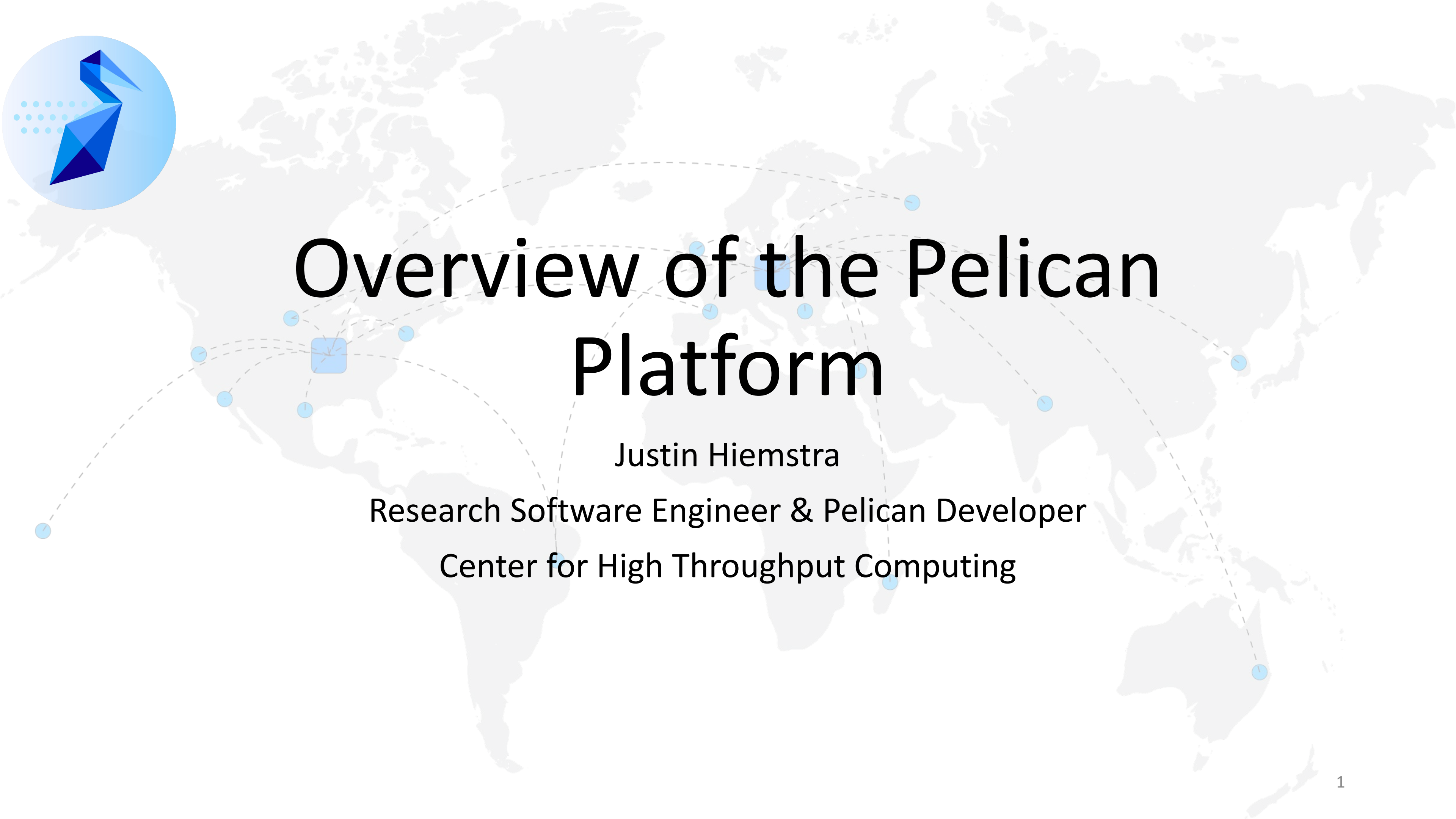


Overview of the Pelican Platform

Justin Hiemstra

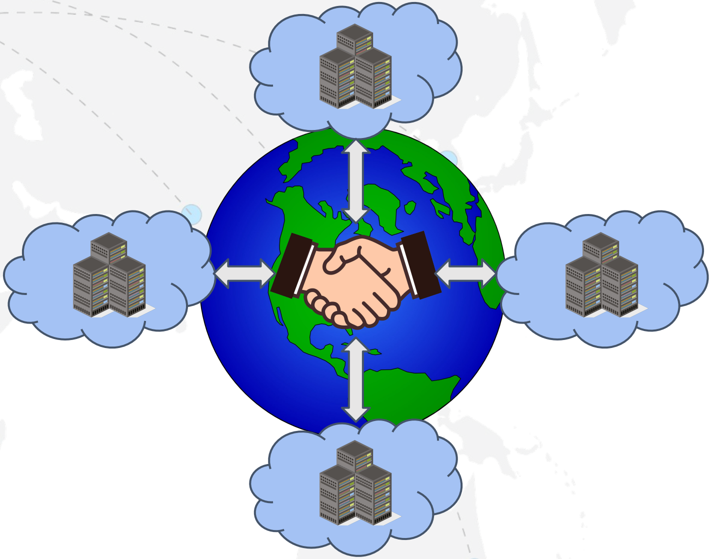
Research Software Engineer & Pelican Developer

Center for High Throughput Computing

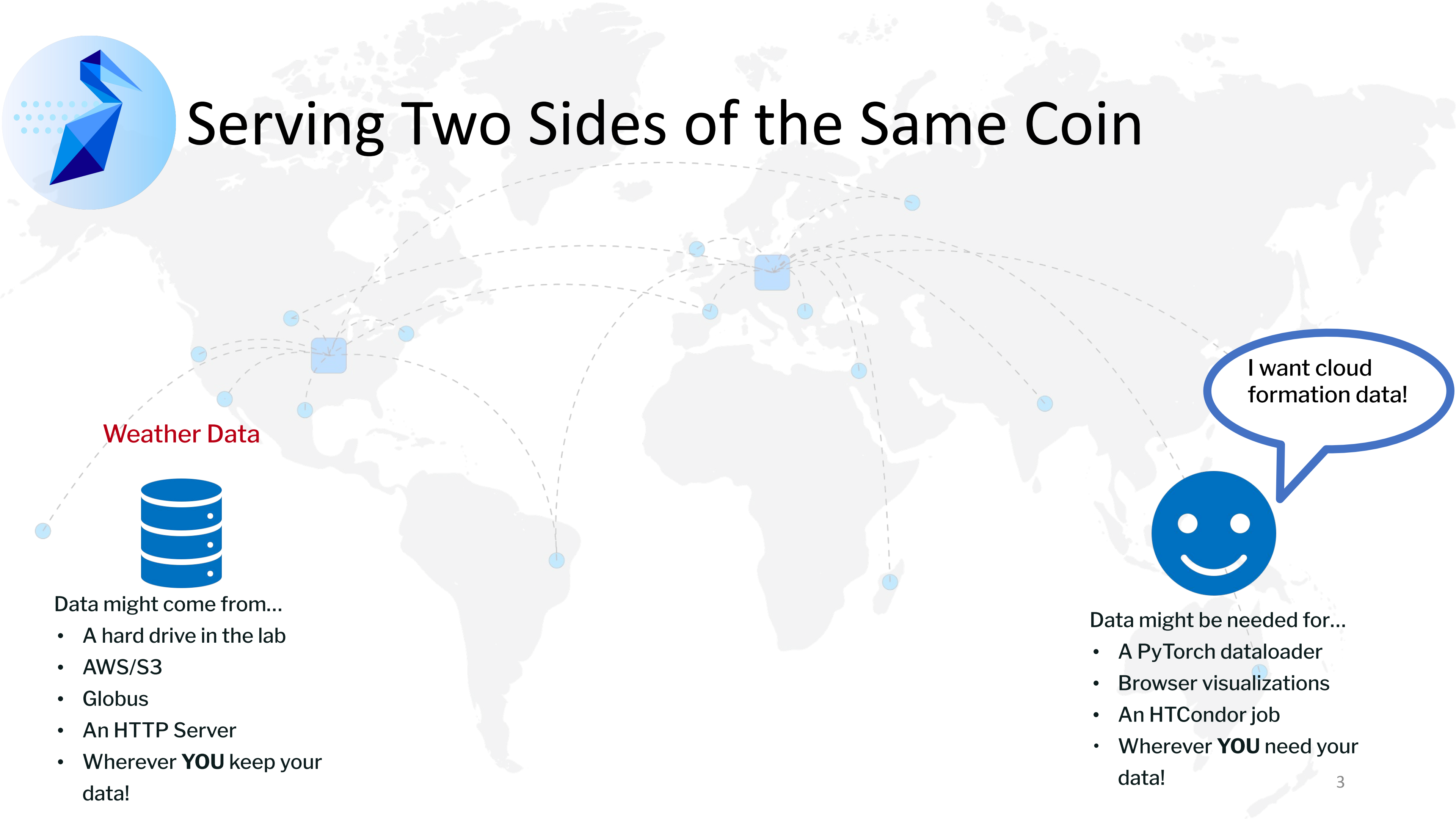
Why Data Federations?

At its core, the Pelican Platform is a set of tools for creating/managing **Data Federations**. Our use cases dictate that:

- Data owners may come and go as they like
- Data owners have the ultimate authority to choose how their data is used/distributed
- Unified namespaces, decentralized storage – to a user, everything feels like it's coming from the same source
- Through combining resources while respecting individual needs, we can tackle bigger challenges → Scalability!



Serving Two Sides of the Same Coin



Weather Data

Data might come from…

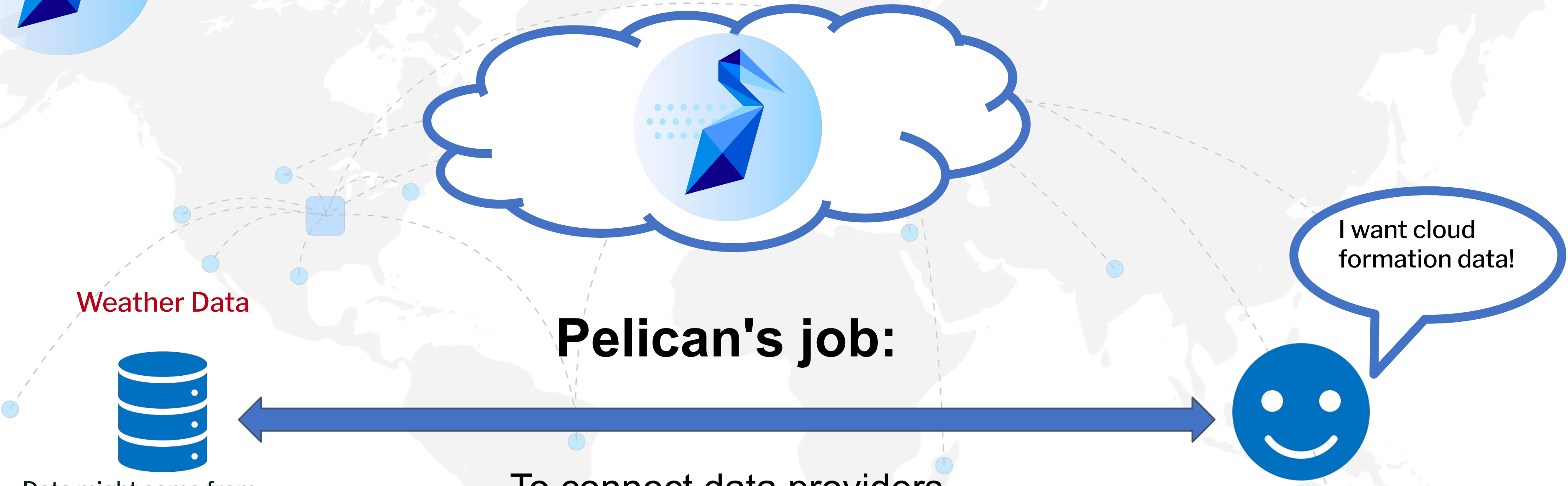
- A hard drive in the lab
- AWS/S3
- Globus
- An HTTP Server
- Wherever **YOU** keep your data!

I want cloud formation data!

Data might be needed for…

- A PyTorch dataloader
- Browser visualizations
- An HTCondor job
- Wherever **YOU** need your data!

Serving Two Sides of the Same Coin



Pelican's job:

To connect data providers with data consumers

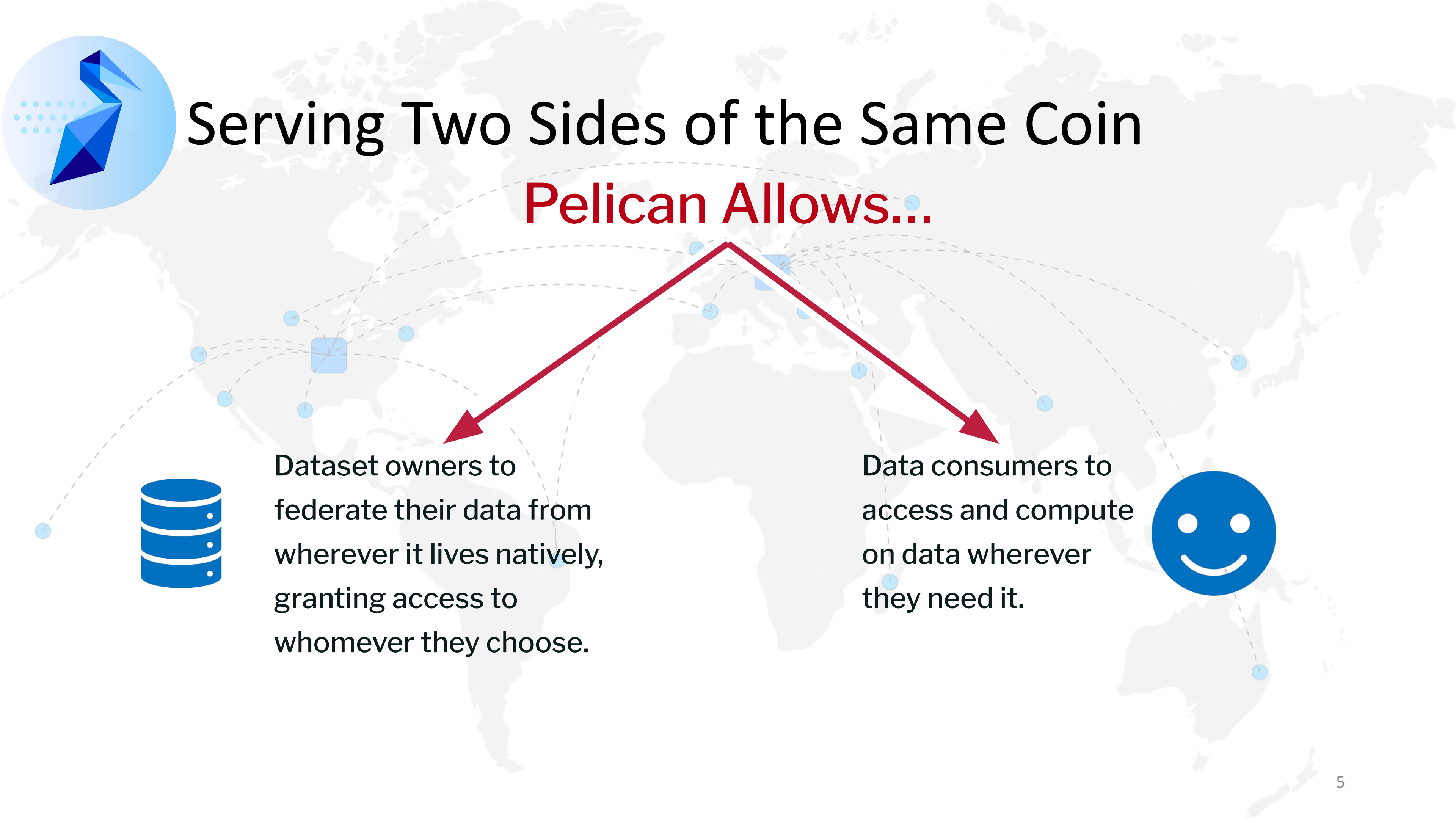
Data might come from…

- A hard drive in the lab
- AWS/S3
- Globus
- An HTTP Server
- Wherever **YOU** keep your data!

Data might be needed for…

- A PyTorch dataloader
- Browser visualizations
- An HTCondor job
- Wherever **YOU** need your data!

Serving Two Sides of the Same Coin



Pelican Allows…

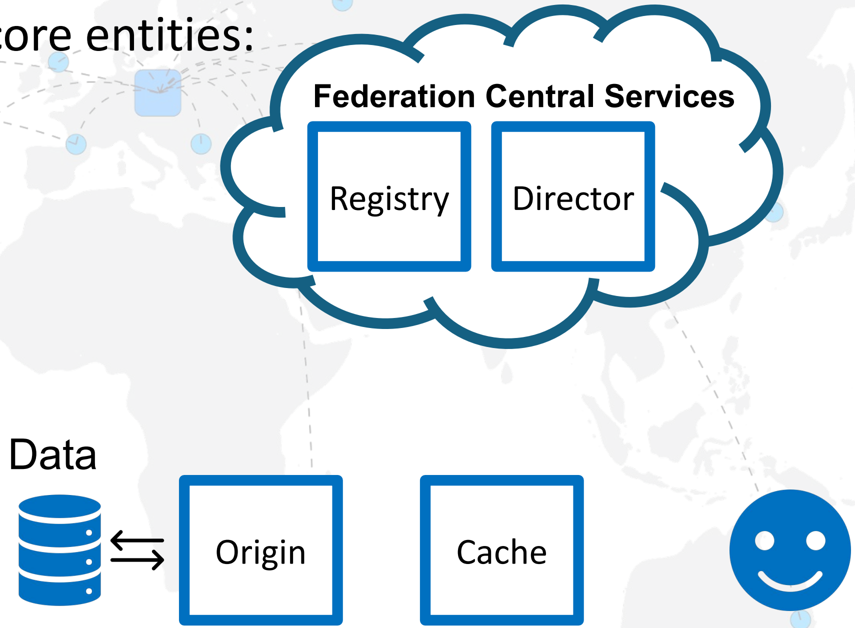
Dataset owners to federate their data from wherever it lives natively, granting access to whomever they choose.

Data consumers to access and compute on data wherever they need it.

Zooming In – Technical Components

Origins, Caches, Registries, and Directors, OH MY!

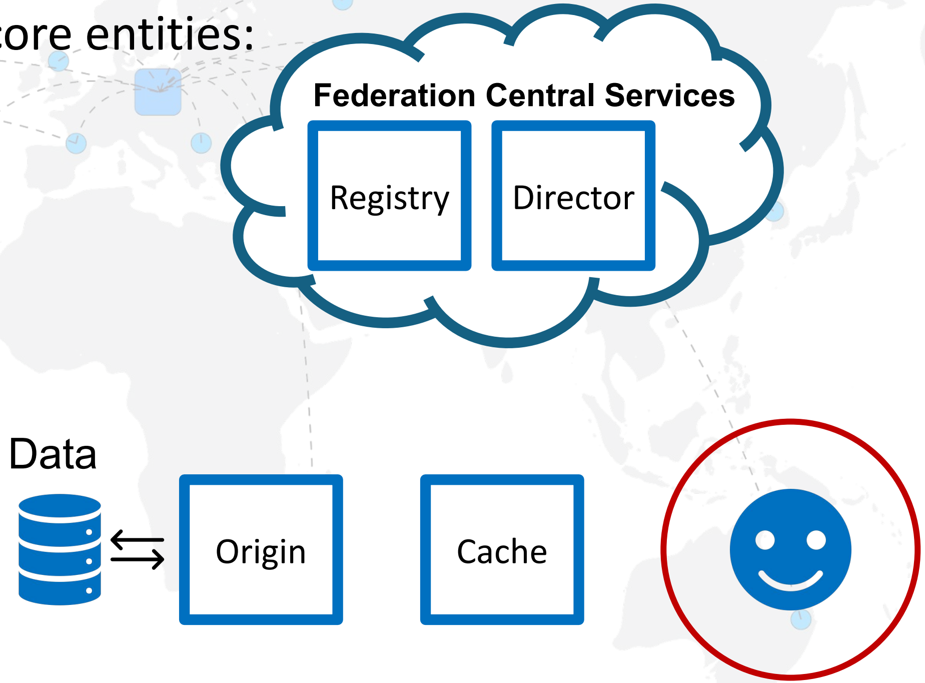
Pelican federations are made up of 6 core entities:



Origins, Caches, Registries, and Directors, OH MY!

Pelican federations are made up of 6 core entities:

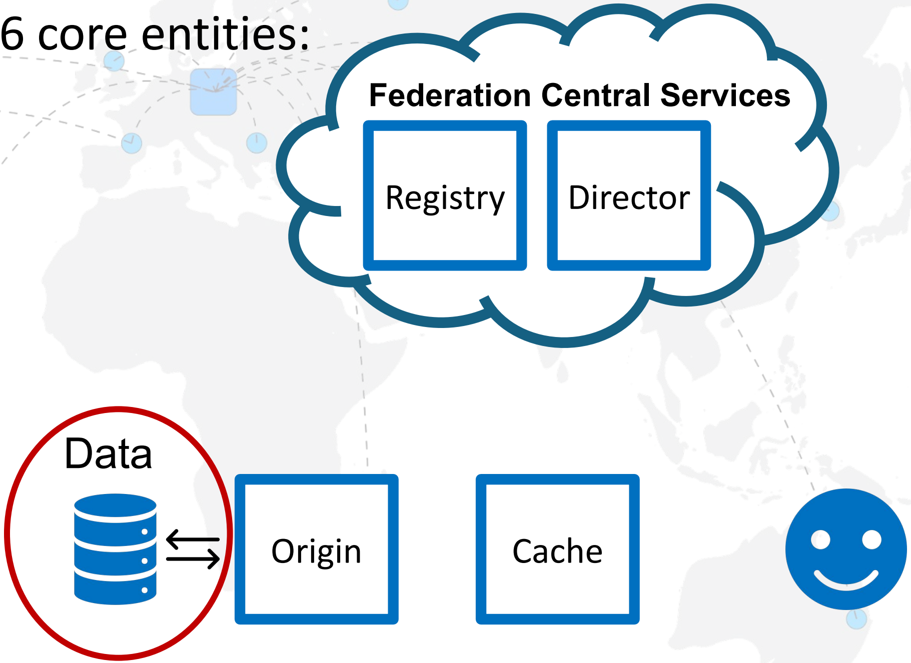
1. **Clients** – used to interact with objects, perform uploads/downloads, etc.



Origins, Caches, Registries, and Directors, OH MY!

Pelican federations are made up of 6 core entities:

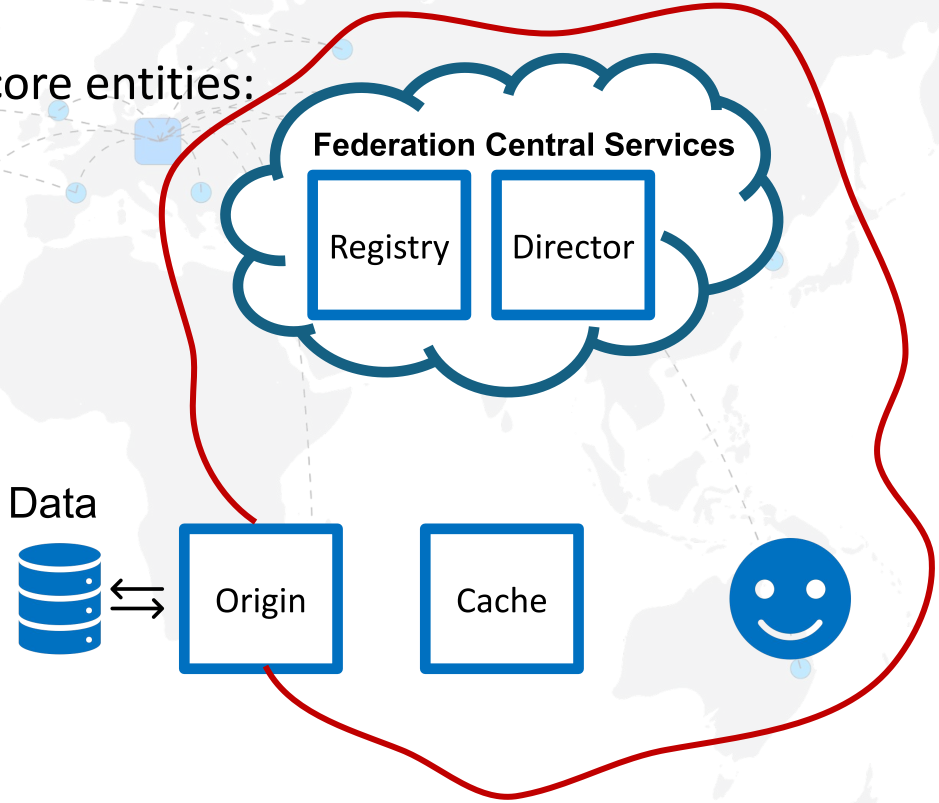
1. **Clients** – used to interact with objects, perform uploads/downloads, etc.
2. **Object Store** – where the data actually lives



Origins, Caches, Registries, and Directors, OH MY!

Pelican federations are made up of 6 core entities:

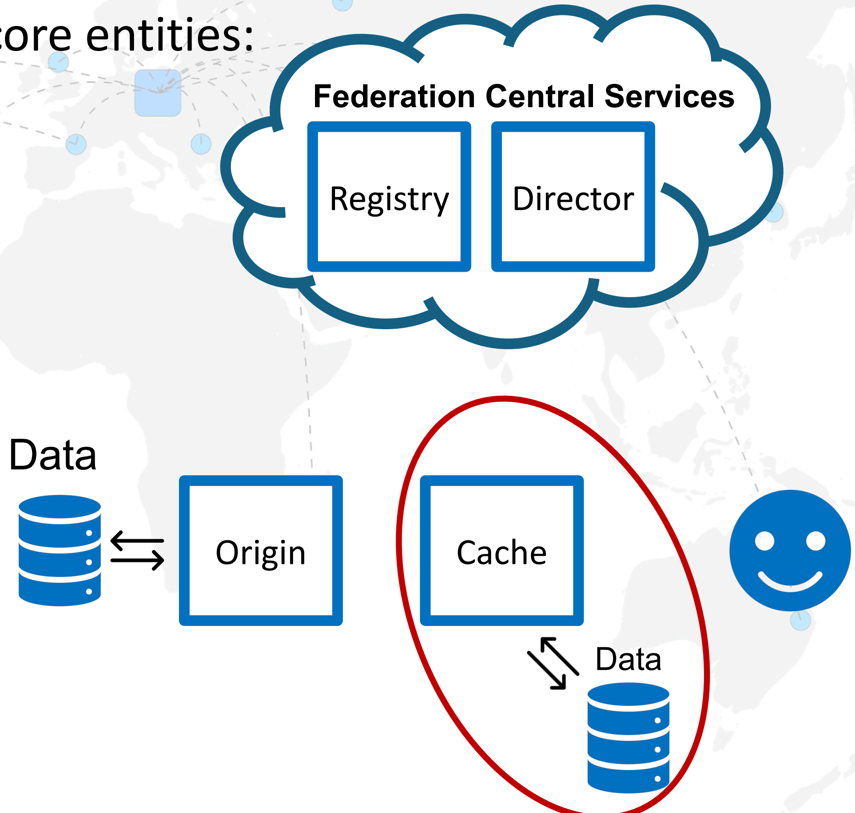
1. **Clients** – used to interact with objects, perform uploads/downloads, etc.
2. **Object Store** – where the data actually lives
3. **Origins** – acts as a connector between federation and repository



Origins, Caches, Registries, and Directors, OH MY!

Pelican federations are made up of 6 core entities:

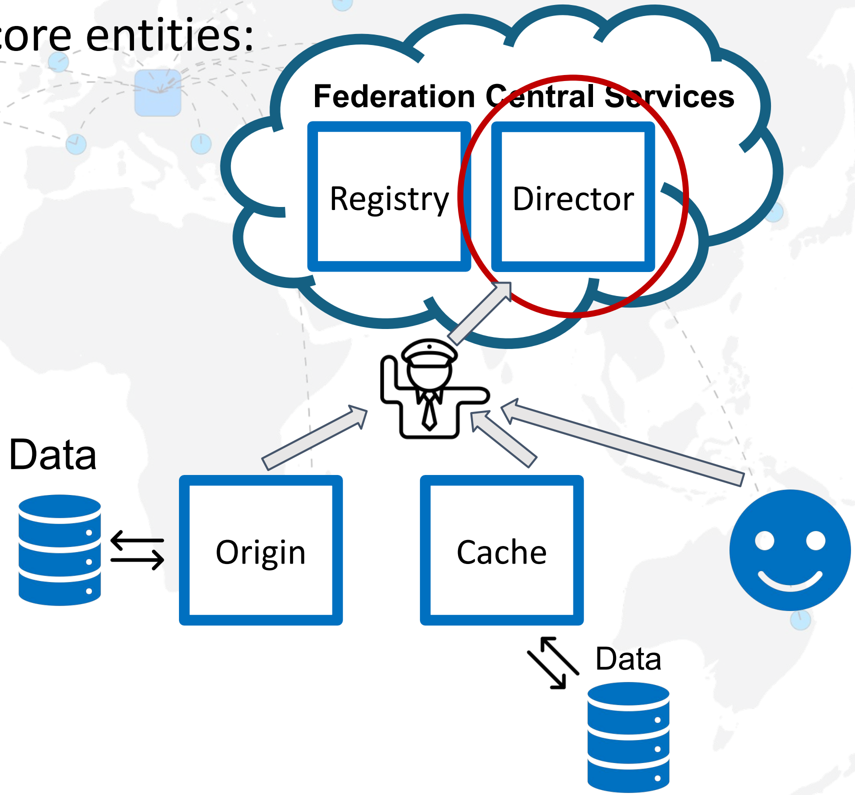
1. **Clients** – used to interact with objects, perform uploads/downloads, etc.
2. **Object Store** – where the data actually lives
3. **Origins** – acts as a connector between federation and repository
4. **Caches** – stores copies of objects in federation



Origins, Caches, Registries, and Directors, OH MY!

Pelican federations are made up of 6 core entities:

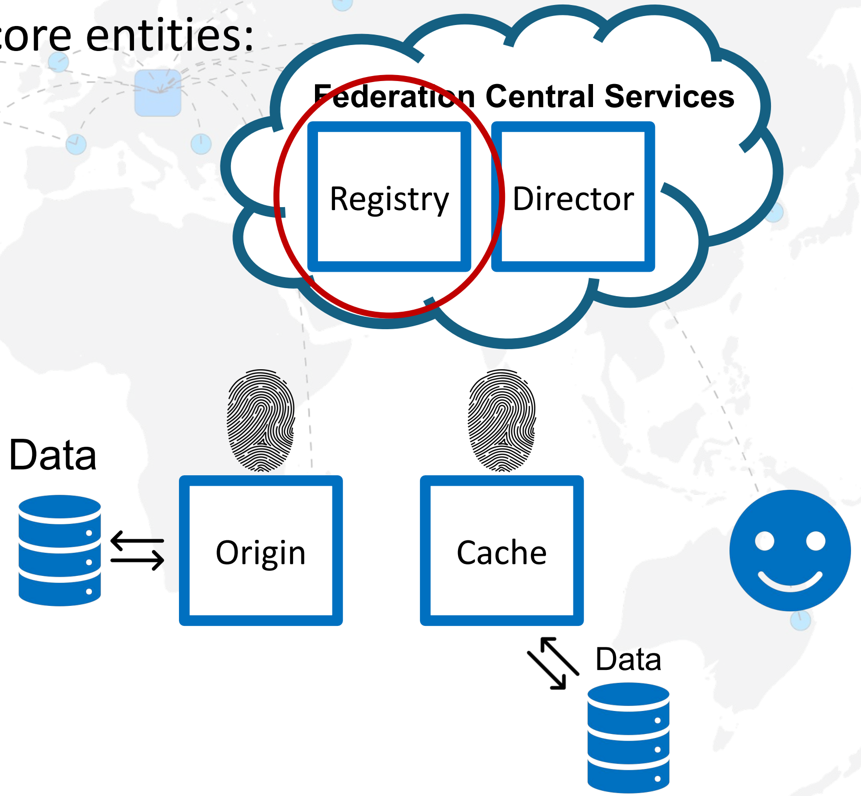
1. **Clients** – used to interact with objects, perform uploads/downloads, etc.
2. **Object Store** – where the data actually lives
3. **Origins** – acts as a connector between federation and repository
4. **Caches** – stores copies of objects in federation
5. **Director** – tells people where to get the data they're looking for



Origins, Caches, Registries, and Directors, OH MY!

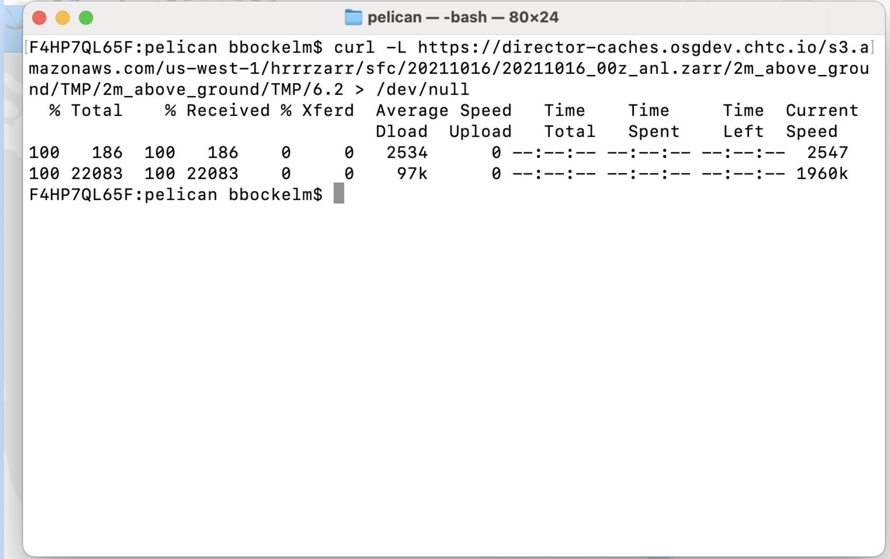
Pelican federations are made up of 6 core entities:

1. **Clients** – used to interact with objects, perform uploads/downloads, etc.
2. **Object Store** – where the data actually lives
3. **Origins** – acts as a connector between federation and repository
4. **Caches** – stores copies of objects in federation
5. **Director** – tells people where to get the data they're looking for
6. **Registry** – persistent storage for identity information



Pelican Uses HTTP

- Pelican uses HTTP to move bytes.
- While Pelican Clients come bundled with nice-to-haves and 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
  - Need a list of prefix-matches? => PROPFIND



```
F4HP7QL65F:pelican bbockelm$ curl -L https://director-caches.osgdev.chtc.io/s3.a
mazonaws.com/us-west-1/hrrrzarr/sfc/20211016/20211016_00z_anl.zarr/2m_above_grou
nd/TMP/2m_above_ground/TMP/6.2 > /dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   186  100   186    0     0   2534      0 --:--:-- --:--:-- --:--:--  2547
100 22083  100 22083    0     0    97k      0 --:--:-- --:--:-- --:--:-- 1960k
F4HP7QL65F:pelican bbockelm$
```

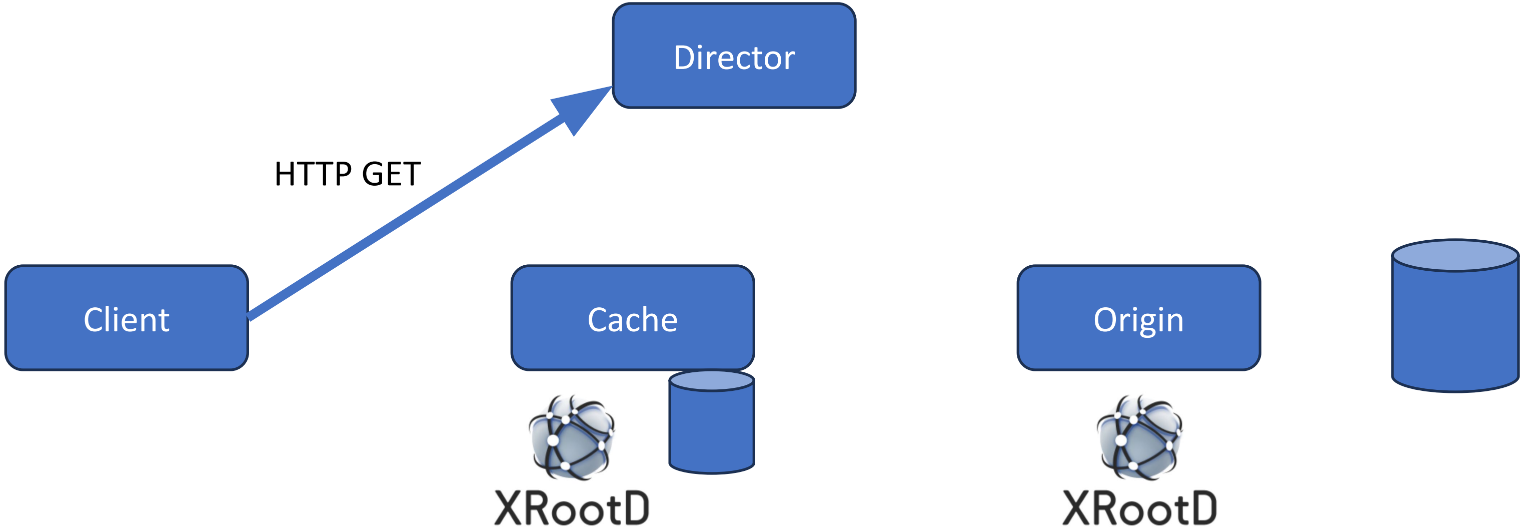
Client – CLI

- While curl *can* be used, we have quite a bit of specialized knowledge:
  - Immutable objects means 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, coupling distributed high throughput computing with distributed high throughput data management/transfer.
- The Client is all in the same static binary as the server – the entire system is the one file.

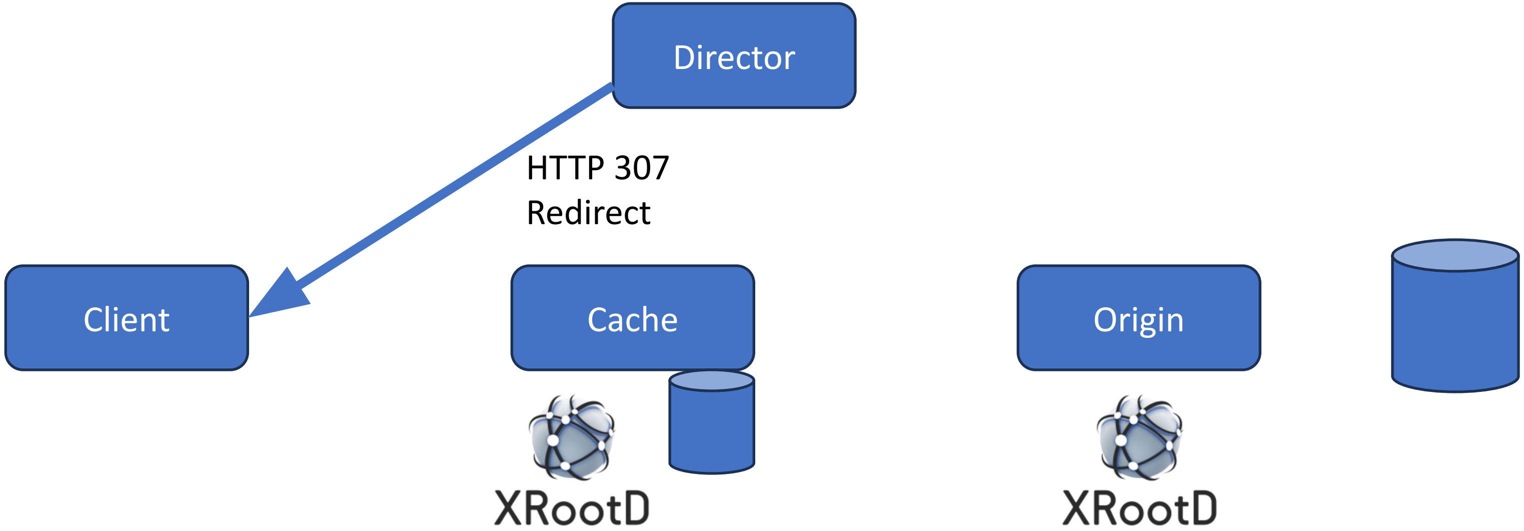
Client – Python

- 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).

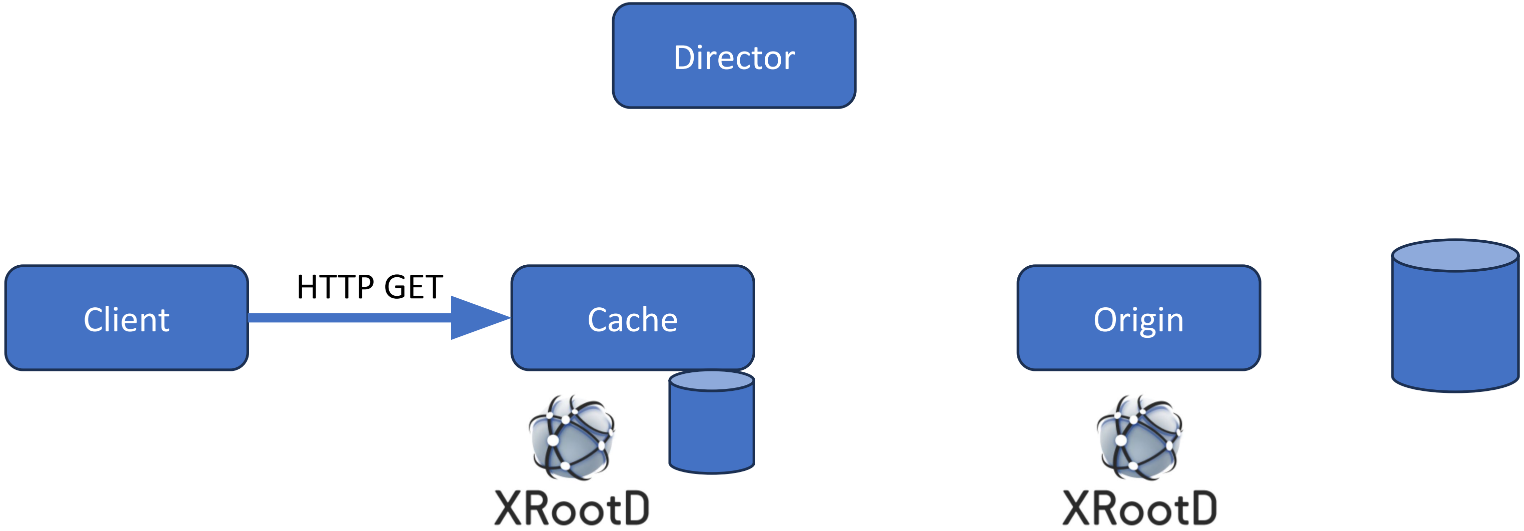
Pelican Data Flow



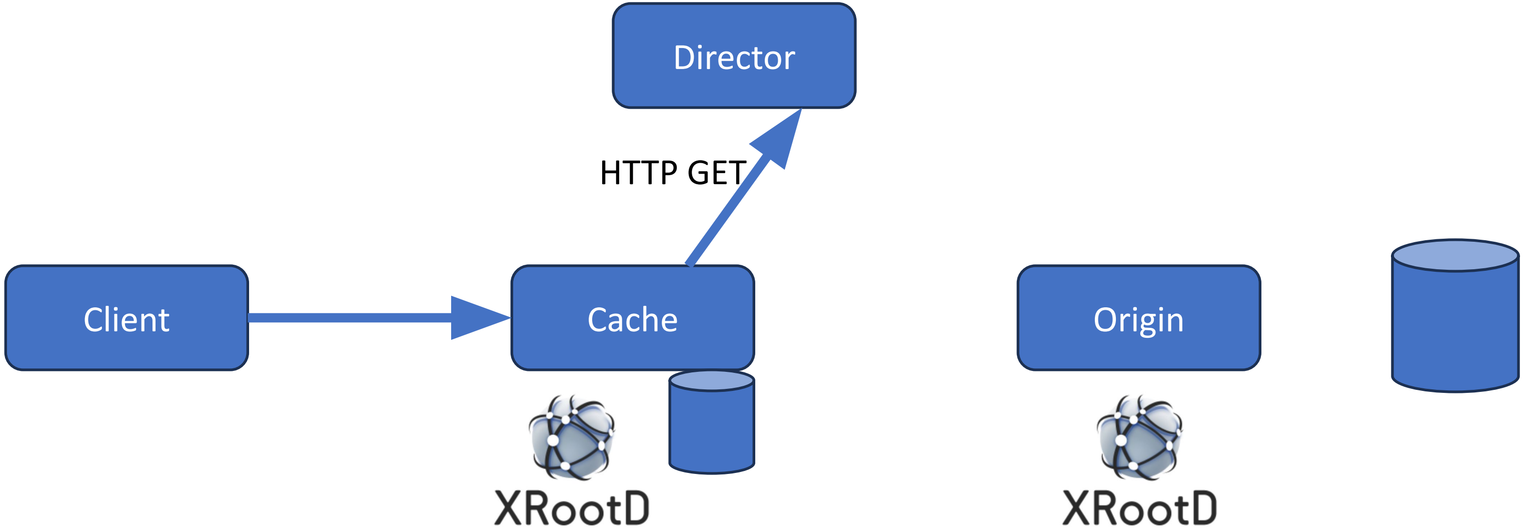
Pelican Data Flow



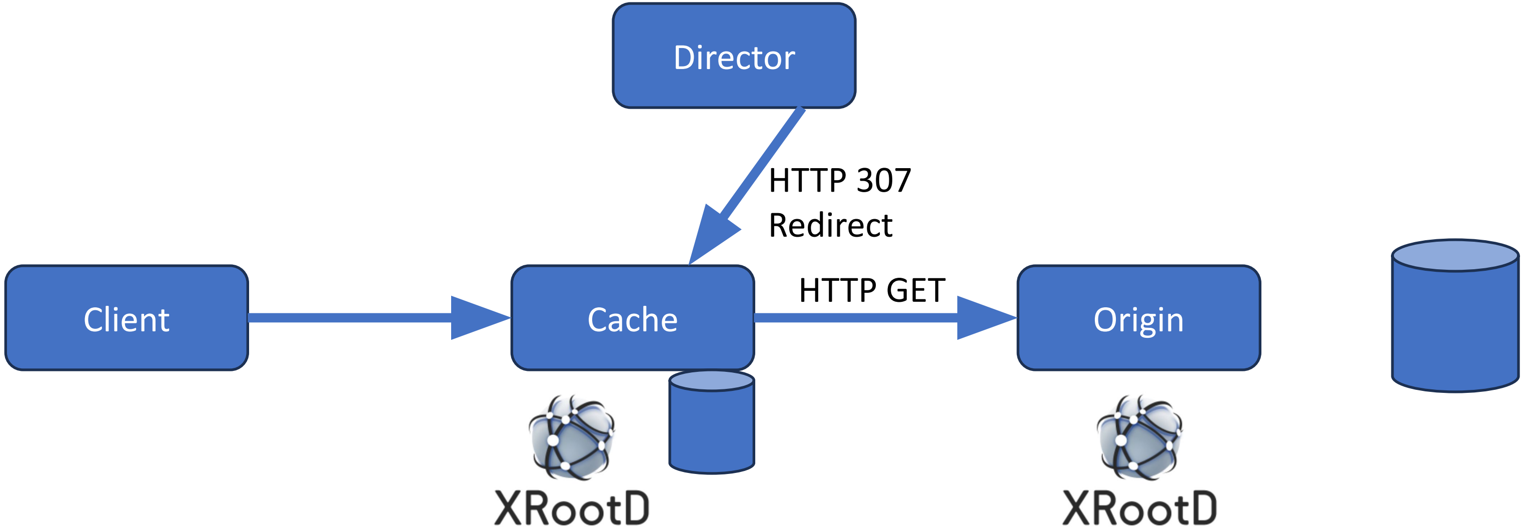
Pelican Data Flow



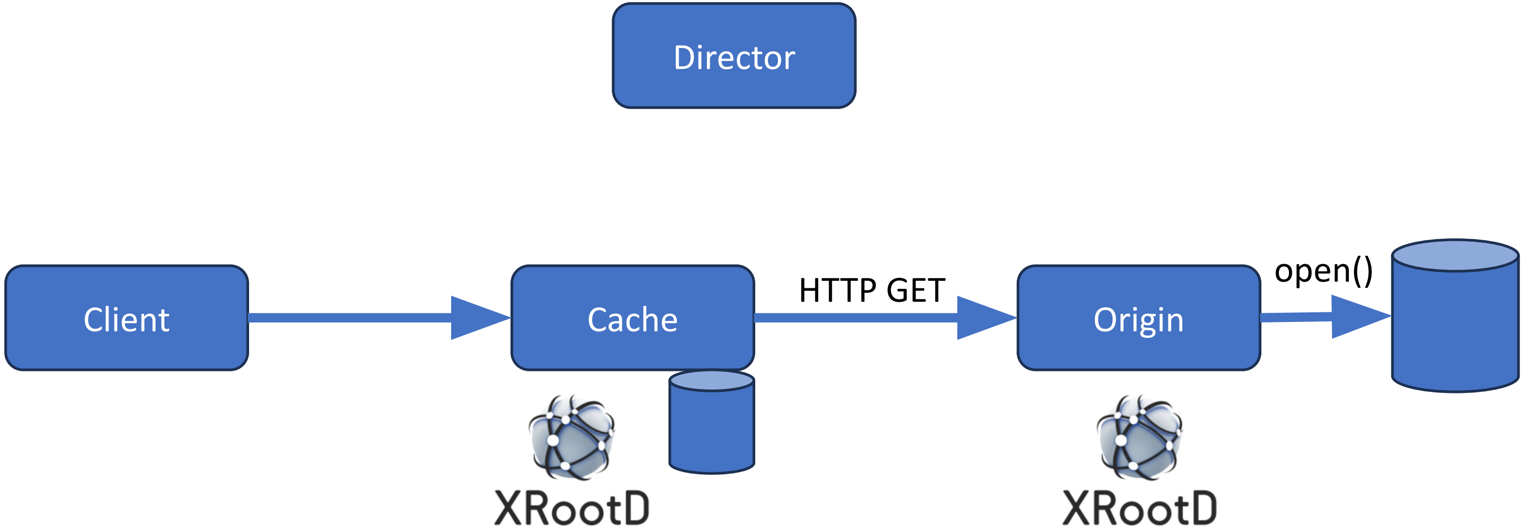
Pelican Data Flow



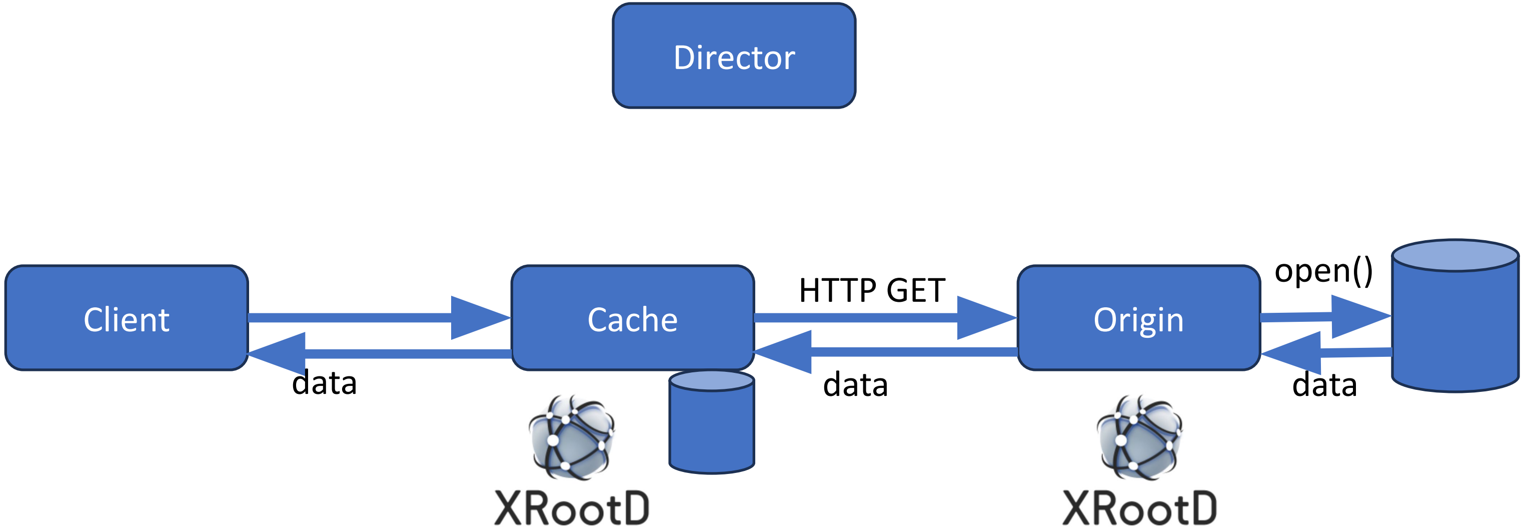
Pelican Data Flow



Pelican Data Flow



Pelican Data Flow



Note that the protocol between the client, cache, director, and origin is based on XRootD's HTTP plugin.

Example request from Client to Director

```
> GET /chtc/staging/jhiemstra/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/jhiemstra/testfile>; rel="duplicate"; pri=1;
depth=3, <https://stash-cache.osg.chtc.io:8443/chtc/staging/jhiemstra/testfile>; rel="duplicate"; pri=2;
depth=3,...
< location: https://osdf-uw-cache.svc.osg-htc.org:8443/chtc/staging/jhiemstra/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/jhiemstra/testfile>; rel="duplicate"; pri=1;
depth=3, <https://stash-cache.osg.chtc.io:8443/chtc/staging/jhiemstra/testfile>; rel="duplicate"; pri=2;
depth=3,...
< location: https://osdf-uw-cache.svc.osg-htc.org:8443/chtc/staging/jhiemstra/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
```

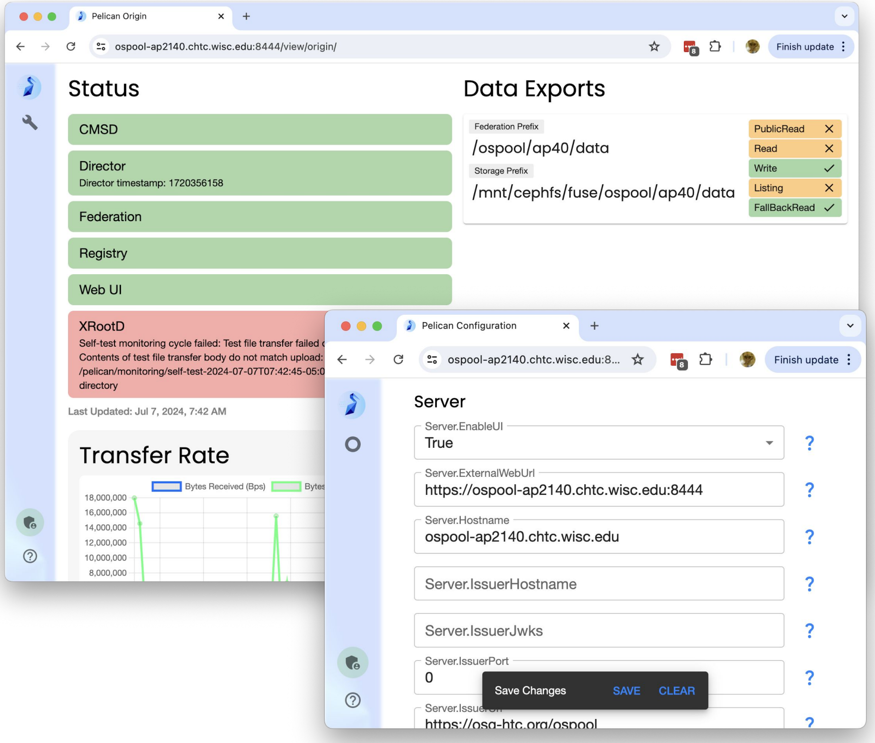
Example Director Response

```
< 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/jhiemstra/testfile>; rel="duplicate"; pri=1;
depth=3, <https://stash-cache.osg.chtc.io:8443/chtc/staging/jhiemstra/testfile>; rel="duplicate"; pri=2;
depth=3,...
< location: https://osdf-uw-cache.svc.osg-htc.org:8443/chtc/staging/jhiemstra/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).

"Batteries Included" Origin



Status

CMSD

Director
Director timestamp: 1720356158

Federation

Registry

Web UI

XRootD
Self-test monitoring cycle failed: Test file transfer failed
Contents of test file transfer body do not match upload:
/pelican/monitoring/self-test-2024-07-07T07:42:45-05:0
directory

Last Updated: Jul 7, 2024, 7:42 AM

Transfer Rate

Data Exports

Federation Prefix
/ospool/ap40/data
Storage Prefix
/mnt/cephfs/fuse/ospool/ap40/data

PublicRead	✗
Read	✗
Write	✓
Listing	✗
FallBackRead	✓

Server

Server.EnableUI: True

Server.ExternalWebUrl: https://ospool-ap2140.chtc.wisc.edu:8444

Server.Hostname: ospool-ap2140.chtc.wisc.edu

Server.IssuerHostname

Server.IssuerJwks

Server.IssuerPort: 0

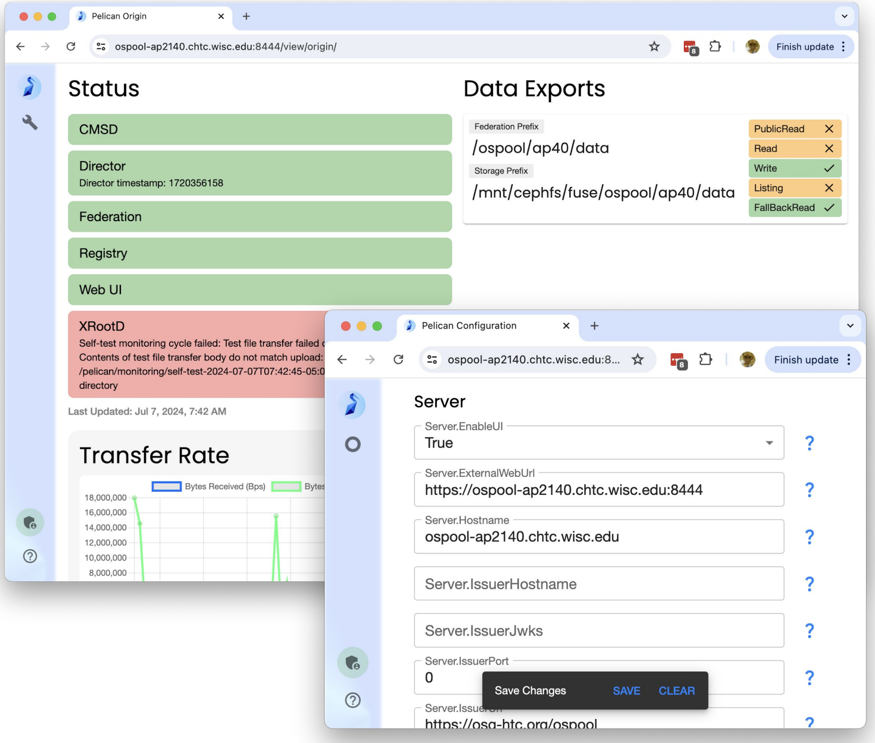
Server.IssuerUrl: https://osg-htc.org/ospool

Save Changes

We aim to simplify the art of running an Origin:

- New web UI for viewing, monitoring, and configuring the Origin.
- Origin runs built-in health checks
- Can use “connection reversing” so incoming firewall port / hostname / host certificate not needed.

"Batteries Included" Origin



Status

CMSD

Director
Director timestamp: 1720356158

Federation

Registry

Web UI

XRootD
Self-test monitoring cycle failed: Test file transfer failed c
Contents of test file transfer body do not match upload:
/pelican/monitoring/self-test-2024-07-07T07:42:45-05:0
directory

Last Updated: Jul 7, 2024, 7:42 AM

Transfer Rate

Data Exports

Federation Prefix
/ospool/ap40/data

Storage Prefix
/mnt/cephfs/fuse/ospool/ap40/data

PublicRead	✕
Read	✕
Write	✓
Listing	✕
FallBackRead	✓

Server

Server.EnableUI: True

Server.ExternalWebUrl: https://ospool-ap2140.chtc.wisc.edu:8444

Server.Hostname: ospool-ap2140.chtc.wisc.edu

Server.IssuerHostname

Server.IssuerJwks

Server.IssuerPort: 0

Server.IssuerU...: https://osg-htc.org/ospool

We aim to simplify the art of running an Origin:

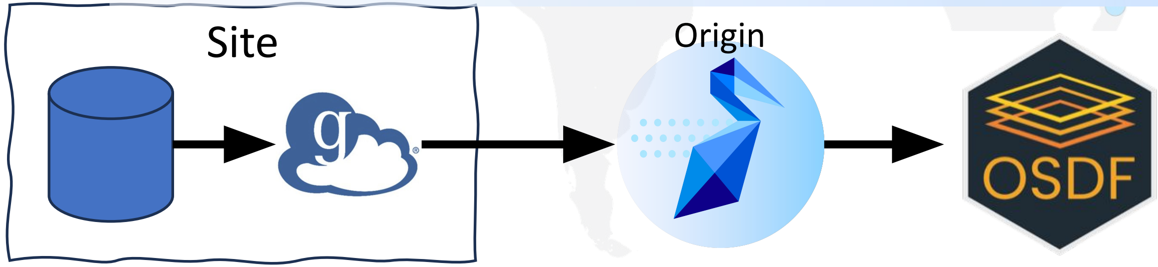
- New web UI for viewing, monitoring, and configuring the Origin.
- Origin runs built-in health checks
- Can use “connection reversing” so incoming firewall port / hostname / host certificate not needed.
- Our Goal – If you can set up a home router, you can run an Origin

Origin Backends

Beyond the traditional POSIX storage, we've added the following backends:

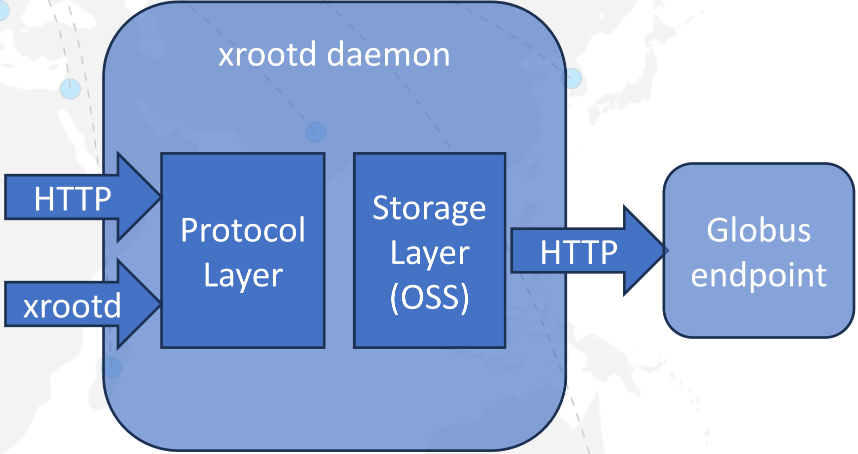
- **S3**: Works with any S3-compatible endpoint
- **Generic HTTP**: Integrate existing HTTP endpoint into a federation
- **Globus**: Users must authorize sharing a collection to the origin
- **XRootD**: Uses XRootD proxying module.

Note each of these backends can be used remotely – origin does not need to be present at the local site.



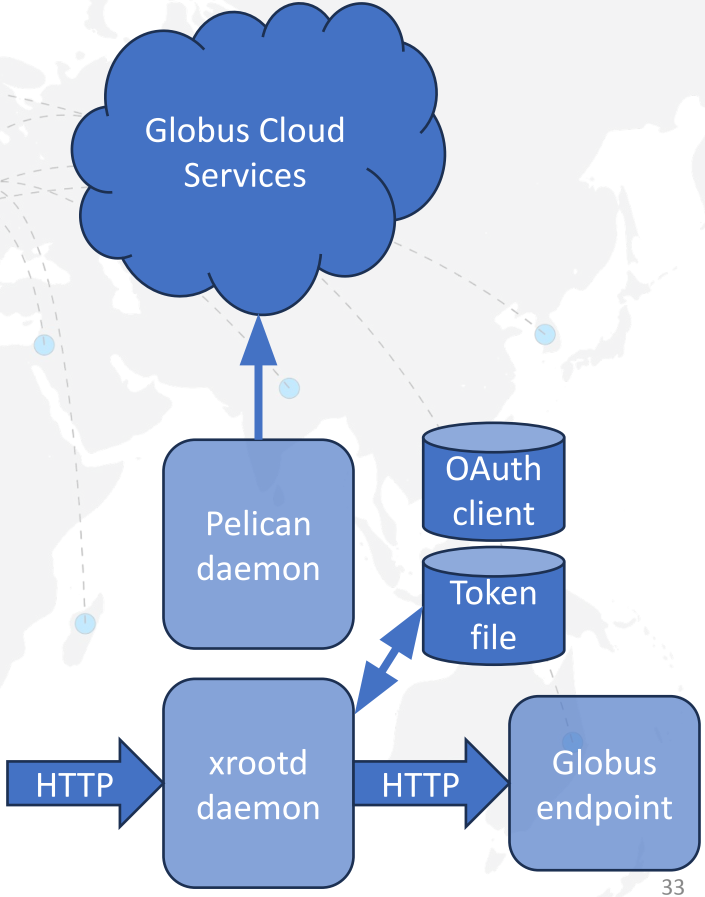
Globus Integration

- Globus's "bread and butter" is transferring files between two Globus endpoints.
  - Proprietary protocol (GridFTP-ish), no guarantee of version stability.
  - Historically, no such thing as "downloading" from a Globus endpoint – closed system.
- Recently, Globus added HTTP functionality and a corresponding API.
  - Can even do "curl" if you'd like!



Globus Integration

- To contact a Globus endpoint, you need a valid Globus token.
  - Globus uses traditional OAuth2 flows to hand tokens to web applications.
  - **Idea**: The Pelican daemon exports a web interface – use that as the OAuth2 client!
- We then use our underlying HTTP backend to communicate with Globus.
  - No Globus-specific code!



Globus – What works now, what doesn't

Currently Works:

- Read-only file operations.
- ‘Stat’ files

Future Work:

- Writes
- Directory listing (will need to Globus-specific code).

Pelican At Scale – A Look at the OSDF

Introducing the OSDF

The OSDF (Open Science Data Federation) is the flagship federation for delivering datasets from repositories to compute* in an effective, scalable manner.

* ‘Compute’ is viewed broadly; everything from a browser to a cluster.

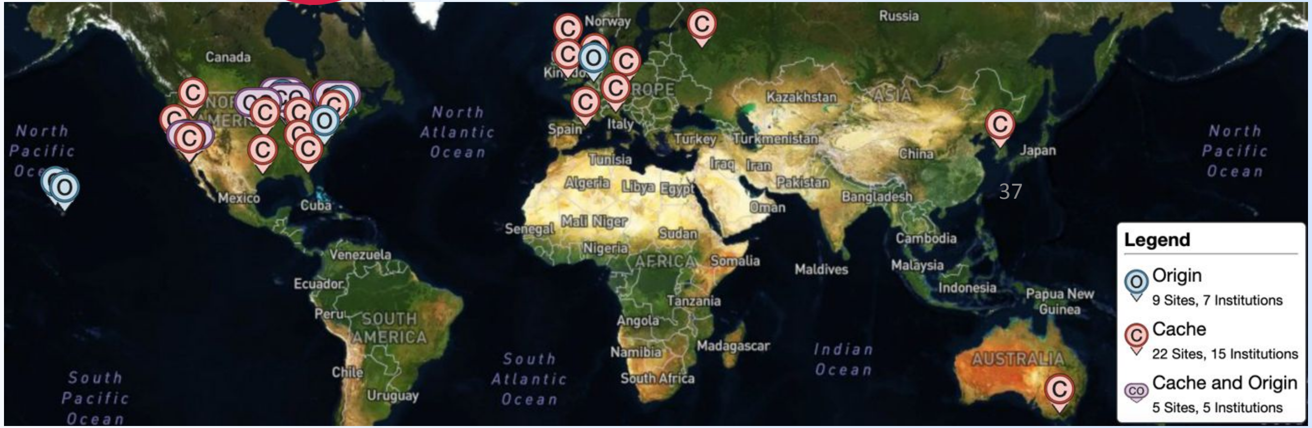
Connecting your repository

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,

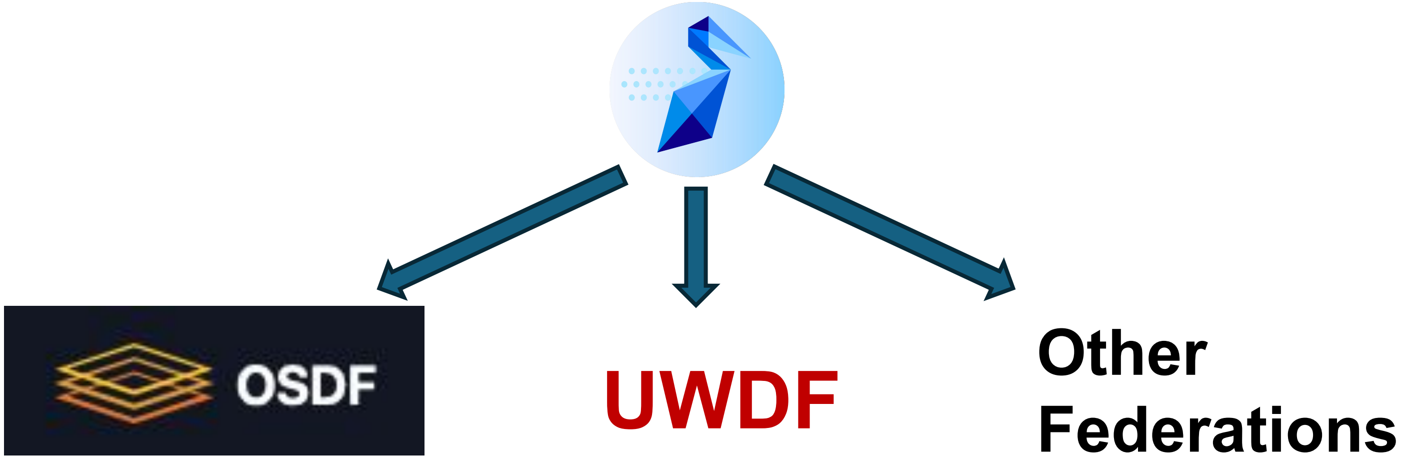


As part of the OSG Consortium’s Fabric of Services

Pelican versus the OSDF Explained

What's the difference between "OSDF" and "Pelican"?

- Pelican is a tool for creating ***data federations***
- The OSDF is one federation that's (mostly) underpinned by Pelican



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

Boise
Sunnyvale
Pasadena
San Diego
Kansas City
Cincinnati
New York
Newport News
Atlanta
Jacksonville
Houston

Converting OSDF to Pelican

- We are rolling out new services and protocols via a new software stack … onto the existing infrastructure!
  - E.g., a Pelican-based cache must be 100% compatible with old and new origins and clients.
  - No "flag day" option, cannot force client upgrades.
- Transition of services is >50% done.
  - Slower than anticipated. Familiar story: periodically pause to implement previously-unknown use cases, cleanup old messes.
  - Until we've 100% cutover, Pelican carries the burden of supporting both old and new clients.

Microsoft Copilot's interpretation of "changing the engine while the Pelican is flying"

Main Website

https://pelicanplatform.org

GH Repository

https://github.com/pelicanplatform/pelican

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.

Further Reading

Globus Collections and Authorization

Pelican will:

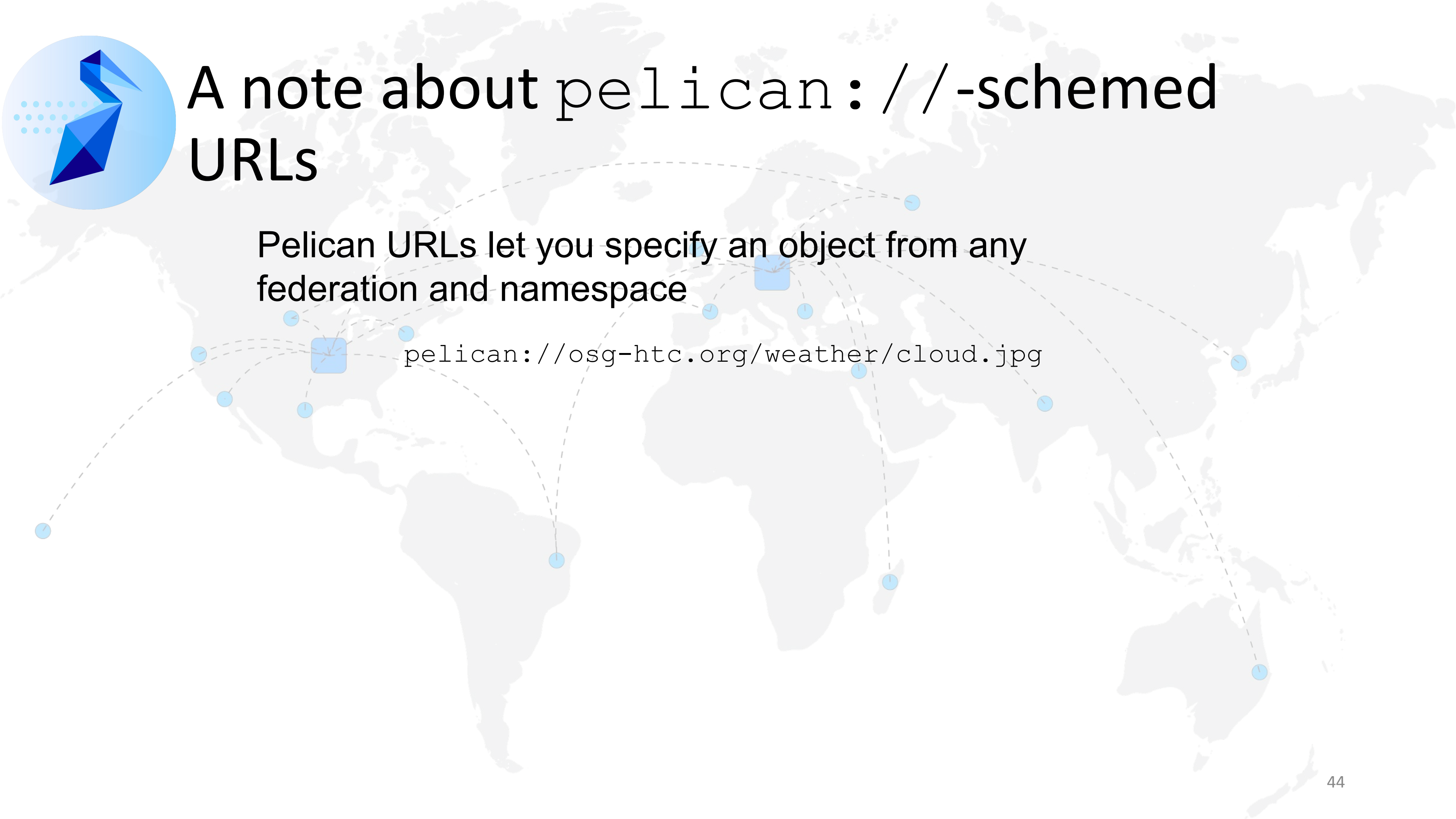
- (One-time) Request user to perform an OAuth2 flow with Globus, approving the origin’s access to the configured collection.
  - Pelican receives refresh and access token, writes it to disk.
- (Periodically) Pelican runs refresh flow to get a new access token, writes it to disk.
- (Per-request) HTTP backend loads globus token from disk, adds it to the `Authorization` header of the HTTP request.

Globus and XRootD auth'z are decoupled

Pelican configuration YAML snippet:

```
Origin:
  - GlobusCollectionID: "abc-123-some-key"
  - GlobusCollectionName: "Human-Friendly-Name"
  - GlobusClientIDFile: "/etc/pelican/glbs.client"
  - GlobusClientSecretFile: "/etc/pelican/glbs.secret"
```

A note about `pelican://`-schemed URLs



Pelican URLs let you specify an object from any federation and namespace

```
pelican://osg-htc.org/weather/cloud.jpg
```

A note about `pelican://`-schemed URLs

Pelican URLs let you specify an object from any federation and namespace

```
pelican://osg-htc.org/weather/cloud.jpg
```

🟦 Defines a metadata lookup protocol

🟧 The federation’s hostname/root

🟩 The desired object name

Note that we also support "osdf://" and "stash://" schemes. The above is equivalent to:

```
osdf:///weather/cloud.jpg
```

Pelican/OSDF URLs Give Us Query Parameters

Pelican URLs let us interact with objects – they also let us choose *how* we interact with those objects.

- `?directreads` - skip the caching mechanism, get data straight through the Origin
- `?recursive` - download collections/directories recursively
- `?pack` - upload/download using compression schemes on the fly

```
pack = < tar, tar.gz, tar.xz, zip >
```

E.g. `pelican://osg-htc.org/weather/cloud.jpg`**`?directread`**