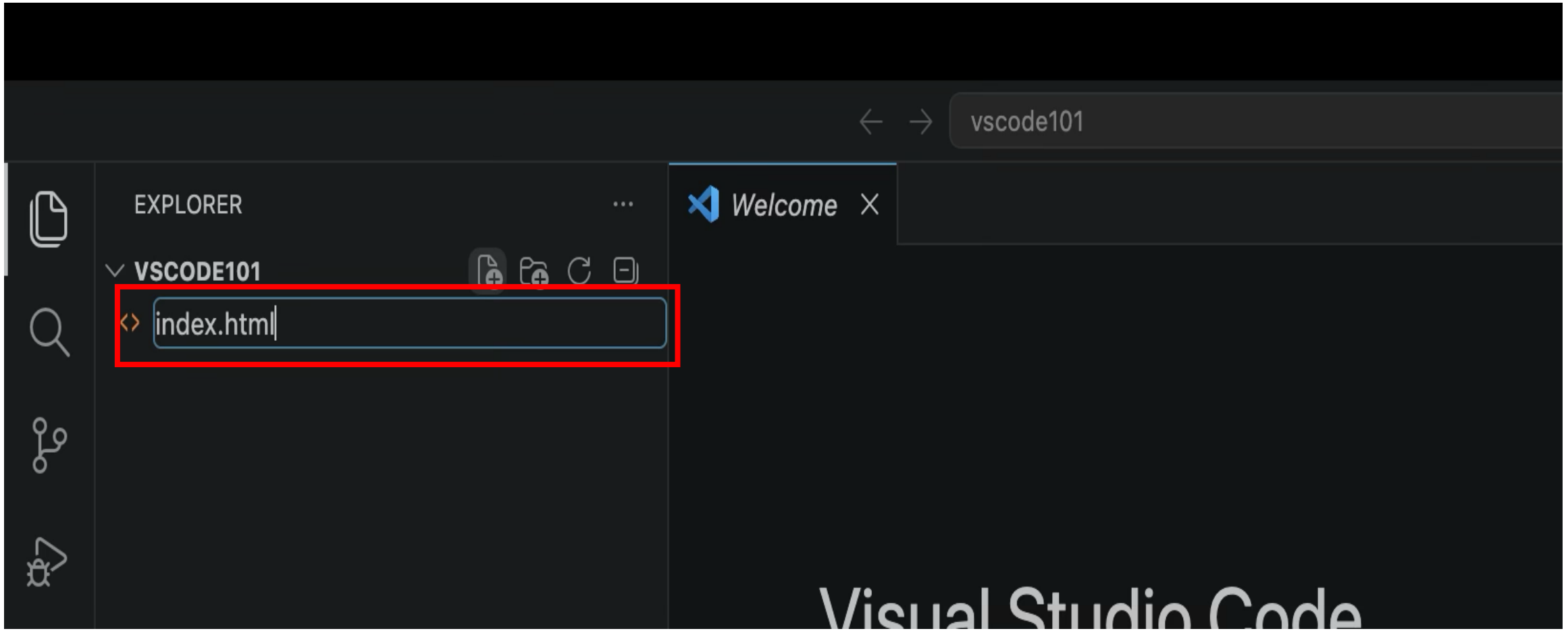
VSCODE – VIEW AND EDIT FILES

Select the Explorer view in the Activity Bar, and select the **New File** button to create a new file in your workspace.



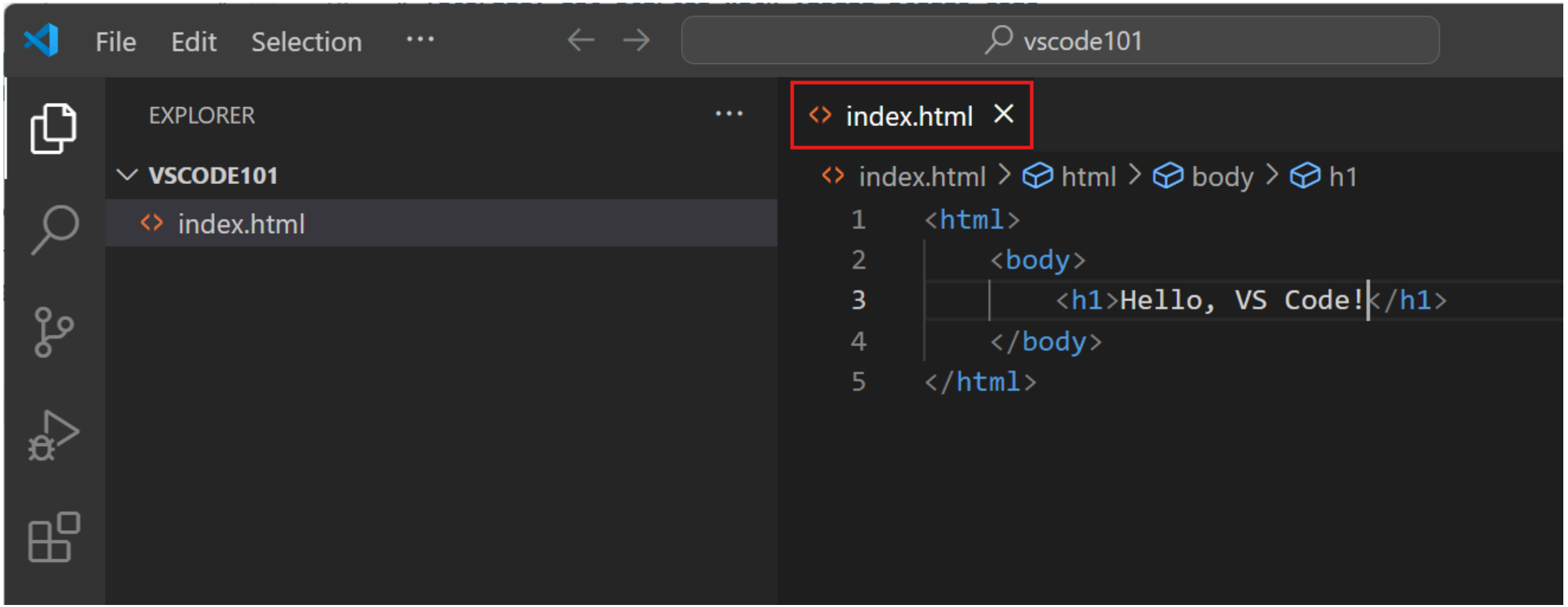
VSCODE – VIEW AND EDIT FILES

- Enter the name index.html and press Enter. A file will be added to your workspace and an Editor will open in the main area of the window.



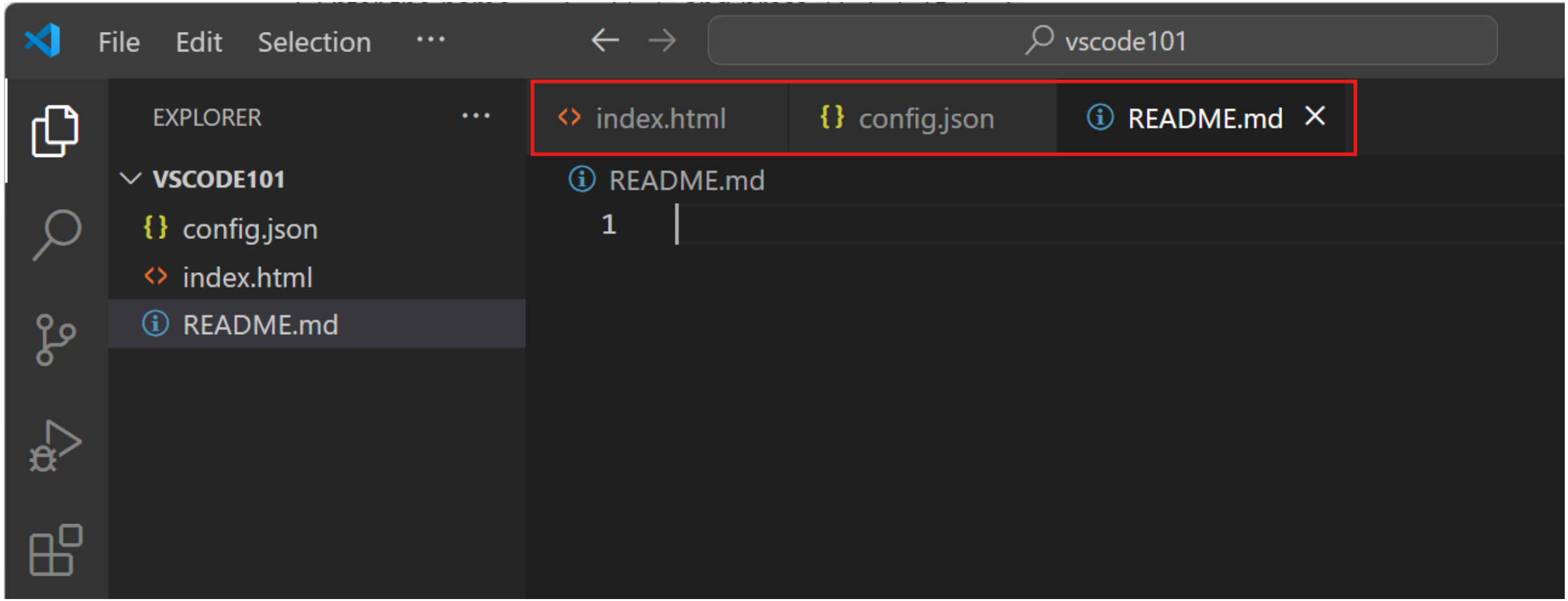
VSCODE – VIEW AND EDIT FILES

- Start typing some HTML code in the index.html file.



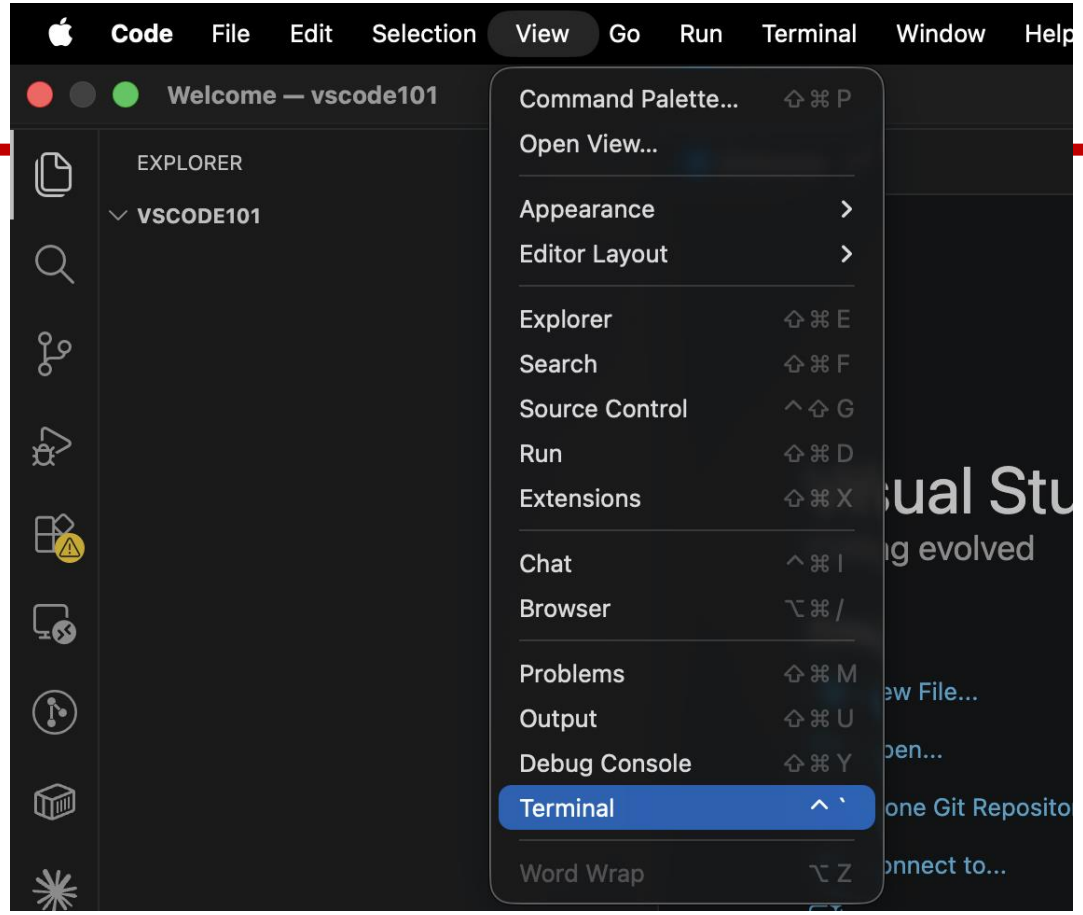
VSCODE – VIEW AND EDIT FILES

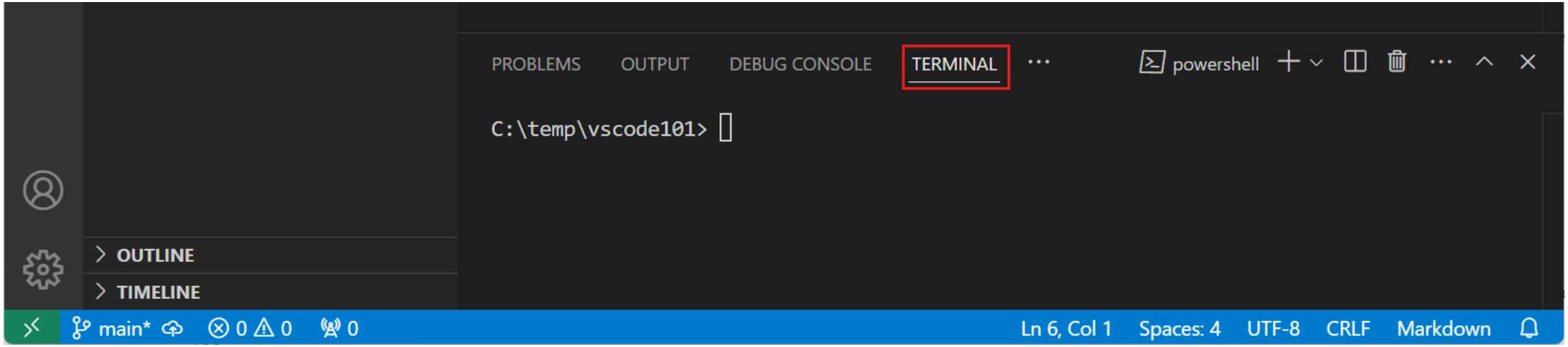
- Add more files to your workspace and notice that each file opens a new Editor tab.
- You can open as many editors as you like and view them side by side vertically or horizontally.
  - Learn more about side by side editing.



VSCODE – OPEN TERMINAL

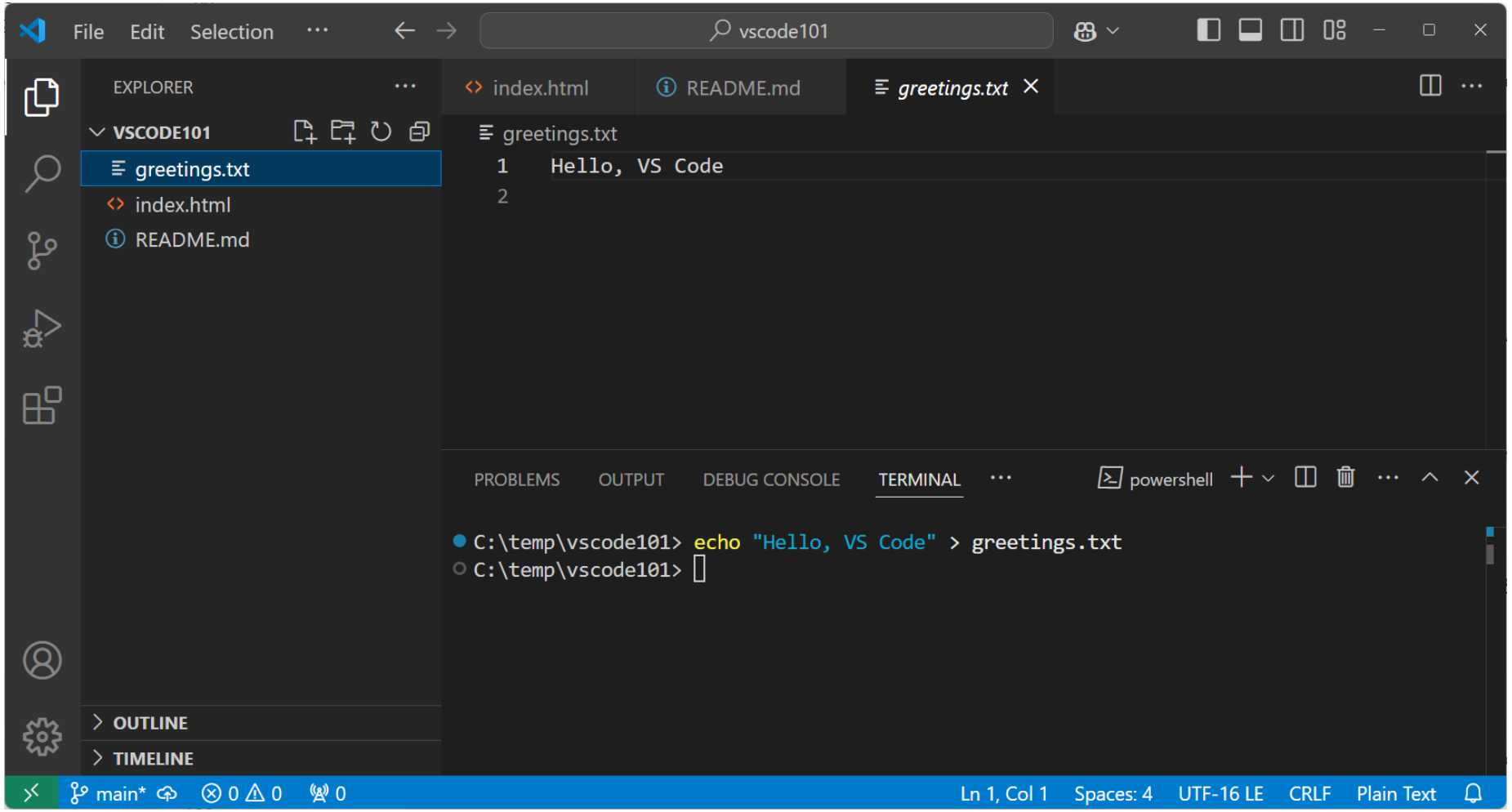
- VS Code has an integrated terminal.
- Open it by pressing ^`.
- You can also use the **View** > **Terminal** menu item.





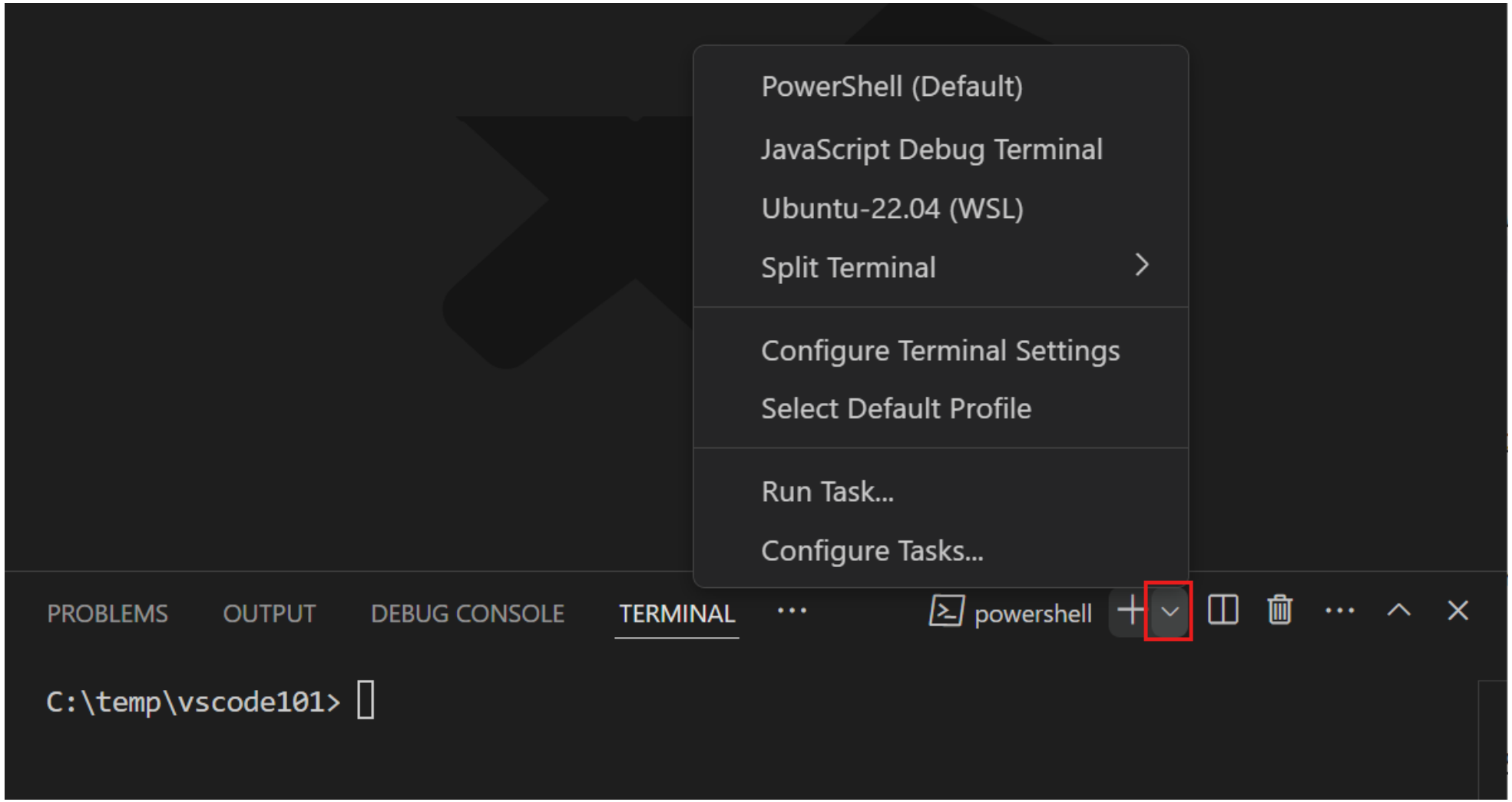
VSCODE – RUN A COMMAND IN TERMINAL

- In the terminal, enter the following command to create a new file in your workspace.
  - echo "Hello, VS Code" > greetings.txt
- The default working folder is the root of your workspace. *Notice that the Explorer view automatically picks up and shows the new file*



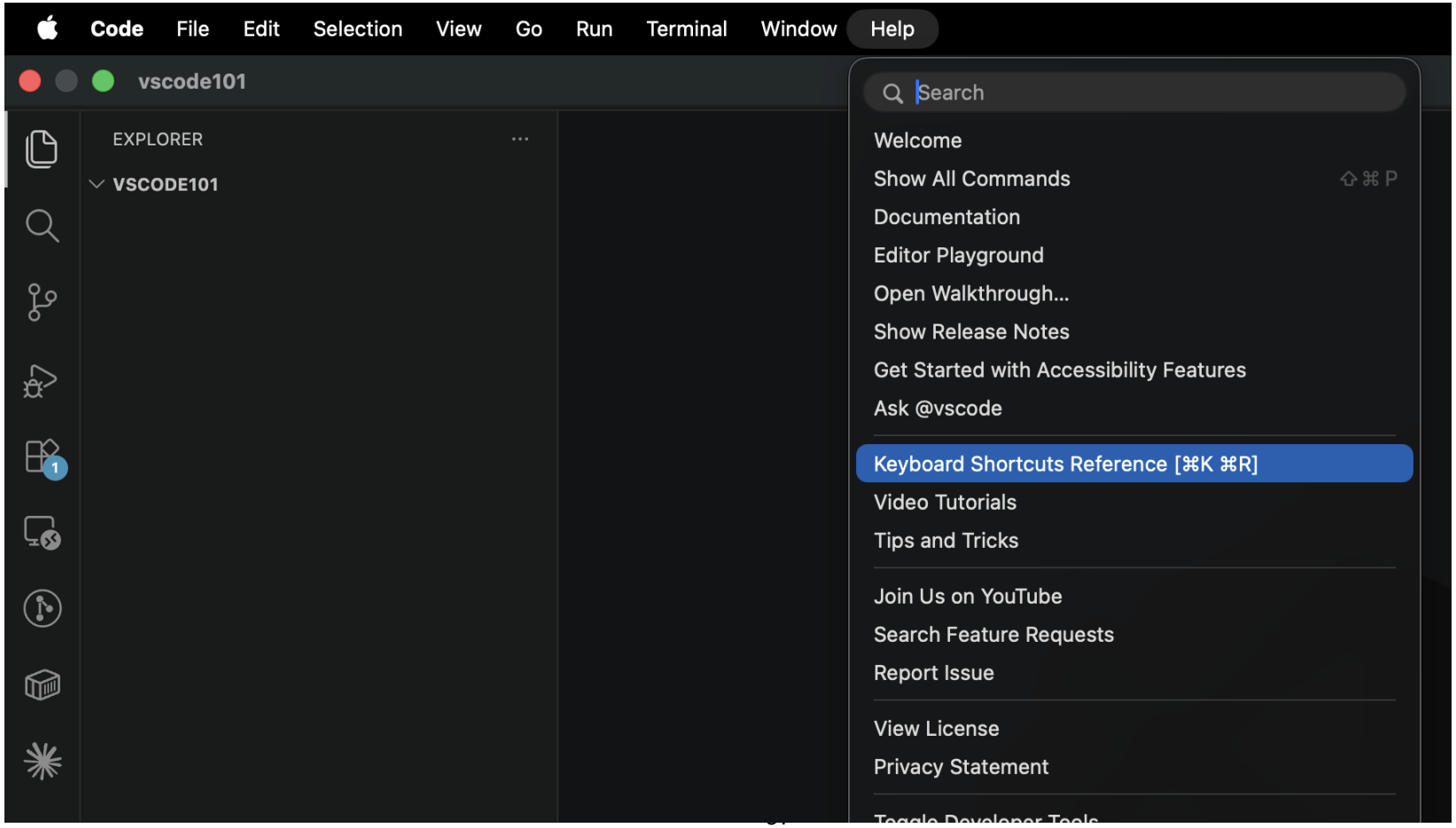
VSCODE – MULTIPLE TERMINALS

- You can open multiple terminals simultaneously. Select the Launch Profile dropdown to view the available shells and choose one.



VSCODE – KEYBOARD SHORTCUTS

- You can view all VS Code keyboard shortcuts for your operating system by going to the Help menu (top-left corner) and selecting “Keyboard Shortcuts Reference.”
  - This will open a webpage with a complete shortcut cheat sheet tailored to your OS.
- The next slide also highlight some of the most useful shortcuts to get you started.



VSCODE – KEYBOARD SHORTCUTS

Prefix: Cmd (Mac) / Ctrl (Windows/Linux)

Editing

Select text → <prefix> + D
- Select next matching word (multi-cursor editing)

Select text → <prefix> + Shift + L
- Select ALL matching words in file

<prefix> + / → Comment / uncomment a line or selection

<Alt + ↑ / ↓> (Mac: Option + ↑ / ↓) → Move line up or down

<Shift + Alt + ↑ / ↓> → Copy line up or down

File & Code Productivity

<prefix> + S → Save file

<prefix> + N → New file

<prefix> + W → Close tab/file

<prefix> + Z → Undo

<prefix> + Shift + Z → Redo

Search & Navigation

<prefix> + P → Quick Open (search files in project)

<prefix> + Shift + F → Search across entire project (workspace)

<prefix> + F → Search inside current file

UI / Panels

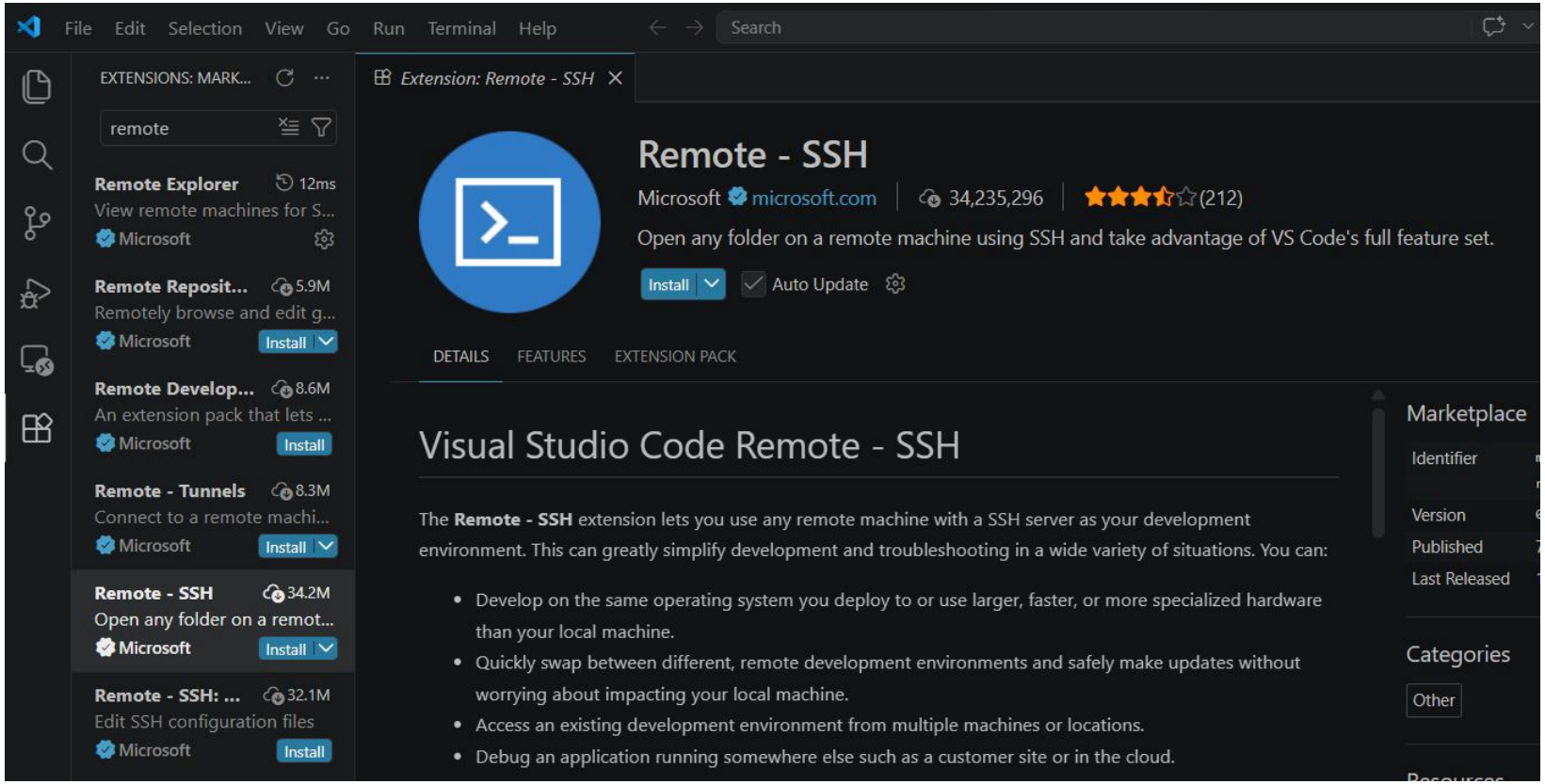
<prefix> + B → Toggle sidebar (file explorer)

<prefix> + J → Toggle bottom panel (terminal)

SECTION 2: REMOTE SSH TO IFARM

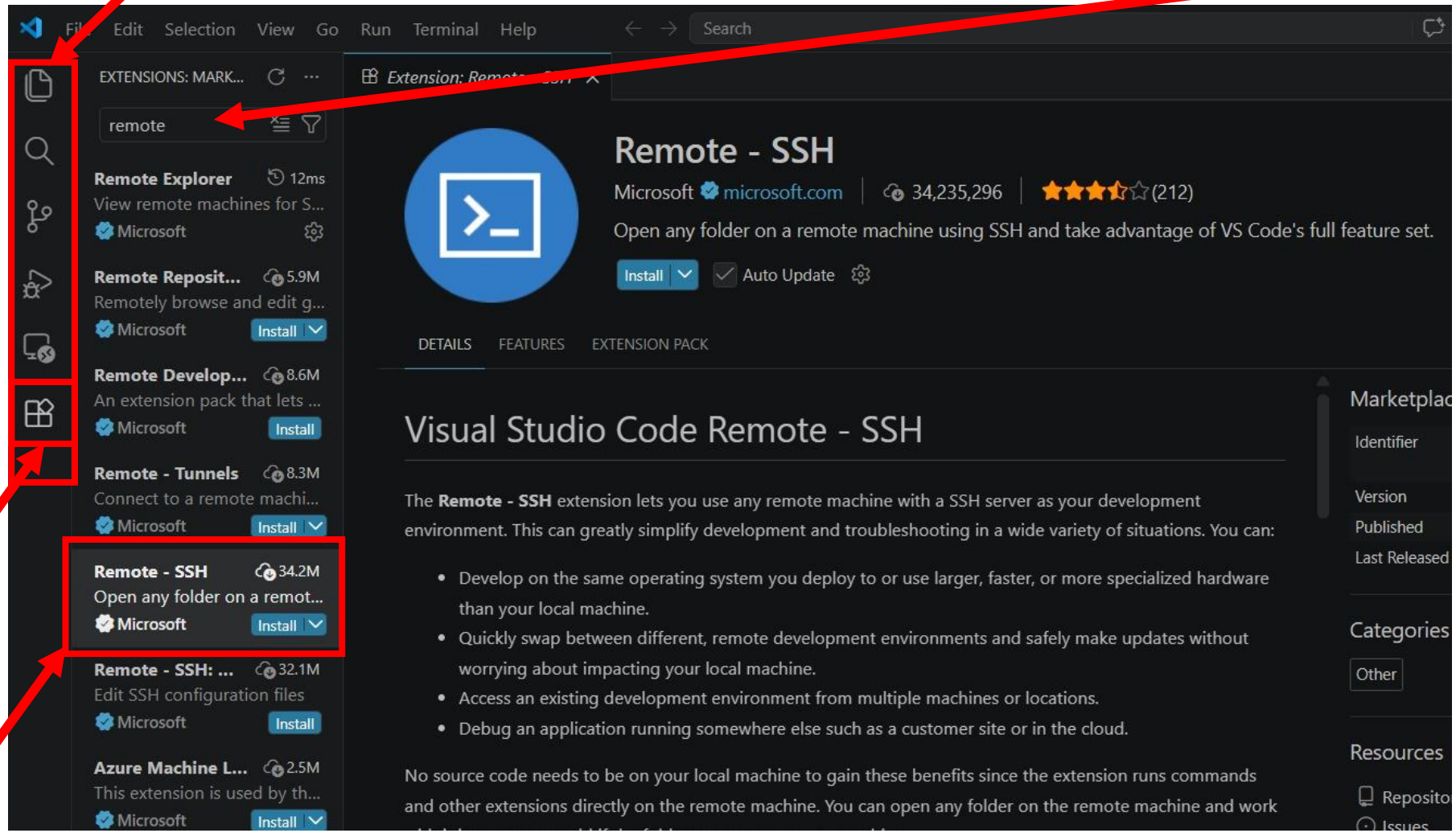
VSCODE – REMOTE SSH TO IFARM

- For SSH access to ifarm using VSCode, we just need these two:
  - VSCode - already downloaded & installed (previous slides).
  - Remote-SSH extensions - upcoming slides.



VSCODE - INSTALLING REMOTE SSH EXTENSIONS

- Go to Activity Bar and switch to extensions panel and install Remote-SSH extension (search remote)

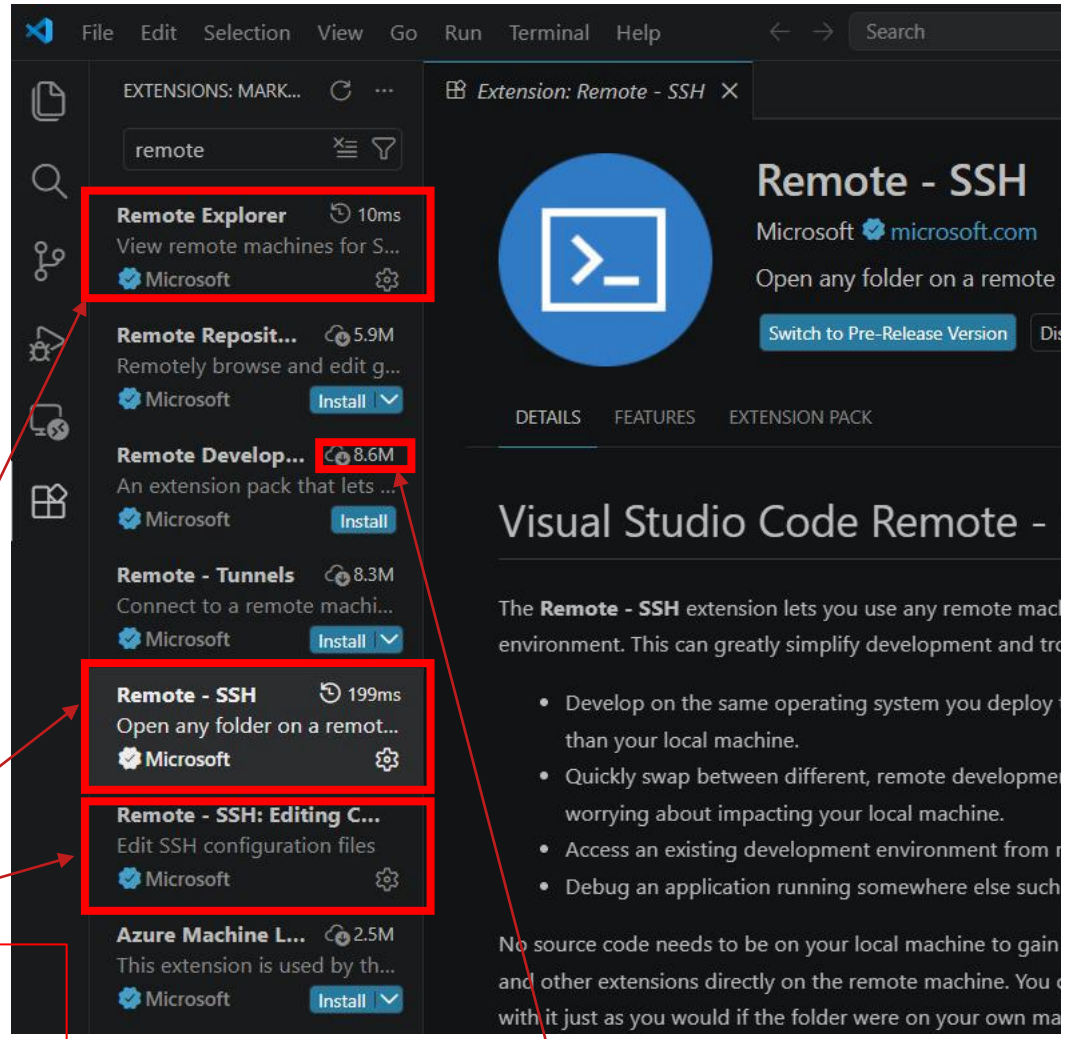


Extensions

Click here

VSCODE - INSTALLING REMOTE SSH EXTENSIONS

- Installing “Remote – SSH” extension also installs:
  - Remote Explorer
  - Remote - SSH: Editing Configuration Files
- Only install Remote – SSH (others come automatically)
  - Remote – SSH (Core)
    - Connect to remote machines via SSH
    - Work as if VS Code runs on the remote system
    - Open files, run terminal, debug
    - *Engine for remote development*
  - Remote Explorer (UI)
    - View saved hosts & environments
    - One-click connect
    - *Dashboard/sidebar*
  - Remote - SSH: Editing Configuration Files
    - Edit ~/.ssh/config in VS Code
    - Add hosts, users, ProxyJump
    - *Configuration tool*

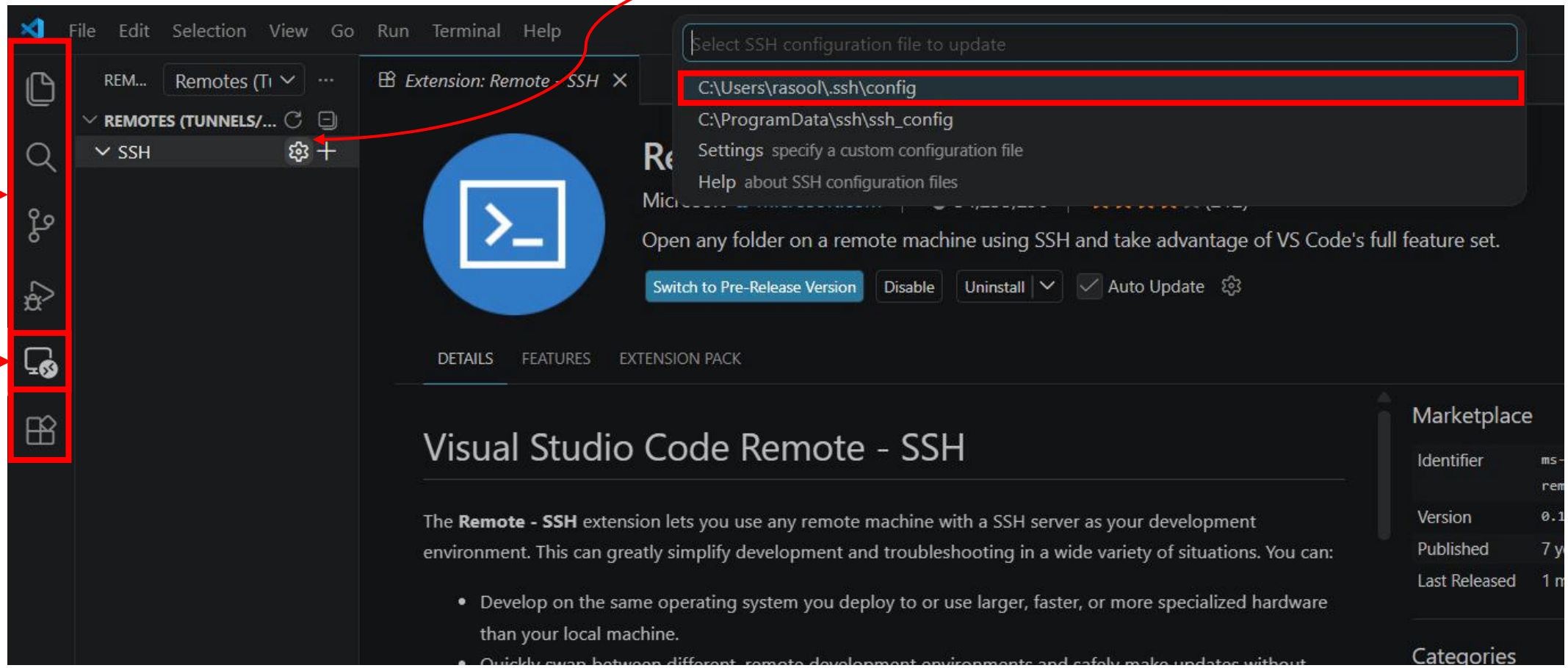


All three extensions show no download icon, showing they’re automatically installed with **Remote - SSH**.

Download icon

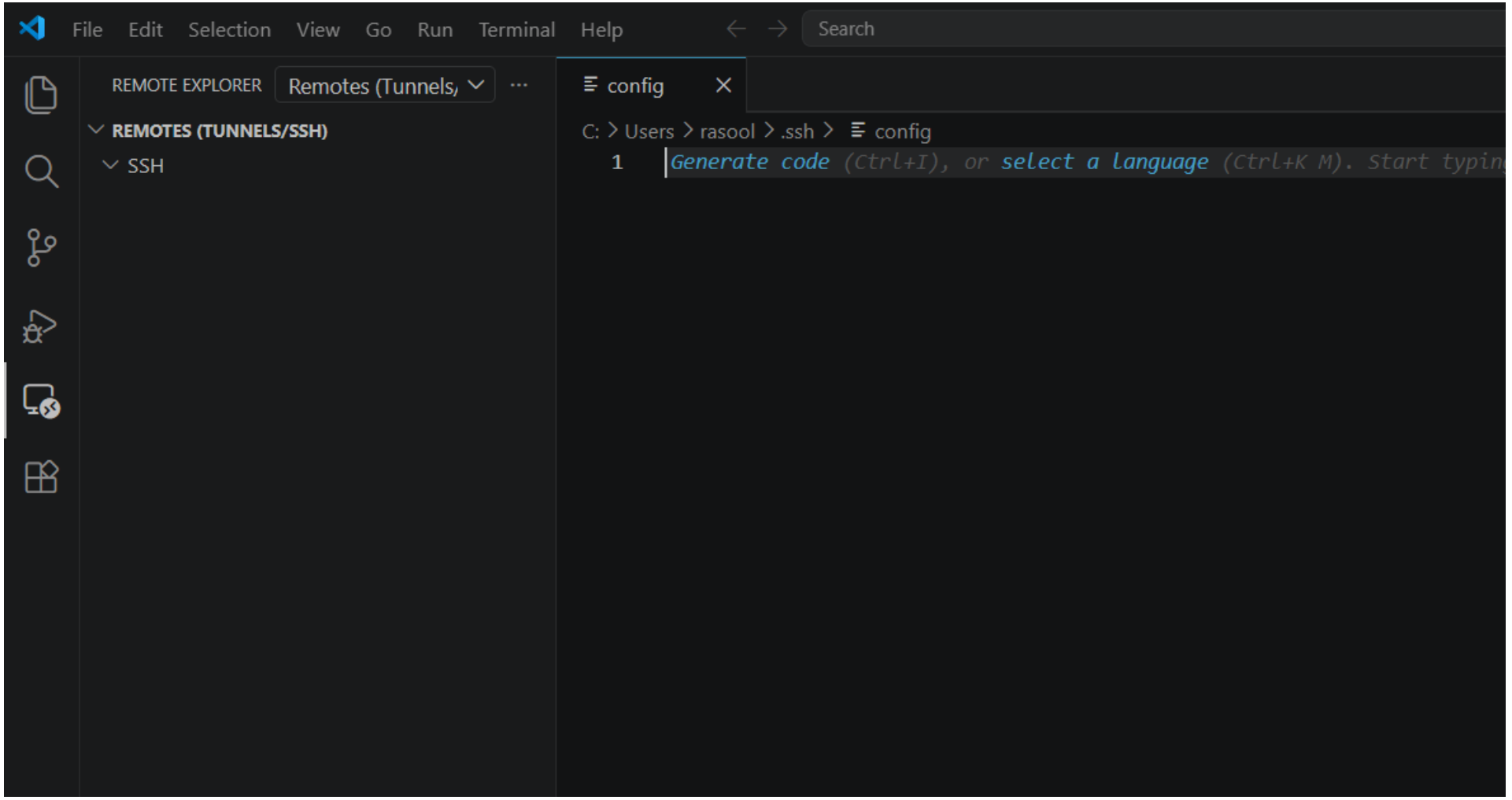
VSCODE – SETTING UP SSH CONFIG FILE

- Go to Remote Explorer panel using Activity Bar and click on settings icon. Now hit enter selecting ~/.ssh/config file



VSCODE – SETTING UP SSH CONFIG FILE

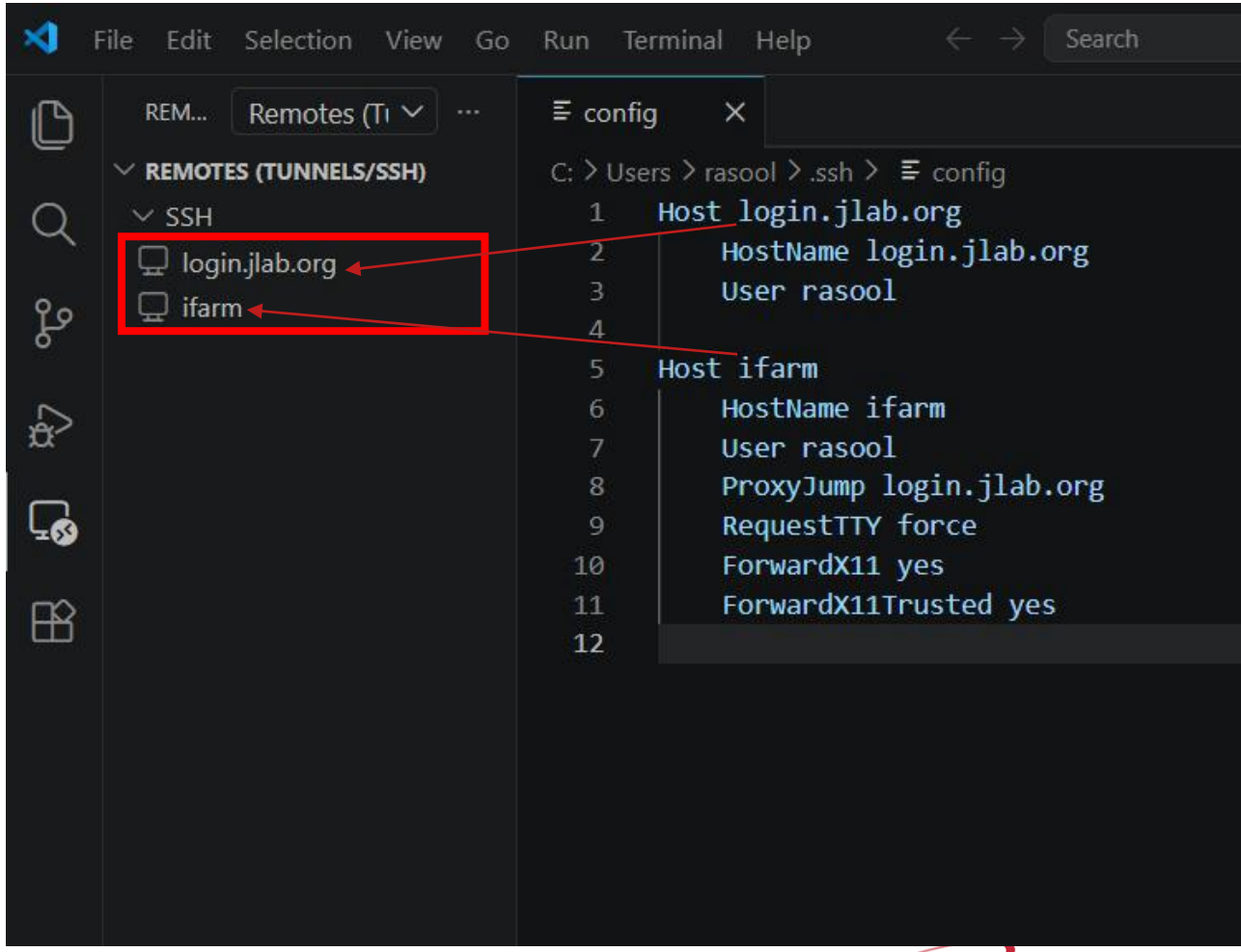
- This will open an empty ~/.ssh/config file to edit:



VSCODE – SETTING UP SSH CONFIG FILE

- Paste the exact content into the config file, then press **Ctrl/Cmd + S** to save. Make sure the indentation (tabs and spacing) is correct.

```
Host login.jlab.org
    HostName login.jlab.org
    User <your_jlab_username>

Host ifarm
    HostName ifarm
    User <your_jlab_username>
    ProxyJump login.jlab.org
    RequestTTY force
    ForwardX11 yes
    ForwardX11Trusted yes
```



- You will start seeing host names in the Remote Explorer Panel

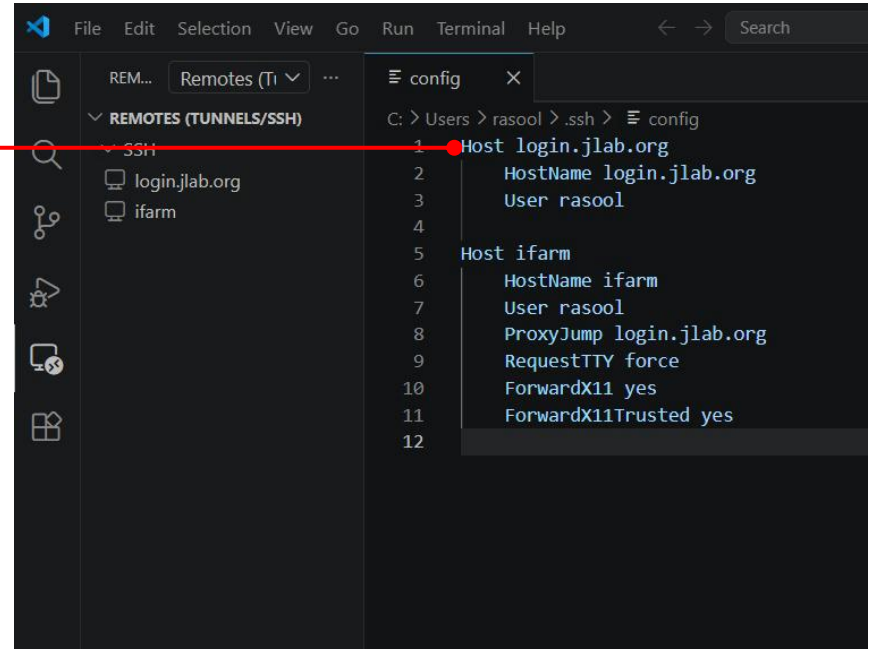
Note: Replace <your_jlab_username> with your actual JLab username in the file.

VSCODE – SSH CONFIG FILE EXPLANATION

- **login.jlab.org**
  - Host login.jlab.org
    - Defines a connection named login.jlab.org
  - HostName: actual server address
  - User: username used to log in (rasool)

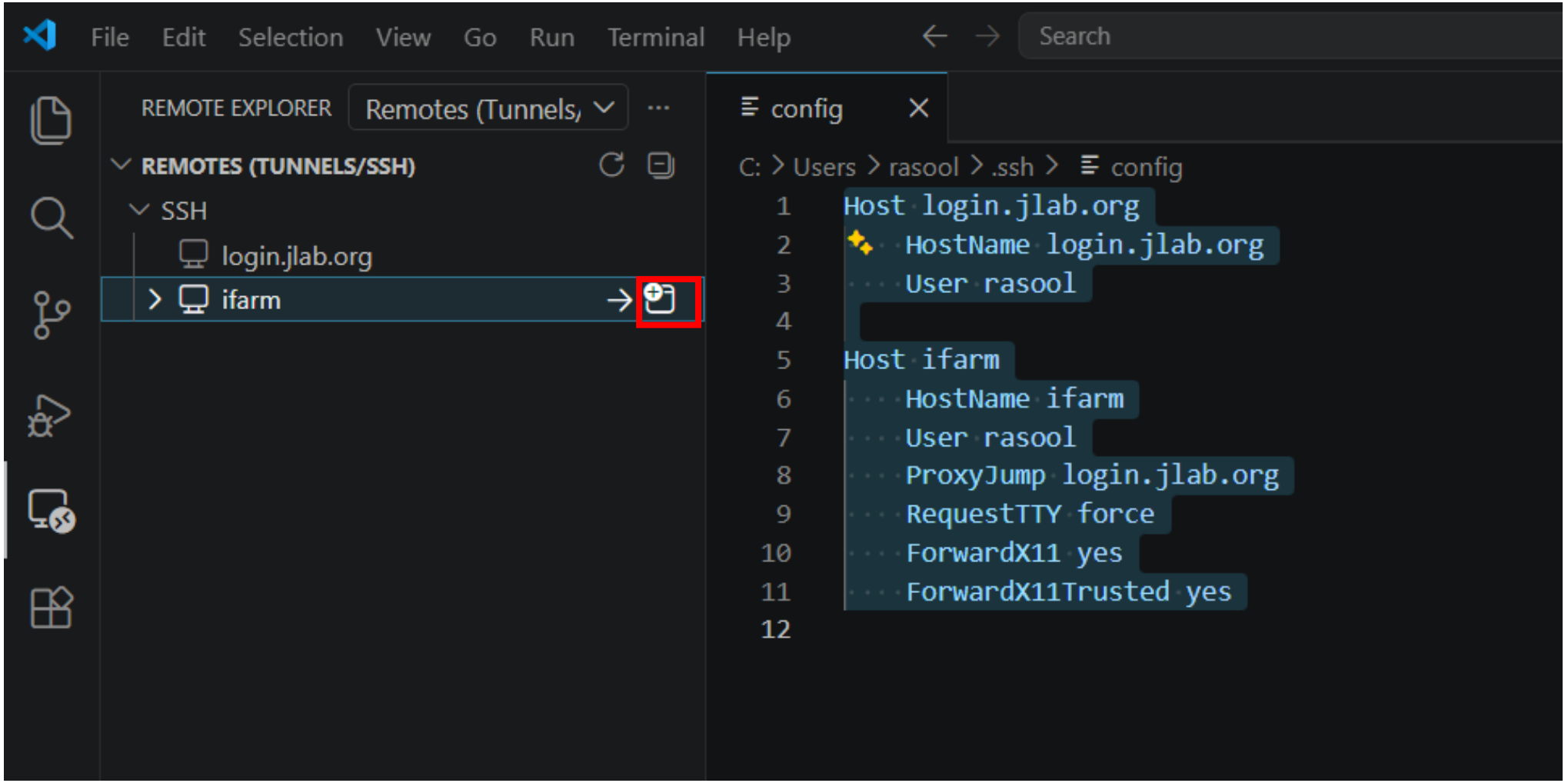
This defines the display name shown in the Remote Explorer panel for this connection—you can name it anything you prefer.

- **ifarm**
  - Host ifarm
    - Defines a connection named ifarm
  - HostName: actual server address
  - User: username used to log in (rasool)
  - ProxyJump login.jlab.org:
    - connects via login server first (jump host)
  - RequestTTY force: forces interactive terminal session
  - ForwardX11 yes / ForwardX11Trusted yes
    → *Required for X11 forwarding (later slides)*



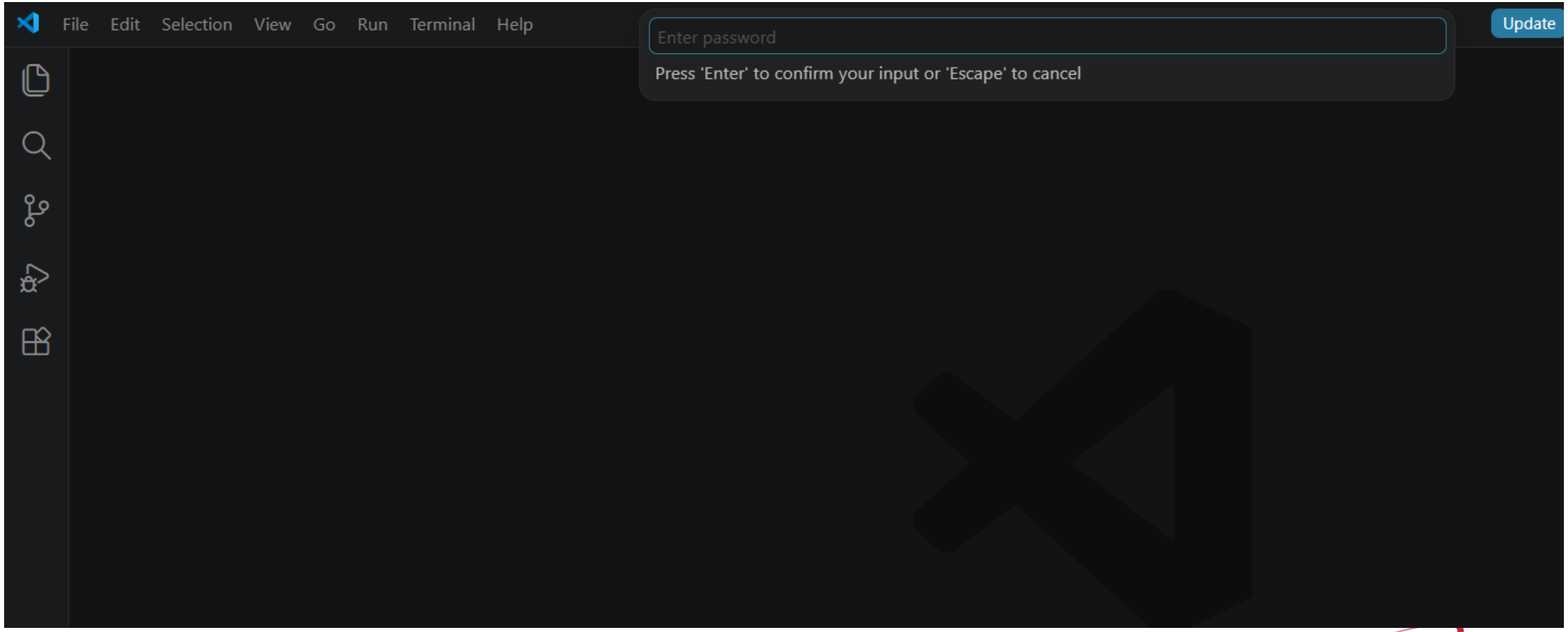
VSCODE - CONNECT TO IFARM

- Go to Remote Explorer and click on open in new window icon:



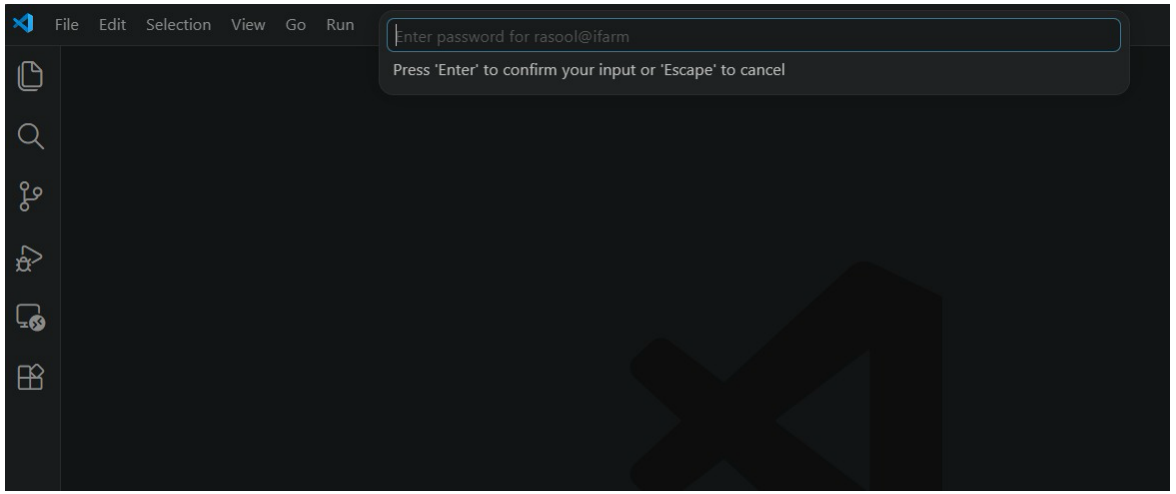
VSCODE - CONNECT TO IFARM

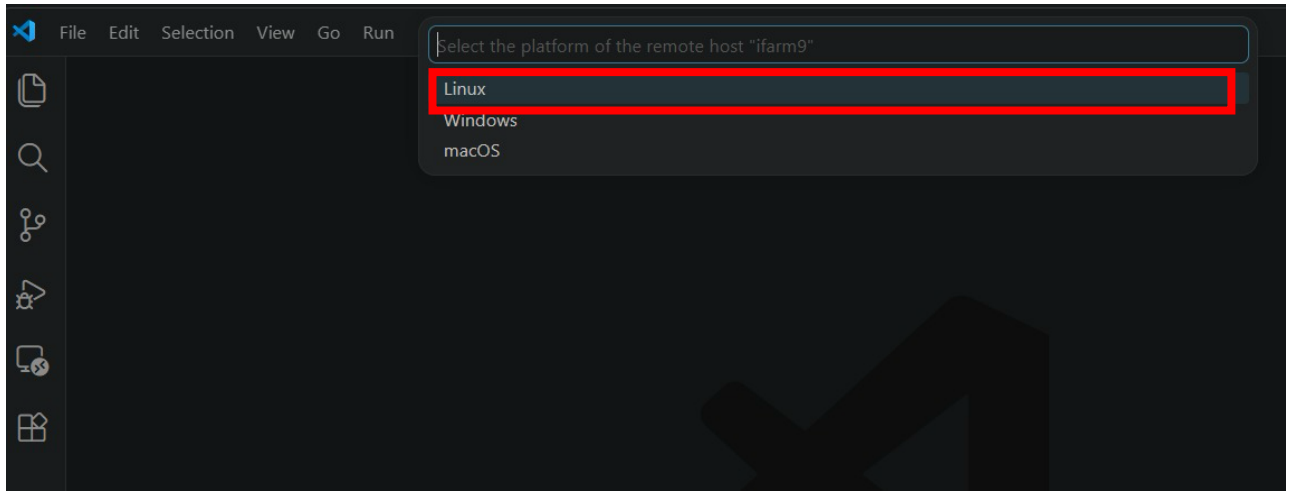
- A new window will open with a password prompt.
  - Enter your **6-digit PIN + MobilePass/USB token**.
  - You are logging into login.jlab.org, which acts as the gateway to access ifarm.
    - The ProxyJump login.jlab.org setting in the config automatically handles this intermediate step.

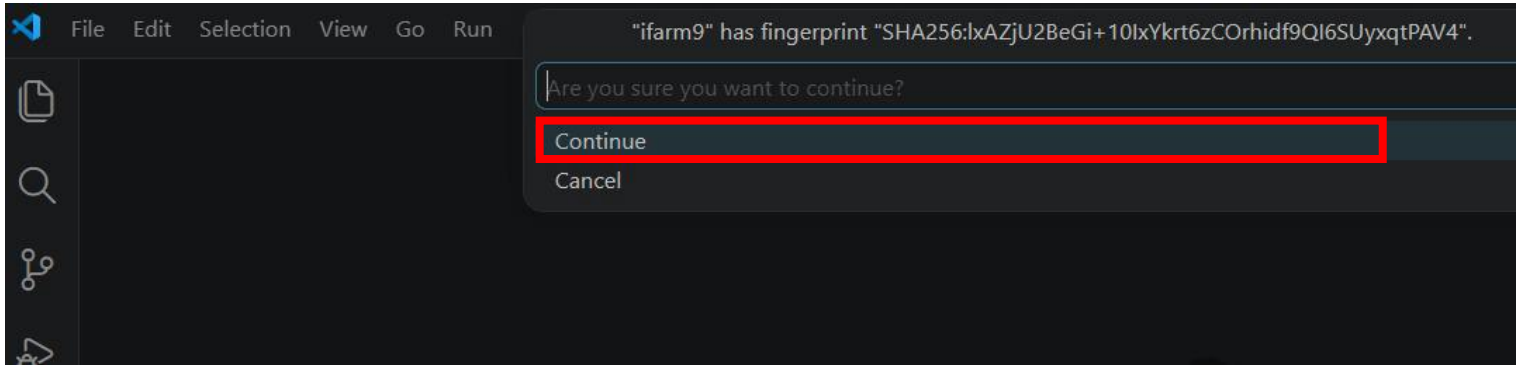


VSCODE - CONNECT TO IFARM

- After entering your password for login.jlab.org, a second prompt will appear for @ifarm. Enter your CUE (JLab) account password to complete the login.
- For any additional pop-ups, simply select the default trusted option such as “Linux” or “Continue”, depending on what is shown.

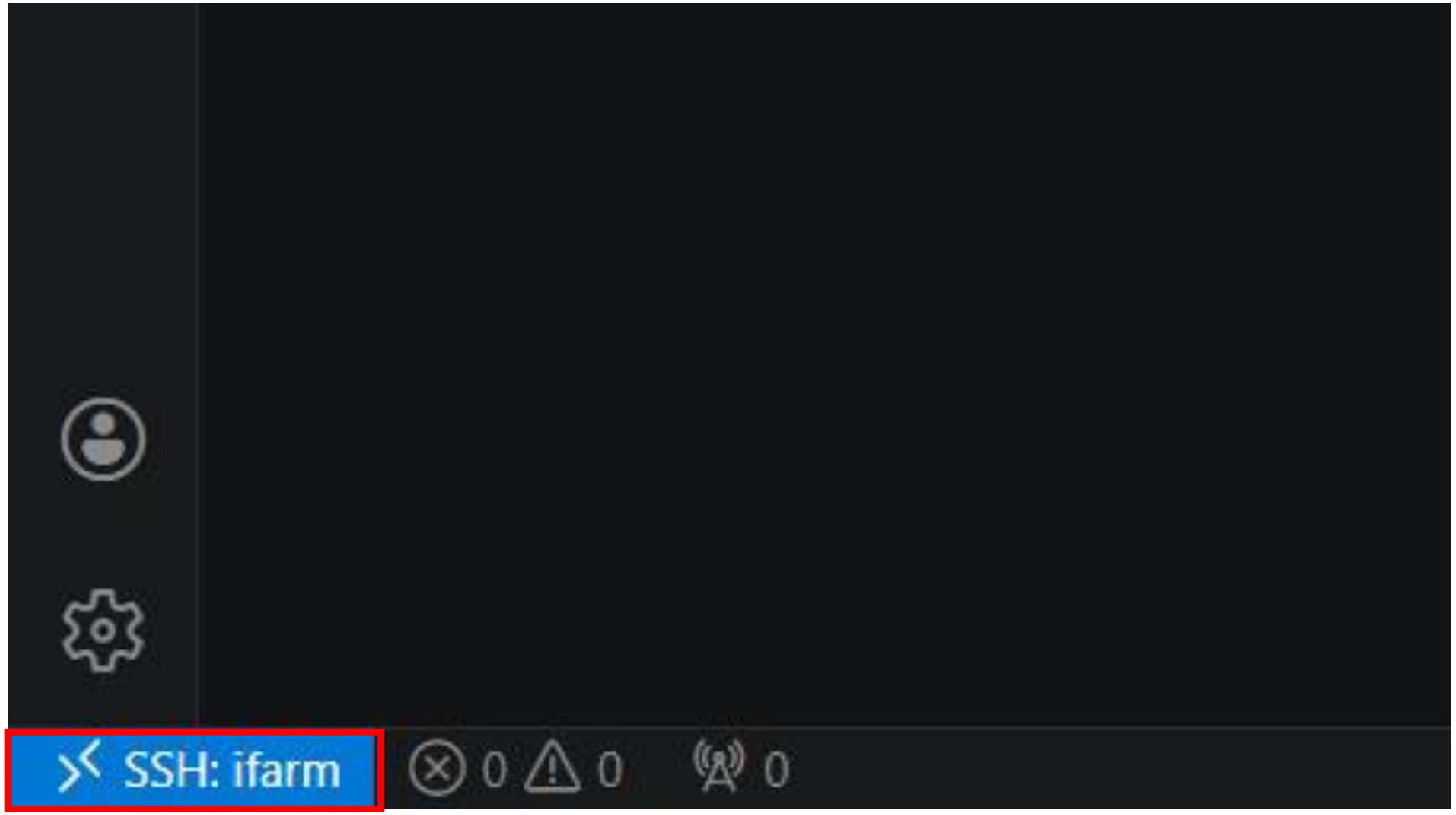






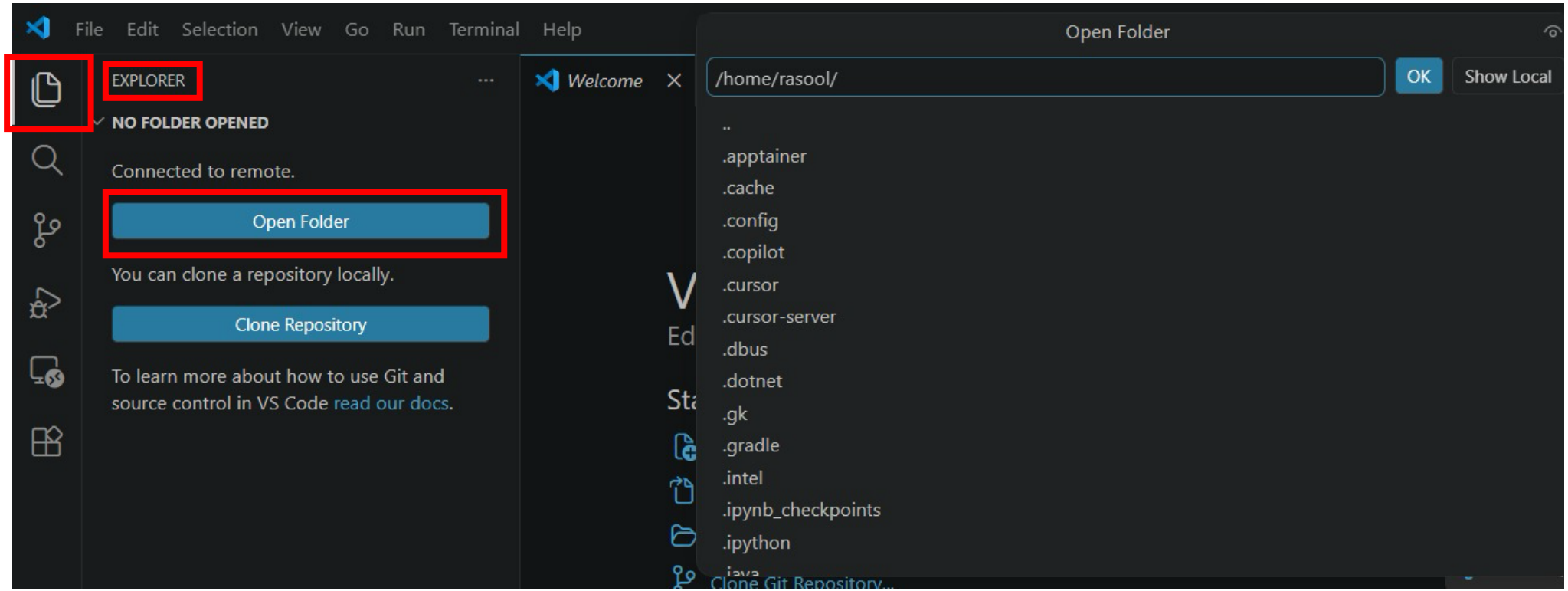
VSCODE - CONNECT TO IFARM

- After completing all the prompts, you will be successfully connected to ifarm. You should now see “SSH: ifarm” displayed in the bottom-left corner of VS Code, confirming the connection is active.



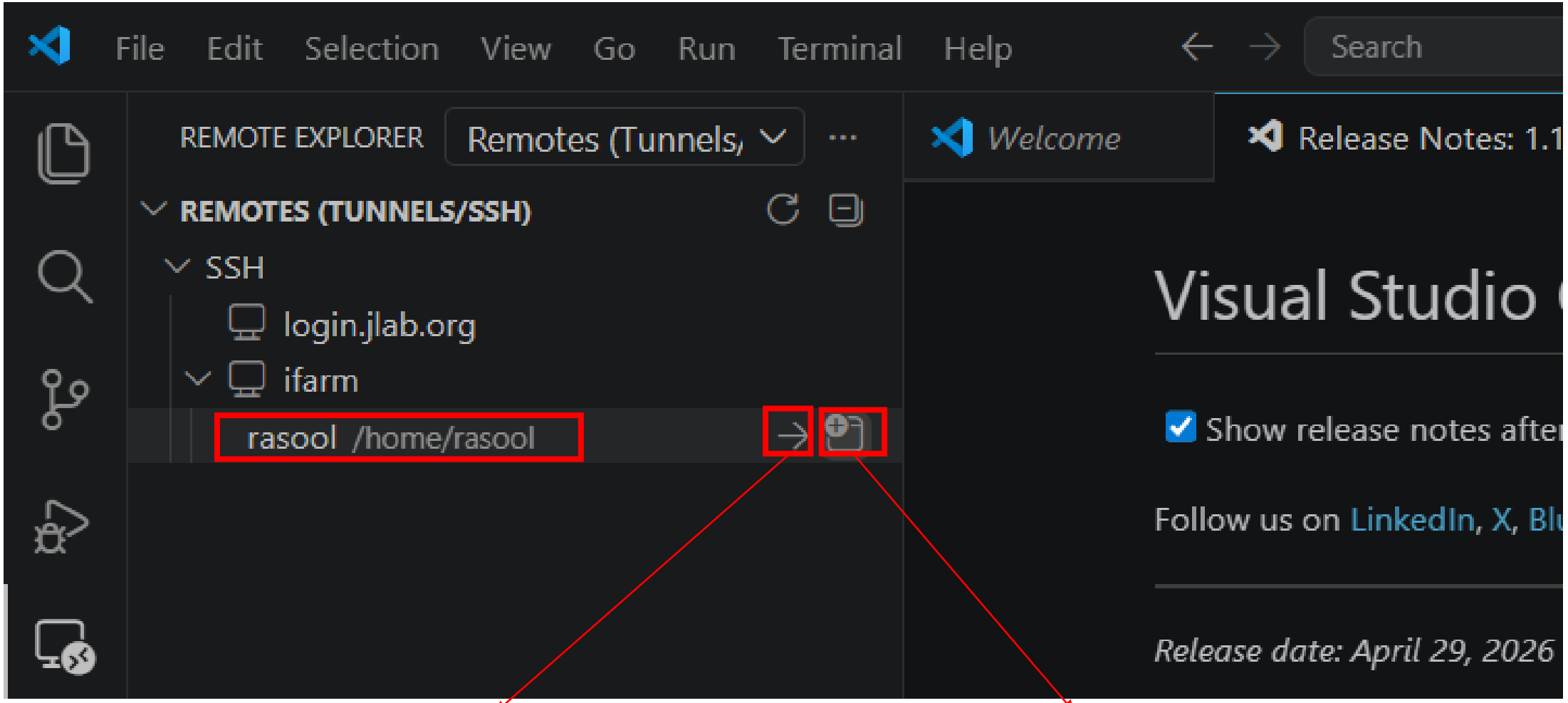
VSCODE – OPEN A FOLDER ON IFARM

- To open a folder on ifarm, go to the Explorer panel in the Activity Bar and click “Open Folder”.
  - A prompt will appear at the top where you can enter a directory path to navigate to.
  - For now, simply hit Enter to use the default path, which is your user directory (/home/<username>).



VSCODE – OPEN A FOLDER ON IFARM

- After connecting to a folder once on a host that path get saved and next time you can directly connect to host on that path by just clicking on that path

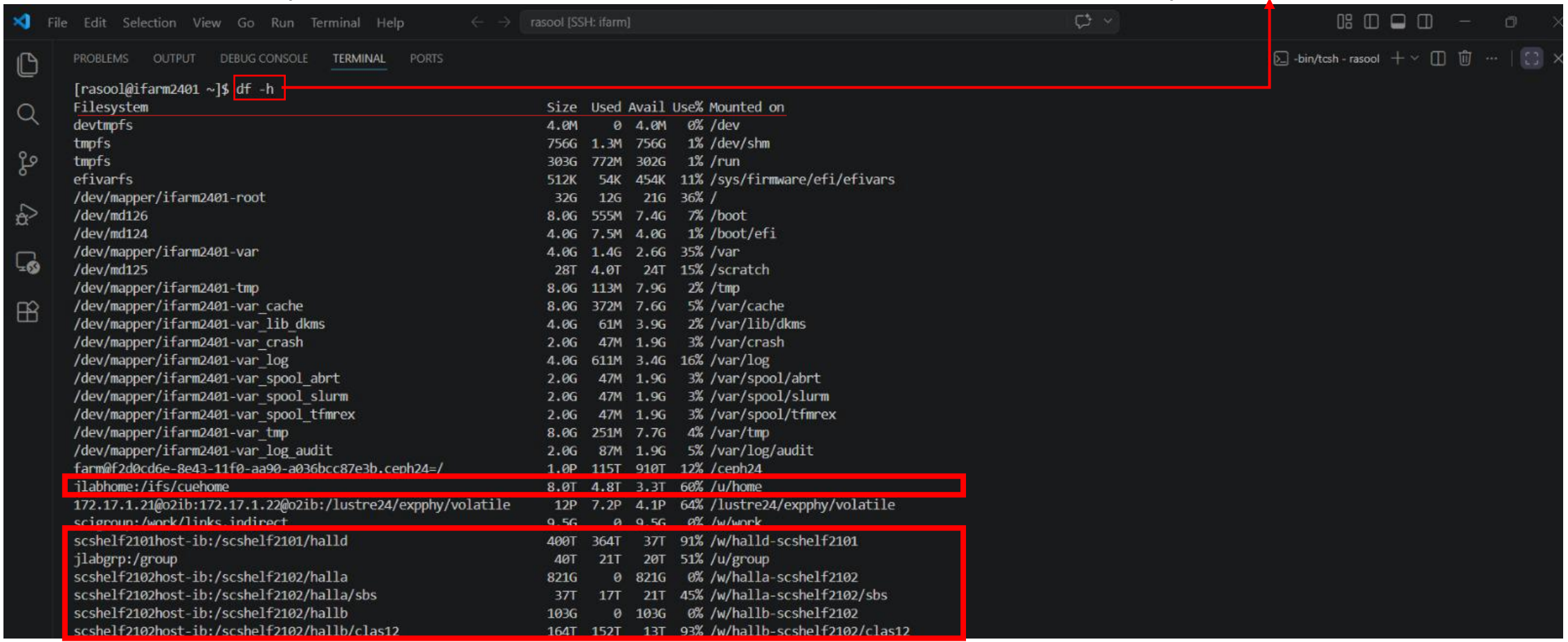


SECTION 3: FILESYSTEMS DISK CAPACITY

IFARM – CHECKING FILESYSTEMS DISK CAPACITY

- In Slide 11 ("JLAB Filesystems"), we discussed that /home and /work have hard storage limits. If the total capacity is reached, you will encounter a "disk quota exceeded" error.
- Now, let's look at some useful commands to check disk usage:
  - df -h → Displays disk usage for all filesystems on the system
    - Focus mainly on /w (/work) and /home, as these are most relevant for you

-h (human-readable) displays storage in KB, MB, GB, etc., instead of raw bytes



IFARM – CHECKING FILESYSTEMS DISK CAPACITY

- df -h . → Shows filesystem details for the current directory
- df -h <path_to_directory> → Shows filesystem details for the given directory
- If you encounter a quota error, navigate to the affected directory and run:
  - df -h . or df -h <path_to_directory>

```
[rasool@ifarm2401 ~]$ df -h .
Filesystem              Size  Used Avail Use% Mounted on
jlabhome:/ifs/cuehome   8.0T  4.8T  3.3T  60% /u/home
[rasool@ifarm2401 ~]$ df -h /work/epsci/rasool/
Filesystem                                Size  Used Avail Use% Mounted on
scshelf2103host-ib:/scshelf2103/epsci     2.0T  1.6T  426G  80% /w/epsci-scshelf2103
[rasool@ifarm2401 ~]$
```

Note: df shows overall disk usage and capacity for entire filesystem(s) (across all users)

IFARM – CHECKING FILESYSTEMS DISK CAPACITY

- Since you cannot delete other users' files, the quickest way to resolve a disk quota issue is to check your own disk usage and remove unnecessary files.
- To see how much space your files are using within a directory, run:
  - du -h --max-depth=1 → size of current directory + immediate subfolders
    - -h → human-readable sizes (KB, MB, GB)
  - du -ah –max-depth=1 → includes individual files as well
    - -a → includes all files (not just directories)

```
[rasool@ifarm2401 gluex_hydra]$ du -ah --max-depth=1
251M    ./backend
24K     ./robots.txt
24K     ./.htaccess-old
24K     ./index.html
24K     ./manifest.json
24K     ./.htaccess
1.2G    ./static
512K    ./favicon_io
24K     ./asset-manifest.json
1.4G    .
[rasool@ifarm2401 gluex_hydra]$
```

Note: --max-depth=1 limits the output to one level of subdirectories, avoiding deeply nested listings

IFARM – FILESYSTEMS DISK CAPACITY & VSCODE

- As mentioned in Slide 17 ("VS Code Remote SSH"), VS Code installs a VS Code Server on the remote machine before connecting.
  - By default, it is downloaded to your home directory: /u/home/<username>
- If the /home filesystem is full → the server cannot download → connection fails

- **How to Avoid**
  - Store any large or space-taking files in the /w(/work) directory instead of /home to prevent the home filesystem from filling up
  - If you frequently face this issue, you can redirect VS Code to install the server in /work (/w) instead of the default /home directory:
    - https://stackoverflow.com/questions/62613523/how-to-change-vscode-server-directory
    - In most cases, using your allocated /work directory for project work significantly reduces the chances of running into storage quota issues

IFARM – FILESYSTEMS DISK CAPACITY & VSCODE

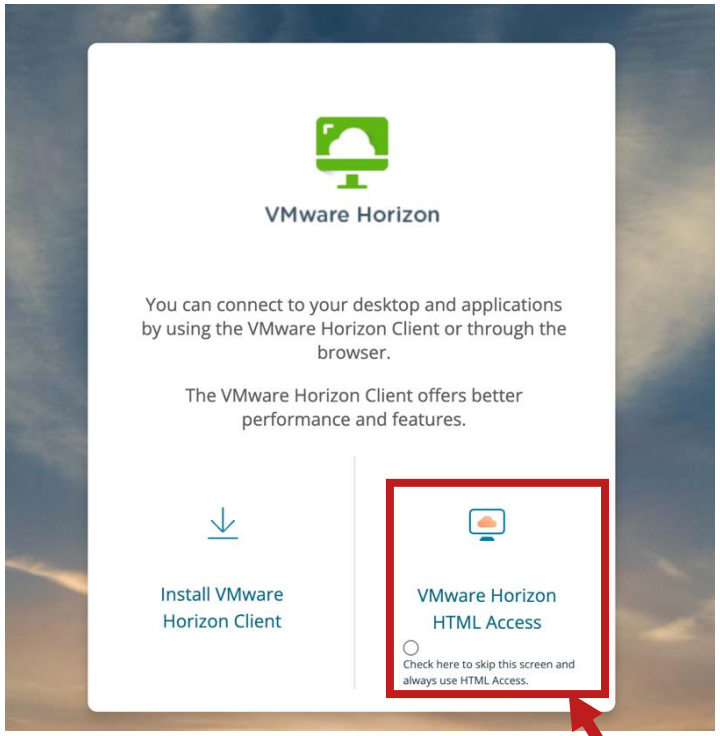
- As mentioned in Slide 17 ("VS Code Remote SSH"), VS Code installs a VS Code Server on the remote machine before connecting.
  - By default, it is downloaded to your home directory: /u/home/<username>
- If the /home filesystem is full → the server cannot download → connection fails

- **How to Fix**
  - Free up space in /home
    - Login to VDI (since VS Code won't connect and VDI has access to the same filesystem)
      - Open terminal in your home directory and check capacity & usage:
        - df -h .
        - du -ah --max-depth=1
      - Delete unnecessary files:
        - rm -rf ./.cache
        - rm -rf ./.vscode-server

IFARM – FILESYSTEMS DISK CAPACITY & VSCODE

- Login to VDI

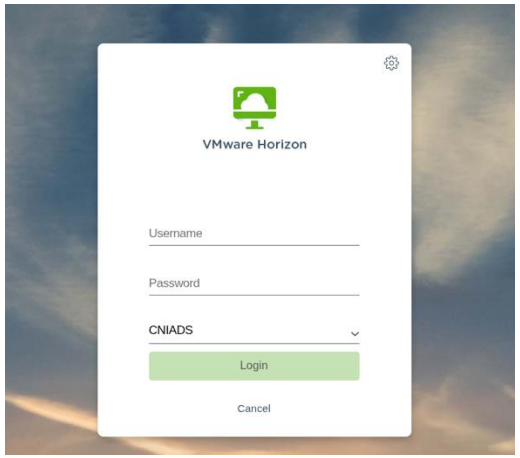
1.



https://vdi.jlab.org

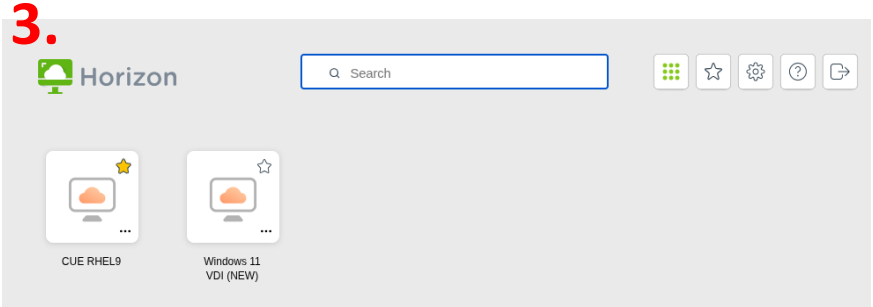
Web access

2.



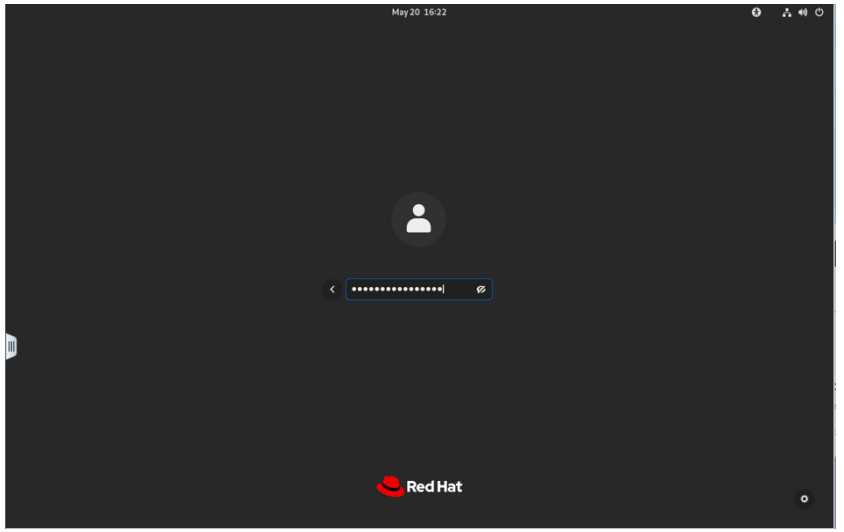
Log in with CUE Username and Password

3.



Select RHEL9 machine (Windows requires SmartCard!!)

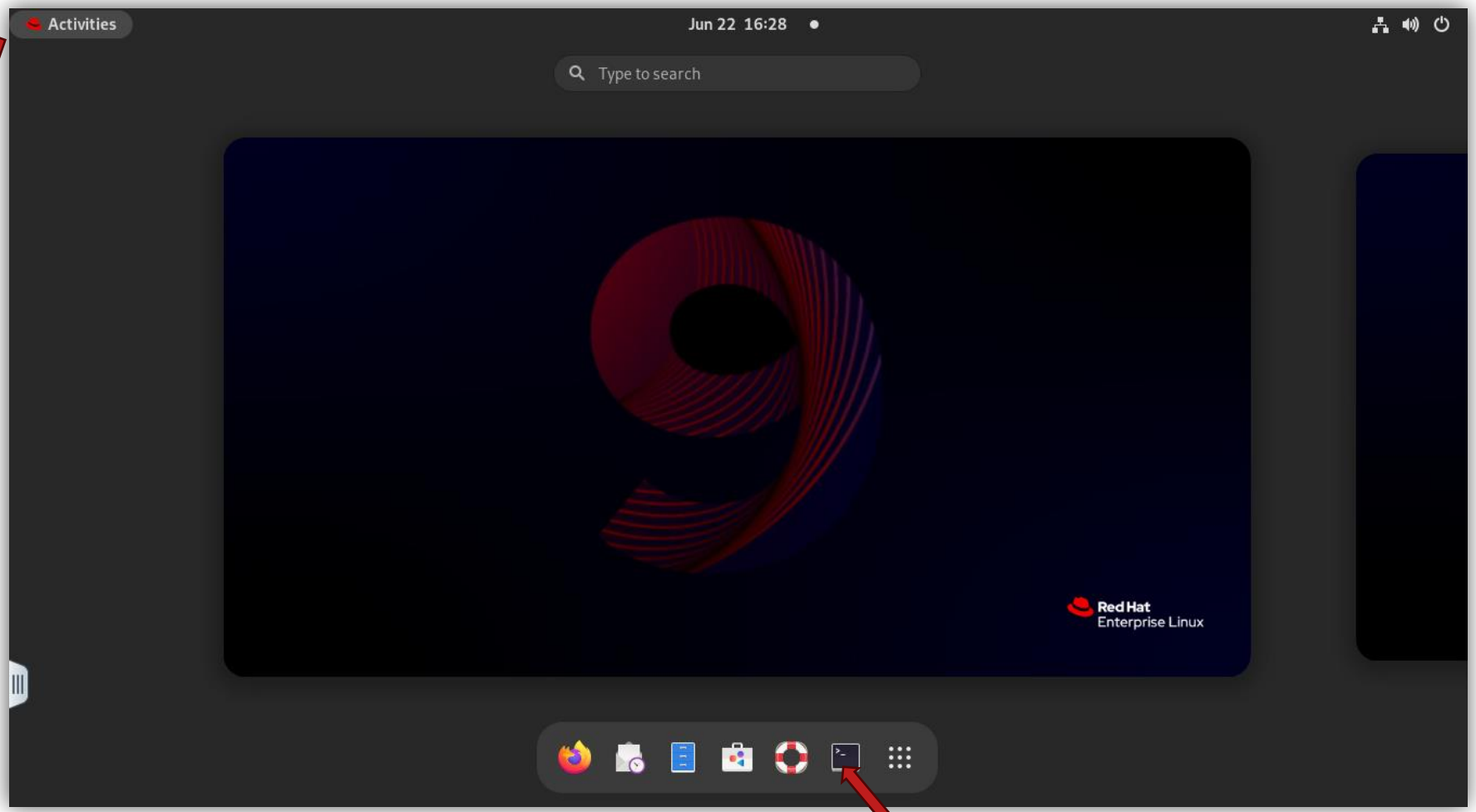
4. Enter CUE Username
(Password = **MobilePASS** SAS MFA PIN+Code)

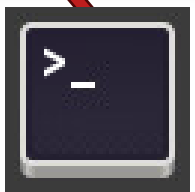


IFARM – FILESYSTEMS DISK CAPACITY & VSCODE

- Open terminal:

1. Click activities button

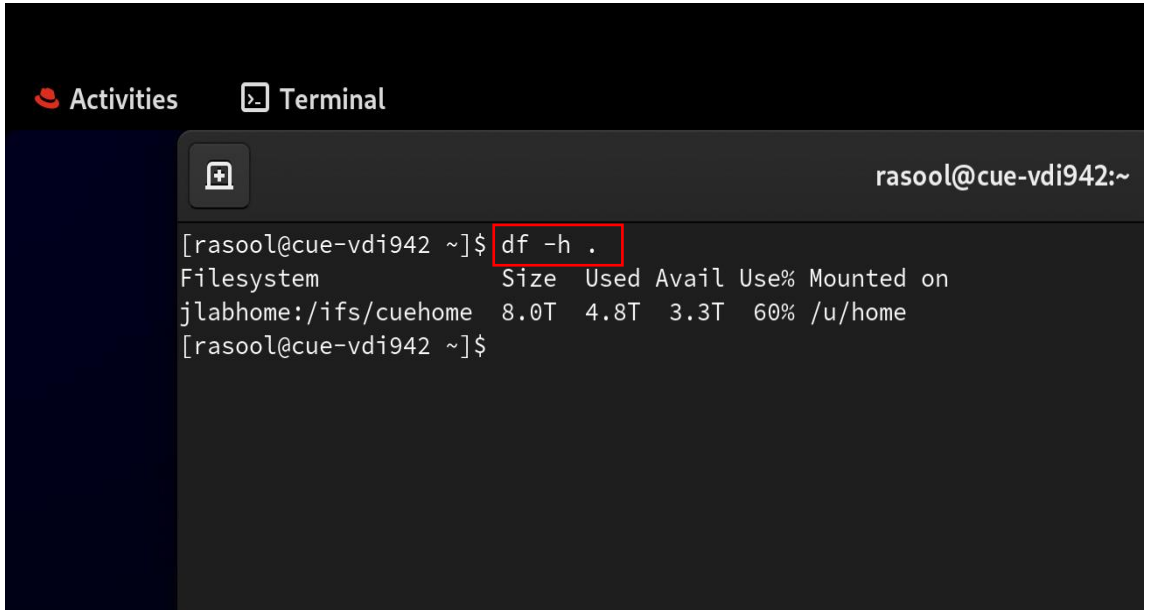


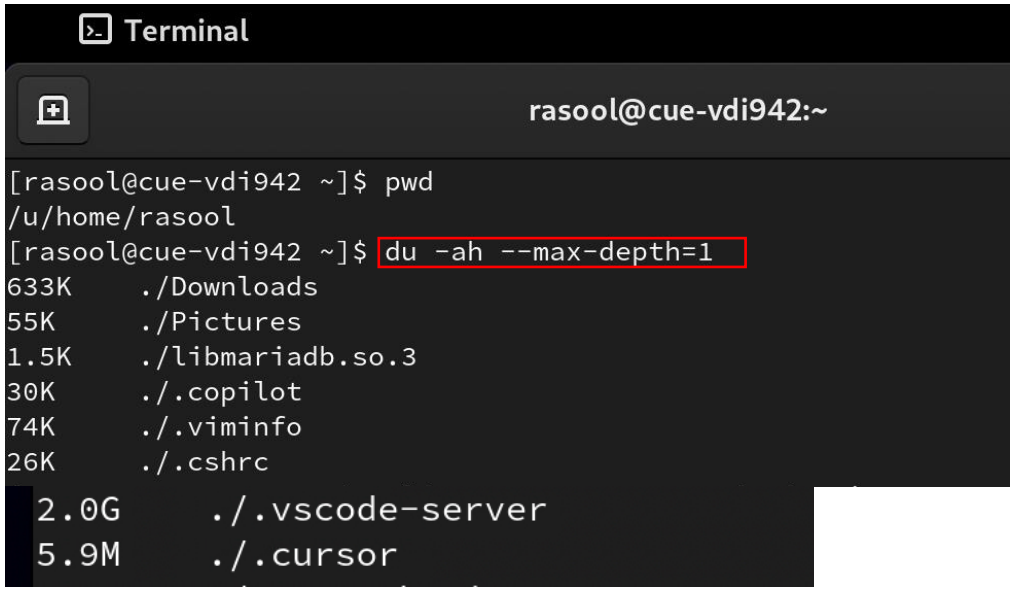


2. Click Terminal Icon

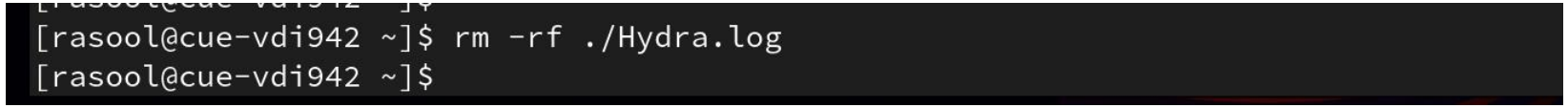
IFARM – FILESYSTEMS DISK CAPACITY & VSCODE

- Check usage:





- Now delete as required using the remove command:

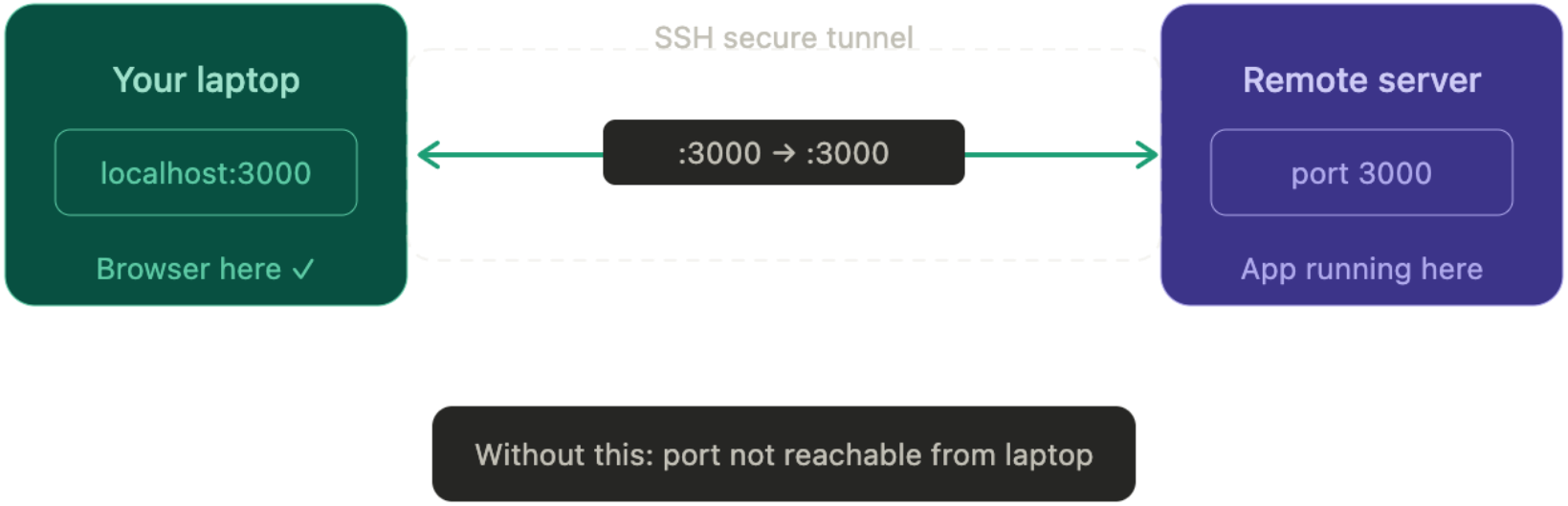


- Once you see sufficient free space using df -h ., you can retry connecting through VS Code and it should work.
  - In most cases, deleting ./.vscode-server and ./.cache is enough to resolve the issue.

SECTION 4: PORT FORWARDING

PORT FORWARDING

- Port Forwarding
  - What is this?
    - Remote server runs an app on a specific port (e.g., 3000)
    - This port is not directly accessible from your laptop
    - Port forwarding creates a secure tunnel
    - localhost:3000 → connects to remote port 3000
  - *Makes the remote app behave like it's running locally*
  - *Without it, you can't access remote services in your browser*



PORT FORWARDING

- *Port Forwarding in Visual Studio Code*
  - Automatically detects running ports (e.g., 3000, 8000)
  - Forwards them with no manual setup
  - Open via:
    - Browser by entering localhost URL
    - Or Ctrl/Cmd + click the link on terminal

- *Without VS Code*
  - Manual setup required for each port:
    - ssh -L 3000:localhost:3000 -J <username>@login.jlab.org <username>@ifarm.jlab.org
      - -L forwards a local port to a remote service – Port Forwarding
      - -J uses an intermediate jump host (ProxyJump) to reach the target server.
  - Need to repeat for every new port

RUNNING NETWORK BASED APPS ON IFARM

⚠ Port Already in Use Error

- When running network-based applications on ifarm that require a fixed port, you may encounter a "Port Already in Use" error
  - It's because ifarm is a shared system and ports may already be used by other users
  - If a port is occupied, your app cannot start because it cannot bind to the required port
  - *Solution:* You must switch to an available (unused) port in that case.
    - Check used ports:
      - netstat -tulp → lists all active ports
      - netstat -tulp | grep 6062 → check a specific port

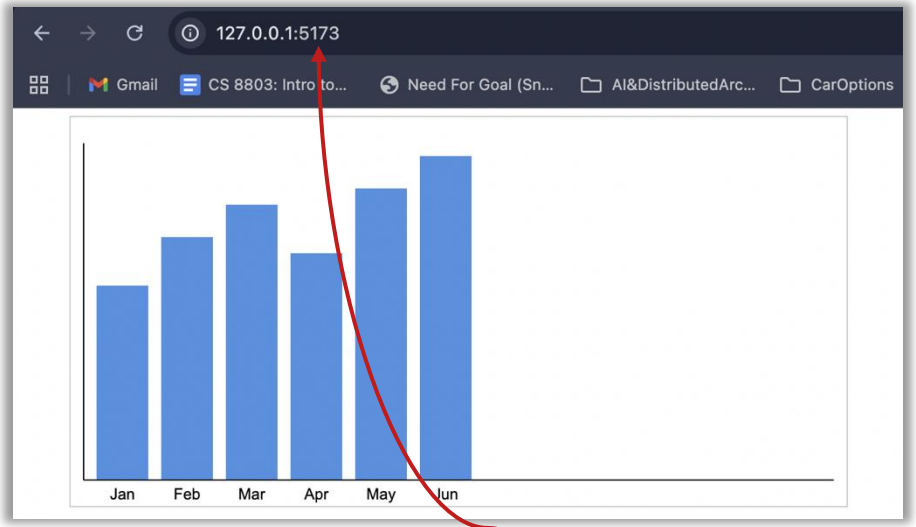
```
[rasool@ifarm2402 ~]$ netstat -tulp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 localhost:8962          0.0.0.0:*               LISTEN      -
tcp        0      0 localhost:dynamid       0.0.0.0:*               LISTEN      -
tcp        0      0 localhost:9003          0.0.0.0:*               LISTEN      -
tcp        0      0 localhost:cslistener    0.0.0.0:*               LISTEN      -
tcp        0      0 localhost:etlservicemgr 0.0.0.0:*               LISTEN      -
tcp        0      0 localhost:9004          0.0.0.0:*               LISTEN      -
```

```
[rasool@ifarm2402 ~]$ netstat -tulp | grep 6062
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
tcp        0      0 localhost:6062          0.0.0.0:*               LISTEN      -
tcp6       0      0 localhost:6062          [::]:*                  LISTEN      -
```

- In upcoming example, the setup automatically selects an available port and Visual Studio Code forwards it.
  - So you likely won't face this issue in the demo

VSCODE – PORT FORWARDING (EXAMPLE)

- After **SSH'ing into ifarm** open terminal (Ctrl/Cmd + `):
  - Clone a simple app that visualizes static data:
    - git clone https://code.jlab.org/rasool/sales-viz.git
  - Move into the cloned directory:
    - cd sales-viz
  - Install all required dependencies before running the app:
    - npm install
  - Run the application:
    - npm run dev



Local Browser

1.

```
[rasool@ifarm2402 ~/gspda-vscode101]$ git clone https://code.jlab.org/rasool/sales-viz.git

Cloning into 'sales-viz'...
remote: Enumerating objects: 2403, done.
remote: Counting objects: 100% (2403/2403), done.
remote: Compressing objects: 100% (1676/1676), done.
remote: Total 2403 (delta 688), reused 2403 (delta 688), pack-reused 0 (from 0)
Receiving objects: 100% (2403/2403), 9.32 MiB | 30.59 MiB/s, done.
Resolving deltas: 100% (688/688), done.
Updating files: 100% (2256/2256), done.
```

2.

```
[rasool@ifarm2402 ~/gspda-vscode101]$ cd sales-viz/
```

3.

```
[rasool@ifarm2402 sales-viz]$ npm install

up to date, audited 65 packages in 810ms

7 packages are looking for funding
  run `npm fund` for details

2 moderate severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force
```

4.

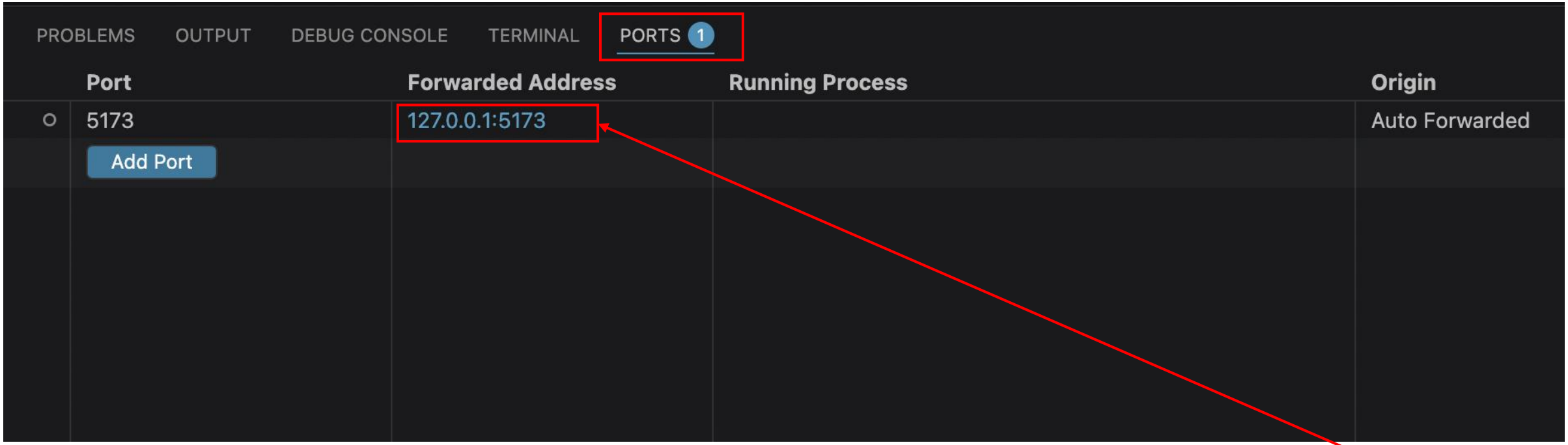
```
[rasool@ifarm2402 sales-viz]$ npm run dev

> sales-viz@0.0.0 dev
> vite


  VITE v4.5.14

  ➜  Local:   http://127.0.0.1:5173/
  ➜  Network: use --host to expose
  ➜  press h to show help
```

Follow link (cmd + click)

Hover over link to see this message

VSCODE – PORT FORWARDING

- You can also open the Ports panel in VS Code to view all currently running application ports and see where they are being forwarded. With the previous app running, the Ports panel looks like this.



- To stop the running application, press Ctrl + C in the terminal.

You can also use **Cmd/Ctrl + click** here to open the port directly.

SECTION 5: X11 FORWARDING

X11 FORWARDING

- X11 Forwarding (GUI over SSH)
  - Allows running GUI apps on a remote server and displaying them locally
  - *Unlike port forwarding (network apps), X11 is for graphical applications*
  - Example
    - Run apps like xclock, ROOT on ifarm
    - GUI window appears on your local machine

- Requirements
  - Local system needs an X server
    - macOS → XQuartz (https://www.xquartz.org/)
    - Windows → VcXsrv (https://sourceforge.net/projects/vcxsrv/)
    - Linux → built-in support

X11 FORWARDING – USING VDI

- **Easier Option — VDI**
  - Use VDI for GUI applications
  - Already supports graphical display (no setup needed)
  - Shares the same filesystem with ifarm

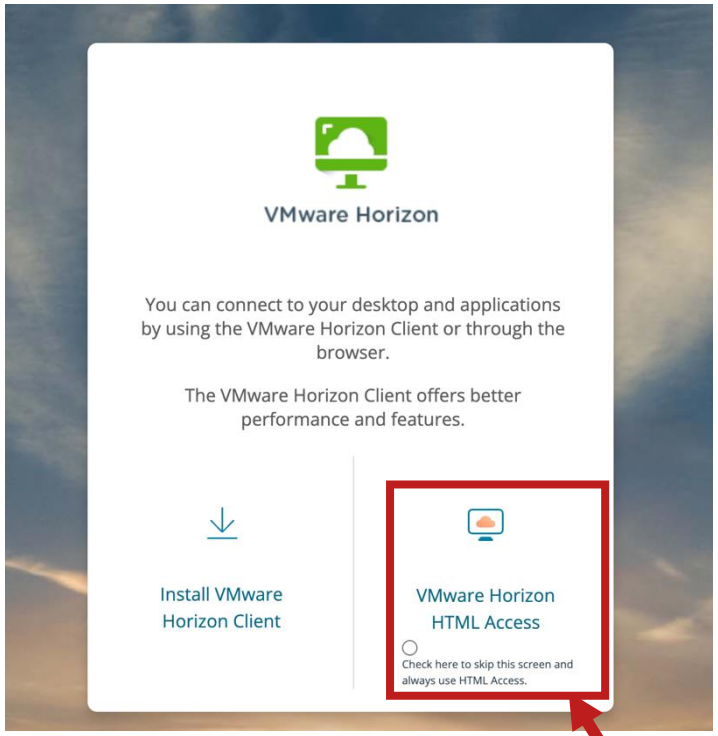
- When to use
  - Works if the app does not require ifarm-specific compute resources or environment
  - If heavy computation or ifarm-only setup is needed:
    - use ifarm with X11 forwarding instead (next slides)

- How it works
  - Log in to VDI
  - Run GUI apps directly from the terminal
  - No X11 forwarding or X server required

X11 FORWARDING – USING VDI

- Login to VDI

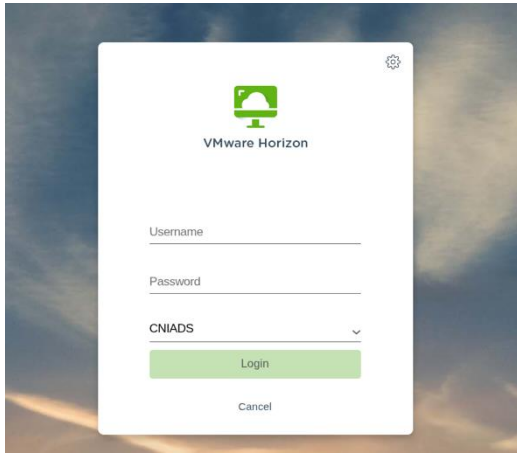
1.



https://vdi.jlab.org

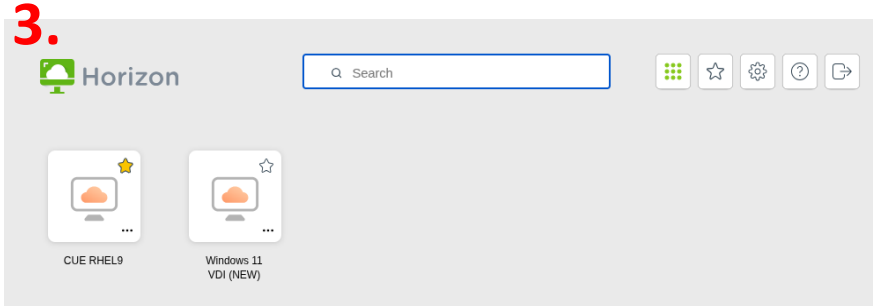
Web access

2.



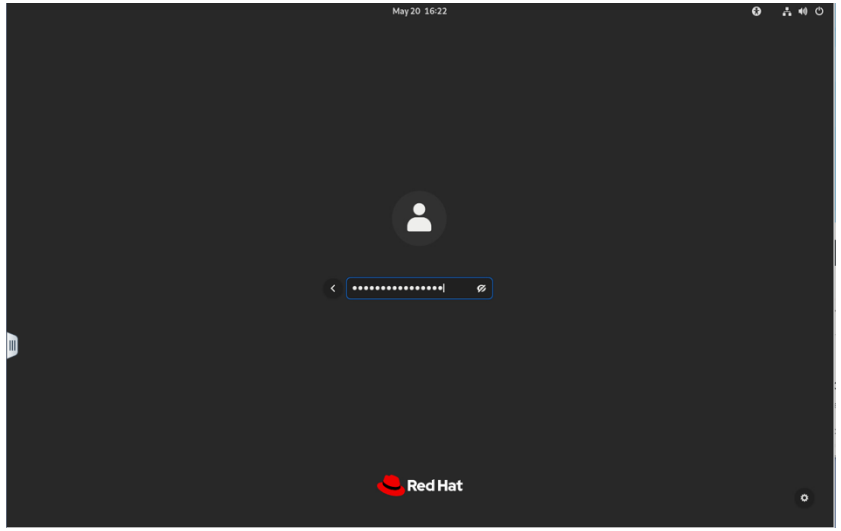
Log in with CUE Username and Password

3.



Select RHEL9 machine (Windows requires SmartCard!!)

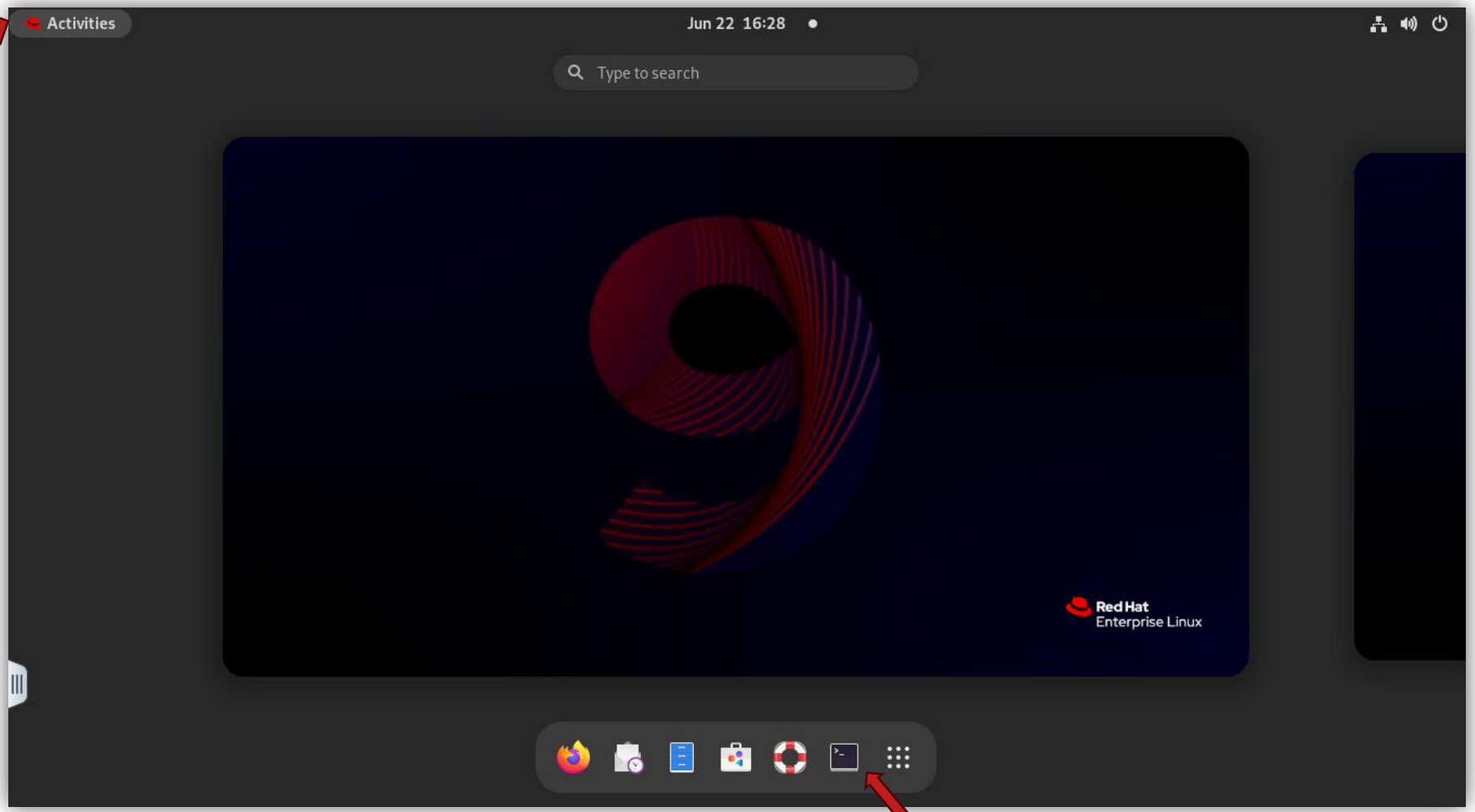
4. Enter CUE Username
(Password = **MobilePASS** SAS MFA PIN+Code)



X11 FORWARDING – USING VDI

- Open the terminal

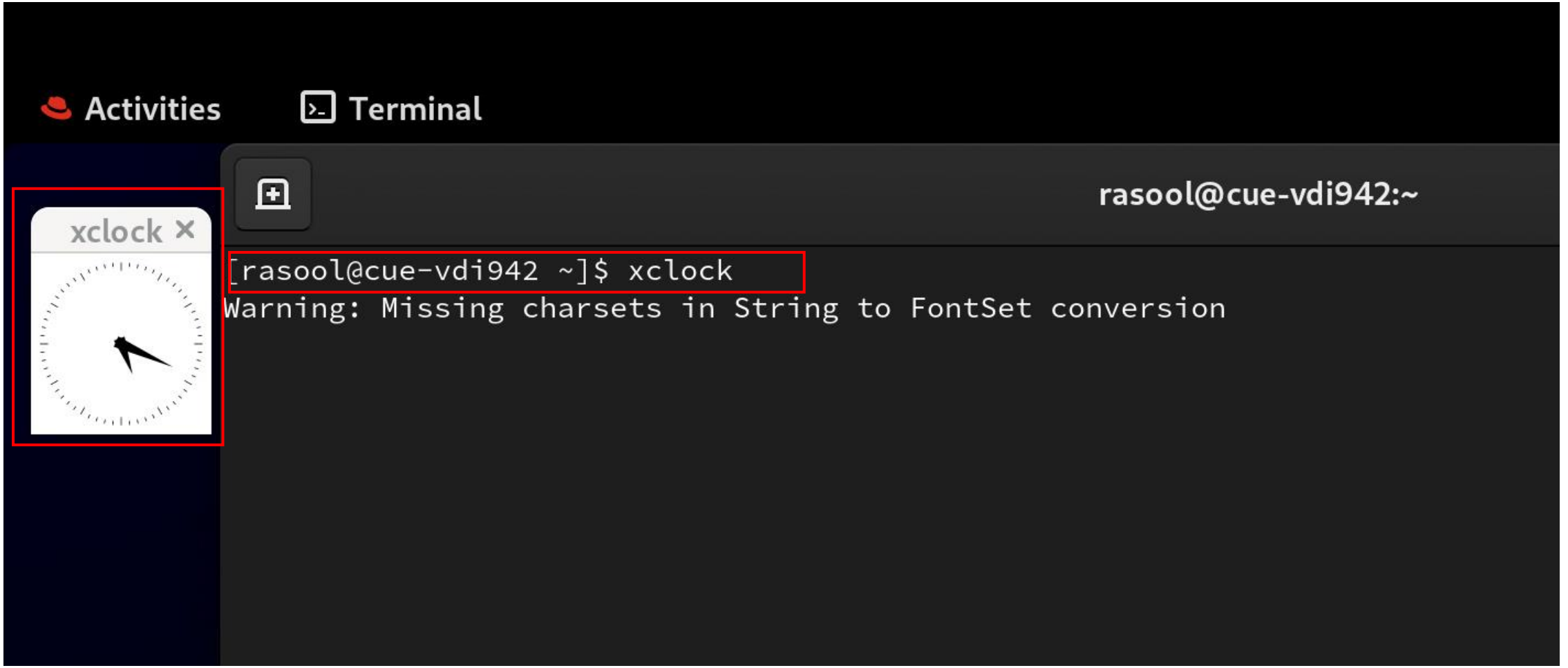
1. Click activities button



2. Click Terminal Icon

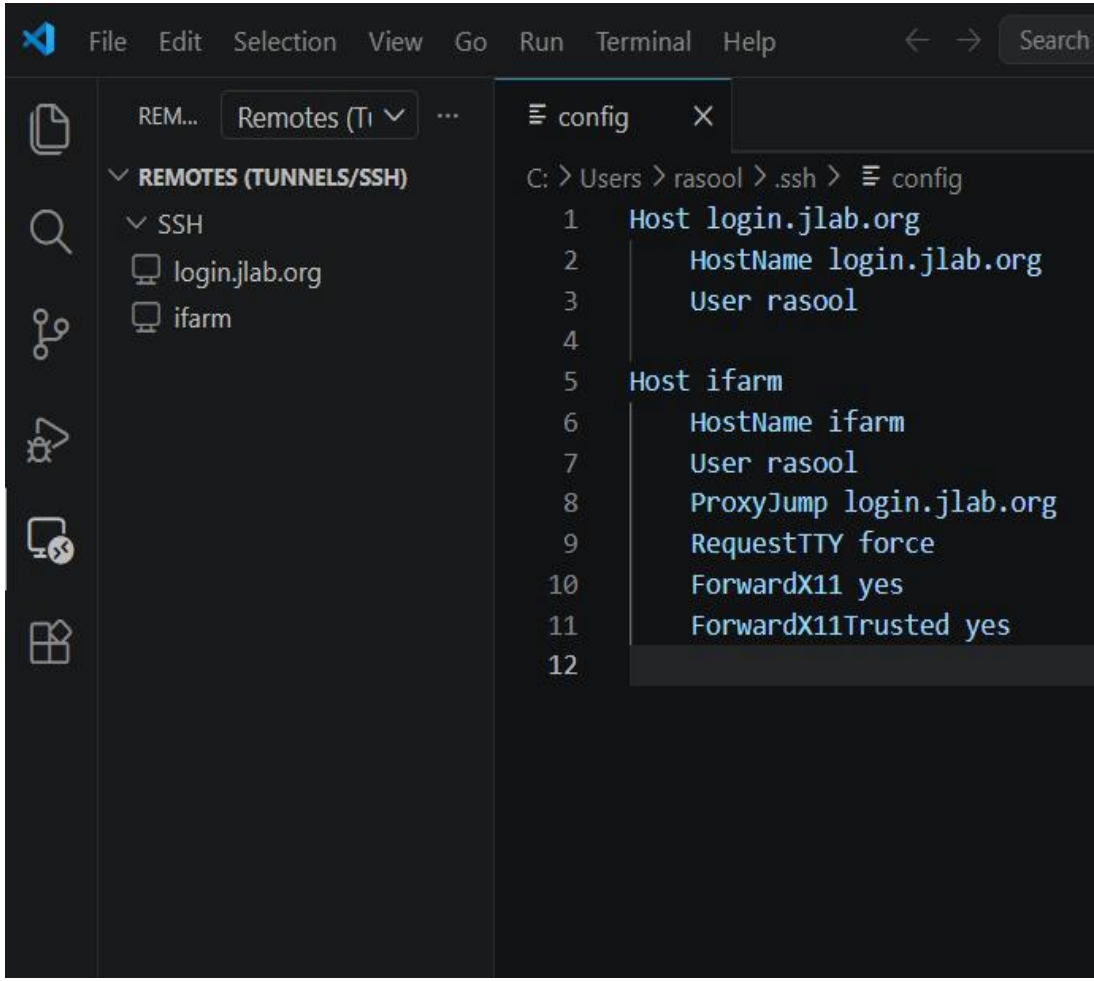
X11 FORWARDING – USING VDI

- Run the GUI app



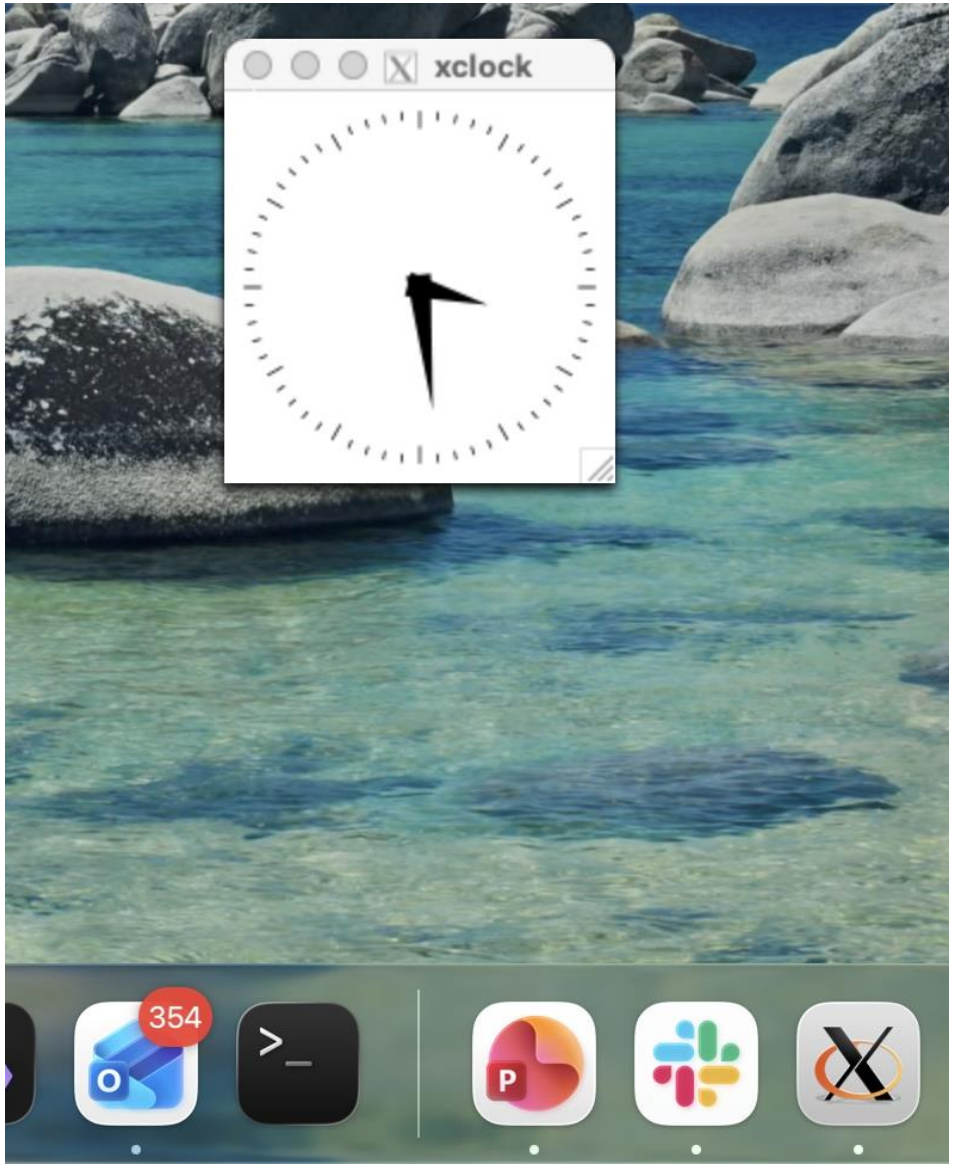
X11 FORWARDING – BY SSH'ING TO IFARM

- As during SSH config setup for ifarm, we already added following to ~/ssh/config file:
  - ForwardX11 yes
  - ForwardX11Trusted yes
- Now, you only need to install an X server on your local machine based on your OS:
  - macOS → XQuartz
    - (https://www.xquartz.org/)
  - Windows → VcXsrv
    - (https://sourceforge.net/projects/vcxsrv/)
  - Linux → Built-in support
    - (no additional setup needed)



X11 FORWARDING – BY SSH'ING TO IFARM

- Once X server is installed:
  - SSH into ifarm using VS Code as before.
  - Open the VS Code terminal and run any GUI application
    - The X server will:
      - Start automatically
      - Display the remote app's GUI on your local machine.

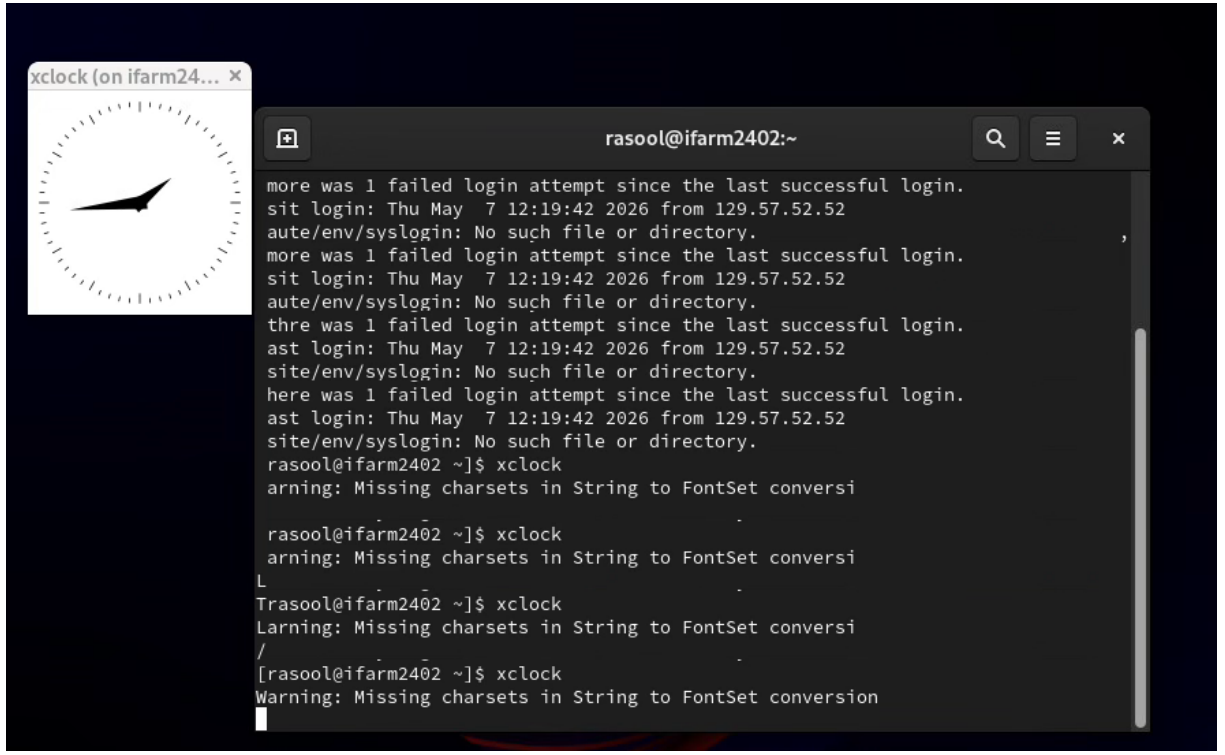


X11 FORWARDING – BY SSH'ING TO IFARM

⚠ **Offsite GUI Slowness (ifarm)**

- If you run a GUI app on ifarm while offsite, it may become very slow
- This happens because the X Server is running on your local machine, outside the JLab network
- As a result, GUI rendering and interactions can lag significantly
- **Fix:**
  - Log into the VDI
  - Open a terminal inside VDI
  - SSH into ifarm from there and run your application

  This way, the X Server runs inside the JLab network, making GUI performance much smoother



X11 FORWARDING – BY SSH'ING TO IFARM

⚠ **Offsite GUI Slowness (ifarm)**

- **Fix:**
  - SSH command:
    - ssh -Y -J <username>@login.jlab.org <username>@ifarm.jlab.org
      - -Y = X11 forwarding with encryption
      - -J = ProxyJump
- In most cases performance is fine, but if you experience slowness, this is the recommended workaround

SSH to ifarm

```
[rasool@cue-vdi9122 ~]$ ssh -Y -J rasool@login.jlab.org rasool@ifarm.jlab.org
(rasool@login.jlab.org) Password:
rasool@ifarm.jlab.org's password:
```

Enter PIN + USB/MobilePass Token

Enter JLab Account Password

Run the app

```
[rasool@ifarm2402 ~]$ xclock
Warning: Missing charsets in String to FontSet conversion
```

SECTION 6: GITLAB FIRST COMMIT & PUSH

GITLAB - (CODE.JLAB.ORG)

- **What is Git?**
  - A version control system that tracks changes in your code over time
  - Think of it as a save history for your project

- **Why use Git?**
  - Go back to previous versions if something breaks
  - See what changed and when
  - Work on new features without affecting the main code

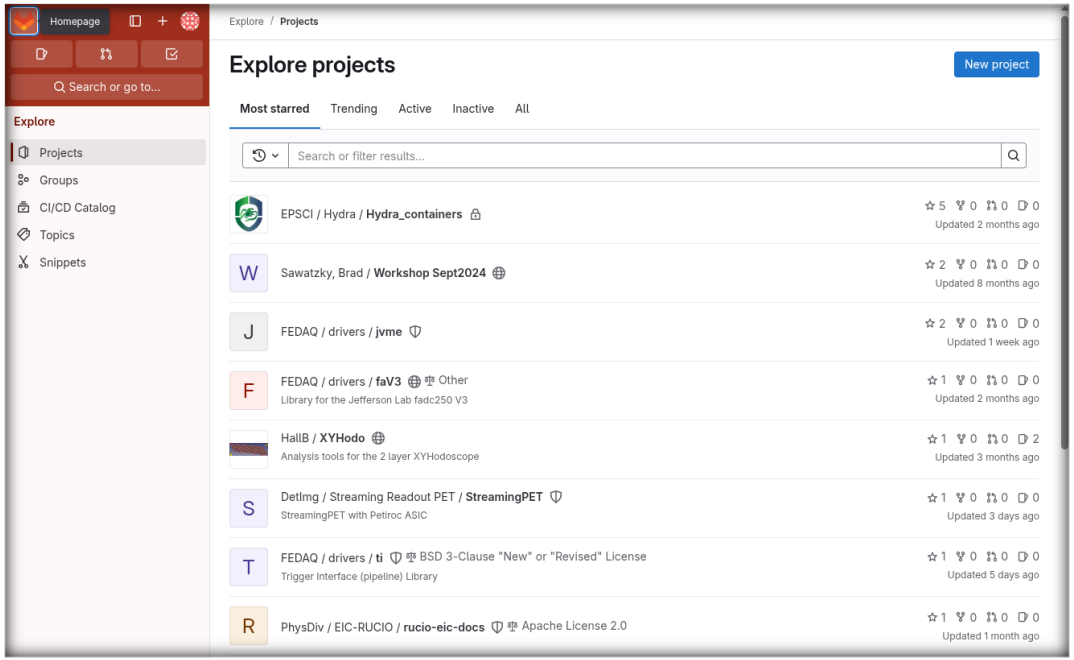
- **Collaboration**
  - Multiple people can work on the same project
  - Changes can be combined safely

- In short: Git helps you manage, track, and safely update your code

GITLAB - (CODE.JLAB.ORG)

GitLab

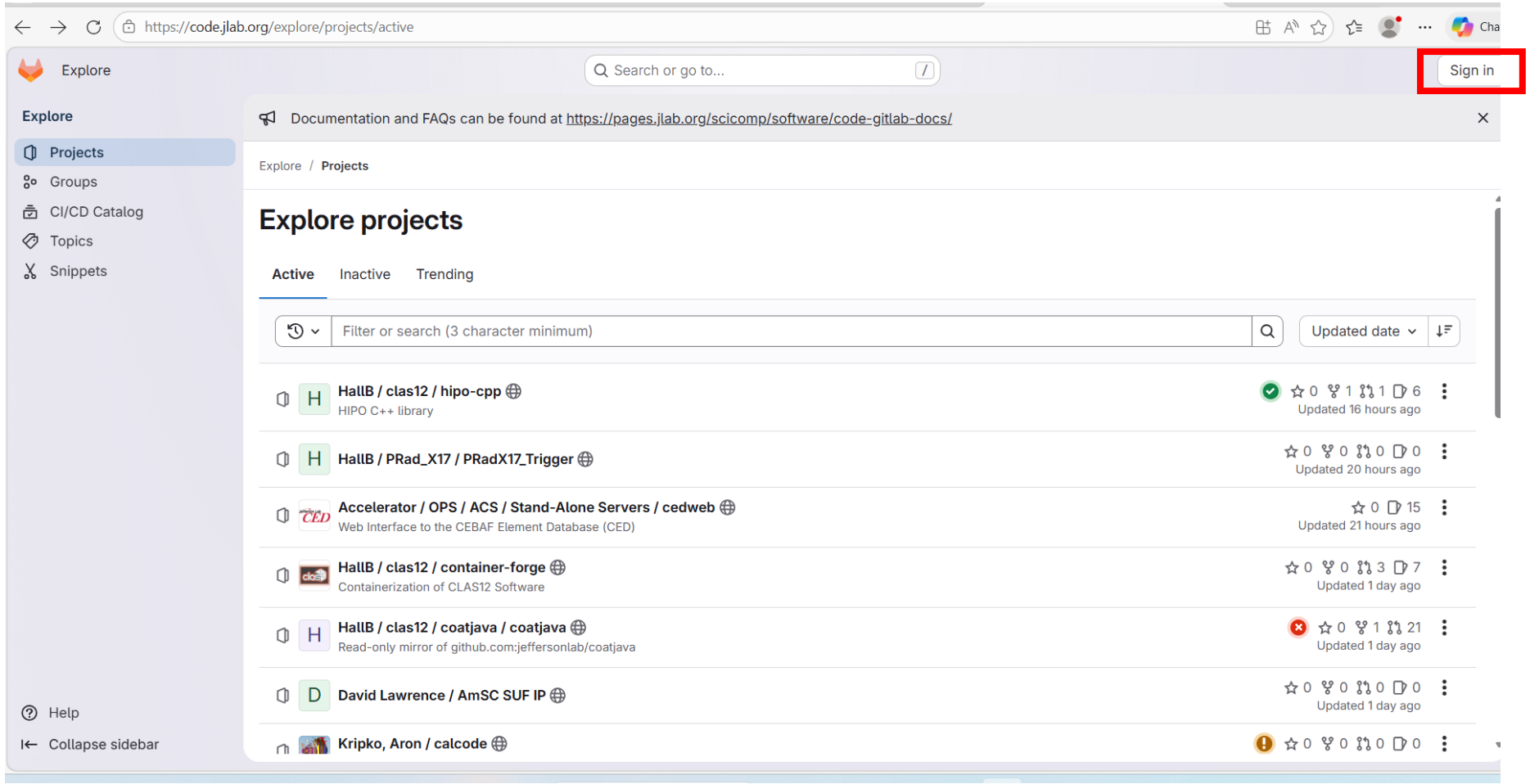
- A web-based platform built on top of Git for collaboration and project management
- Makes working with Git repositories easier through a visual interface
- Adds team-level tools beyond version control
- On top of Git, it enables:
  - Remote repositories (cloud-based code storage)
  - Collaboration via merge requests (code reviews)
  - Issue tracking (bugs, tasks, features)
  - CI/CD pipelines (automated testing & deployment)
  - Project wikis and documentation



In short: Git handles version control. GitLab turns it into a collaborative development platform.

GitLab – Creating a new repo

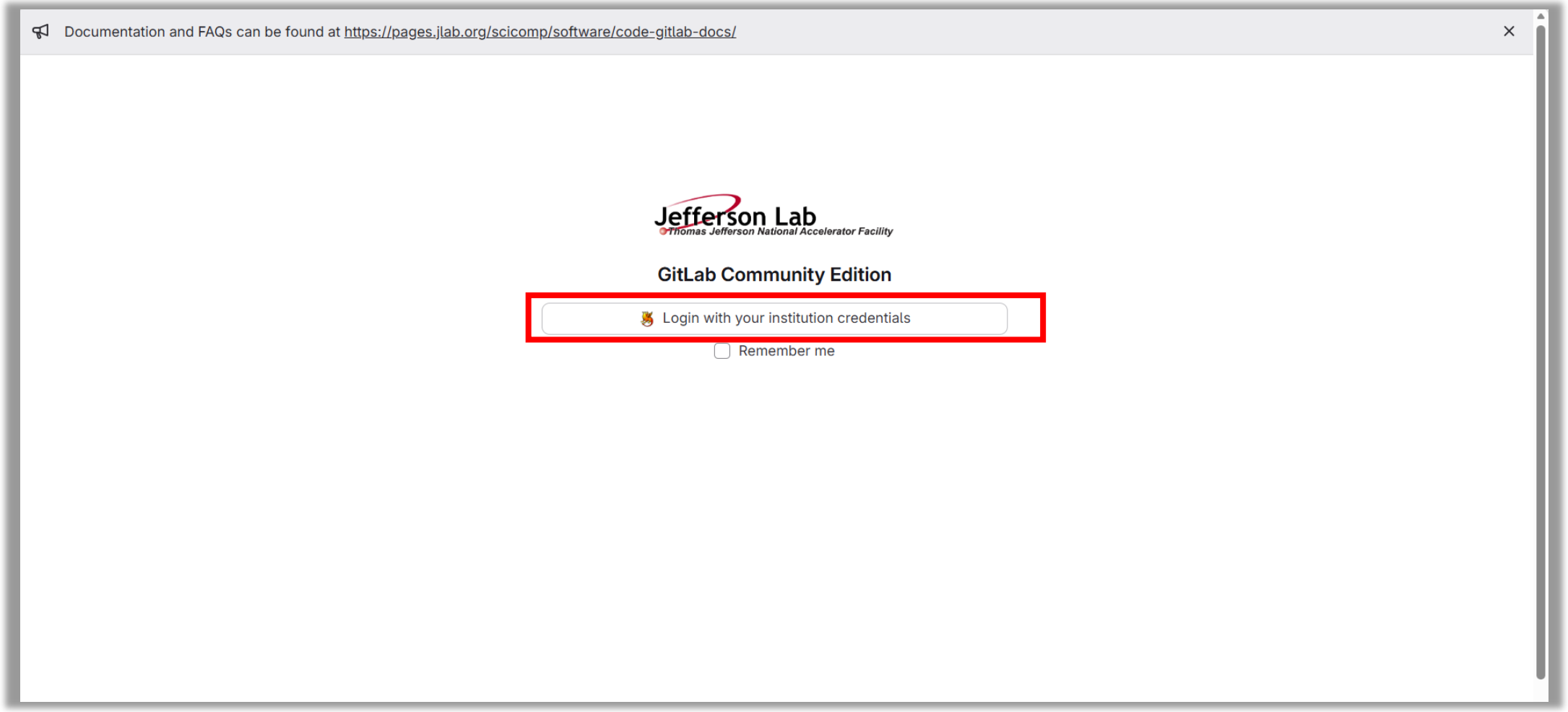
- Log in to the GitLab instance deployed at **code.jlab.org** using your JLab or federated account credentials.
  - Use your JLab CUE account if you have one for creating repo



https://code.jlab.org/

GITLAB – CREATING A NEW REPO

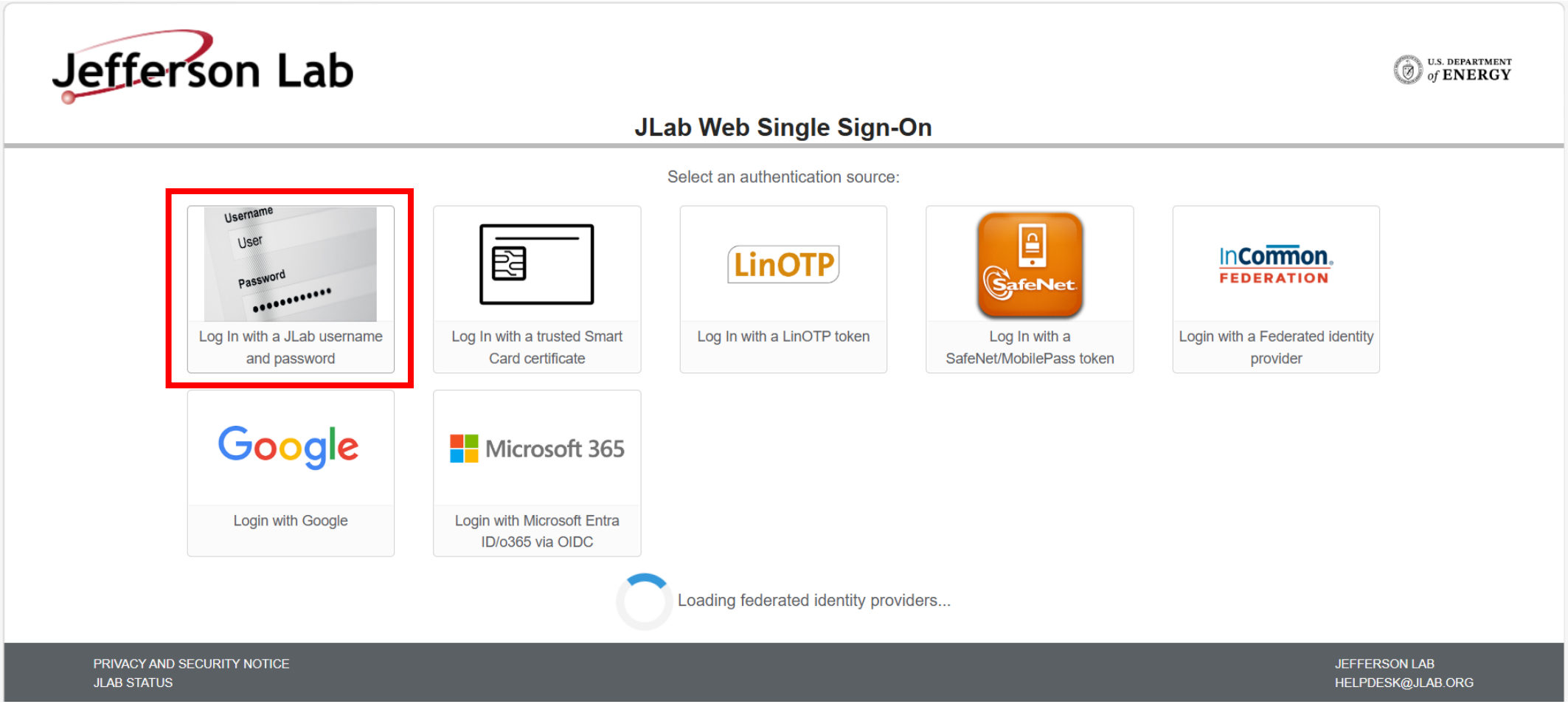
- Log in to the GitLab instance deployed at **code.jlab.org** using your JLab account credentials.



https://code.jlab.org/

GITLAB – CREATING A NEW REPO

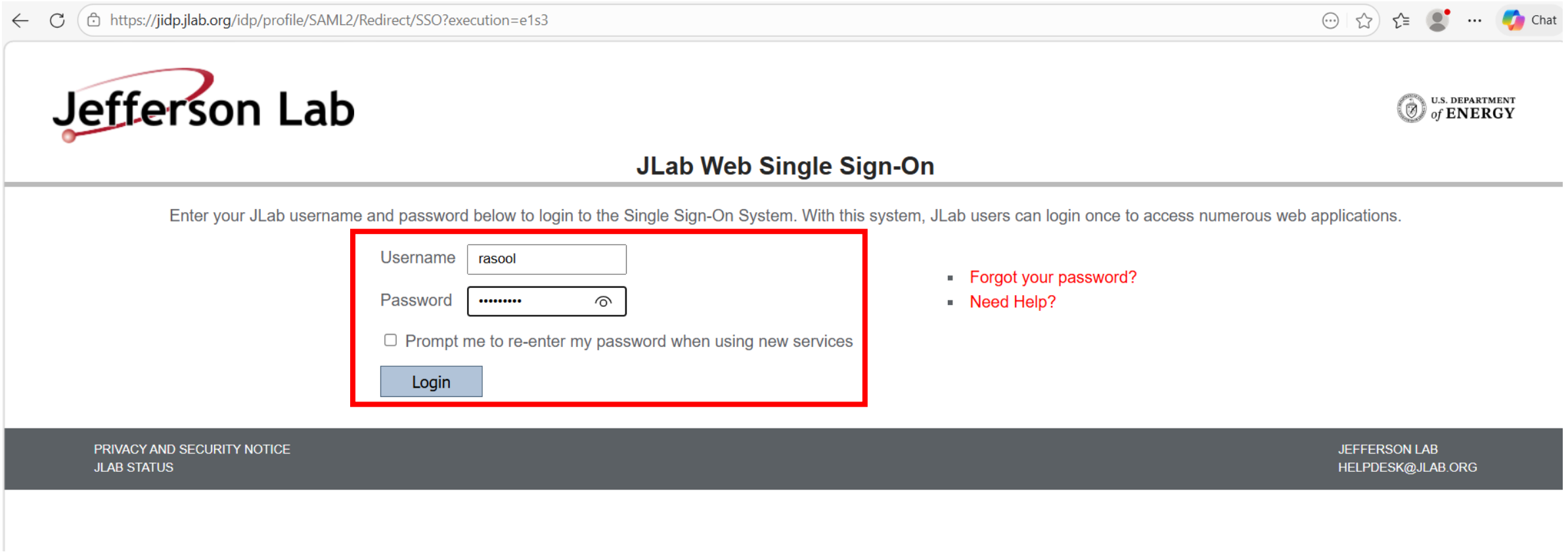
- Log in to the GitLab instance deployed at **code.jlab.org** using your JLab account credentials.



https://code.jlab.org/

GitLab – Creating a new repo

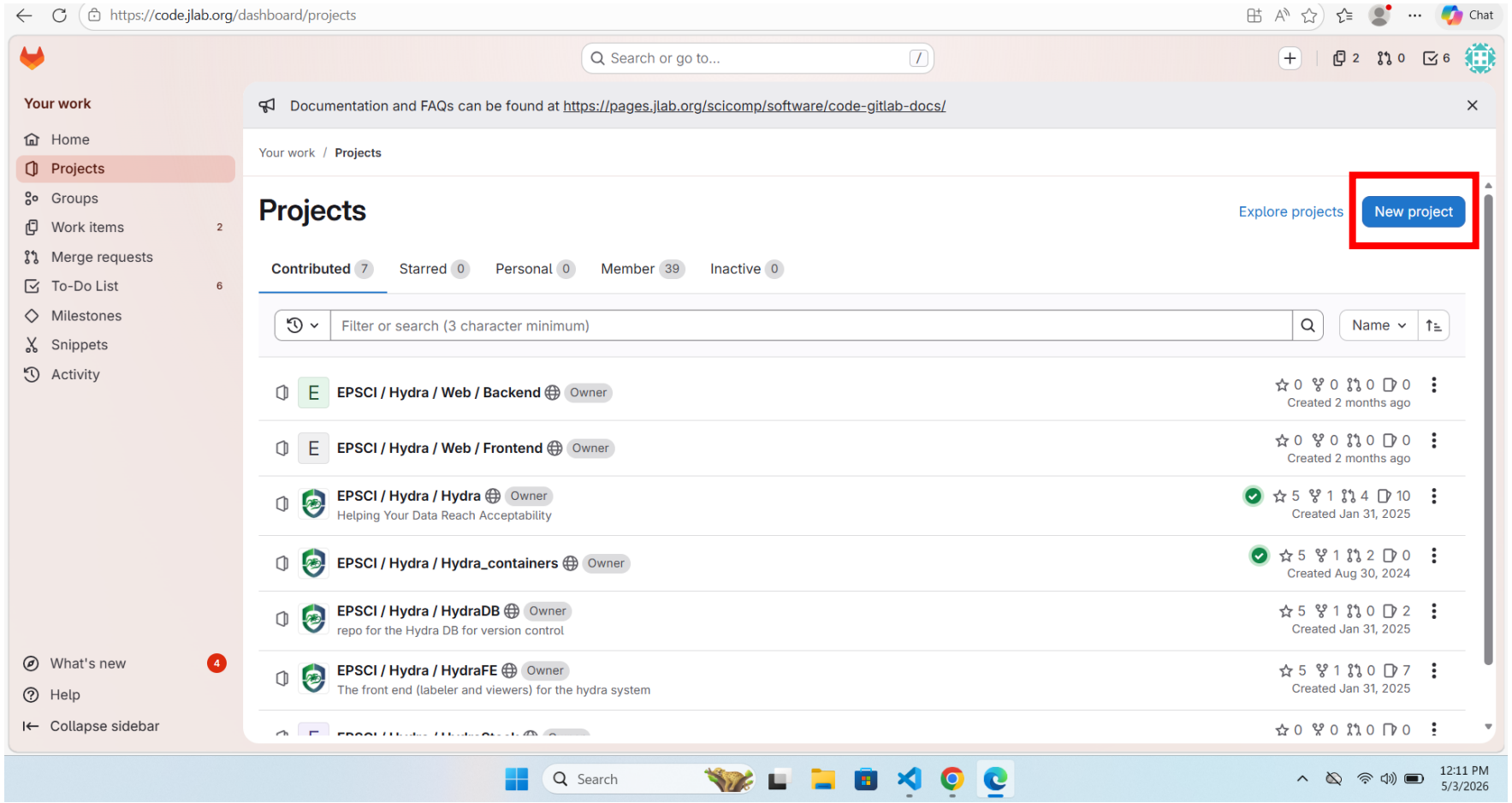
- Log in to the GitLab instance deployed at **code.jlab.org** using your JLab account credentials.



https://code.jlab.org/

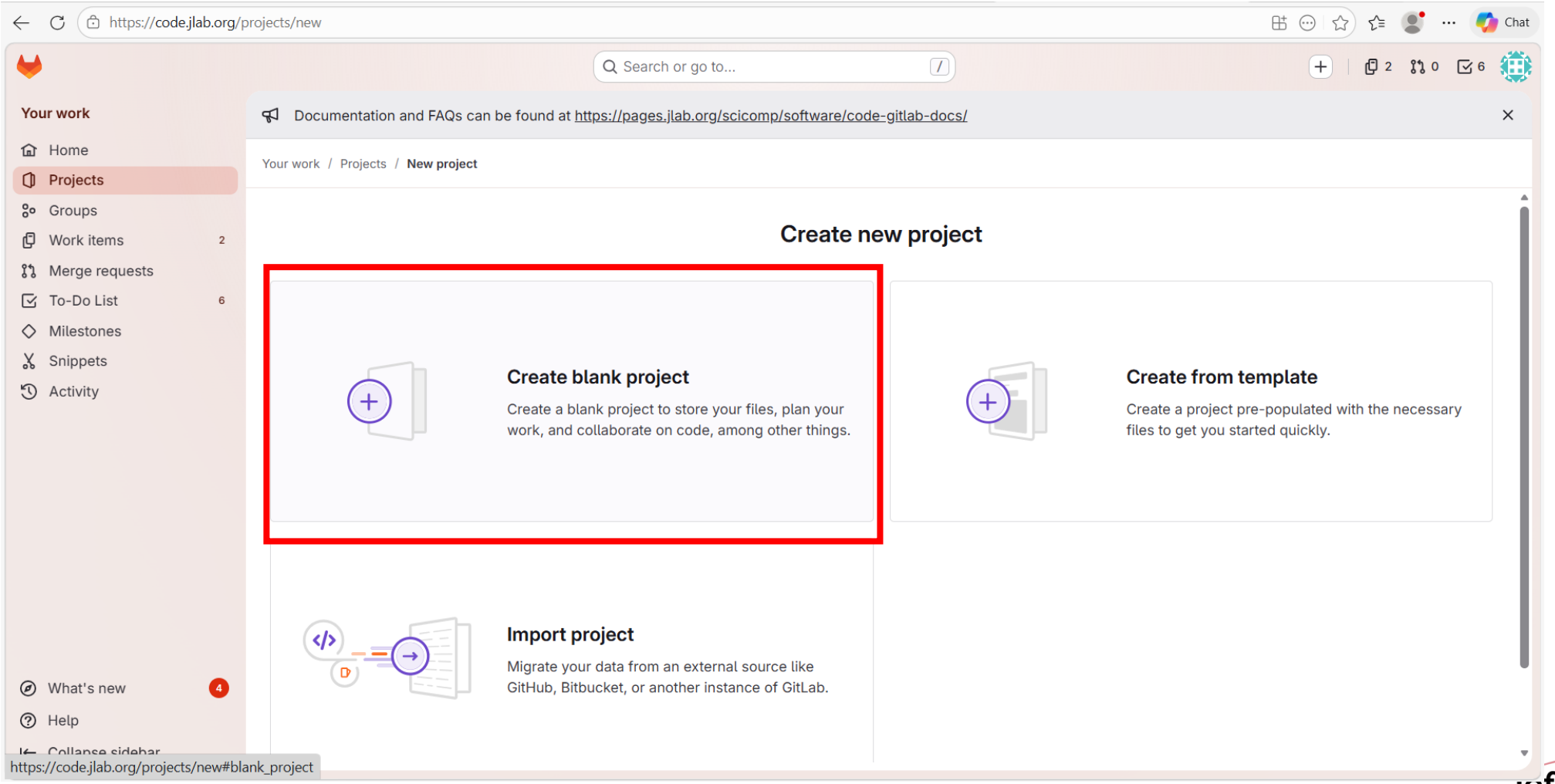
GITLAB – CREATING A NEW REPO

- Once logged in, you'll see the list of projects under JLAB. Click the "New Project" (repository) button in the top-right corner.



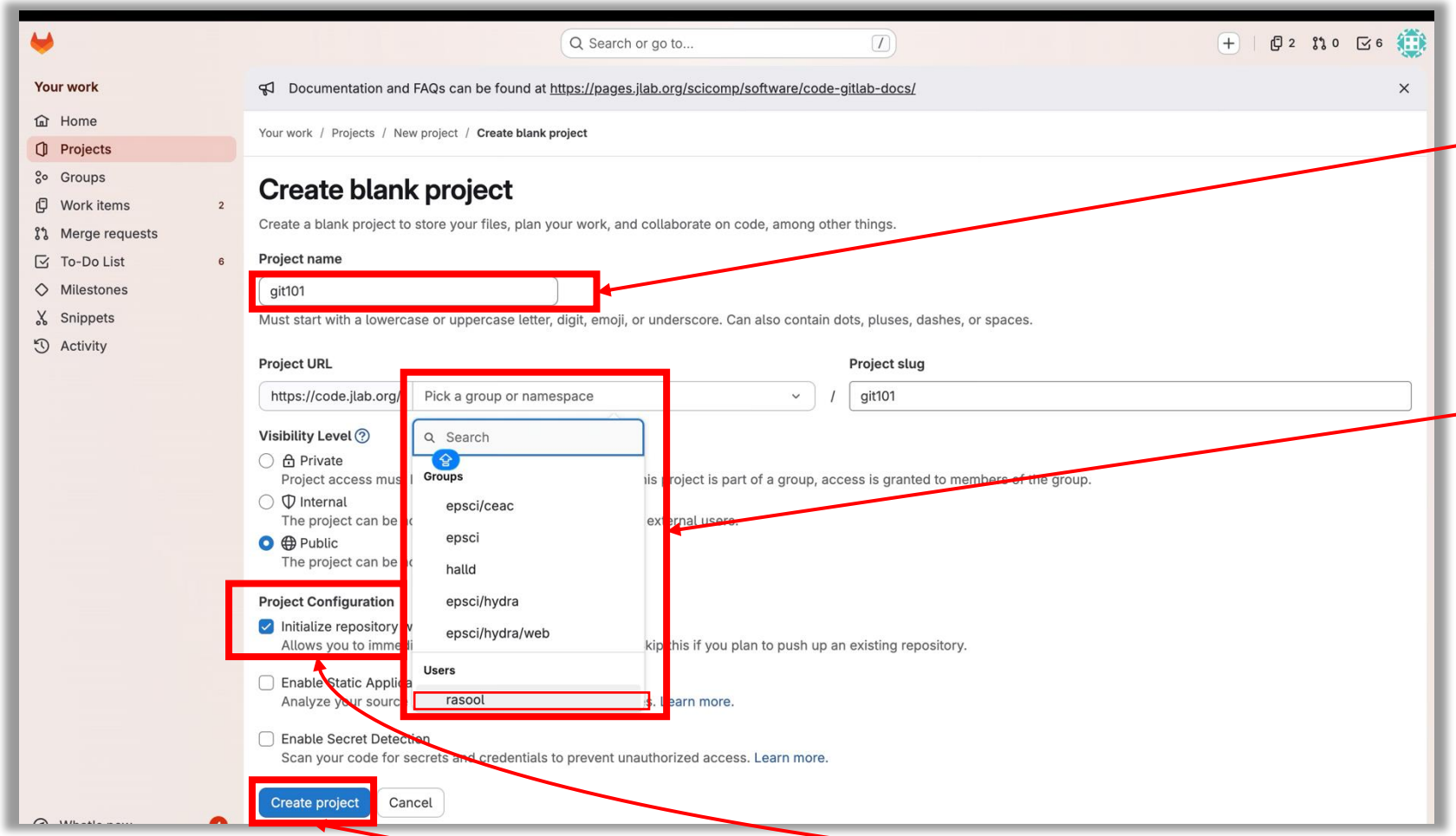
GITLAB – CREATING A NEW REPO

- Select “Create blank project.” Other options can also be used, but for simplicity, start with a blank project.



GITLAB – CREATING A NEW REPO

- Now fill in all the required details to create the GitLab project as shown below.



1. Add Project Name

2. Choose a namespace:
- For now, select your own username.
- For future work, choose the group associated with your project if any available.
  - You can also create groups

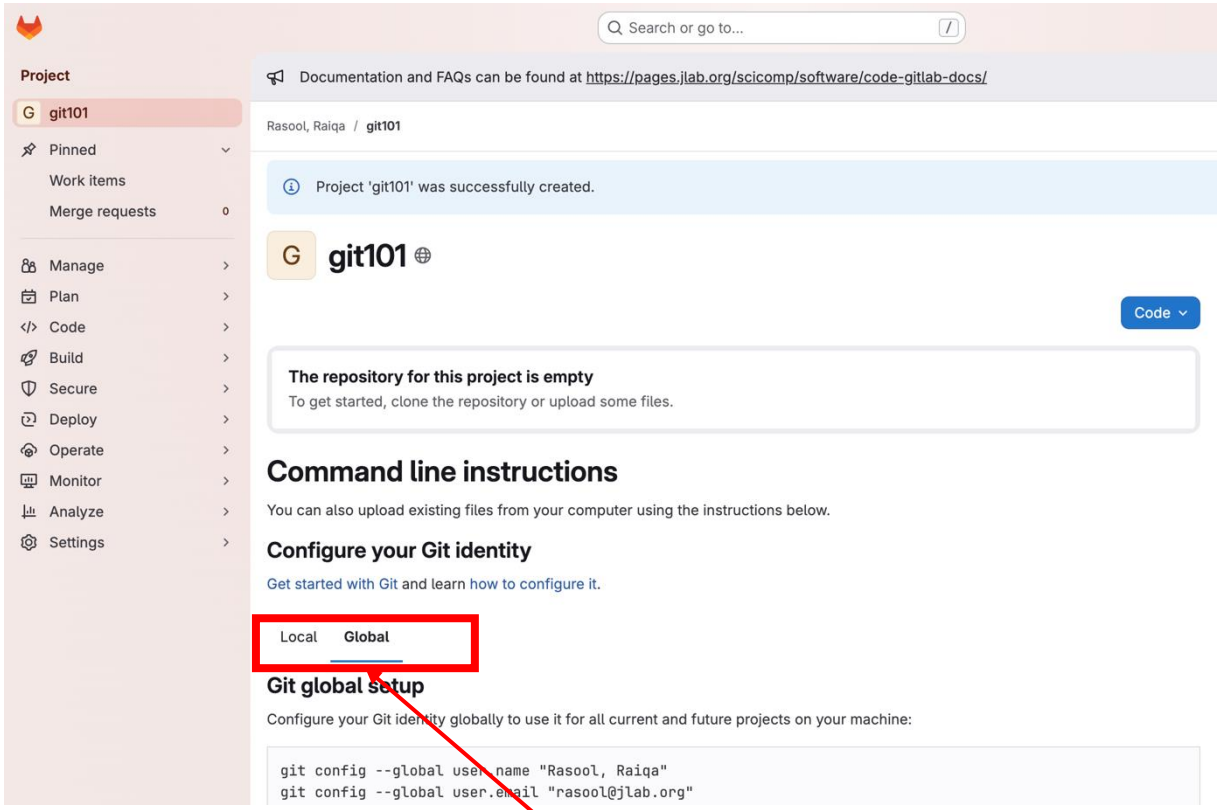
3. Uncheck "Initialize repository with a README"
We will add the README in our first commit, so it's not needed at setup.

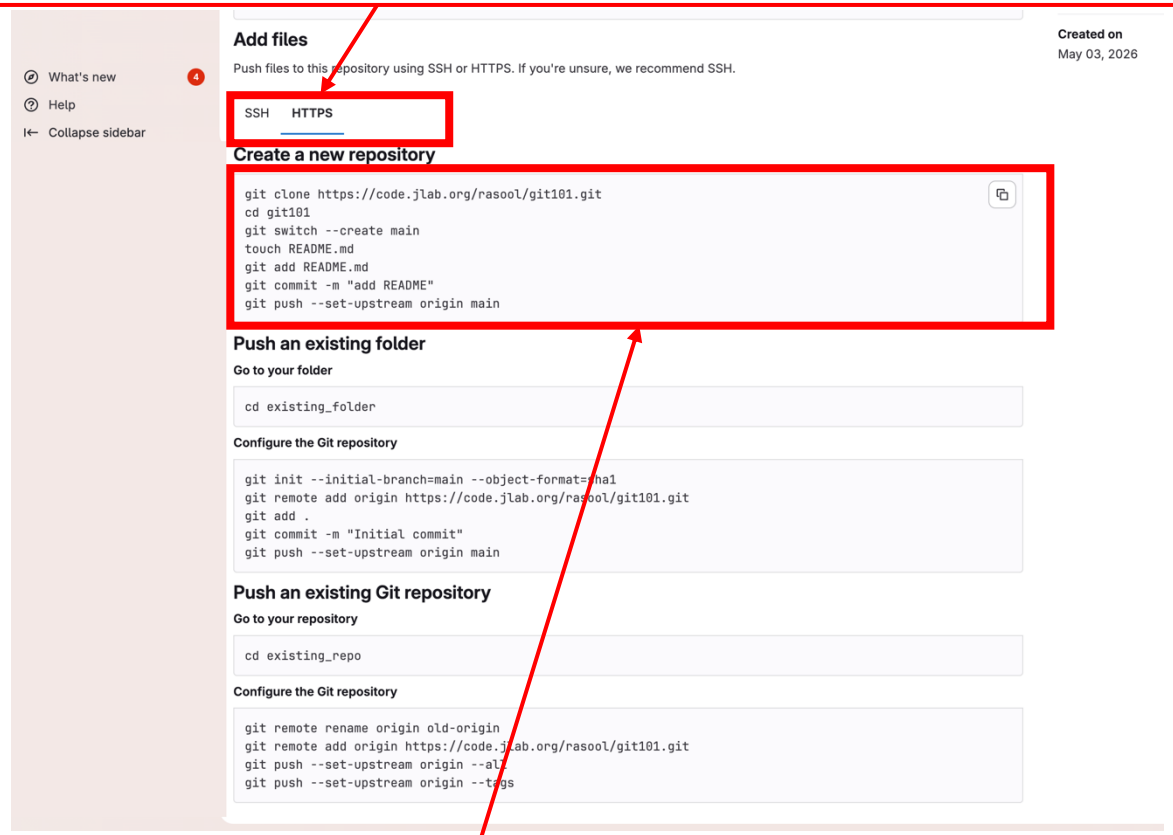
4. Create Project

GitLab – Creating a new repo

- You should now land on a page that provides all the details needed to connect your local repository to this remote GitLab repository.

2. We will use HTTPS to push changes after committing, so switch to the HTTPS tab.





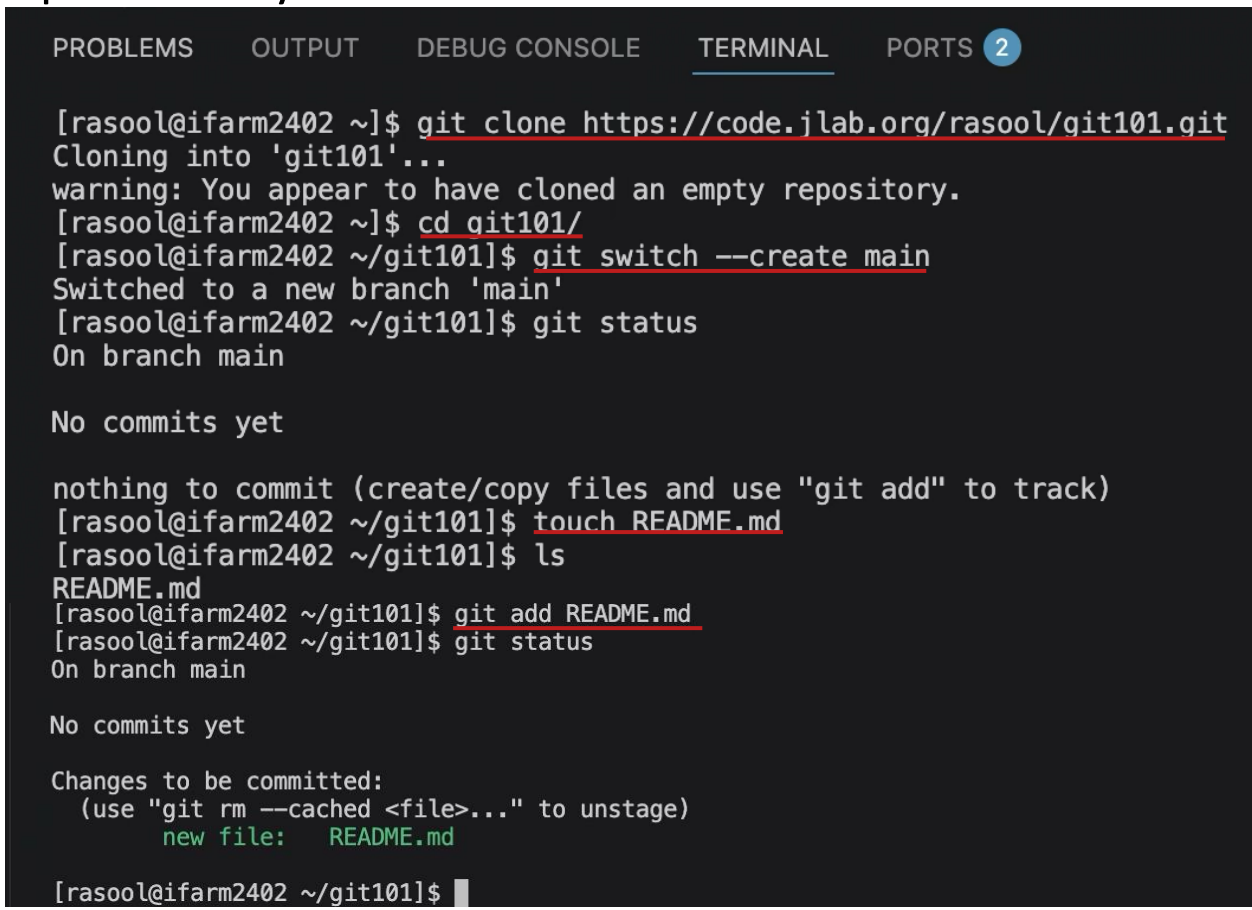
1. We will set the configuration globally so you don't have to repeat it for every repository. Switch to the Global tab.

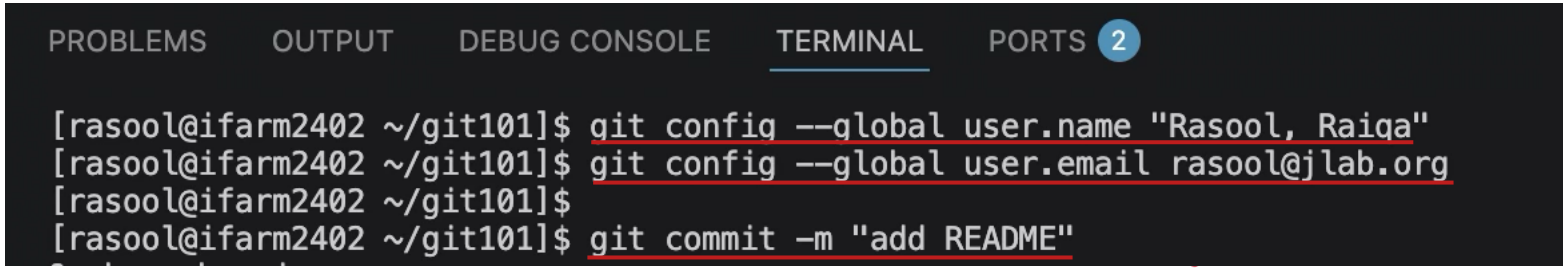
3. We are creating a new repository so we will follow these instructions

GITLAB – YOUR FIRST COMMIT

SSH into ifarm, open the terminal, and run the commands provided by GitLab:

- Clone the repository
  - git clone https://code.jlab.org/rasool/git101.git
- Move into the project directory
  - cd git101
- Create and switch to a new branch
  - git switch --create main
- Create a README file
  - touch README.md
- Stage the changes
  - git add README.md
- Set global Git identity (required for commits)
  - git config --global user.name "Rasool, Raiqa”
  - git config --global user.email rasool@jlab.org
- Make your first commit
  - git commit -m "add README"

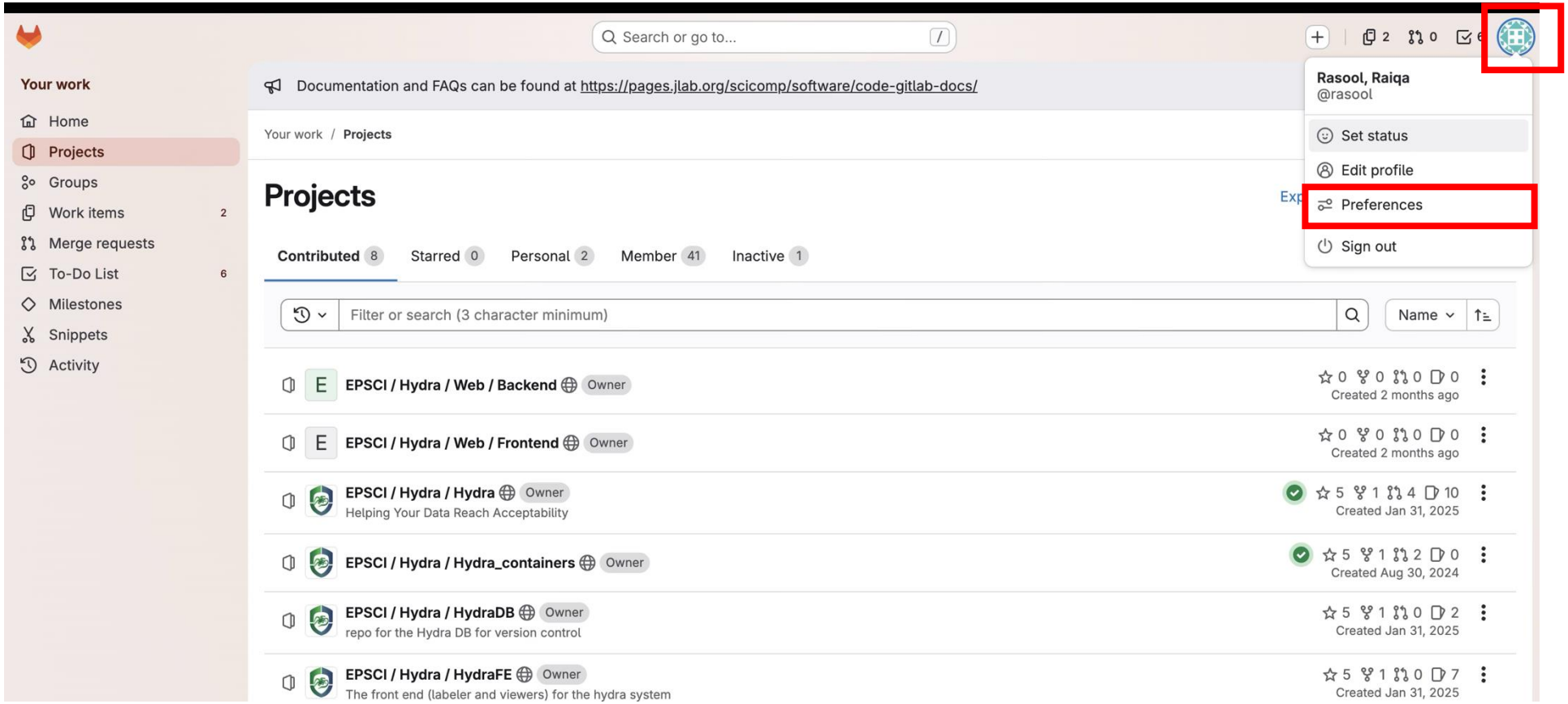




GITLAB – PUSHING FIRST COMMIT

- Before doing your first push, you need to set up a Personal Access Token (PAT) and enable credential storage so you don't have to re-enter your login details every time.

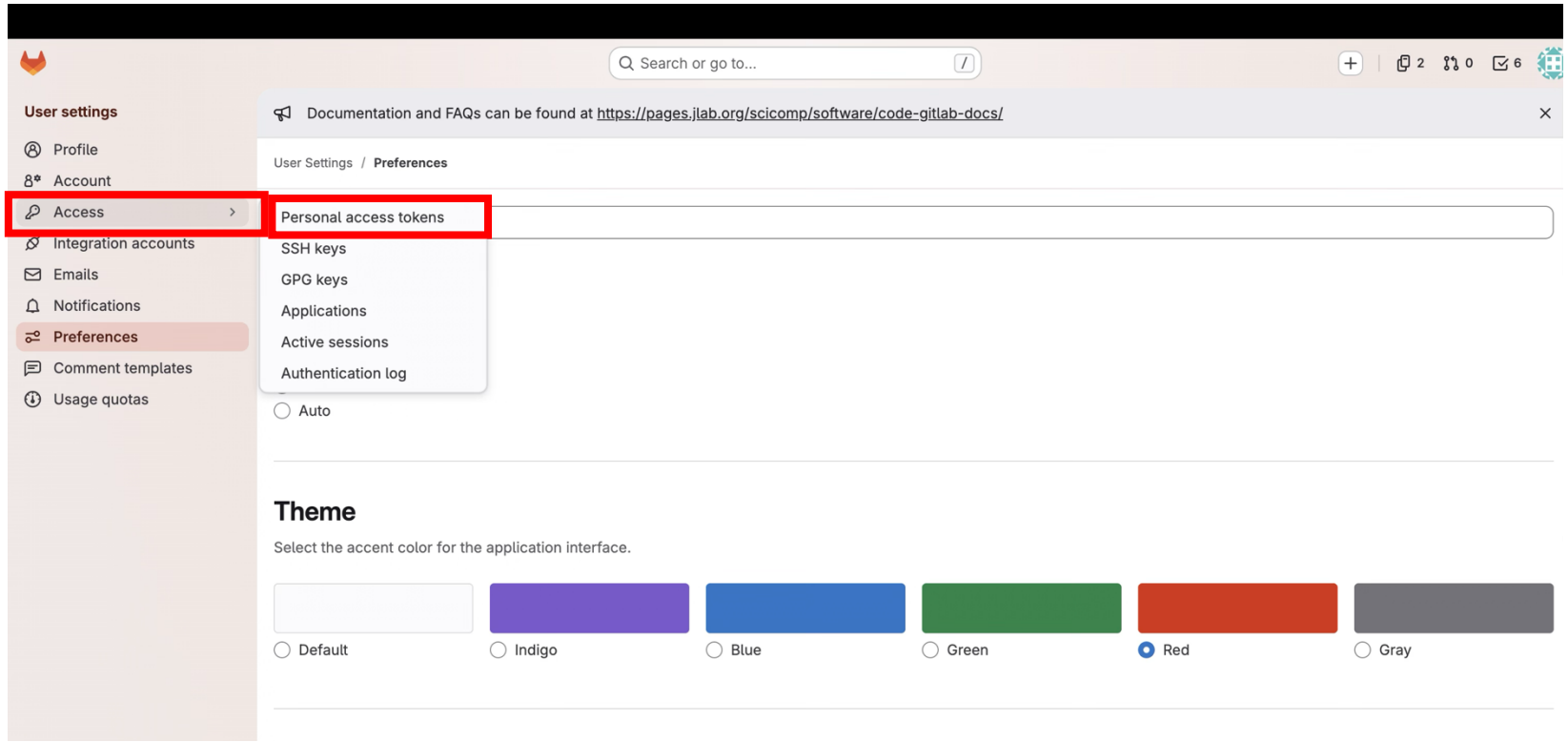
- **Generate a Personal Access Token**
  1. Click your profile avatar in the top-right corner, then select Preferences.



GITLAB – PUSHING FIRST COMMIT

- Before doing your first push, you need to set up a Personal Access Token (PAT) and enable credential storage so you don't have to re-enter your login details every time.

- **Generate a Personal Access Token**
  2. Go to "Access" and select "Personal access tokens."

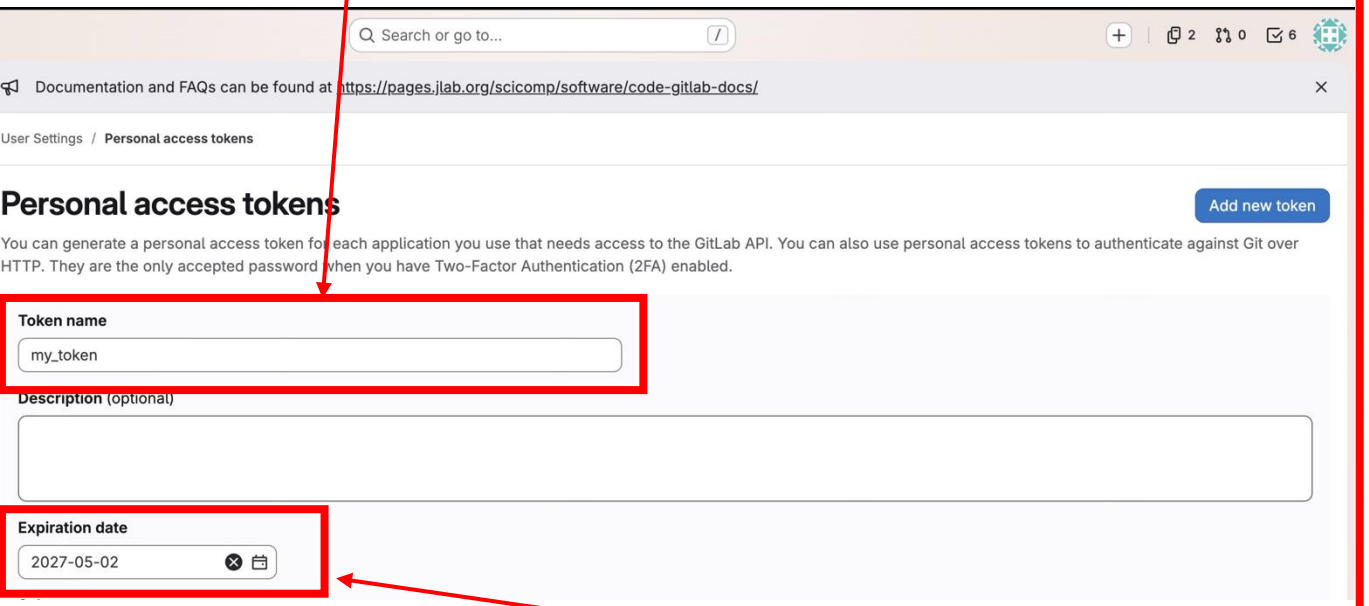


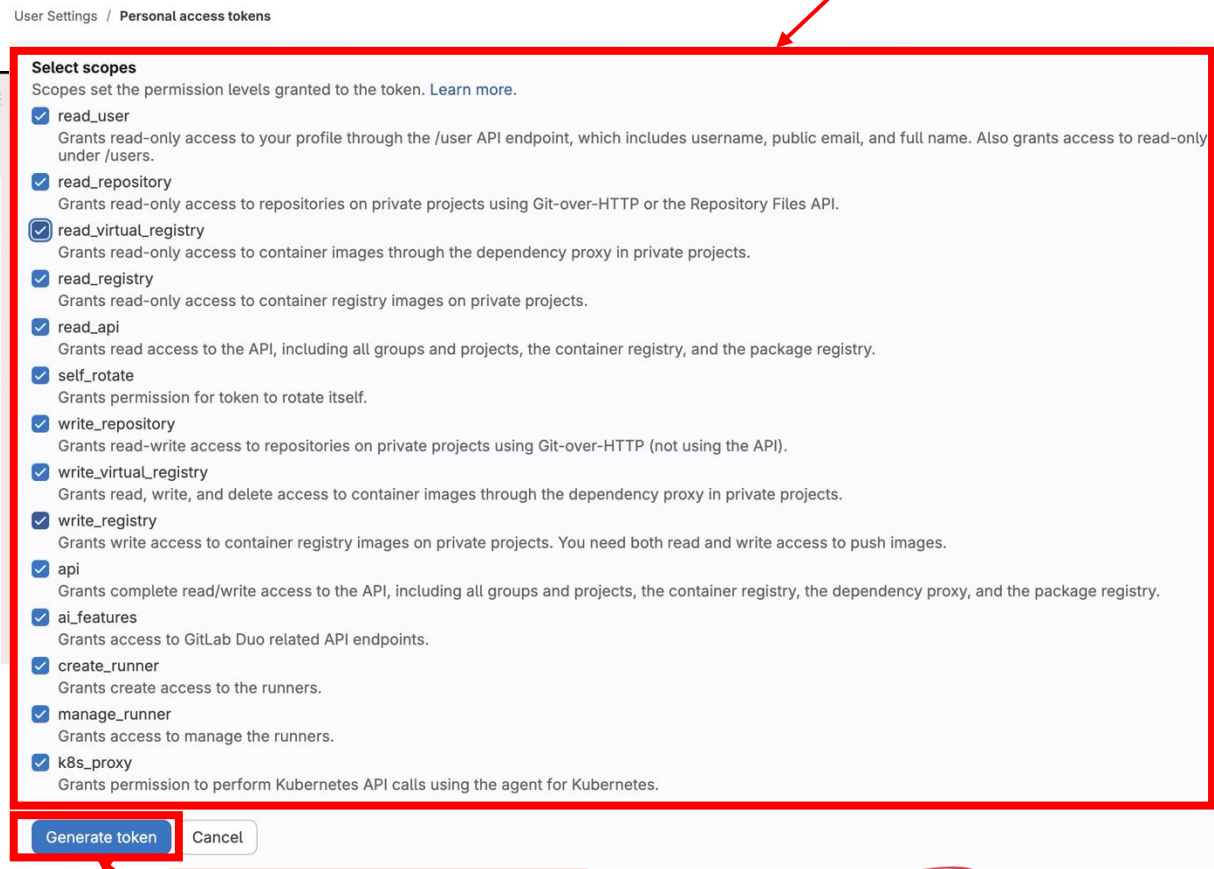
GITLAB – PUSHING FIRST COMMIT

- **Generate a Personal Access Token**
  3. Enter all required details and hit “Generate token” button:

1. Give Token a name
- This will show as this token’s name in access tokens list

3. Set token permissions
- Preferably, choose only the permissions you need.
- For simplicity in this setup, check all





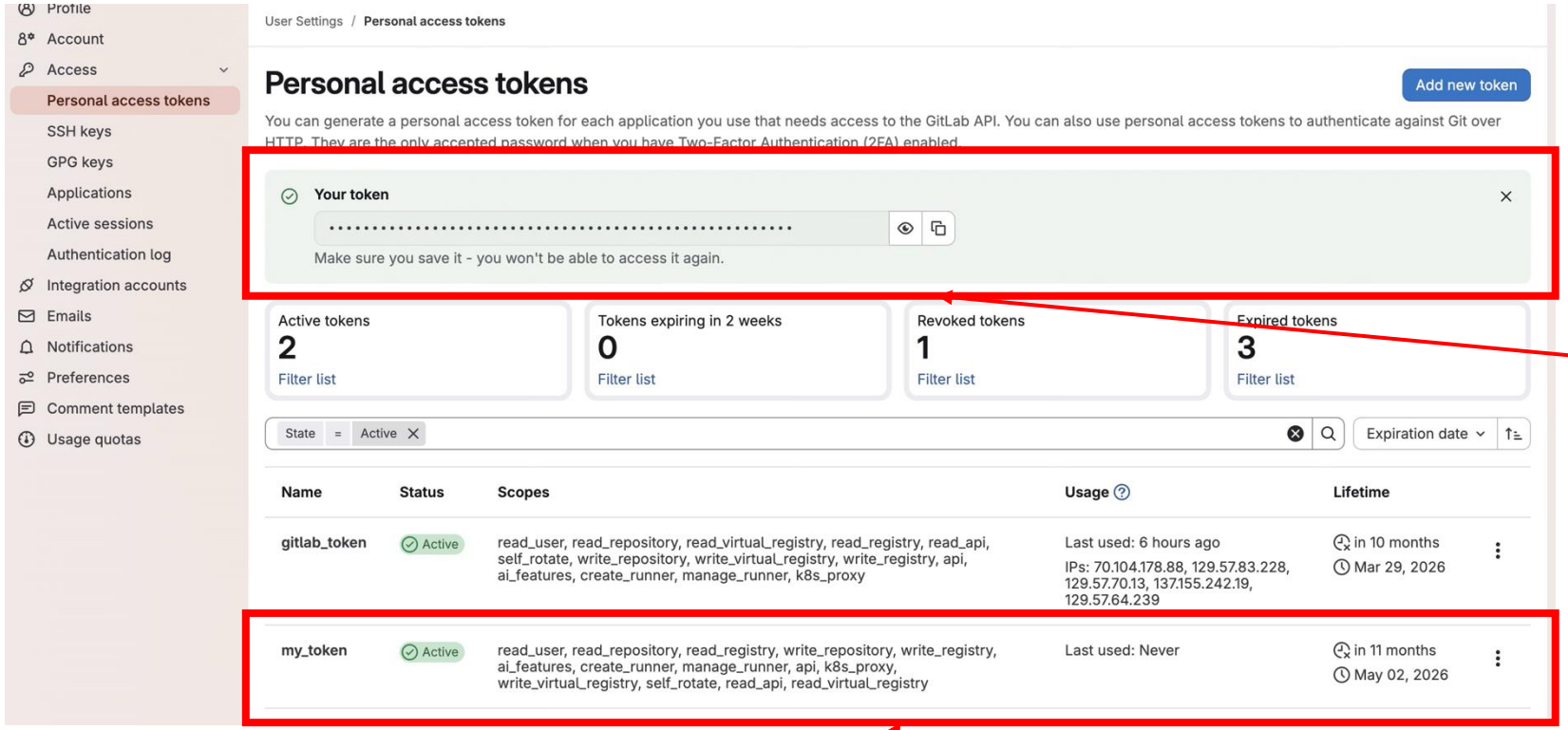
2. Set expiration date
- Recommended: ~6 months (more secure, requires renewal).
- Alternatively: set to max (usually 1 year) to avoid frequent re-setup.

4. Generate Token

GITLAB – PUSHING FIRST COMMIT

- **Generate a Personal Access Token**

  4. After clicking “Generate token”, your personal access token will be created, and you should see a confirmation screen like this:



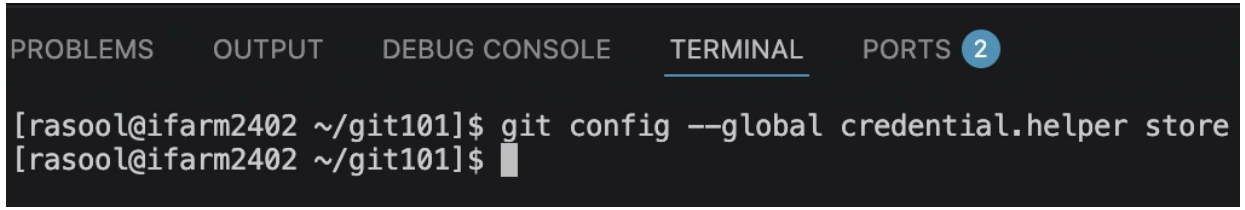
Copy this token—we will use it in later steps as well. Please keep this tab open until the setup is complete, as this token will not be shown again.

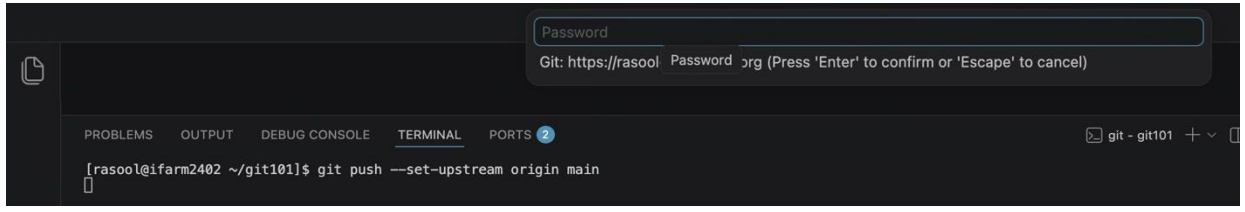
Note: This token is like a key to your GitLab home, so do not share it publicly. Once you have completed the next slides setup steps, you can remove it from wherever you temporarily copied it.

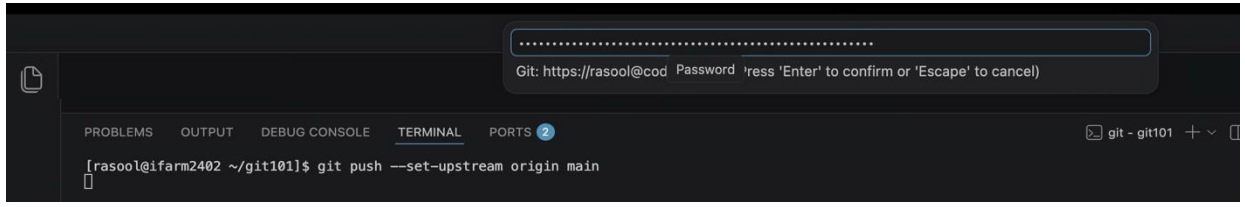
Token added to PATs (Personal access tokens) list

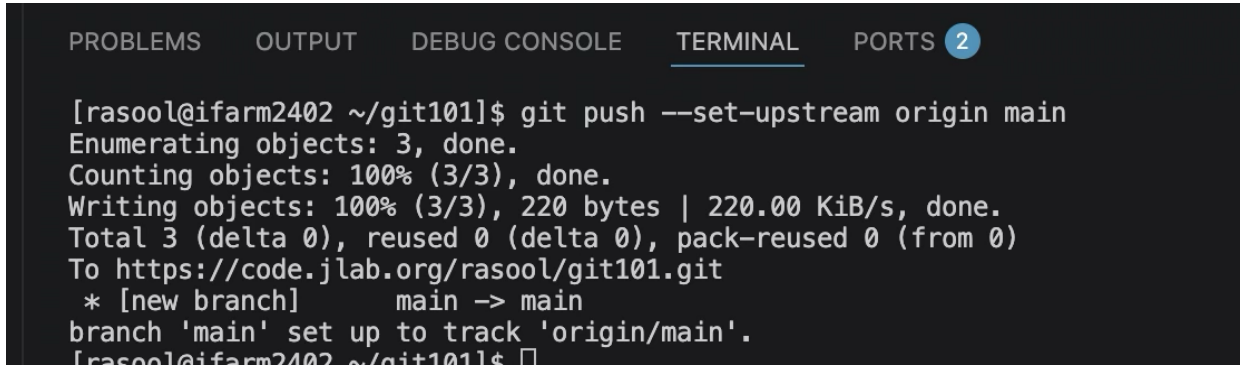
GITLAB – PUSHING FIRST COMMIT

- **Enable credential storage**
  - Run this command so Git stores your login credentials locally:
    - git config --global credential.helper store
- **Push your code**
  - git push --set-upstream origin main
  - Authenticate when prompted
    - Username: your JLab username
    - Password: paste the Personal Access Token you just generated
- **Verify remote**
  - Get your remote repository URL:
    - git remote -v
  - You can Ctrl/Cmd + click the URL to open it in your browser and confirm your first commit has been successfully pushed.









GITLAB – GIT CHEAT SHEET

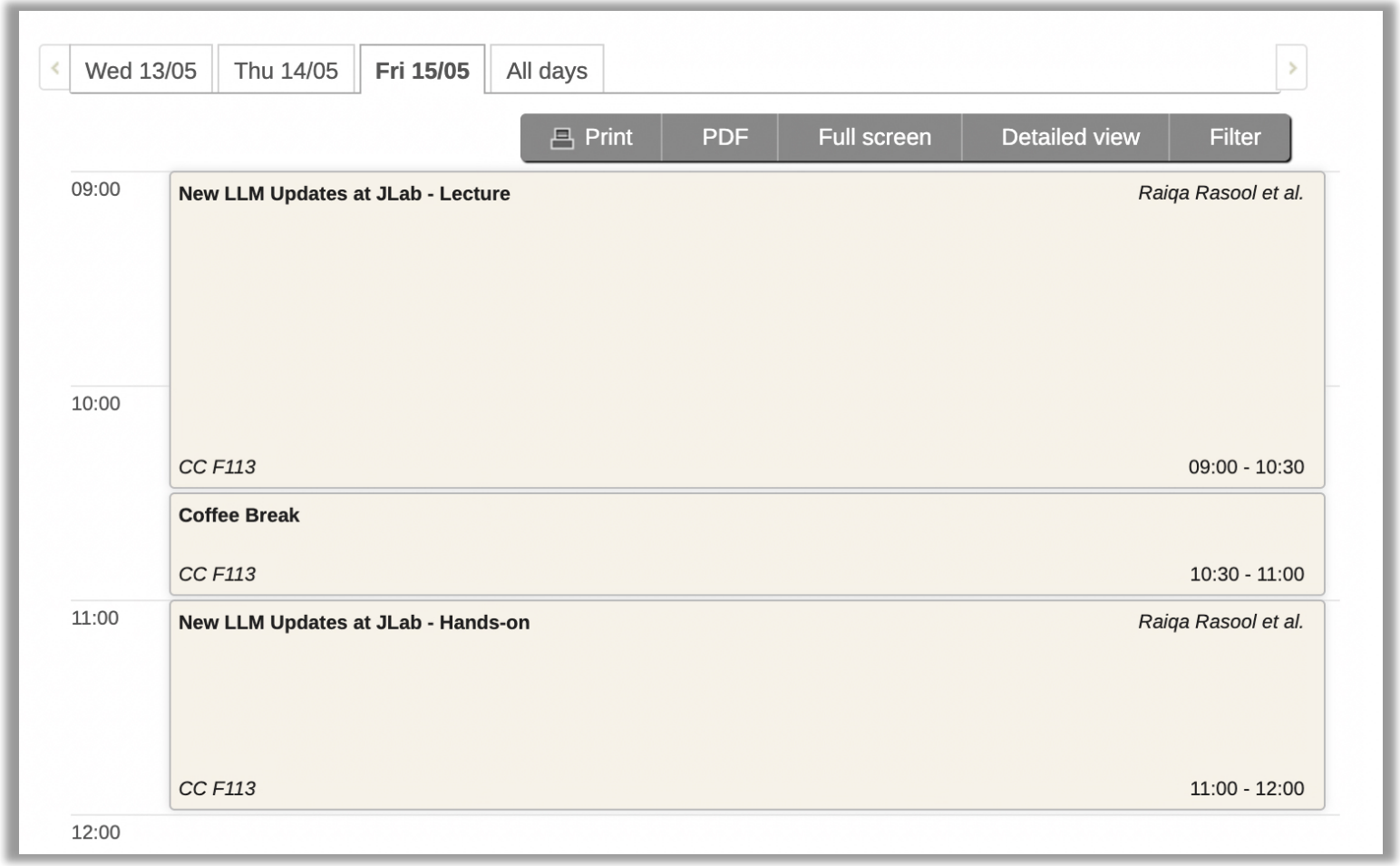
- Refer to the Git cheat sheet and explore additional commands:
  - https://git-scm.com/cheat-sheet

- Some commonly used commands:

```bash
git status          # shows current changes and staging state
git log --oneline   # shows commit history in compact form
git branch          # lists branches in the repository
git remote -v       # shows connected remote repositories
```

WHAT'S COMING NEXT?

- Learn how to use AI agents in VS Code to code—***turning ideas into working apps is now faster than ever.***



- See ***what others are building with AI***—and ***build your own too.***
- Join us on *May 15 for New LLM Updates at JLab* and turn ideas into working products.

https://indico.jlab.org/event/1069/timetable/#20260515