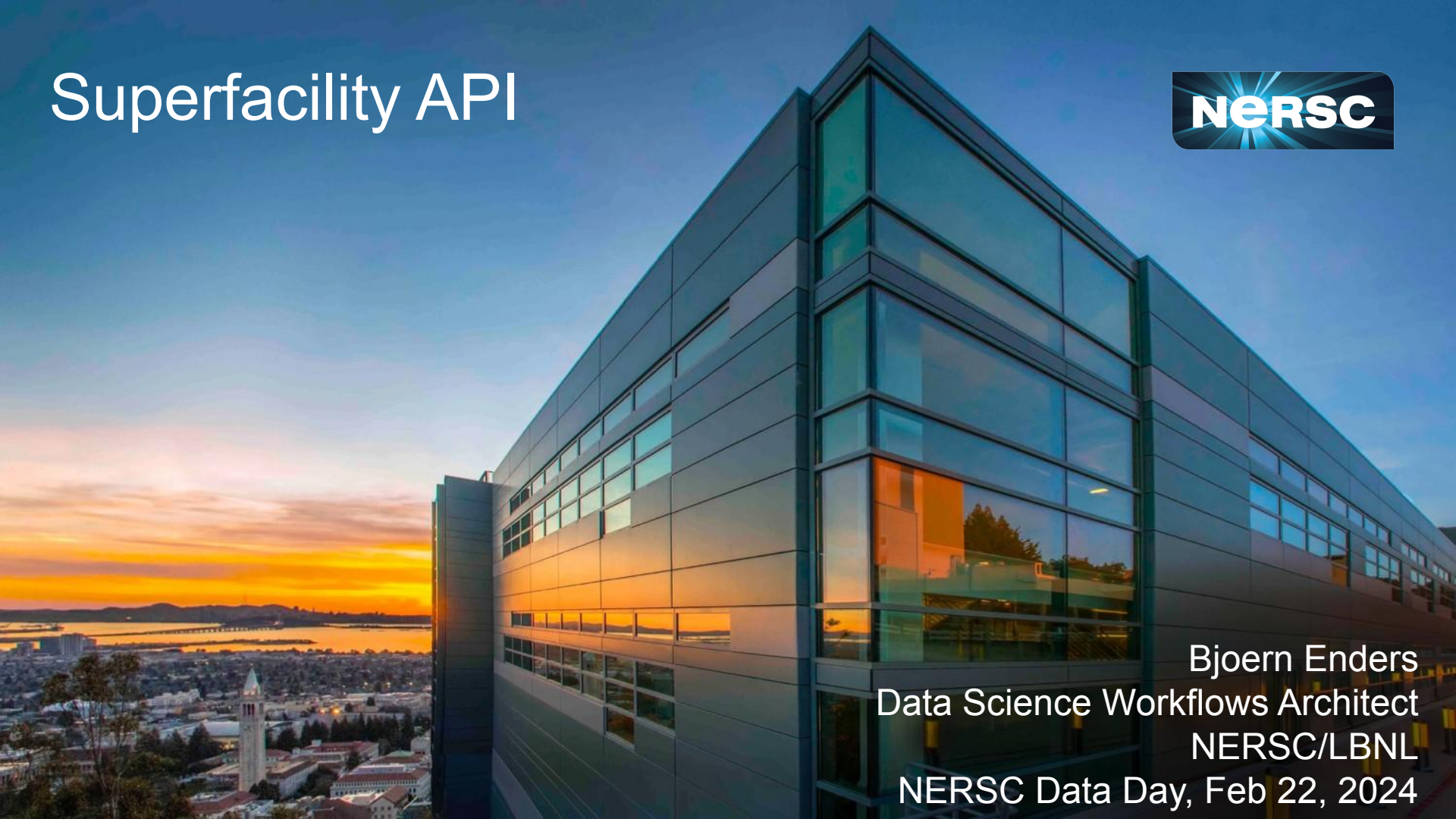
Superfacility API

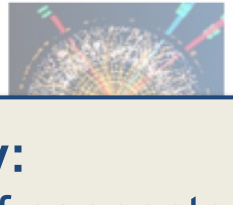


Bjoern Enders
Data Science Workflows Architect
NERSC/LBNL
NERSC Data Day, Feb 22, 2024

NERSC supports a large number of users and projects from DOE SC's experimental and observational facilities

Palomar Transient Factory Supernova

Planck Satelli
Cosmic Micro
Background
Radiation

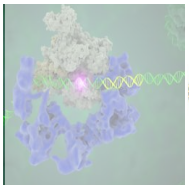


Dayabay Neutrinos

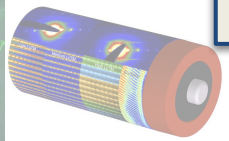
ALS Light Source

Superfacility:
Ecosystem of connected facilities, software and expertise to enable new modes of discovery

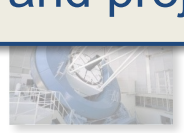
Superfacility API:
An API into NERSC to embed HPC into cross-facility workflows. It is also a general purpose API for all NERSC users and projects.



Cryo-EM

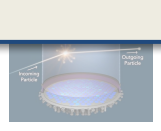


NCEM



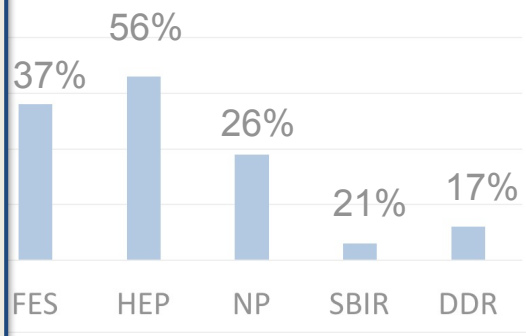
DESI

LSST-DESC



LZ

# of Projects Analyzing Experimental Data or Combining Modeling and Experimental Data by SC Office



~35% (235) of NERSC projects self identified as confirming the primary role of the project is to 1) analyze experimental data or; 2) create tools for experimental data analysis or; 3) combine experimental data with simulations and modeling

Model case

Experiments at ext. facilities use high frame rate 2D detectors for their science.

Hosting data & compute on site has become increasingly demanding.

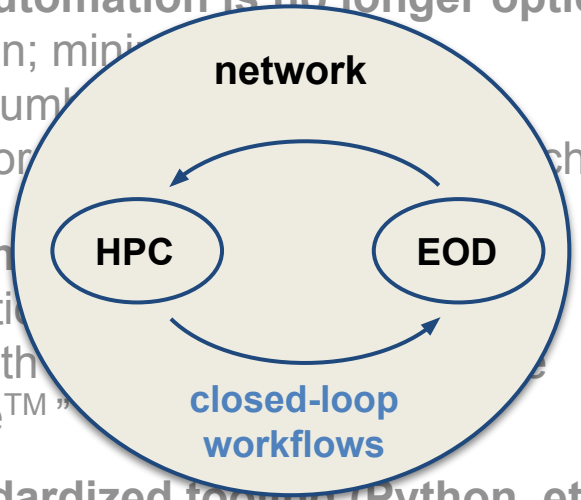
Requirements

- Planning (HPC as reliable partner)
  - machine-readable status
- Resiliency (needs failover)
  - compatible interfaces
- Realtime (can't wait in queue)
  - workflow endpoint
- Services (portals, data, db)

1. Plan / Check availability of NERSC resource for experiment.
   - check status / accounts
2. Get raw data to NERSC, when experiment is live.
   - move data
3. Start analysis job quasi synchronous with data
   - submit job / monitor job
4. Gather feedback, ideally immediate.
   - download / execute command
5. Move data and results to archive after analysis.
   - move data

Why an API?

- **Meets a critical need; automation is no longer optional**
  - Unattended operation; mini
  - Track/submit large numb
  - Interface with collabor chines
- **NERSC becomes "mach**
  - Enables easier creati
  - Allows integration with
    - " NERSC inside™ "
- **Less DIY: simpler, standardized tooling (Python, etc)**
  - Stable refactor target for established projects or easier on-ramp for new ones
  - Contribute to HPC interface standards for portability
  - Authentication and security models



Drivers:

- **Complex workflows**
- **Data-driven projects**
- **Real-time compute and streaming data from instruments**
- **Automation**

What is the API good for?

Vision: all NERSC interactions are callable to facilitate seamless, automated "NERSC inside" workflows without a human in the loop.

Endpoints

- /status
- /account
- /compute
- /storage
- /tasks
- /storage
- /utilities

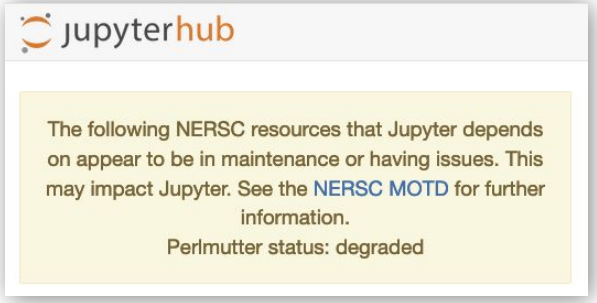
Example request

```
curl -X 'GET' \
  'https://api.nersc.gov/api/v1.2/ status/perlmutter' \
  -H 'accept: application/json'
```

Result:

```
{
  "name": "perlmutter",
  "full_name": "Perlmutter",
  "description": "System is active",
  "system_type": "compute",
  "notes": [],
  "status": "active",
  "updated_at": "2023-03-02T18:00:00-08:00"
}
```

API is used both internally at NERSC and externally for our users



Action	Manual steps	With SuperFacility API
Check status	Test SSH or ping specific services for status	Query the `/status` API endpoint if resources are active.
Submit job	SSH in and submit jobs with `sbatch ...`	Create jobs using POST calls from a script or Spin service to the `/compute` endpoint.
Monitor job	SSH in (again) and do `squeue | grep | sort | ...`	Consult the `/compute` and `/tasks` endpoints.
Plan ahead	Read the NERSC MOTD to see if any down time is planned	Query the `/status/outages/planned` API endpoint for planned outages
Move data	SSH in and run file transfer tools to move data	POST to the `/storage` API endpoint.
Check account	Log into "Iris" (our accounting web app) and check allocation account balance.	Query the `/account` API to get the same information.

Model use

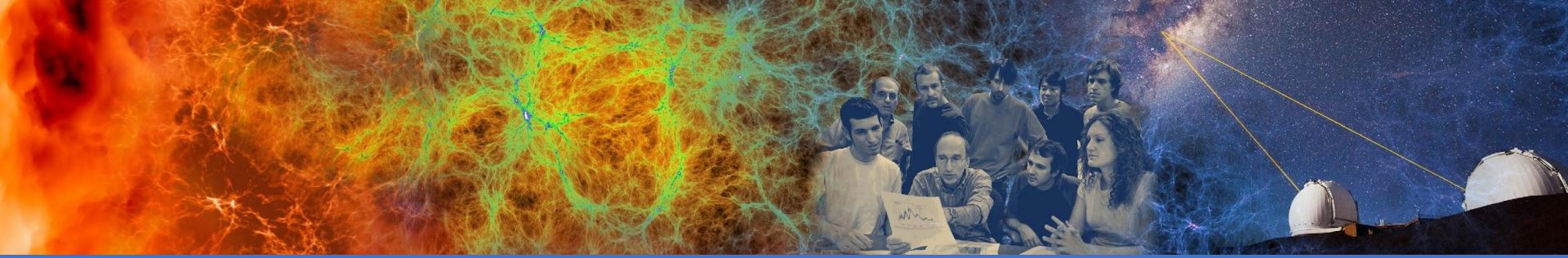
Experiments at ext. facilities use high frame rate 2D detectors for their science.

Hosting data & compute on site has become increasingly demanding.

Requirements

- Planning (HPC as reliable partner)
  - machine-readable status
- Resiliency (needs failover)
  - compatible interfaces
- Realtime (can't wait in queue)
  - workflow endpoint
- Services (portals, data, db)

1. Plan / Check availability of NERSC resource for experiment.
   - `/status (/reservations)`
2. Get data to NERSC, when experiment is live.
   - `/storage`
3. Start analysis job quasi synchronous with data
   - `/compute /tasks`
4. Gather feedback, ideally immediate.
   - `/utilities /storage`
5. Move data and results to archive after analysis.
   - `/storage`



The API

Projects using the SF API

~ 12M logged requests from May 2022 - Feb 2023

= one request every 2 sec

Since release in 2022 (Feb 2023 numbers)

- 27 non-staff users made clients
- members of 40 different non-staff projects.

Examples

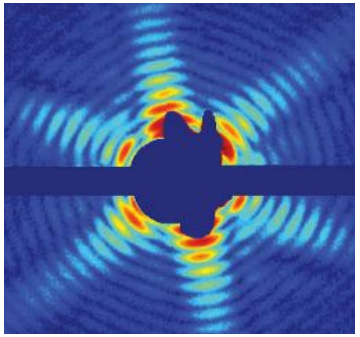
automated data processing for serial diffraction/scattering workflows

automated processing (HTC) of 4D STEM scan data with the Distiller app.

automated data processing between each of their plasma shots to reconstruct the shape and properties of the plasma.

SLAC NATIONAL ACCELERATOR LABORATORY

LCLS



Linac Coherent Light Source, a BES User Facility

- How does photosynthesis happen?
- How do drugs dock with proteins in our cells?
- Why do jet engines fail?

- By ~2024, 5% of LCLS experiments will produce >10PB data/year, and some need >128PFlops for real-time analysis

NERSC

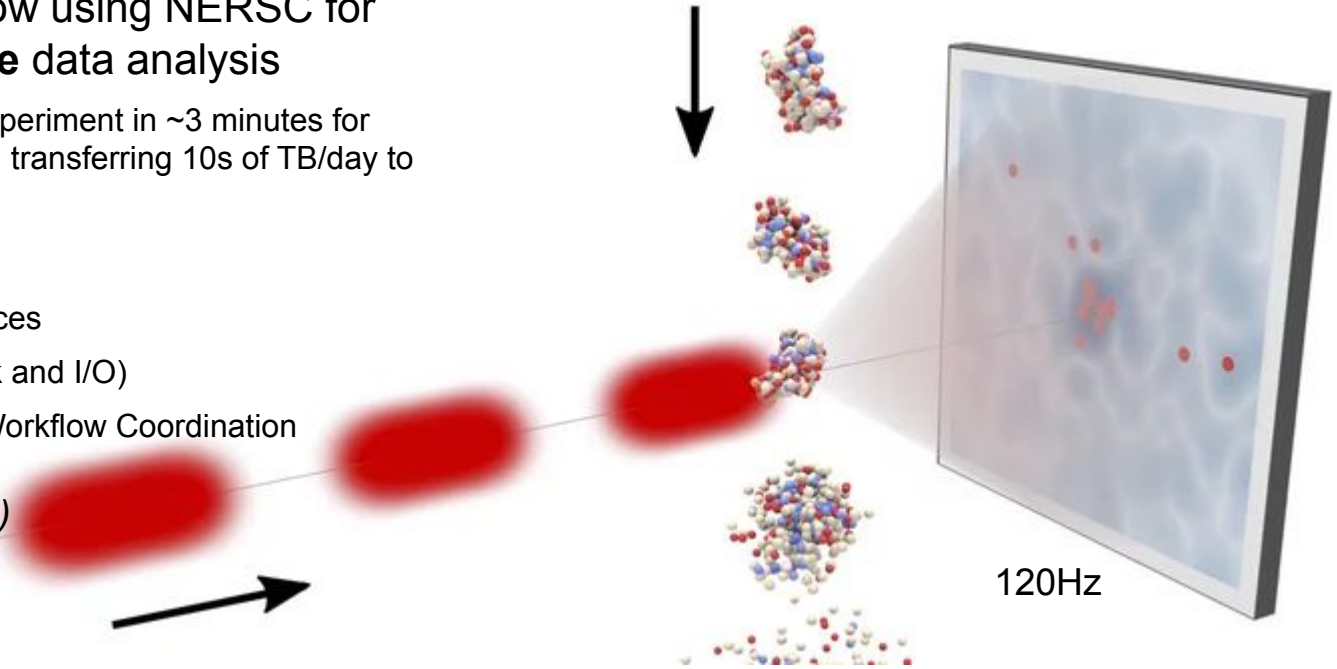
Linac Coherent Light Source - Serial Diffraction/Scattering Workflows

- LCLS experiments are now using NERSC for semi-automated **real-time** data analysis
  - Can analyze a 5 minute experiment in ~3 minutes for feedback to beamline staff, transferring 10s of TB/day to NERSC
- LCLS workflow requires
  - Urgent Computing Resources
  - High-Speed Data (Network and I/O)
  - Realtime Monitoring and Workflow Coordination

J. Blaschke et. al (ExaFEL team)

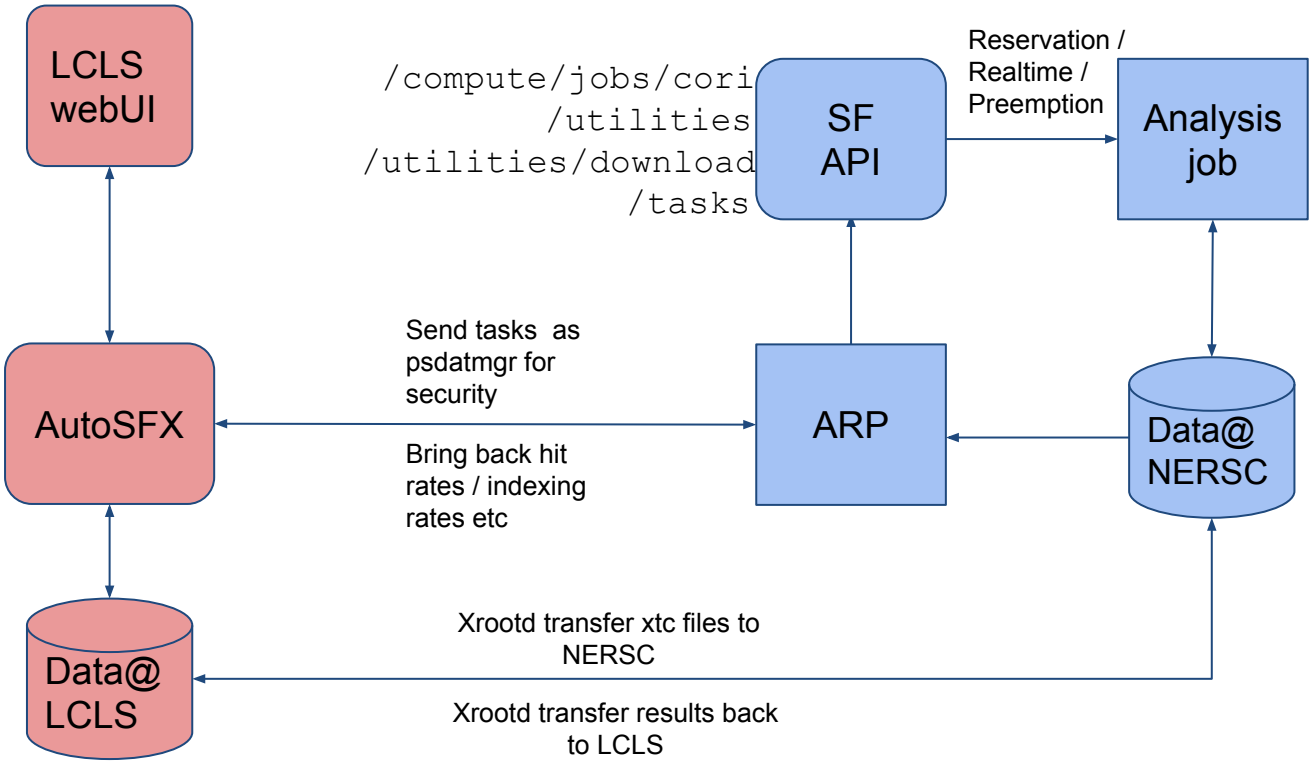
https://arxiv.org/abs/2106.11469

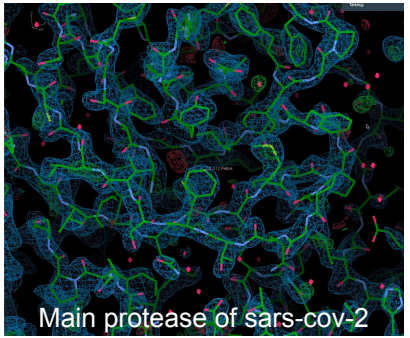
Winner of CUG21
Best Paper award



Source: https://www.mpibpc.mpg.de/9660732/SM-Ultrafast-XRay-Diffraction

SF API in LCLS workflow

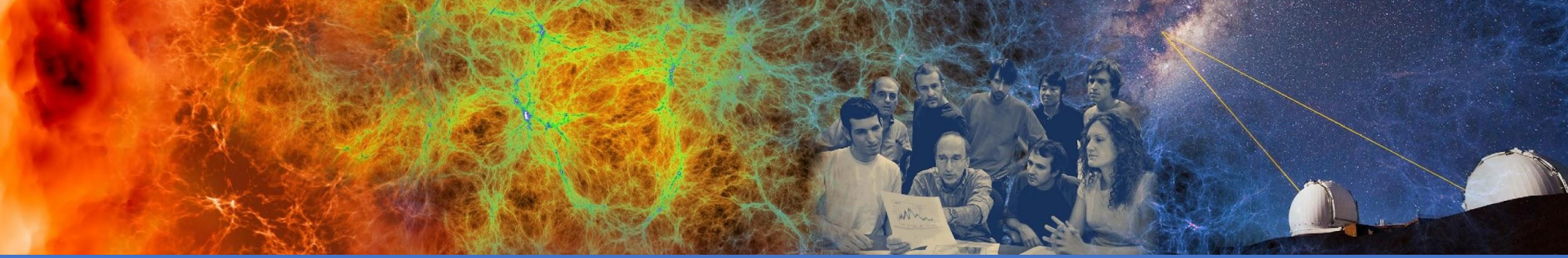




LCLS

SLAC NATIONAL ACCELERATOR LABORATORY

Adapted from Chuck Yoon



Technical overview

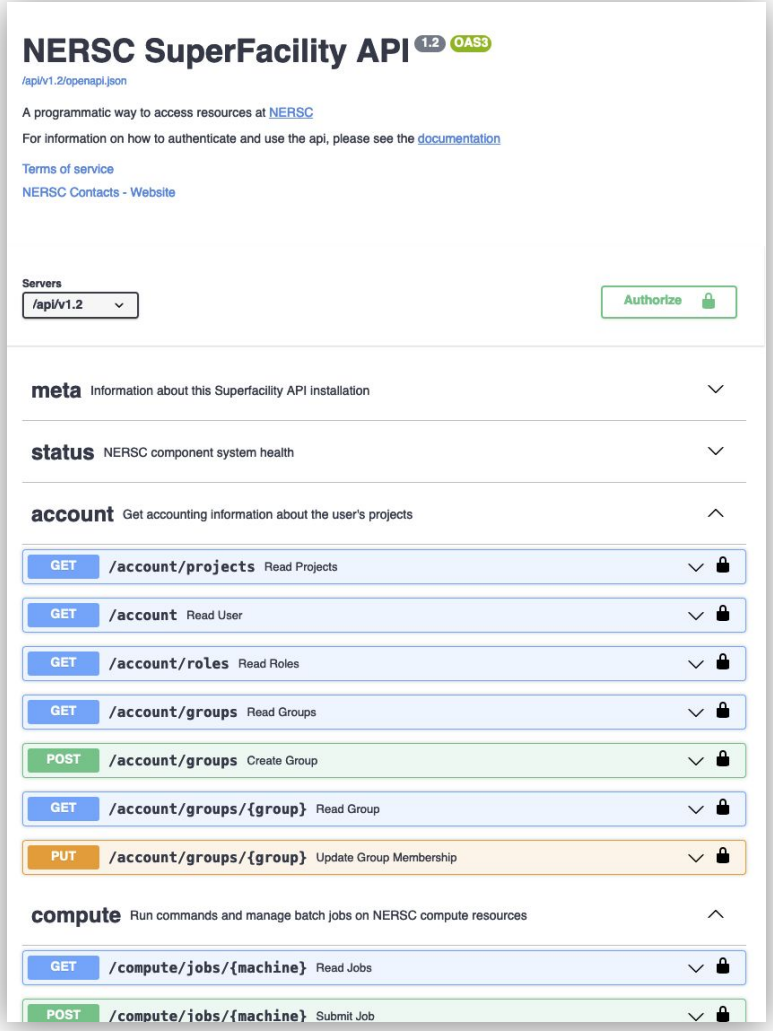
Interface documentation

https://api.nersc.gov/api/v1.2

- Interactive, up-to-date and self-documenting
- See endpoints, payloads, example code
- Works with any dev environment
- End user docs and examples: https://docs.nersc.gov/services/sfapi/

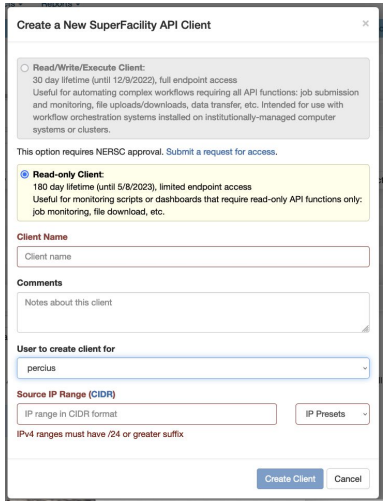
Authorization

- Some endpoints are public and don't need an access token (no lock icon)
- For authorized endpoints (with lock icon), click the "Authorize" button at the top page and paste the access token

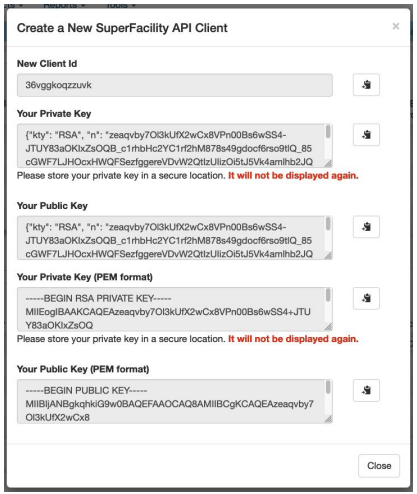


Users need to create a SF API client in Iris.

- Some endpoints (e.g. `/status`) are public and don't require authentication.
- For the authenticated part of the API, users need to create client credentials (via MFA in iris.nersc.gov) and exchange these for short-lived API access tokens.



1 Iris > Profile > Superfacility API clients > new client



2 Store oauth client credentials in safe place

jose/openssl authlib (Python)

client assertion (JWT)

3 Sign client assertion with private key

https://oidc.nersc.gov

access token (10 min)

api.nersc.gov/api/v1.2

4 Use access token for API

Live walkthrough

```
Poll 04:{'id': '2236', 'status': 'completed', 'result': '{"status": "ok", "jobid": "484275", "error": null}'}
{"status": "ok", "jobid": "484275", "error": null}
```

Check queue status, wait for job to complete.

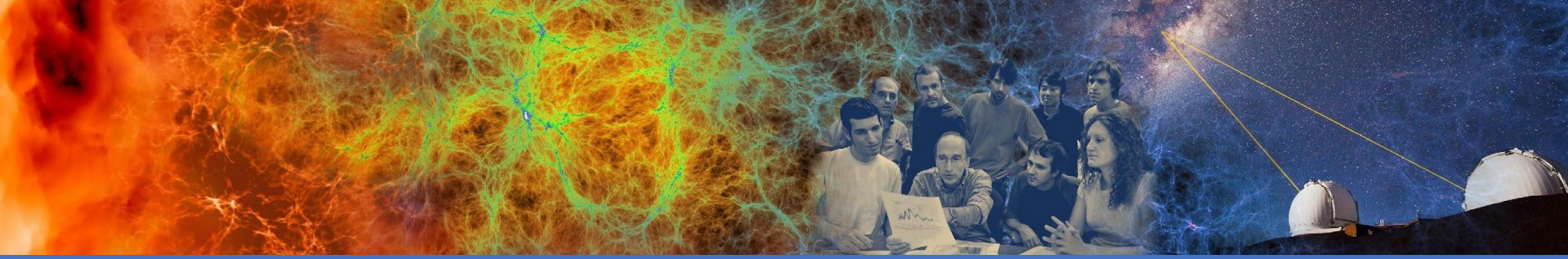
[338]:
```
#print(json.dumps(api("compute/jobs/perlmutter/"+jobid+"?sacct=true"), indent=2))
print(api("compute/jobs/perlmutter/"+jobid+"?sacct=true")['output']['state'])
```

```
RUNNING
```

Read from the slurm output file

[334]:
```
slurmfile = home+"/apidemo/slurm-"+jobid+".out"
response = api("utilities/command/perlmutter", { "executable": "tail -n 20 "+slurmfile })
if isinstance(response, Task):
    print(response.wait_for_result()['output'].strip())
```

```
Poll 03:{'id': '2237', 'status': 'completed', 'result': '{"status": "ok", "output": "  * subpix               : linear\\n  * update_object_first  : True\\n  * overlap_converge_... : 0.05\\n  * overlap_max_itera... : 10\\n  * probe_inertia        : 1e-09\\n  * object_inertia       : 0.0001\\n  * fourier_power_bound  : None\\n  * fourier_relax_factor : 0.05\\n  * obj_smooth_std       : None\\n  * clip_object          : None\\n  * probe_center_tol     : None\\n  * compute_log_likel... : True\\n  * probe_update_cuda... : False\\n  * object_update_cud... : True\\n  * fft_lib              : reikna\\n  * alpha                : 1.0\\n  * name                 : DM_pycuda\\n================================================================================\\nIteration #10 of DM_pycuda :: Time 1.249\\nErrors :: Fourier 5.94e+01, Photons 4.73e+01, Exit 3.51e+01\\n", "error": null}'}
{"status": "ok", "output": "  * subpix               : linear\n  * update_object_first  : True\n  * overlap_converge_... : 0.05\n  * overlap_max_itera... : 10\n  * probe_inertia        : 1e-09\n  * object_inertia       : 0.0001\n  * fourier_power_bound  : None\n  * fourier_relax_factor : 0.05\n  * obj_smooth_std       : None\n  * clip_object          : None\n  * probe_center_tol     : None\n  * compute_log_likel... : True\n  * probe_update_cuda... : False\n  * object_update_cud... : True\n  * fft_lib              : reikna\n  * alpha                : 1.0\n  * name                 : DM_pycuda\n================================================================================\nIteration #10 of DM_pycuda :: Time 1.249\nErrors :: Fourier 5.94e+01, Photons 4.73e+01, Exit 3.51e+01\n", "error": null}
* subpix               : linear
  * update_object_first  : True
  * overlap_converge_... : 0.05
  * overlap_max_itera... : 10
  * probe_inertia        : 1e-09
  * object_inertia       : 0.0001
  * fourier_power_bound  : None
  * fourier_relax_factor : 0.05
  * obj_smooth_std       : None
  * clip_object          : None
  * probe_center_tol     : None
  * compute_log_likel    : True
```



The sfapi_client Python library

Using Python to interact with the API

- The `sfapi_client` is a pythonic way to interact with the API
  - Many NERSC users are familiar with python already
- Uses pydantic to take API responses and make equivalent python objects
  - Can easily be updated with changes to the API specification
- Both an Asynchronous and Synchronous Client
  - Allows for a wider range of programming styles
  - More advanced users can interact Asynchronously
  - Still supports all (python) users
  - Easily place the client in existing tools

Using Python to interact with the API

- Code on Github
  - https://github.com/NERSC/sfapi_client
- Documentation Page
  - https://nersc.github.io/sfapi_client
- Nersc User Slack
  - https://www.nersc.gov/users/NUG/nersc-users-slack
  - #sfapi_client
- help.nersc.gov
  - Include `sfapi_client` in your title or description

sfapi_client library needs to be installed

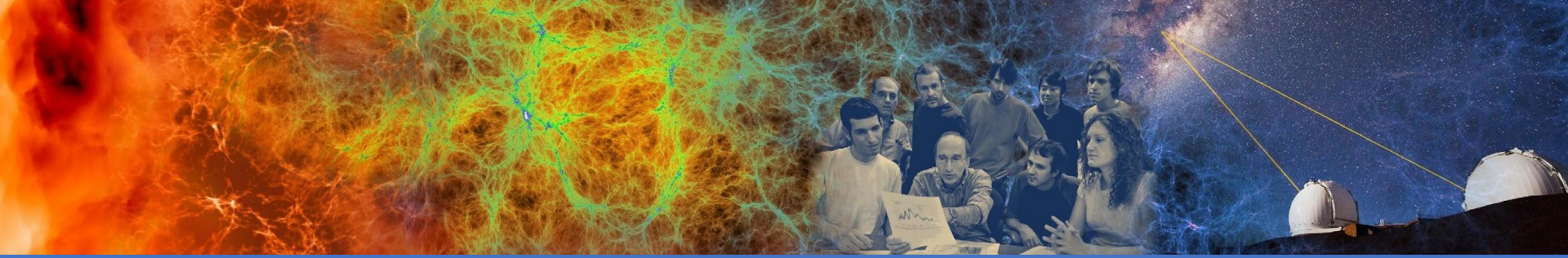
- First load the python module

```
module load python
```

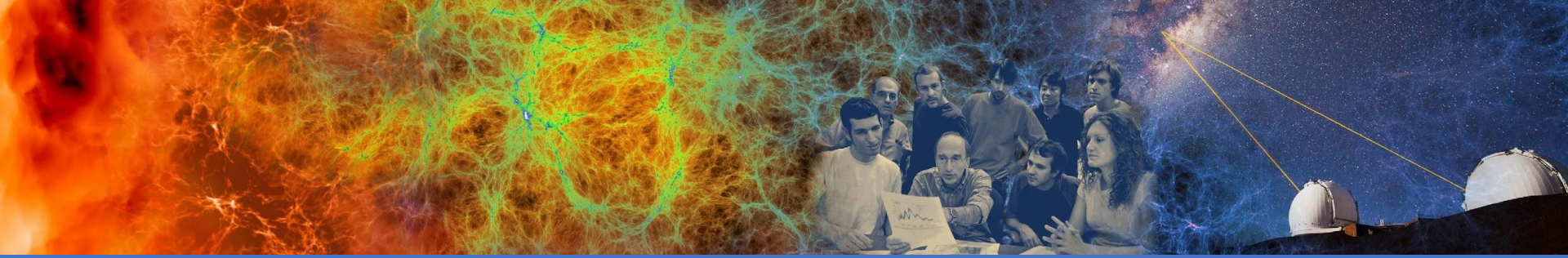
- Then create custom environment and install `sfapi_client`

```
conda create -n sfapi-demo python=3.11
conda activate sfapi-demo
pip install "sfapi_client==0.0.8" ipykernel
python -m ipykernel install \
--user --name sfapi --display-name sfapi
```

Live walkthrough



Thank you for your attention!



Technical overview

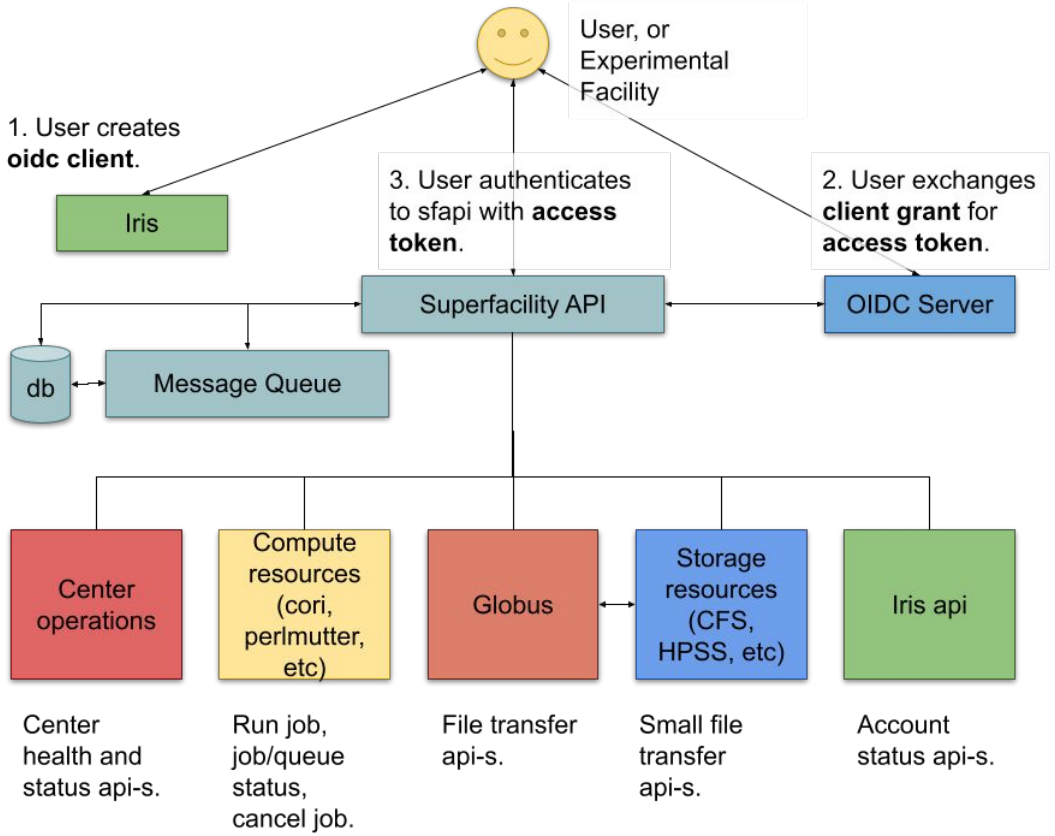
Superfacility API architecture

The API consists of:

- JWT-based authentication
- Each REST endpoint a thin layer on top of a NERSC resource
- Message Queue (RabbitMQ) to handle Asynchronous tasks
- Redis & PostgreSQL database for application state

Docker images on NERSC cloud service.

Hides actual complexity of interacting with NERSC behind simple API.



Example API process flow

Using API client and access token for authentication

POST /compute/jobs to start job (async)

GET /task/id to get status of async task (job submission)

GET /compute/job/<job_id> to see job status