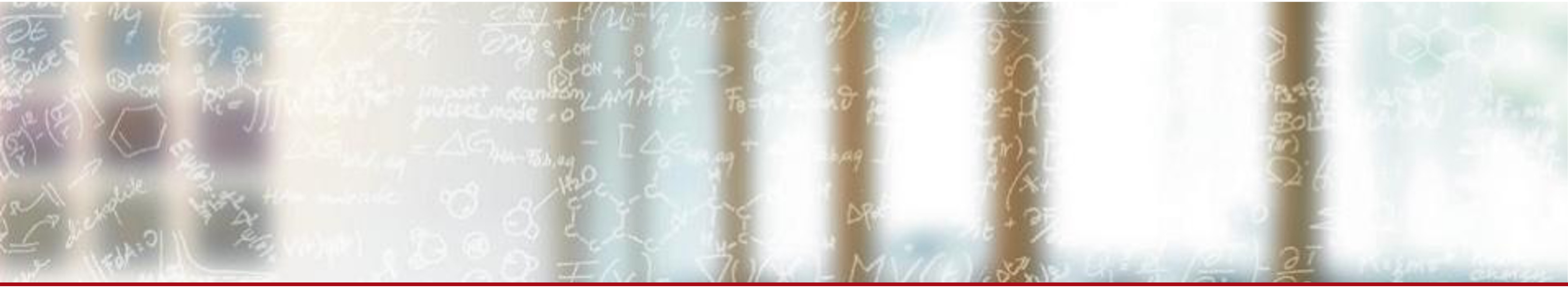
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Enhancing HPC Service Management on Alps using FirecREST API

J.P. Dorsch, A. Fink, E. Koutsaniti, R. Sarmiento
CSCS - Swiss National Supercomputing Centre
ETH-Zürich

May 5-9, 2024
CUG 2024
Perth, WA, Australia

Motivation

Motivation

- As HPC evolves there is an increasing need from the user community on creating and accessing sophisticated services on HPC
- Use cases such as CI/CD Pipelines, Workflow Orchestrators, Interactive Computing, Web Portals, and Regression Testing are just few examples of those requirements
- These needs create a challenge on the HPC infrastructure in terms of scaling the support for such diverse number of services
- Using RESTful API technology interfacing HPC resources (like FirecREST API) can facilitate the integration, support, and maintenance of complex services for HPC infrastructure

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

Introducing FirecREST

FirecREST in a nutshell

- **FirecREST is an open-source web-enabled API to HPC resources developed by CSCS**

FIRECREST

FirecREST in a nutshell

- **FirecREST is an open-source web-enabled API to HPC resources developed by CSCS**

- Presents standard programming interface
  - Based on RESTAPI concept
  - Independent of programming language (HTTP)
  - Translates web requests into HPC business logic
  - Parses back HPC results into web-friendly format

FIRECREST

FirecREST in a nutshell

- **FirecREST is an open-source web-enabled API to HPC resources developed by CSCS**

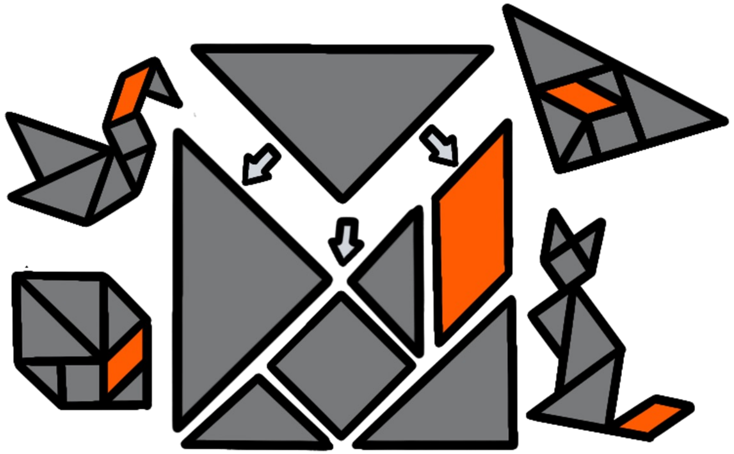
- Presents standard programming interface

- Provides web interface for classic HPC
  - Creation of web applications over HPC
  - Enables support for multiple devices

FIRECREST

FirecREST in a nutshell

- **FirecREST is an open-source web-enabled API to HPC resources developed by CSCS**
- Presents standard programming interface
- Provides web interface for classic HPC
- Allows modular design to support different workflows and HPC systems
    - Abstracts HPC resources into components and objects



FirecREST features

- **FirecREST is an open-source web-enabled API to HPC resources developed by CSCS**

- Presents standard programming interface

- Provides web interface for classic HPC

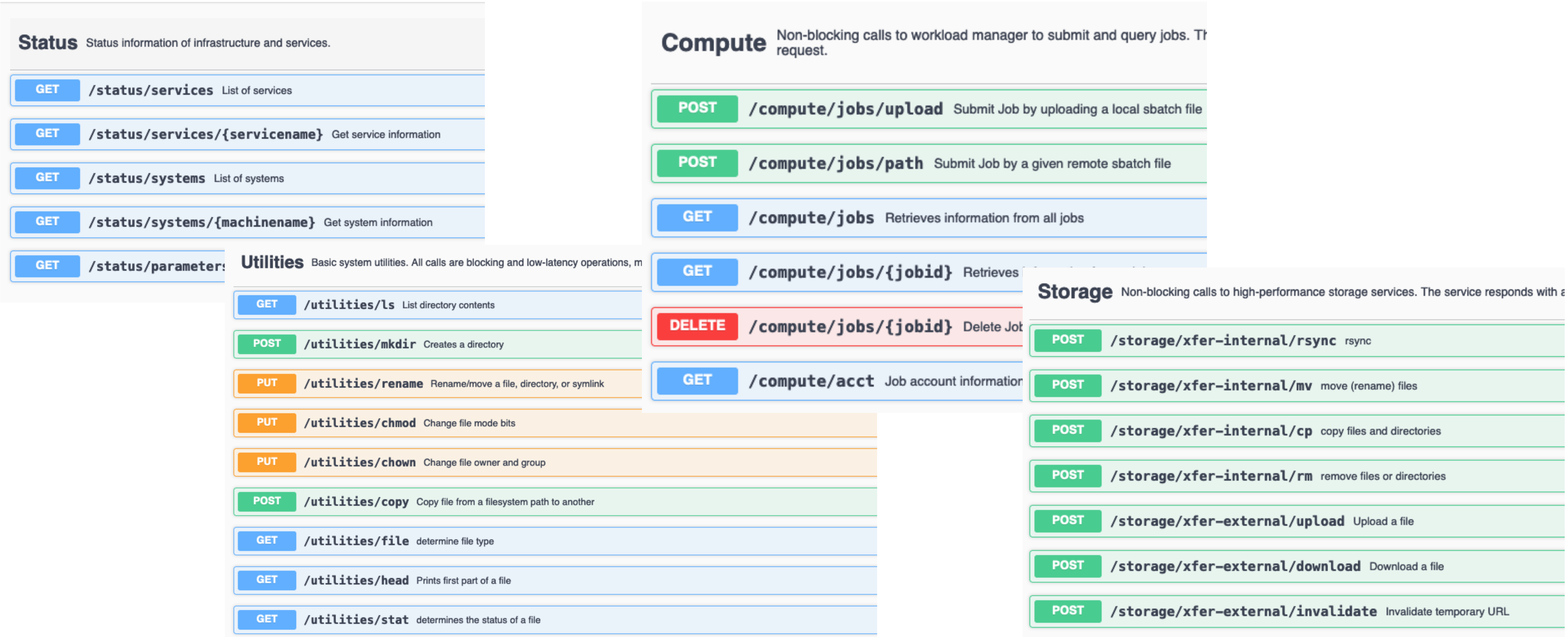
- Allows modular design to support different workflows and HPC systems

- Integrates with authentication and authorization layers
    - Relies on standard IAM solutions for authentication

FIRECREST

The FirecREST API

- OpenAPI documentation: https://firecrest-api.cscs.ch



Status Status information of infrastructure and services.

GET `/status/services` List of services

GET `/status/services/{servicename}` Get service information

GET `/status/systems` List of systems

GET `/status/systems/{machinename}` Get system information

GET `/status/parameter`

Compute Non-blocking calls to workload manager to submit and query jobs. Tr
request.

POST `/compute/jobs/upload` Submit Job by uploading a local sbatch file

POST `/compute/jobs/path` Submit Job by a given remote sbatch file

GET `/compute/jobs` Retrieves information from all jobs

GET `/compute/jobs/{jobid}` Retrieves

DELETE `/compute/jobs/{jobid}` Delete Jol

GET `/compute/acct` Job account informatior

Utilities Basic system utilities. All calls are blocking and low-latency operations, m

GET `/utilities/ls` List directory contents

POST `/utilities/mkdir` Creates a directory

PUT `/utilities/rename` Rename/move a file, directory, or symlink

PUT `/utilities/chmod` Change file mode bits

PUT `/utilities/chown` Change file owner and group

POST `/utilities/copy` Copy file from a filesystem path to another

GET `/utilities/file` determine file type

GET `/utilities/head` Prints first part of a file

GET `/utilities/stat` determines the status of a file

Storage Non-blocking calls to high-performance storage services. The service responds with a

POST `/storage/xfer-internal/rsync` rsync

POST `/storage/xfer-internal/mv` move (rename) files

POST `/storage/xfer-internal/cp` copy files and directories

POST `/storage/xfer-internal/rm` remove files or directories

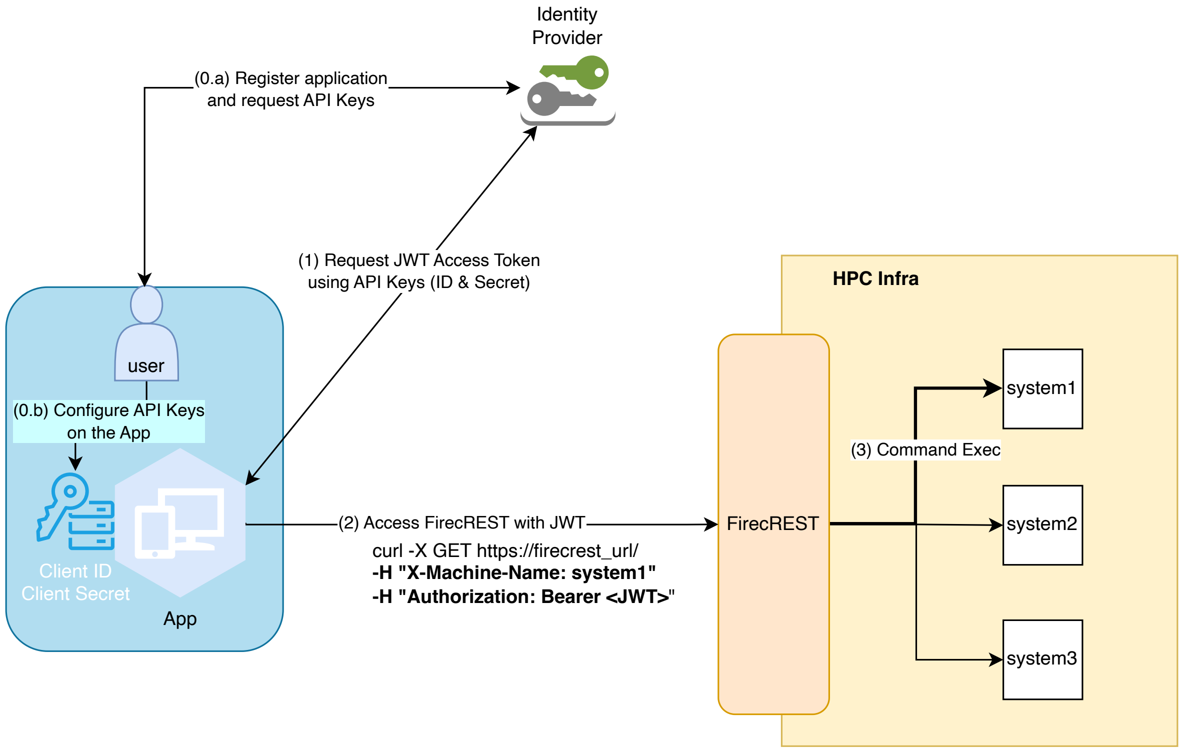
POST `/storage/xfer-external/upload` Upload a file

POST `/storage/xfer-external/download` Download a file

POST `/storage/xfer-external/invalidate` Invalidate temporary URL

FirecREST IAM layer

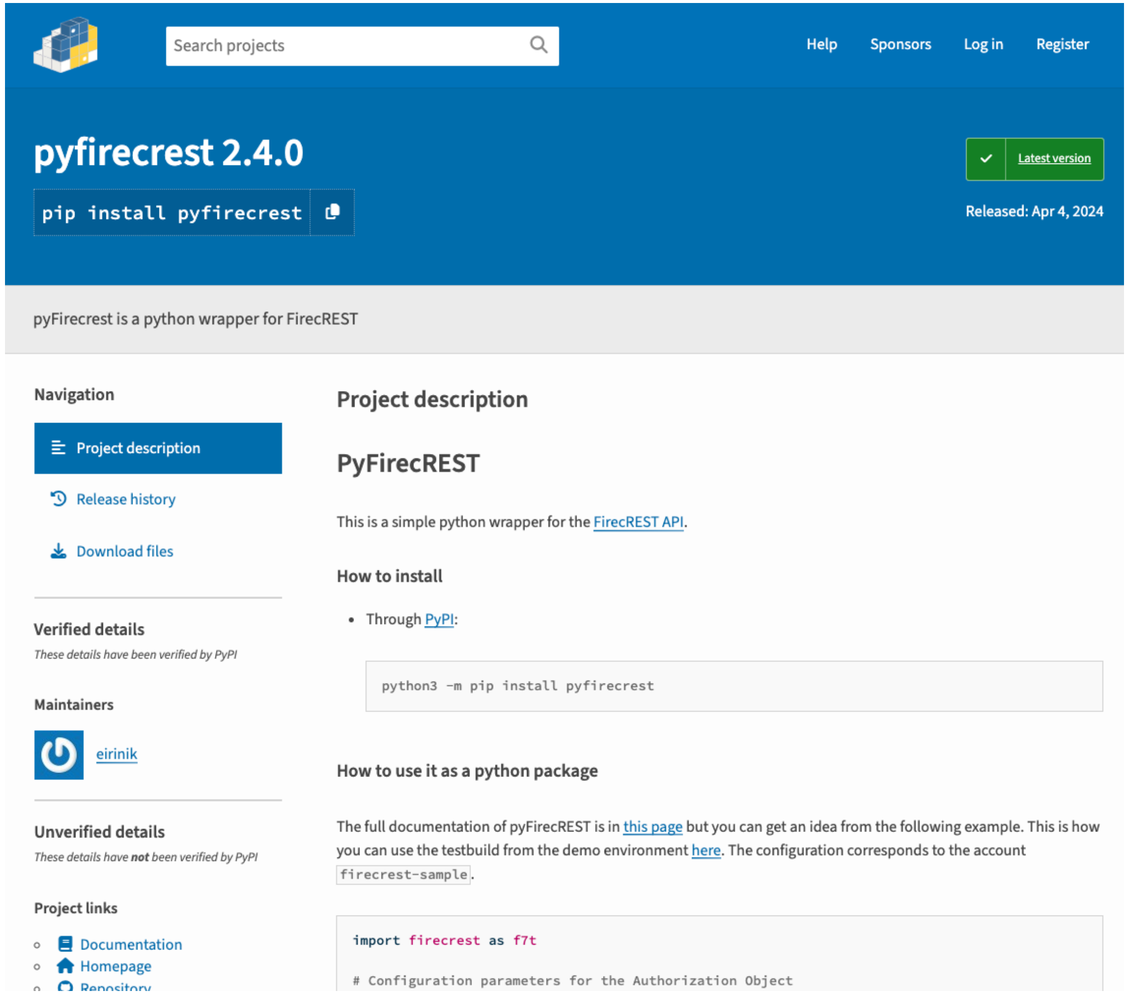
- IAM relies on JWT from an IdP supporting Open ID Connect (OIDC)/OAuth2 standard
- FirecREST users (or clients) need to register their applications on the IdP
- A key pair is obtained and used to obtain JWT to access FirecREST
- Client ID and Secret can be used as secrets in an application for fetching JWT access token automatically, enabling robot-to-API communication


Identity Provider
(0.a) Register application and request API Keys
(1) Request JWT Access Token using API Keys (ID & Secret)
user
(0.b) Configure API Keys on the App
Client ID
Client Secret
App
(2) Access FirecREST with JWT
HPC Infra
FirecREST
(3) Command Exec
system1
system2
system3

```
curl -X GET https://firecrest_url/
-H "X-Machine-Name: system1"
-H "Authorization: Bearer <JWT>"
```

pyFirecREST Library

- pyFirecREST is a Python library that simplify the usage of the FirecREST for scripting
- Includes transparent integration with OIDC/OAuth2 for JWT Access Token
- Enhances response time using AsyncIO interface (Async pyFirecREST)
- **Facilitates integration with several tools that exposes APIs or SDK via Python or scripting languages**



pyfirecrest 2.4.0

```
pip install pyfirecrest
```

Released: Apr 4, 2024

pyFirecrest is a python wrapper for FirecREST

Verified details

These details have been verified by PyPI

Maintainers

eirinik

Unverified details

*These details have **not** been verified by PyPI*

Project links

Project description

PyFirecREST

This is a simple python wrapper for the FirecREST API.

How to install

- Through PyPI:

```
python3 -m pip install pyfirecrest
```

How to use it as a python package

The full documentation of pyFirecREST is in this page but you can get an idea from the following example. This is how you can use the testbuild from the demo environment here. The configuration corresponds to the account `firecrest-sample`.

```
import firecrest as f7t

# Configuration parameters for the Authorization Object
```

Use Cases

Use Cases

- Continuous Integration (CI) Pipelines
    - CI pipelines are used to facilitate testing and integration of scientific software releases across programming environments and hardware systems
    - Challenges to setup a CI Pipeline in HPC are mostly related to SSH connection
        - Access with valid credentials
        - Cloning source code repository in target machine's node
        - Keep alive the connection during pipeline execution
        - Providing constant output from commands
    - With the help of FirecREST users and sysadmins can
        - Use the same approach for different technologies (GitLab CI, GitHub Actions, Jenkins CI, etc)
        - Thanks to the abstraction layer, test the software for different architectures and software stack
        - Solve authentication and connectivity issues

Use Cases

- Continuous Integration (CI) Pipelines
  - `ci/ci_script.py`

Importing pyFirecREST

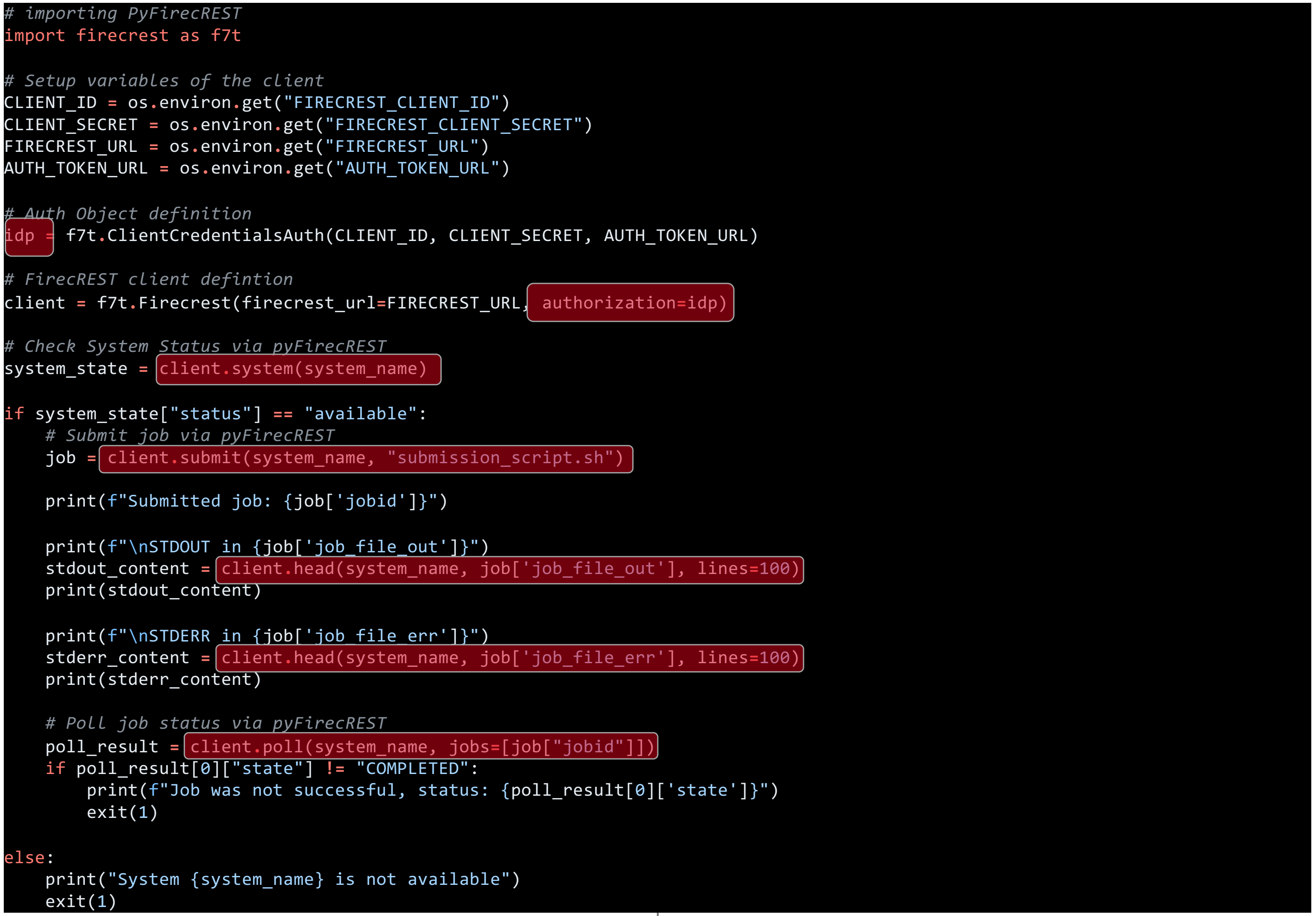
FirecREST credentials from environment

Creating FirecREST object

System check and job submission

Printing job output

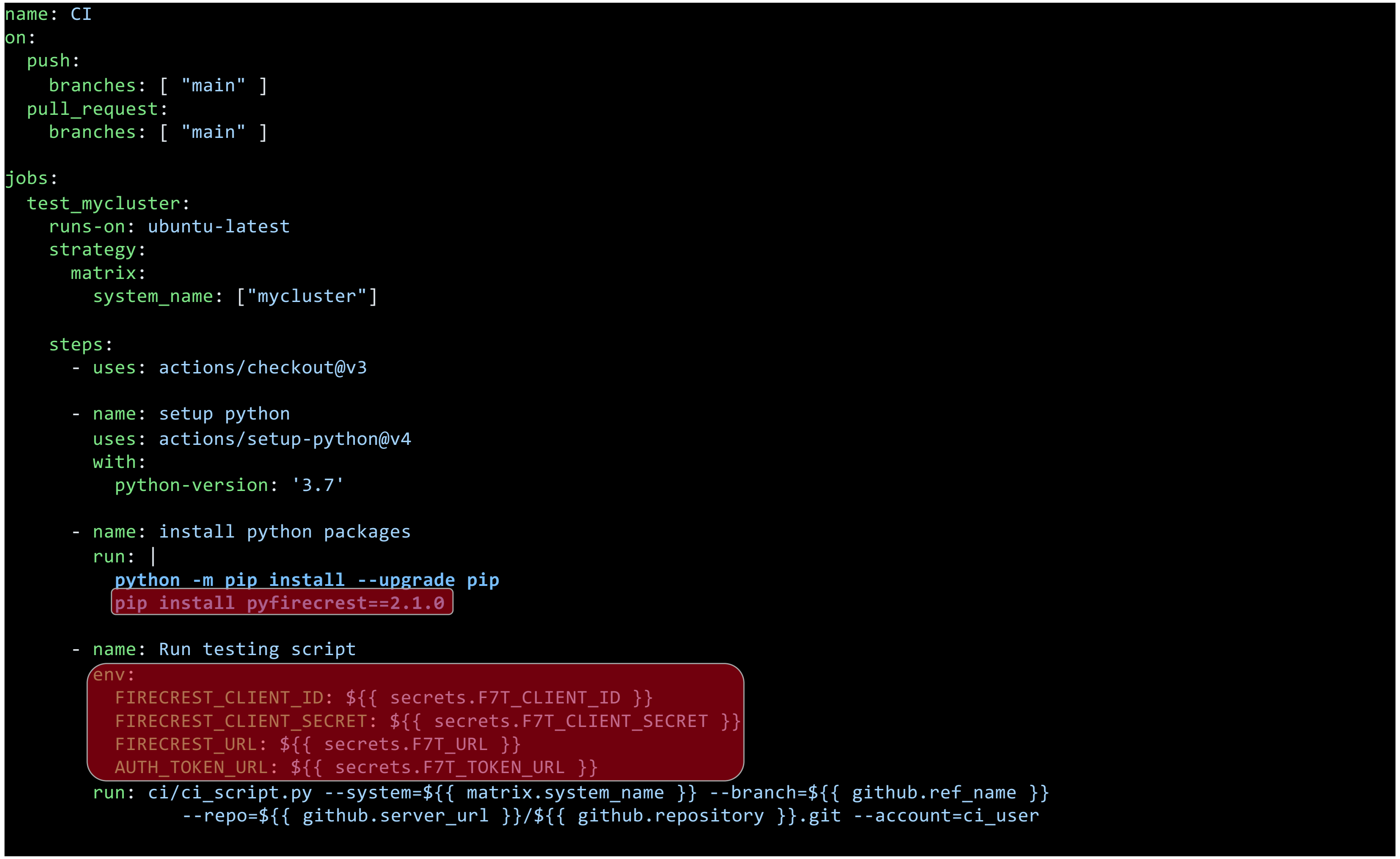
Job polling and result check



```python
# importing PyFirecREST
import firecrest as f7t

# Setup variables of the client
CLIENT_ID = os.environ.get("FIRECREST_CLIENT_ID")
CLIENT_SECRET = os.environ.get("FIRECREST_CLIENT_SECRET")
FIRECREST_URL = os.environ.get("FIRECREST_URL")
AUTH_TOKEN_URL = os.environ.get("AUTH_TOKEN_URL")

# Auth Object definition
idp = f7t.ClientCredentialsAuth(CLIENT_ID, CLIENT_SECRET, AUTH_TOKEN_URL)

# FirecREST client defintion
client = f7t.Firecrest(firecrest_url=FIRECREST_URL, authorization=idp)

# Check System Status via pyFirecREST
system_state = client.system(system_name)

if system_state["status"] == "available":
    # Submit job via pyFirecREST
    job = client.submit(system_name, "submission_script.sh")

    print(f"Submitted job: {job['jobid']}")

    print(f"\nSTDOUT in {job['job_file_out']}")
    stdout_content = client.head(system_name, job['job_file_out'], lines=100)
    print(stdout_content)

    print(f"\nSTDERR in {job['job_file_err']}")
    stderr_content = client.head(system_name, job['job_file_err'], lines=100)
    print(stderr_content)

    # Poll job status via pyFirecREST
    poll_result = client.poll(system_name, jobs=[job["jobid"]])
    if poll_result[0]["state"] != "COMPLETED":
        print(f"Job was not successful, status: {poll_result[0]['state']}")
        exit(1)

else:
    print("System {system_name} is not available")
    exit(1)
```

Use Cases

- Continuous Integration (CI) Pipelines
  - `.github/workflows/ci.yml`



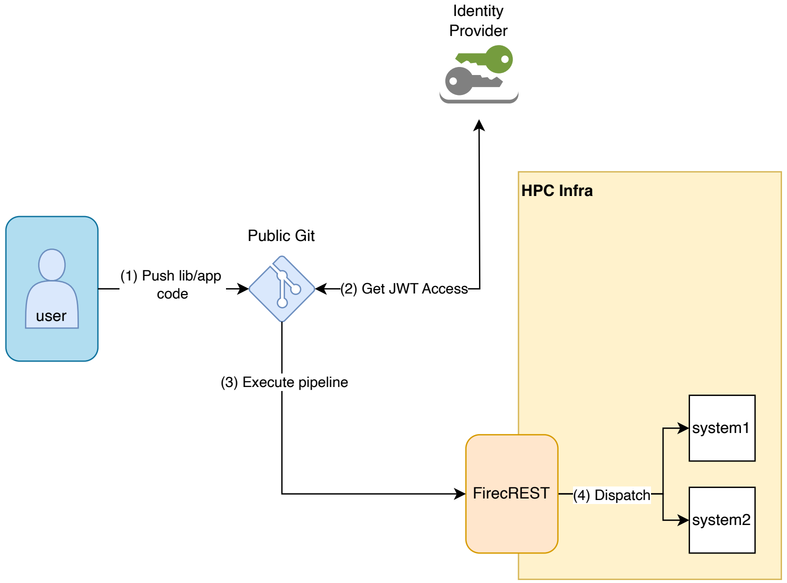
```yaml
name: CI
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  test_mycluster:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        system_name: ["mycluster"]

    steps:
      - uses: actions/checkout@v3

      - name: setup python
        uses: actions/setup-python@v4
        with:
          python-version: '3.7'

      - name: install python packages
        run: |
          python -m pip install --upgrade pip
          pip install pyfirecrest==2.1.0

      - name: Run testing script
        env:
          FIRECREST_CLIENT_ID: ${{ secrets.F7T_CLIENT_ID }}
          FIRECREST_CLIENT_SECRET: ${{ secrets.F7T_CLIENT_SECRET }}
          FIRECREST_URL: ${{ secrets.F7T_URL }}
          AUTH_TOKEN_URL: ${{ secrets.F7T_TOKEN_URL }}
        run: ci/ci_script.py --system=${{ matrix.system_name }} --branch=${{ github.ref_name }}
                --repo=${{ github.server_url }}/${{ github.repository }}.git --account=ci_user
```

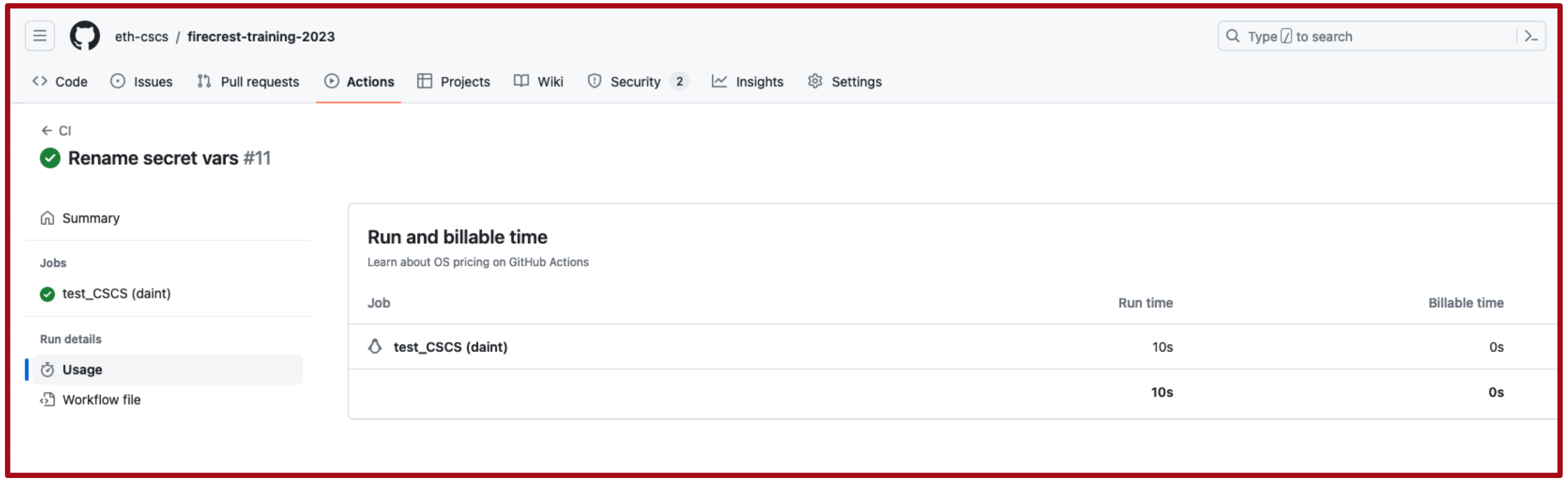
pyFirecREST installation

Environment setup

Use Cases

- Continuous Integration (CI) Pipelines





Rename secret vars #11

test_CSCS (daint)

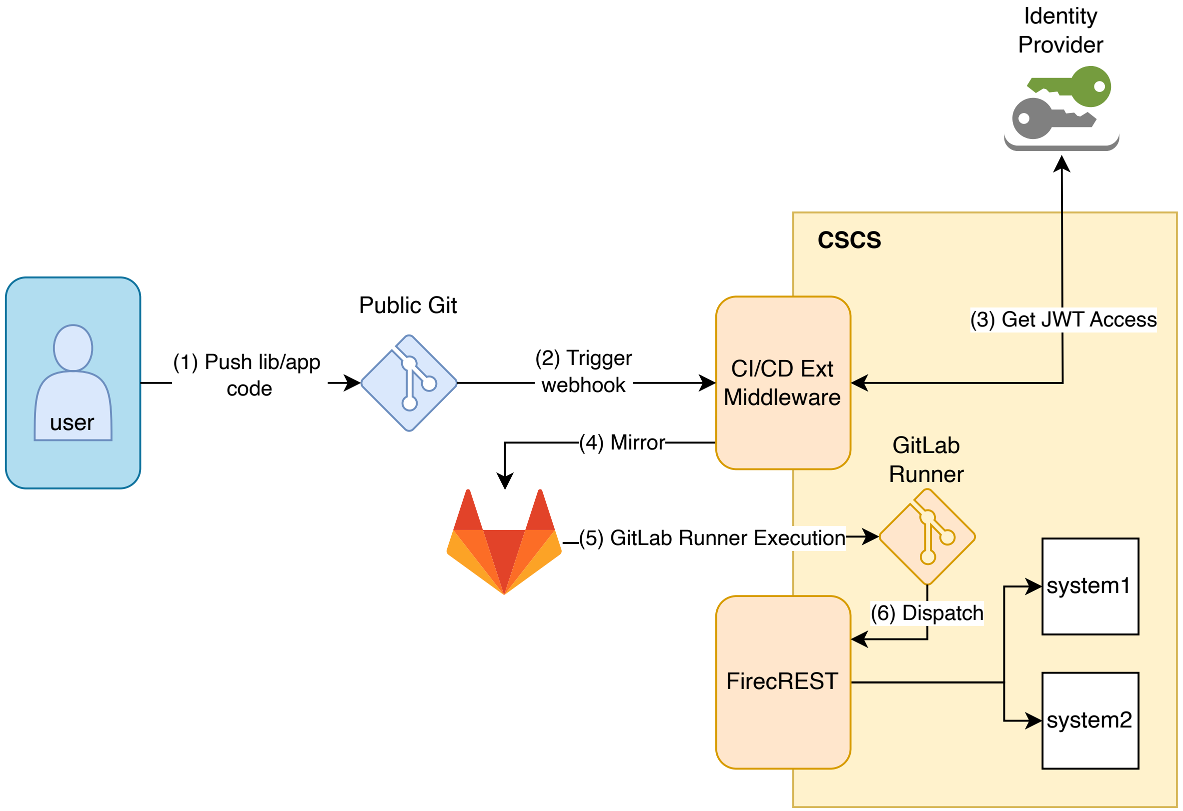
Run and billable time

Learn about OS pricing on GitHub Actions

Job	Run time	Billable time
test_CSCS (daint)	10s	0s
	10s	**0s**

Use Cases

- Continuous Integration (CI) Pipelines
  - CICD-Ext Service

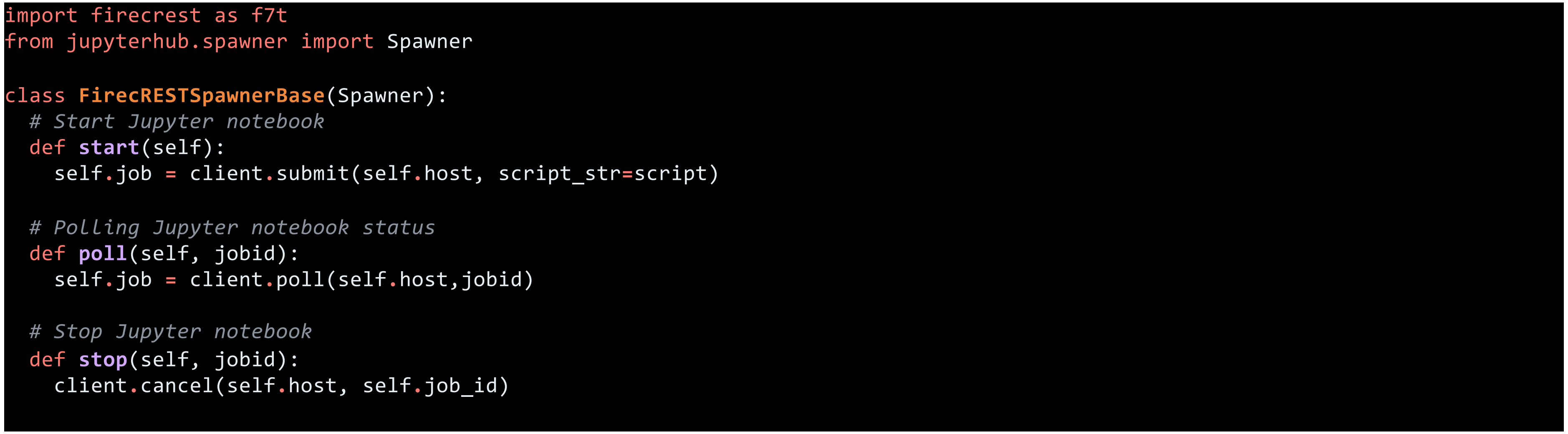


Use Cases

- Interactive Computing
  - JupyterHub (JH) it's a multi-user hub that enables launching Jupyter Notebooks from a web browser to compute nodes
  - JH is usually used for interactive computing for PoC of code, dataset exploration, and educational/training purposes
  - In HPC Clusters, JH is commonly paired with the `batchspawner` package to submit jobs in compute nodes.
  - The `batchspawner` configuration requires sysadmins to install and configure the WLM daemon in JH host and configure the key sharing between daemon and controller
  - This complicates the deployment of JH and restrict the systems that can operate with this tool

Use Cases

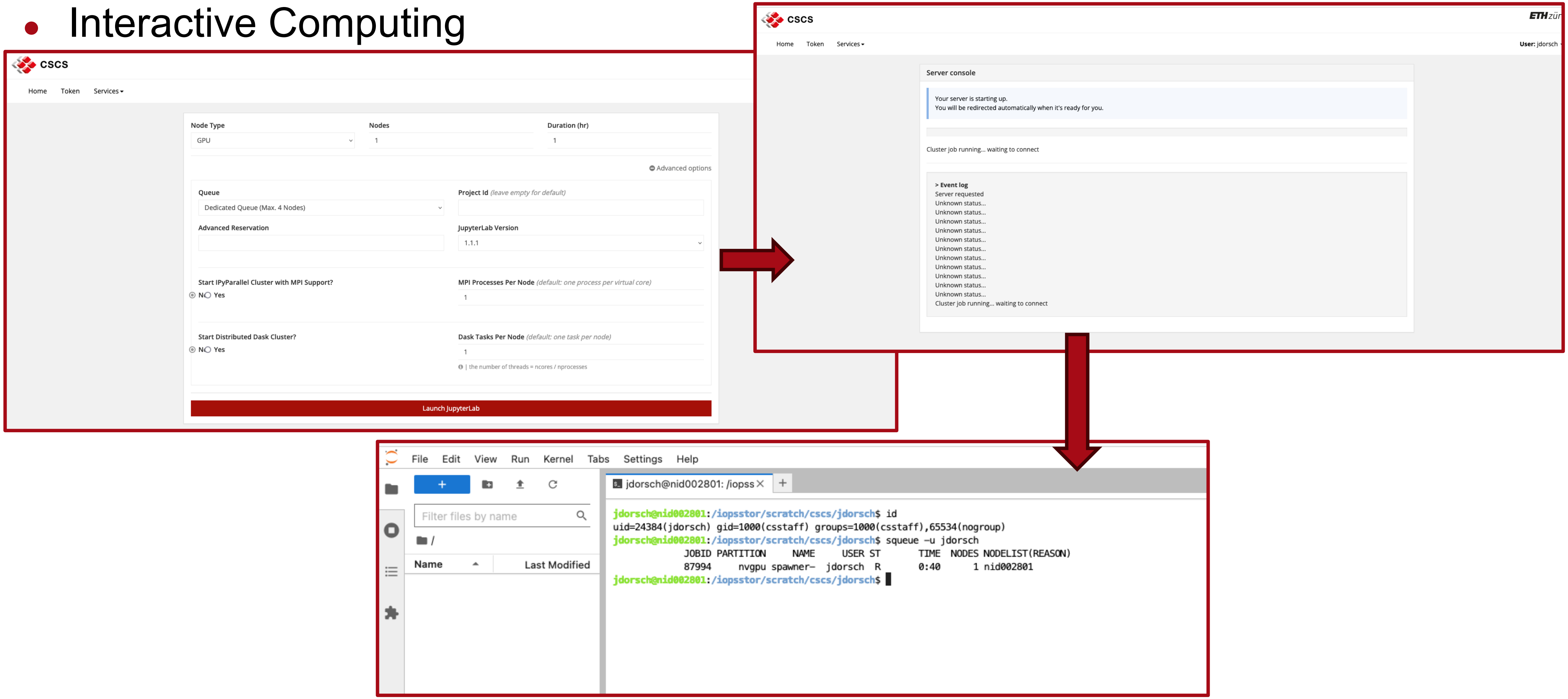
- Interactive Computing
  - With pyFirecREST, and taking advantage of the JupyterHub `Spawner` base class, a customized FirecREST Spawner (`FirecRESTSpawnerBase`) has been created and configured in a JupyterHub image
  - `Spawner` base class needs **`start()`**, **`poll()`**, and **`stop()`** methods to be implemented



```
import firecrest as f7t
from jupyterhub.spawner import Spawner

class FirecRESTSpawnerBase(Spawner):
  # Start Jupyter notebook
  def start(self):
    self.job = client.submit(self.host, script_str=script)

  # Polling Jupyter notebook status
  def poll(self, jobid):
    self.job = client.poll(self.host,jobid)

  # Stop Jupyter notebook
  def stop(self, jobid):
    client.cancel(self.host, self.job_id)
```

Use Cases

- Interactive Computing



Node Type: GPU
Nodes: 1
Duration (hr): 1

Queue: Dedicated Queue (Max. 4 Nodes)
Project Id *(leave empty for default)*:
Advanced Reservation:
JupyterLab Version: 1.1.1

Start IPyParallel Cluster with MPI Support? ● No ○ Yes
MPI Processes Per Node *(default: one process per virtual core)*: 1

Start Distributed Dask Cluster? ● No ○ Yes
Dask Tasks Per Node *(default: one task per node)*: 1
| the number of threads = ncores / nprocesses

Server console

Your server is starting up.
You will be redirected automatically when it's ready for you.

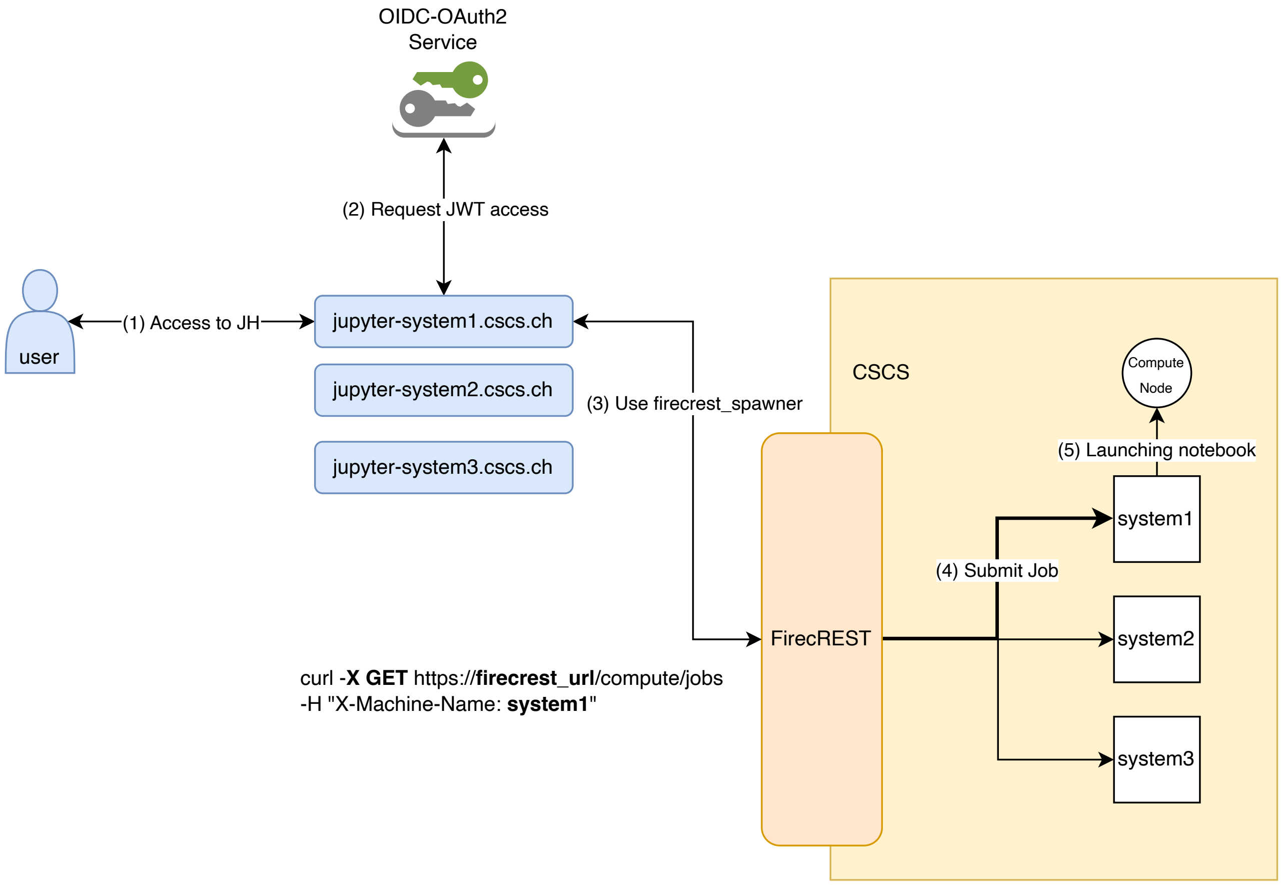
Cluster job running... waiting to connect

```
> Event log
Server requested
Unknown status...
Unknown status...
Unknown status...
Unknown status...
Unknown status...
Unknown status...
Unknown status...
Unknown status...
Unknown status...
Unknown status...
Unknown status...
Cluster job running... waiting to connect
```

```
jdorsch@nid002801:/iopsstor/scratch/cscs/jdorsch$ id
uid=24384(jdorsch) gid=1000(csstaff) groups=1000(csstaff),65534(nogroup)
jdorsch@nid002801:/iopsstor/scratch/cscs/jdorsch$ squeue -u jdorsch
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             87994     nvgpu spawner-  jdorsch  R       0:40      1 nid002801
jdorsch@nid002801:/iopsstor/scratch/cscs/jdorsch$
```

Use Cases

- Interactive Computing
    - Reduces the requirement on the HPC infrastructure side in terms of administration, machine provisioning, networking, etc.
    - The “recipe” can be replicated for several HPC systems by changing the configuration to a different system
    - Integration with IAM allows the same OIDC client for JH and FirecREST



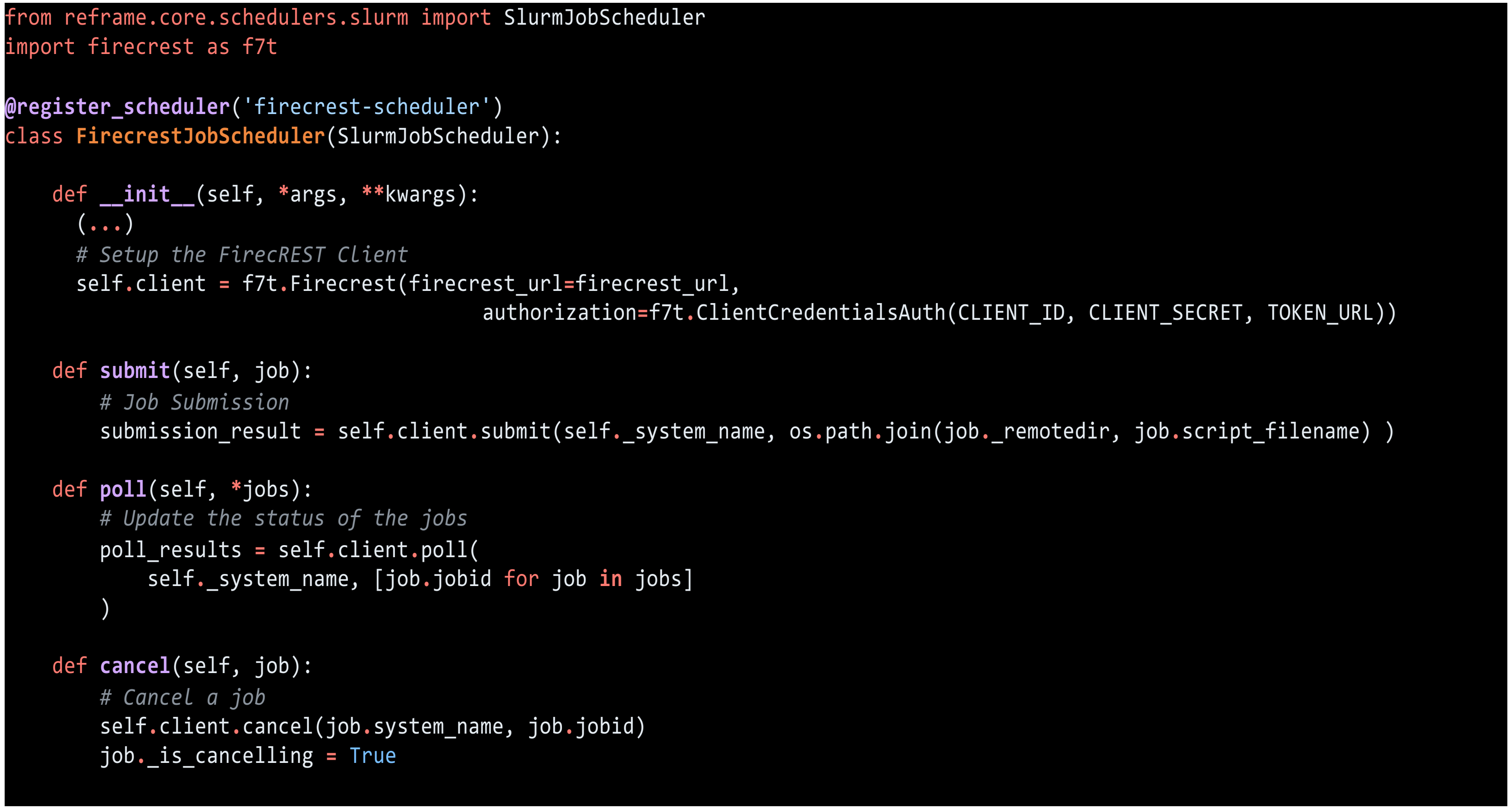
```
curl -X GET https://firecrest_url/compute/jobs
-H "X-Machine-Name: system1"
```

Use Cases

- Regression Testing
  - ReFrame is a framework for regression testing on HPC system
  - It allows periodic testing of scientific software ensuring performance and integrity
  - The pipeline of ReFrame for each test presents the following stages: (1) setup, (2) compile, (3) run, (4) sanity, (5) performance, and (6) cleanup
  - ReFrame needs to be installed and executed in the HPC system in which the software is being tested.
  - With FirecREST it is possible to run a ReFrame test from a laptop or any public cloud provider, thus de-attaching the operation of the service from the HPC provider

Use Cases

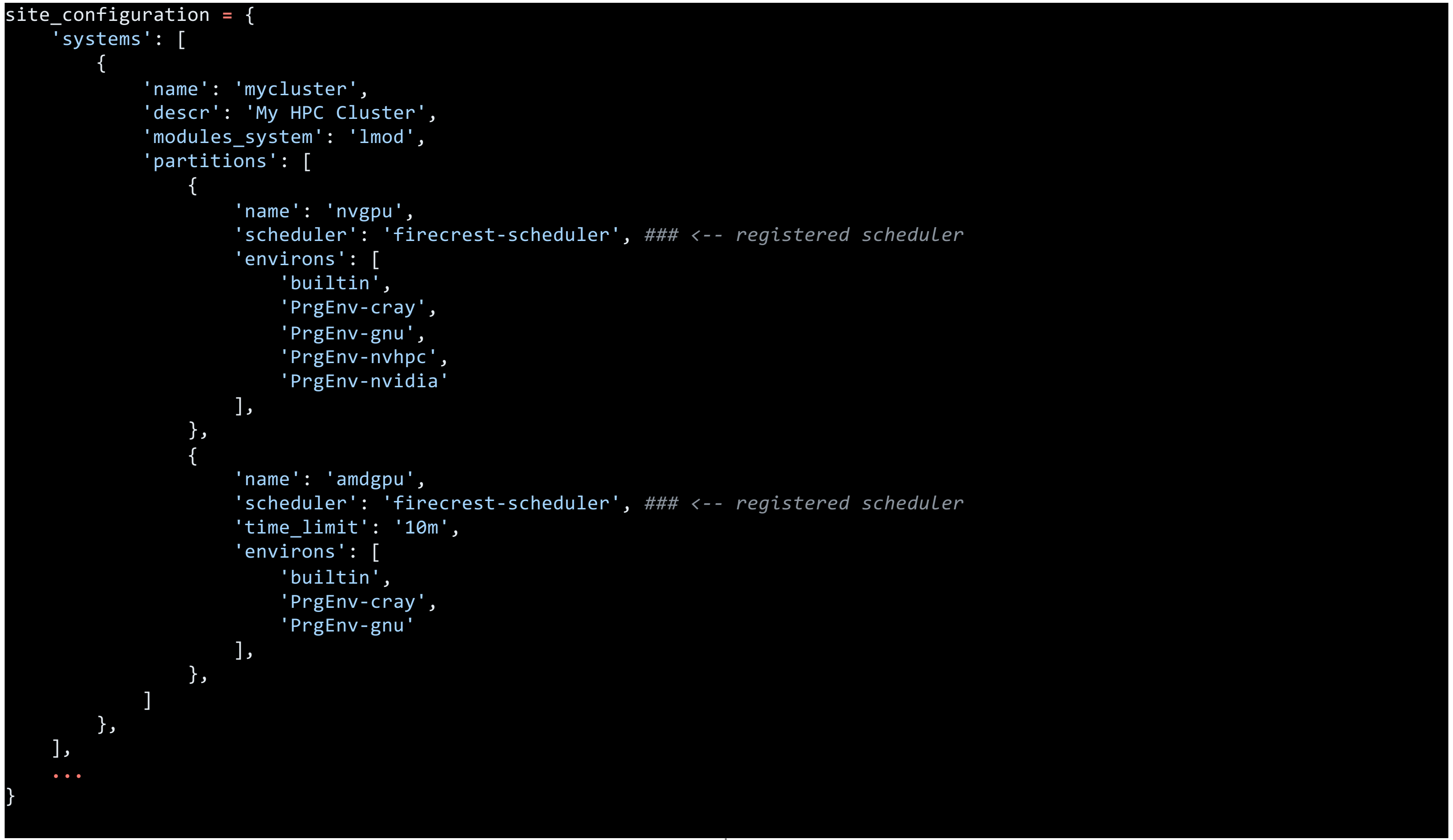
- Regression Testing
  - ReFrame provides a Python class for schedulers. We can use pyFirecREST to adapt a `"firecrest-scheduler"` scheduler by extending the `SlurmJobScheduler` class



```
from reframe.core.schedulers.slurm import SlurmJobScheduler
import firecrest as f7t

@register_scheduler('firecrest-scheduler')
class FirecrestJobScheduler(SlurmJobScheduler):

    def __init__(self, *args, **kwargs):
      (...)
      # Setup the FirecREST Client
      self.client = f7t.Firecrest(firecrest_url=firecrest_url,
                                        authorization=f7t.ClientCredentialsAuth(CLIENT_ID, CLIENT_SECRET, TOKEN_URL))

    def submit(self, job):
        # Job Submission
        submission_result = self.client.submit(self._system_name, os.path.join(job._remotedir, job.script_filename) )

    def poll(self, *jobs):
        # Update the status of the jobs
        poll_results = self.client.poll(
            self._system_name, [job.jobid for job in jobs]
        )

    def cancel(self, job):
        # Cancel a job
        self.client.cancel(job.system_name, job.jobid)
        job._is_cancelling = True
```

Use Cases

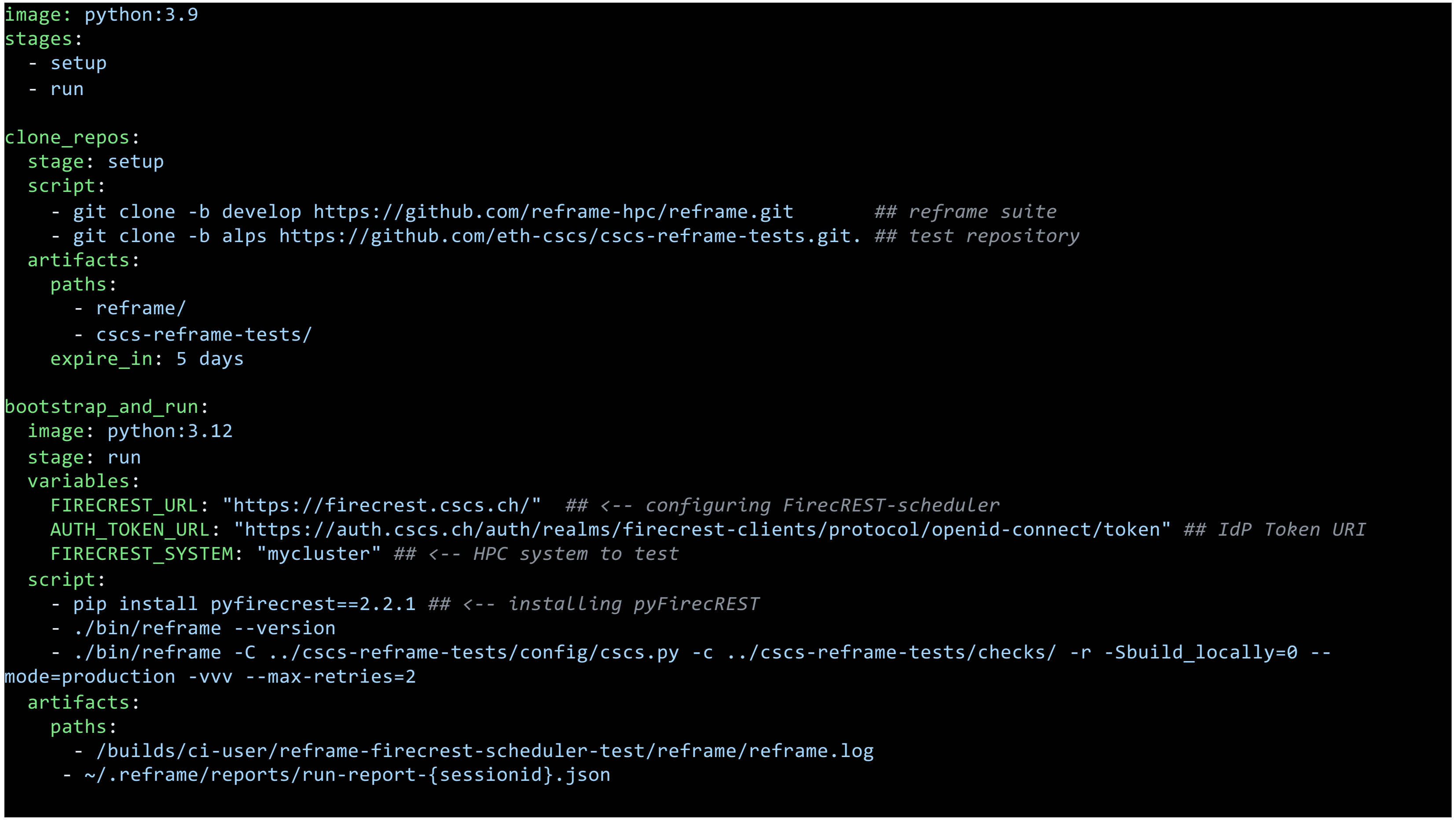
- Regression Testing
    - ReFrame requires of a configuration file, where the `"firecrest-scheduler"` among other settings, must be set



```
site_configuration = {
    'systems': [
        {
            'name': 'mycluster',
            'descr': 'My HPC Cluster',
            'modules_system': 'lmod',
            'partitions': [
                {
                    'name': 'nvgpu',
                    'scheduler': 'firecrest-scheduler', ### <-- registered scheduler
                    'environs': [
                        'builtin',
                        'PrgEnv-cray',
                        'PrgEnv-gnu',
                        'PrgEnv-nvhpc',
                        'PrgEnv-nvidia'
                    ],
                },
                {
                    'name': 'amdgpu',
                    'scheduler': 'firecrest-scheduler', ### <-- registered scheduler
                    'time_limit': '10m',
                    'environs': [
                        'builtin',
                        'PrgEnv-cray',
                        'PrgEnv-gnu'
                    ],
                },
            ]
        },
    ],
    ...
}
```

Use Cases

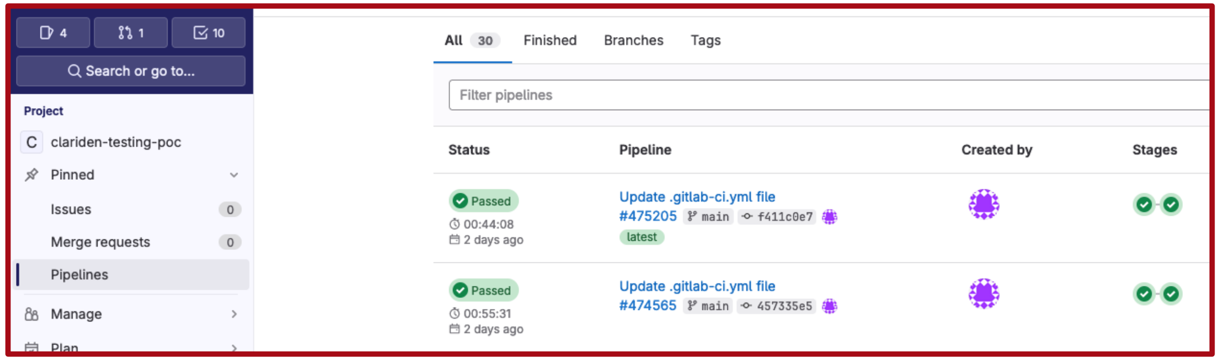
- Regression Testing
  - Finally, this is set on a CI Pipeline and it can be executed by a Runner from any server



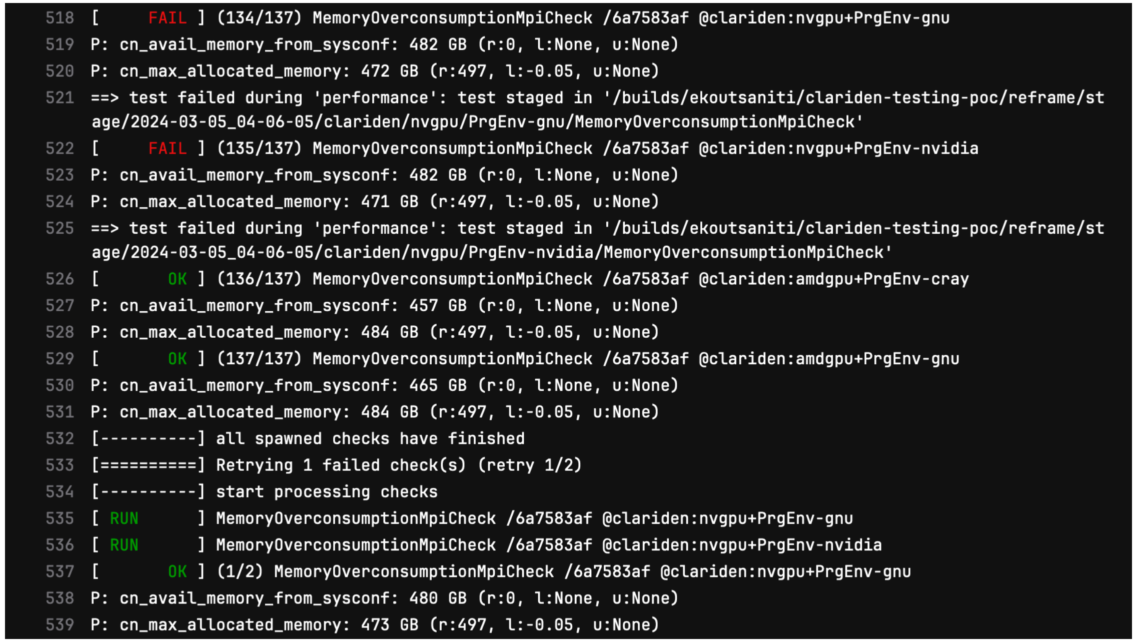
```
image: python:3.9
stages:
  - setup
  - run

clone_repos:
  stage: setup
  script:
    - git clone -b develop https://github.com/reframe-hpc/reframe.git       ## reframe suite
    - git clone -b alps https://github.com/eth-cscs/cscs-reframe-tests.git. ## test repository
  artifacts:
    paths:
      - reframe/
      - cscs-reframe-tests/
    expire_in: 5 days

bootstrap_and_run:
  image: python:3.12
  stage: run
  variables:
    FIRECREST_URL: "https://firecrest.cscs.ch/"  ## <-- configuring FirecREST-scheduler
    AUTH_TOKEN_URL: "https://auth.cscs.ch/auth/realms/firecrest-clients/protocol/openid-connect/token" ## IdP Token URI
    FIRECREST_SYSTEM: "mycluster" ## <-- HPC system to test
  script:
    - pip install pyfirecrest==2.2.1 ## <-- installing pyFirecREST
    - ./bin/reframe --version
    - ./bin/reframe -C ../cscs-reframe-tests/config/cscs.py -c ../cscs-reframe-tests/checks/ -r -Sbuild_locally=0 --
mode=production -vvv --max-retries=2
  artifacts:
    paths:
      - /builds/ci-user/reframe-firecrest-scheduler-test/reframe/reframe.log
     - ~/.reframe/reports/run-report-{sessionid}.json
```

Use Cases

- Regression Testing



Status	Pipeline	Created by	Stages
Passed 00:44:08 2 days ago	Update .gitlab-ci.yml file #475205 main f411c0e7 latest		✓ ✓
Passed 00:55:31 2 days ago	Update .gitlab-ci.yml file #474565 main 457335e5		✓ ✓



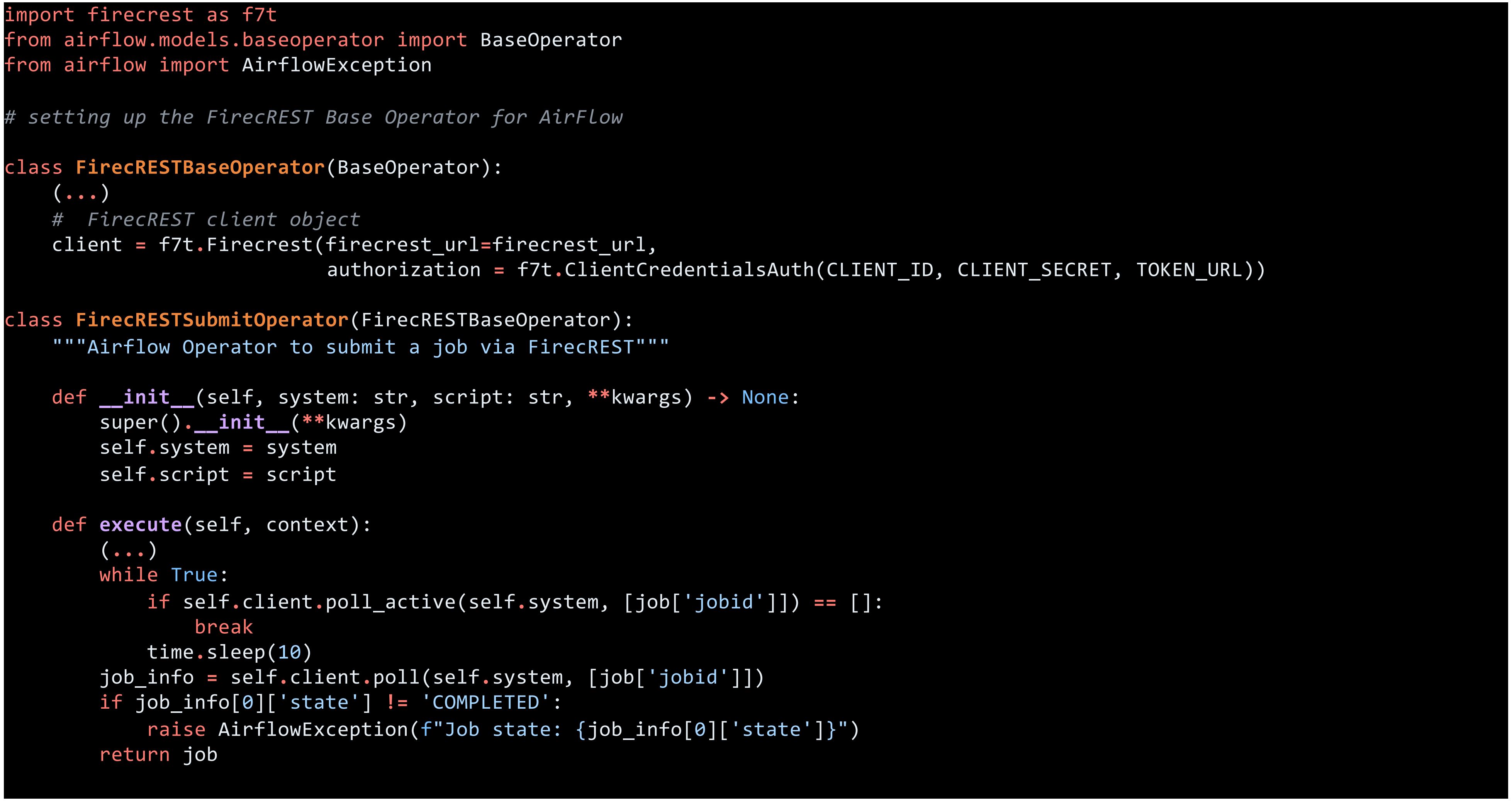
```
[     FAIL ] (134/137) MemoryOverconsumptionMpiCheck /6a7583af @clariden:nvgpu+PrgEnv-gnu
P: cn_avail_memory_from_sysconf: 482 GB (r:0, l:None, u:None)
P: cn_max_allocated_memory: 472 GB (r:497, l:-0.05, u:None)
==> test failed during 'performance': test staged in '/builds/ekoutsaniti/clariden-testing-poc/reframe/stage/2024-03-05_04-06-05/clariden/nvgpu/PrgEnv-gnu/MemoryOverconsumptionMpiCheck'
[     FAIL ] (135/137) MemoryOverconsumptionMpiCheck /6a7583af @clariden:nvgpu+PrgEnv-nvidia
P: cn_avail_memory_from_sysconf: 482 GB (r:0, l:None, u:None)
P: cn_max_allocated_memory: 471 GB (r:497, l:-0.05, u:None)
==> test failed during 'performance': test staged in '/builds/ekoutsaniti/clariden-testing-poc/reframe/stage/2024-03-05_04-06-05/clariden/nvgpu/PrgEnv-nvidia/MemoryOverconsumptionMpiCheck'
[       OK ] (136/137) MemoryOverconsumptionMpiCheck /6a7583af @clariden:amdgpu+PrgEnv-cray
P: cn_avail_memory_from_sysconf: 457 GB (r:0, l:None, u:None)
P: cn_max_allocated_memory: 484 GB (r:497, l:-0.05, u:None)
[       OK ] (137/137) MemoryOverconsumptionMpiCheck /6a7583af @clariden:amdgpu+PrgEnv-gnu
P: cn_avail_memory_from_sysconf: 465 GB (r:0, l:None, u:None)
P: cn_max_allocated_memory: 484 GB (r:497, l:-0.05, u:None)
[----------] all spawned checks have finished
[==========] Retrying 1 failed check(s) (retry 1/2)
[----------] start processing checks
[ RUN      ] MemoryOverconsumptionMpiCheck /6a7583af @clariden:nvgpu+PrgEnv-gnu
[ RUN      ] MemoryOverconsumptionMpiCheck /6a7583af @clariden:nvgpu+PrgEnv-nvidia
[       OK ] (1/2) MemoryOverconsumptionMpiCheck /6a7583af @clariden:nvgpu+PrgEnv-gnu
P: cn_avail_memory_from_sysconf: 480 GB (r:0, l:None, u:None)
P: cn_max_allocated_memory: 473 GB (r:497, l:-0.05, u:None)
```

Use Cases

- Workflow Orchestrator
  - Apache AirFlow (AF) offers a framework for defining workflows, particularly on the Machine Learning (ML) domain
  - AF doesn’t provide a native HPC integration for WLM
  - The workaround on integration with HPC systems is to use custom commands for job submission and monitoring.
  - FirecREST can be integrated in AF using the Operator API
  - The integration with FirecREST allows writing Directed Acyclic Graphs (DAGs) that could include tasks that run on HPC facilities

Use Cases

- Workflow Orchestrator

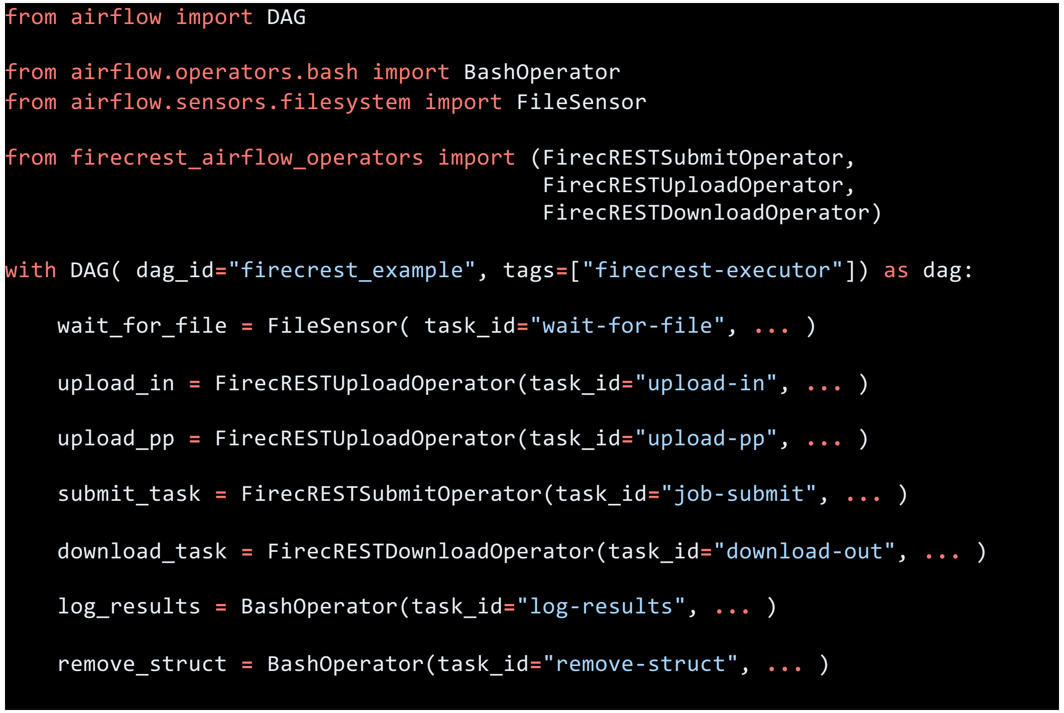


```python
import firecrest as f7t
from airflow.models.baseoperator import BaseOperator
from airflow import AirflowException

# setting up the FirecREST Base Operator for AirFlow

class FirecRESTBaseOperator(BaseOperator):
    (...)
    #  FirecREST client object
    client = f7t.Firecrest(firecrest_url=firecrest_url,
                           authorization = f7t.ClientCredentialsAuth(CLIENT_ID, CLIENT_SECRET, TOKEN_URL))

class FirecRESTSubmitOperator(FirecRESTBaseOperator):
    """Airflow Operator to submit a job via FirecREST"""

    def __init__(self, system: str, script: str, **kwargs) -> None:
        super().__init__(**kwargs)
        self.system = system
        self.script = script

    def execute(self, context):
        (...)
        while True:
            if self.client.poll_active(self.system, [job['jobid']]) == []:
                break
            time.sleep(10)
        job_info = self.client.poll(self.system, [job['jobid']])
        if job_info[0]['state'] != 'COMPLETED':
            raise AirflowException(f"Job state: {job_info[0]['state']}")
        return job
```

Use Cases

- Workflow Orchestrator
  - DAG example (`firecrest-airflow-dag.py`)

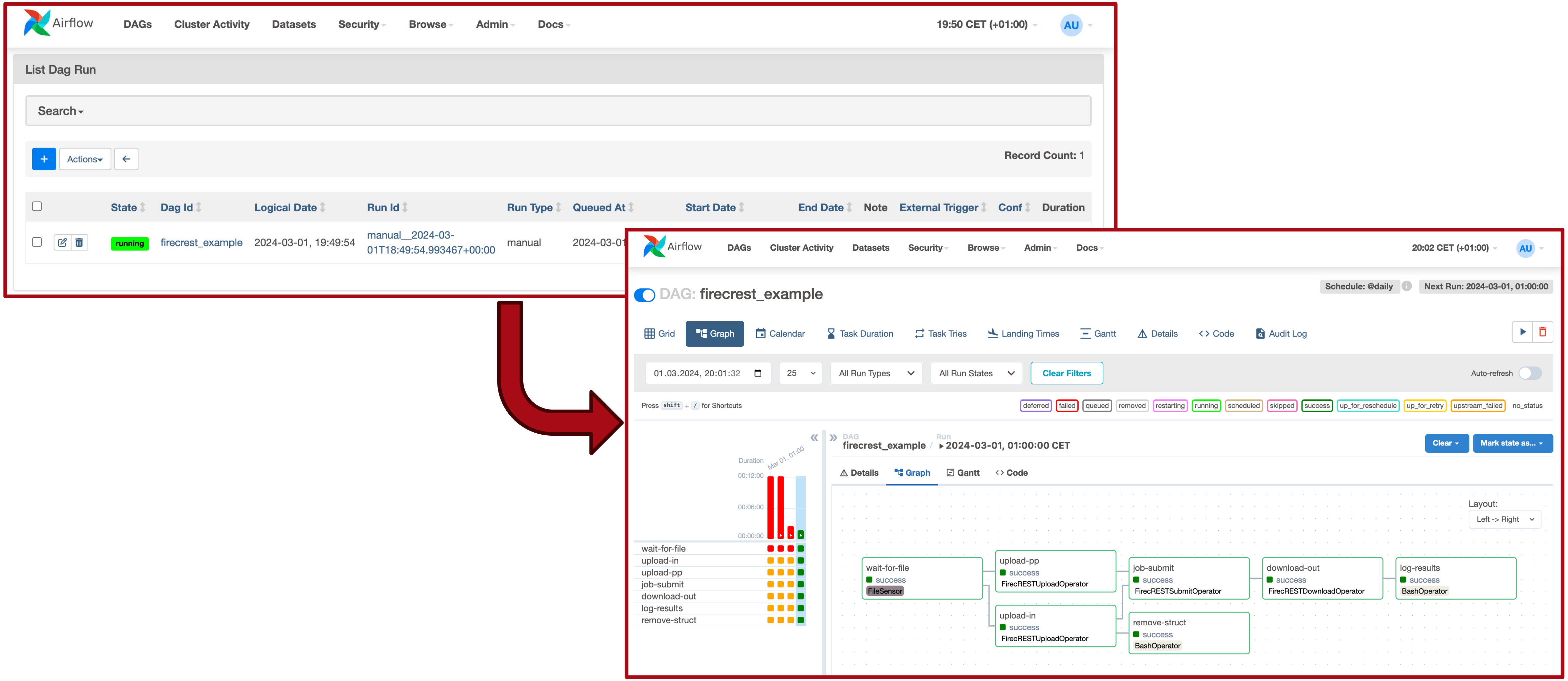
    1. Detect that a new structure has been produced
    2. Upload the structure and its pseudopotential to the HPC Cluster
    3. Submit a job to the HPC Cluster to compute the properties
    4. Download the output of the calculation
    5. Log the relevant values
    6. Delete the file with the structure



```
from airflow import DAG

from airflow.operators.bash import BashOperator
from airflow.sensors.filesystem import FileSensor

from firecrest_airflow_operators import (FirecRESTSubmitOperator,
                                         FirecRESTUploadOperator,
                                         FirecRESTDownloadOperator)

with DAG( dag_id="firecrest_example", tags=["firecrest-executor"]) as dag:

    wait_for_file = FileSensor( task_id="wait-for-file", ... )

    upload_in = FirecRESTUploadOperator(task_id="upload-in", ... )

    upload_pp = FirecRESTUploadOperator(task_id="upload-pp", ... )

    submit_task = FirecRESTSubmitOperator(task_id="job-submit", ... )

    download_task = FirecRESTDownloadOperator(task_id="download-out", ... )

    log_results = BashOperator(task_id="log-results", ... )

    remove_struct = BashOperator(task_id="remove-struct", ... )
```

Use Cases

- Workflow Orchestrator



List Dag Run

Record Count: 1

□		State	Dag Id	Logical Date	Run Id	Run Type	Queued At	Start Date	End Date	Note	External Trigger	Conf	Duration
□		running	firecrest_example	2024-03-01, 19:49:54	manual__2024-03-01T18:49:54.993467+00:00	manual	2024-03-01						

DAG: firecrest_example

Schedule: @daily

Next Run: 2024-03-01, 01:00:00

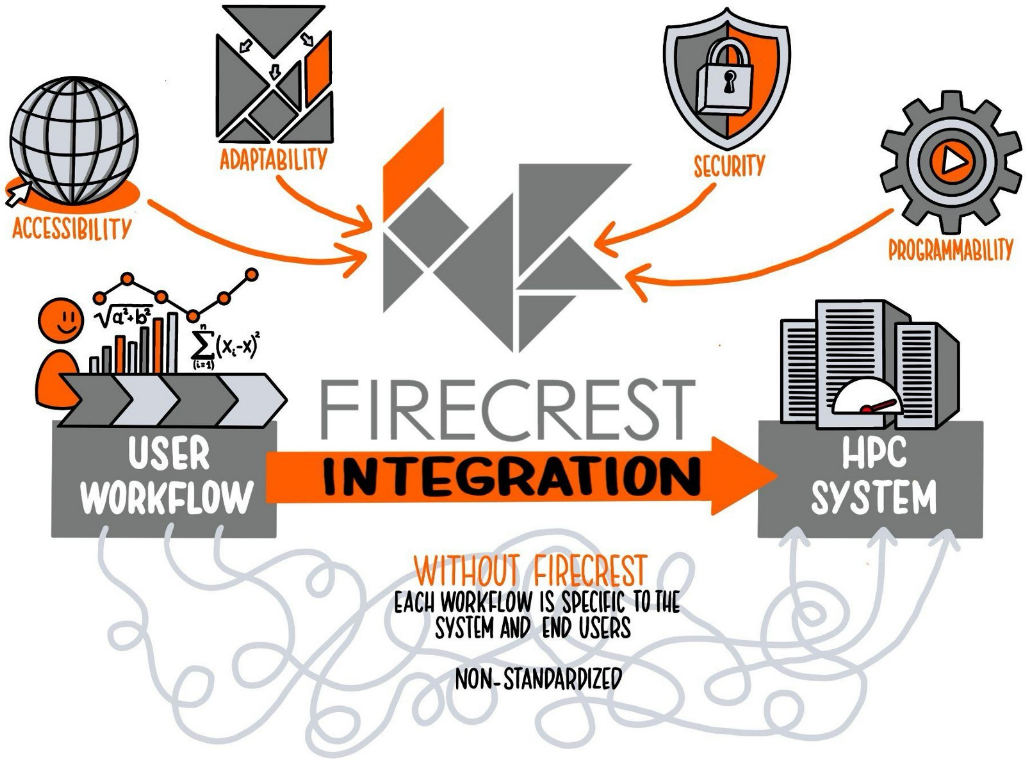
Conclusions

Conclusions

- FirecREST facilitates the integration of complex services for HPC, which allows the scientific and academic communities to deploy their own services

- Reduces the intervention of the HPC staff in terms of maintenance and support for users and their workflows

- Provides a standard service management layer for HPC and allows workflow execution across supercomputing facilities

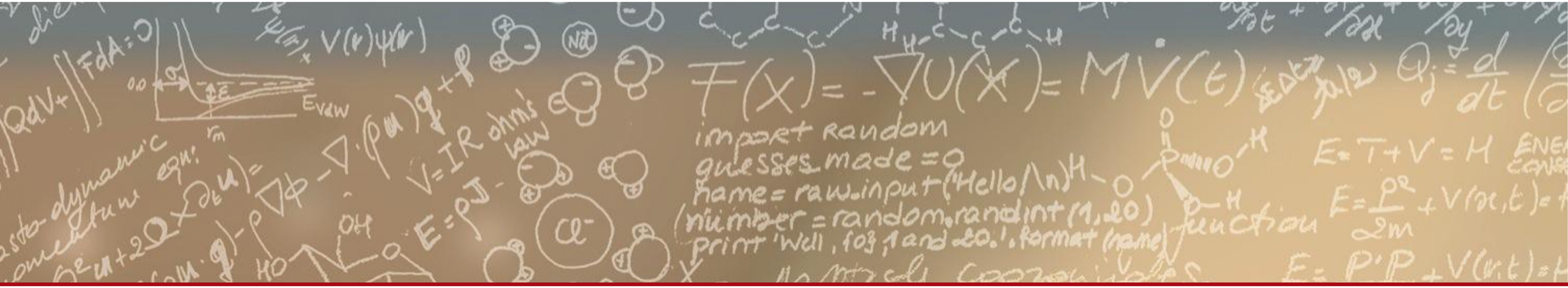


Links and references

- More on FirecREST
  - API Reference: firecrest-api.cscs.ch
  - FirecREST product page at CSCS: products.cscs.ch/firecrest
  - FirecREST public repository: github.com/eth-cscs/firecrest
  - FirecREST Docs (use cases): firecrest.readthedocs.io
  - pyFirecREST and CLI Docs: pyfirecrest.readthedocs.io
  - Join our community on Slack: firecrest-community.slack.com

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich


F(x) = -∇U(x) = MV(t)
import random
guesses_made = 0
name = raw_input('Hello\n')
number = random.randint(1, 20)
print 'Well, {0}, 1 and 20.'.format(name)
V=IR ohm's law
E = T + V = H
function

Thank you for your attention.