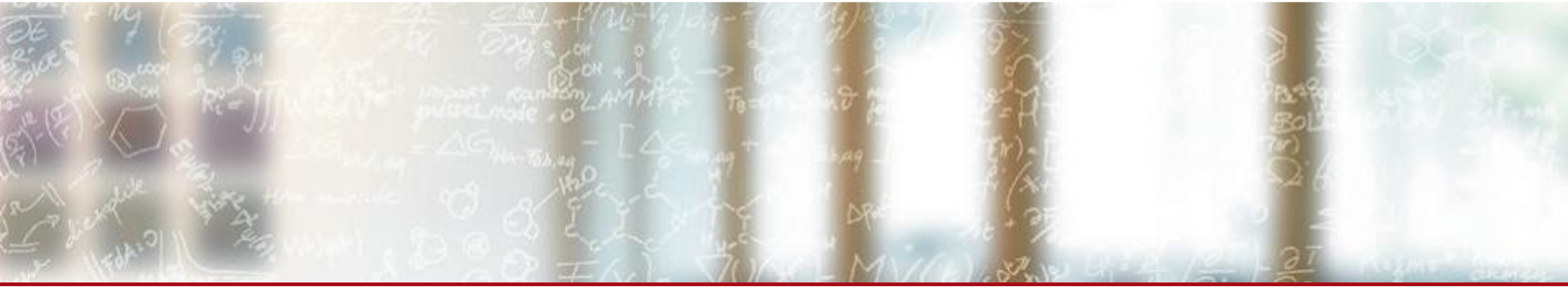
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

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

Introducing FirecREST

FirecREST in a nutshell

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

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

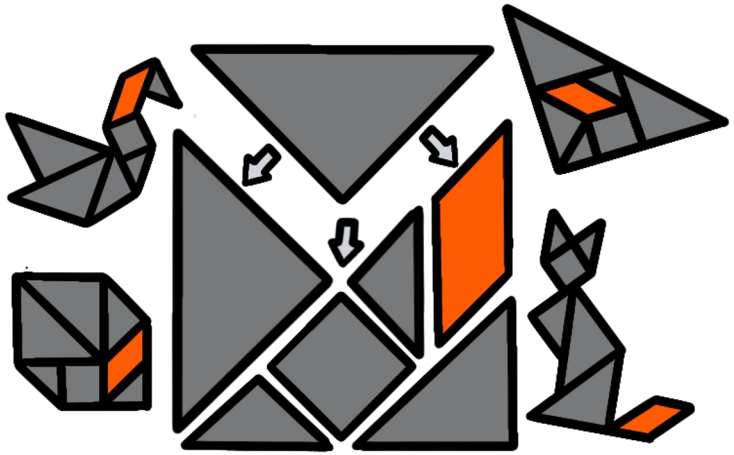
FIRECREST

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 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

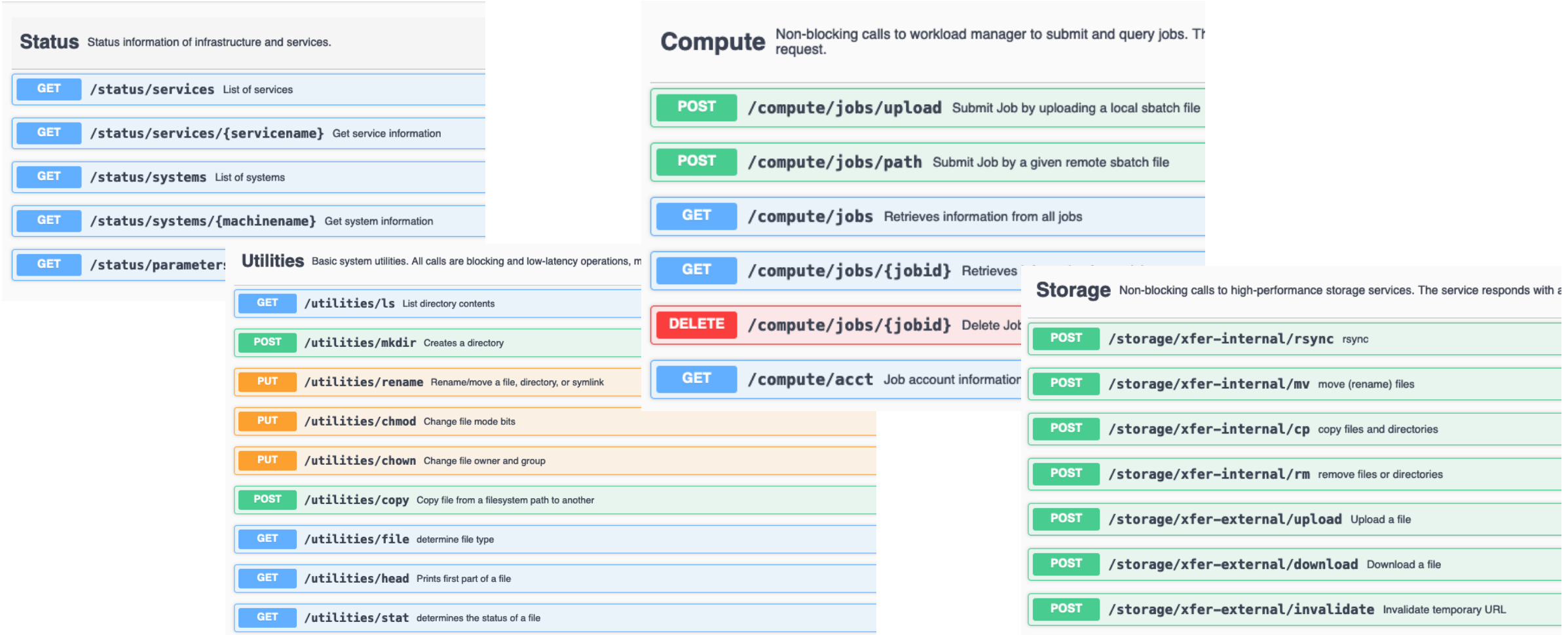


FirecREST features

- **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

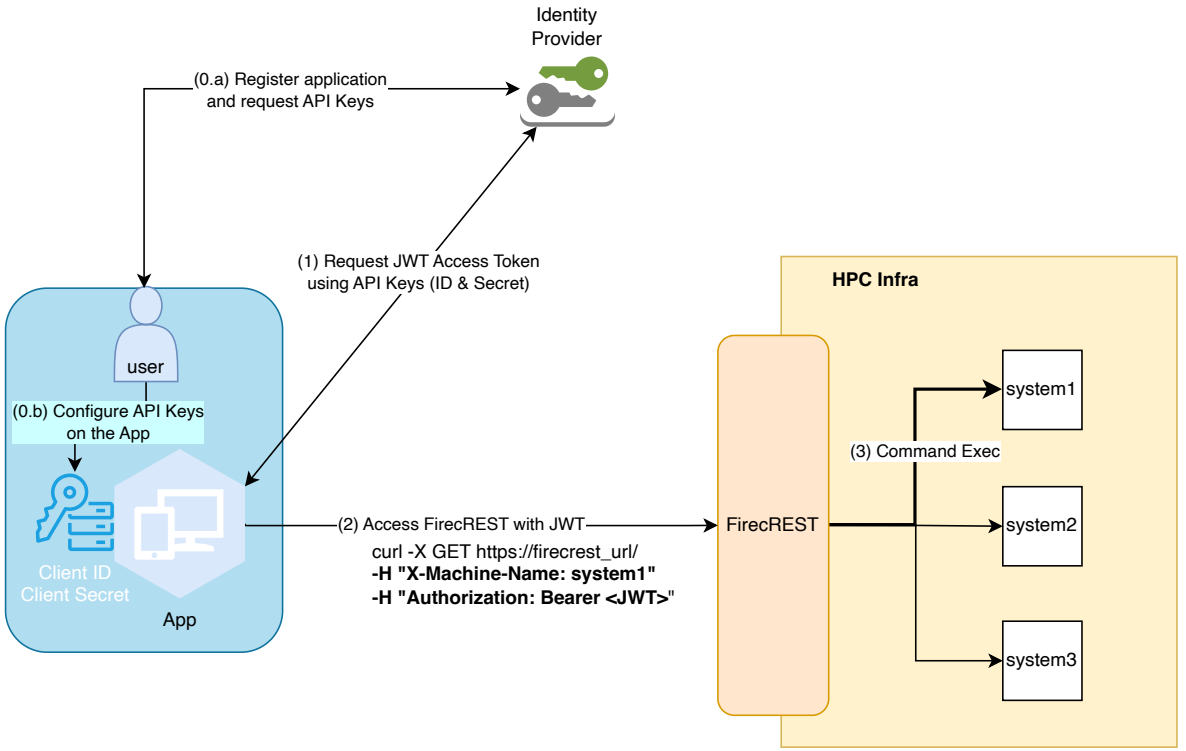
The FirecREST API

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



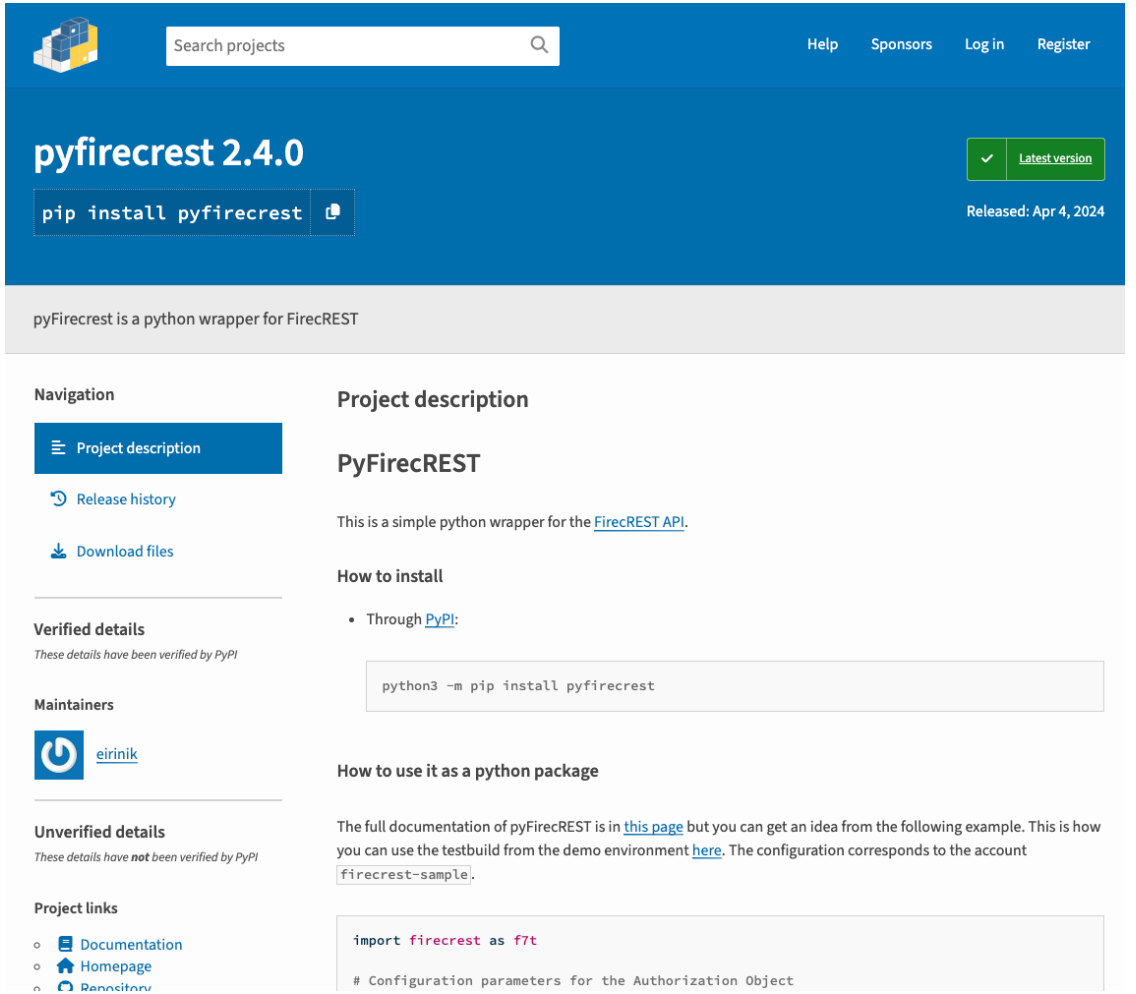
FirecREST IAM layer

- 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



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**



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`

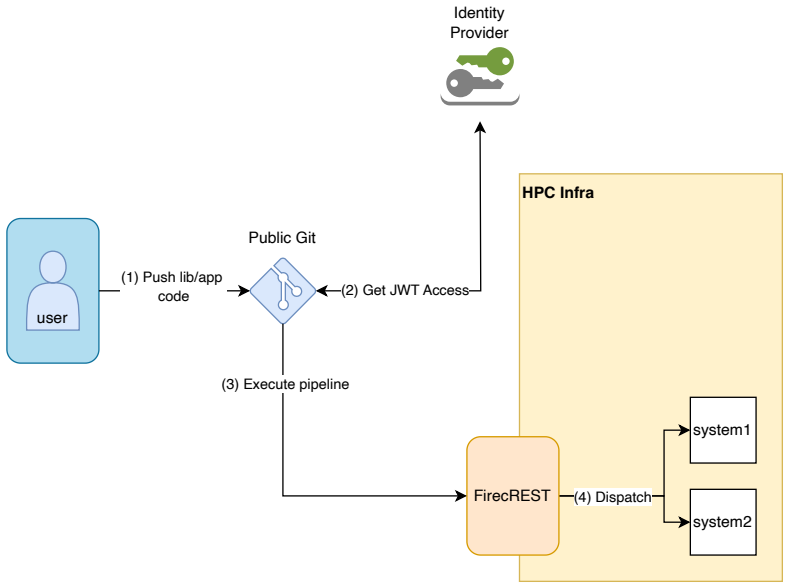
```yaml
name: CI
on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

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
```

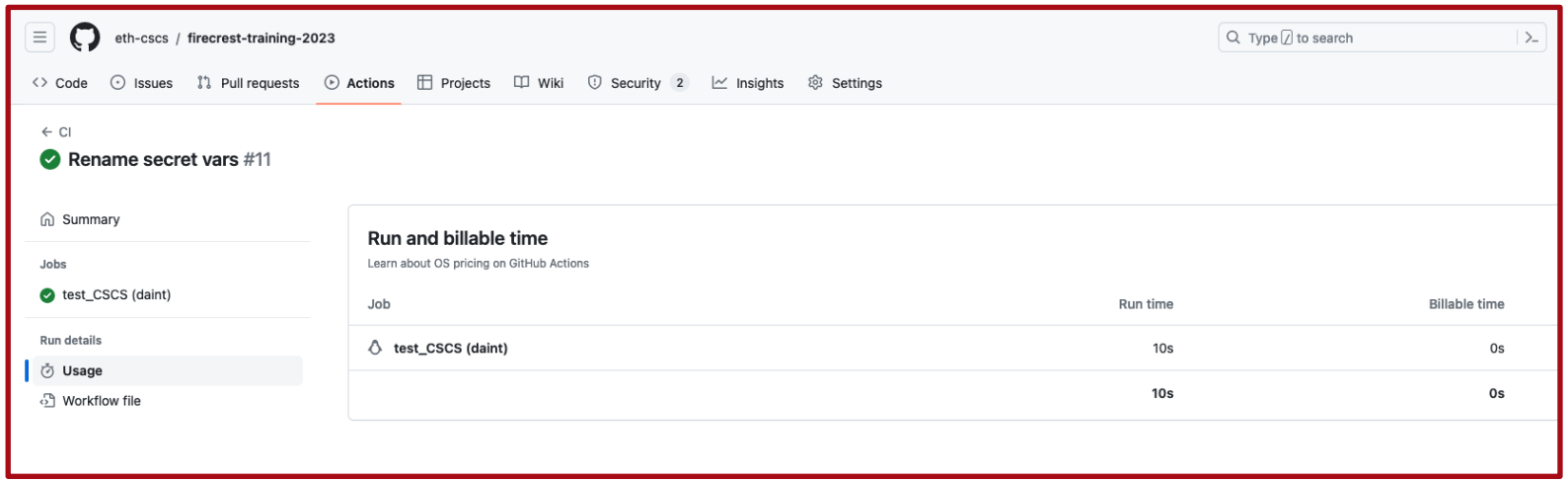
pyFirecREST installation

Environment setup

Use Cases

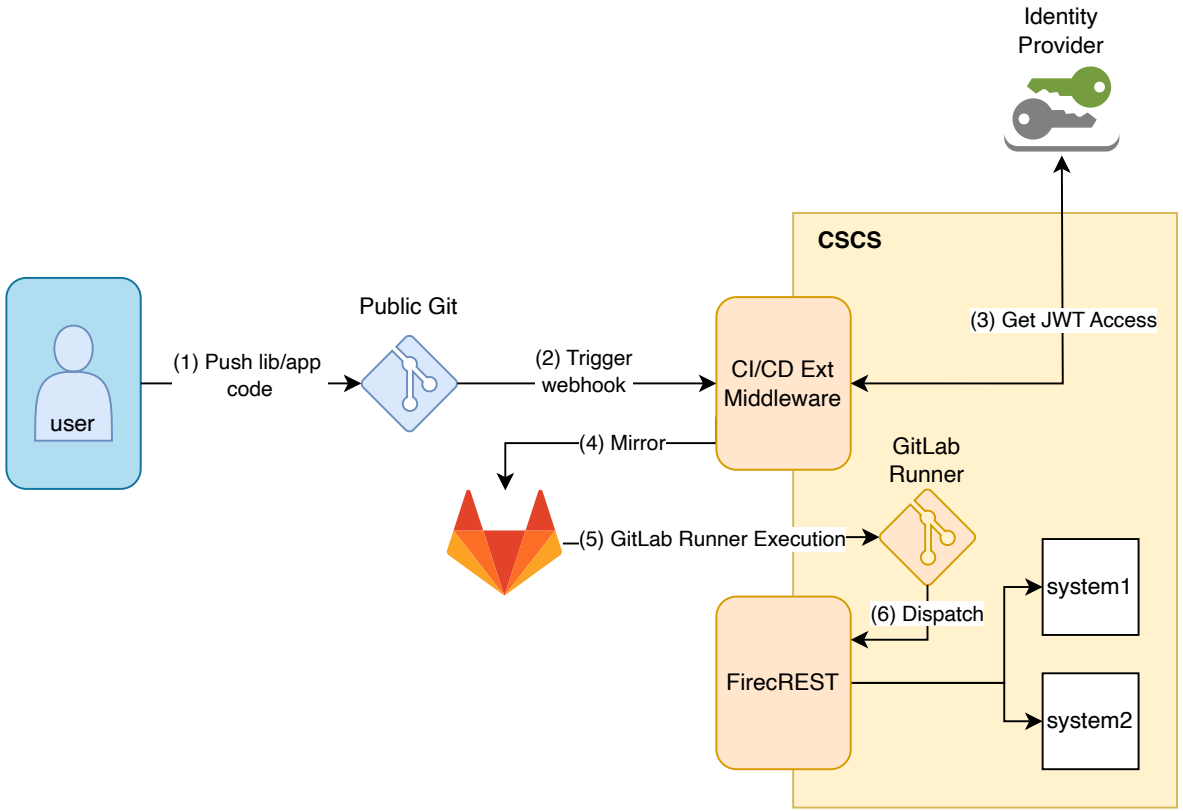
- Continuous Integration (CI) Pipelines





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

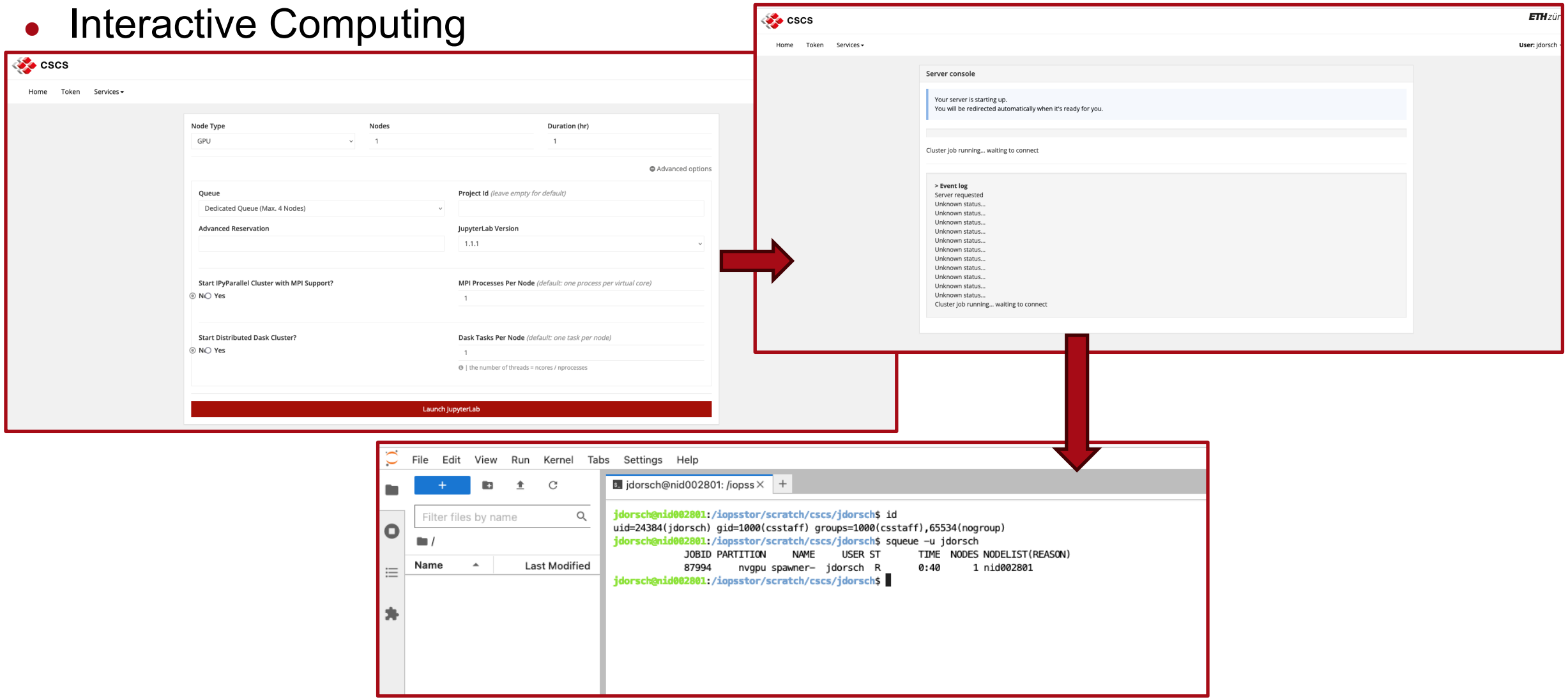
Use Cases

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

```python
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)
```

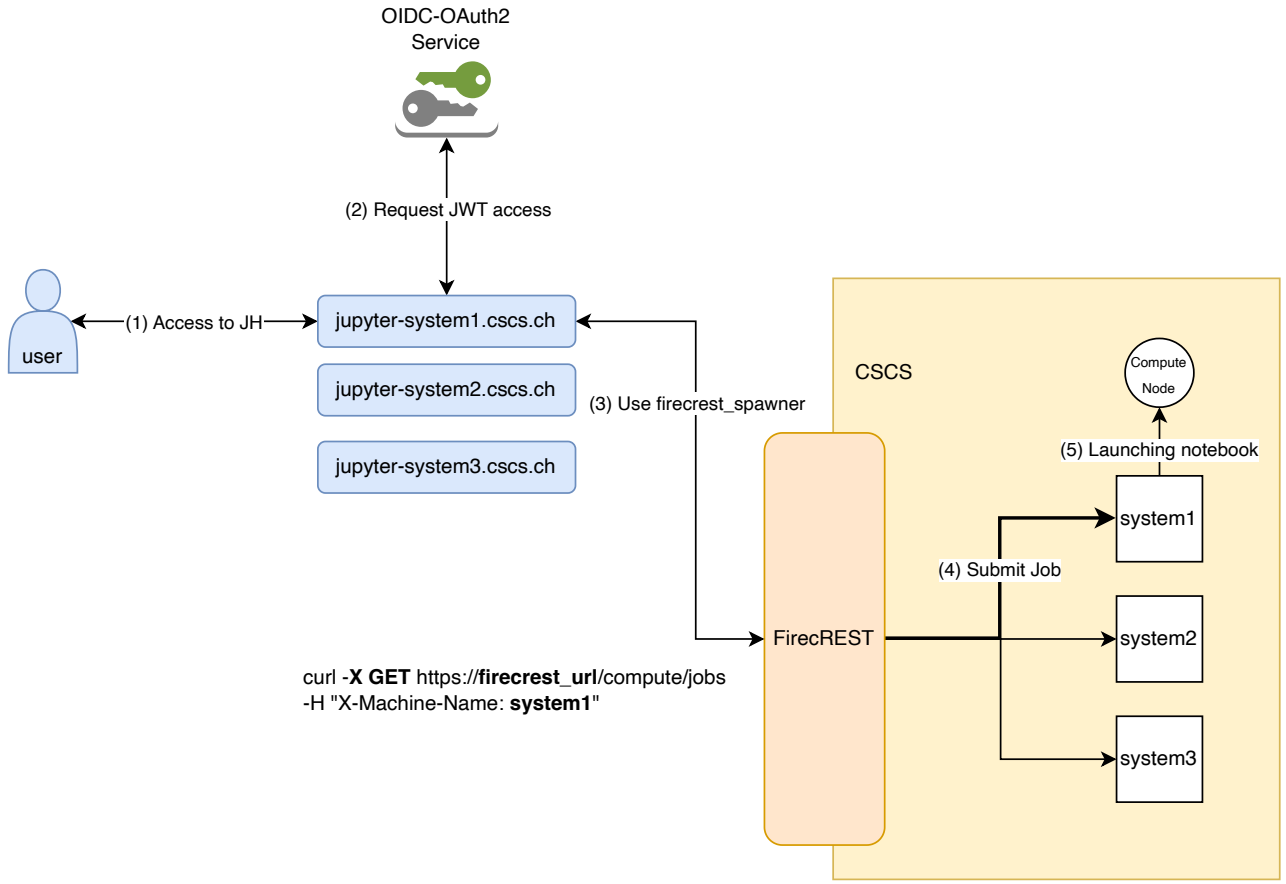
Use Cases

- Interactive Computing



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



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

```python
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'
                    ],
                },
            ]
        },
    ],
    ...
}
```

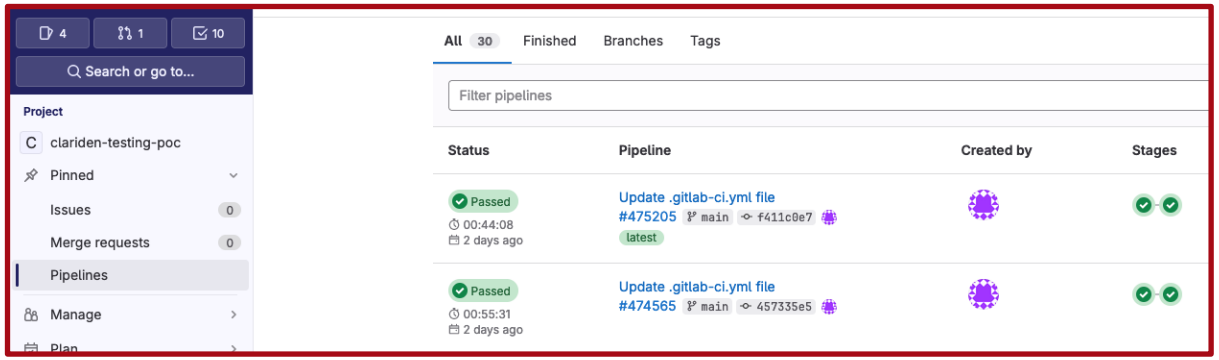
Use Cases

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

```yaml
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




```
518 [     FAIL ] (134/137) MemoryOverconsumptionMpiCheck /6a7583af @clariden:nvgpu+PrgEnv-gnu
519 P: cn_avail_memory_from_sysconf: 482 GB (r:0, l:None, u:None)
520 P: cn_max_allocated_memory: 472 GB (r:497, l:-0.05, u:None)
521 ==> test failed during 'performance': test staged in '/builds/ekoutsaniti/clariden-testing-poc/reframe/st
    age/2024-03-05_04-06-05/clariden/nvgpu/PrgEnv-gnu/MemoryOverconsumptionMpiCheck'
522 [     FAIL ] (135/137) MemoryOverconsumptionMpiCheck /6a7583af @clariden:nvgpu+PrgEnv-nvidia
523 P: cn_avail_memory_from_sysconf: 482 GB (r:0, l:None, u:None)
524 P: cn_max_allocated_memory: 471 GB (r:497, l:-0.05, u:None)
525 ==> test failed during 'performance': test staged in '/builds/ekoutsaniti/clariden-testing-poc/reframe/st
    age/2024-03-05_04-06-05/clariden/nvgpu/PrgEnv-nvidia/MemoryOverconsumptionMpiCheck'
526 [       OK ] (136/137) MemoryOverconsumptionMpiCheck /6a7583af @clariden:amdgpu+PrgEnv-cray
527 P: cn_avail_memory_from_sysconf: 457 GB (r:0, l:None, u:None)
528 P: cn_max_allocated_memory: 484 GB (r:497, l:-0.05, u:None)
529 [       OK ] (137/137) MemoryOverconsumptionMpiCheck /6a7583af @clariden:amdgpu+PrgEnv-gnu
530 P: cn_avail_memory_from_sysconf: 465 GB (r:0, l:None, u:None)
531 P: cn_max_allocated_memory: 484 GB (r:497, l:-0.05, u:None)
532 [----------] all spawned checks have finished
533 [==========] Retrying 1 failed check(s) (retry 1/2)
534 [----------] start processing checks
535 [ RUN      ] MemoryOverconsumptionMpiCheck /6a7583af @clariden:nvgpu+PrgEnv-gnu
536 [ RUN      ] MemoryOverconsumptionMpiCheck /6a7583af @clariden:nvgpu+PrgEnv-nvidia
537 [       OK ] (1/2) MemoryOverconsumptionMpiCheck /6a7583af @clariden:nvgpu+PrgEnv-gnu
538 P: cn_avail_memory_from_sysconf: 480 GB (r:0, l:None, u:None)
539 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

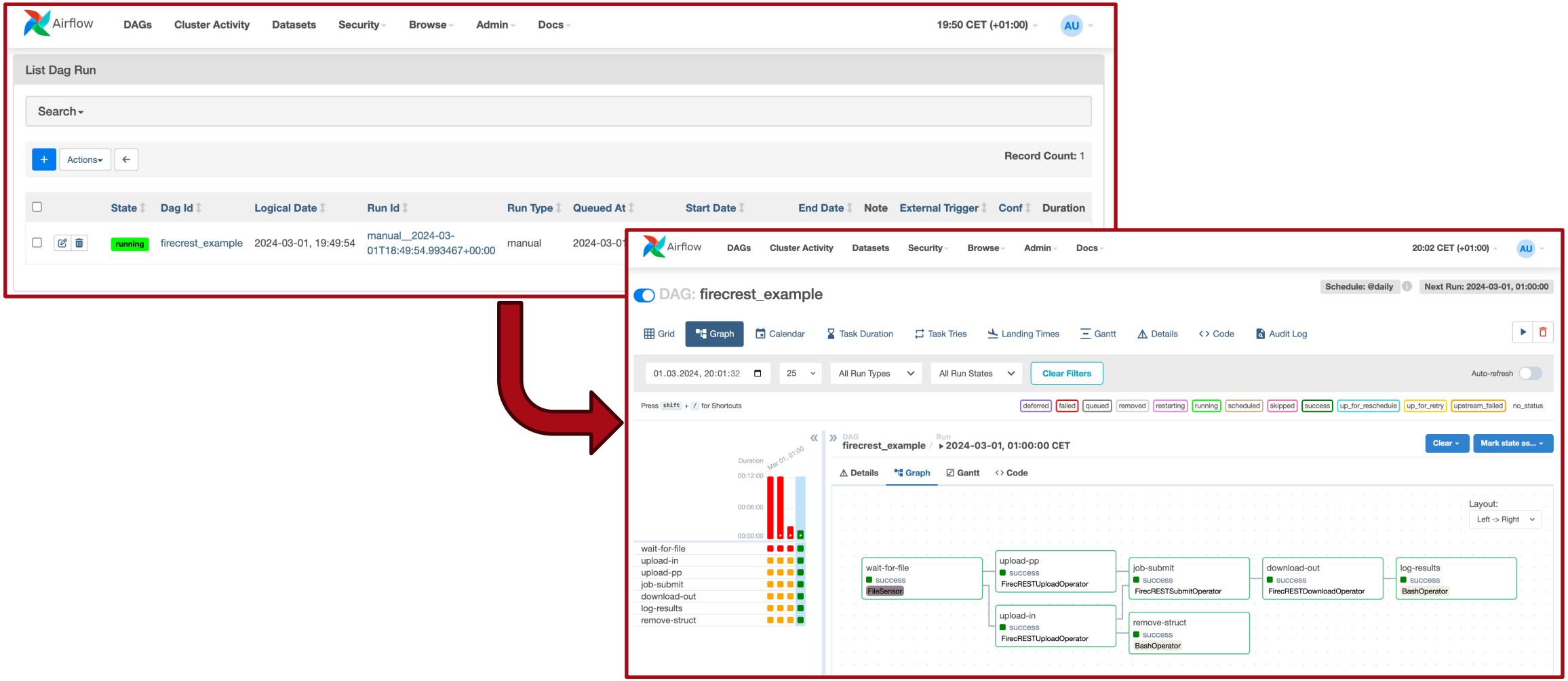
- Workflow Orchestrator
  - DAG example (`firecrest-airflow-dag.py`)

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

```python
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

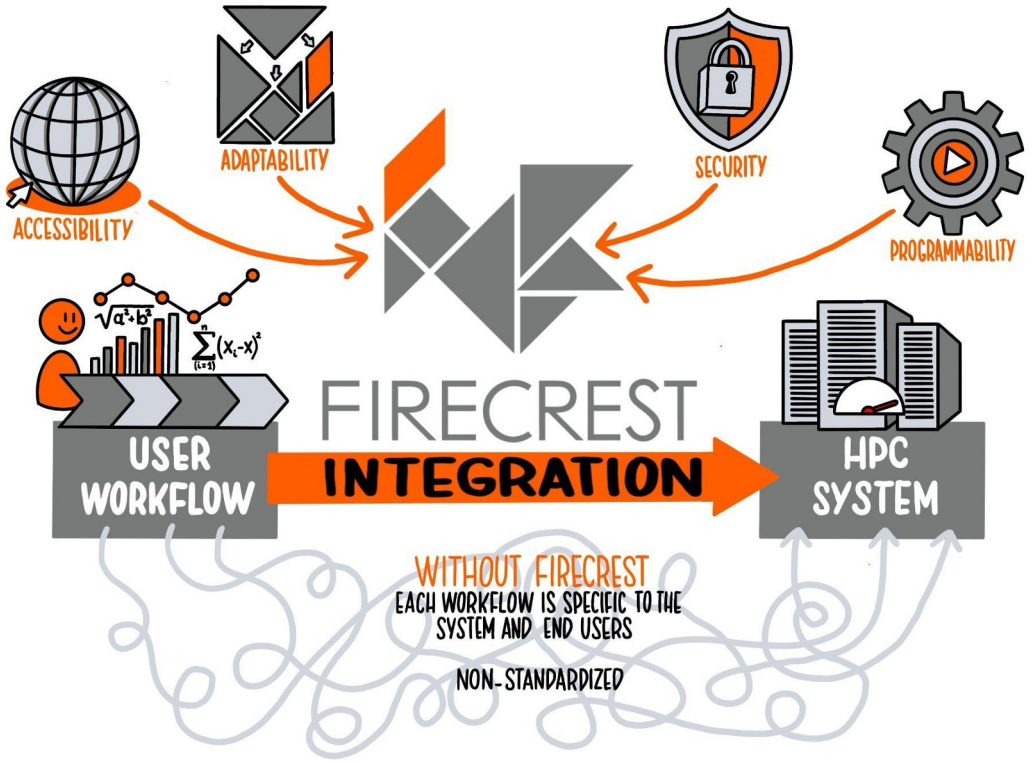
- Workflow Orchestrator



Conclusions

Conclusions

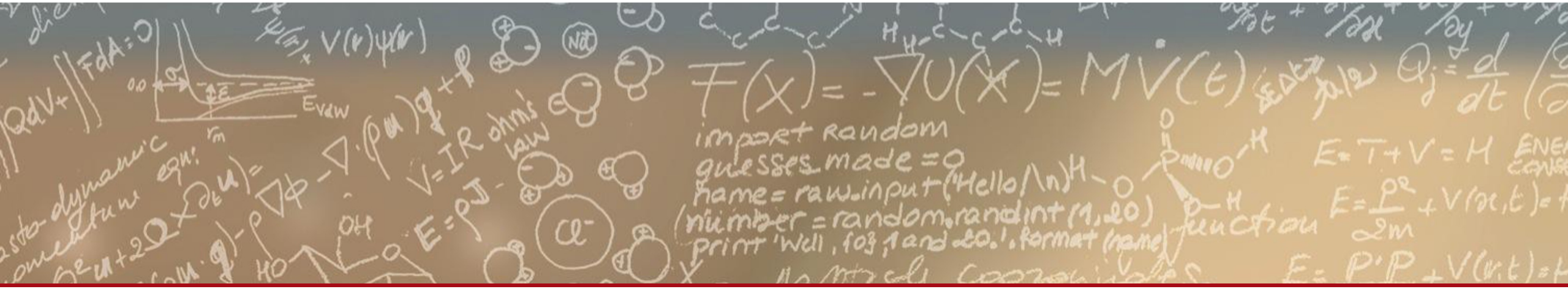
- FirecREST facilitates the integration of complex services for HPC, which allows the scientific and academic communities to deploy their own services
- Reduces the intervention of the HPC staff in terms of maintenance and support for users and their workflows
- Provides a standard service management layer for HPC and allows workflow execution across supercomputing facilities



Conclusions

- 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



Thank you for your attention.