

# Automate your workflows on HPC with FirecREST API

Juan Dorsch (jdorsch[at]cscs.ch)  
CSCS - Swiss National Supercomputing Centre  
ETH-Zürich

September 2, 2024  
CSCS UserLab Day  
Luzern, Switzerland



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH**zürich

# Motivation

---

# Motivation

- As cloud technology and HPC evolve there is an increasing need from the user community to enable sophisticated services on the cloud to interface HPC resources
- Use cases such as CI/CD Pipelines, Workflow Orchestrators, Web Portals, are just few examples of what users can easily setup on a cloud provider but that present issues when interfacing HPC
- This shows the need of a programmatic layer on the HPC facility that abstracts and standardizes access to HPC resources



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH**zürich

# Introducing FirecREST

---

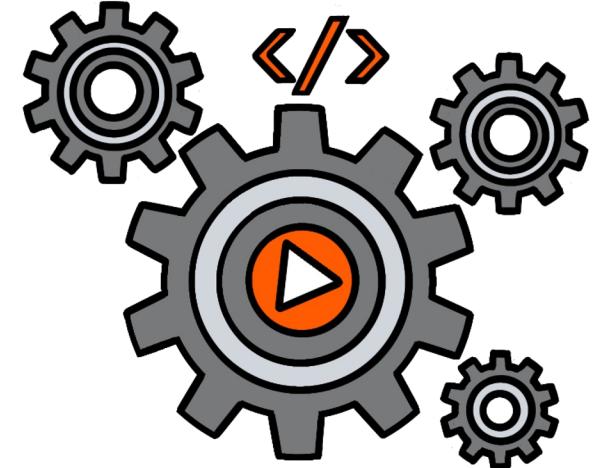
# FirecREST in a nutshell

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



# FirecREST 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



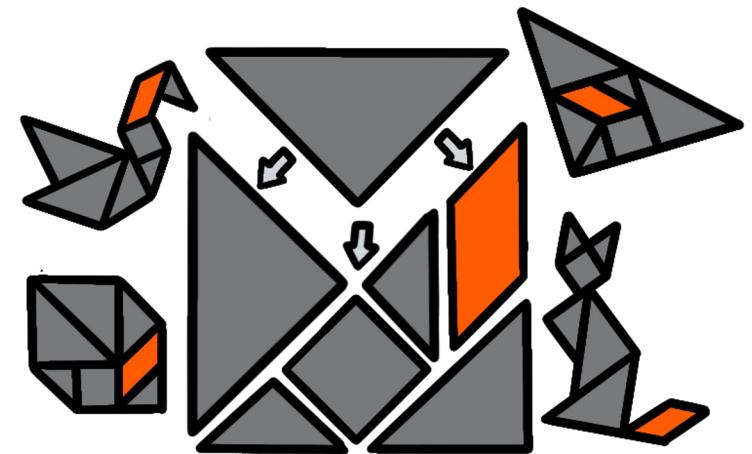
# FirecREST in a nutshell

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





**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH**zürich

# FirecREST at CSCS

---

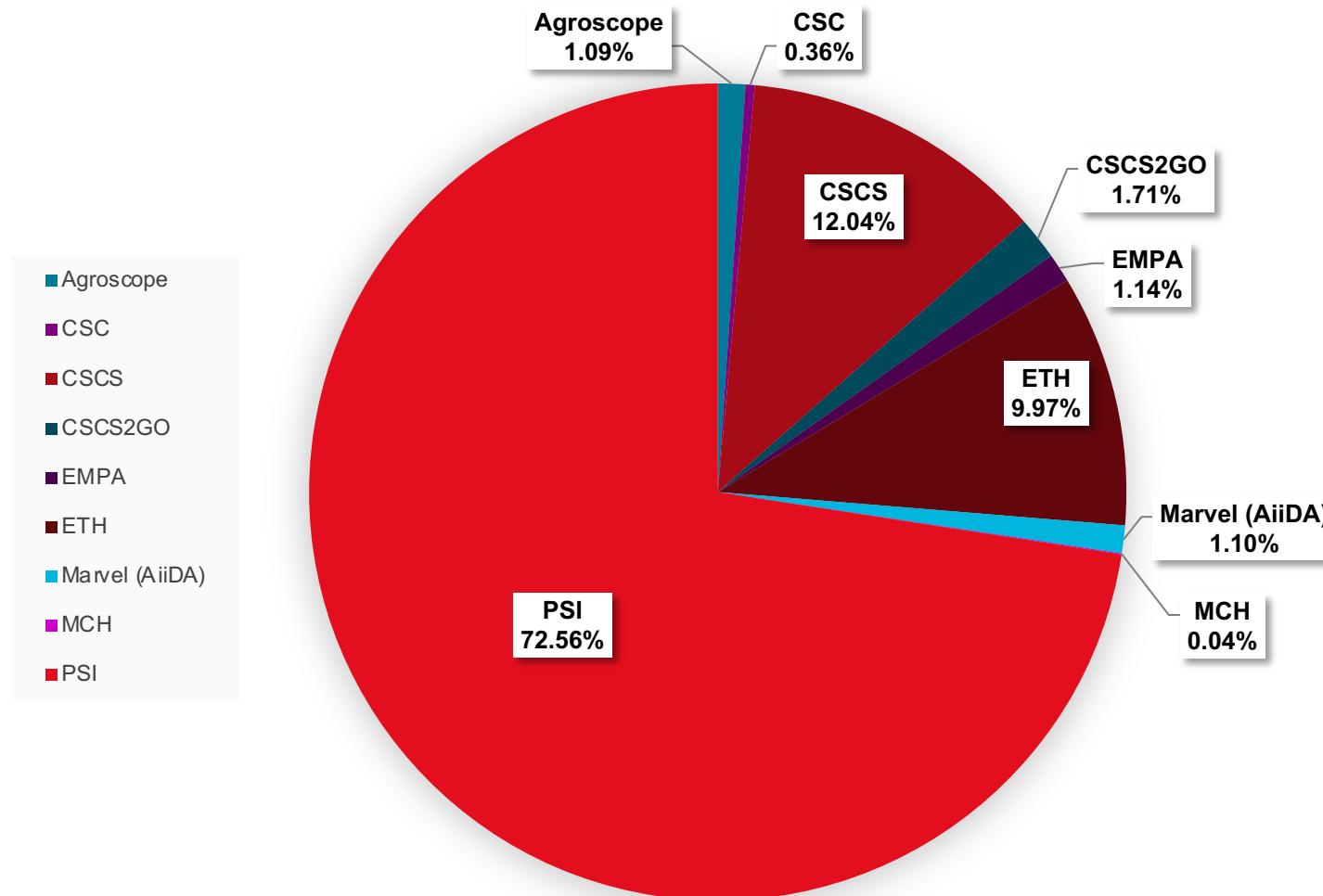
# The FirecREST API

- FirecREST API specification: <https://firecrest-api.csrs.ch>

<b>Status</b> Status information of infrastructure and services.	<b>Compute</b> Non-blocking calls to workload manager to submit and query jobs. This request.	<b>Storage</b> Non-blocking calls to high-performance storage services. The service responds with a temporary URL.
<b>GET</b> /status/services List of services	<b>POST</b> /compute/jobs/upload Submit Job by uploading a local sbatch file	<b>POST</b> /storage/xfer-internal/rsync rsync
<b>GET</b> /status/services/{servicename} Get service information	<b>POST</b> /compute/jobs/path Submit Job by a given remote sbatch file	<b>POST</b> /storage/xfer-internal/mv move (rename) files
<b>GET</b> /status/systems List of systems	<b>GET</b> /compute/jobs Retrieves information from all jobs	<b>POST</b> /storage/xfer-internal/cp copy files and directories
<b>GET</b> /status/systems/{machinename} Get system information	<b>GET</b> /compute/jobs/{jobid} Retrieves job information	<b>POST</b> /storage/xfer-internal/rm remove files or directories
<b>GET</b> /status/parameters Basic system utilities. All calls are blocking and low-latency operations, mostly synchronous.	<b>DELETE</b> /compute/jobs/{jobid} Delete Job	<b>POST</b> /storage/xfer-external/upload Upload a file
<b>GET</b> /utilities/ls List directory contents	<b>GET</b> /compute/acct Job account information	<b>POST</b> /storage/xfer-external/download Download a file
<b>POST</b> /utilities/mkdir Creates a directory		<b>POST</b> /storage/xfer-external/invalidate Invalidate temporary URL
<b>PUT</b> /utilities/rename Rename/move a file, directory, or symlink		
<b>PUT</b> /utilities/chmod Change file mode bits		
<b>PUT</b> /utilities/chown Change file owner and group		
<b>POST</b> /utilities/copy Copy file from a filesystem path to another		
<b>GET</b> /utilities/file determine file type		
<b>GET</b> /utilities/head Prints first part of a file		
<b>GET</b> /utilities/stat determines the status of a file		

# FirecREST usage per Project

- Data from last 2 years (systems: **daint** and **eiger**)



# What's coming: FirecREST on Alps



- HPC Platform

- `https://api.cscs.ch/hpc/firecrest/v1 -H "X-Machine-Name: daint"`
- `https://api.cscs.ch/hpc/firecrest/v1 -H "X-Machine-Name: eiger"`



- Machine Learning Platform

- `https://api.cscs.ch/ml/firecrest/v1 -H "X-Machine-Name: bristen"`
- `https://api.cscs.ch/ml/firecrest/v1 -H "X-Machine-Name: clariden"`



- Weather & Climate Platform

- `https://api.cscs.ch/wc/firecrest/v1 -H "X-Machine-Name: santis"`



- MeteoSwiss Platform (only accessible from MCH network)

- `https://api.cscs.ch/mch/firecrest/v1 -H "X-Machine-Name: balfrin"`

# CSCS Developer Portal

- Used for subscribing to FirecREST Platform APIs

[developer.cscs.ch](https://developer.cscs.ch)

The screenshot shows the CSCS Developer Portal interface. At the top, there is a navigation bar with the CSCS logo, the text "Centro Svizzero di Calcolo Scientifico" and "Swiss National Supercomputing Centre", and tabs for "APIs" (which is highlighted in red), "Applications", "Production" (with a dropdown arrow), and a search bar. Below the navigation bar, there are three API cards, each featuring the FIRECREST logo (a stylized orange and grey mountain peak icon). The first card is for "FIRECREST for HPC platform", the second for "FIRECREST for MCH", and the third for "FIRECREST for ML Platform". Each card provides details about the API: version (v2 or v1), endpoint (/hpc/firecrest or /mch/firecrest), context, a 5-star rating, and user count (0.0/5.0 (0 users)).

Platform	API Name	Version	Endpoint	Context	Rating	Users
HPC	FirecREST-HPC	v2	/hpc/firecrest	Version Context	★★★★★	0.0/5.0 (0 users)
MCH	FirecREST-MCH	v1	/mch/firecrest	Version Context	★★★★★	0.0/5.0 (0 users)
ML	FirecREST-ML	v1	/ml/firecrest	Version Context	★★★★★	0.0/5.0 (0 users)

# CSCS Developer Portal

- List of your applications

The screenshot shows the CSCS Developer Portal interface. At the top, there is a navigation bar with the CSCS logo, a search bar labeled "Search APIs", and a dropdown menu set to "Production". The "Applications" tab is highlighted with a red box. On the right, a user profile for "JDORSCH" is visible.

The main content area is titled "Applications" and features a "ADD NEW APPLICATION" button. A descriptive text below the button states: "An application is a logical collection of APIs. Applications allow you to use a single access token to invoke a collection of APIs and to subscribe to one API multiple times and allows unlimited access by default."

A table lists the following applications:

Name	Owner	Policy	Workflow Status	Subscriptions	Actions
DefaultApplication	jdorsch	50PerMin	ACTIVE	4	
firecrest-client	jdorsch	10PerMin	ACTIVE	2	
firecrest-mch	jdorsch	10PerMin	ACTIVE	2	
MyCSCSApplication	jdorsch	50PerMin	ACTIVE	1	

# CSCS Developer Portal

- Create a new application

The screenshot shows the CSCS Developer Portal interface. At the top, there is a navigation bar with the CSCS logo, menu items for APIs and Applications, a dropdown for Production, a search bar for APIs, and a user profile for JDORSCH. Below the navigation bar, the main content area is titled "Create an application". A sub-instruction says: "Create an application providing name and quota parameters. Description is optional. Required fields are marked with an asterisk (\*)". The "Application Name\*" field contains "MyNewApplication". Below it, a note says: "Enter a name to identify the Application. You will be able to pick this application when subscribing to APIs". A dropdown menu for "Shared Quota for Application Tokens\*" shows options: "10PerMin" (selected), "20PerMin", and "50PerMin". A text area for "Application Description" contains "This is my new application for FirecREST". A character counter "(472) characters remaining" is shown below the description. At the bottom, there are "SAVE" and "CANCEL" buttons.

**Create an application**

Create an application providing name and quota parameters. Description is optional.  
Required fields are marked with an asterisk (\*)

Application Name\*

MyNewApplication

Enter a name to identify the Application. You will be able to pick this application when subscribing to APIs

Shared Quota for Application Tokens\*

10PerMin

20PerMin

50PerMin

Application Description

This is my new application for FirecREST

(472) characters remaining

SAVE CANCEL

# CSCS Developer Portal

- Subscribe your new application to one of FirecREST APIs

The screenshot shows the CSCS Developer Portal interface. At the top, there is a navigation bar with the CSCS logo, a search bar labeled "Search APIs", and a user profile for "JDORSCH". Below the navigation bar, the main content area displays an application named "MyNewApplication" with "0 Subscriptions". On the left, a sidebar lists "Overview", "Production Keys", "OAuth2 Tokens", "API Key", and "Subscriptions", with "Subscriptions" being the active tab. A modal window titled "Subscribe APIs" is open, showing a search bar with "FirecREST-ML" and a list of filtered APIs. The list includes "Name", "Version", and "Subscription Status" columns. For the entry "FirecREST-ML v1", the "Subscription Status" dropdown is open, showing options: "Bronze" (selected), "Gold", and "Silver". A "SUBSCRIBE" button is located next to the Gold option.

# CSCS Developer Portal

- You can migrate your client created on [oidc-dashboard-prod.cscs.ch](https://oidc-dashboard-prod.cscs.ch)

**Important:** if you don't migrate your existing client, it won't work on Alps!!!

The screenshot shows the CSCS Developer Portal interface. On the left, a sidebar lists 'Overview', 'Production Keys' (which is selected), 'OAuth2 Tokens', 'API Key', and 'Subscriptions'. The main area displays an application named 'MyNewApplication' with '0 Subscriptions'. Below this, under 'Production OAuth2 Keys', there's a section for 'Key and Secret' with a button labeled 'PROVIDE EXISTING OAUTH KEYS'. A red arrow points from the text 'Important: if you don't migrate your existing client, it won't work on Alps!!!' to this button. The 'Key Configuration' section includes fields for 'Token Endpoint' (set to <https://auth.cscs.ch/auth/realms/firecrest-clients/protocol/openid-connect/token>) and 'Revoke Endpoint' (set to <https://auth.cscs.ch/auth/realms/firecrest-clients/protocol/openid-connect/revoke>). A modal window titled 'Provide Existing OAuth Keys' is open, containing fields for 'Consumer Key' (set to 'firecrest-jdorsch-client02') and 'Consumer Secret' (redacted). The modal has 'CANCEL' and 'PROVIDE' buttons at the bottom.

# CSCS Developer Portal

- If you want to create new credentials for your application to interface FirecREST API

The screenshot shows the CSCS Developer Portal interface. A red box highlights the 'Consumer Key' field, which is set to `FIRECREST_CLIENT_ID`. Another red box highlights the 'Token Endpoint' field, which is set to `AUTH_TOKEN_URL`. A third red box highlights the 'Consumer Secret' field, which is set to `FIRECREST_CLIENT_SECRET`. The portal displays various configuration options for the application, including grant types (Client Credentials selected, Refresh Token unselected) and a callback URL.

Consumer Key = `FIRECREST_CLIENT_ID`

Token Endpoint = `AUTH_TOKEN_URL`

Consumer Secret = `FIRECREST_CLIENT_SECRET`

Key and Secret

Consumer Key

Consumer Key of the application

GENERATE ACCESS TOKEN

CURL TO GENERATE ACCESS TOKEN

Key Configurations

Token Endpoint

<https://auth.cscs.ch/auth/realms/firecrest-clients/protocol/openid-connect/token>

Revoke Endpoint

<https://auth.cscs.ch/auth/realms/firecrest-clients/protocol/openid-connect/revoke>

Grant Types

Client Credentials  Refresh Token

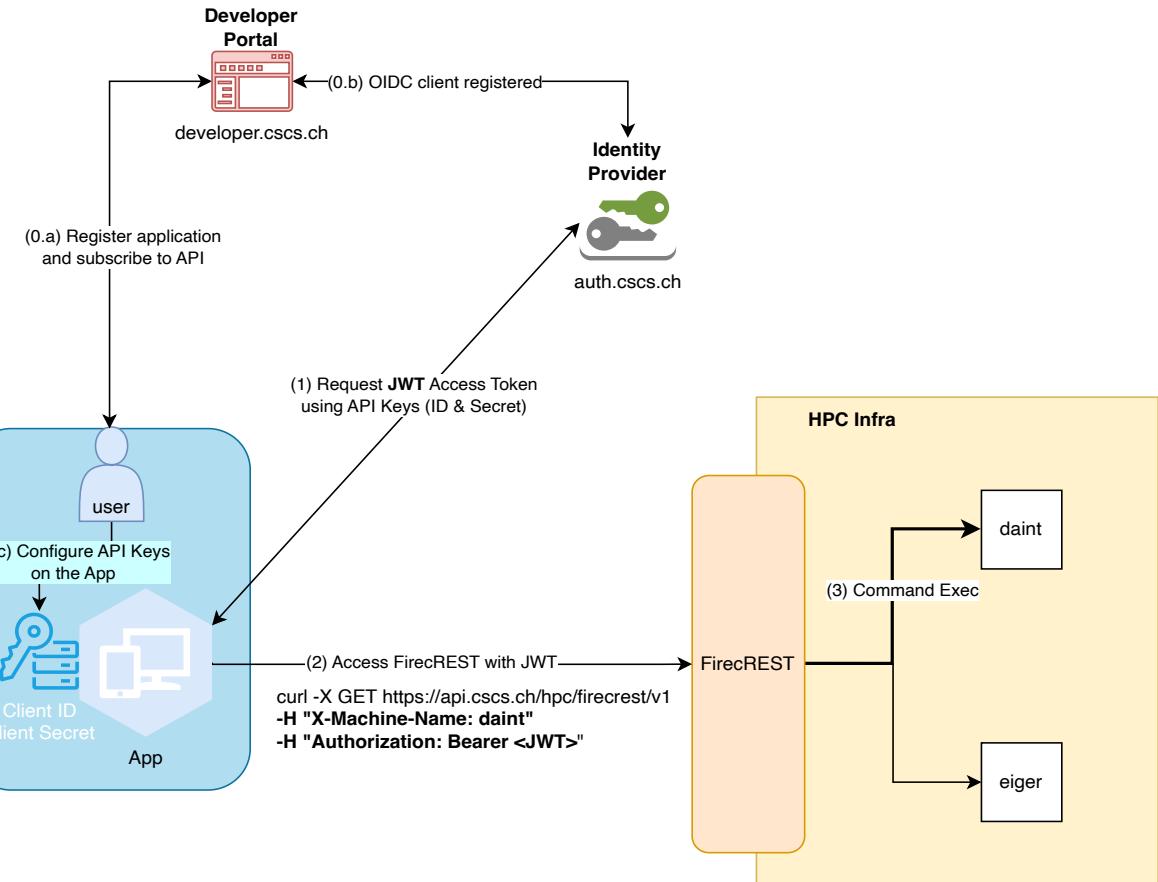
The application can use the following grant types to generate Access Tokens. Based on the application requirement, you can enable or disable grant types for this application.

Callback URL

Callback URL is a redirection URI in the client application which is used by the authorization server to send the client's user-agent (usually web browser) back after granting access.

# FirecREST IAM layer

- IAM relies on JWT from an IdP supporting Open ID Connect (OIDC)/OAuth2 standard
- The key pair obtained on Developer Portal is 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 simplifies 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 2.6.0

`pip install pyfirecrest`

Released: Jul 19, 2024

pyFirecrest is a python wrapper for FirecREST

**Navigation**

- Project description (selected)
- Release history
- Download files

---

**Verified details**  
These details have been verified by PyPI

**Maintainers**  
 eirinik

---

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

**Project links**

- Documentation
- Homepage
- Repository

**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 ft
```

# pyFirecREST Library

- From pyFirecREST version 1.3.0 also support a CLI for FirecREST

```
jdorsch@laptop ~$ export FIRECREST_URL=https://api.cscs.ch/hpc/firecrest/v1
jdorsch@laptop ~$ export FIRECREST_CLIENT_ID=<CONSUMER_KEY>
jdorsch@laptop ~$ export FIRECREST_CLIENT_SECRET=<CONSUMER_SECRET>
jdorsch@laptop ~$ export AUTH_TOKEN_URL=https://auth.cscs.ch/auth/realms/firecrest-clients/protocol/openid-connect/token
jdorsch@laptop ~$ 
jdorsch@laptop ~$ firecrest filesystems

          Status of filesystems for `daint` 

+-----+-----+-----+-----+-----+
| Name | Path           | Status code | Status    | Description      |
+-----+-----+-----+-----+-----+
| HOME | /users        | 200         | available | Home filesystem |
| SCRATCH | /capstor/scratch/cscs | 200         | available | Scratch filesystem |
+-----+-----+-----+-----+-----+
```

# pyFirecREST Library

- From pyFirecREST version 1.3.0 also support a CLI for FirecREST

```
jdorsch@laptop ~$ firecrest ls /capstor/scratch/cscs/jdorsch/newdir --system=daint
      Files in system `daint` and path `/capstor/scratch/cscs/jdorsch/newdir`

+-----+-----+-----+-----+-----+-----+-----+-----+
| Filename | Type | Group | Permissions | Size | User | Last modified | Link target |
+-----+-----+-----+-----+-----+-----+-----+-----+
| demo.ini | - | csstaff | rw-r----+ | 782 | jdorsch | 2023-12-19T10:02:29 | |
| demo.yaml | - | csstaff | rw-r----+ | 1666 | jdorsch | 2023-12-20T13:56:01 | |
| README.md | - | csstaff | rw-r----+ | 2564 | jdorsch | 2023-12-20T09:27:08 | |
| testsbatch_todi.sh | - | csstaff | rw-r----+ | 668 | jdorsch | 2023-12-19T17:44:17 | |
| testsbatch_todi2.sh | - | csstaff | rw-r----+ | 582 | jdorsch | 2023-12-19T09:39:24 | |
| uploaded_file1.tar | - | csstaff | rw-r----+ | 180788580 | jdorsch | 2023-12-18T16:36:39 | |
| uploaded_file2.tar | - | csstaff | rw-r----+ | 180788580 | jdorsch | 2023-12-13T14:35:41 | |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

```
jdorsch@laptop ~$ firecrest submit /Users/jdorsch/tests/sbatch/testsbatch_daint.sh --system=daint
{
    'job_data_err': '',
    'job_data_out': '',
    'job_file': '/capstor/scratch/cscs/jdorsch/firecrest/5d73fec9d0a03596caff2c153f5b3931/testsbatch_daint.sh',
    'job_file_err': '/capstor/scratch/cscs/jdorsch/firecrest/5d73fec9d0a03596caff2c153f5b3931/slurm-358188.out',
    'job_file_out': '/capstor/scratch/cscs/jdorsch/firecrest/5d73fec9d0a03596caff2c153f5b3931/slurm-358188.out',
    'job_info_extra': 'Job info returned successfully',
    'jobid': 358188,
    'result': 'Job submitted',
    'firecrest_taskid': '5d73fec9d0a03596caff2c153f5b3931'
}

jdorsch@laptop ~$ firecrest poll --system=daint
      Accounting data for jobs

+-----+-----+-----+-----+-----+-----+-----+-----+
| Job ID | Name | Nodelist | Nodes | Partition | Start time | State | Time | Time left | User |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 358188 | f7t_test | nid006592 | 1 | normal | 2024-08-28T09:34:41 | RUNNING | 00:38:24 | Unknown | jdorsch |
+-----+-----+-----+-----+-----+-----+-----+-----+
```



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH**zürich

# 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 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 stacks
    - Solve authentication and connectivity issues

# Use Cases

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

```
# 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

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

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

    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

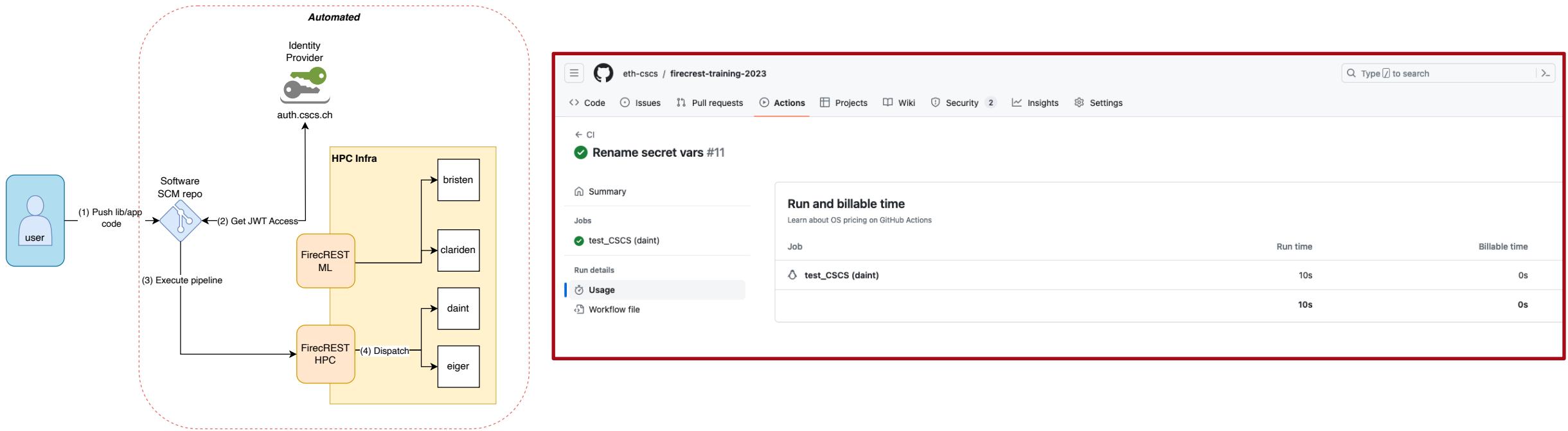
      - name: Run testing script
        env:
          FIRECREST_CLIENT_ID: ${{ secrets.F7T_CLIENT_ID }}
          FIRECREST_CLIENT_SECRET: ${{ secrets.F7T_CLIENT_SECRET }}
          FIRECREST_URL: ${{ secrets.F7T_URL }}
          AUTH_TOKEN_URL: ${{ secrets.F7T_TOKEN_URL }}
        run: ci/ci_script.py --system=${{ 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

- 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 detaching 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': 'daint',
            'descr': 'Daint-Alps vCluster',
            '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://api.cscs.ch/hpc/firecrest/v1" ## <-- configuring FirecREST-scheduler
    AUTH_TOKEN_URL: "https://auth.cscs.ch/auth/realm/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

The screenshot shows a GitLab pipeline interface. On the left, there's a sidebar with project navigation: 'Project' (clariden-testing-poc), 'Pinned' (Issues: 0, Merge requests: 0), 'Pipelines' (selected), 'Manage', and 'Plan'. The main area displays two pipeline runs under 'All' status. Both runs are labeled 'Passed' with a green checkmark icon. The first run was created by user #475205 and has a commit hash of f411c0e7, with the latest stage being green. The second run was created by user #474565 and has a commit hash of 457335e5, also with a green checkmark for the latest stage. A red border highlights the pipeline table.

```
518 [ FAIL ] (134/137) MemoryOverconsumptionMpich /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/stage/2024-03-05_04-06-05/clariden/nvgpu/PrgEnv-gnu/MemoryOverconsumptionMpich'
522 [ FAIL ] (135/137) MemoryOverconsumptionMpich /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/stage/2024-03-05_04-06-05/clariden/nvgpu/PrgEnv-nvidia/MemoryOverconsumptionMpich'
526 [ OK ] (136/137) MemoryOverconsumptionMpich /6a7583af @clariden:amdgpu+PrgEnv-crav
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) MemoryOverconsumptionMpich /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 ] MemoryOverconsumptionMpich /6a7583af @clariden:nvgpu+PrgEnv-gnu
536 [ RUN ] MemoryOverconsumptionMpich /6a7583af @clariden:nvgpu+PrgEnv-nvidia
537 [ OK ] (1/2) MemoryOverconsumptionMpich /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 in 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

```
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", ...)

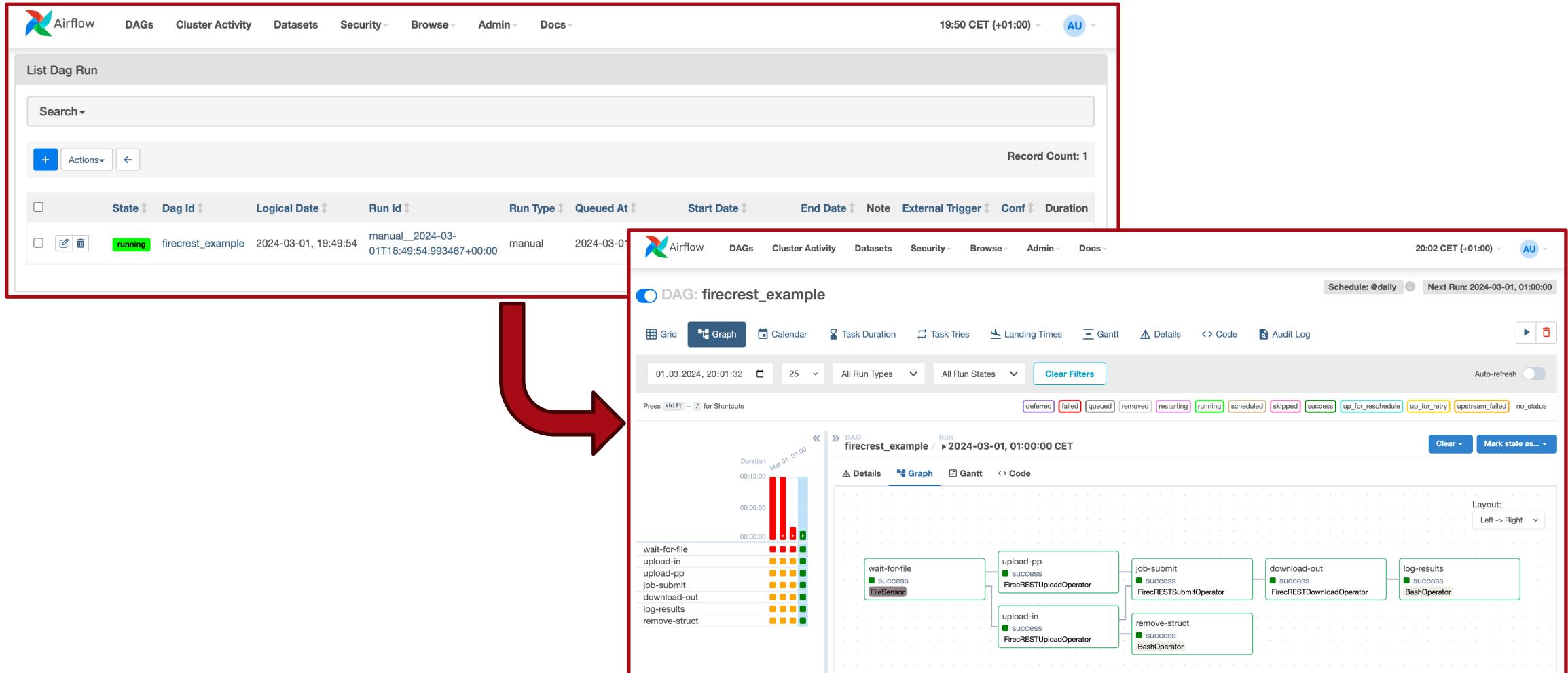
    download_task = FirecRESTDownloadOperator(task_id="download-out", ...)

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

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

# Use Cases

- Workflow Orchestrator



# 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 to 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

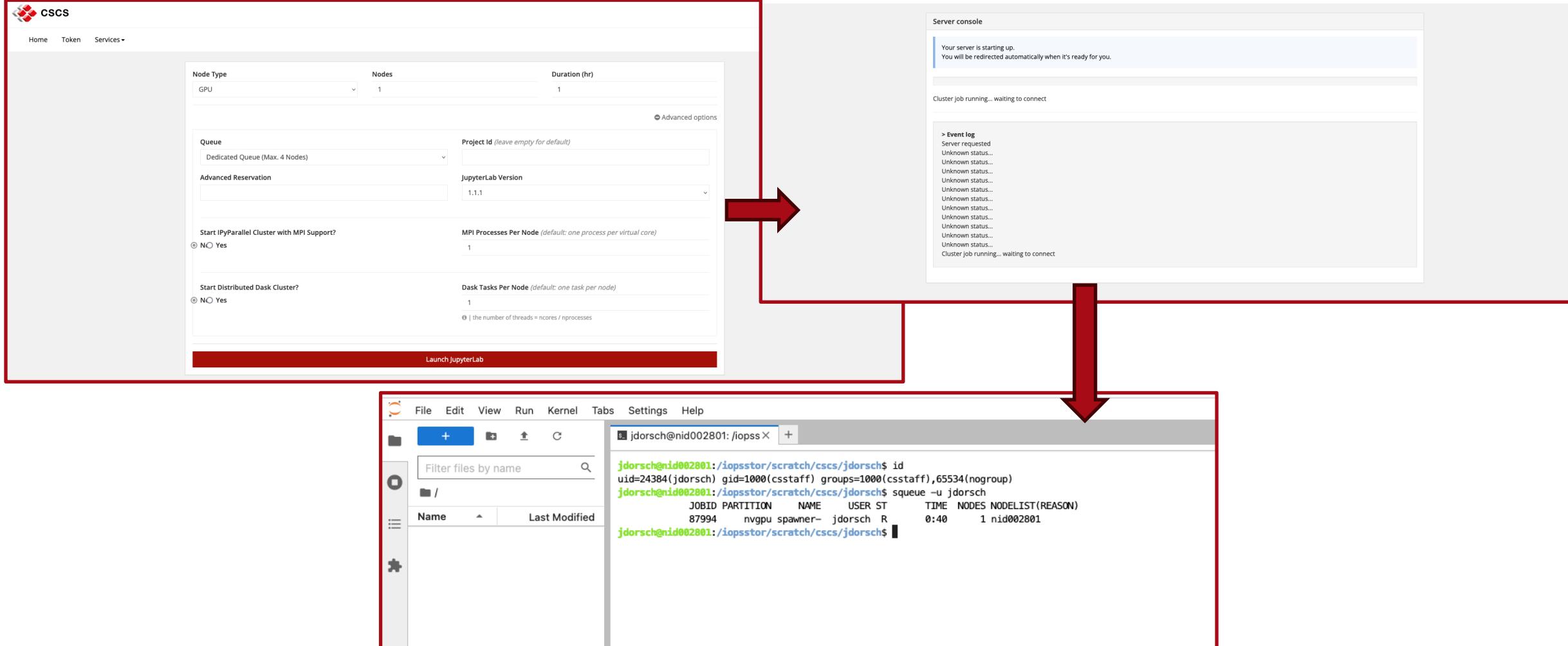
class FirecRESTSpawnerBase(Spawner):
    # Start Jupyter notebook
    def start(self):
        self.job = client.submit(self.host, script_str=script)

    # 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

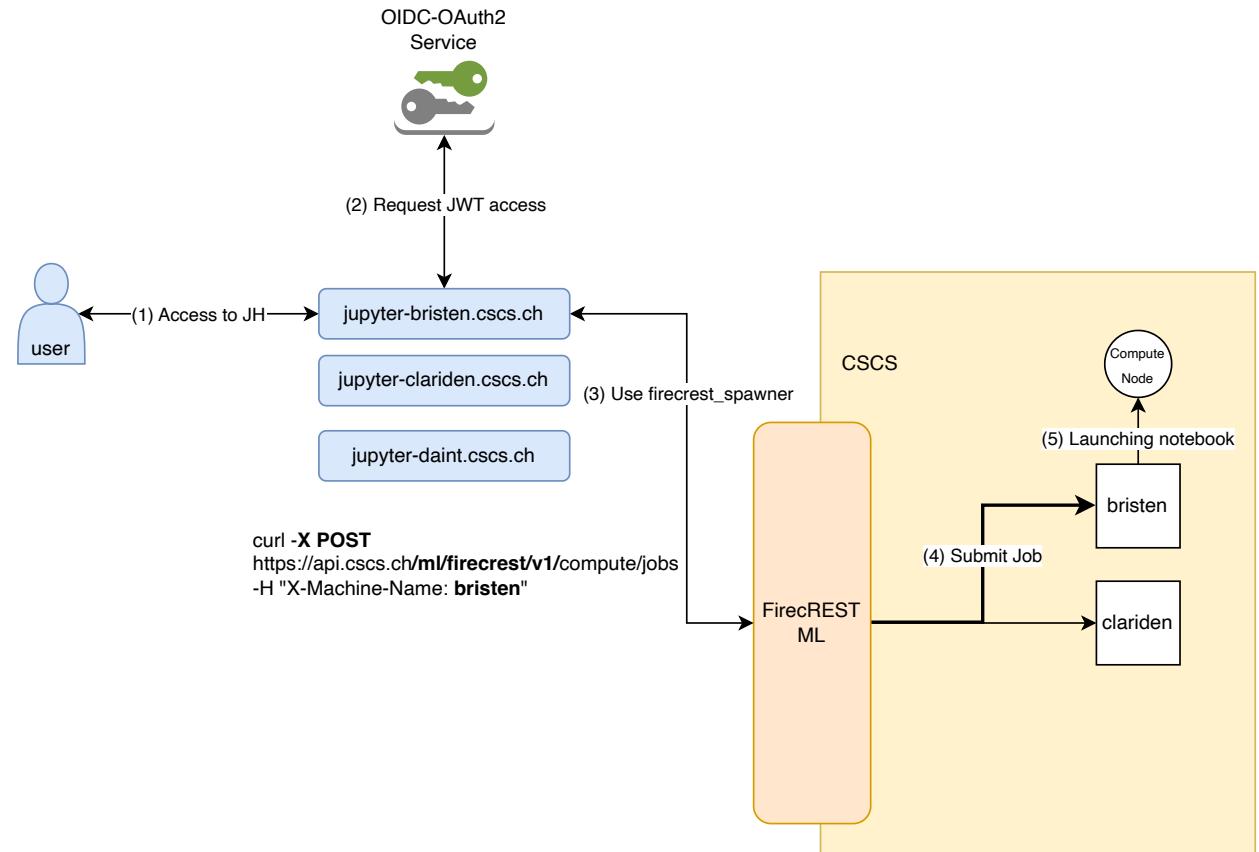
## • Interactive Computing



# Use Cases

- Interactive Computing

- Reduces the requirements 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

- Scientific Portals
  - Another example of the use of FirecREST is to create Graphical User Interface applications that allow users to interface filesystems, workload managers, and data transfer operations in a visual interface
  - At CSCS we've developed the FirecREST UI for the HPC Platform

[firecrest-ui-hpc.cscs.ch](http://firecrest-ui-hpc.cscs.ch)



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

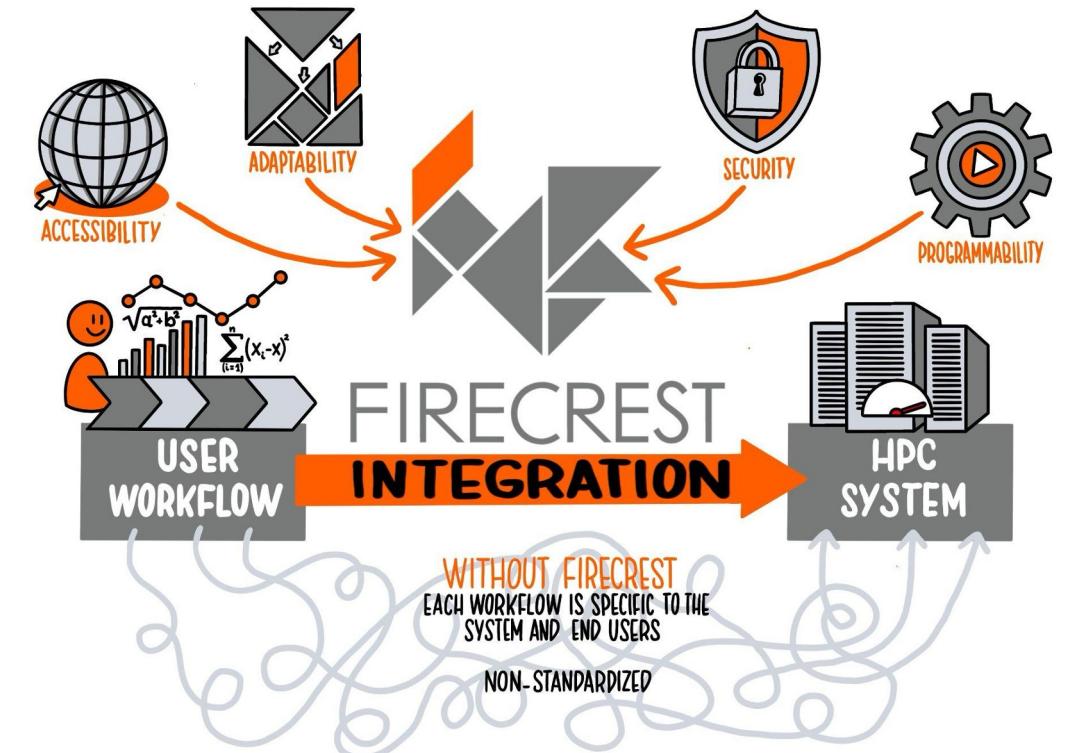
**ETH**zürich

# Conclusions

---

# Conclusions

- FirecREST provides a uniform and standard interface for workflows, which empowers you to create applications on the top of HPC
- Facilitates the integration of complex services for HPC, which allows scientific and academic communities to deploy their own services



# Links and references

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



# This the team that will support you

- Alejandro Dabin
- Eirini Koutsaniti
- Elia Palme
- Francesco Pagnamenta
- Ivano Bonesana
- Luca Curella
- Juan Dorsch
- Rafael Sarmiento
- Tim Robinson

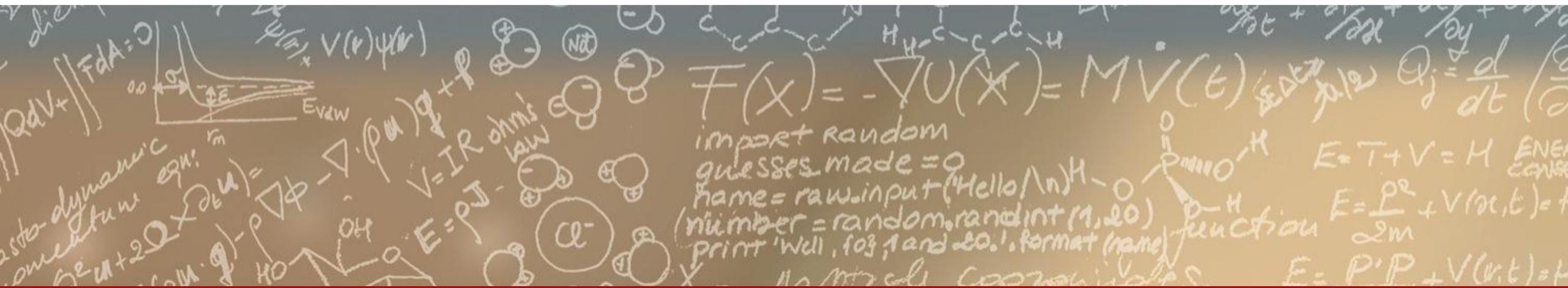




CSCS

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

ETH zürich



Thank you for your attention.