Pelican and the OSDF: An Overview

Introducing the OSDF

The OSDF is a federated platform for delivering datasets from repositories to compute in an effective, scalable manner.

OSDF Integrates Independent Repositories into a common fabric

★ AWS Open Data

NCAR ★

LIGO ★

- About a dozen repositories integrated already, more on the way.
- Working to grow:
  - clients,
  - integrated resources, and
  - environments.

jupyter

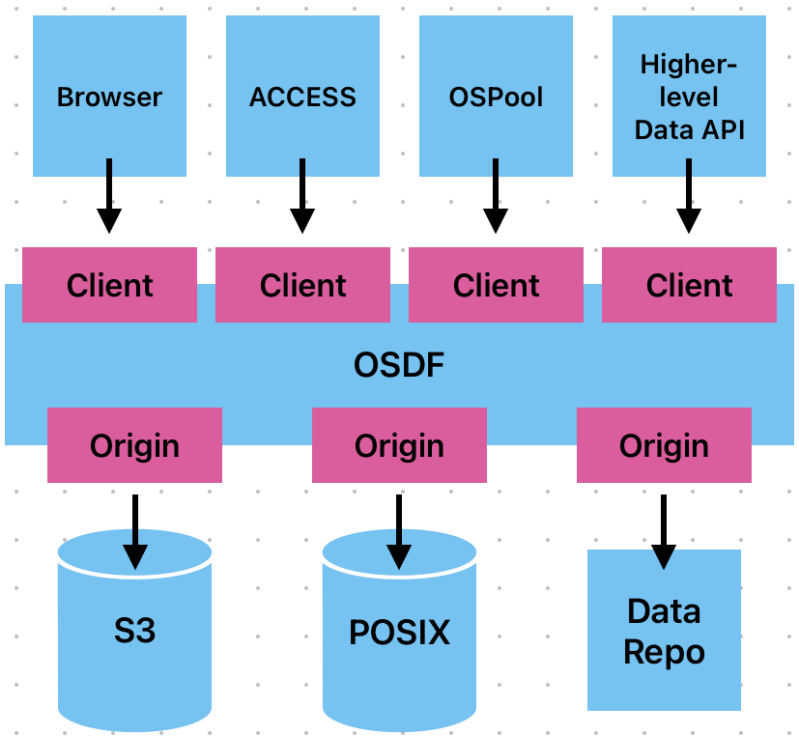
DeltaAI

I | NCSA

OSPool ★

★ = existing integration

OSDF Architecture - Vision



<u>Long-term vision</u>:
We want OSDF to be an “all-science” CDN.

Requires:

- Connect many repositories to the distribution fabric.
- Provide clients that enable as many use cases.

And benefit from the network effects.

A bit on the cache layer…

- Anyone can run a cache!
- However, the OSDF centrally runs regional caches, mostly at network locations.
- Builds on top of a distributed Kubernetes cluster run by the National Research Platform (NRP).
  - Single, uniform interface to run services across the country.
- “Typical” cache hardware is ~100GbE / 20TB NVMe.

The OSDF: Connecting your repository

The OSDF provides an “adapter plug”, connecting your science repository to the national and international cyberinfrastructure.

The OSDF is operated by PATh

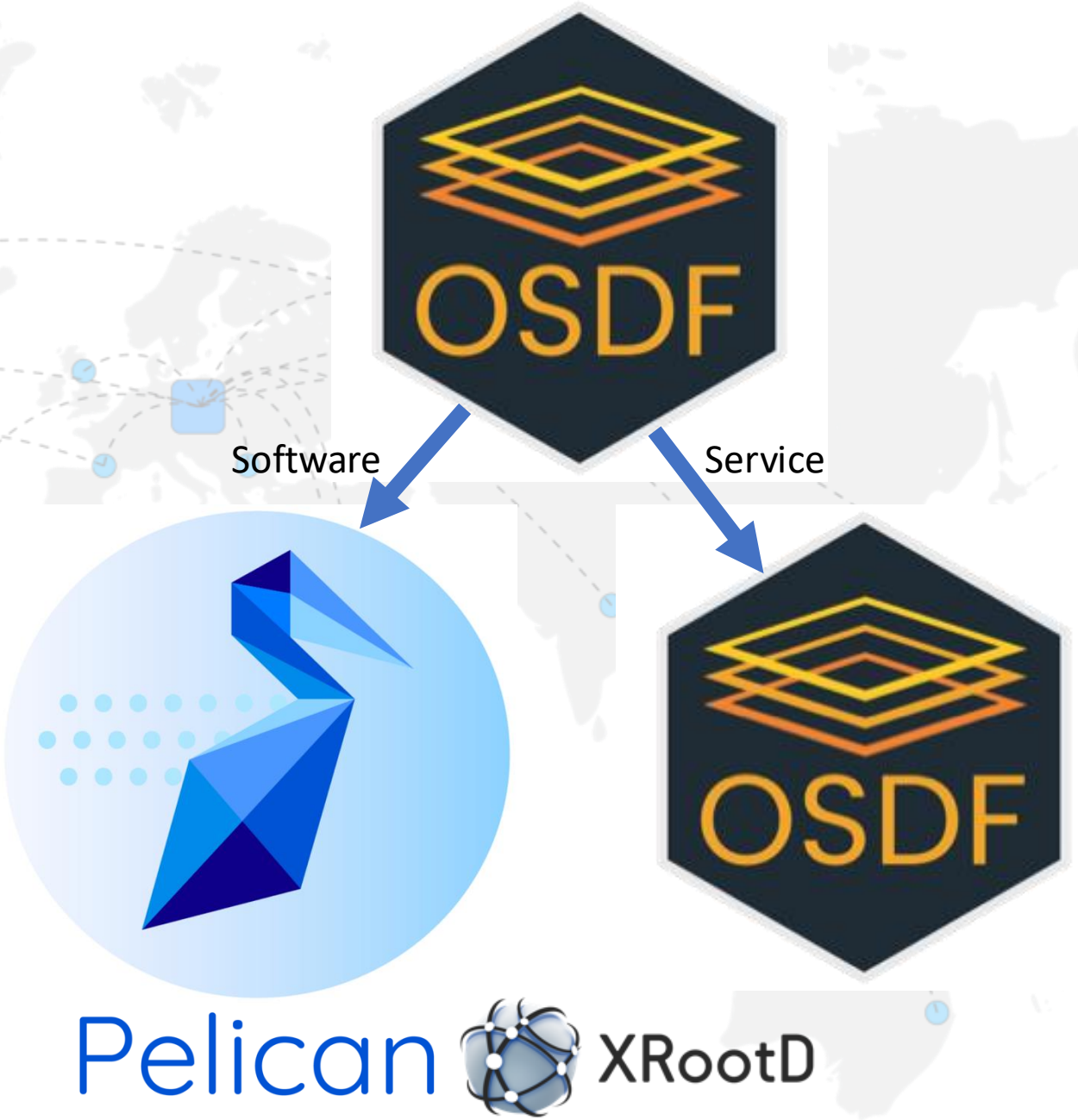
Using hardware from NRP

And integrates a wide range of open science,

As part of the OSG Consortium’s Fabric of Services

OSDF & Pelican

- You may have seen prior presentations about the OSDF – it (or predecessors) have existed for ~10 years.
- We split out the technology powering the OSDF and christened it the "**Pelican Platform**".
  - Same components as before, just integrated into a standalone platform.



The Pelican Project

The OSDF is operated by PATh using hardware from NRP and others.

Who develops the software?

The Pelican project (OAC-2331480) is a newly-funded, $7M/4-year project with the following goals:

1. Strengthen and Advance the OSDF.
2. Expand the types of computing where OSDF is impactful.
3. Expand the science user communities.
   - With a particular driver of the climate community.

Who uses the OSDF

A few notable use cases:

- LIGO distributes their proprietary data via OSDF.
- NRAO has used it for image processing.
- NCAR connects its Research Data Archive (~5PB of climate data).
- DUNE uses it for conditions data
- Several experiments use it for container distribution
- Individual PIs on the OSPool use it for managing inputs and outputs.

OSDF by the numbers

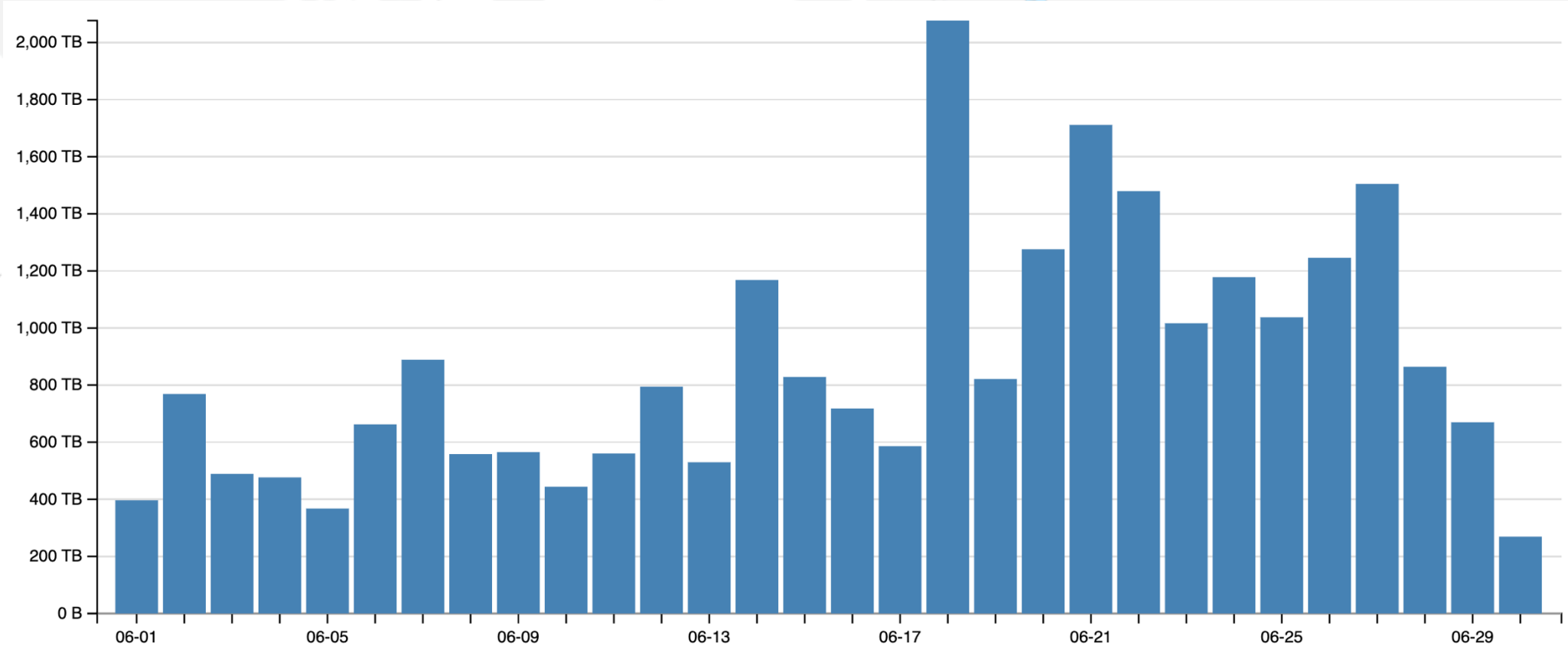
Over the last 12 months, the OSDF transferred

230PB &
125 req/s

Data used by

15 science collaborations &
~120 OSPool users

Example Daily Volume – June 2024

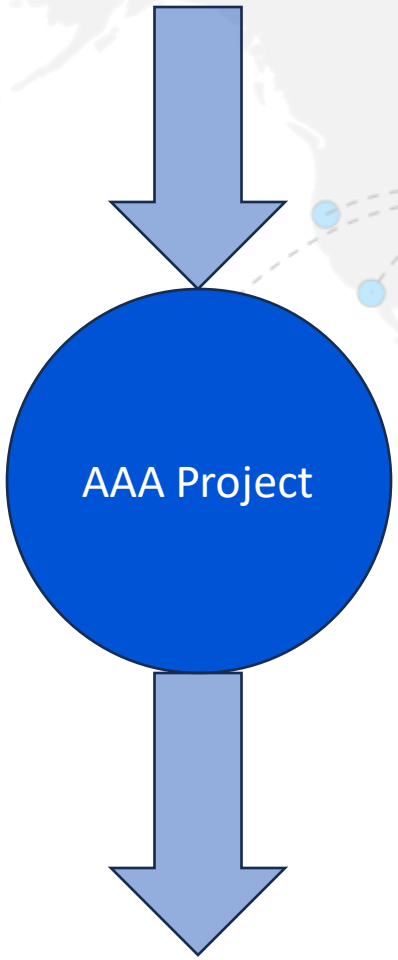


Note: individual experiments can still dominate a day's activities.

The OSDF: A brief history

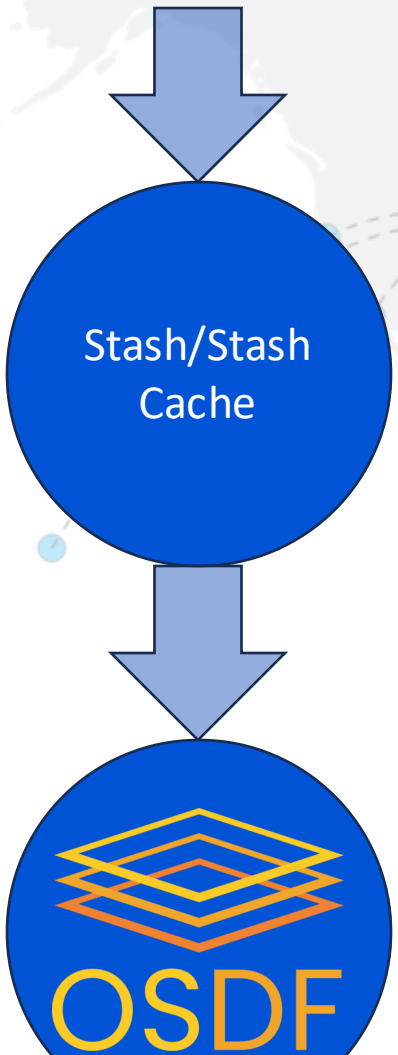
Starting in ancient times – 2009.

Any time, Any Data, Anywhere



- In ~2009, the teams at Nebraska, Wisconsin, and UCSD started using the XRootD software to build out a data federation for CMS.
- This turned into a 3-year funded project, AAA (NSF #1104664), starting in 2011.
- Outcomes include:
  - A robust data federation, based on the XRootD protocol, that delivers petabytes to CMS through this day.
  - Highly tailored to the needs of the High Energy Physics community.

Evolution toward the OSDF



- Around 2016, the OSG Consortium built on top of the AAA approach.
  - Originally used to export the “Stash” filesystem at U.Chicago; hence, the caching infrastructure was “StashCache”.
- Evolutions from being CMS-specific:
  - Used the cache software, developed in AAA, for data delivery.
  - Switched to SCITOKENS-based authorization (OAC-1738962, 2114989).
  - Origins could register with the OSG registration service.
- This evolved into the OSDF in ~2021:
  - Hardware was placed into the network.
  - Emphasis on Kubernetes-based packaging.
  - Distributed service operations with Kubernetes and the NRP.
  - NSF-funded hardware projects join in the federation.

Resulted in the Pelican Project in 2023!

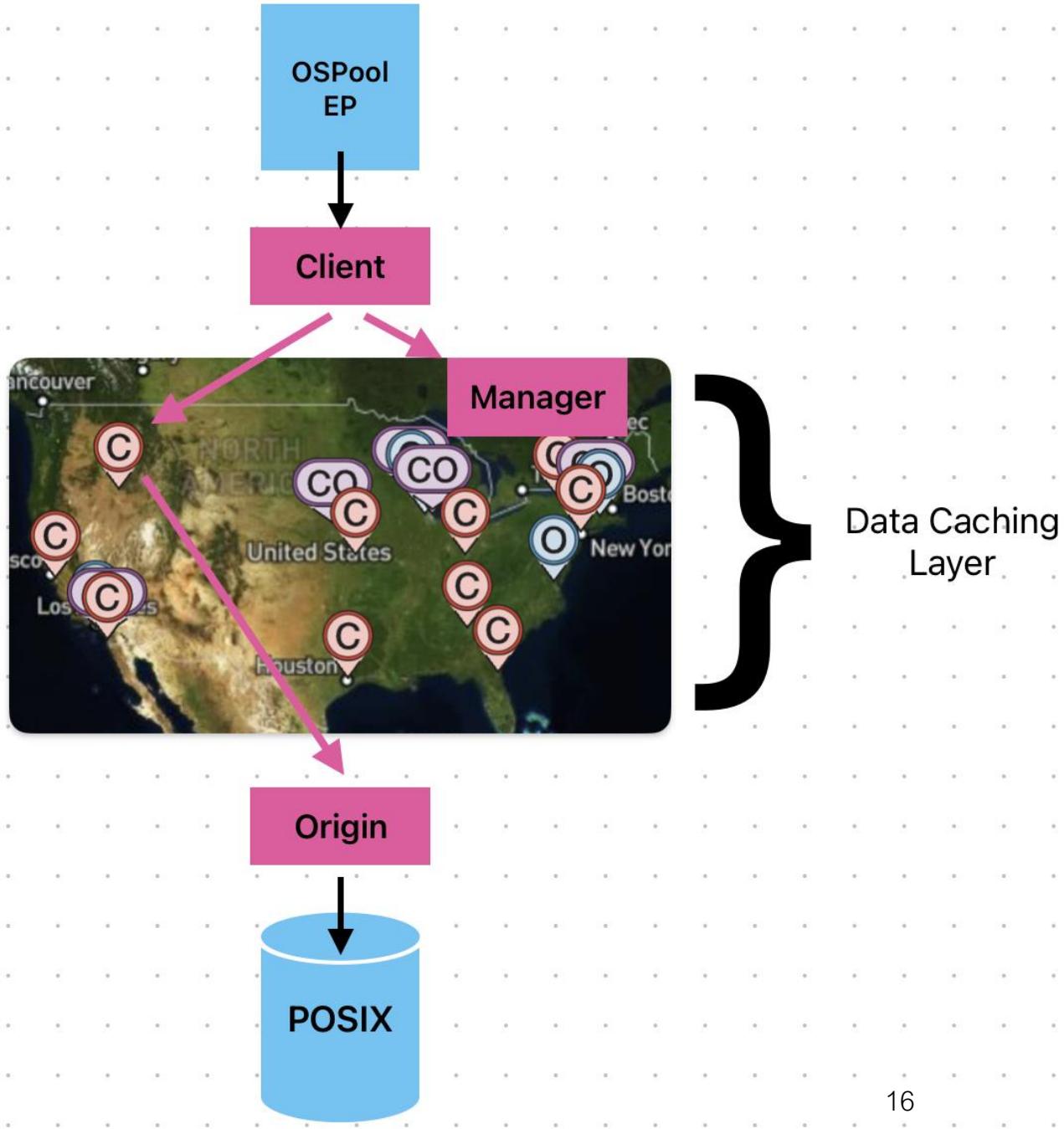
How does the OSDF work?

A brief tour through the Pelican architecture as implemented by the OSDF.

OSDF in Practice

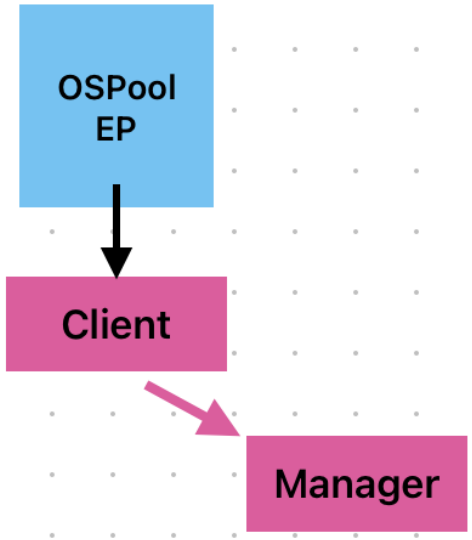
- Currently, the most common client for the OSDF is the OSPool.
- The OSPool is a distributed High Throughput Computing service, part of the OSG Consortium and run by PATh.
  - The OSPool is a distributed HTCondor pool, run across ~60 US sites, including **28 CC* awardees** (active + 'alumni').

Let's run through a HTCondor Example



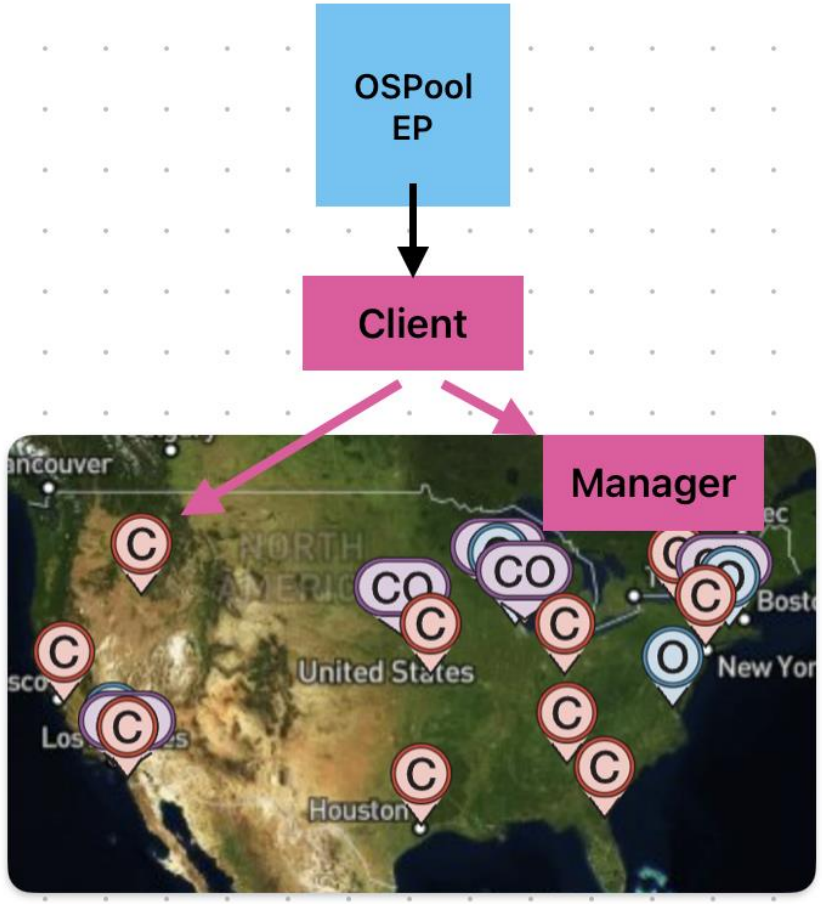
OSDF In Practice

- If HTCondor needs an object – say, a container – for a job, the first step is to start the OSDF client.
- The OSDF client contacts the **manager**, requesting to read the object.



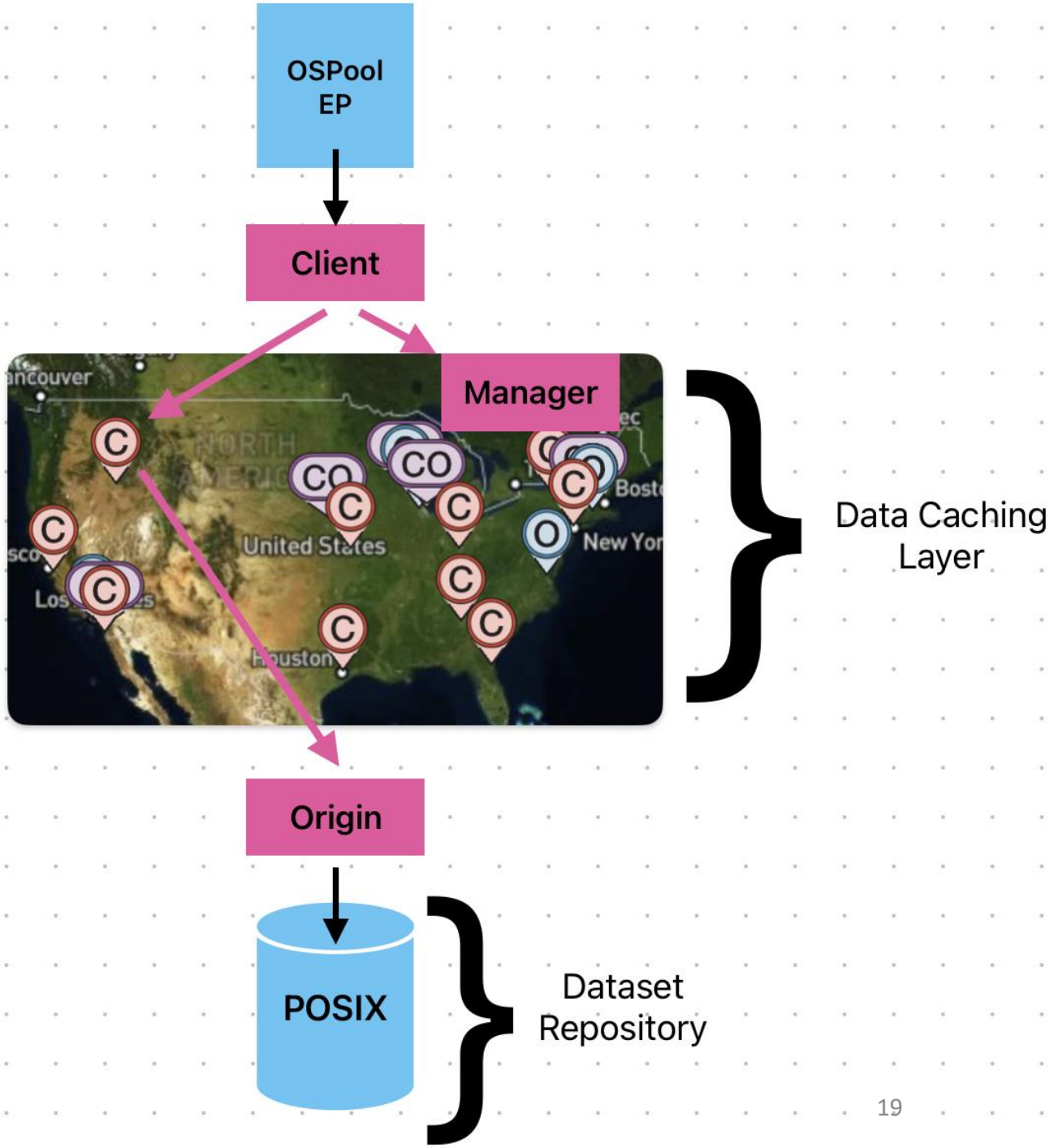
OSDF In Practice

- The manager determines a nearby **cache** to serve the object.
  - Every location in the lower 48 states is within 500 miles from an OSDF cache hosted by the NRP.
- If the object is in cache, it is served to the client immediately.
  - Otherwise…



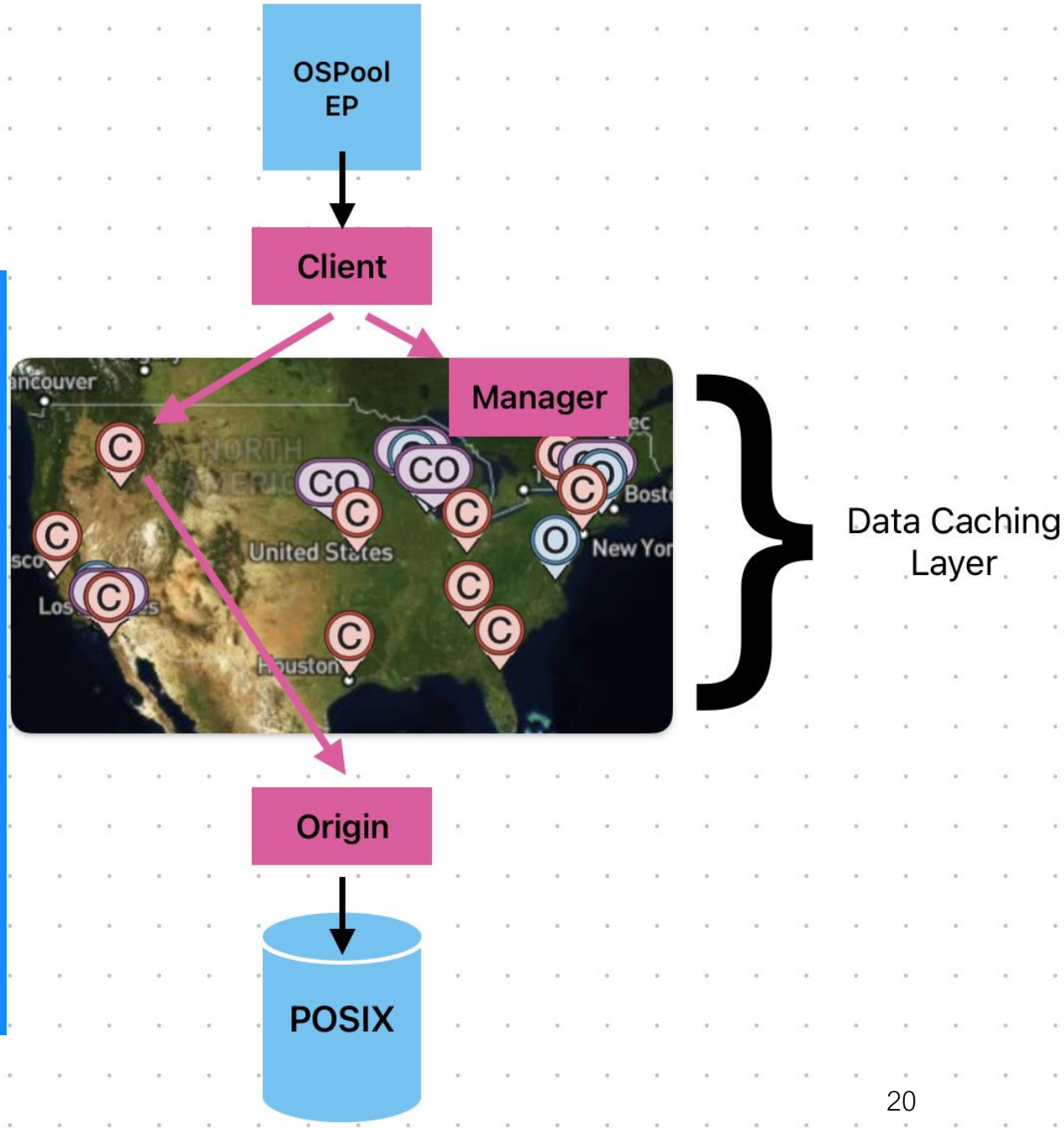
OSDF In Practice

- The cache contacts the origin hosting the object.
  - The object prefix is used as a routing key to determine the correct origin.
- The origin will read the object from the underlying object store.
  - Typically, a filesystem – but expanding to many dataset repository types!



Architecture: Recap

- An **origin service** integrates the object store into the OSDF in the same way a CE integrates a batch system into the OSPool. Interfaces to move data and map authorizations.
- The **cache service** stores and forwards objects, providing scalability to the data access.
- The **manager** selects a source/sink of an object for clients and maintains the namespace.

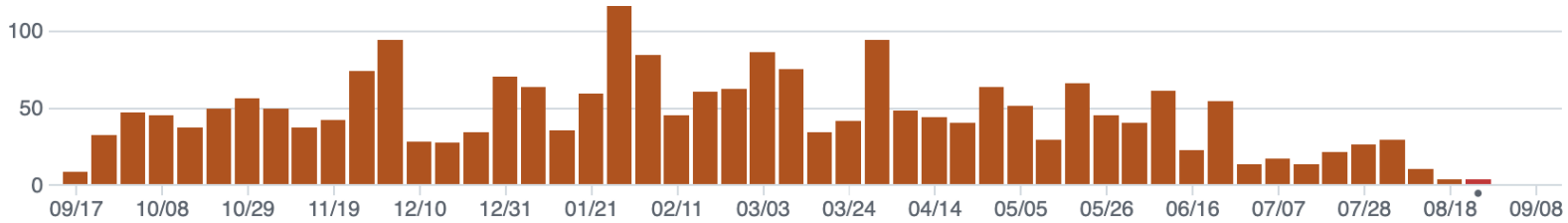


Zooming in – Technical Components

Pelican Implementation

https://github.com/PelicanPlatform/pelican

- The Pelican core is a standalone software project.
  - Golang for core; Next.js for web UI.
  - Shipped as a single statically-linked executable.
  - Fairly significant reasonable test suite (~50% code coverage).
- For origins/caches, forks & manages an XRootD process.
  - Dynamically generates XRootD configuration. One, YAML-based config file for admins to manage.
- All components have a web (management) interface.
- Distributed via RPM and containers. Majority use is containers.

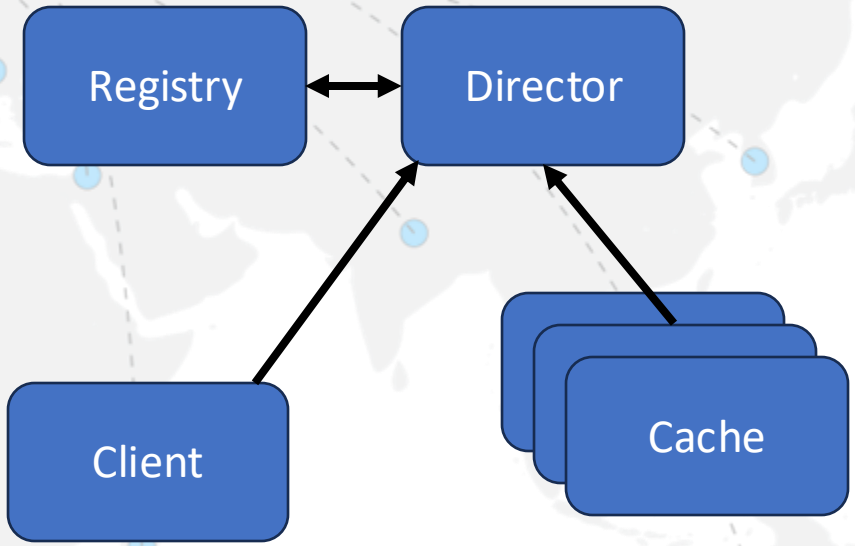


Commit graph from the last 12 months

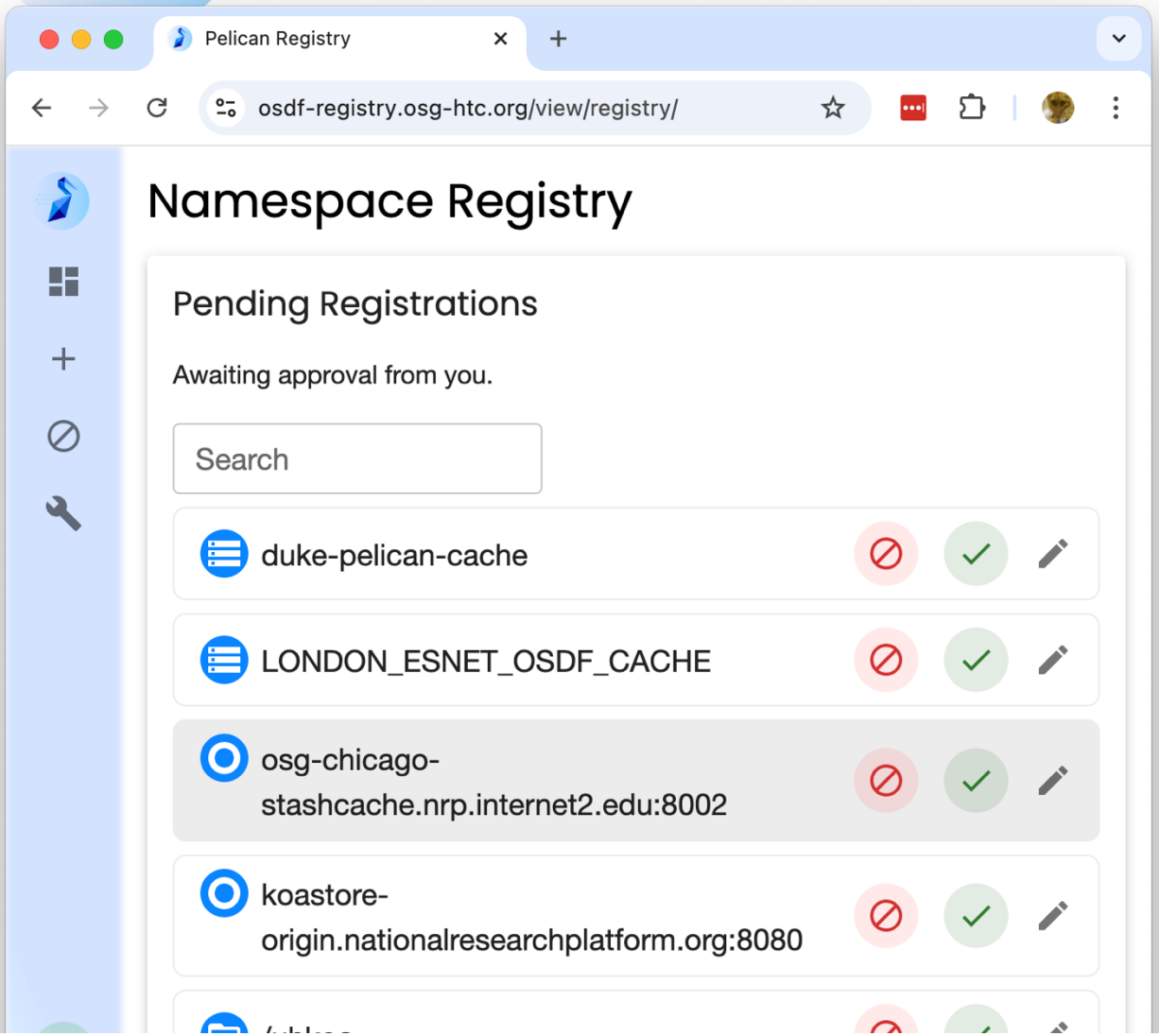
Pelican “Manager” Components

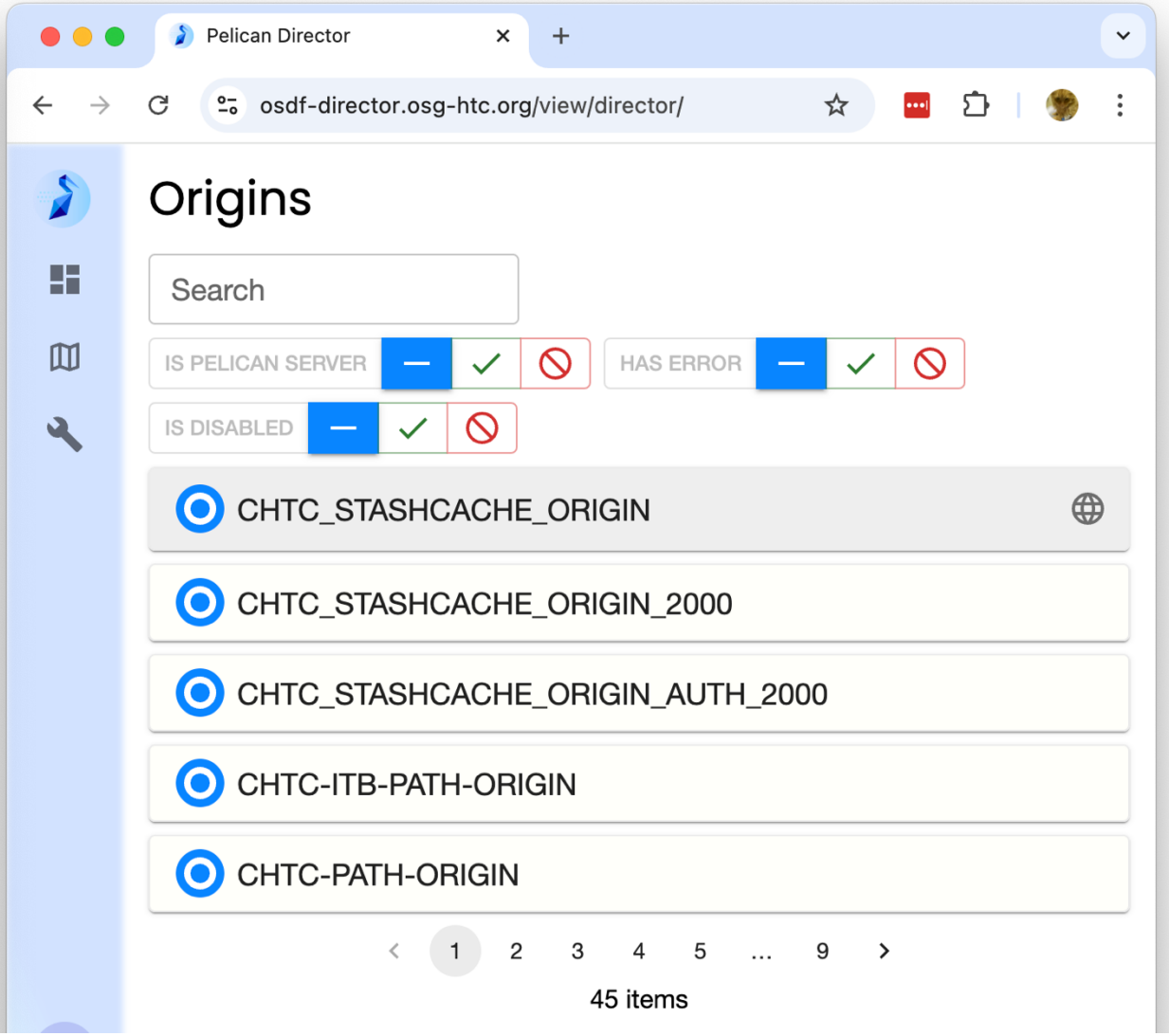
The central manager contains two components:

- The **Registry** maintains the authoritative list of known caches, origins, and namespaces.
  - Also associates each entity with a list of public keys.
  - Authorization is done by signing an appropriate token with the pubkey.
- The **Director** receives requests from clients / caches and selects an appropriate service.
  - All communication done over HTTP!



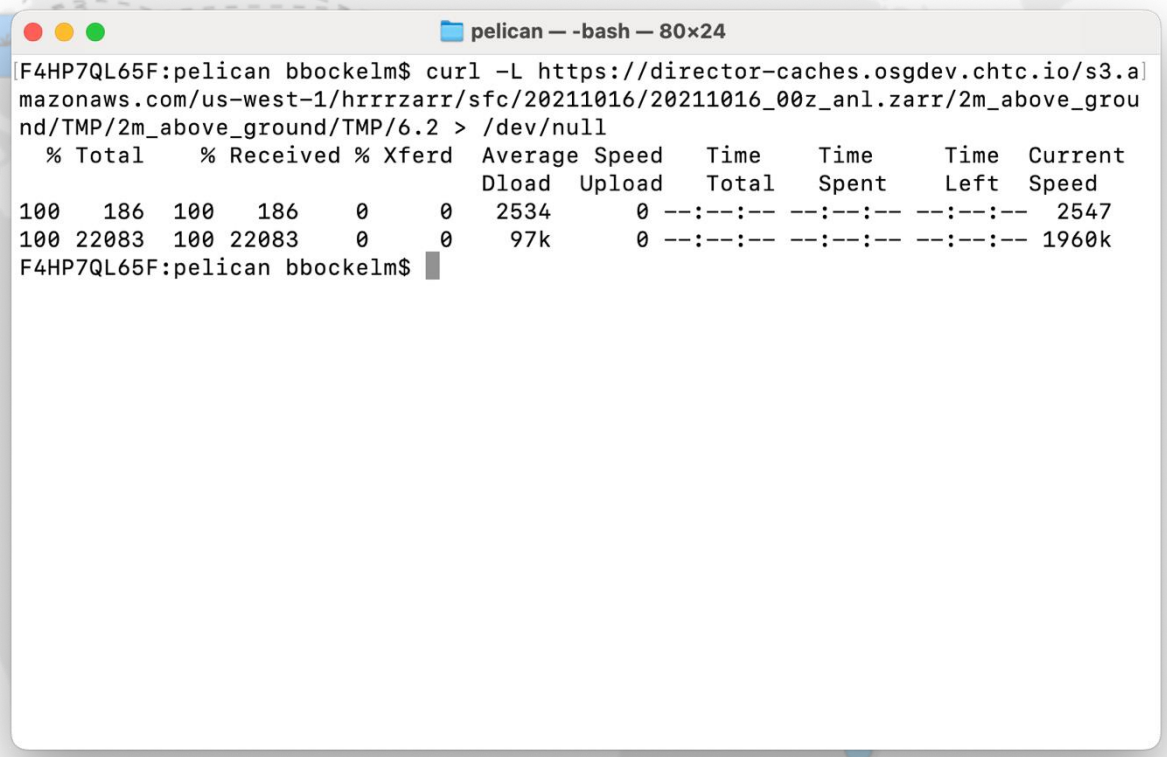
Web UI – Registry and Director





Pelican uses HTTP

- Pelican uses HTTP to move bytes.
- We hew to using standard HTTP where possible. While we *prefer* you use the Pelican client, any HTTP client suffices.
  - Downloading an object? => GET
  - Uploading an object? => PUT
  - Want to know if the object exists? => HEAD



Example request from client to director

> GET /chtc/staging/bbockelm/testfile HTTP/2
> Host: osdf-director.osg-htc.org
> User-Agent: curl/8.4.0
> Accept: */*

Example director response

< HTTP/2 307
< content-type: text/html; charset=utf-8
< date: Mon, 08 Jul 2024 17:17:17 GMT
< link: <https://osdf-uw-cache.svc.osg-htc.org:8443/chtc/staging/bbockelm/testfile>; rel="duplicate"; pri=1; depth=3, <https://stash-cache.osg.chtc.io:8443/chtc/staging/bbockelm/testfile>; rel="duplicate"; pri=2; depth=3,...
< location: https://osdf-uw-cache.svc.osg-htc.org:8443/chtc/staging/bbockelm/testfile
< x-pelican-authorization: issuer=https://chtc.cs.wisc.edu
< x-pelican-namespace: namespace=/chtc, require-token=true, collections-url=https://origin-auth2000.chtc.wisc.edu:1095
< x-pelican-token-generation: issuer=https://chtc.cs.wisc.edu, max-scope-depth=3, strategy=OAuth2
< content-length: 109

Example director response

< HTTP/2 307
< content-type: text/html; charset=utf-8
< date: Mon, 08 Jul 2024 17:17:17 GMT
< link: <https://osdf-uw-cache.svc.osg-htc.org:8443/chtc/staging/bbockelm/testfile>; rel="duplicate"; pri=1; depth=3, <https://stash-cache.osg.chtc.io:8443/chtc/staging/bbockelm/testfile>; rel="duplicate"; pri=2; depth=3,...
< location: https://osdf-uw-cache.svc.osg-htc.org:8443/chtc/staging/bbockelm/testfile
< x-pelican-authorization: issuer=https://chtc.cs.wisc.edu
< x-pelican-namespace: namespace=/chtc, require-token=true, collections-url=https://origin-auth2000.chtc.wisc.edu:1095
< x-pelican-token-generation: issuer=https://chtc.cs.wisc.edu, max-scope-depth=3, strategy=OAuth2
< content-length: 109

Example director response

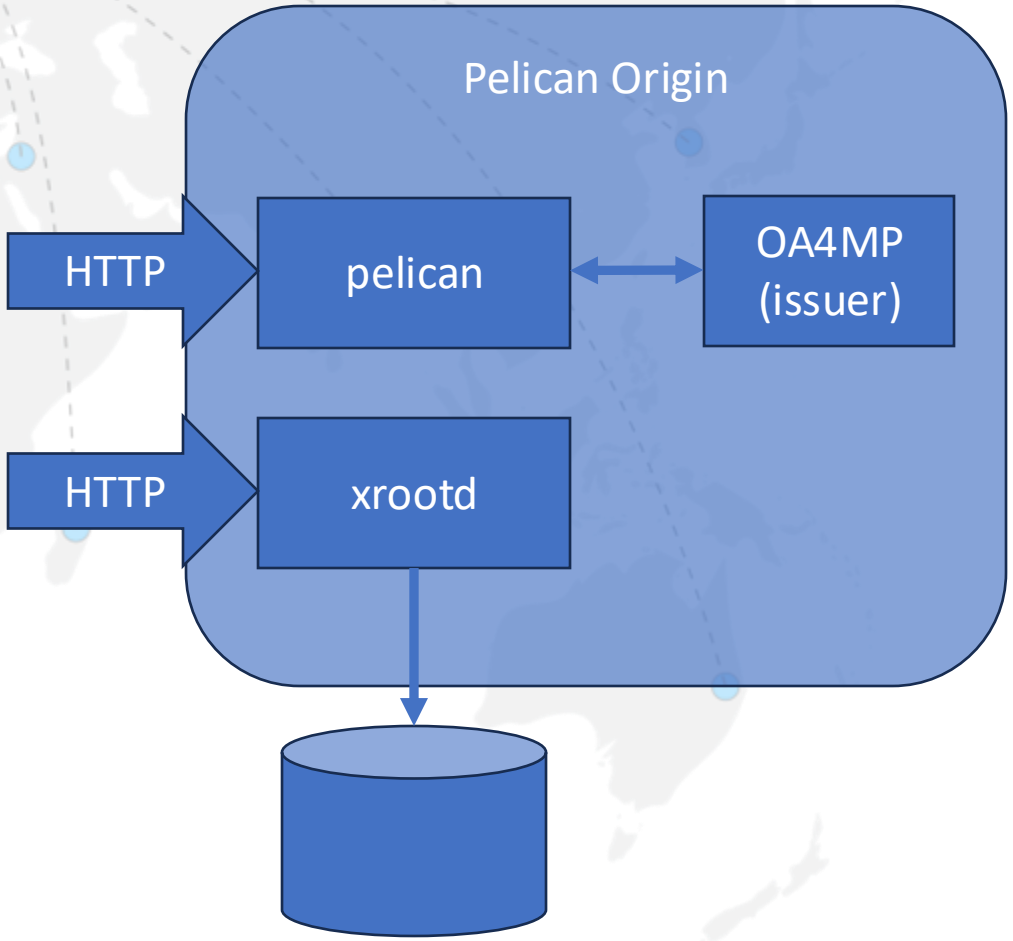
< HTTP/2 307
< content-type: text/html; charset=utf-8
< date: Mon, 08 Jul 2024 17:17:17 GMT
< link: <https://osdf-uw-cache.svc.osg-htc.org:8443/chtc/staging/bbockelm/testfile>; rel="duplicate"; pri=1; depth=3, <https://stash-cache.osg.chtc.io:8443/chtc/staging/bbockelm/testfile>; rel="duplicate"; pri=2; depth=3,...
< location: https://osdf-uw-cache.svc.osg-htc.org:8443/chtc/staging/bbockelm/testfile
< x-pelican-authorization: issuer=https://chtc.cs.wisc.edu
< x-pelican-namespace: namespace=/chtc, require-token=true, collections-url=https://origin-auth2000.chtc.wisc.edu:1095
< x-pelican-token-generation: issuer=https://chtc.cs.wisc.edu, max-scope-depth=3, strategy=OAuth2
< content-length: 109

Director Response

- If you speak “plain HTTP”, you only understand the “blue” headers and will successfully access the data.
- If you are the “Pelican client”, you can interpret the “red” headers:
  - X-pelican-authorization: What token the client needs to successfully access the data.
  - X-pelican-namespace: What namespace the object is in. Informs client how to reuse the director response; no need to return to director for each object.
  - X-pelican-token-generation: If the client doesn’t have a usable token, how to receive one.
  - Link: An ordered list of potential endpoints (caches) that can serve the requests. Actually, a standard RFC header (RFC 6249).

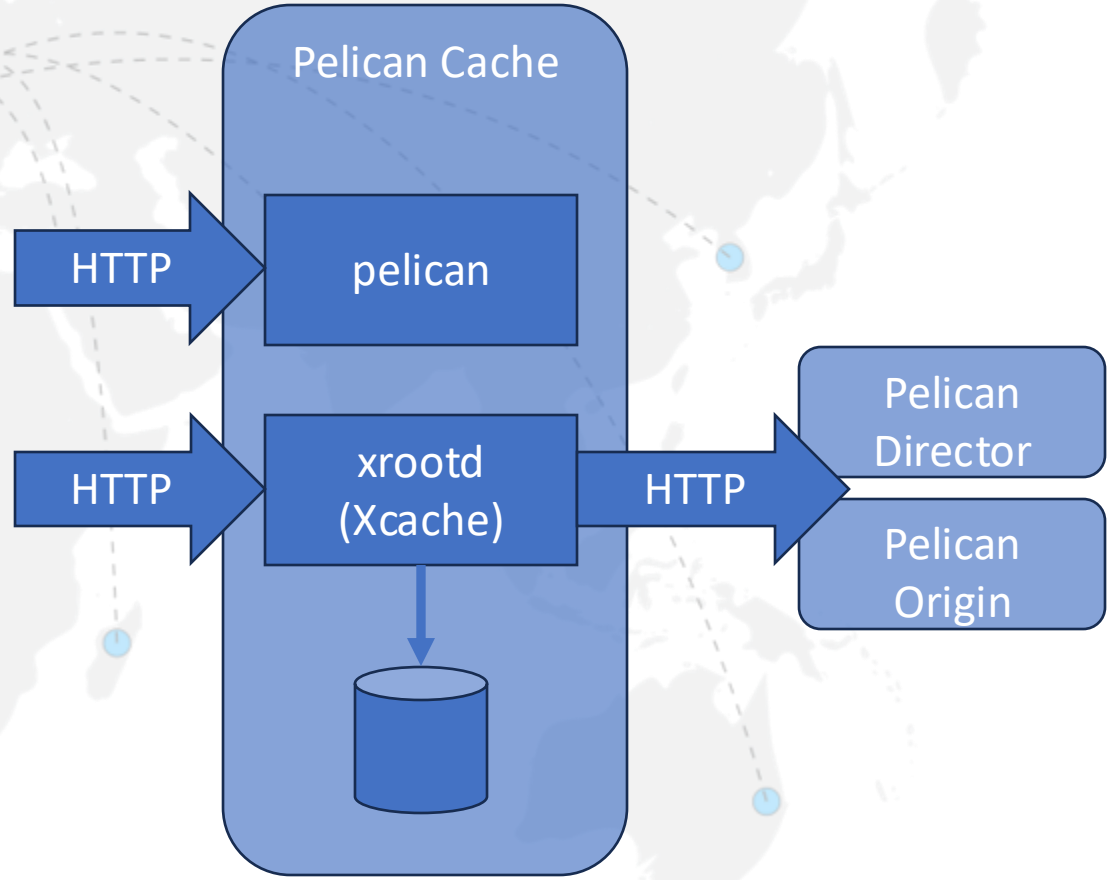
Pelican Origin

- Pelican daemon launches and manages the xrootd daemon.
  - However, HTTP data movement requests go straight to the xrootd process.
  - pelican's HTTP interface is used for monitoring, management, and token issuer.
- XRootD can be configured for a variety of backends.

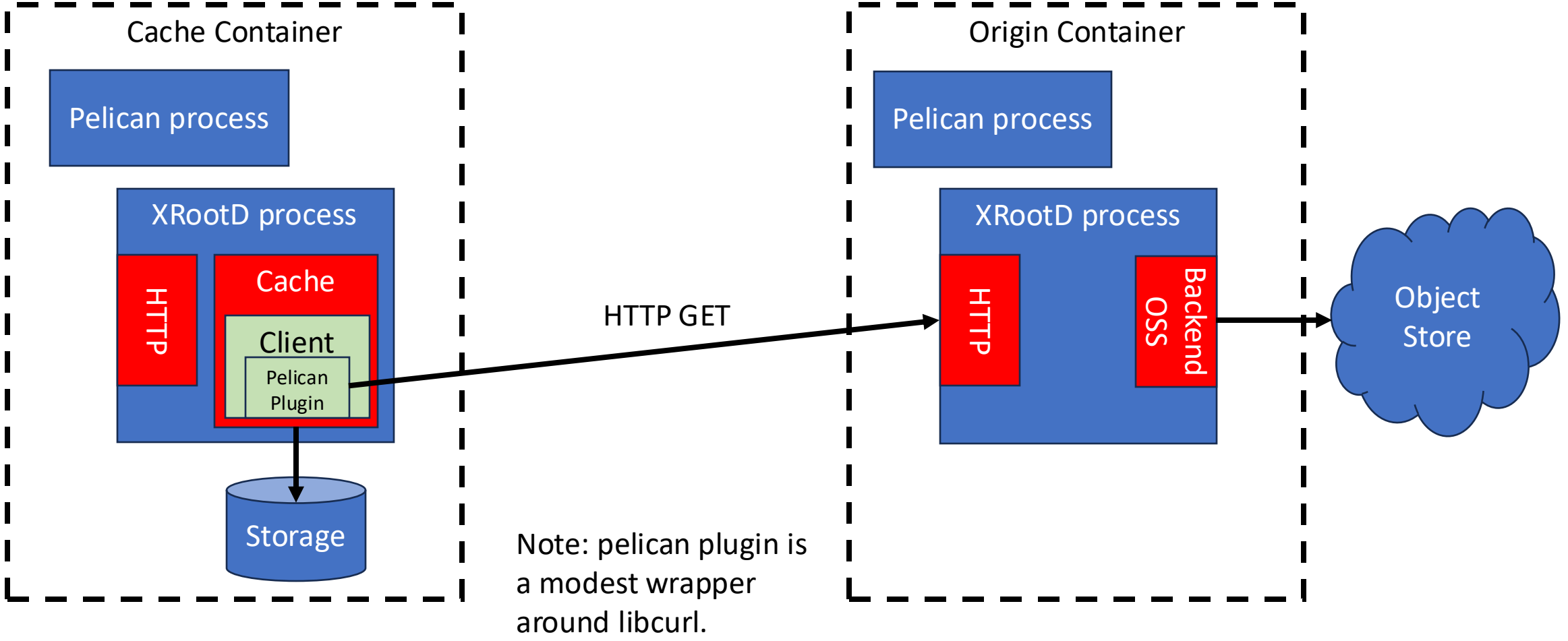


Pelican Cache

- Similar setup to the origin: two separate processes, two ports for HTTP.
  - Given the director and origin works exclusively over HTTP, the XCache must talk to them over HTTP as well.
  - How is this done? See next talk!



A slide for the XRootD people out there…



Client - CLI

- While curl *can* be used, we have quite a bit of specialized knowledge:
  - Immutable files means file download resumption is straightforward.
  - Parse the extra director headers to understand where backup caches are. Retry as necessary.
  - From the director headers, we know what tokens are required and how to generate them.
- The client can also do metadata operations (“stat”, “list”), recursive upload/downloads of directories.
- The client also serves as a plugin to HTCondor, enabling HTCondor to do the data movement (instead of buried inside user scripts).
- The client is all in the same static binary as the server – the entire system is the one file.

Client - Python

- While we love CLIs, we want to tap into the Python community (which is more interactive/visualization focused).
- Accordingly, we started a FSSpec for Pelican.
  - Summer student was able to use the FSSpec to run PyTorch against the OSDF.
- Allows us to tap into more communities (particularly, a large contingent of climate science).

Monitoring

- Pelican natively uses Prometheus for monitoring.
  - Embeds Prometheus itself, meaning a full Prometheus setup is at each origin/cache: **You have all monitoring you need locally!**
- Embeds an instance of the Shoveler. Allows you to forward to non-ES systems.
- The built-in Prometheus monitoring powers the web dashboards and (increasingly) used for performance statistics from both XRootD and Pelican.

Zooming out

Pelican Year 1 – quite the whirlwind!

- We reengineered the origin and cache services, added new central services, and greatly improved the OSDF's integration with HTCSS.
  - OSDF saw corresponding enormous growth, with some days moving >2PB.
- We've picked up new science partners (notably, NCAR) and supported some great science (NRAO).
- Working to provide more visibility into the system: what's my cache hardware doing? who's using my objects? who am I impacting?

Planning Ahead

- Development only:
  - Additional backends (DataVerse), stabilizing/completing the new-er backends.
  - More robust web interface, better monitoring dashboards.
  - Additional configuration options for the issuer.
  - New “collections API”, allowing user sharing at the prefix level.
  - Client functionality around synchronizing.
- Larger project:
  - Much to do in training, documentation.
  - Expand within the climate user community.
  - Expand work with some bigger use cases (LIGO, NRAO).

The OSDF: Connecting to your datasets

To acknowledge all of the partners working together…

OAC-2331480

Provides the software

OAC-2030508

Operates the OSDF services

OAC-2112167

Operates (most of) the OSDF hardware

The OSG Consortium is the “umbrella” we work within.

Questions?

This project is supported by the National Science Foundation under Cooperative Agreements OAC-2331480. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.