YOTTA HPC GUIDE

For accessing and using our HPC infrastructure

1. Getting an account

After you sign an agreement, you will have to provide us with your ssh public key.
We use **rsa 2k+** or **ed22519** keys.

You can provide us with multiple ssh key for one or more users. All users from your organization will be attributed to the same account (this way you can use and share the data between you and see the usage and job status).

LINUX users:

If you don't already have your ssh keypair, create it using a command like this in your command line:

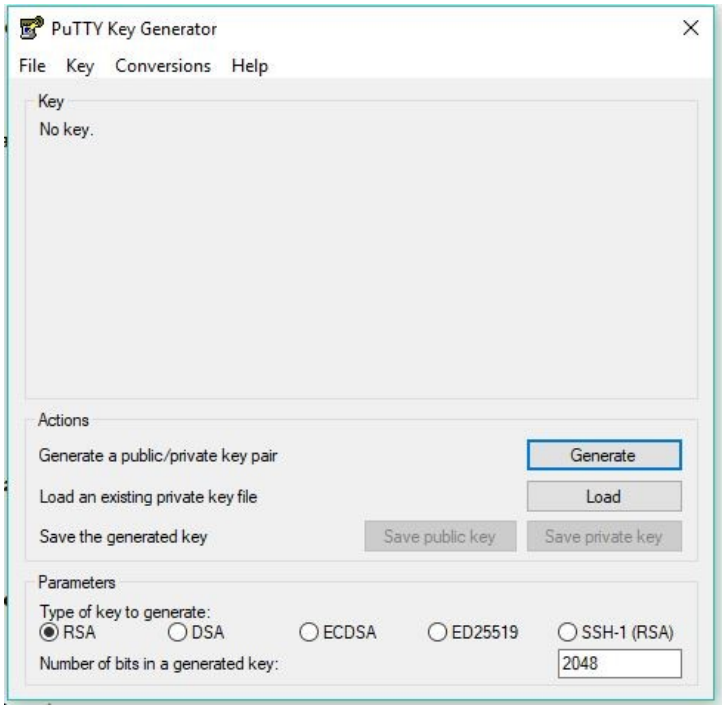
```
$ ssh-keygen -t rsa -b 2048
```

WINDOWS users:

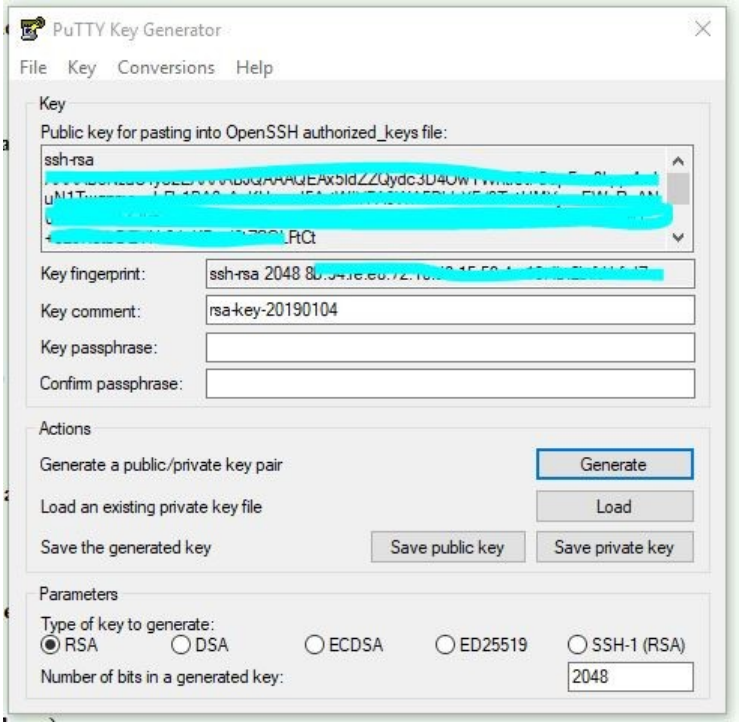
You will need a dedicated application like puttygen to generate the keypair.

Follow this procedure to generate the keypair:

1. Download the puttygen application to your computer (official link provided above);



2. Select the settings as shown in the screenshot below;



3. Click on "Generate" button and move your mouse cursor in the grey area randomly until the green status bar is full and the generated key is revealed

4. Click on the "Save public key" button and save the new file in a location of your choosing.

5. Click on the "Save private key" button and save the new file in a **secure** location of your choosing.

ONCE DONE:

Take great care of your private key, as this is your half of key to access our system. If you lose it, this whole procedure must be repeated.

You will have to send the **PUBLIC** part of your key to the Yotta team at hpc@yac.hr. They will prepare the access to the HPC system. While you will need the private key to be loaded on your system.

Once your account has been established and confirmed, you can proceed and access the system.

2. Logging in

Simply point your ssh client to **hpc-login.yac.hr** with port **8022.**

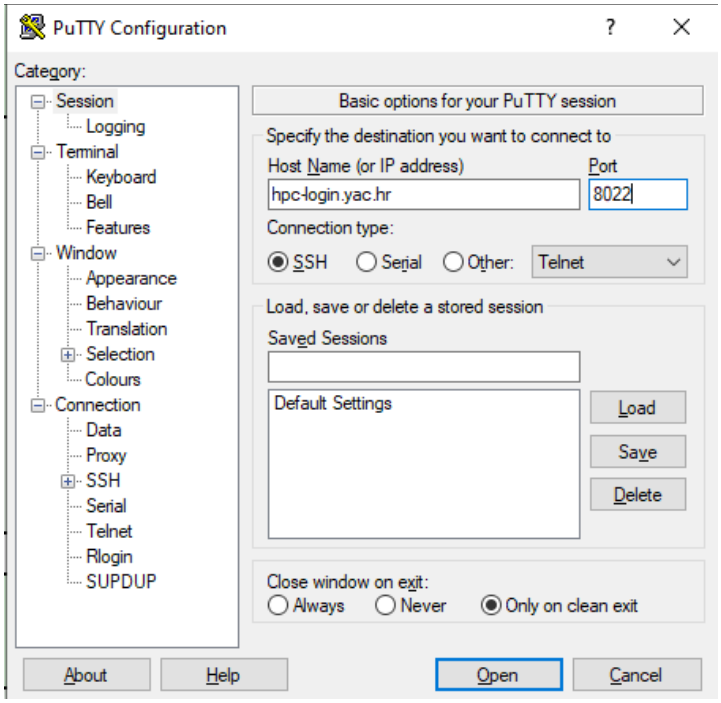
LINUX users:
you can use the following command

```
ssh username@hpc-login.yac.hr -p 8022
```

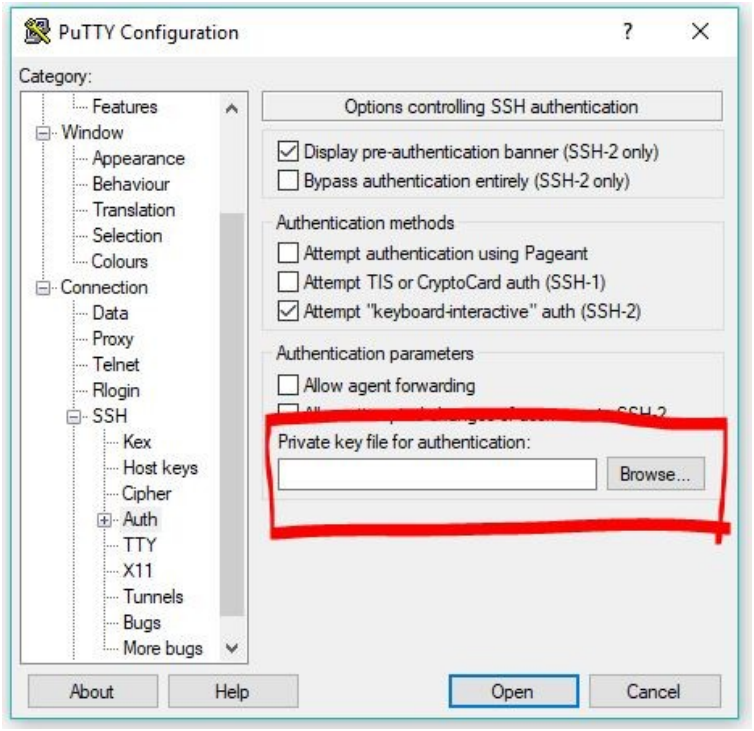
using the username you were given when your user account was created.

WINDOWS users:

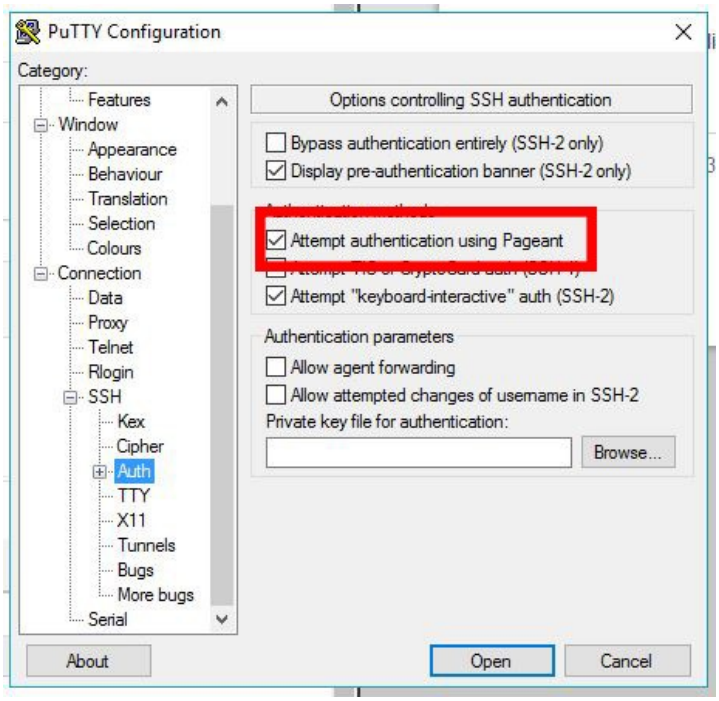
You will have to download a dedicated SSH client application like Putty.



Before establishing the connection make sure that you have uploaded your private SSH key. (e.g. in Putty in the menu on the left side of the window you must select "SSH > Auth" [see screeenshot] and load your key).



While you are using a key management application (like Pageant, also available from the Putty download page), make sure to load the appropriate key in the key management application and then allow key forwarding from this application. (see example of this in Putty in the screenshot below).



3. Transferring files onto the system

If you are an linux user you can use the command:

```
$ scp -P 8022 (-r) file username@hpc-login.yac.hr:
```

which will copy the file to your home folder, or

```
$ sftp -P 8022 username@hpc-login.yac.hr:
```

which will open an interactive ftp session. If you want to place things in subfolders, remmember, the path to your home folder is '/home/users/username'. The command will then look like this:

```
$ scp -P 8022 (-r) file username@hpc-
login.yac.hr:/home/users/username/subfolder
```

Alternatively, if you are more comfortable with a GUI and would like to drag and drop files to our system, you can connect via your file manager. For example, in Nautilus (default in Ubuntu and Fedora) you click on 'Other locations' or 'Browse network' (depending on version) and click 'Connect to server'. In the server address just enter:

```
sftp://hpc-login.yac.hr:8022/home/users/username
```

and click connect.

For Microsoft Windows users we recommend using a client application like WinSCP. Create a new session using the same data and credentials as in you have received for the login (don't forget to have your private SSH key loaded) and login. By default you will be directed to your /home folder.

*Make sure that you have your SSH key available systemwide. Alternatively you can load your key also into WinSCP, by clicking on "Advanced" in the login window and then navigate to "SSH>Authentication". You load your key in the "Authentication parameters" section. (see also the screenshot below)

4 .Getting Started with YOTTA HPC

This overview will guide you through your first steps on the HPC system.

Before proceeding:

- make sure you have an account and an SSH client ready
- ensure you operate from a Linux / Mac environment. Most commands below assumes running in a Terminal in this context. If you're running Windows, you can use Putty and similar tools, yet it's probably better that you familiarize "natively" with a Linux-based environment by having a Linux Virtual Machine (consider VirtualBox for that).

4.1 System overview

Firstly, we will take a quick look how the system is organized and go over the general specifications. Currently, we have 2 different partitions:

- compute
- gpu

The ' **compute**' partition is made out of 14 nodes, with the names node01 to node14. This is the default partition and your jobs will run on it if not specified otherwise.
Each of these nodes have 2 Intel Xeon E5-2690v4 processors, together having 28 cores. Every node also has 512GB of fast DDR4 RAM. They also have 480GB of local SSD storage.

The ' **gpu**' partition consists of 8 nodes (gpu01 to gpu08). The only difference with compute nodes is that they each have 4 NVIDIA Tesla M60 GPUs. The Tesla M60 is a very powerfull GPU as it is made out of two physical NVIDIA Maxwell GPUs with combined 16GB of memory. Many aplications perceive the card as two separate GPUs, essentially having 8 GPUs per node.

Your home folder is located on a shared NFS, which consists of SSD cached disks. The default limits are 100GB per user, but more can be granted upon request.

All the nodes and the filesystem are interconnected with a 2x25GbE connection.

4.2 SLURM basics

Before we dive in into SLURM, it would be good to consult the official quickstart guide:
https://slurm.schedmd.com/quickstart.html
We are going to explain and show how to use SLURM and some of our custom tools.

SLURM is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. It is used on our system.

- It allocates exclusive access to the resources (compute nodes) to users during a job or reservation so that they can perform they work
- It provides a framework for starting, executing and monitoring work
- It arbitrates contention for resources by managing a queue of pending work
- it permits to schedule jobs for users on the cluster resource

Commonly used SLURM commands

sacct is used to report job or job step accounting information about active or completed jobs.

salloc is used to allocate resources for a job in real time. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute srun commands to launch parallel tasks.

srun is used to run a parallel job, it will also first create a resource allocation if necessary.

There are two types of jobs:

- interactive: you get a shell on the first reserve node
- passive: classical batch job where the script passed as argument to sbatch is executed

We will now see the basic commands of Slurm.

Connect to YOTTA HPC. You can request resources in interactive mode like this:

```
$ srun --pty bash
```

You should now be directly connected to the node you reserved with an interactive shell. Keep in mind that only you have access to the node, and it will be billed as you are running a job. Now exit the reservation:

```
$ exit  # or CTRL-D
```

When you run exit, you are disconnected and your reservation is terminated (billing stops).

Currently, there are not time limits enfored on the reservetions or jobs.

To run a passive job, use srun or sbatch.

One example is the following:

```
$ srun -N2 hostname
```

If you use the command 'hostname' (which prints the hostname of the host it is running on) on two nodes, you should see which nodes were allocated to you, this should be a very short job.

Be sure to check out all optional arguments srun can take by typing in **'man srun'** or by looking at the official documentation on https://slurm.schedmd.com/srun.html

Using **srun** like this will give the job output in you terminal session, and you can't really do anything else in that session until the job is done. A better approach for submitting jobs is to use **sbatch**. The command **sbatch** takes a batch script as an argument and submits the job. In the script you specify all options such as the partition you want resources from, the number of nodes and similar.

An example of a simple batch script which runs a command (for example the command 'hostname') on 2 compute nodes is this:

```
#!/bin/bash -l
#SBATCH -account=AccountName
#SBATCH --partition=compute
#SBATCH --nodes=2
#SBATCH --time=00:20:00
#SBATCH -job-name=my_job

srun hostname
```

Using your favorite text editor, save this as myjob.sh and use sbatch to run it:

```
$ sbatch myjob.sh
```

You will get a jobID back from sbatch, which you can use to control your job (we cover this later). Unless specified otherwise, the output sill be stored in a text file in the same folder in which the script is.

Running jobs on different partitions

Just define the partition name in the appropriate place in the job submission script like this:

```
#SBATCH --partition=gpu
```

As shown before, available partitions are: compute and gpu.

Job management

To check the state of the cluster (idle and allocated nodes) run:

```
$ sinfo
```

This is useful to see the state of the resources, and how many are available to you immediately. All the *idle* nodes are ready for use. If you need more nodes than currently available(if some other jobs are running in the system), just submit your job and it will wait in queue until requested resources are available.

You can check the status of your(and only your) running jobs using *squeue* command:

```
$ squeue
```

Then you can stop your job by running the command:

```
$ scancel JOBID
```

You can see your system-level utilization (memory, I/O, energy) of a *running* job using:

```
$ sstat
```

In all remaining examples of reservation in this section, remember to delete the reserved job allocation afterwards (using scancel or CTRL-C)

Pausing, resuming jobs

To stop a waiting job from being scheduled and later to allow it to be scheduled:

```
$ scontrol hold
$ scontrol release
```

To pause a running job and then resume it:

```
$ scontrol suspend
$ scontrol resume
```

Non-root users have only a subset of all SLURM commands available for them to use (most of them will work fine, but display only data for the user who runs it).

4.3 Budgeting

Every time you log in you will be greeted with your resource usage records for the past three months. Additionally, you can check your resource usage for a desired time period with the following command,

Checking the status of used resources

```
$ sausage –s 010121 –e 060121
```

Displays the amount of used resources (per month, per partition and sum total) from the specified date until the current time. You can change the start date (-s) and the end date (-e) as you like, keep in mind that the dates are in a MMDDYY format.

Additional parameters:

-m Displays the amount of used resources in node-minutes

If your organization has enabled budgeting (having a defined number of Accounting Units paid up front), you can use the following command to see how much of the AU you have left:

```
$ sbudget
```

4.4. Support

Please direct all questions and support queries to **hpc@yac.hr**

5. Using environment modules

If you want to use some existing software suites and tools that are installed on our system, you will benefit from modules. Environment Modules is a software package that allows us to provide a multitude of applications and libraries in multiple versions. The tool itself is used to manage environment variables such as PATH, LD_LIBRARY_PATH and MANPATH, enabling the easy loading and unloading of application/library profiles and their dependencies.

Various software environments can be loaded via the environment modules. A lot of software, tools, and stacks are installed, you can list them with this command

```
$ module avail
```

You can search for more detailed descriptions based on a keyword

```
$ module spider
```

Each module can be asked about its description with

```
$ module whatis
```

Modules can be loaded with

```
$ module load
```

You only need to load modules of desired software, it will load all dependencies automatically. For example, if you want to use OpenFOAM compiled with GNU tools:

```
$ module load OpenFOAM/7-foss-2019b
```

6. Example job: Blender rendering

Using our HPC to render Blender images and animations is pretty easy!
Blender has a command line interface that we will use, the full up-to-date documentation is located at:
https://docs.blender.org/manual/en/latest/render/workflows/command_line.html

First, load Blender via the modules command:

```
$ module load Blender
```

Running on compute nodes

The easiest way to run a render is to transfer your .blend file to you home folder and use **srun**:

```
$ srun -N1 blender -b .blend  -o //_###.png -a
```

This command will ask for one node on the **compute** partition and will run **blender** on your .blend file. The output images will be stored in the same folder with the names you define in the **-o** option. The **-a** argument tells Blender that you want to render the whole animation, if you want to render specific frames, use the **-f** option. Keep in mind that the **-a** and **-f** options have to be the last options. A comma separated list of frames can also be used (no spaces) or you can define a range of frames using the '..' seperator between the first and last frames (inclusive). Also make sure to replace .blend with the name of your .blend render file, and with the name you want for the rendered output frames.

Running on gpu nodes

We also have a partition where nodes have a lot of GPU power, which can be beneficial for rendering. First, we need to load the CUDA module:

```
$ module load CUDA
```

Next, in order to run Blender jobs on our GPUs, copy and save the following in a file in the same folder with your .blend file:

```
import bpy
bpy.context.user_preferences.addons['cycles'].preferences.compute_device_type= 'CUDA'
bpy.context.user_preferences.addons['cycles'].preferences.devices[0].use = True

bpy.context.scene.render.tile_x = 256
bpy.context.scene.render.tile_y = 256

bpy.context.scene.cycles.device = 'GPU'
bpy.ops.render.render(write_still=True)
```

Save it as **cuda.py**
This script tells blender to use out NVIDIA Tesla M60 GPUs running CUDA. It is necessary since you probably prepared your render files on your local machine with different hardware.

Run Blender, the command is similar to the one we used before:

```
$ srun -N1 -pgpu blender -b .blend  -o //_###.png -P cuda.py -a
```

If you want to split the rendering to multiple nodes, the best way is to define sets of frames using the **-f** option and run multiple jobs. In that case, you can write a Slurm batch script and run the jobs with **sbatch** in the background.

A simple Slurm batch script that loads all the modules and renders frames 1 to 10 on the GPU nodes is here:

```
#!/bin/bash -l
#SBATCH --partition=gpu
#SBATCH --nodes=1

module load Blender
module load CUDA
```

```
$ srun blender -b .blend -o //_###.png -P cuda.py -f 1..10
```

Again, replace and with the correct names, save the script in the same directory as your .blend and cuda.py files as blender.slurm and run it with:

```
$ sbatch blender.slurm
```