# HDFCloud Workshop

HDF5 in the Cloud

The HDF Group

John Readey
The HDF Group
*jreadey@hdfgroup.org*

# My Background

The HDF Group

Sr. Architect at The HDF Group

Started in 2014

Have been exploring remote interfaces to HDF

Previously: Dev Manager at Amazon/AWS

amazon web services

More previously: Used HDF5 while a developer at Intel

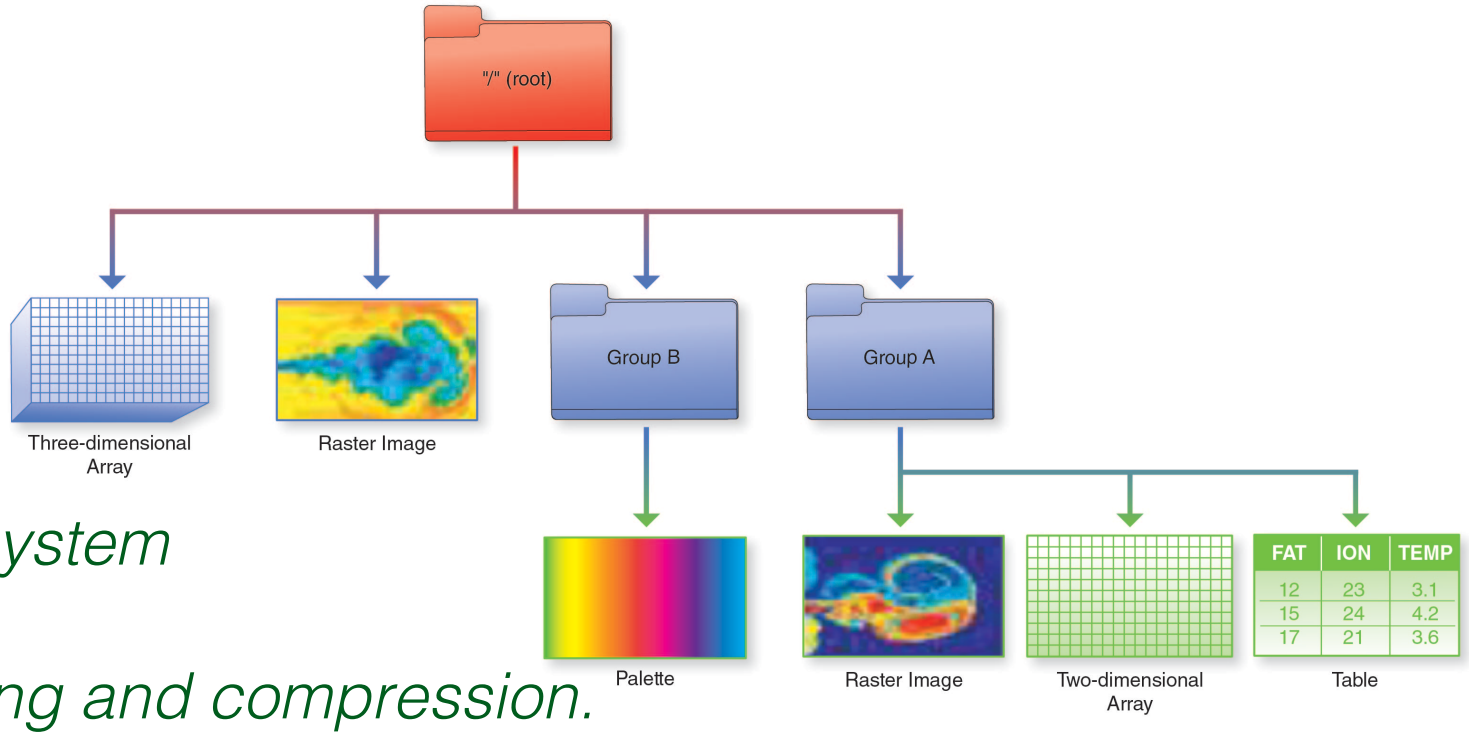
intel inside

# What is HDF5?



## Depends on your point of view:

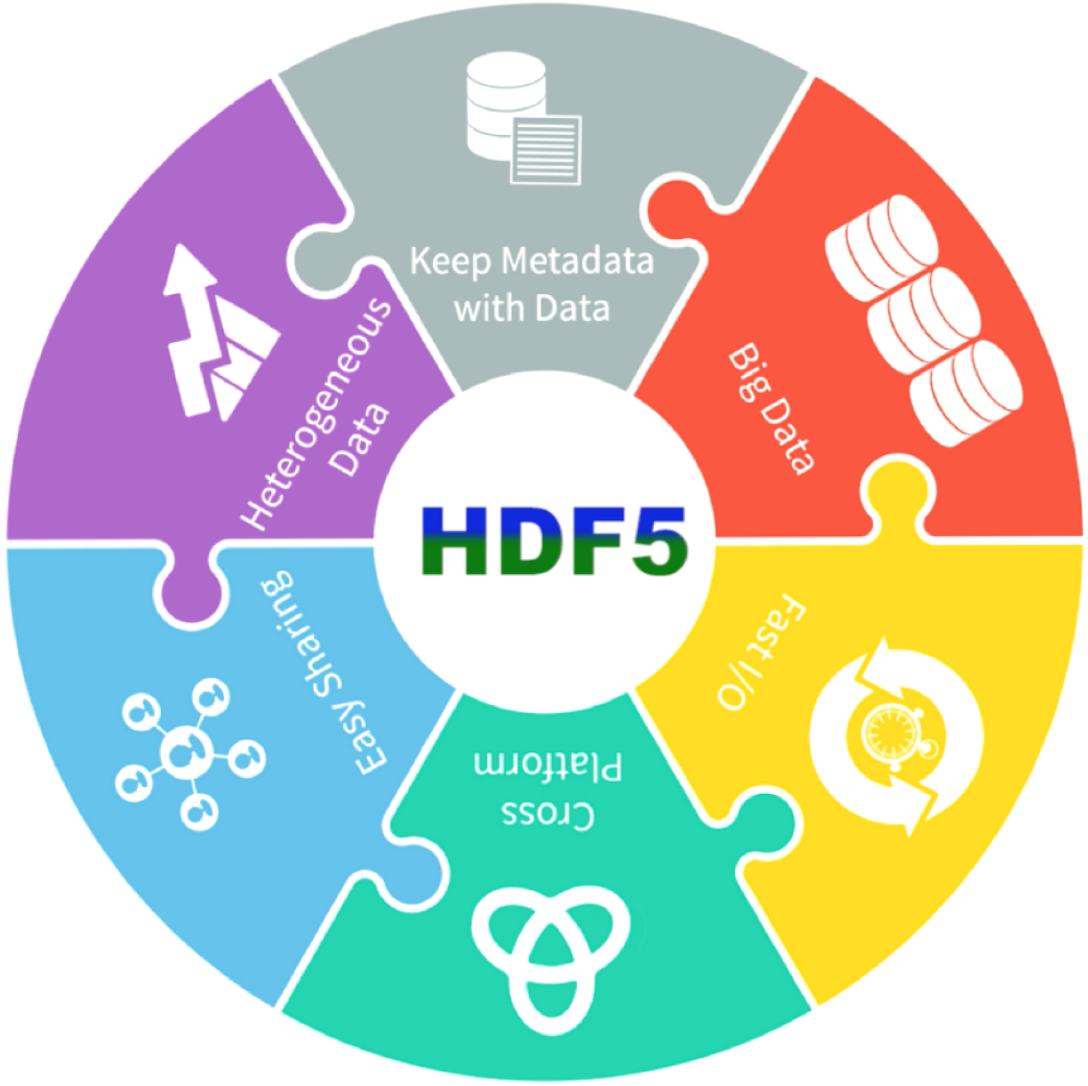
- **a C-API**
- **a File Format**
- **a data model**




*Think of HDF5 as a file system within a file.*
*Store arrays with chunking and compression.*
*Add NumPy style data selection.*

# Why is this concept so different + useful?






- **Native support for multidimensional data**
- **Data and metadata in one place => streamlines data lifecycle & pipelines**
- Portable, no vendor lock-in
- Maintains logical view while adapting to storage context
- In-memory, over-the-wire, on-disk, parallel FS, object store
- Pluggable filter pipeline for compression, checksum, encryption, etc.
- High-performance I/O
- Large ecosystem (700+ Github projects)

# Why HDF in the Cloud

The HDF Group

- It can provide a cost-effective infrastructure
  - Pay for what you use vs pay for what you may need
  - Lower overhead: no hard ware setup/network configuration, etc.
- Potentially can benefit from cloud-based technologies:
  - Elastic compute – scale compute resources dynamically
  - Object based storage – low cost/built in redundancy
- Community platform (potentially)
  - Enables interested users to bring their applications to the data
  - Share data among many users

# Cost Factors



- **Most public clouds bill per usage**
- **For HDF in the cloud, there are three big cost drivers:**
  - **Storage – What storage system will be used? (see next slide)**
  - **Compute – Elastic compute on demand better than fixed cost**
    - Scale compute to usage not size of data
  - **Egress charges**
    - Ingress is free but getting data out will cost you ($0.09/GB)
    - Enabling users to get just the data they need will tend to lower egress charges

# Storage Costs


How much will it costs to store 1PB for one year on AWS?
Answer depends on the technology and tradeoffs you are willing to accept…

| Technology | What it is | Cost for 1PB/1yr | Fine Print |
|---|---|---|---|
| Glacier | Offline (tape) Storage | $125K | - 4 hour latency for first read<br>- Additional costs for restore |
| S3 Infrequent Access | Nearline Object Storage | $157K | - $0.01/GB data retrieval charge<br>- $10K to read entire PB! |
| S3 | Online Object Storage | $289K | - Request pricing $0.01 per 10K req<br>- Transfer out charge $0.01/GB |
| EBS | Attachable Disk Storage | $629K | - Extra charges for guaranteed IOPS<br>- Need backups |
| EFS | Shared Network (NFS) | $3,774K | - Not all NFSv4.1 features supported<br>- E.g. File Locking |
| DynamoDB | NoSQL Database | $3,145K | - Extra charge for IOPS |

# Introducing Highly Scalable Data Service (HSDS)



- RESTful interface to HDF5 using object storage
- Storage using AWS S3
  - Built in redundancy
  - Cost effective
  - Scalable throughput
- Runs as a cluster of Docker containers
  - Elastically scale compute with usage
- Feature compatible with HDF5 library
- Implemented in Python using asyncio
  - Task oriented parallelism

# HSDS Features



- Clients can interact with service using REST API
- SDKs provide language specific interface (e.g. h5pyd for Python)
- Can read/write just the data they need (as opposed to transferring entire files)
- No limit to the amount of data that can be stored by the service
- Multiple clients can read/write to same data source
- Scalable performance:
  - Can cache recently accessed data in RAM
  - Can parallelize requests across multiple nodes
  - More nodes -> better performance

# What makes it RESTful?

- Client-server model
- Stateless – (no client context stored on server)
- Cacheable – clients can cache responses
- Resources identified by URIs (datasets, groups, attributes, etc)
- Standard HTTP methods and behaviors:

| Method | Safe | Idempotent | Description |
|---|---|---|---|
| GET | Y | Y | Get a description of a resource |
| POST | N | N | Create a new resource |
| PUT | N | Y | Create a new named resource |
| DELETE | N | Y | Delete a resource |

# Example POST Request – Create Dataset


**POST** /datasets **HTTP/**1.1
Content-Length**:** 39
User-Agent**:** python-requests/2.3.0 CPython/2.7.8 Darwin/14.0.0
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
host**:** newdset.datasettest.test.hdfgroup.org
Accept**:** */*
Accept-Encoding**:** gzip, deflate
{ "shape": 10, "type": "H5T_IEEE_F32LE" }

**HTTP/**1.1 201 **Created**
Date**:** Thu, 29 Jan 2017 06:14:02 GMT
Content-Length**:** 651
Content-Type**:** application/json
Server**:** aiohttp/3.2.2
{ "id": "0568d8c5-a77e-11e4-9f7a-3c15c2da029e", "attributeCount": 0, "created":
"2017-01-29T06:14:02Z", "lastModified": "2017-01-29T06:14:02Z", … ] }

# Object Storage Challenges for HDF

The HDF Group

- Not POSIX!
- High latency (>0.1s) per request
- Not write/read consistent
- High throughput needs some tricks
  - (use many async requests)
- Request charges can add up (public cloud)

*For HDF5, using the HDF5 library directly on an object storage system is a non-starter. Will need an alternative solution…*

# HSDS S3 Schema



How to store HDF5 content in S3?

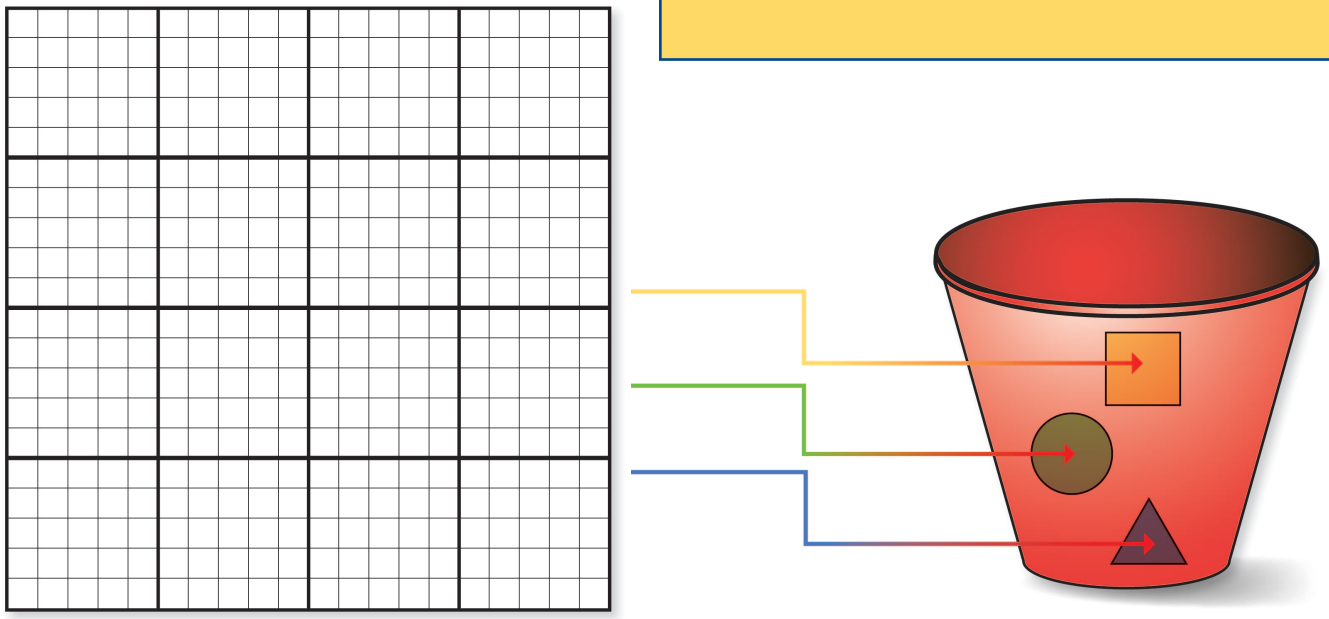
- Limit maximum storage object size
- Support parallelism for read/write
- Only data that is modified needs to be updated
- (Potentially) Multiple clients can be reading/updating the same “file”

**Big Idea: Map individual HDF5 objects (datasets, groups, chunks) as Object Storage Objects**

Each chunk (heavy outlines) get persisted as a separate object

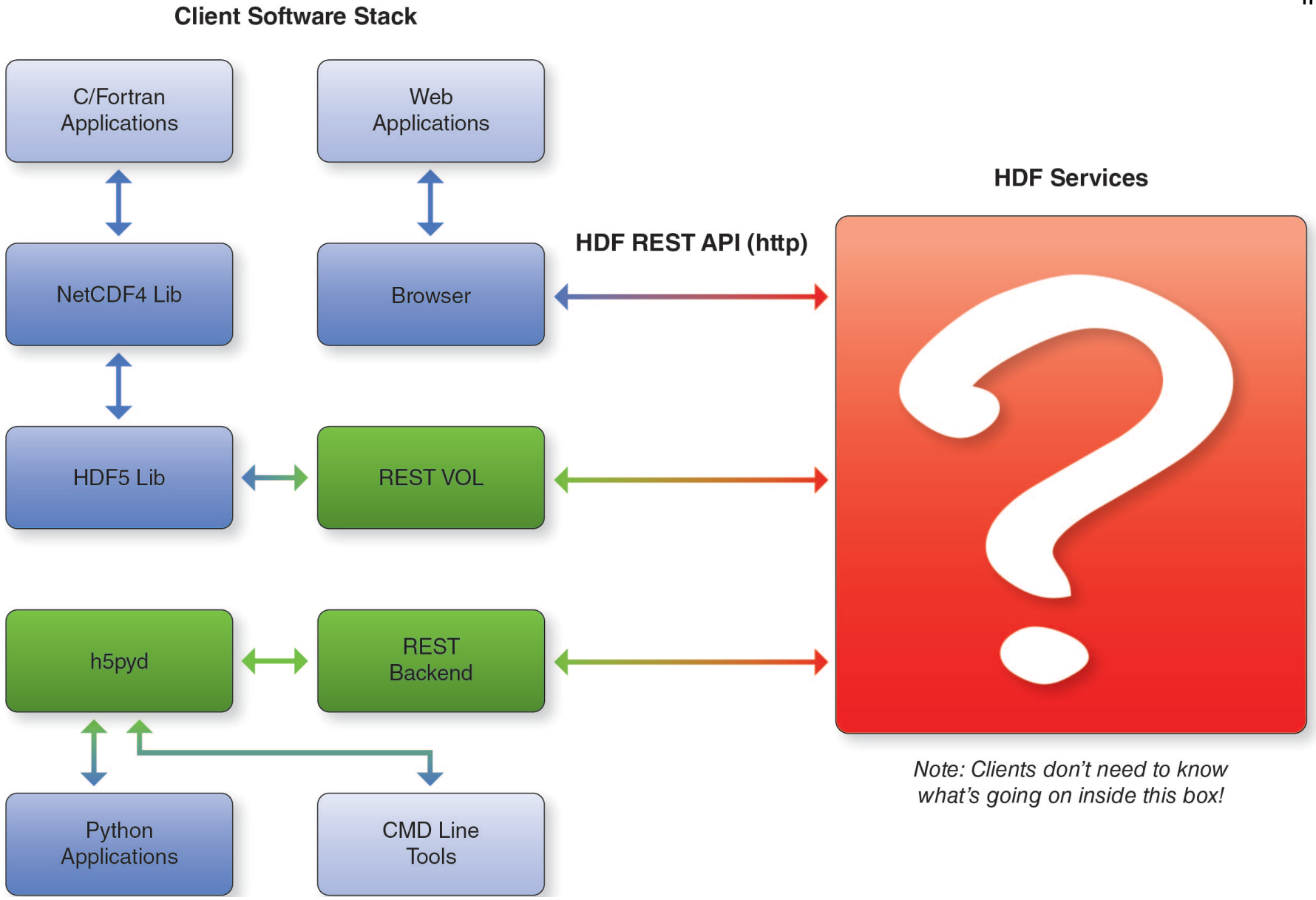
*Legend:*

- *Dataset is partitioned into chunks*
- *Each chunk stored as an S3 object*
- *Dataset meta data (type, shape, attributes, etc.) stored in a separate object (as JSON text)*



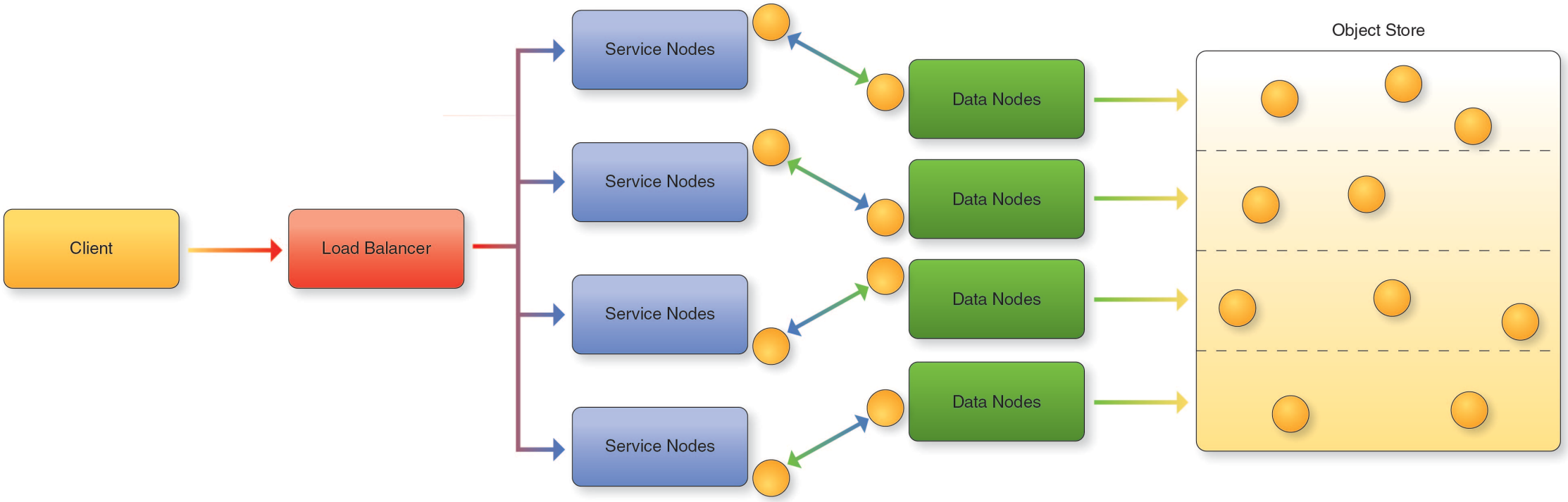
# Client/Server Architecture





# Architecture for HSDS






Legend:

- Client: Any user of the service
- Load balancer – distributes requests to Service nodes
- Service Nodes – processes requests from clients (with help from Data Nodes)
- Data Nodes – responsible for partition of Object Store
- Object Store: Base storage service (e.g. AWS S3)

# Implementing HSDS with asyncio



- **HSDS relies heavily on Python's new asyncio module**
  - **Concurrency based on *tasks* (rather than say multithreading or multiprocessing)**
  - **Task switching occurs when process would otherwise wait on I/O**

```
async def my_func():
    a_regular_function_call()
    await a_blocking_call()
```

- Control will switch to another task when await is encountered
- Result is the app can do other useful work vs. blocking
- Supporting 1000's of concurrent tasks within a process is quite feasible

# Parallelizing data access with asyncio



- **SN node invoking parallel requests on DN nodes**

```
tasks = []
for chunk_id in my_chunk_list:
    task = asyncio.ensure_future(read_chunk_query(chunk_id))
    tasks.append(task)
await asyncio.gather(*tasks, loop=loop)
```

- Read_chunk_query makes a http request to a specific DN node
- Set of DN nodes can be reading from S3, decompression and selecting requested data in parallel
- Asyncio.gather waits for all tasks to complete before continuing
- Meanwhile, new requests can be processed by SN node

# Python and Docker



- Docker makes developing clustered applications sooo much easier
  - Can run dozens of containers on a moderate laptop
  - Containers communicate with each other just like on a physical network
  - Use docker stats to check up cpu, net i/o, disk i/o usage per container
  - Can try out different constraints for amount of memory, disk per container
  - Same code “just works” on an actual cluster
  - ”scale up” by launching more containers on production hardware
  - AWS ECS enables running containers in a machine agnostic way
- Using docker does require a reversion to the edit/build/run paradigm
  - The build step is now the creation of the docker image
  - Run is launching the container(s)

# Python package MVPs



- numpy – python arrays
  - Used heavily in server and client stacks
  - Great performance for common array operations
  - Simplifies much of the logic needed for hyperslab selection
- aiohttp – async http client/server
  - Use of asyncio requires async enabled packages
  - Aiohttp is used in HSDS as both web server and client
- Aiobotocore – async aws s3 client
  - Enables async read/write to S3
- H5py – template for h5pyd package

# H5pyd – Python client for HDF Server



- H5py is a popular Python package that provide a Pythonic interface to the HDF5 library
- H5pyd (for h5py distributed) provides a h5py compatible h5py for accessing the server
- Pure Python – uses requests package to make http calls to server
- Compatible with h5serv (the reference implementation of the HDF REST API)
- Include several extensions to h5py:
  - List content in folders
  - Get/Set ACLs (access control list)
  - Pytables-like query interface

# HDF REST VOL



- The HDF5 VOL architecture is a plugin layer for HDF5
- User API stays the same, but different backends can be implemented
- REST VOL substitutes REST API requests for file i/o actions
- C/Fortran applications should be able to run as is
- Still in development – Beta expected this year

# HSDS CLI (Command Line Interface)



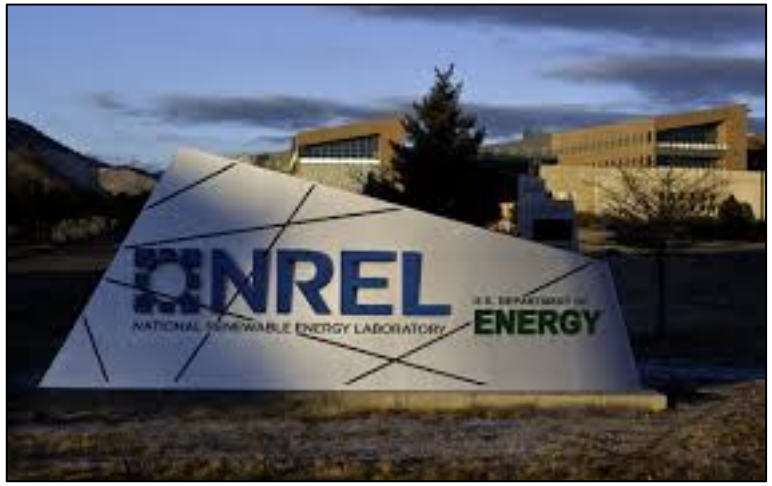
- Accessing HDF via a service means can't utilize usual shell commands: ls, rm, chmod, etc.
- Command line tools are a set of simple apps to use instead:
  - hsinfo: display server version, connect info
  - hsls: list content of folder or file
  - hstouch: create folder or file
  - hsdel: delete a file
  - hsload: upload an HDF5 file
  - hsget: download content from server to an HDF5 file
  - hsacl: create/list/update ACLs (Access Control Lists)
- Implemented in Python & uses h5pyd

# Future Work



- Work planned for the next year
  - Compression
  - Variable length datatypes
  - NetCDF support
  - Auto Scaling
  - Scalability and performance testing

# Demo Time!




NREL (National Renewable Energy Laboratory) is using HSDS to make 7TB of wind simulation data accessible to the public.

Datasets are three-dimensional covering the continental US:

- Time (one slice/hour)
- Lon (~2k resolution)
- Lat (~2k resolution)

Initial data covers one year (8760 slices), but will be soon be extended to 5 years (35 TBs).

Rather than downloading TB's of files, interested users can now use the HSDS client libraries to explore the datasets.

# To Find out More:



- H5serv: https://github.com/HDFGroup/h5serv
- Documentation: http://h5serv.readthedocs.io/
- H5pyd: https://github.com/HDFGroup/h5pyd
- RESTful HDF5 White Paper: https://www.hdfgroup.org/pubs/papers/RESTful_HDF5.pdf
- Blog articles:
  - https://hdfgroup.org/wp/2015/04/hdf5-for-the-web-hdf-server/
  - https://hdfgroup.org/wp/2015/12/serve-protect-web-security-hdf5/
  - https://www.hdfgroup.org/2017/04/the-gfed-analysis-tool-an-hdf-server-implementation/

# HDF5 Community Support



- Documentation - https://support.hdfgroup.org/documentation/
  - Tutorials, FAQs, examples
- HDF-Forum – mailing list and archive
  - Great for specific questions
- Helpdesk Email – help@hdfgroup.org
  - Issues with software and documentation

https://support.hdfgroup.org/services/community_support.html

# Questions? Comments?

The HDF Group

## www.hdfgroup.org

Dave Pearah
CEO
David.Pearah@hdfgroup.org

Dax Rodriguez
Director of Commercial Services and Solutions
Dax.Rodriguez@hdfgroup.org