A Cloud-Native Python Framework

Scalable Access and Analysis of Environmental Data for Forecasting and Decision Systems

Ryan Paul Lafler

Founder, CEO, and Lead Consultant

Premier Analytics Consulting, LLC

rplafler@premier-analytics.com

AMS American Meteorological Society

www.Premier-Analytics.com

Premier Analytics Consulting, LLC

Ryan Paul Lafler | Presenter Bio

Ryan Paul Lafler is the Founder, CEO, and Lead Consultant of **Premier Analytics Consulting, LLC**, a California-certified small business based in San Diego specializing in localized AI and machine learning systems, data engineering, enterprise GIS, clinical and statistical analysis, and full-stack analytics platform development. Through project-based assignments and consulting roles, Ryan provides advisory, development, and training support for custom AI/ML systems, full-stack platforms, data engineering infrastructure, GIS solutions, and statistical and clinical workflows for businesses, public-sector agencies, and research institutions. Leading a small specialized consulting team, Ryan's cross-industry expertise includes systems architecture, integration, and enterprise platform development using Python, R, SAS®, SQL/NoSQL, and modern JavaScript frameworks. He also serves as an Adjunct Professor in the Big Data Analytics Graduate Program and the Department of Mathematics and Statistics at San Diego State University. He earned his Master of Science in Big Data Analytics following the publication of his thesis, and his Bachelor of Science in Statistics with a Minor in Quantitative Economics, from San Diego State University.

Premier Analytics Consulting, LLC

www.Premier-Analytics.com

LinkedIn.com/in/RyanPaulLafler

rplafler@premier-analytics.com

Premier Analytics Consulting, LLC

AMS American Meteorological Society

1. Climatological and Meteorological Data Structures and Formats

Review of common data formats, structures, and how gridded climate and weather data are organized.

Understanding Gridded Climate and Weather Data Structures

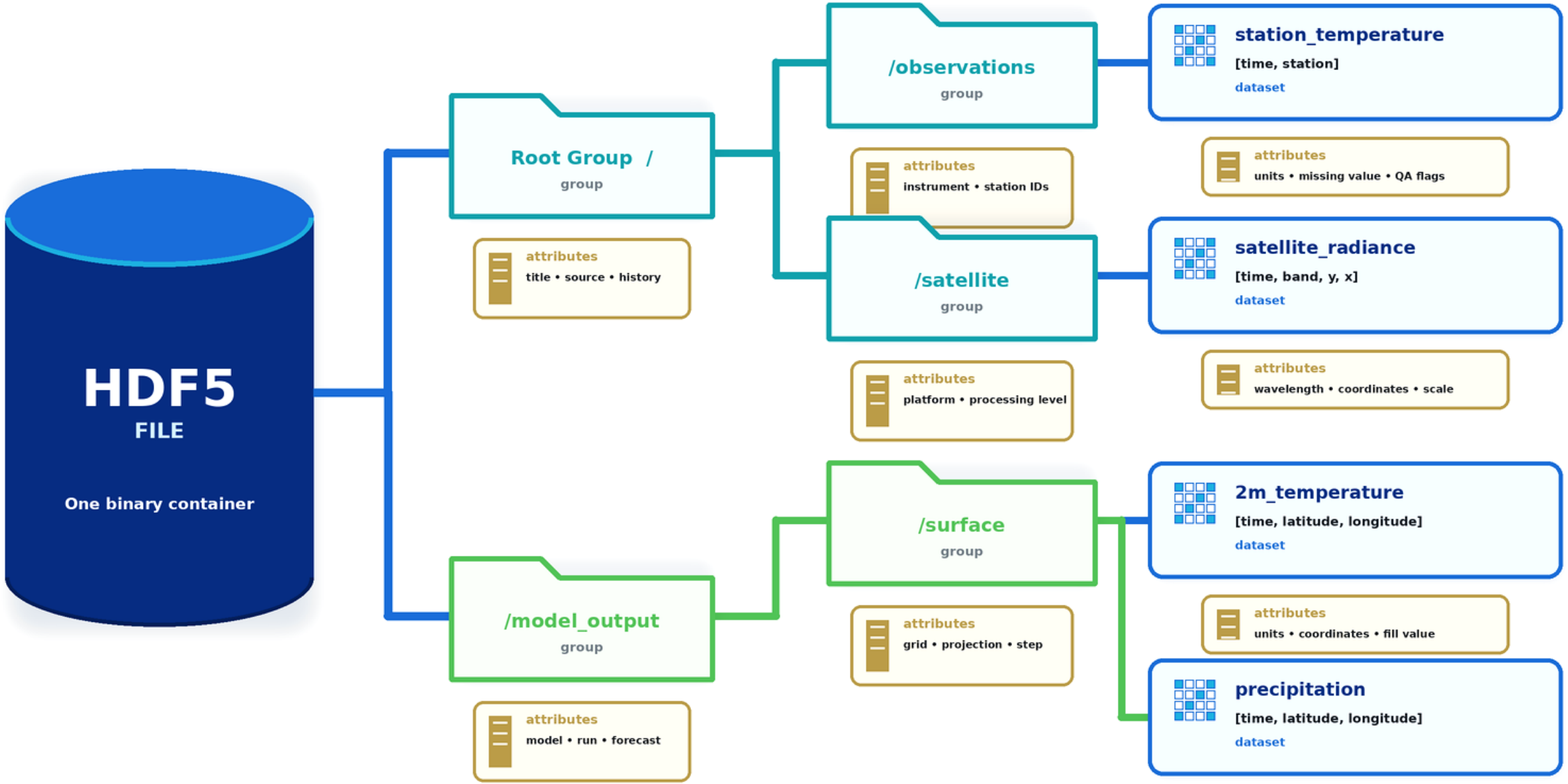
NetCDF, HDF5, Zarr, GeoTIFF, and GRIB / GRIB2 may represent similar gridded variables, but their internal structure determines how data are stored, retrieved, indexed, and analyzed at different scales.

- **Examine how each file format structures data** through multidimensional arrays, groups, chunks, tiles, raster bands, or individual message records
- **Compare access and retrieval patterns** across local files, cloud object storage, partial reads, streaming, parallel access, and terabyte- to petabyte-scale archives
- **Identify when each structure is appropriate** based on dataset size, compression, metadata, update frequency, interoperability, and analytical workflow
- **Connect formats to industry use cases** including numerical weather prediction, climate reanalysis, satellite products, remote sensing, GIS, historical archives, and cloud-optimized archives

HDF5: Hierarchical Scientific Data

- **HDF5 (Hierarchical Data Format)** stores large, multidimensional scientific datasets within single binary file
- Organizes data into groups, datasets, and attributes using hierarchical file structure with folders and subfolders
- Supports multidimensional arrays, compression, chunking, and partial reads for efficiently accessing selected portions of large datasets
  - Supports parallel I/O and slicing files to only load in data that is needed, not entire file
- Commonly used for satellite observations, radar products, remote sensing imagery, atmospheric measurements, and scientific model output
- Provides underlying storage structure for NetCDF-4 and works well when datasets contain multiple related variables, instruments, or processing levels
  - **Note:** Small datasets with many variables suffer from poor performance and no advantage from parallel I/O

HDF5: Hierarchical Scientific Data

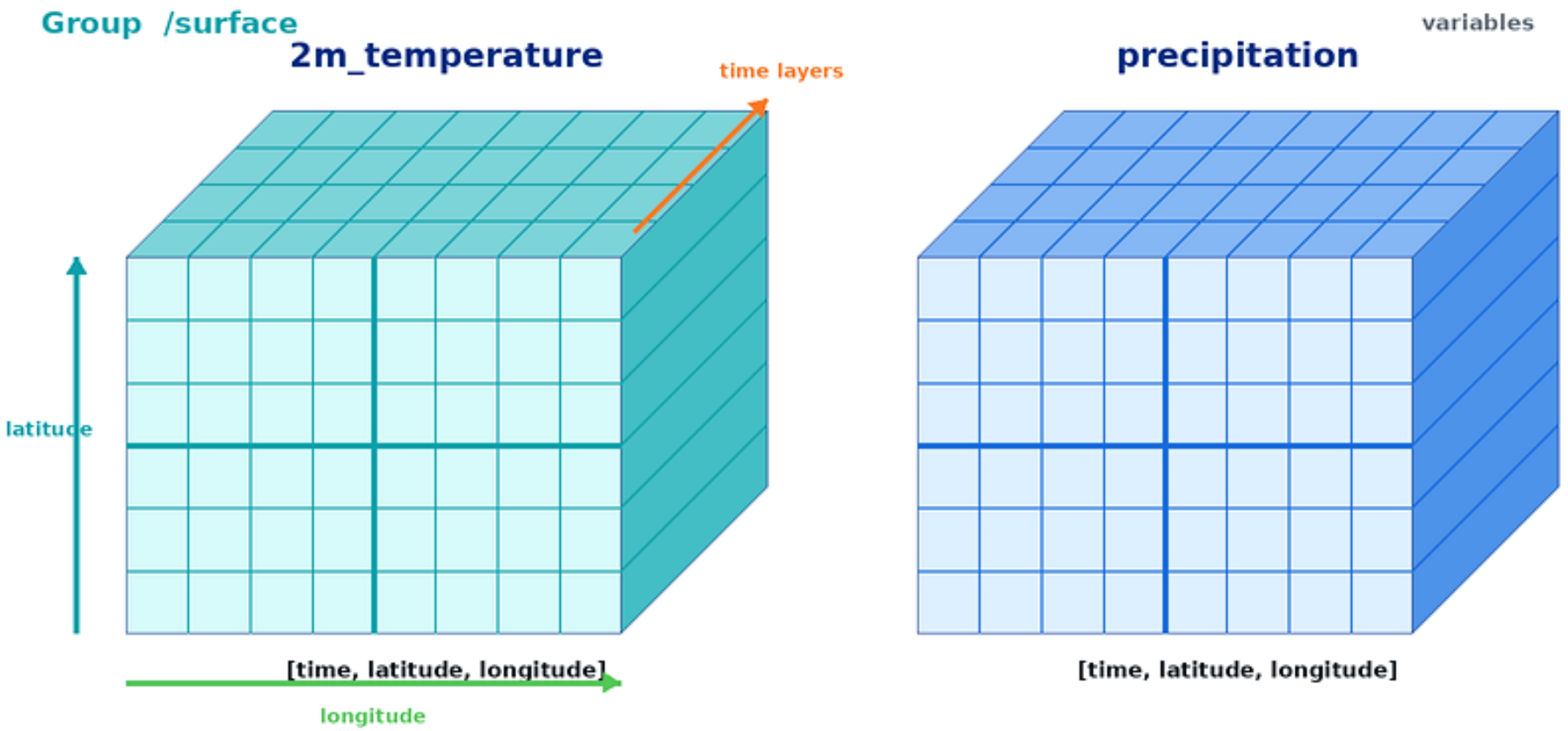


NetCDF (.nc): From Classic Formats to NetCDF-4

- **NetCDF (Network Common Data Form)** is self-describing, machine-independent family of formats for storing and exchanging multidimensional scientific data
- Earlier NetCDF formats use simpler flat data model built around dimensions, variables, and attributes
- NetCDF-4 uses HDF5 as storage layer adding hierarchical groups, chunked storage, optional compression, larger variables, multiple unlimited dimensions, partial reads, and parallel I/O
- Common uses include climate model output, reanalysis, numerical weather prediction, oceanographic data, atmospheric observations, GIS topography layers, and historical reanalysis archives
- Sensitive to chunking and compression schemes, metadata overhead for small variables, limited support for enhanced features in legacy software, and inefficiencies with distributed processing in cloud environments

In a NetCDF-4 file, HDF5 groups organize related variables and attributes hierarchically, named dimensions define shared array axes, and gridded values are stored as chunked multidimensional arrays within a single file.

NetCDF-4 (.nc): Multidimensional Variables with Shared Dimensions



Within the `/surface` group, NetCDF-4 file stores each variable as separate multidimensional cube with shared (`longitude`, `latitude`, `time`) coordinates. Variable cubes share dimensions and coordinate values but maintain their own data, metadata, compression, and chunking schemes.

Zarr (.zarr): Cloud-Native Chunked Multidimensional Arrays

- **Zarr is an open specification for storing compressed, chunked multidimensional arrays** across local file systems and cloud object storage
- **Zarr store is collection of metadata (JSON files) and chunked array data objects organized into hierarchical groups of folders and subfolders**
- Stores arrays as independently indexed chunks allowing selected regions and time ranges to be retrieved without reading *entire* dataset
- Supports parallel and distributed processing across **Amazon S3**, **Google Cloud Storage**, **Azure Blob Storage**, and other local (i.e., **MinIO**, **SeaweedFS**) and cloud object storage systems
- Uses include climate reanalysis, model ensembles, and petabyte-scale cloud archives too large to download

Zarr stores data chunks as independently addressable objects within a hierarchical store for parallel cloud access and targeted data retrieval. Metadata and attributes recorded in readable JSON files.

Zarr Store: CMIP6 ECMWF Daily Precipitation

1 Bucket and Group Hierarchy

v20170915/ root Zarr group

- **.zgroup** — declares Zarr format 2 group
- **.zattrs** — CMIP6 and CF global attributes
- **.zmetadata** — consolidated hierarchy metadata

Array subfolders

lat/ | lat_bnds/ | lon/

lon_bnds/ | time/ | time_bnds/

pr/

Selected precipitation array

2 pr/ Metadata and Chunks

.zarray

```
{ "zarr_format": 2,
  "shape": [23741, 361, 720],
  "chunks": [186, 361, 720],
  "dtype": "<f4",
  "compressor": "blosc / lz4" }
```

.zattrs

```
"_ARRAY_DIMENSIONS": ["time","lat","lon"]
"standard_name": "precipitation_flux"
"units": "kg m-2 s-1"
"original_name": "tp"
```

Chunk objects indexed by grid position

0.0.0 | 1.0.0 | 2.0.0

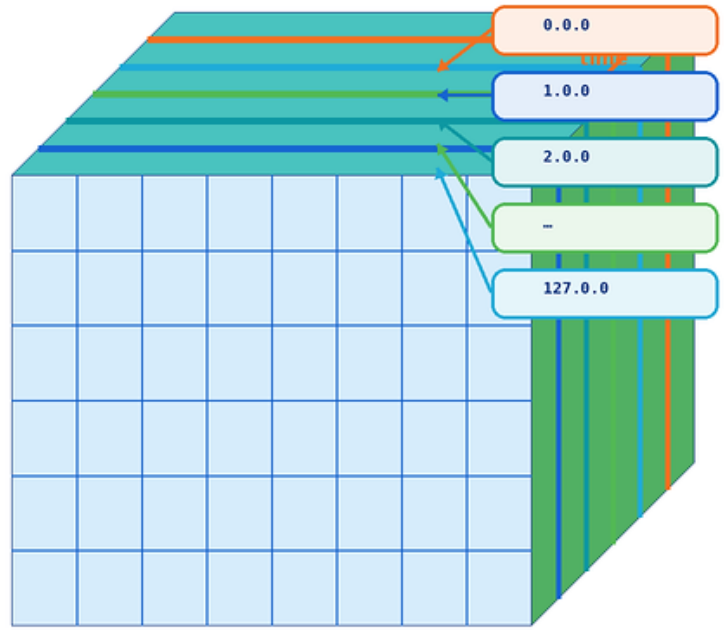
... | 127.0.0

time chunk • latitude chunk • longitude chunk

3 Indexed Chunked Array

pr[time, lat, lon]

23,741 daily time steps × 361 × 720 grid



Each chunk object stores 186 daily time steps across the complete 361 × 720 spatial grid

`cmip6/CMIP6/HighResMIP/ECMWF/ECMWF-IFS-HR/highresSST-present/r1i1p1f1/day/pr/gr/v20170915`

GRIB2 (.grib2): Message-Based Operational Weather Data

- GRIB2 is a WMO-standard binary format optimized for compact storage and rapid exchange of gridded meteorological data → data are encoded in bytes with starting and ending positions for each record
- Organizes data as sequence of self-contained messages representing one parameter, vertical level, reference time, forecast step, and grid
- Supports sequential and indexed access allowing tools such as **`ecCodes`** and **`wgrib2`** to locate and extract desired variables, levels, times, and forecast fields **without downloading entire file**
- Uses compression that significantly reduces file size and accelerates transmission of high-frequency numerical weather prediction output
- Common uses include operational forecasts, model dissemination, reanalysis, ensemble prediction, and aviation and marine weather products
- **Optimized for operational delivery, not analysis-ready historical workflows** → building long time series across many messages or files requires indexing, concatenation, grid alignment, and often conversion to NetCDF or Zarr

Remote GRIB2 Record Extraction: Atmospheric Variables

Remote GRIB2 file

Forecast cycle • 2024-04-18 12 UTC

MESSAGE 041
HGT • 500 hPa — Geopotential height

MESSAGE 042 • SELECTED
UGRD • 500 hPa — U wind component

MESSAGE 043 • SELECTED
VGRD • 500 hPa — V wind component

MESSAGE 044
DPT • 500 hPa — Dew point temperature

MESSAGE 045
TMP • 500 hPa — Air temperature

MESSAGE 046
RH • 500 hPa — Relative humidity

Sequential message stream

Each message carries one encoded field plus its GRIB2 metadata sections

Inventory / index

Maps messages to byte offsets

MESSAGE 042
UGRD • 500 hPa
byte range A–B

MESSAGE 043
VGRD • 500 hPa
byte range B–C

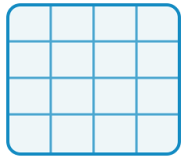
Common sidecars: .idx or .inv

HTTP RANGE GET
selected bytes

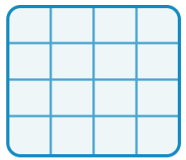
A GRIB2 inventory identifies the byte ranges needed to retrieve selected 500 hPa wind fields and metadata without downloading the entire file.

Selected 500 hPa fields

500 hPa (500 mb) • mid-troposphere



EXTRACTED MESSAGE
UGRD • 500 hPa
U wind component grid



EXTRACTED MESSAGE
VGRD • 500 hPa
V wind component grid

Field values + message metadata

- parameter identifiers and units
- pressure level and level type
- reference time, valid time, forecast step
- grid definition and packing information

Only selected messages are transferred

Premier Analytics Consulting, LLC

AMS American Meteorological Society

2. Cloud Object Storage Providers and Associated Python Libraries

Introduction to cloud object storage providers and Python wrapper libraries for accessing and interfacing with them.

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

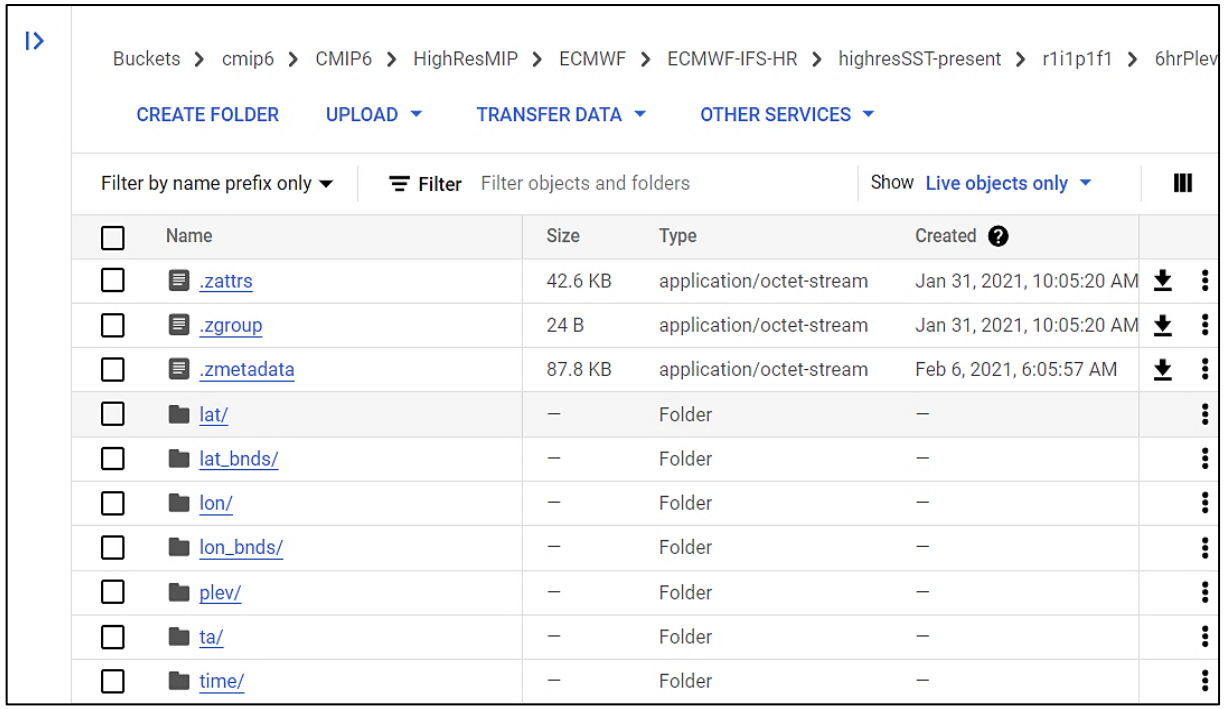
- Python contains a growing number of community-supported libraries for interfacing with cloud object storage
- Cloud object storage similar to file systems on local computers → uploaded files and data organized into hierarchies of folders and subfolders with role-based access control (RBAC) and permissions
  - **Google Cloud Storage (GCS)**
  - **Amazon S3**
  - **Azure Blob Storage**
- Cloud Object Storage writes files into BLOBs (Binary Large OBject) 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

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

Cloud Storage	Python Wrapper	Path Format	Best Use
Amazon S3 / S3-Compatible	`S3FS`	`s3://bucket/key`	File-like S3 access for Pandas, Xarray, Dask, and Zarr. Includes reads and writes for chunked and distributed workflows.
Google Cloud Storage	`GCSFS`	`gs://bucket/object`	Direct reads and writes to GCS for chunked and distributed workflows using Pandas, Xarray, Dask, and Zarr.
Azure Blob / Azure Data Lake Storage (ADLS) Gen2	`ADLFS`	`az://container/blob` *or* `abfs://container/path`	Consistent access to Azure Blob and Data Lake storage.
Provider-Agnostic File System Backend	`FSSPEC`	`protocol://location`	Common interface that selects the installed provider-specific wrapper.

Navigating CMIP6 ECMWF 6-Hour Air Temperature Zarr Store

CMIP6 / HighResMIP / ECMWF / ECMWF-IFS-HR / highresSST-present / r1i1p1f1 / 6hrPlevPt / ta / gr / v20170915 /



- Typical Zarr archive containing JSON metadata files and folder hierarchy structure for each dimension of data cube
- Google Cloud Storage pathway shows route to Zarr store
- Zarr store contains multidimensional index explicitly showing chunked dimensions: **latitude**, **longitude**, **time**, **pressure level**, and **air temperature**

Premier Analytics Consulting, LLC

AMS American Meteorological Society

3. Accessing and Retrieving Data from a CMIP6 Zarr Store in GCS

Python framework to access, query, and retrieve large-scale data repositories in Google Cloud Storage (GCS).

Create Reusable Python Pipeline to Retrieve GCS Zarr Store Data

```
class ECMWF_Historical_Pipe :
    def __init__ (self, bucket = "cmip6") :
        self.bucket = bucket
        # GOOGLE Cloud Storage (GCS) File System Client
        self.gcs_client = gcsfs.GCSFileSystem(
            project = bucket
        )

    def make_gcs_connection(
        self, main_path = "CMIP6/HighResMIP/ECMWF/ECMWF-IFS-HR",
        experiment = "highresSST-present", simulation = "r1i1p1f1",
        time = "day", var = "pr", gr = "gr", version = "v20170915" ,
    ) :
        gcs_path = f"gs://{self.bucket}/{main_path}/{experiment}/{simulation}/{time}/{var}/{gr}/{version}"

        # Return the connected client to the user
        return self.gcs_client.get_mapper(
            gcs_path
        )
```

Connect Pipeline to ECMWF Pressure-Level Air Temperature

```
# Connect to and initialize the GCS CMIP6 Client using our pipeline
pipe = ECMWF_Historical_Pipe()

# Connect to fixed ocean temp experiment and find simulation results for 6-hour air temperature
gcs_client = pipe.make_gcs_connection(
    experiment = "highresSST-present",
    simulation = "r1i1p1f1",
    time = "6hrPlevPt",
    var = "ta"
)

# Lazily load dataset and view its metadata
ds = xr.open_zarr(
    gcs_client, consolidated = True, chunks = "auto"
)
ds["ta"]
```

Connected HighResMIP ECMWF to air temperature Zarr store and lazily loaded data repository to understand metadata, chunking structure, and dimensions.

Lazily-Load and View Pressure Level Air Temperature

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

- 6-hour air temperature dataset contains 643.66 GiB worth of data → nearly 3/4ths of 1 terabyte!
  - Each chunk is partitioned to include, on average, 250 MB of data
  - **Air pressure dimension (plev)** contains 7 distinct levels
  - Each chunk includes 36 time slices and all 7 air pressure levels
- 94,964 time slices (6-hour measurements) from 1950-01-01 to 2014-12-31

Query 925 hPa Air Temp for Aug. 29, 2005 at 16:00 UTC

```
# Create graph workflow to filter and extract air temperature array at pressure level
## 2-Step query: Must first retrieve correct time chunk, then extract air pressure level array
graph = ds["ta"].sel(
    plev = 92500,
    time = datetime(2005, 8, 29, 18),
    method = "nearest"
)

# Normalize longitude values to --> [-180, 180]
graph["lon"] = xr.where(graph["lon"] > 180, graph["lon"] - 360, graph["lon"])
graph = graph.sortby("lon")

with ProgressBar() :
    result = graph.compute()
```

Use nearest neighbor search to extract closest matching date from relevant array chunk. Then filter by pressure level (in pascals) from extracted time array and normalize longitude values. Build graph then execute and persist retrieved array result in memory.

Visualize 925 hPa Air Temp for Aug. 29, 2005 at 16:00 UTC

```
# Generate wide blank canvas for plotting
fig, ax = plt.subplots(
    nrows = 1, ncols = 1,
    figsize = (15, 10),
    subplot_kw = {
        "projection": ccrs.PlateCarree()
    }
)

# Plot climate data (first layer)
result.plot(
    ax = ax, cmap = "turbo",
    transform = ccrs.PlateCarree(),
    zorder = 10,
    cbar_kwargs={
        "shrink": 0.5,
        "orientation": "vertical",
        "pad": 0.05
    },
)
```

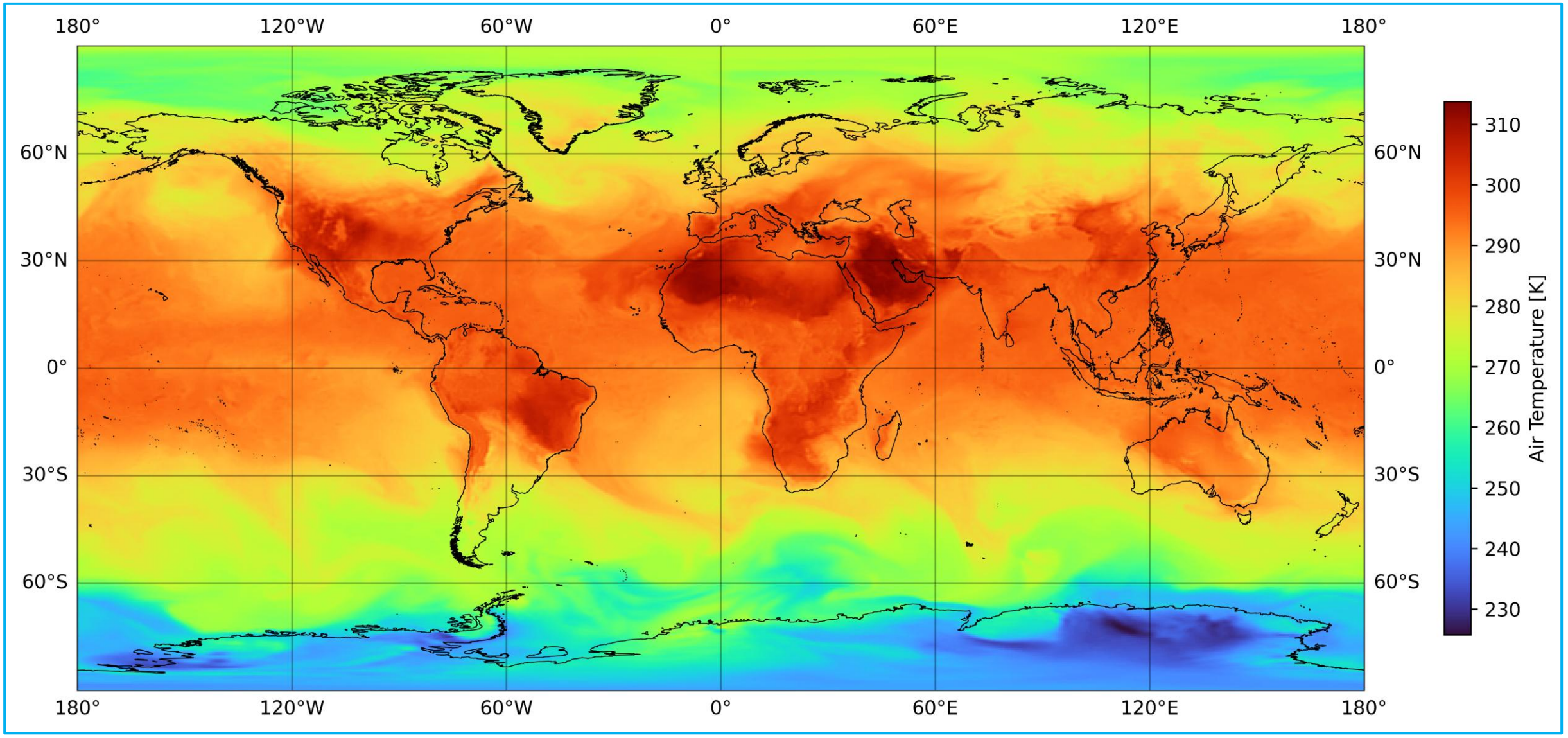
```
# Add coastlines to plot using Cartopy
ax.coastlines(
    resolution = "10m",
    lw = 0.45, color = "black", zorder = 20
)

# Add longitude-latitude gridlines
ax.gridlines(
    draw_labels = True, color = "black",
    alpha = 0.35, zorder = 30
)

# Remove title
ax.set_title(None)

# Export figure as high-resolution PNG
fig.savefig(
    "Images/2005-08-29-18UTC-ta.png",
    bbox_inches = "tight", dpi = 900
)
```

Simulated Air Temperature at 925 hPa for Aug. 29, 2005 at 16:00 UTC



Connecting Cloud Data to Forecasting and Decision Systems

- **Select the right structure** → GRIB2 for operational delivery, NetCDF/HDF5 for structured scientific data, and Zarr for parallel cloud analysis of large, chunked data repositories
- **Connect directly to cloud storage** → Use FSSPEC and associated wrappers to access data across Amazon S3, Google Cloud Storage, and Azure
- **Retrieve only what is needed** → Subset variables, locations, time ranges, and pressure levels without downloading entire archives
- **Scale computation efficiently** → Combine Xarray, chunking, Dask lazy execution, and parallel processing across laptops or distributed clusters
- **Build reusable forecasting pipelines** → Validate, transform, and convert cloud data into model inputs, historical baselines, anomalies, and risk indicators
- **Deliver decision-ready outputs** → Publish forecasts and indicators through maps, dashboards, APIs, reports, and automated alerts

Premier Analytics Consulting, LLC

AMS American Meteorological Society

4. Questions, Recap, and Summary

Thank You for Attending! Please contact me with any questions at:

rplafler@premier-analytics.com

www.Premier-Analytics.com

LinkedIn.com/in/RyanPaulLafler