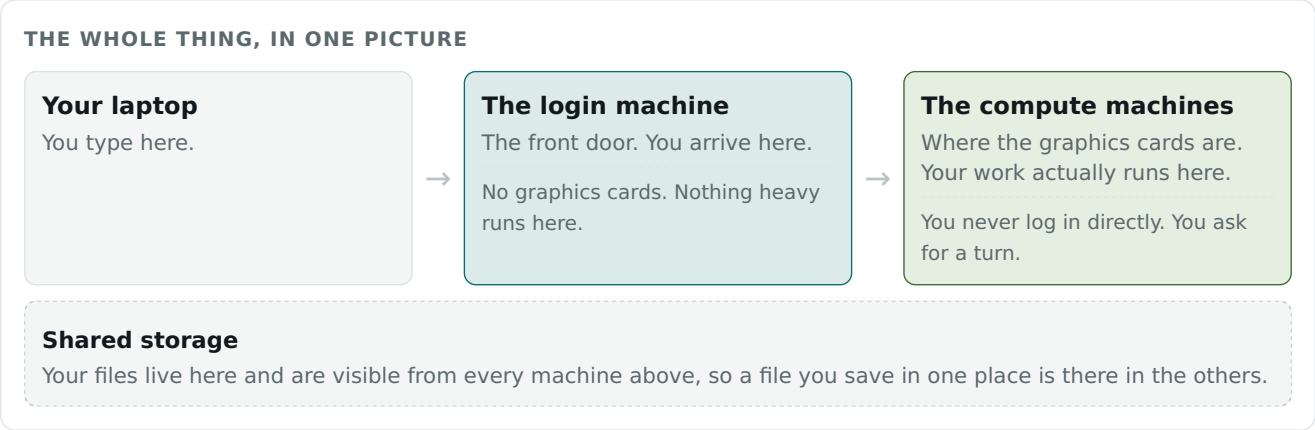
# Using the cluster

VALab AI Resources Management

1 · Getting access
2 · Getting in
3 · Where your data goes
4 · Running something
5 · Installing software
6 · Letting a script talk to the cluster
7 · When your access ends
8 · If something is wrong

Version 0.9.2 · 09 September 2026
https://ai-infra.valab.iti.gr
VALab / Information Technologies Institute, CERTH

This is a shared computer with very fast graphics cards in it, used for training and running AI models. Several people use it at the same time. Everything in this guide exists so that you can get your work done without tripping over anybody else — and so that nothing you build disappears when your access ends.



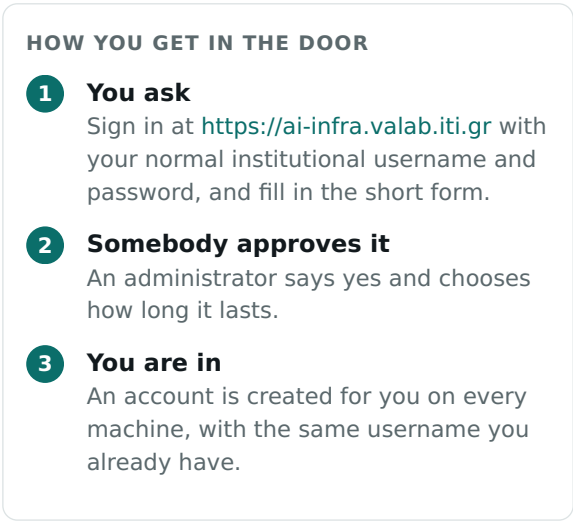

**In plain words:** think of a hotel. The login machine is reception — you check in there, but you do not sleep in the lobby. The compute machines are the rooms, and the scheduler is the receptionist who decides who gets which room and for how long.

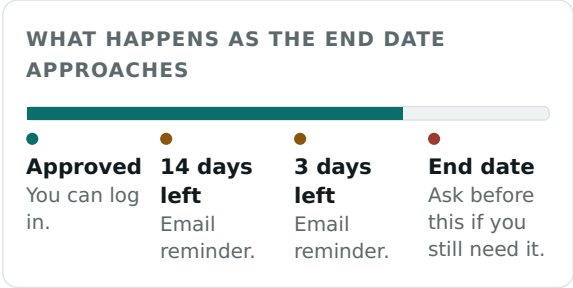
## 1 · Getting access

Having an institutional account is not enough on its own. You have to ask, and somebody has to say yes. It takes three steps.

When you ask, say what you want the cluster for. One sentence is fine. This is not a test — it just helps whoever approves it decide how much and how long.

Your access has an **end date**. Some are deliberately short — a visiting student, one paper. That is completely normal and is not a sign that anyone distrusts you. You will be emailed twice before it runs out, and you can ask for more time from the same page.



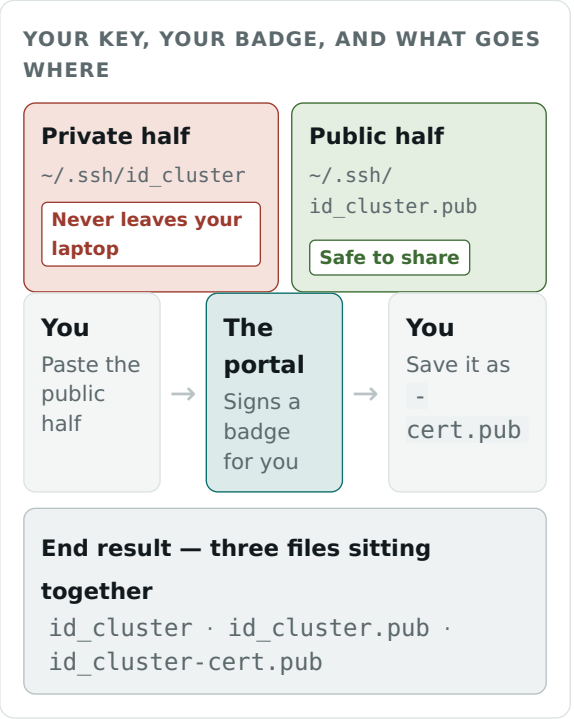



## 2 · Getting in

You do not log in with a password. You log in with a **key** and a **certificate**. That sounds harder than it is, and you only do the setup once.

**In plain words:** a key comes in two halves that belong together. The **private half** is yours and never leaves your laptop — treat it like your house key. The **public half** is safe to hand out; it is more like a padlock than a key, and on its own it opens nothing.

**And the certificate?** It is a visitor badge. You show the portal your public half, and the portal gives you back a badge that says "this person may come in, until this date". The door checks the badge. Nobody has to add you to a list, and nobody has to remember to take you off one.




### Do this once — three steps

**1 Make the key**

On your own laptop. Press Enter at every question it asks — the answers it suggests are the right ones.

```
ssh-keygen -t ed25519 \
  -f ~/.ssh/id_cluster
```

You now have two files. `id_cluster` is the private half — never send this to anybody, ever. `id_cluster.pub` is the public half.

**2 Hand in the public half**

Open `~/.ssh/id_cluster.pub`, copy everything inside it, and paste it into the portal.

Check the name ends in `.pub`. If it does not, you are about to paste your private key into a website — stop, and open the other file.

**3 Save the badge**

The portal hands you a certificate. Save it beside your key, with exactly this name:

```
~/.ssh/id_cluster-cert.pub
```

The `-cert.pub` ending is not decoration. SSH finds the certificate only because of that name. Get it wrong and nothing works.

### Then, every time

```
ssh -i ~/.ssh/id_cluster you@ai-infra-mng.iti.gr
```

**Your badge expires, and that is on purpose.** It lasts at most 90 days, and never longer than your access. When it runs out, come back to the portal and get another one. There is no "please remove my old key" step to forget, because expiry does that job for you.

## 3 · Where your data goes

There are four places to put files. They are not the same, and choosing the wrong one is the single most common way to make the cluster slow — for you and for everybody else. Here they are, worst-to-best-case at a glance.

**`/home/<you>`**

Your own folder. Code, notebooks, settings, Python environments.

| | |
|---|---|
| Size | **Small** |
| Speed | **Fast enough** |
| Backed up | **Yes, daily** |
| Survives your access ending | **Archived** |

**`$TMPDIR`**

A scratch space on the machine your job is running on. For whatever that job is reading right now.

| | |
|---|---|
| Size | **Large** |
| Speed | **Fastest by far** |
| Backed up | **No** |
| Survives your access ending | **Deleted when the job ends** |

**`/projects/<project>`**

Your group's shared folder. Checkpoints, results, anything anybody else might need.

| | |
|---|---|
| Size | **Large** |
| Speed | **Slower** |
| Backed up | **Yes, snapshots** |
| Survives your access ending | **Yes, untouched** |

**`/data` · `/models`**

Shared datasets and model weights that everybody uses. Read-only — you cannot change them.

| | |
|---|---|
| Size | **Very large** |
| Speed | **Slower** |
| Backed up | **No — can be downloaded again** |
| Survives your access ending | **Yes** |

**The one rule worth remembering: if your group needs it, it does not live only in your home folder.** Put results in `/projects` from the very start — not "later, when it is tidy". This is exactly what makes time-limited access harmless: when somebody's access ends, their home folder is archived, and if the work was in `/projects` all along, nothing of value goes with it. Files you create under `/projects` are shared with your group automatically. You do not have to remember to do anything.

## 4 · Running something

Do not run your work on the login machine. It has no graphics cards, and what you can do there is deliberately limited. If you type `nvidia-smi` there it will fail — that is not broken, that is the design.

Instead you ask the **scheduler** for a turn. It keeps a queue, and gives each job the machine and the graphics cards it asked for.

**In plain words:** you do not walk into the kitchen and start cooking. You put in an order, and you are told when a stove is free.

**WHY YOU CANNOT JUST RUN IT WHERE YOU LAND**

**The login machine**

No graphics cards

Edit files. Copy things. Submit jobs. That is it.

× `python train.py` — no.

**A compute machine**

Graphics cards

Reached only by asking the scheduler.

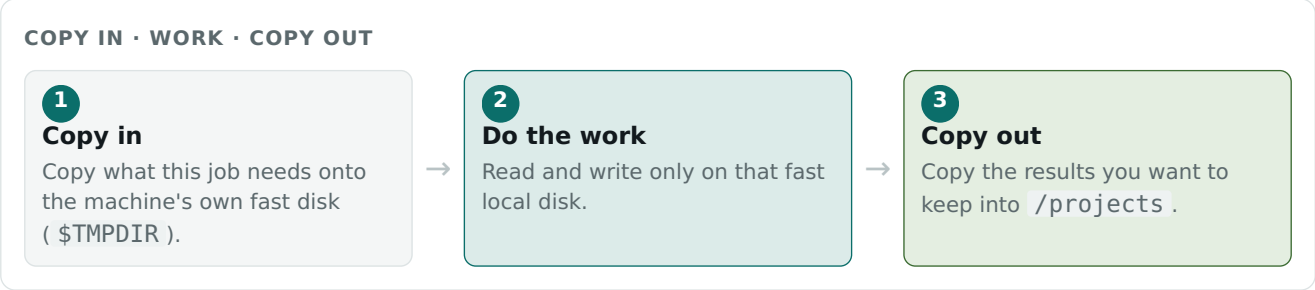
✓ `srun …` or `sbatch …` — yes.

### I just want a shell to poke around in

```
srun --gres=gpu:1 --pty bash -l
```

That puts you on a real compute machine with one graphics card. It behaves like a normal terminal. Type `exit` when you are done, so somebody else can have it.

### The three steps every job should follow




> **This is not busywork.** Reading thousands of small files across the network is slow, and while your job does it, it makes things slow for everybody else too. The same files read from the machine's own disk go at full speed and bother nobody. If your job is much slower than you expected, this is almost always the reason.

### A job you submit and walk away from

Put this in a file called `train.sh`:

```
#!/bin/bash
#SBATCH --job-name=finetune
#SBATCH --partition=gpu
#SBATCH --gres=gpu:1
#SBATCH --cpus-per-task=8
#SBATCH --mem=64G
#SBATCH --time=12:00:00
#SBATCH --output=/projects/myproject/logs/%j.out

# 1. copy in — onto this machine's own fast disk
cp -r /data/mydataset "$TMPDIR/"

# 2. do the work — reading and writing locally
python train.py --data "$TMPDIR/mydataset" --out "$TMPDIR/run"

# 3. copy out — keep only what matters, somewhere it survives
cp -r "$TMPDIR/run" /projects/myproject/runs/
```

Then hand it in, and check on it whenever you like:

```
sbatch train.sh      # hand it in
squeue --me          # what have I got queued or running?
scancel <jobid>      # changed my mind
sacct -j <jobid>     # how did it go, after it finished?
```

### What those `#SBATCH` lines mean

| LINE | WHAT YOU ARE ASKING FOR | WATCH OUT |
|---|---|---|
| `--gres=gpu:1` | One graphics card. | You get exactly that many. The others are invisible to your job. |
| `--cpus-per-task=8` | Eight processor cores. | Loading data needs these. Ask for what you will actually use. |
| `--mem=64G` | Memory. | Go over it and your job is stopped — which is better than taking the whole machine down with it. |
| `--time=12:00:00` | The longest it may run (12 hours here). | Short jobs start sooner. Do not round up out of habit. |
| `--partition=gpu` | Which queue to join. | Use `debug` for quick tests — it turns around faster. |
| `--output=…` | Where the printed output goes. | `%j` becomes the job number, so runs do not overwrite each other. |

## 5 · Installing software

Nobody has administrator rights here, including you — so `sudo apt install` will not work, and you do not need it. There are two ways to get what you want, and between them they cover almost everything.

**Python packages**

Make your own environment in your home folder and install whatever you like into it.

```
python3 -m venv ~/envs/myproject
source ~/envs/myproject/bin/activate
pip install torch
```

Do the `source …` line again each time you log in, or put it in your job script.

**Anything else**

If it needs system-level things installed, run it in a container instead of asking an administrator.

```
apptainer exec --nv \
  docker://pytorch/pytorch:latest \
  python train.py
```

`--nv` is the part that lets the container see the graphics cards. Forget it and they vanish.

## 6 · Letting a script talk to the cluster

Some services run here permanently and can be called over the network. If a script or an application of yours needs to call one, do not put your password anywhere. Get an **API token** from the portal instead.

**In plain words:** a token is a long password made for one machine to use, not for you to remember. It stops working the moment your access ends — so there is nothing to go hunting for in old configuration files afterwards.

## 7 · When your access ends

Nothing dramatic happens, and **nothing is deleted**. Here is exactly what changes.

**WHAT SURVIVES, AND WHAT STOPS**

**Jobs already running**
Left alone to finish. Nothing is killed halfway.

**Your `/projects` files**
Completely untouched. Your group keeps working.

**Your home folder**
Archived — kept safe, but no longer live.

**Logging in**
Stops.

**Submitting new jobs**
Stops.

**Your API tokens**
Stop working at the same moment.

If you still need access, ask for an extension before the end date. The portal will have emailed you twice by then.

## 8 · If something is wrong

Almost everything that goes wrong is one of these five things.

`Permission denied (publickey)`

**Usually means:** your badge has expired, or it is not saved with the right name.

**Do this:** get a new certificate from the portal, and check the file is called `id_cluster-cert.pub` and sits in the same folder as your key.

`Your account has expired`

**Usually means:** your access ran out.

**Do this:** ask for an extension in the portal. Your files are all still there.

`nvidia-smi fails when I log in`

**Usually means:** nothing. It is supposed to. You are on the login machine, which has no graphics cards.

**Do this:** `srun --gres=gpu:1 --pty bash -l` and try again there.

`My job just sits there saying PENDING`

**Usually means:** it is queued, waiting for what it asked for to be free.

**Do this:** `squeue --me --start` guesses when it will begin. The reason column says what it is waiting for. Asking for less time or fewer cards usually starts sooner.

`My job is far slower than I expected`

**Usually means:** it is reading files over the network instead of from the machine's own disk.

**Do this:** copy what it reads into `$TMPDIR` first — steps 1 to 3 above.

Still stuck? Nothing here is a silly question. Ask an administrator, and say what you typed and what came back — that is almost always enough to sort it out straight away.