Before You Start

When you first get access to the cluster, the maintainers need to set up your account such that you have a home directory and correct permissions. You or your mentor should contact the maintainers via email or Glip.

Cluster Overview

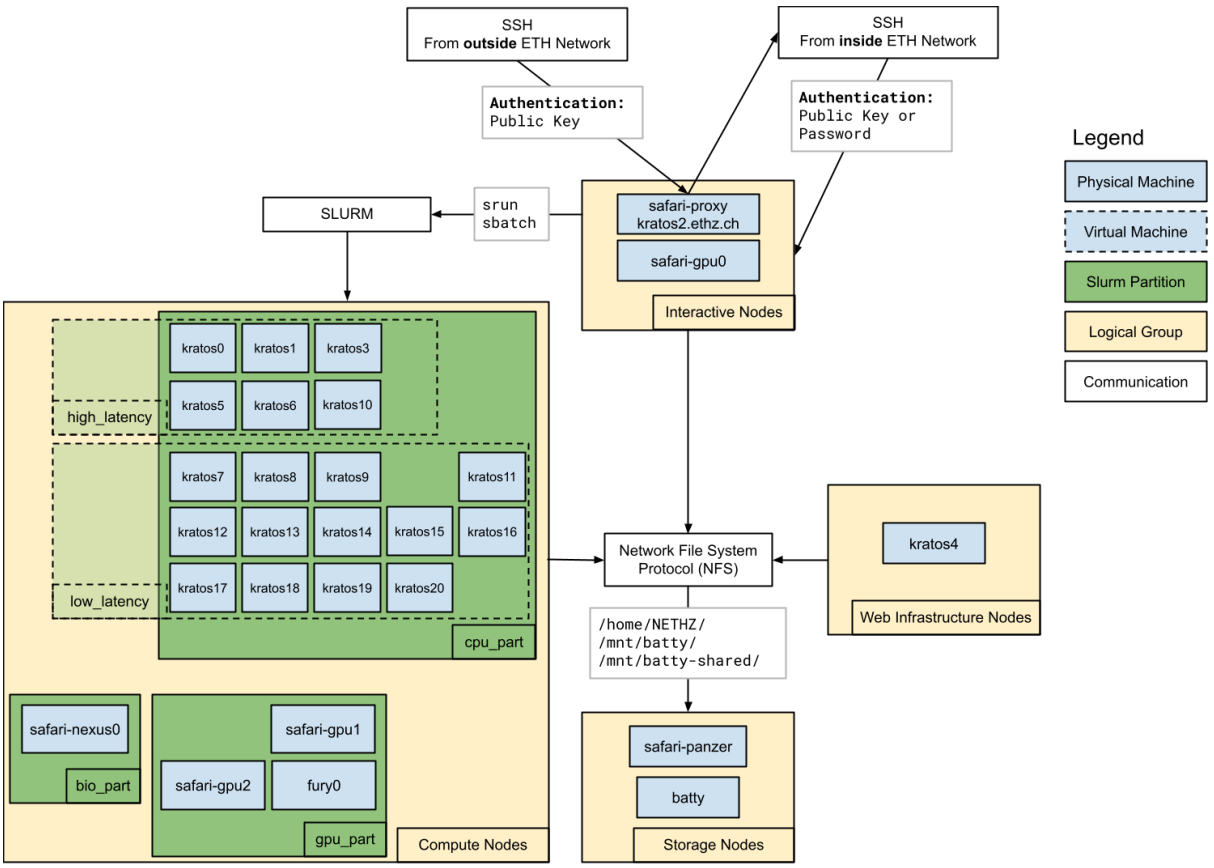
Our cluster consists of a number of physical and virtual machines. They can be logically grouped into sets of *interactive nodes*, *compute nodes*, *storage nodes*, and *web infrastructure nodes*.

Interactive nodes are used to interact with the cluster via SSH. From here, you can interact with the filesystem, monitor running jobs, and prepare new ones. **Do not** run compute-intensive tasks on these machines; they should only be used for light tasks (e.g., small compilation scripts, assembling docker/podman containers, and starting slurm jobs). Currently, we have two login nodes, `safari-proxy.ethz.ch` and `safari-gpu0.ethz.ch` (recommended for testing and debugging GPU programs). You can SSH directly into `safari-proxy.ethz.ch` from outside the ETH network without needing to connect to the VPN to start jobs and/or interact with other nodes in the cluster. You can use the `sinfo` command to list the available compute nodes on each login node.

Compute nodes are larger servers with either Intel Xeon or AMD Epyc processors, i.e., have tens to hundreds of cores each, hundreds of GiB of RAM, and possibly have GPUs attached. These nodes can be accessed by starting SLURM jobs from a login node. **Do not** access these nodes through SSH. Anything you need to do can be accomplished using SLURM and with proper resource reservations. See the SLURM section. Currently we have several nodes with Intel Xeon CPUs (`kratos[0-20].ethz.ch`, `fury0.ethz.ch`, `safari-gpu0.ethz.ch`), two nodes with AMD Epyc CPUs (`kratos10.ethz.ch`, `safari-nexus0.ethz.ch`), and three nodes with GPUs (`fury0.ethz.ch`, `safari-gpu[1-2].ethz.ch`). You can find a detailed list of all servers' properties here: SAFARI Server Properties

Storage nodes provide significant amounts of reliable storage (tens of TiB) on HDDs, with SSD caches for performance. `panzer.ethz.ch` stores all users' home directories, which are mounted to all interactive and compute nodes at `/home/NETHZ/`. `batty.ethz.ch` provides two partitions, mounted as `/mnt/batty/` (e.g., large bioinformatics data, individually per user) and `/mnt/batty-shared/` (e.g., shared files, tools).

Web infrastructure nodes run the public SAFARI website, internal wiki, and course pages. You will only interact with these nodes through the web interface, e.g., by modifying media wiki pages or uploading files to course pages.



Setting up SSH

SSH is a tool that connects a terminal running on your personal machine to a shell (such as bash) on a remote machine over a secured channel. SSH is the primary way users interact with the SAFARI cluster. The next paragraphs will guide you through setting up SSH.

SSH is a secured channel. To establish trust, you provide your identity to the server through either a password or through public key cryptography. Supplying a public key is the preferred option, and the SAFARI cluster accepts password-based authentication only from within the ETH network. The next steps will generate your personal key pair and move them to the cluster for first-time setup.

Generate the Key Pair

On your local machine, open a terminal, and cd into `YOUR_HOME/.ssh/` . If it doesn't exist, create it.

```
~$ cd .ssh
```

Then run `ssh-keygen` as shown below. Replace `KEYNAME` with something human-readable, e.g. `laptop_safari_key`.

```
~/.ssh$ ssh-keygen -t ed25519 -f KEYNAME -N ""
```

This created two files: a public key, and a private key.
`~/.ssh/KEYNAME.pub` is the public key of the pair. We will move this to the cluster.
`~/.ssh/KEYNAME` is the private key, later used by your local SSH client. **Do not share this file with anyone.** In general, it's best if this file never leaves your machine.

Move the Key Pair to the SAFARI Cluster

Next, move the public key you generated to the cluster. **This step can only be executed from within the ETH network by connecting physically or through the ETH VPN.**

Run the following command. Use the same `KEYNAME` as before. `NETHZ` is your username that you use for all ETH services, like email (e.g., `lijoel`).
On Windows:

```
~/.ssh$ ssh NETHZ@safari-proxy.ethz.ch "umask 077; test -d .ssh ||
mkdir .ssh ; cat >> .ssh/authorized_keys || exit 1" < KEYNAME.pub
```

On Linux or MacOS:

```
~/.ssh$ ssh-copy-id -i KEYNAME NETHZ@safari-proxy.ethz.ch
```

The command will ask you to accept `safari-proxy`'s ECDSA fingerprint. Confirm it's the same fingerprint as below, and accept it by typing `yes`:

```
The authenticity of host 'safari-proxy.ethz.ch (129.132.63.163)' can't be established.
ECDSA key fingerprint is SHA256:Q5Xw/gy6yreezgA8eILSGHFD4AlUhyvUN74wpfI2tAY.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'safari-proxy.ethz.ch,129.132.63.163' (ECDSA) to the list of known hosts.
```

You will be prompted for your NETHZ password, then the key is set up.

You can now SSH into `safari-proxy` from inside and as well as outside the ETH network, without entering your password, with the following command. Use the same `KEYNAME` and `NETHZ` as before.

```
$ ssh -i ~/.ssh/KEYNAME NETHZ@safari-proxy.ethz.ch
```

You can return to your local shell by pressing `Ctrl+D`.

Set up the SSH Config

You may find it inconvenient to always type out the path to the key file, your username, and the domain name. Let's set up the SSH config file to avoid this.

Open `~/.ssh/config` with the text editor of your choice, or create it if it doesn't exist.
Add the following lines and save it. Use the same `KEYNAME` and `NETHZ` as before.

```
Host safari-proxy
     HostName safari-proxy.ethz.ch
```

```
    User NETHZ
    IdentityFile ~/.ssh/KEYNAME
```

You can now SSH into `safari-proxy` without specifying all the details:

```
$ ssh safari-proxy
```

You can return to your local shell by pressing `Ctrl+D`.

Accessing Compute Nodes with Slurm

Compute-intensive tasks should be run exclusively on compute nodes. We access these nodes through Slurm. Slurm provisions resources (e.g., “I need 10 CPU cores and a GPU”) for compute jobs (e.g., some Python script) , and then schedules the jobs wherever and as soon as the requested resources are available.

This section will introduce you to a minimal set of commands for interacting with Slurm on our cluster. If you need more information or run into issues, please use Google, the Slurm documentation, ask experienced SAFARI members, or get help at https://safari.ethz.ch/support/.

The srun Command

The most basic slurm command is `srun`. By default, `srun` provisions one CPU core on any kratos machine or waits until one is available. Then it executes the command that it received as a parameter. For example, the following will execute the `hostname` command (which prints the machine’s hostname) on one of the compute nodes:

```
[NETHZ]@safari-proxy:~$ srun hostname
```

You should get an output like “kratos6”, depending on which machine the job was scheduled to. If the terminal hangs, that means all CPUs are currently busy, and Slurm is waiting for one to free up. If so, hit Ctrl+C to cancel the command.

Interactive Shell on Compute Nodes

While users should not directly SSH into compute nodes, it can still be convenient to work from an interactive shell session (e.g., with bash) directly on a compute node. Executing the following command on safari-proxy will open a bash session on a compute node as if one had logged in with SSH (but with an appropriate Slurm resource reservation):

```
[NETHZ]@safari-proxy:~$ slurmsh kratos6
[NETHZ]@kratos6:~$
```

`slurmsh` is our own attempt at roughly reproducing ssh via Slurm. Underneath, it executes something like the following:

```
[NETHZ]@safari-proxy:~$ srun -w kratos6 --pty bash -l
[NETHZ]@kratos6:~$
```

You can exit the opened session by sending an EOF signal by hitting Ctrl+D.

Specifying Resources

By default, srun reserves a single core on any machine. If a workload is multi-threaded, you may want to reserve a larger number of cores (e.g., 48):

```
$ srun -c 48 hostname
```

Slurm ensures that the specified resources are allocated exclusively to your process and that the process is scheduled to a node with sufficient resources, such as a number of cores, memory capacity, storage, etc.

Specifying Partitions

A Slurm cluster consists of a number of partitions. When scheduling a job, you can specify one or multiple partitions where the job may run (only the default partition, if not specified). Our cluster has the following partitions:

- `low_latency`: regular CPU nodes, with a maximum job run time of 3 days
    - jobs that run for longer get killed automatically
- `high_latency`: CPU nodes, without any maximum job run time
- `cpu_part`: the combination of low_latency and high_latency
    - This is the default partition
- `bio_part`: CPU nodes with an exceptionally large amount of memory (>=1TiB)
    - These nodes are **not** reserved for bioinformatics research. The naming stems from the original primary use case for these nodes, but anyone can use them for whatever project they need them
- `gpu_part`: Nodes containing one or more GPUs

The list of partitions and corresponding restrictions is also available with the `sinfo` command on the cluster.

Partitions other than the default can be specified with the -p flag. For example, to avoid long-running jobs getting killed on `cpu_part`, instead run them on `high_latency` with:

```
$ srun -p high_latency my_long_running_simulation
```

The sbatch Command

`sbatch` is the non-blocking analog to `srun`, meaning a job submitted with `sbatch` is held in the background until the requested resources are available, and it will execute in the background until the job is finished. `sbatch` has only a slightly different syntax than `srun`:

```
$ sbatch -c 48 --wrap “hostname”
```

Standard output will be written to a file in the current working directory, named `slurm-[jobid].out`

Using GPUs

GPUs must be reserved by passing the `--gres` parameter to srun or sbatch, and it may appear like the system does not have GPUs installed if none were reserved. GPUs are only available in the `gpu_part` slurm partition, which also must be specified in the reservation. The following are examples of valid reservations for a GPU node:

```
$ srun -p gpu_part --gres gpu:1 --pty bash
$ srun -p gpu_part --gres gpu:A6000:1 --pty bash
$ srun -p gpu_part --gres gpu:A100-40GB:1 --pty bash
$ srun -p gpu_part --gres gpu:A100-80GB:1 --pty bash
```

We set safari-gpu0.ee.ethz.ch (with a Titan V and an RTX 2080Ti) up as a testing environment for GPU users. You may connect to this machine directly via SSH using your NETHZ ID. safari-gpu0 does not receive jobs over Slurm, there is no scheduling done, it is free for everyone to use at any time.

CUDA Runtime

On some nodes, we have multiple CUDA versions installed. It is up to you to choose the right CUDA runtime for your programs. We have added the latest installed CUDA versions to `$PATH` and `$LD_LIBRARY_PATH` by default, but you can use different ones by adding your desired CUDA version to those variables. The examples below may be outdated. It is up to you to navigate to /usr/local/ and check which CUDA versions are installed in the system.

You have three options to access these different CUDA binaries:

1. Call them by their full path, e.g., call `nvcc` as

```
$ /usr/local/cuda-12.2/bin/nvcc
```

2. Add CUDA to your `$PATH`, then call `nvcc` directly. **You will need to run the export command each time you start a new shell.**

```
$ export PATH="/usr/local/cuda-12.2/bin:$PATH"
$ nvcc
```

3. Add CUDA to your `$PATH` in your `.bash_profile`, then call `nvcc` directly. **This is a permanent solution, i.e., you only need to add it to $PATH once.**

```
$ echo 'export PATH="/usr/local/cuda-12.2/bin:$PATH"' >> ~/.bash_profile
$ nvcc
```

Installing Packages

Generally, we do not install new packages globally. I.e., requests of the form “please run the command ‘sudo apt-get install xyz’ for me” are generally not fulfilled. Instead, we provide a number of user-space alternatives for package installation. You can use these freely and on your own, without accidentally interfering with anyone else.

1. **Installing via pip, conda, or similar**
   Many packages have versions available via pip, or as a conda package. These will be kept in your home directory, and thus, conveniently, be available on all machines once installed.
2. **Getting a binary and adding it to your PATH**
   Having an executable “installed” in Linux really means “there is a binary in some directory, and that directory is in the `PATH` environment variable”. Many tools are available online and can be downloaded, possibly compiled, and added to your `PATH`. From there on, you won’t notice these aren’t installed globally. Equally, for a library, it’s typically a .so file and the `LD_LIBRARY_PATH`. You can check out these paths with e.g., `echo $PATH`.
   ChatGPT is excellent at setting up such things, or get help at https://safari.ethz.ch/support/.
3. **Using Podman/Docker**
   Podman and Docker are “container” systems. A container can be thought of as an extremely lightweight virtual machine, within which the user has root permissions. Especially Podman is fully in user space and recommended by us.
   There is a `gpupodman` command that gives you access to the GPUs. You will likely use this when you are in one of the GPU nodes.

Cluster Network Connectivity

The available network connectivity on each node of the cluster depends on the type of IP it has: *public*, *private with NAT*, or *private without NAT*. Here’s a full list of each node’s IP.

- Servers with public IPs (e.g., safari-proxy) or private IPs with NAT (e.g., safari-gpu0) have the best connectivity. Here, outgoing TCP and UDP connections/packets (e.g., HTTP, HTTPS, SSH) work “as expected”.
- Servers with private IPs without NAT (e.g., kratos[0,1,3,5-15]) cannot communicate directly with the internet. They can communicate ETH-internally. The internet can be accessed indirectly via ETH-internal proxies. If a request fails to go through a proxy, you may see some commands fail unexpectedly that worked fine on safari-proxy.

HTTP(S) Proxy

ETH provides proxy.ethz.ch for HTTP(S) requests. We have ensured that this proxy is automatically configured in the user's environment variables HTTP_PROXY and HTTPS_PROXY on compute nodes, thus HTTP(S) requests should work "out of the box" for you.

Sometimes, these environment variables do not get propagated, such as inside Docker containers. To still access the internet (e.g., to download packages with `apt-get`), you can set the environment variables again inside the Docker:

```
$ export HTTP_PROXY=http://proxy.ethz.ch:3128
$ export HTTPS_PROXY=http://proxy.ethz.ch:3128
```

Note that podman automatically sets these variables inside the containers to match the host, so you do not need to do anything.