

Integrate your own workflow with job automation APIs

OAsis offers job automation APIs that allow you to enqueue and inspect jobs easily. This means you can seamlessly integrate HPC into your workflow engine as a component. For example, you can use it to finetune machine learning models with new data or set up simulation jobs using your own tooling.

This article will use [the Molecular Dynamics Simulation case](#) as an example, but the concept should generally apply to all types of workloads.

For the simulation of the ApoA1 protein in a water box and the study of energy changes, we will be using NAMD. NAMD is available in various distributions, with OAsis offering it as both the Lmod system and the container version packaged by NVIDIA NGC. We will use the container version in this article.

Run the simulation using quick job

We can run container-based jobs easily in the web portal. Access Jobs > Run container and fill in the following:

Enqueue job

Path: /pfss/toolkit/run_container.sbatch

Run a container for your workload.

Node*

Partition: gpu

Node: oaklbhpcgpu001

CPU: 4 cores

Memory: 8 GB

GPU: 1g.10gb x 1

Select another

Container*

/pfss/containers/ngc.namd.3.0-beta2.sif

Command*

```
cd ~/apoa1  
namd3 +p1 +devices 0 +setcpuaffinity apoa1.namd
```

Output path*

container.out

Export job script

Enqueue now

Download this file

Toggle file content

Enqueue the job, and you should see the job output (log file) on the following screen:

Enqueued job file

Job Information

Job ID: 164336

Job State: RUNNING

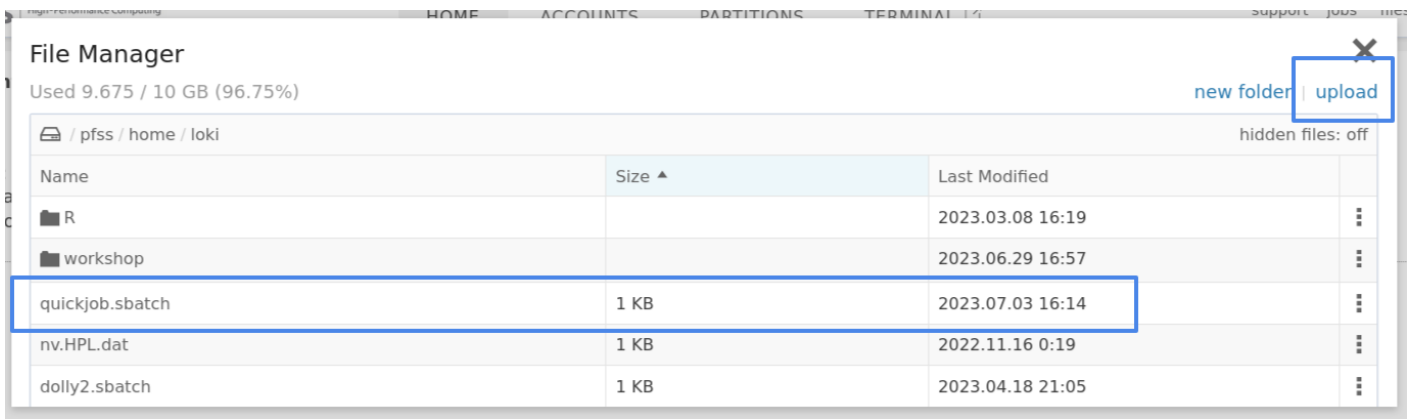
Check the job information by following link

View running jobs

View log file

Please wait until the job is completed. Once that's done, repeat step 1, but click on the "Export job script" button this time. This will allow you to download a **quickjob.sbatch** file. This file is a standard SLURM job script that can be enqueued using the **sbatch** command.

Upload it back to the cluster via the file browser to finish the setup.



Run the job programmatically using the hc CLI client

Ensure you are using the latest hc client before proceeding. This article is tested with hc version 0.0.196.

We are going to use two APIs: **enqueue-job** and **job-status**. You may inspect their usage using the **--help** argument.

```
> hc enqueue-job --help
hc-enqueue-job
Enqueue a job

USAGE:
  hc enqueue-job [OPTIONS] --path <PATH>

OPTIONS:
  -h, --help
      Print help information

  -o, --output-format <OUTPUT_FORMAT>
      set output format

  -p, --path <PATH>
      File path in the cluster. e.g., /pfss/toolkit/mpitest.sbatch
> hc job-status --help
hc-job-status
Show specific job status

USAGE:
  hc job-status [OPTIONS] --job-id <JOB_ID>

OPTIONS:
  -h, --help
      Print help information

  -j, --job-id <JOB_ID>
      ID of the job

  -o, --output-format <OUTPUT_FORMAT>
      set output format
```

Let's use the two sub-commands to enqueue a new job and then inspect it's status.

```
$ hc enqueue-job --path=/pfss/home/loki/quickjob.sbatch
```

```
Submitted batch job 164337
```

```
$ hc job-status -j 164337
```

```
JobState: completed
```

Run the job programmatically using Python

Below is a Python sample code demonstrating how to call the Oasis RESTful API, specifically the "me", "enqueue-job", and "job-status" API functions.

```
import requests
import time
import hashlib
import hmac
import json

# hide InsecureRequestWarning message, you may delete these two lines
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

api_key = "<API-KEY>" # replace your api key
api_key_secret = "<API-KEY-SECRET>" # replace your api key secret
baseUrl = "https://hpccenter.hk"

def getSignature(api_key_secret, path, ts):
    mac = hmac.new(api_key_secret.encode(), digestmod=hashlib.sha256)
    mac.update(f"{path}|{ts}".encode())
    return mac.hexdigest()

def getHeaders(api_key, api_key_secret, path):
    ts = int(time.time())
    headers = {
        "req-at": str(ts),
        "req-api-key": api_key,
        "req-signature": getSignature(api_key_secret, path, ts),
    }
    return headers

def getResult(baseUrl, path, method = "get", payload={}):
```

```

url = baseUrl + path
if method == "get":
    response = requests.get(url, headers=getHeaders(api_key, api_key_secret, path),
verify=False)
else:
    response = requests.post(url, json=payload, headers=getHeaders(api_key,
api_key_secret, path), verify=False)

if response.status_code == requests.codes.ok:
    return response.json()
else:
    print("Error:", response.status_code, response.text)
    raise Exception("api call error")

# get login user information
path = "/api/me"
result = getResult(baseUrl, path)
print(result)

# get job status
jobId = "<JOB-ID>" # replace your job id
path = "/api/job-status?job-id=" + str(jobId)
result = getResult(baseUrl, path)
print(result)

# enqueue job
jobFilePath = "<JOB-FILE-PATH>" # replace your job file location, e.g.
/pfss/home/<USERNAME>/<FILENAME>.sbatch
payload = {
    "path": jobFilePath
}
path = "/api/enqueue-job"
result = getResult(baseUrl, path, "post", payload)
print(result)

```

Make sure to replace `"<API-KEY>","<API-KEY-SECRET>"` with your actual API key obtained from the OAsis platform. Also replace `"<JOB-ID>","<JOB-FILE-PATH>"` with the actual data.

This code covers the basic functionality of retrieving user information, enqueueing a job with specified parameters, and checking the status of a job using the RESTful API endpoints provided by OAsis.

Revision #23

Created 3 July 2023 07:38:20 by Loki Ng

Updated 4 July 2023 09:01:30 by Milo Cheung