Getting Started

Note

This page is actively maintained by the Grid'5000 team. If you encounter problems, please report them (see the Support page). Additionally, as it is a wiki page, you are free to make minor corrections yourself if needed. If you would like to suggest a more fundamental change, please contact the Grid'5000 team.

This tutorial will guide you through your **first steps on Grid'5000**. Before proceeding, make sure you have a Grid'5000 account (if not, follow this procedure), and an SSH client.

Contents

Getting support

The **Support page** describes how to get help during your Grid'5000 usage.

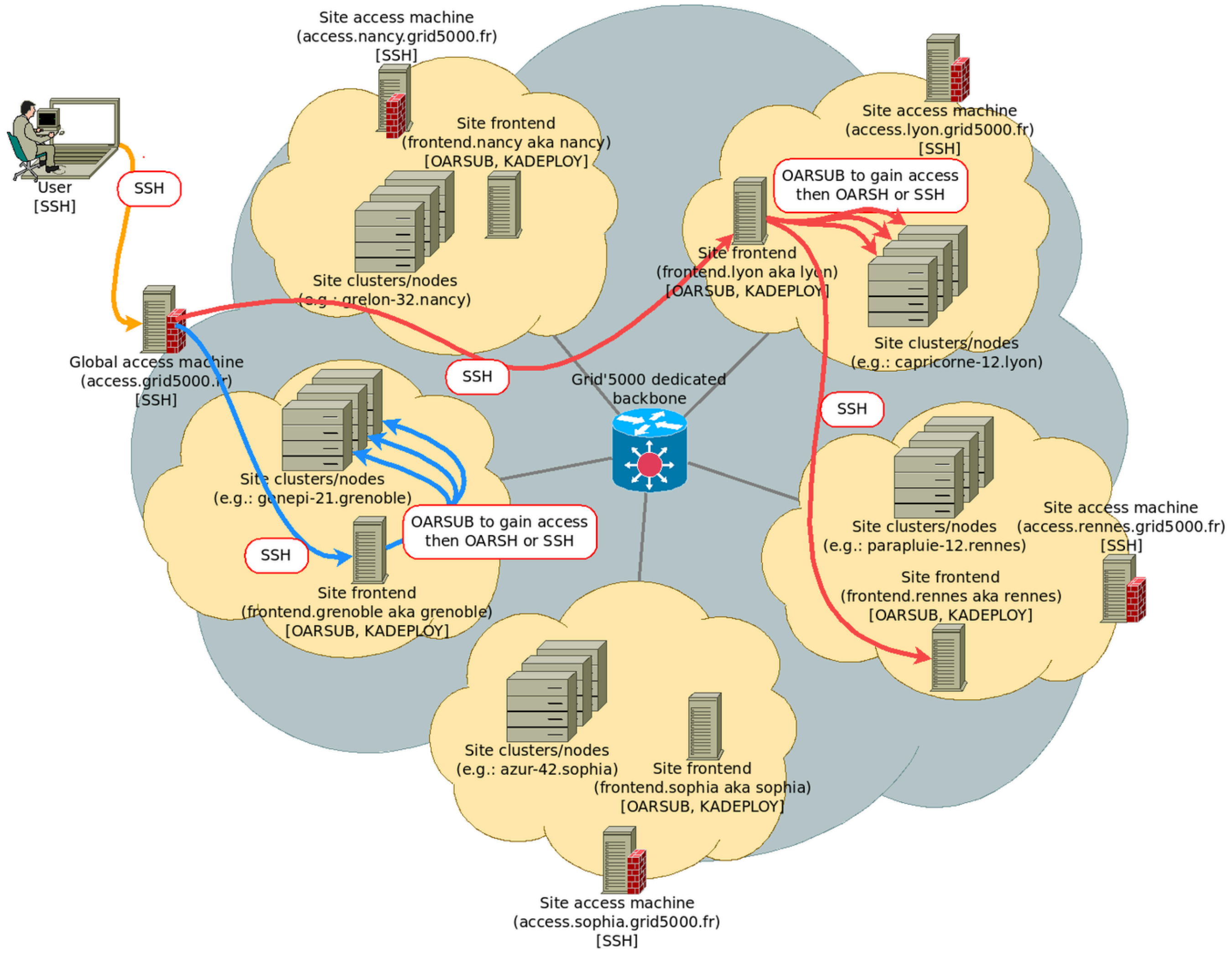
There's also an **FAQ page** with the most common commands.

Connecting for the first time

The primary way to move around Grid'5000 is using SSH. A reference page for SSH is also maintained with advanced configuration options that frequent users will find useful.

As described in the figure below, when using Grid'5000, you will typically:

1. connect, using SSH, to an access machine
2. connect from this access machine to a site frontend
3. on this site frontend, reserve resources (nodes), and connect to those nodes



Grid'5000 for Microsoft Windows Users

Documentation for users using Microsoft Windows on Grid'5000 is available : Grid5000_for_Microsoft_Windows_users

SSH connection through a web interface

If you want an out-of-the-box solution which does not require you to setup SSH, you can connect through a web interface. The interface is available at https://intranet.grid5000.fr/shell/SITE/. For example, to access nancy's site, use: https://intranet.grid5000.fr/shell/nancy/ To connect you will have to type in your credentials twice (first for the HTTP proxy, then for the SSH connection).

This solution is probably suitable to follow this tutorial, but is unlikely to be suitable for real Grid'5000 usage. So you should probably read the next sections about how to setup and use SSH at some point.

Connect to a Grid'5000 access machine

To enter the Grid'5000 network from Internet, one must use an access machine: `access.grid5000.fr` (Note that `access.grid5000.fr` is a round robin alias to either: `access-north` which is currently hosted in Lille, or `access-south` currently hosted in Sophia-Antipolis).

For all connections, you must use the `login` that was provided to you when you created your Grid'5000 account.

```
outside: ssh login@access.grid5000.fr
```

You will get authenticated using the SSH public key you provided in the account creation form. Password authentication is disabled.

Note

You can modify your SSH keys in the account management interface (https://api.grid5000.fr/ui/account)

Connecting to a Grid'5000 site

Grid'5000 is structured in **`sites`** (`Grenoble`, `Rennes`, `Nancy`, ...). Each site hosts one or more clusters (homogeneous sets of machines, usually bought at the same time).

To connect to a particular site, do the following (blue and red arrow labeled SSH in the figure above).

```
access: ssh site
```

Home directories

You have a **different home directory on each Grid'5000 site**, so you will usually use Rsync or `scp` to move data around. On `access` machines, you have direct access to each of those home directories, through NFS mounts (but using that feature to transfer very large volumes of data is inefficient). Typically, to copy a file to your home directory on the Nancy site, you can use:

```
outside: scp myfile.c login@access.grid5000.fr:nancy/targetdirectory/mytargetfile.c
```

Grid'5000 does NOT have a BACKUP service for users' home directories: it is **your responsibility** to save important data in someplace outside Grid'5000 (or at least to copy data to several Grid'5000 sites in order to increase redundancy).

Quotas are applied on home directories -- by default, you get 25 GB per Grid'5000 site. If your usage of Grid'5000 requires more disk space, it is possible to request quota extensions in the account management interface, or to use other storage solutions (see Storage).

Recommended tips and tricks for an efficient use of Grid'5000

Better exploit SSH and related tools

There are also several **recommended tips and tricks for SSH and related tools** (more details in the SSH page).

- Configure SSH aliases using the ProxyCommand option. Using this, you can avoid the two-hops connection (access machine, then frontend) but establish connections directly to frontends. This requires using OpenSSH, which is the SSH software available on all GNU/Linux systems, MacOS, and also recent versions of Microsoft Windows.

Note

Please really take the time to setup the following ssh configuration on the workstation or laptop from where you access Grid'5000 (*outside*). It makes many tasks significantly easier and will save you time if you use Grid'5000 on a regular basis.

```
outside: editor ~/.ssh/config
```

```
Host g5k
  User login
  Hostname access.grid5000.fr
  ForwardAgent no

Host *.g5k
  User login
  ProxyCommand ssh g5k -W "$(basename %h .g5k):%p"
  ForwardAgent no
```

Reminder: `login` is your Grid'5000 username

Warning: the `ProxyCommand` line works if your login shell is `bash`. If not you may have to adapt it. For instance, for the `fish` shell, this line must be: `ProxyCommand ssh g5k -W (basename %h .g5k):%p`.

Once done, you can establish connections to any machine (first of all: frontends) inside Grid'5000 directly, by suffixing `.g5k` to its hostname (instead of first having to connect to an access machine). E.g.:

```
outside: ssh rennes.g5k
```

```
outside: scp a_file lille.g5k:
```

- Use `rsync` instead of `scp` for better performance with multiple files.
- Access your data from your laptop using SSHFS
- Edit files over SSH with your favorite text editor, with e.g.:

```
outside: vim scp://nancy.g5k/my_file.c
```

There are more in this talk from Grid'5000 School 2010 (http://www.loria.fr/~lnussbau/files/g5kss10-grid5000-efficiently.pdf), and this talk more focused on SSH (https://github.com/lnussbaum/slides-lectures/blob/master/ssh/ssh.pdf).

- For a **better bandwidth or latency**, you may also be able to connect directly via the **local access machine of one of the Grid'5000 sites**.

Local accesses use `access.site.grid5000.fr` instead of `access.grid5000.fr`. However, mind that **per-site access restrictions are applied**: see External access for details about local access machines.

VPN

A VPN service is also available, allowing to connect directly to any Grid'5000 machines (bypassing the access machines). See the VPN page for more information.

HTTP reverse proxies

If you only require HTTP/HTTPS access to a node, a reverse HTTP proxy is also available, see the HTTP/HTTPs_access page.

Bash prompt

It is possible to modify your bash prompt to display useful informations related to your current job, such as its jobid, the reserved nodes and the remaining time.

```
fnancy: jdoe@fnancy:~$ oarsub -C 3241912
```

```
grisou-15:
Connect to OAR job 3241912 via the node grisou-15.nancy.grid5000.fr
[OAR] OAR_JOB_ID=3241912
[OAR] Your nodes are:
      grisou-15.nancy.grid5000.fr*16

[jdoe@grisou-15 ~](3241912-->57mn)$ sleep 1m
[jdoe@grisou-15 ~](3241912-->55mn)$
```

You will find here (https://oar.imag.fr/wiki:use_cases_and_user_tips#oar_aware_shell_prompt_for_interactive_jobs) all the information you need to setup such a prompt if you are interested.

Discovering, visualizing and reserving Grid'5000 resources

At this point, you should be connected to a site frontend, as indicated by your shell prompt (`login@fsite:~$`). This machine will be used to reserve and manipulate resources on this site, using the OAR software suite.

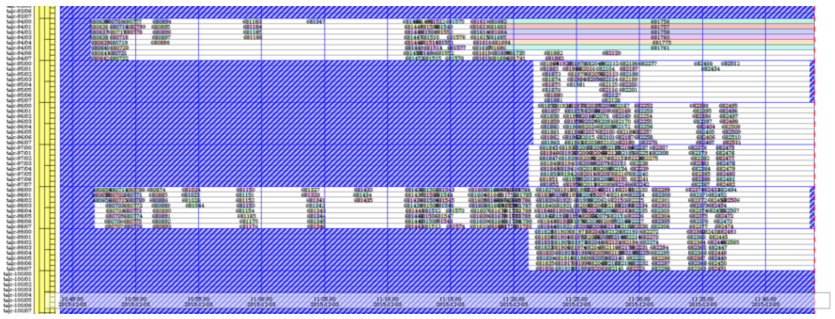
Discovering and visualizing resources

There are several ways to learn about the site's resources and their status:

- The site's MOTD (message of the day) lists all clusters and their features. Additionally, it gives the list of current or future downtimes due to maintenance, which is also available from https://www.grid5000.fr/status/ (https://www.grid5000.fr/status/).
- Site pages on the wiki (e.g. Nancy:Home) contain a detailed description of the site's hardware and network:

Home: Global | Grenoble | Lille | Louvain | Luxembourg | Lyon | Nancy | Nantes | Rennes | Sophia | Strasbourg | Toulouse

- The Status page links to the resource status on each site, with two different visualizations available:
  - Monika, that provides the current status of nodes (see Nancy's current status (https://intranet.grid5000.fr/oar/Nancy/monika.cgi))
  - Gantt, that provides current and planned resources reservations (see Nancy's current status (https://intranet.grid5000.fr/oar/Nancy/drawgantt-svg/); example in the figure below).



- Hardware pages contain a detailed description of the site's hardware

Hardware: Global | Grenoble | Lille | Louvain | Luxembourg | Lyon | Nancy | Nantes | Rennes | Sophia | Strasbourg | Toulouse

Reserving resources with OAR: the basics

Note

OAR is the resources and jobs management system (a.k.a batch manager) used in Grid'5000, just like in traditional HPC centers. **However, settings and rules of OAR that are configured in Grid'5000 slightly differ from traditional batch manager setups in HPC centers, in order to match the requirements for an experimentation testbed**. Please remember to read again **Grid'5000 Usage Policy** to understand the expected usage.

In Grid'5000 the smallest unit of resource managed by OAR is the core (cpu core), but by default a OAR job reserves a host (physical computer including all its cpus and cores, and possibly gpus). Hence, what OAR calls *nodes* are hosts (physical machines). In the `oarsub` resource request (`-l` arguments), *nodes* is an alias for *host*, so both are equivalent. But prefer using *host* for consistency with other arguments and other tools that expose *host* not *nodes*.

Note

Most of this tutorial uses the site of Nancy (with the frontend: `fnancy`), but other sites can be used alternatively.

Interactive usage

To reserve a single host (one node) for one hour, in interactive mode, do:

```
fnancy: oarsub -I
```

As soon as the resource becomes available, you will be directly connected to the reserved resource with an interactive shell, as indicated by the shell prompt, and you can run commands on the node:

```
grisou-1: lscpu
```

Reserving only part of a node

To reserve only one CPU core in interactive mode, run:

```
fnancy: oarsub -l core=1 -I
```

Note

When reserving only a share of the node's cores, you will have a share of the memory with the same ratio as the cores. If you take the whole node, you will have all the memory of the node. If you take half the cores, you will have half the memory, and so on... You cannot reserve a memory size explicitly.

When reserving several CPU cores, there is no guarantee that they will be allocated on a single node. To ensure this, you need to specify that you want a single host:

```
fnancy: oarsub -l host=1/core=8 -I
```

Non-interactive usage (scripts)

You can also simply launch your experiment along with your reservation:

```
fnancy: oarsub -l host=1/core=1 "python my_mono_threaded_script.py --in $HOME/data --out $HOME/results"
```

Your program will be executed as soon as the requested resources are available. As this type of job is not interactive, you will have to check for its termination using the `oarstat` command.

Batch job using OAR scripts

Similarly to what it is the standard use for batch scheduler (e.g. SLURM), a good practice is to use a script that include the OAR directives to define the resource submission. Here is a simple example of such script that select a GPU with specific characteristics.

Properties list can be found on OAR Properties and OAR_Syntax_simplification.

```
#!/bin/bash

#OAR -q production
#OAR -l host=1/gpu=1
#OAR -l walltime=3:00:00
#OAR -p gpu-16GB AND gpu_compute_capability_major>=5
#OAR -O OAR_%jobid%.out
#OAR -E OAR_%jobid%.err

# display some information about attributed resources
hostname
nvidia-smi

# make use of a python torch environment
module load conda
conda activate pytorch_env
python3 -c "import torch; print(torch.cuda.is_available()); print(torch.cuda.get_device_name(0))";
```

The script must be executable

```
frennes: chmod u+x ./my_script_oar.sh
```

and can be called from frontend using

```
frennes: oarsub -S ./my_script_oar.sh
```

and will start when resources will be available.

Other types of resources

To reserve only one GPU (with the associated CPU cores and share of memory) in interactive mode, run:

```
flille: oarsub -l gpu=1 -I
```

Note

Even if the node has several GPUs, this reservation will only be able to access a single one. It's a good practice if you only need one GPU: other users will be able to run jobs on the same node to access the other GPUs. Of course, if you need all GPUs of a node, you have the option to reserve the entire node which includes all its GPUs.

Or in Nancy where GPUs are only available in the production queue:

```
fnancy: oarsub -l gpu=1 -I -q production
```

To reserve several GPUs and ensure they are located in a single node, make sure to specify `host=1`:

```
flille: oarsub -l host=1/gpu=2 -I
```

Tips and tricks

To terminate your reservation and return to the frontend, simply exit this shell by typing `exit` or `CTRL+d`:

```
graffiti-1: exit
```

To avoid unanticipated termination of your jobs in case of errors (terminal closed by mistake, network disconnection), you can either use tools such as tmux (https://tmux.github.io/) or screen, or reserve and connect in 2 steps using the job id associated to your reservation. First, reserve a node, and run a `sleep` command that does nothing for an infinite time:

```
fnancy: oarsub "sleep infinity"
```

Of course, the job will not run for an infinite time: the command will be killed when the job expires.

Then:

```
fnancy: oarsub -C job_id
```

```
grisou-42: hostname && ps -ef | grep sleep

           java -version
           mpirun --version
           module available # List scientific-related software available using module
           whoami

           env | grep OAR # discover environment variables set by OAR
```

Choosing the job duration

Of course, you might want to run a job for a different duration than one hour. The `-l` option allows you to pass a comma-separated list of parameters specifying the needed resources for the job, and `walltime` is a special resource defining the duration of your job:

```
fnancy: oarsub -l host=1/core=2,walltime=0:30 -I
```

The walltime is the expected duration you envision to complete your work. Its format is `[hour:min:sec|hour:min|hour]`. For instance:

- `walltime=5` => 5 hours
- `walltime=1:22` => 1 hour and 22 minutes
- `walltime=0:03:30` => 3 minutes and 30 seconds

Working with more than one node

You will probably want to use more than one node on a given site.

To reserve two hosts (two nodes), in interactive mode, do:

```
fnancy: oarsub -l host=2 -I
```

or equivalently (*nodes* is an alias for *host*):

```
fnancy: oarsub -l nodes=2 -I
```

You will obtain a shell **on the first node of the reservation**. It is up to you to connect to the other nodes and distribute work among them. By default, you can only connect to nodes that are part of your reservation. If you completely own the nodes within one job (or with one job per **complete** node), you will be able to connect those by using `ssh`. In the case of nodes that are not completely owned within a job (if you have reserved only a part of the nodes or by having multiple jobs on nodes) you will have to use `oarsh` connector to go from one node to the other. The connector supports the same options as the classical `ssh` command, so it can be used as a replacement for software expecting ssh.

```
gros-49: uniq $OAR_NODEFILE # list of resources of your reservation
         ssh gros-1 # try to connect a node not in the file (should fail)
         oarsh gros-54 # connect to the other node of your reservation (should work)

         ssh gros-54 # connect to the other node of your reservation (should work)
```

Note

To take advantage of several nodes and distribute work between them, a good option is GNU_Parallel.

`oarsh` is a wrapper around `ssh` that enables the tracking of user jobs inside compute nodes (for example, to enforce the correct sharing of resources when two different jobs share a compute node). If your application does not support choosing a different connector, be sure to reserve nodes entirely (which is the default with `oarsub`) to be able to use `ssh`.

Selecting specific resources

So far, all examples have been letting OAR decide which resource to allocate to a job. It is possible to obtain a finer-grained control of the allocated resources, by using filters.

Selecting nodes from a specific cluster or cluster type

- Reserve nodes from a specific cluster

```
fgrenoble: oarsub -p dahu -l host=2,walltime=2 -I
```

- Reserve nodes in the **production** queue

```
fnancy: oarsub -q production -p grappe -l host=2,walltime=2 -I
```

- Reserve nodes from an **exotic** cluster type

```
flyon: oarsub -t exotic -p pyxis -l host=2,walltime=2 -I
```

Clusters with the **exotic** type either have a non-x86 architecture or are specific enough to warrant this type. Resources with an exotic type are never selected by default by OAR. Using `-t exotic` is required to obtain such resources.

The type of a cluster can be identified on the Hardware pages, see for instance Lyon:Hardware.

Warning

When using the `-t exotic` option, you can still obtain non-exotic resources! You should filter on the cluster name or other properties if you want exclusively exotic resources.

Selecting specific nodes

If you know the exact node you want to reserve, you can specify the hostname of the node you require:

```
fgrenoble: oarsub -p dahu-12 -l host=1,walltime=2 -I
```

If you want several specific nodes, you can use a list:

```
fgrenoble: oarsub -p "host IN (dahu-5, dahu-12)" -l host=2,walltime=2 -I
```

Using OAR properties

The OAR nodes database contains a set of properties for each node, and the `-p` option actually filters based on these properties:

- Nodes with Infiniband FDR interfaces:

```
fnancy: oarsub -p "ib=FDR" -l host=5,walltime=2 -I
```

- Nodes with power sensors and GPUs:

```
flyon: oarsub -p "wattmeter=YES AND gpu_count > 0" -l host=2,walltime=2 -I
```

- Nodes with 2 GPUs:

```
flille: oarsub -p "gpu_count = 2" -l host=3,walltime=2 -I
```

- Nodes with a specific CPU model:

```
flille: oarsub -p "cputype = 'Intel Xeon E5-2630 v4'" -l host=3,walltime=2 -I
```

- Since `-p` accepts SQL, you can write advanced queries:

```
fnancy: oarsub -p "wattmeter=YES AND host NOT IN (graffiti-41, graffiti-42)" -l host=5,walltime=2 -I
```

```
flille: oarsub -p "cputype LIKE 'AMD%'" -l host=3,walltime=2 -I
```

The OAR properties available on each site are listed on the Monika pages linked from Status (example page for Nancy (https://intranet.grid5000.fr/oar/Nancy/monika.cgi)). The full list of OAR properties is available on this page.

Note

Since this is using a SQL syntax, quoting is important! Use double quotes to enclose the whole query, and single quotes to write strings within the query.

Advanced job management topics

Reservations in advance

By default, `oarsub` will give you resources as soon as possible: once submitted, your request enters a queue. This is good for non-interactive work (when you do not care when exactly it will be scheduled), or when you know that the resources are available immediately.

You can also reserve resources at a specific time in the future, typically to perform large reservations over nights and week-ends, with the `-r` parameter:

```
fnancy: oarsub -l host=3,walltime=3 -r '2020-12-23 16:30:00'
```

Note

Remember that **all your resource reservations must comply with the Usage Policy**. You can verify your reservations' compliance with the Policy with `usagepolicycheck -t`.

Job management

To list jobs currently submitted, use the `oarstat` command (use -u option to see only your jobs). A job can be deleted with:

```
fnancy: oardel 12345
```

Extending the duration of a reservation

Provided that the resources are still available after your job, you can extend its duration (walltime) using e.g.:

```
fnancy: oarwalltime 12345 +1:30
```

This will request to add one hour and a half to job 12345.

For more details, see the oarwalltime section of the Advanced OAR tutorial.

Using nodes in the default environment

When you run `oarsub`, you gain access to physical nodes with a default (*standard*) software environment. This is a Debian-based system that is regularly updated by the technical team.

Storage

Home directory

On each node, the home directory is a network filesystem (NFS): data in your home directory is not actually stored on the node itself, it is stored on a storage server managed by the Grid'5000 team. In particular, it means that all reserved nodes share the same home directory, and it is also shared with the site frontend. For example, you can compile or install software in your home, and it will be usable on all your nodes.

Note

The home directory is only shared within a site. Two nodes from different sites will not have access to the same home.

/tmp

The `/tmp/` directory is stored on a local disk of the node. Use this directory if you need to access data locally.

Additional local disks

Some nodes have additional local disks, see Hardware#Storage for a list of available disks for each cluster.

There are two ways to access these local disks:

1. On some clusters, **local disks need to be reserved** to be accessible. See Disk reservation for a list of these clusters and for documentation on the reservation process.
2. On other clusters, **local disks can be used directly**. In this case, jump directly to Using local disks.

In both cases, the disks are simply provided as raw devices, and it is the responsibility of the user to partition them and create a filesystem. Note that there may still be partitions and filesystems present from a previous job.

Other storage options

More storage options are also available.

Getting access to the software you need

There are several options to get access to software :

- Many software packages are already installed and directly accessible: Git, editors, GCC, Python, Pip, Ruby, Java, ...
- Some software (mostly scientific software, such as MatLab) is available through modules. For a list, use `module avail`. Documentation (including how to access license tokens) is available in the Modules page.
- If the software you need is not available through the above options, you can:
  - Install it manually in your home directory
  - Get root access on your node using the sudo-g5k command, and then customize the operating system. The node will be reinstalled at the end of your resource reservation, so that it is in a clean state for the next user. It is thus best to avoid running sudo-g5k in very short jobs, as this has a cost for the platform.
  - Install it using a user-level package manager, such as Guix (especially suitable for HPC software) and Conda (especially suitable for AI software)
  - Install it using container technology, with Docker or Singularity/Apptainer
  - Boot a virtual machine image on the node
  - Re-install the node using a custom image with Kadeploy, as described in the following section
  - Engage in a discussion with the support team to see if the software you need could be added to the software available by default

You might also be interested in documentation about running MPI programs, or using GPUs with CUDA or AMD ROCm / HIP.

Deploying your nodes to get root access and create your own experimental environment

Using `oarsub` gives you access to resources configured in their default (*standard*) environment, with a set of software selected by the Grid'5000 team. You can use such an environment to run Java or MPI programs, boot virtual machines with KVM, or access a collection of scientific-related software. However, you cannot deeply customize the software environment in a way or another.

Most Grid'5000 users use resources in a different, much more powerful way: they use Kadeploy (https://kadeploy.gitlabpages.inria.fr/) to re-install the nodes with their software environment for the duration of their experiment, using Grid'5000 as a *Hardware-as-a-Service* Cloud. This enables them to use a different Debian version, another Linux distribution, or even Windows, and get root access to install the software stack they need.

Note

There is a tool, called `sudo-g5k` (see the sudo-g5k page for details), that provides root access on the *standard* environment. It does not allow deep reconfiguration as Kadeploy does, but could be enough if you just need to install additional software, with e.g. `sudo-g5k apt-get install your-favorite-editor`. The node will be transparently reinstalled using Kadeploy after your reservation. Usage of `sudo-g5k` is logged.

Deploying a system on nodes with Kadeploy

Reserve one node (the `deploy` job type is required to allow deployment with Kadeploy):

```
fnancy: oarsub -I -l host=1,walltime=1:45 -t deploy
```

Start a deployment of the `debian11-min` environment on that node (this takes 5 to 10 minutes):

```
fnancy: kadeploy3 debian11-min
```

By default, all the nodes of the reservation are deployed. Alternatively, you can use `-m` to specify a node (such as `-m gros-42.nancy.grid5000.fr`).

Kadeploy copies your SSH key from `~/.ssh/authorized_keys` to the node's root account after deployment, so that you can connect without password. You may want to use another SSH key with `-k` (such as `-k ~/custom_authorized_keys`).

On Grid'5000 reference environments

Grid'5000 reference environments are named accordingly to the following scheme: `OS version-variant`.

- `OS version` is the OS distribution name and version, for instance `debian11` (Debian 11 "Bullseye", released on 08/2021), `ubuntu2204` (Ubuntu 2204 "Jammy Jellyfish", released on 04/2022), or `centosstream9` (CentOS Stream 9, clone of RHEL, released on 12/2021), or `rocky9` (Rocky Linux 9, released on 07/2022)
- `variant` defines the set of features included in the environment, as follows (for the `x86_64` architecture -- upport might differ on more experimental architectures like `ppc64le` (POWER processors) and `aarch64` (ARM64 processors)) :

Variant	OS available	Installed tools						Network storage	HPC networks support (Infiniband, Omni-Path)	Grid'5000-specific tuning for performance (e.g., TCP buffers for 10 GbE)
		Standard system utilities*	Common utilities**	Advanced packages***	Scientific software available via *module*	*Guix* package manager	*Conda* package manager			
min	Debian 10,11,12,testing	✓	✗	✗	✗	✗	✗	✗	✗	✗
	Ubuntu, CentOS, etc.									
nfs	Debian 10,11,12	✓	✓	✗	partial support****	✓	✓	Support for: - mounting your home and group storage. - using your Grid'5000 user account on nodes.	✓	✓
	Debian testing, Ubuntu, CentOS, etc.		✗		✗	✗	✗		✗	✗
big	Debian 10,11,12	✓	✓	✓	partial support****	✓	✓		✓	✓
default environment without deployment, based on Debian 11		✓	✓	✓	✓	✓	✓		✓	✓

* *Including SSH server and network drivers.*
** *Including among others: Python, Ruby, curl, git, vim, etc.*
*** *Packages for development, system tools, editors and shells.*
**** *Supported modules include Conda and Singularity. Others might work, with no guarantee.*

The list of all supported environments is available by running `kaenv3` on any frontend. Note that environments are versioned: old versions can be listed using the `kaenv3 -l -s` command and a former version retrieved and used by adding the `--env-versionYYYYMMDDHH` option to the `kaenv3` or `kadeploy3` commands (also see the `man` pages). This can be useful to reproduce experiments months or years later, using a previous version of an environment. On some sites, environments exist on different architectures (`x86_64`, `ppc64le` and `aarch64`). The full list can be found in the Advanced Kadeploy page.

The Grid'5000 reference environments are built using the **`kameleon`** tool from recipes detailing the whole construction process, and updated on a regular basis (see versions). See the Environment creation page for details.

Customizing nodes

Now that your nodes are deployed, the next step is usually to copy data (usually using `scp` or `rsync`) and install software.

First, connect to the node as root:

```
fnancy: ssh root@gros-42
```

You can access websites outside Grid'5000 : for example, to fetch the Linux kernel sources:

```
gros-42: wget http://www.kernel.org/pub/linux/kernel/v5.x/linux-5.4.43.tar.xz
```

Warning

Please note that, for legal reasons, your Internet activity from Grid'5000 is logged and monitored.

Let's install `stress` (a simple load generator) on the node from Debian's APT repositories:

```
gros-42: apt-get install stress
```

Installing all the software required for your experiment can be quite cumbersome. Several approaches can be taken to address this:

- Deploy one of the reference environments, and automate the installation of your software environment after it is deployed (one may use a simple bash script, or more advanced tools for configuration management such as Ansible (https://www.ansible.com/), Puppet (http://puppetlabs.com/) or Chef (http://www.opscode.com/chef/)).
- Or build a custom environment including all your requirements, then deploy it ready to use on all nodes. See the **Environment creation** page for more information.

Checking nodes' changes over time

The Grid'5000 team puts on strong focus on ensuring that nodes meet their advertised capabilities. A detailed description of each node is stored in the **Reference API**, and the node is frequently checked against this description in order to detect hardware failures or misconfigurations.

To see the description of grisou-1.nancy.grid5000.fr, use:

```
fnancy: curl https://api.grid5000.fr/stable/sites/nancy/clusters/grisou/nodes/grisou-1.json?pretty
```

Cleaning up after your reservation

At the end of your resources reservation, the infrastructure will automatically reboot the nodes to put them back in the default (*standard*) environment. There's no action needed on your side.

Using efficiently Grid'5000

Until now you have been logging, and submitting jobs manually to Grid'5000. This way of doing is convenient for learning, prototyping, and exploring ideas. But it may quickly become tedious when it comes to performing a set of experiments on a daily basis. In order to be more efficient and user-friendly, Grid'5000 also support more convenient ways of submitting jobs, such as API requests and computational notebooks.

A quick example of grid5000 API usage with python requests

There are many ways to send requests to an API. In this section, we will present two examples of using the API by using the `requests` python package. This python package has been written in order to provide a quick and easy way to submit API requests. Its documentation can be found here : https://docs.python-requests.org/en/latest/ This package is already available in Grid'5000 frontends and nodes' default environment and can be easily installed with `pip install requests`.

Retrieving information from API with python

Here is a simple script that will fetch the names of all clusters of all sites :

```python
import os
import requests

user = input(f"Grid'5000 username (default is {os.getlogin()}): ") or os.getlogin()
password = input("Grid'5000 password (leave blank on frontends): ")
g5k_auth = (user, password) if password else None

sites = requests.get("https://api.grid5000.fr/stable/sites", auth=g5k_auth).json()["items"]

print("Grid'5000 sites:")
for site in sites:

    site_id = site["uid"]
    print(site_id + ":")

    site_clusters = requests.get(
        f"https://api.grid5000.fr/stable/sites/{site_id}/clusters",
        auth=g5k_auth,
    ).json()["items"]

    for cluster in site_clusters:
        print("-", cluster["uid"])
```

This script can be launched as follows:

```
outside: (echo your-username ; echo your-password ) | python3 my-script.py
```

Scripting Job submission with python

By scripting API calls, you can easily control the lifecycle of your jobs. The following script submits a job requesting one *taurus* node in Lyon for echoing a message (it is redirected into a file `api-test-stdout` in your home).

```python
import os
import requests
from time import sleep

user = input(f"Grid'5000 username (default is {os.getlogin()}): ") or os.getlogin()
password = input("Grid'5000 password (leave blank on frontends): ")
g5k_auth = (user, password) if password else None

site_id = "lyon"
cluster = "taurus"

api_job_url = f"https://api.grid5000.fr/stable/sites/{site_id}/jobs"

payload = {
    "resources": "nodes=1",
    "command": 'echo "APIs are awesome !"',
    "stdout": "api-test-stdout",
    "properties": f"cluster='{cluster}'",
    "name": "api-test"
}
job = requests.post(api_job_url, data=payload, auth=g5k_auth).json()
job_id = job["uid"]

print(f"Job submitted ({job_id})")

sleep(60)
state = requests.get(api_job_url+f"/{job_id}", auth=g5k_auth).json()["state"]

if state != "terminated":
    # Deleting the job, because it takes too much time.
    requests.delete(api_job_url+f"/{job_id}", auth=g5k_auth)
    print("Job deleted.")
```

This script can be launched as follow :

```
outside: login=your-username password=your-password python3 my-script.py
```

Warning

Keep in mind that you are sharing clusters with other users. Please take the time to carefully debug your scripts so that you don't reserve more resources than your experiment requires.

Scripting your experiments is a very important step if you seek reproducibility, and efficiency. By scripting API calls, you can automate your whole experiments.

If you are interested in using the Grid'5000 API, a tutorial is available on the **API_tutorial** page.

Another way to use the Grid'5000 API from Python is the **python-grid5000 (https://pypi.org/project/python-grid5000/)** package, or its higher-level counterpart **EnOSlib (https://discovery.gitlabpages.inria.fr/enoslib/)**.

You can also read Experiment scripting tutorial which presents several scripting libraries built on top of the Grid'5000 API.

Notebooks

Grid'5000 also supports Jupyter notebooks and Jupyter lab servers. Jupyter lab servers provide you with a simple web interface to submit jobs on Grid'5000 and run python Notebooks. Using notebooks will allow you to track your experiment evolution during your exploratory phase while scripting part of your process.

You can find more information about Jupyter Lab and python notebooks on the **Notebooks** page.

Going further

In this tutorial, you learned the basics of Grid'5000:

- The general structure of Grid'5000, and how to move between sites
- How to manage you data (one NFS server per site; remember: it is not backed up)
- How to find and reserve resources using OAR and the `oarsub` command
- How to get root access on nodes using Kadeploy and the `kadeploy3` command

You should now be ready to use Grid'5000.

Additional tutorials

There are **many more tutorials available on the Users Home page**. Please have a look at the page to continue learning how to use Grid'5000.

Retrieved from "https://www.grid5000.fr/mediawiki/index.php?title=Getting_Started&oldid=103433"