QARNOT

How to use the Qarnot's computing platform

What is Qarnot?

Qarnot is a green cloud computing platform in which you can load your input files from your personal computer and get the output files once the calculation is done on the platform.

Run a computation on the platform

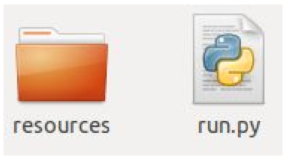
Here is a description of the procedure to run computations on the platform.

1. Log in to the Qarnot account created for the FreeFem Days

- Here is the link to the Qarnot console : https://console.qarnot.com/app/tasks
- The email address to log-in : freefemdays@qarnot.com
- The password will be communicated during the FreeFem Days
- To find your **Token** here : https://account.qarnot.com/compute
- The link to the public GitHub repository

You can also create a personal account with 15€ offered.

2. Create on your personal computer a folder with your input files and the run.py file as depicted hereafter :



Inside the folder "resources" are your input files and the run.py file will be described in the "Examples" section

3. For **Linux** users : Download, install and launch your virtual environment in the same folder as your case study. Check the python version you are using by typing in the terminal : "python --version" and follow those instructions depending on the installed version :

Python 2	Python 3

$ apt-get install python-virtualenv
$ easy install virtualenv
$ pip install virtualenv
	$ apt-get install python3-venv
	$pip3 install virtualenv

Now virtual environment is installed	
$ virtualenv venv	$ python3 -m venv venv
Now virtual environment is created, the next steps are to activate it and install the qarnot library used in the run.py file	
$. venv/bin/activate $ pip install qarnot	$. venv/bin/activate $ pip install qarnot

Here "(venv)" should be written on the left of each line on your command prompt.

For **Windows** and **Mac** users :

If you don't have python follow this link.

Then, install pip (**mac** : "brew install python-pip" and Windows : go to "Installing pip" section) and follow those steps :

Python 2	**Python 3**
$ virtualenv venv	$ python3 -m venv venv
Now virtual environment is created, the next steps are to activate it and install the qarnot library used in the run.py file	
$. venv/bin/activate $ pip install qarnot	$. venv/bin/activate $ pip install qarnot

To install the virtual environment : "pip install virtualenv" or "pip3 install virtualenv"

To create the virtual environment

4. Execute the run.py by first making it executable with the command "chmod +x run.py" (same for mac and Linux systems) and the command "./run.py" that will launch the calculation.

If you don't want to run a run.py script but prefer to interactively launch a computation, you can do it with python or ipython. With ipython the interface is more user friendly.

To install ipython type the following command :

- pip install ipython

Hereafter are some good practices to follow when running tasks, and examples of run.py files to run a calculation. Each line of the run.py file can also be typed in ipython.

Good practices

To avoid confusions between all the cases that will be launched and to clearly identify your task, we recommend the following points :

- Give to your task a name to clearly identify it (e.g. “Name+small description”)

```
task = conn.create_task('Name + small description', 'docker-batch', 1)
```

- Same for your output bucket :

```
output_bucket = conn.create_bucket('Name + small description')
```

Examples

1. Hello World - 1

This first example will display ‘hello world’ on your command line and on the STDOUT (STDard OUTput).

Here you import the Qarnot Python SDK

```
import qarnot
```

This line creates a connection to Qarnot computing, don’t forget to replace xx_mytoken_xx by your own API token available on your account.

```
conn = qarnot.connection.Connection(client_token="xx_mytoken_xx")
```

Here the **task** is created, it is named “helloworld”, uses the “docker-batch” **profile** and is running on 1 **instance**.

```
task = conn.create_task('helloworld', 'docker-batch', 1)
```

Assign the docker application constant DOCKER_CMD. The DOCKER_CMD is the command to be executed in the instance.

```
task.constants['DOCKER_CMD'] = 'echo hello world!'
```

Submit the task and wait for completion

```
task.run()
```

Result : 0> hello world!

2. Hello World -2

This detailed run.py file is quite similar as the one above but this time several **instances** will be used, 4 in this case. It means that 4 **chunks** will be running on 4 **nodes**. The result is ‘hello world from node ...’ with ‘...’ the number of the node that ran the code. It will be displayed in the command prompt and the platform STDOUT.

Here you import the Qarnot Python SDK

```
import qarnot
```

This line creates a connection to Qarnot computing, don’t forget to replace xx_mytoken_xx by your own API token available on your account.

```
conn = qarnot.connection.Connection(client_token="xx_mytoken_xx")
```

Here the **task** is created, it is named “helloworld”, uses the “docker-batch” **profile** and is running on 4 **instances**.

```
task = conn.create_task('helloworld', 'docker-batch', 4)
```

Assign the docker application constant DOCKER_CMD

```
task.constants['DOCKER_CMD'] = 'echo hello world from node ${INSTANCE_ID}!'
```

Submit the task and wait for completion

```
task.run()
```

Result :

```
1> hello world from node 1!
3> hello world from node 3!
0> hello world from node 0!
2> hello world from node 2!
```

3. Fibonacci numbers

In this example we will run the Fibonacci numbers on several instances as we did in the previous example. Here, the difference is that the Fibonacci code will be separated from the run.py file and placed in an **input bucket**. The idea is that the run.py file calls the Fibonacci code placed in this input bucket.

Here you import the Qarnot Python SDK

```
import qarnot
```

This line creates a connection to Qarnot computing, don't forget to replace xx_mytoken_xx by your own API token available on your account.

```
conn = qarnot.connection.Connection(client_token="xx_mytoken_xx")
```

This line will create a bucket and call it "Fibonacci_code"

```
bucket = conn.create_bucket("Fibonacci_code")
```

This line will synchronize the local directory called "resources" to the input bucket

```
bucket.sync_directory("resources")
```

Here the task is created, it is called 'run_Fibo' uses the profile 'python' in which the python program is already installed and will be running on 3 instances.

```
task = conn.create_task('run_Fibo', 'python', 3)
```

This line makes the link between the resources used by the task and the bucket defined previously.

```
task.resources = [ bucket ]
```

Here the 'Fibonacci.py' code loaded in the input bucket called 'Fibonacci_code' will be run. '1,21,32' are the first numbers of the Fibonacci numbers. On node with INSTANCE_ID=0 will be calculated the Fibonacci numbers beginning with 1, on node INSTANCE_ID=1 will be calculated the Fibonacci numbers beginning with 21 and on node INSTANCE_ID=2 will be calculated the Fibonacci numbers beginning with 32. Make sure to get the same number of instances as the numbers to begin Fibonacci (for example, 3 instances are set and 3 numbers to start Fibonacci numbers (1, 21 and 32))

```
task.constants['PYTHON_SCRIPT'] = 'Fibonacci.py 1,21,32 ${INSTANCE_ID}'
```

Submit the task and wait for completion

```
task.run()
```

Print the STDOUT on your terminal

```
print(task.stdout())
```

Results :

```
2> On node  2  the 10 firsts Fibonacci numbers beginning with  32  are :  [32, 63, 95, 158, 253, 411, 664, 1075, 1739, 2814]
1> On node  1  the 10 firsts Fibonacci numbers beginning with  21  are :  [21, 41, 62, 103, 165, 268, 433, 701, 1134, 1835]
0> On node  0  the 10 firsts Fibonacci numbers beginning with  1  are :  [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

4. Text analysis

The aim of the example is to count the number of lines, words and characters in a text file. 3 instances will be running, the first numbered “0” will count the number of lines, the second numbered “1” will count the number of words and the third one numbered “2” will count the number of characters.

Here you import the Qarnot Python SDK

```
import qarnot
```

This line creates a connection to Qarnot computing, don’t forget to replace xx_mytoken_xx by your own API token available on your account.

```
conn = qarnot.connection.Connection(client_token="xx_mytoken_xx")
```

This line will create a bucket and call it “demo_text_input_bucket”

```
input_bucket = conn.create_bucket('demo_text_input_bucket')
```

This line will add the file ‘script’ to the bucket called ‘input_bucket’, this script is in the same folder as the run.py, in opposition to the 3rd example where the script “Fibonacci.py” is in another folder.

```
input_bucket.add_file('script')
```

Same as above for the ‘text.txt’ file

```
input_bucket.add_file('text.txt')
```

This line create the output bucket and call it 'demo_text_output_bucket'

```
output_bucket = conn.create_bucket('demo_text_output_bucket')
```

Here the task is created, named ‘text_analysis’, running on the profile ‘docker-batch’ with 3 instances.

```
task = conn.create_task('text_analysis', 'docker-batch', 3)
```

This line add the input bucket to the resources used in the task.

```
task.resources.append(input_bucket)
```

This line add the results in the output bucket defined earlier

```
task.results = output_bucket
```

This command run the script called “script”

```
task.constants['DOCKER_CMD'] = './script'
```

Submit the task and wait for completion

```
task.run()
```

Print the STDOUT on your terminal

```
print(task.stdout())
```

Results :

```
1> Node 1: nb words = 110
0> Node 0: nb lines = 5
2> Node 2: nb characters = 752
```

5. FreeFem++

Here you import the Qarnot Python SDK

```
import qarnot
```

This line creates a connection to Qarnot computing, don't forget to replace xx_mytoken_xx by your own API token available on your account.

```
conn = qarnot.connection.Connection(client_token="xx_mytoken_xx")
```

This line will create a bucket and call it "freefem_input"

```
input_bucket = conn.create_bucket('freefem_input')
```

This line adds the contents of your local directory "freefem_resources" to the bucket you've created on the previous line.

```
input_bucket.sync_directory("freefem_resources")
```

Here you've created the **output bucket**. This storage unit will be filled with the **outputs** of your calculation once the calculation is done

```
output_bucket = conn.create_bucket("freefem-output")
```

The task is created.

```
task = conn.create_task("freefem", "docker-batch-freefemdays", 1)
```

This line adds the input bucket to the resources used by the task.

```
task.resources.append(input_bucket)
```

The results of the computation will be stored in the output bucket

```
task.results = output_bucket
```

Those three lines define constants that will be used for the calculation. Here DOCKER_HOST is for the host name inside the container, DOCKER_REPO for the repository where will be downloaded the docker images, and DOCKER_TAG for the version of the image to be downloaded.

```
task.constants["DOCKER_HOST"] = "localhost"
task.constants["DOCKER_REPO"] = "qarnotlab/freefem"
task.constants["DOCKER_TAG"] = "latest"
```

This line is the command you ask the image to run. In this case, you call the function "ff-mpirun" to run FreeFem++ on 4 cores defined by "-n 4" and the input file is called "navierstokes.edp".

```
task.constants["DOCKER_CMD"] = "/usr/freefem/bin/ff-mpirun -n 1
./navierstokes.edp"
```

Submit the task and wait for completion.

```
task.run()
```

NB. If you want to produce a *.ffglut file to improve the visualization, just make the following change :

```
task.constants["DOCKER_CMD"] = "/usr/freefem/bin/ff-mpirun -n 1
./navierstokes.edp"
```

to :

```
task.constants["DOCKER_CMD"] = "/usr/freefem/bin/ff-mpirun -n 1 ./navierstokes.edp
-fglut output.ffglut"
```

Then, to visualize it:

- if you have freefem on your laptop, type :

```
ffglut output.ffglut
```

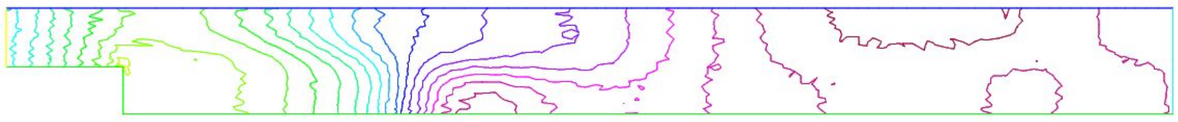
- if you don't have freefem installed but if you have docker :

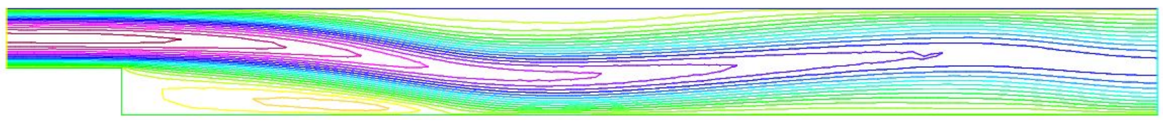
```
docker pull pierrestraebler/freefem
```

```
docker run -e DISPLAY -v $HOME/.Xauthority:/root/.Xauthority -v "$PWD":/data
pierrestraebler/freefem ffglut /data/output.ffglut
```

- If neither freefem nor docker are installed :
  https://doc.freefem.org/introduction/installation.html

Results :





Glossary

- Task : A fragmentable batch computation that contains all the necessary informations to run a simulation on Qarnot's computing platform. It is mounted as follow :
    - task = conn.create_task('**Task_name**', '**profile**', **nb_Instances**)
    - With '**Task_name**' the name you give to your task to quickly identify it on the platform, '**profile**' the profile you want to use to run your simulation and '**nb_Instances**' the number of nodes that will run your simulation.

- Instance : Number of chunks running in the meantime for one task on one CPU.

- Node : A computer (mono CPU) that contains several physical cores (4, 8, 16 or 32)

- Chunk : A portion of a task that will run on a single node. If you chose nb_Instance to 3, 3 chunks will be running independently.

- Profile : It describes the execution environment that will be loaded on every node involved in the task.
    - The main profile is *docker-batch*. On *docker-batch* an insulated docker image is running. Then, you need to call the docker image you want to use in the run.py .
    - Some profiles already contains applications like *python* or *blender* where each node run an optimized python or blender container.

- Bucket : A storage unit for input or output data. It is created as follow :
    - Creation of the input bucket :
      bucket = conn.create_bucket("**Input_bucket_name**")
      Which is linked to you local directory called "benchmark :
      bucket.sync_directory("**benchmark**")
      And loaded as the resources needed to run your case :
      task.resources = [**bucket**]
    - Creation of the output bucket :
      task.results = conn.create_bucket("**Output_bucket_name**")