

# A Python Roadmap

## for Accessing and Interfacing with Big Climate Data in Cloud Storage Providers


Ryan Paul Lafler

Samuel S.P. Shen

Mitch Goldberg



AMS 2025

105TH ANNUAL MEETING

12–16 JANUARY

NEW ORLEANS & ONLINE

AMERICAN METEOROLOGICAL SOCIETY



105TH ANNUAL MEETING
12–16 JANUARY
NEW ORLEANS & ONLINE
AMERICAN METEOROLOGICAL SOCIETY
AMS 2025

# 1. Optimized Cloud Formats for Big Climate Datasets

Copyright © 2024-2025 by Ryan Paul Lafler.

# Typical Data Structures for Big Gridded Climate Data

- Gridded climate data comes in diverse file types adhering to different data structures
- Fundamental data structures include:
  - **Flat files**
    - Cross-language versatility; does not support metadata, grid consistency standards, compression schemes, projections, & efficient data packing
  - **Multi-Dimensional Data Cubes**
    - Explicitly represent relationships between dimensions within climate datasets → supports extensive metadata, compression schemes, & slicing arrays across dimensions
  - **Hierarchical Array Sequence**
    - Climate / meteorological variable arrays are stored as independent records sequentially with support for extensive metadata, different grids, & compression

# Multi-Dimensional Data Cubes

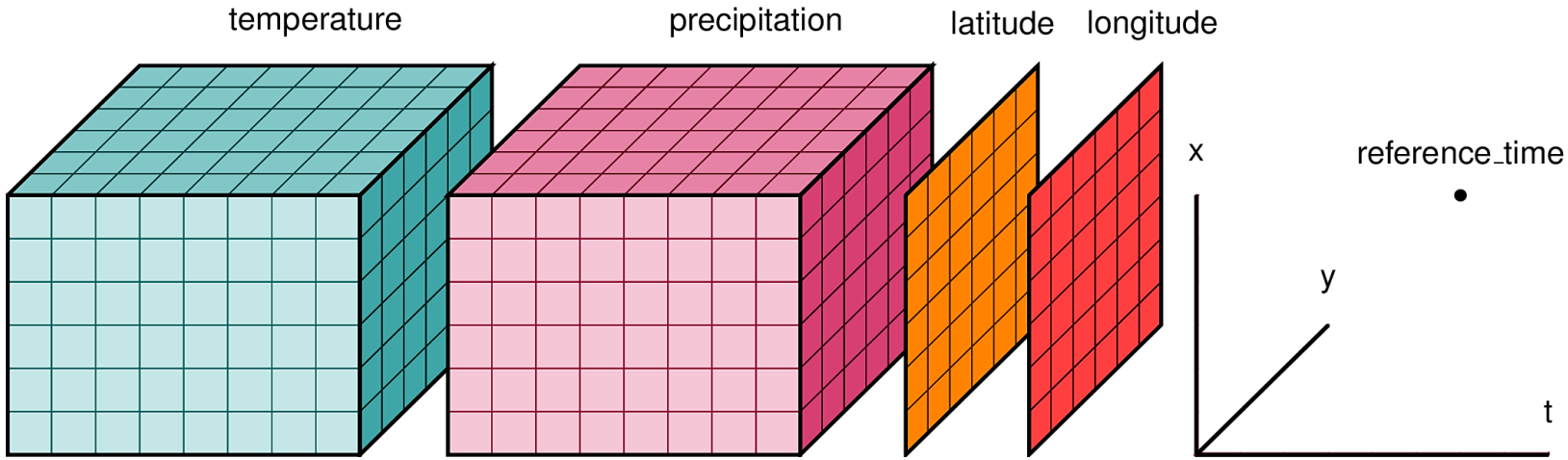



*Image Source: Xarray Documentation*

- Gridded arrays organized with relationships explicitly mapped between dimensions
  - Surface Temperature? **(Longitude, Latitude, Time)**
  - Air Temperature? **(Longitude, Latitude, Time, Pressure Level)**
- Allows for quick slicing, querying, & generating time series for climate variables
- Typical file structures include **NetCDF**, **HDF5**, & **Zarr archives**
- Good for large datasets requiring compression, data chunking, & slicing across dimension(s)

# Hierarchical Array Sequence




**2024-04-18**

- Climate / meteorological variables stored as independent records as a sequence of separate data arrays
- Each record contains its own metadata, grid resolution, & data organization structure
- Every record includes a header that allows programmer to retrieve relevant climate attribute without having to read in every record
  - Isolated records allow user to only read & retrieve variables they need
- Excellent image compression schemes for high-resolution spatial & temporal datasets → efficient binary packing
- Common file formats include **GRIB / GRIB2** filetypes



105TH ANNUAL MEETING
12–16 JANUARY
NEW ORLEANS & ONLINE
AMERICAN METEOROLOGICAL SOCIETY
AMS 2025

# 2. Connecting to CMIP6 in Google Cloud Storage with Python

Copyright © 2024-2025 by Ryan Paul Lafler.

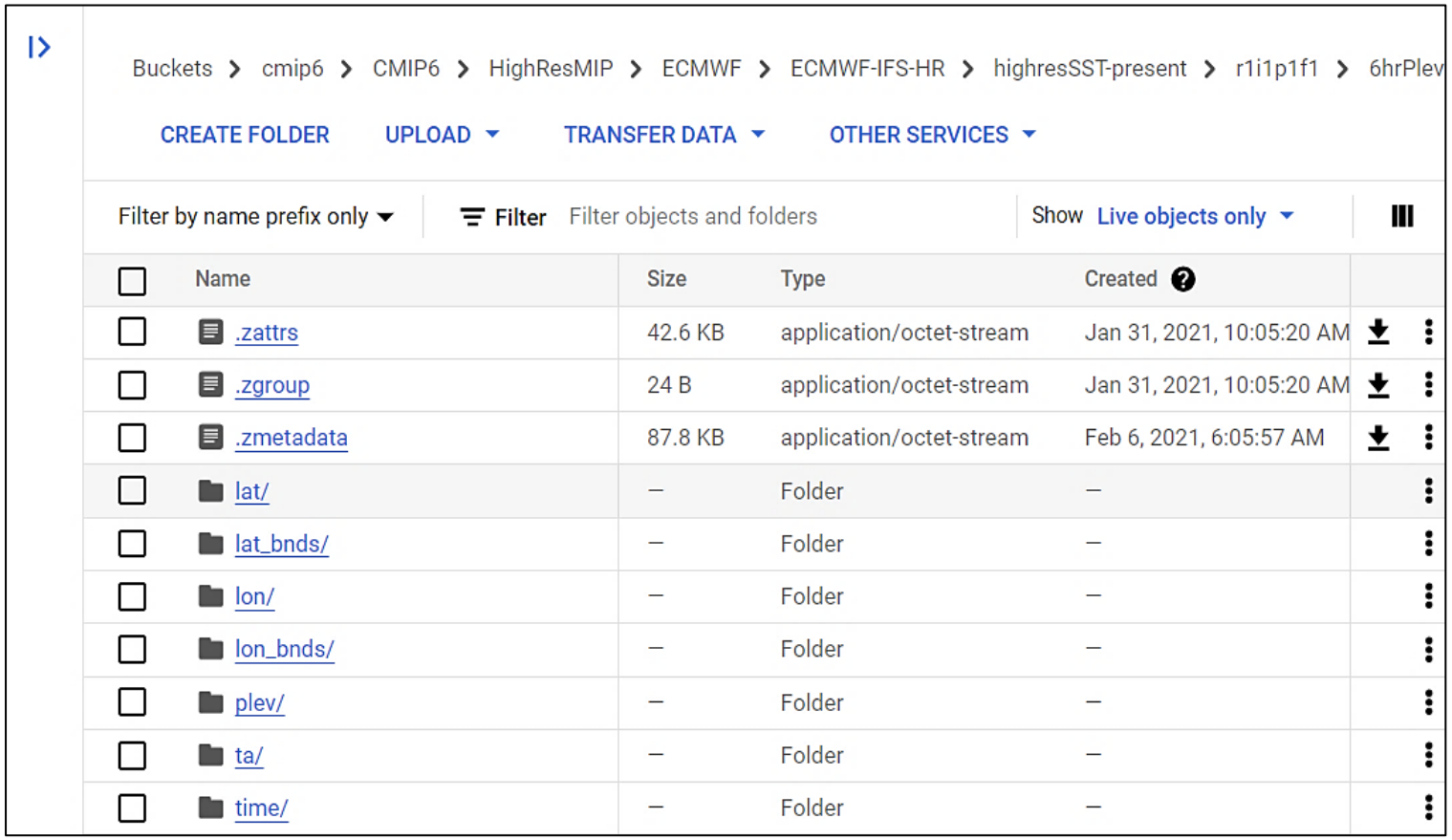
# What is Object Storage? How are Data Stored in the Cloud?

- Python contains a growing number of community-supported libraries for handling data structures in cloud object storage
- Cloud Object Storage services allow developers to create flexible file system hierarchies holding *petabytes* of optimized data structures entirely in the cloud
  - **Google Cloud Storage (GCS)**
  - **Amazon S3**
  - **Microsoft Azure**
- Cloud Object Storage writes files into ***BLOB****s* (**B**inary **L**arge **OB**ject) stored in folder hierarchies
  - Maintainers should employ proper file naming conventions to navigate directories
- Public climate data like CMIP6 are stored as BLOBs inside of *buckets*
- Python APIs exist to access cloud storage providers entirely *within* Python

# Python Libraries for Working with Cloud Object Storage

- Python community and official developers release open-source API tools for cloud storage
  - **GCSFS**: Google Cloud Storage File System API that integrates a *Pythonic* interface for streaming, writing data chunks, & reading data files directly from GCS
  - **Boto3**: Low-level Software Development Kit (SDK) for managing data in Amazon S3 that integrates a *Pythonic* interface for streaming, writing, & reading files directly from S3
  - **S3FS**: Simplified Amazon S3 File System API that allows manipulation & retrieval of data blobs in S3 buckets using folder pathways → acts as a wrapper around Boto3
  - **FSSPEC**: Filesystem Spec that supports wide-versatility for different cloud & local file storage systems → community-supported and generalized for any cloud provider
  - **Azure-Storage-Blob**: Microsoft Azure Cloud Storage library for streaming, downloading, writing, & navigating buckets & blobs for Azure-specific storage

# Visualizing CMIP6 Zarr Archive in Google Cloud Storage (GCS)




- CMIP6 high-resolution ECMWF historical 6-hour air temperature
- Gridded arrays are organized into cloud-friendly Zarr archive
  - Data cube allows efficient multi-dimensional queries
  - Supports chunking & distributed processing
- Each dimension receives its own folder & metadata

# Lazily-Loading CMIP6 Zarr Archive into Python with GCSFS

xarray.DataArray 'ta' (**time**: 94964, **plev**: 7, **lat**: 361, **lon**: 720)

| | Array | Chunk |
|---|---|---|
| **Bytes** | 643.66 GiB | 249.86 MiB |
| **Shape** | (94964, 7, 361, 720) | (36, 7, 361, 720) |
| **Dask graph** | 2638 chunks in 2 graph layers | |
| **Data type** | float32 numpy.ndarray | |







▾ Coordinates:

| | | | |
|---|---|---|---|
| **lat** | (lat) | float64 | -90.0 -89.5 -89.0 ... 89.5 90.0 |
| **lon** | (lon) | float64 | 0.0 0.5 1.0 ... 358.5 359.0 359.5 |
| **plev** | (plev) | float64 | 9.25e+04 8.5e+04 ... 2.5e+04 5e+03 |
| **time** | (time) | datetime64[ns] | 1950-01-01 ... 2014-12-31T18:00:00 |

- Air temperature contains 643.66 GB worth of data (Jan. 1, 1950 – Dec. 31, 2014)
  - Variable partitioned into manageable chunks averaging 250 MB in size
  - Each chunk contains only 36 time slices, all 7 pressure levels, & lon / lat dimensions preserved
    - Entire archive partitioned into 2,638 total data chunks
- Data organized into multi-dimensional cube as **(time, pressure level, lat, lon)**
- Chunks allow efficient querying, filtering, & processing of large records in-parallel

# Connecting to a CMIP6 Dataset Zarr Archive in GCS

```
import gcsfs
# Google Cloud Storage (GCS) Bucket Name
bucket = “cmip6“

# GCS File System Client
client = gcsfs.GCSFileSystem(
    project = bucket
)
# GCS internal connection to ECMWF High-Resolution simulations folder
main_path = f"gs://{bucket}/CMIP6/HighResMIP/ECMWF/ECMWF-IFS-HR"
```

Import GCSFS library. Specify name of GCS bucket → create an instance of GCSFS client in Python runtime and define main path to reach high-resolution ECMWF simulations.

# Connecting to a CMIP6 Dataset Zarr Archive in GCS



```
# Simulation using fully-coupled model for ocean & atmosphere
experiment = "highresSST-present"

# Realization initialization conditions
simulation = "r1i1p1f1"

# Temporal resolution
time = “6hrPlevPt“

# Climate variable
var = “ta"
```

Parameters that modify folder pathway leading to desired Zarr archive → use simulations provided by fully-coupled model at every 6-hours for air temp at all pressure levels.

# Connecting to a CMIP6 Dataset Zarr Archive in GCS



```
# Type of Grid
gr = “gr"

# Simulation dataset version
simulation = “v20170915"

# Concatenate folder pathway to connect with desired Zarr Archive:
gcs_path=f"{main_path}/{experiment}/{simulation}/{time}/{var}/{gr}/{version}"
print(gcs_path)
```

```
gs:// cmip6 / CMIP6 / HighResMIP / ECMWF / ECMWF-IFS-HR / highresSST-present / r1i1p1f1 / 6hrPlevPt / ta / gr / v20170915
```

# Lazily-Load and View the Connected Zarr Archive



```
# Connect GCS Client to Zarr Archive
gcs_client = client.get_mapper(
    gcs_path
)

# Lazily load entire dataset into Python
ds = xr.open_zarr(
    gcs_client ,
    consolidated = True ,
    chunks = "auto"
)
ds  # View lazily-loaded dataset
```

# Lazily-Loading CMIP6 Zarr Archive into Python with GCSFS

xarray.DataArray 'ta' (**time**: 94964, **plev**: 7, **lat**: 361, **lon**: 720)

| | Array | Chunk |
|---|---|---|
| **Bytes** | 643.66 GiB | 249.86 MiB |
| **Shape** | (94964, 7, 361, 720) | (36, 7, 361, 720) |
| **Dask graph** | 2638 chunks in 2 graph layers | |
| **Data type** | float32 numpy.ndarray | |




▾ Coordinates:

| | | | |
|---|---|---|---|
| **lat** | (lat) | float64 | -90.0 -89.5 -89.0 ... 89.5 90.0 |
| **lon** | (lon) | float64 | 0.0 0.5 1.0 ... 358.5 359.0 359.5 |
| **plev** | (plev) | float64 | 9.25e+04 8.5e+04 ... 2.5e+04 5e+03 |
| **time** | (time) | datetime64[ns] | 1950-01-01 ... 2014-12-31T18:00:00 |



# Thank You for Attending!

## Questions, Comments, and/or Further Discussion?


**Ryan Paul Lafler**
rplafler@premier-analytics.com

**Samuel S.P. Shen**
sshen@sdsu.edu

**Mitch Goldberg**
mgoldberg1@ccny.cuny.edu



AMS 2025
105TH ANNUAL MEETING
12–16 JANUARY
NEW ORLEANS & ONLINE
AMERICAN METEOROLOGICAL SOCIETY