Scale-out With Coffea: Coffea-Casa Analysis Facility

Mat Adamec, Ken Bloom, Oksana Shadura

University of Nebraska, Lincoln

Garhan Attebury, Carl Lundstedt, John Thiltges

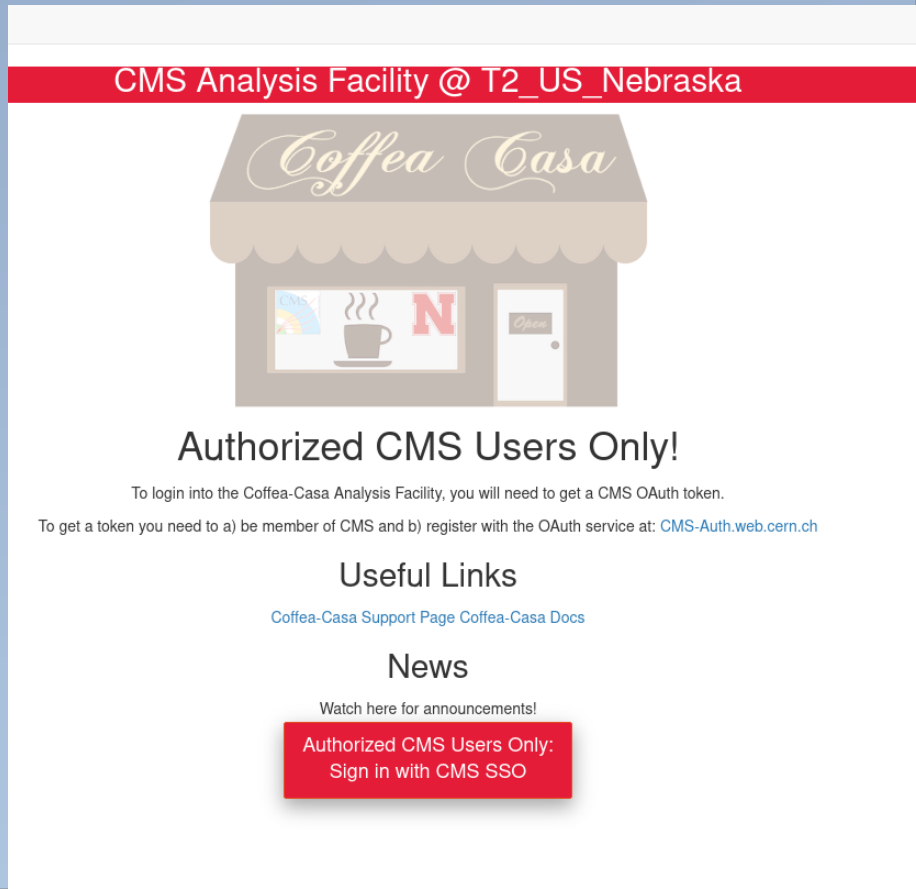
University of Nebraska, Holland Computing Center

Brian Bockelman

Morgridge Institute

Building blocks for designing Coffea-Casa

- Modern authentication (AIM/OIDC), tokens, macaroons
- Efficient data delivery and data management technologies
- Columnar analysis and support new python ecosystem
- Modern deployment and integration techniques
- Support for object storage
- Efficient data caching solutions
- Easy integration with existing HPC/HTC resources

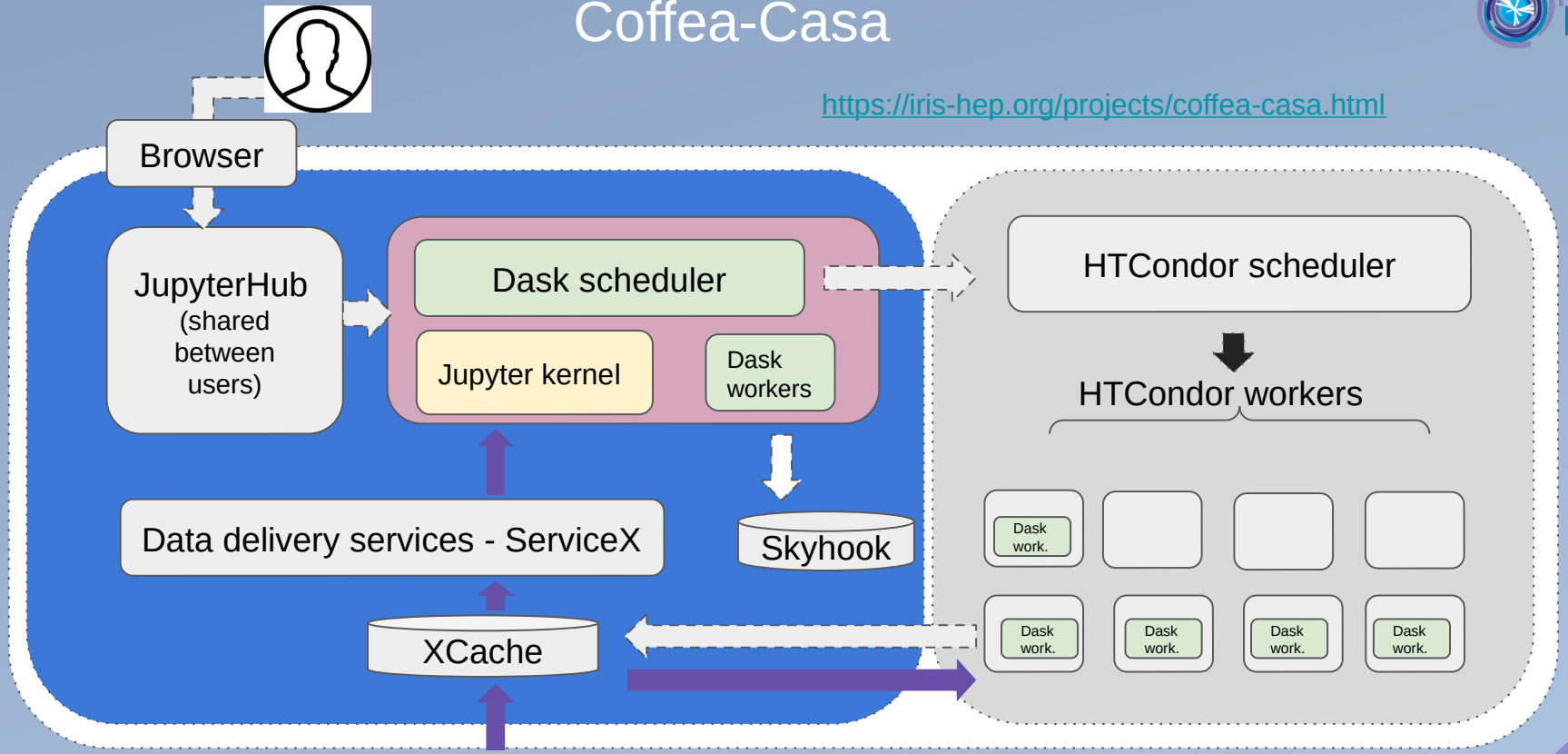


Pain points and Deploying a Coffea-Casa

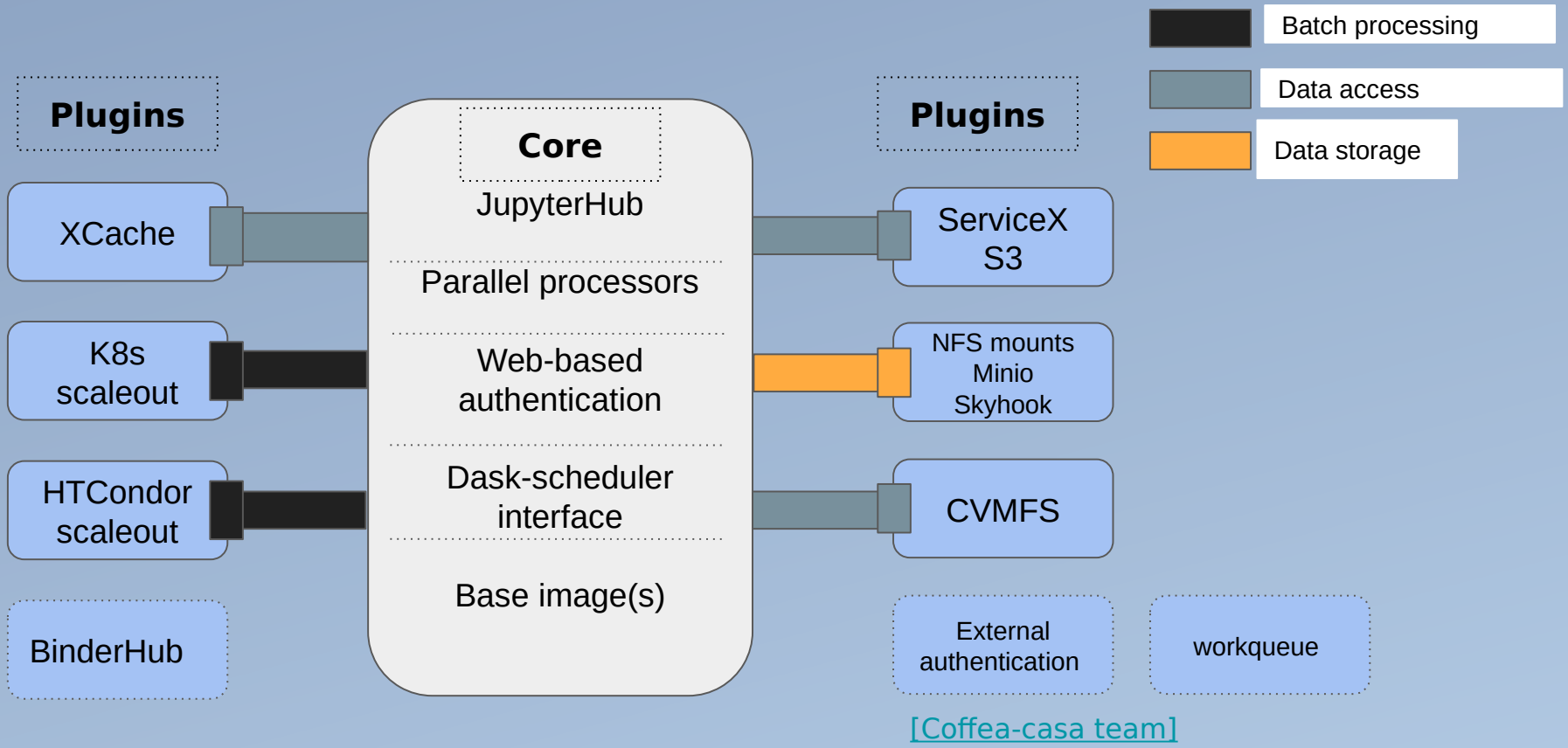
- Modern authentication (AIM/OIDC), tokens, macaroons
- Efficient data delivery and data management technologies
- ~~Columnar analysis and support new python ecosystem~~
- Modern deployment and integration techniques
- Support for object storage
- Efficient data caching solutions
- Easy integration with existing HPC/HTC resources

Coffea-Casa

https://iris-hep.org/projects/coffea-casa.html



Coffea-Casa: Aiming for Remote Adoption



[Coffea-casa team]

Building blocks: Authentication Tools

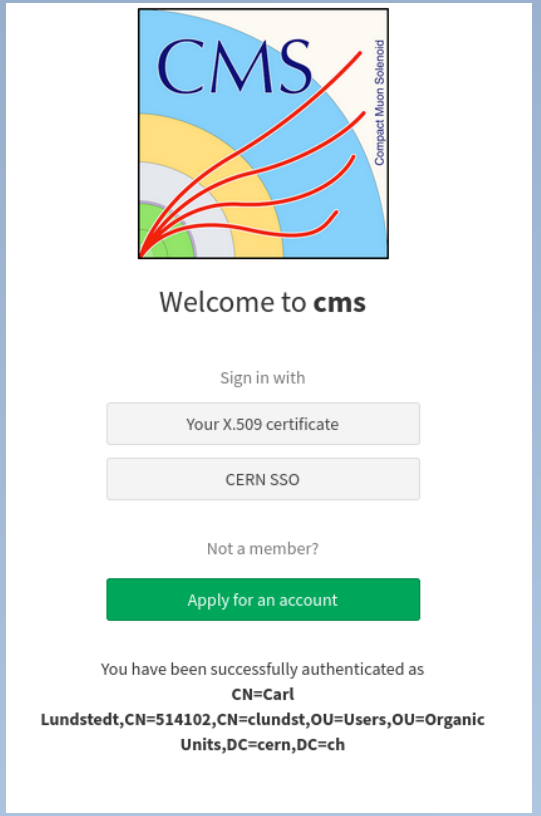
Jupyterhub allows for a variety of authentication methods and we inherit this functionality.

Using OAuth we can select an OIDC service to manage users for us.

Dummy authentication is also useful for spinning up test instances.

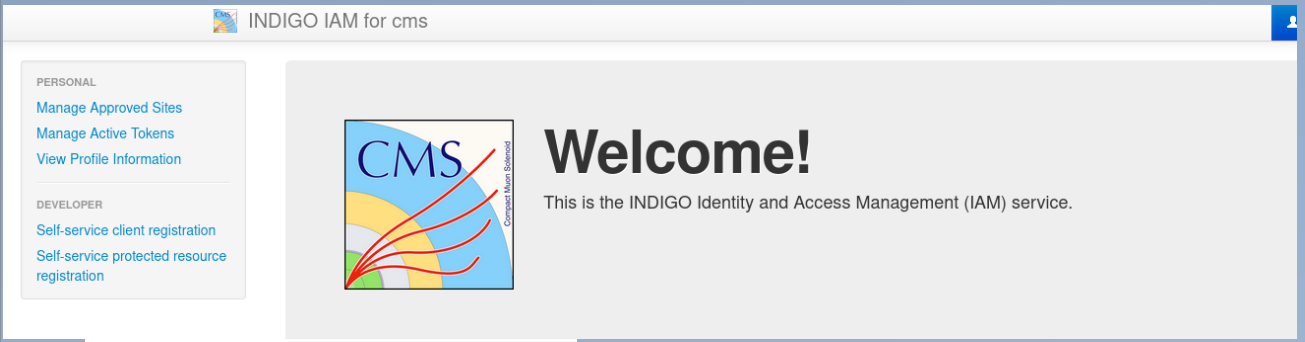
Each instance must be registered and secrets about that client have to be available in the instance. We seal these secrets so we can store them in git encrypted.

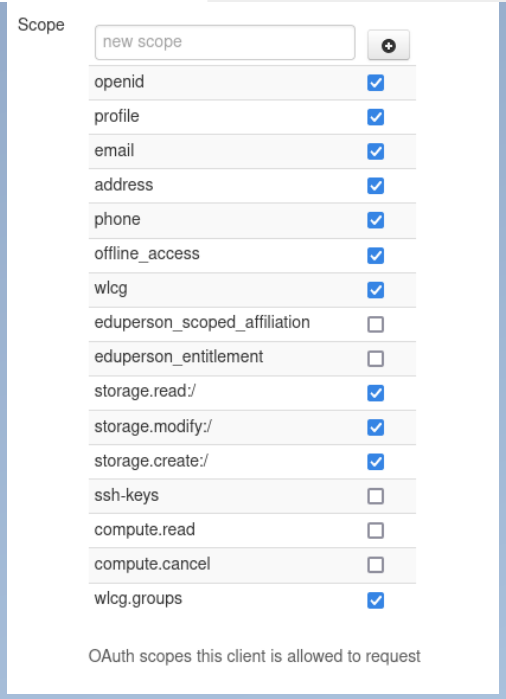
```yaml
values:
  hub:
    extraEnv:
      #OAUTH_CALLBACK_URL: https://cmsaf-jh.unl.edu/hub/oauth_callback
      OAUTH2_AUTHORIZE_URL: https://cms-auth.web.cern.ch/authorize
      OAUTH2_TOKEN_URL: https://cms-auth.web.cern.ch/token
      OAUTH_CLIENT_ID:
        valueFrom:
          secretKeyRef:
            name: jhub-cms-auth
            key: client_id
      OAUTH_CLIENT_SECRET:
        valueFrom:
          secretKeyRef:
            name: jhub-cms-auth
            key: client_secret
```

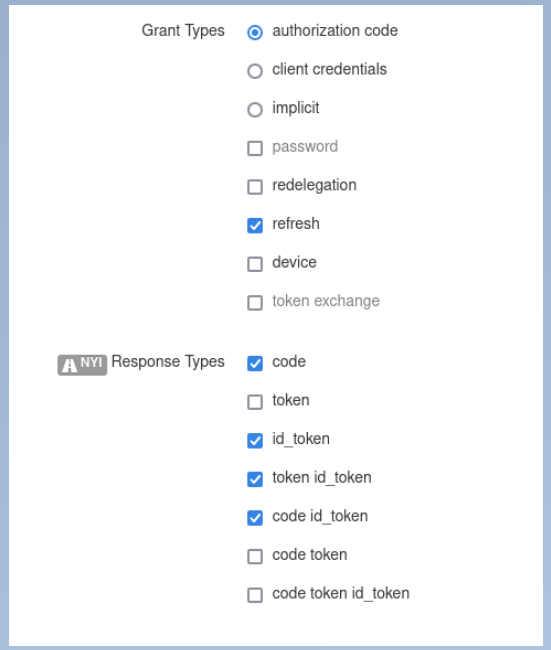


https://cms-auth.web.cern.ch
https://cilogon.org/oauth2/register

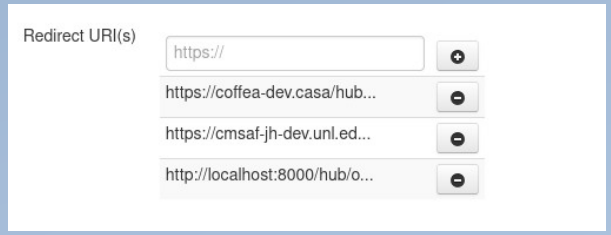
Building blocks: OIDC demo








```yaml
values:
  hub:
    extraEnv:
      #OAUTH_CALLBACK_URL: https://cmsaf-jh.unl.edu/hub/oauth_callback
      OAUTH2_AUTHORIZE_URL: https://cms-auth.web.cern.ch/authorize
      OAUTH2_TOKEN_URL: https://cms-auth.web.cern.ch/token
      OAUTH_CLIENT_ID:
        valueFrom:
          secretKeyRef:
            name: jhub-cms-auth
            key: client_id
      OAUTH_CLIENT_SECRET:
        valueFrom:
          secretKeyRef:
            name: jhub-cms-auth
            key: client_secret
```



Opendata Instance:

```yaml
GenericOAuthenticator:
  scope:
    - openid
    - email
    - org.cilogon.userinfo
Authenticator:
  allowed_groups:
    - CO:COU:OpenData Facility:members:active
```

Building blocks: Secrets (KubeSeal)

```yaml
apiVersion: bitnami.com/v1alpha1
kind: SealedSecret
metadata:
  name: mysecret
  namespace: mynamespace
spec:
  encryptedData:
    foo: AgBy3i4OJSWK+PiTySYZZA9rO43cGDEq.....
```

```yaml
apiVersion: bitnami.com/v1alpha1
kind: SealedSecret
metadata:
  creationTimestamp: null
  name: jhub-cms-auth
  namespace: cmsaf-dev
spec:
  encryptedData:
    client_access_key: AgDAiwO7oV/7lyHBstZxOJQpXnf1b5Q9ZHKRIEWfl2bFw4f1W68AvhAinxrwqOrMprJVJUoVOMFCCJ3cOw2ojDYzmKaL
    client_id: AgBkc7gx4Vr8Wsn3LmxWinuTG92L/bT+/mBbmtbf1IWOusZkB8Mj18B9JMNf+JxzHJO+yU3MNfwk+UhxNo45jlXdweNs9KZqWEZ
    client_secret: AgARi2XcJMHZiyXdLDxZhU8siwIPI256/V3aiU6owVb9R5D1KSk26+4iljxRwXhPZArXKkq1QHly4y99oVDgcsLkeA7wWay7
  template:
    metadata:
      creationTimestamp: null
      name: jhub-cms-auth
      namespace: cmsaf-dev
    type: Opaque
```

Building blocks: Token Management

Tokens are signed strings that allow for various functionality. The pieces of Casa using/needing tokens are HTCondor and XCache access.

We want to abstract access authentication away from the user so things 'just work'.

We also want to obscure any x509 certs as needed.

Token creation takes place in a custom script that's shipped with the base Coffea-casa codebase.

Sample xcache_token payload:

location T2_US_Nebraska
identifier 46a….
cid name:cms-jovyan
cid activity:DOWNLOAD
cid path:/store
cid before:2023-04-24T13:59:47Z

Condor POOL Password vs. Token

Pool passwords are master keys for your cluster and needed to sign any tokens.

The POOL password is stored in K8s but NOT available in any user's environment.

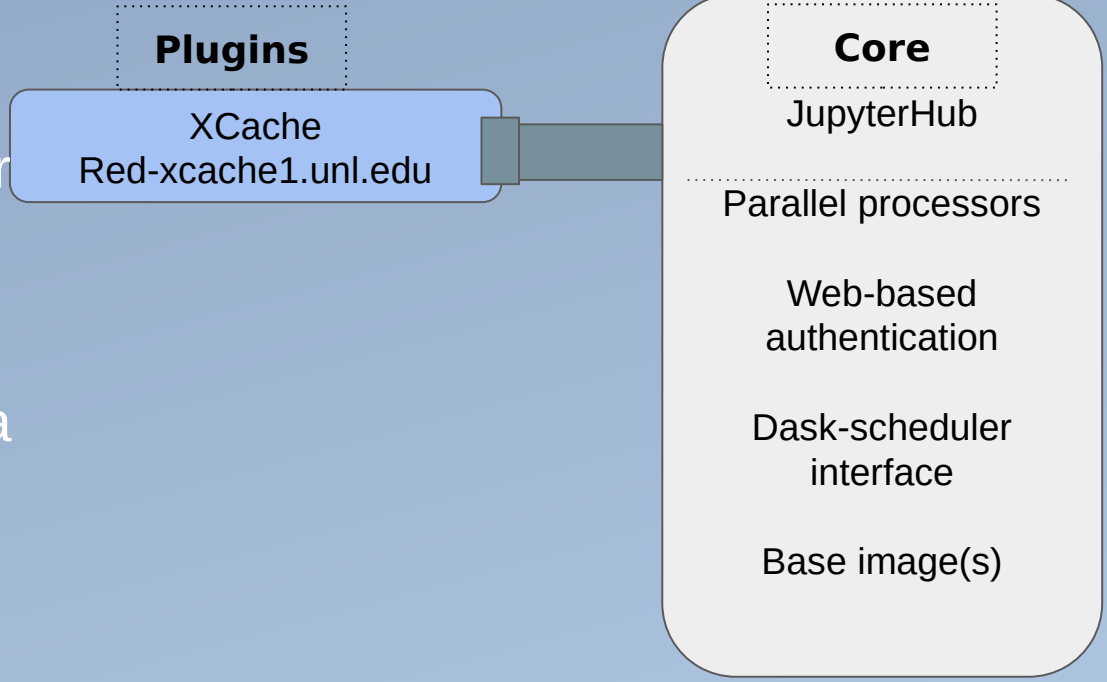
The user only gets a user token to submit jobs to the local queue, whether that queue is a remote resource or a K8s service.

Building blocks: Data Delivery (Stand-alone XCache)

red-xcache1 has 88TB of cache space and allows a cms coffea user access to the CMS data namespace.

Although the casa issues the user a token the mechanism for access is x509 “under the bonnet”.

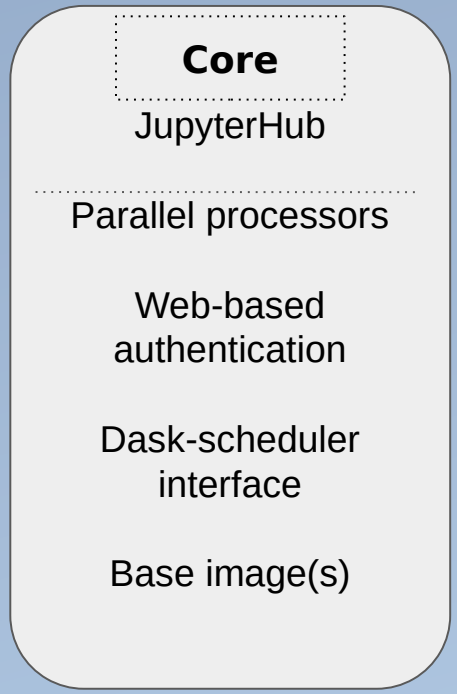
Full K8s native Xcache 'almost there'.



[Coffea-casa team]

Building blocks: Data Delivery (Skyhook)

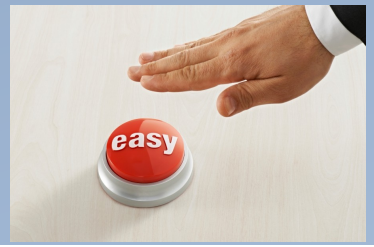
Skyhook bumps into a limitation for scaling out to a cluster as it requires access to the underlying CEPH filesystem. This doesn't easily cross network boundaries. Our skyhook enabled instances are limited to internal Condor worker/resources.

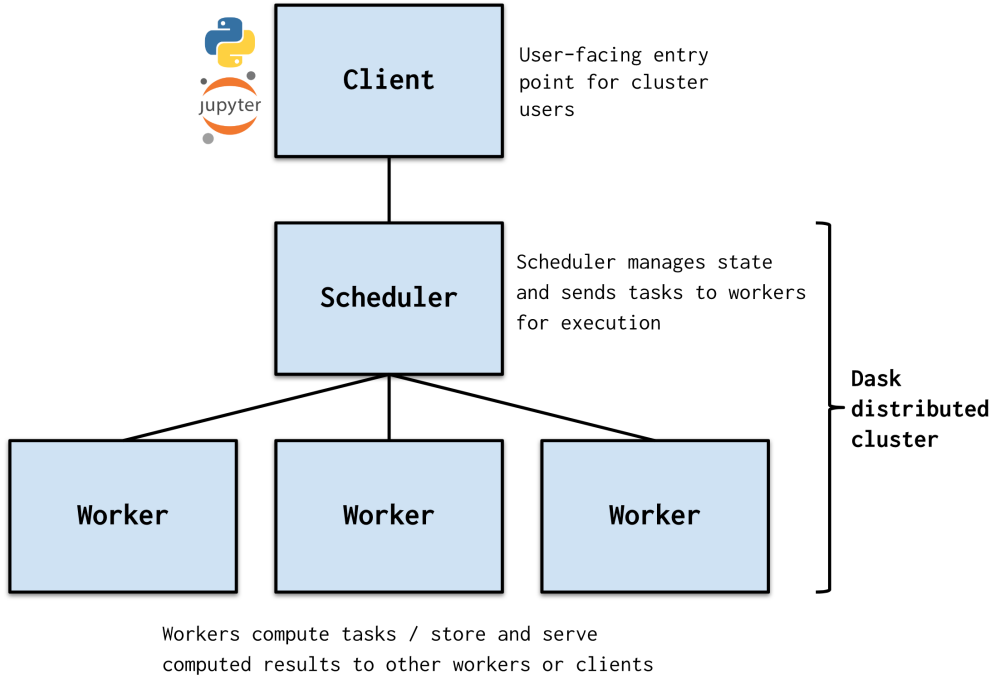


[Coffea-casa team]

Building blocks: Dask

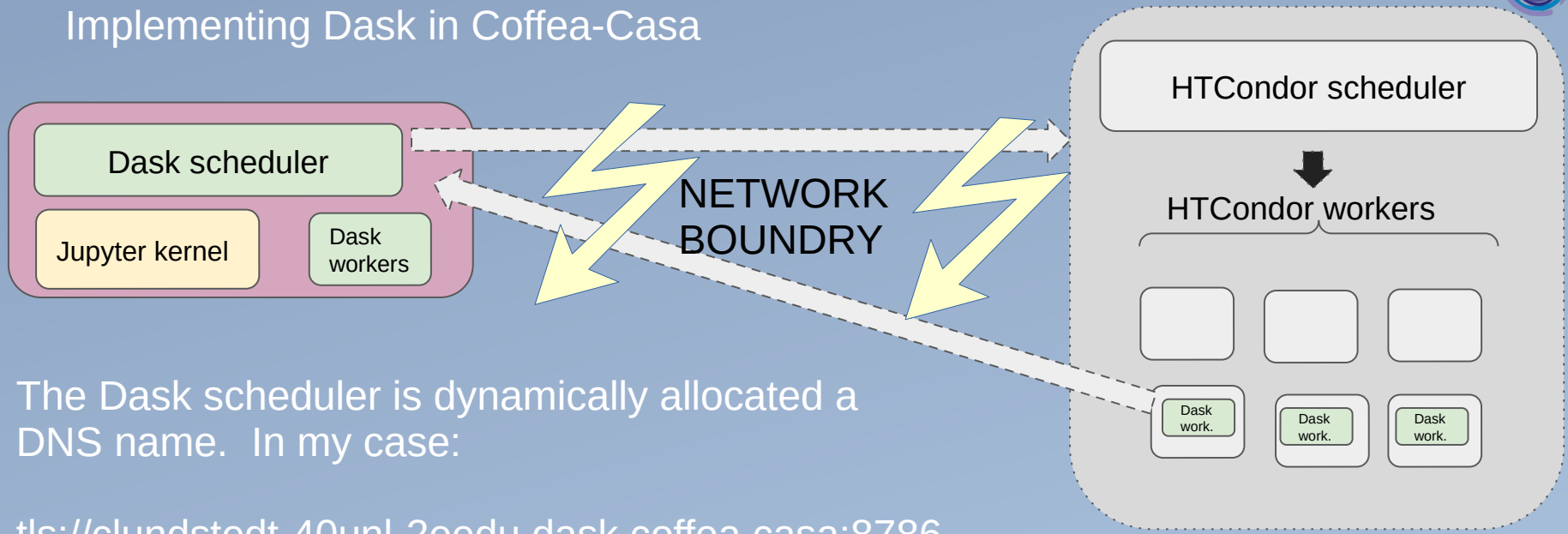
- Dask provides a task-management computational framework in Python based on the manager-worker paradigm
- Dask exposes lower-level APIs letting to build custom systems for in-house applications
- Integrates with HPC clusters, running a variety of schedulers including SLURM, LSF, SGE and *HTCondor via "dask-jobqueue"*
- ***This allows us to create a user-level interactive system via queueing up in the batch system***





Dask can be used inside Jupyter or you can simply launch it through Jupyter and connect directly from your laptop

Implementing Dask in Coffea-Casa



The Dask scheduler is dynamically allocated a DNS name. In my case:

tls://clundstedt-40unl-2eedu.dask.coffea.casa:8786

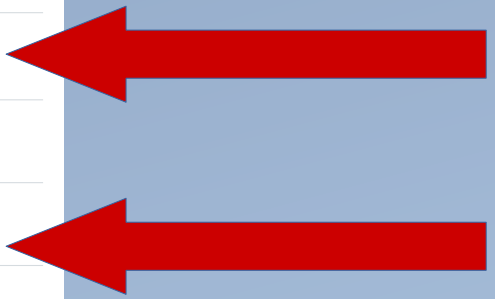
This address is passed to the HTCondor/Dask workers to report back to.

Kubernetes manages this traffic for each instance using a traefik service spun up in each instance

Coffea-Casa

At the DNS service (GoDaddy in this case):

	Type	Nom	Données	TTL		
☐	A	@	129.93.183.36	600 secondes	Supprimer	Modifier
☐	A	*.dask	129.93.183.35	1 heure	Supprimer	Modifier
☐	A	*.dask-worker	129.93.183.35	1 heure	Supprimer	Modifier
☐	A	opendataaf-servicex-aod-minio.servicex	129.93.183.35	1 heure	Supprimer	Modifier
☐	A	opendataaf-servicex-minio.servicex	129.93.183.35	1 heure	Supprimer	Modifier
☐	A	opendataaf-servicex.servicex	129.93.183.35	1 heure	Supprimer	Modifier



Two IPs were allocated to this instance, coffea-opendataaf.casa

Implementing Dask in Coffea-Casa

The Coffea-casa class extends the Dask job-queue to automatically configure the job queue environment. Users need know NO Condor or service names.

Service providers/admins can tune the size of workers here.

```
## Job extra settings (HTCondor ClassAd)
job_config["job_extra"] = merge_dicts(
    {
        "universe": "docker",
        "docker_image": worker_image or dask.config.get(f"jobqueue.{cls.config_name}.worker-image")
    },
    {
        #"container_service_names": "dask,nanny",
        #"dask_container_port": DEFAULT_CONTAINER_PORT,
        #"nanny_container_port": nanny_port,
        "container_service_names": "dask",
        "dask_container_port": DEFAULT_CONTAINER_PORT,
    },
    {"transfer_input_files": files},
    {"encrypt_input_files": files},
    {"transfer_output_files": ""},
    {"when_to_transfer_output": "ON_EXIT"},
    {"should_transfer_files": "YES"},
    {"Stream_Output": "False"},
    {"Stream_Error": "False"},
    {"+DaskSchedulerAddress": external_ip_string},
    job_kwargs.get(
                                fig.get(f"jobqueue.{cls.config_name}.job-extra")
```

```
jobqueue:
  coffea-casa:

    # Dask worker options, taken from https://github.com/dask/dask-jobqueue/tree/master/dask_jobqueue
    cores: 2                # Total number of cores per job
    memory: "6 GiB"                # Total amount of memory per job
    processes: null                # Number of Python processes per jobs
    worker-image: "coffeateam/coffea-casa-analysis:latest"

    # Comunication settings
    interface: null             # Network interface to use like eth0 or ib0
    death-timeout: 60           # Number of seconds to wait if a worker can not find a scheduler
    local-directory: null       # Location of fast local storage like /scratch or $TMPDIR
    extra: []
```

GitOps and Flux

Flux is a service on K8s that checks the state of the cluster vs. what is requested in the charts/yaml files. The state of the cluster is defined in files stored in git, flux reconciles the state with the demand.

All Coffea-casa administration is done via git.

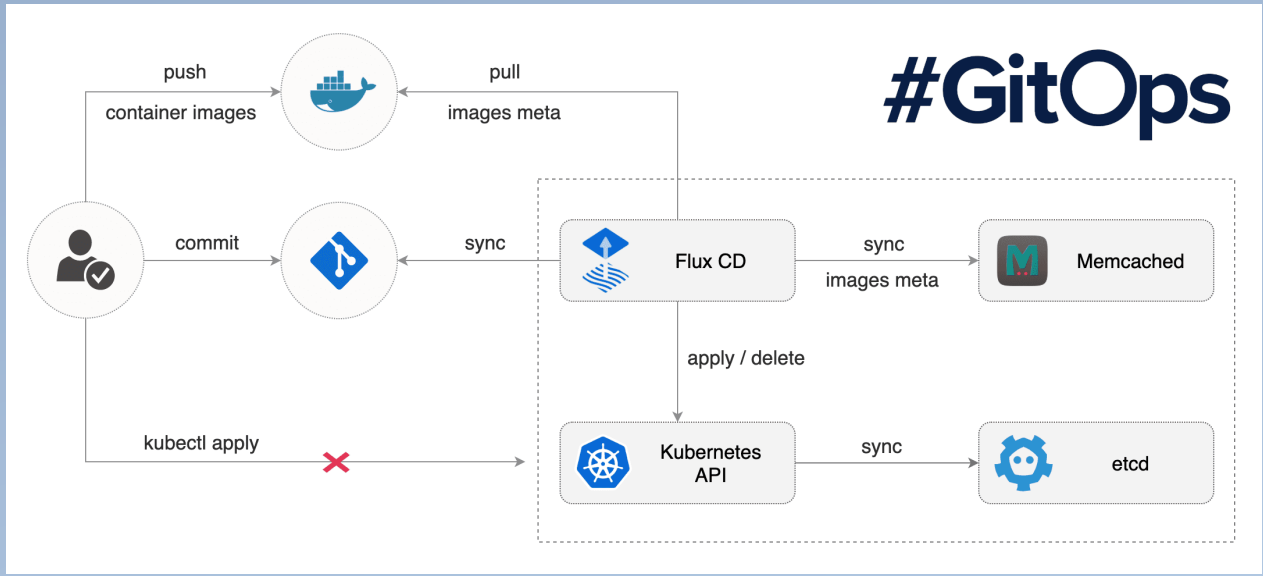
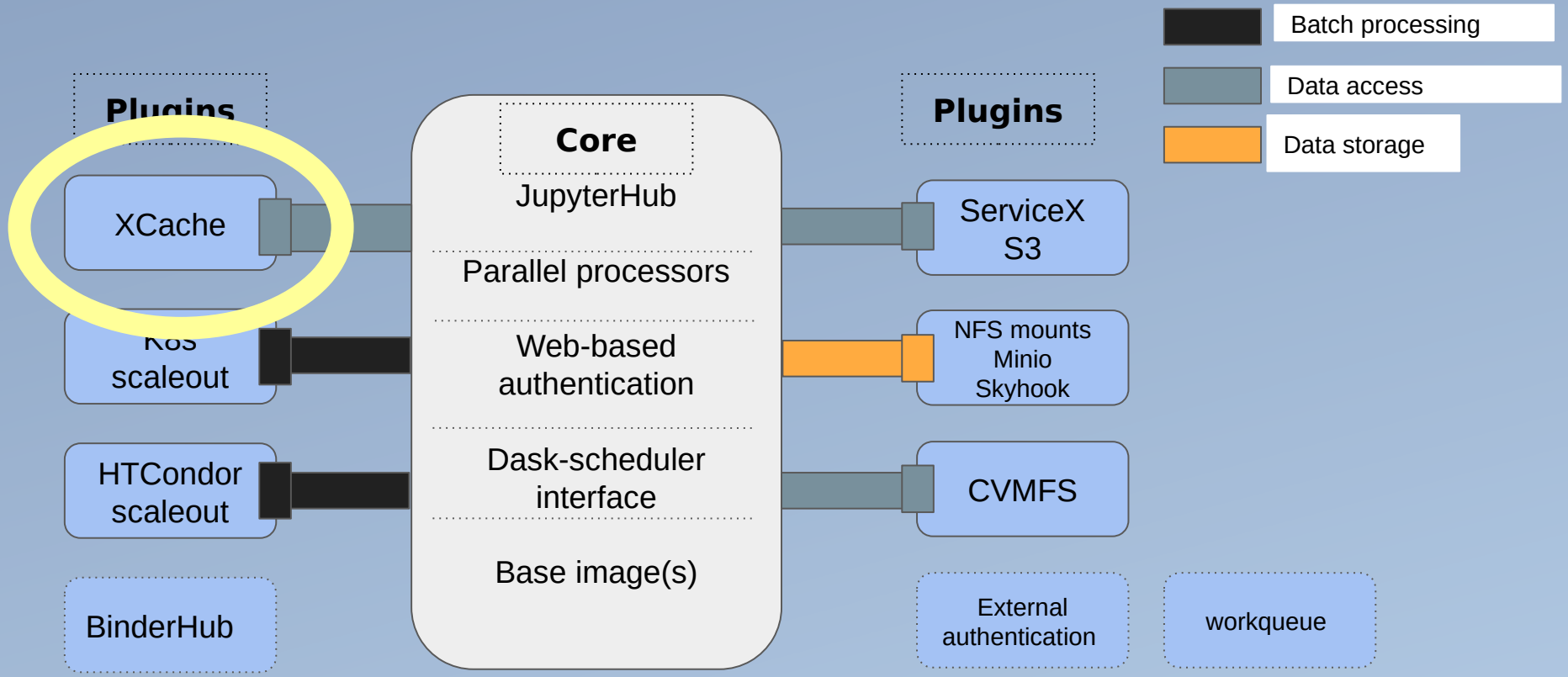


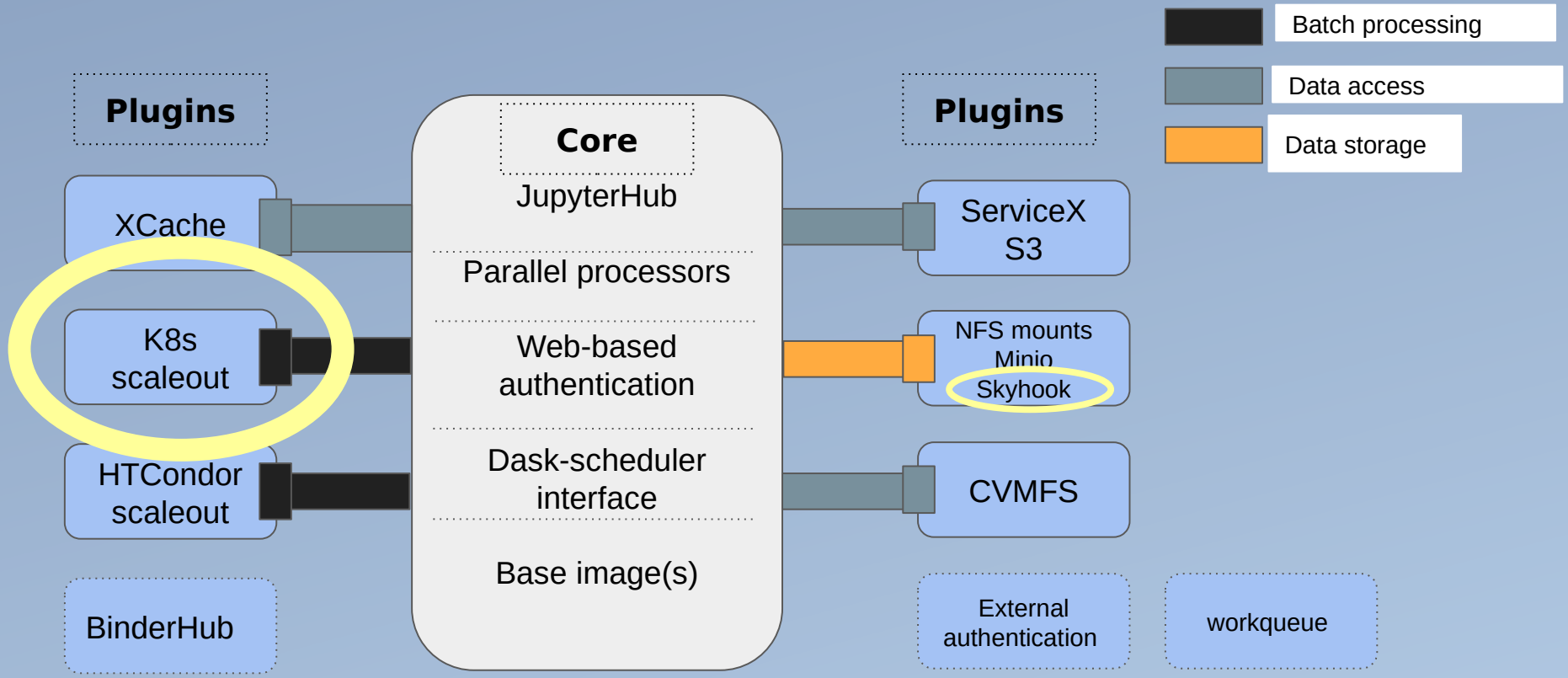
Image courtesy of https://fluxcd.io/legacy/flux/

Further extensions/improvements for Casa



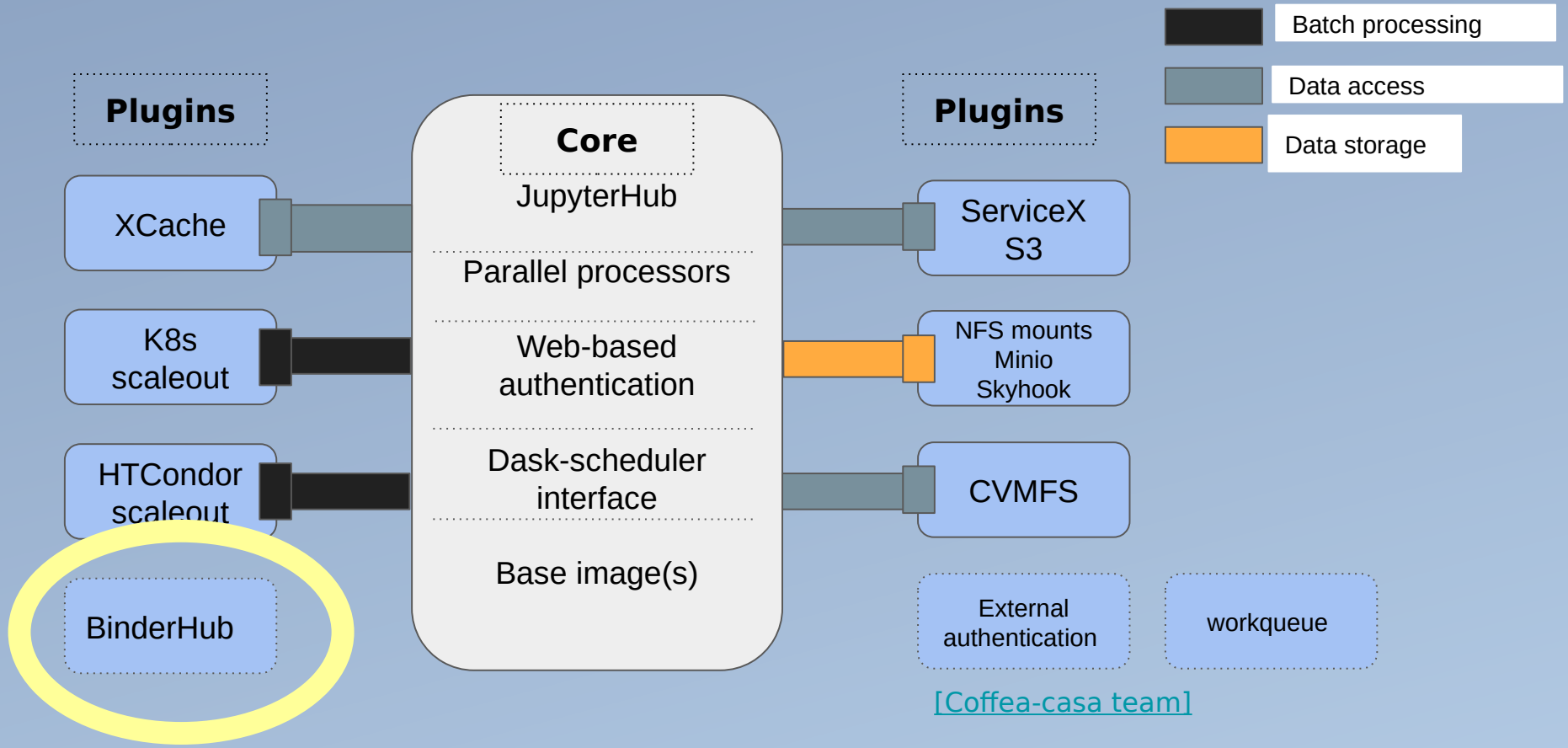
[Coffea-casa team]

Further extensions/improvements for Casa



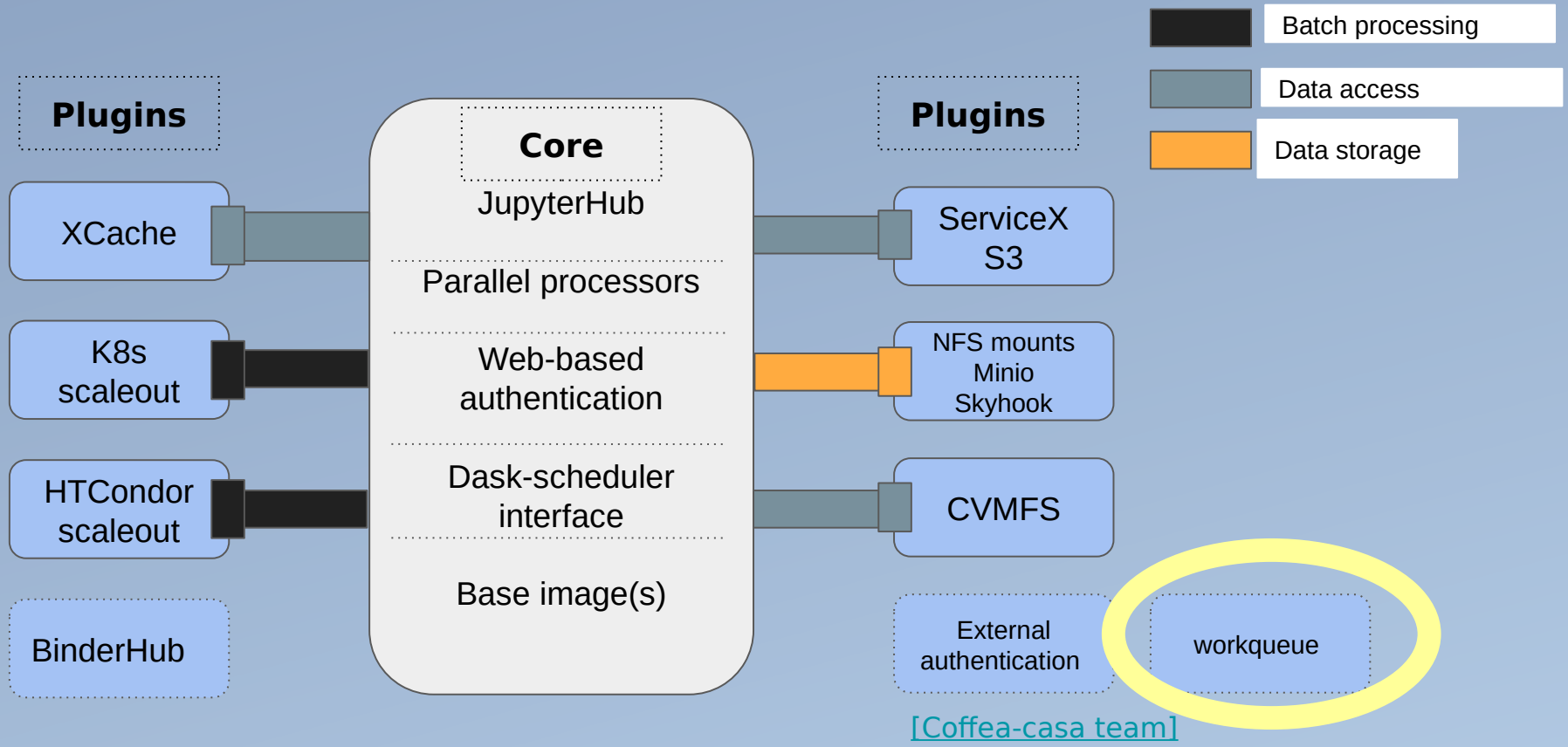
[Coffea-casa team]

Further extensions/improvements for Casa

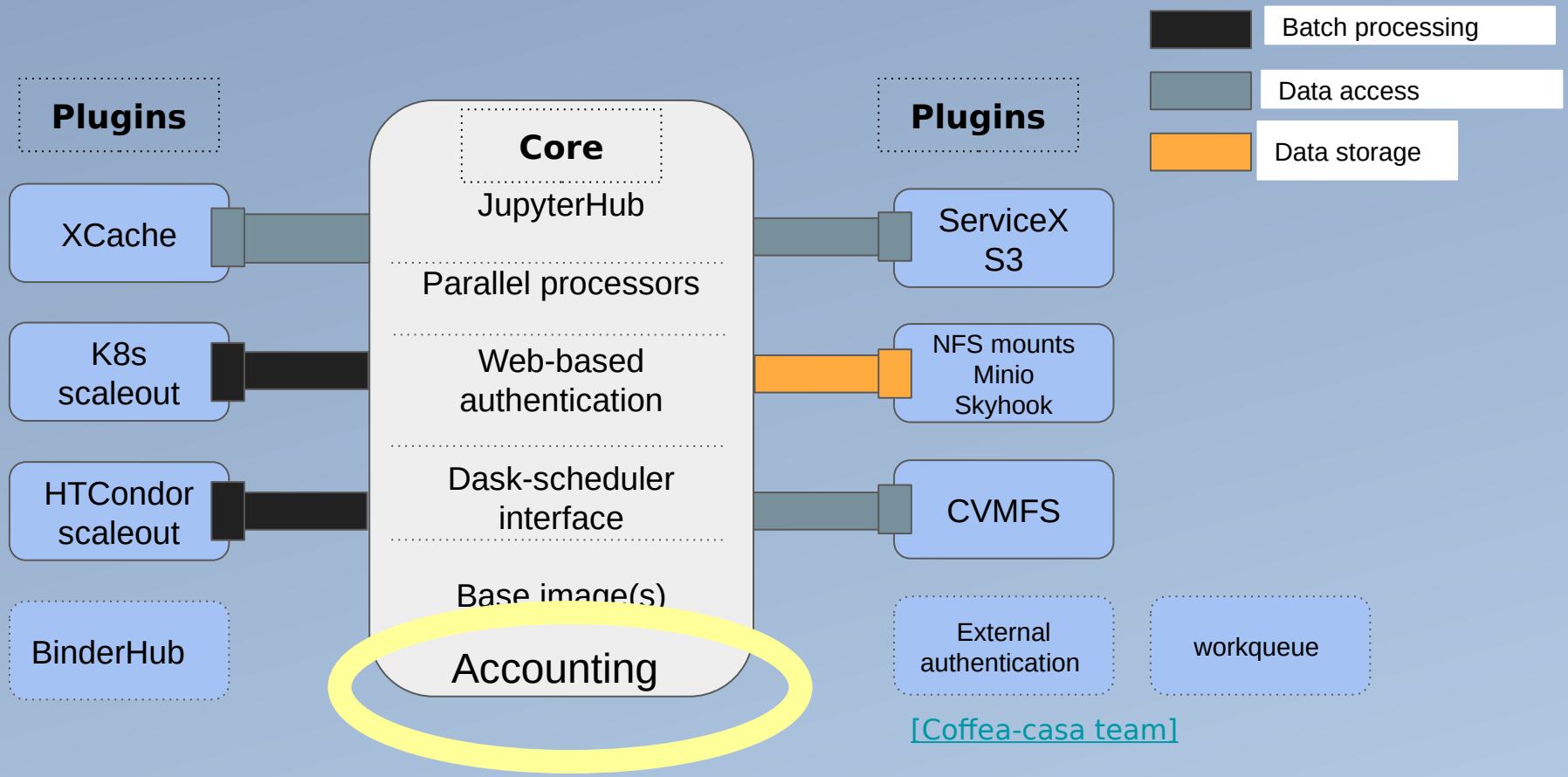


[Coffea-casa team]

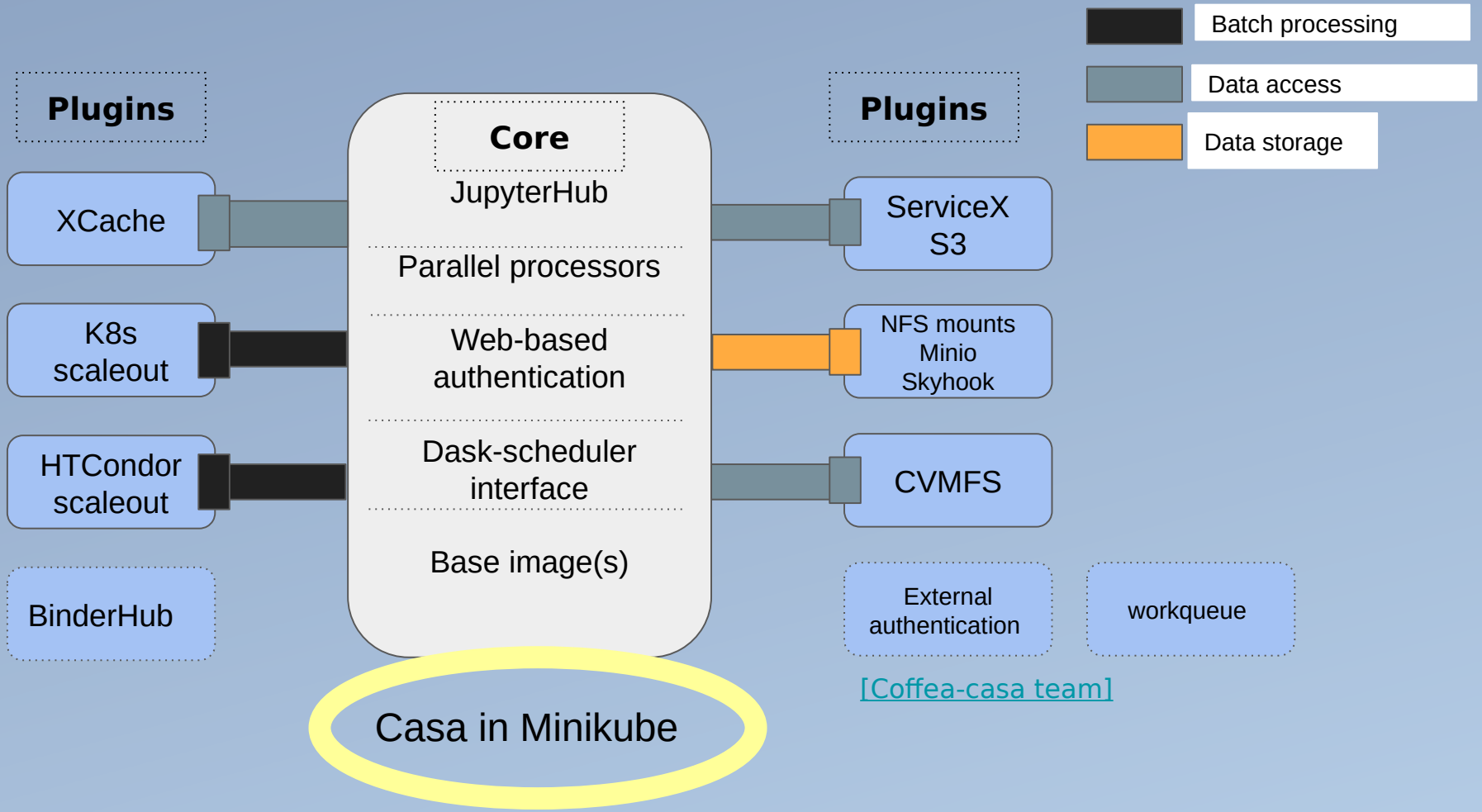
Further extensions/improvements for Casa



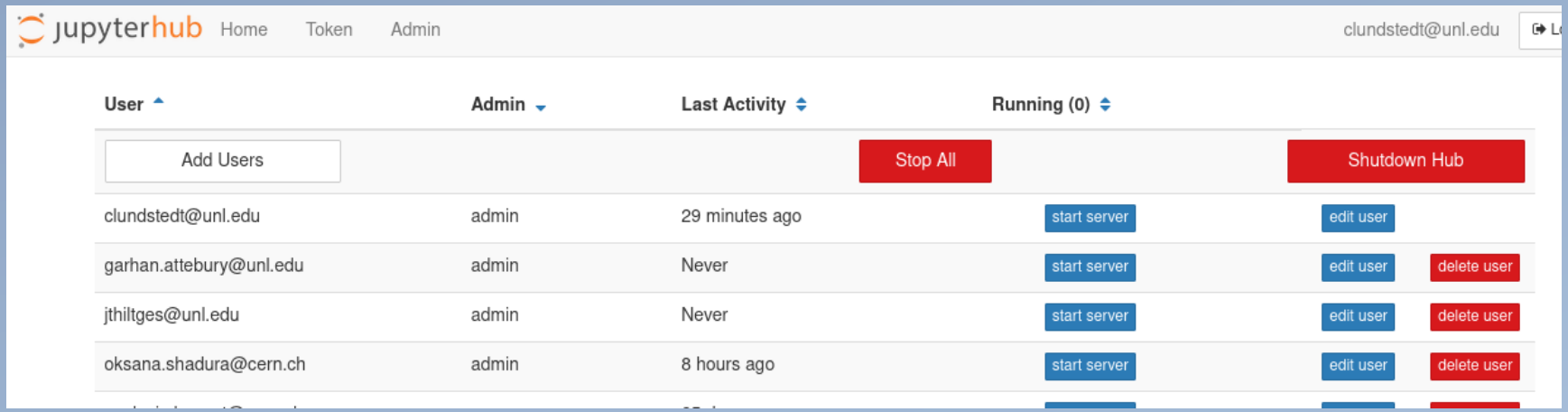
Further extensions/improvements for Casa



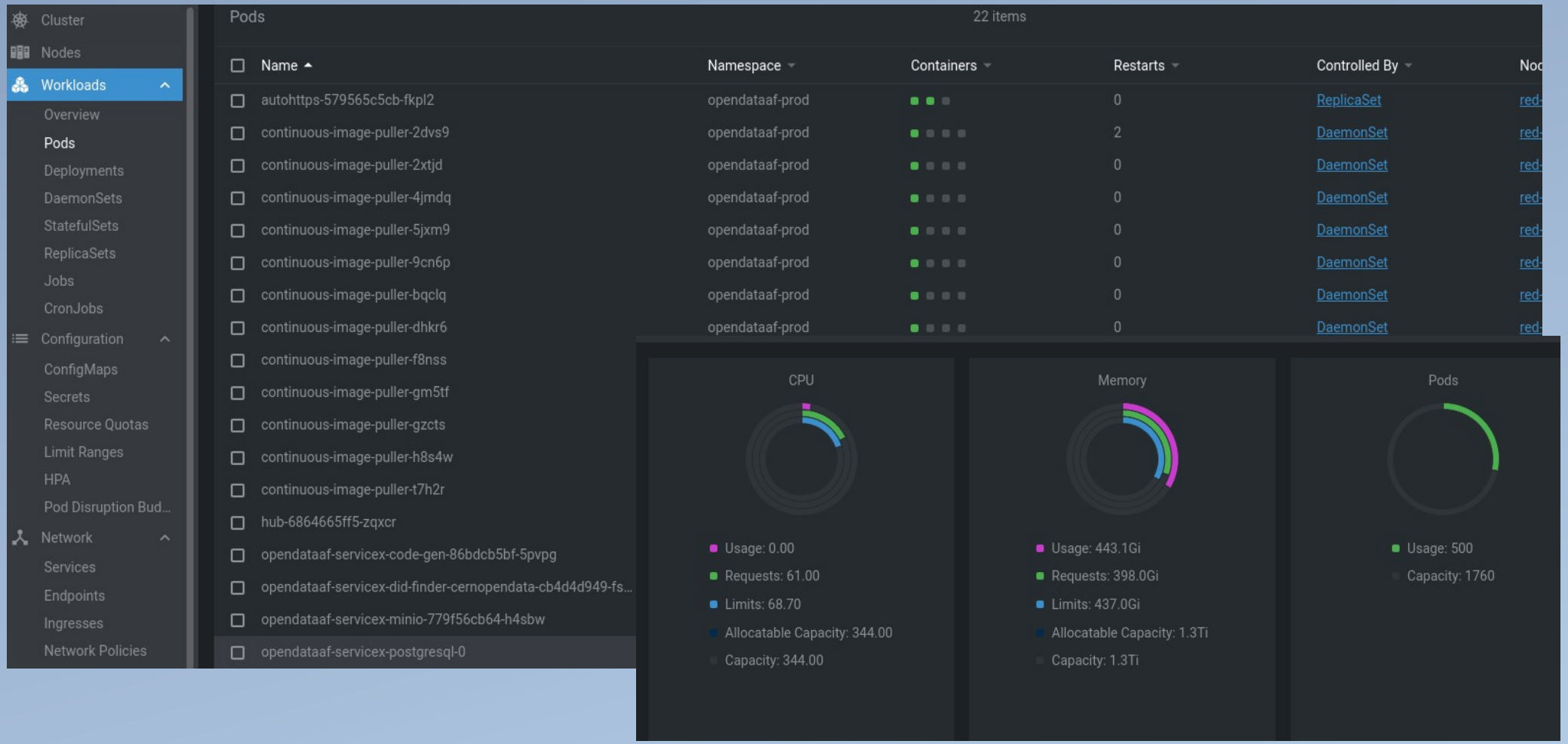
Further extensions/improvements for Casa



Tools we use to admin: Jupyterhub Native



Tools we use to admin: Lens



Scale-out With Coffea: Coffea-Casa Analysis Facility

Questions?

Designing Coffea-Casa Assets vs. Demands

Large HTC clusters (T2s)

Batch Experts

Xroot data services

Data caching

Container experts

Kubertnetes resources

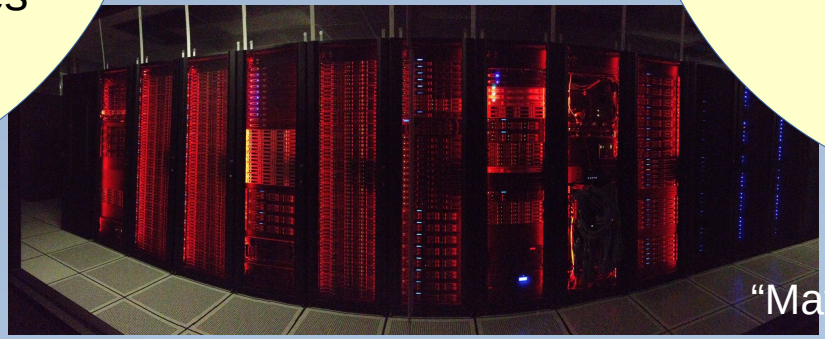
CASA

Support new analysis tools

Interactive

Transparent token management

Familiar



“May you live in an interesting age”