OPTIMIZING FOR THE CLOUD

Facilitating use of challenging datasets using Kerchunk and Dask

REACHING FOR THE CLOUD

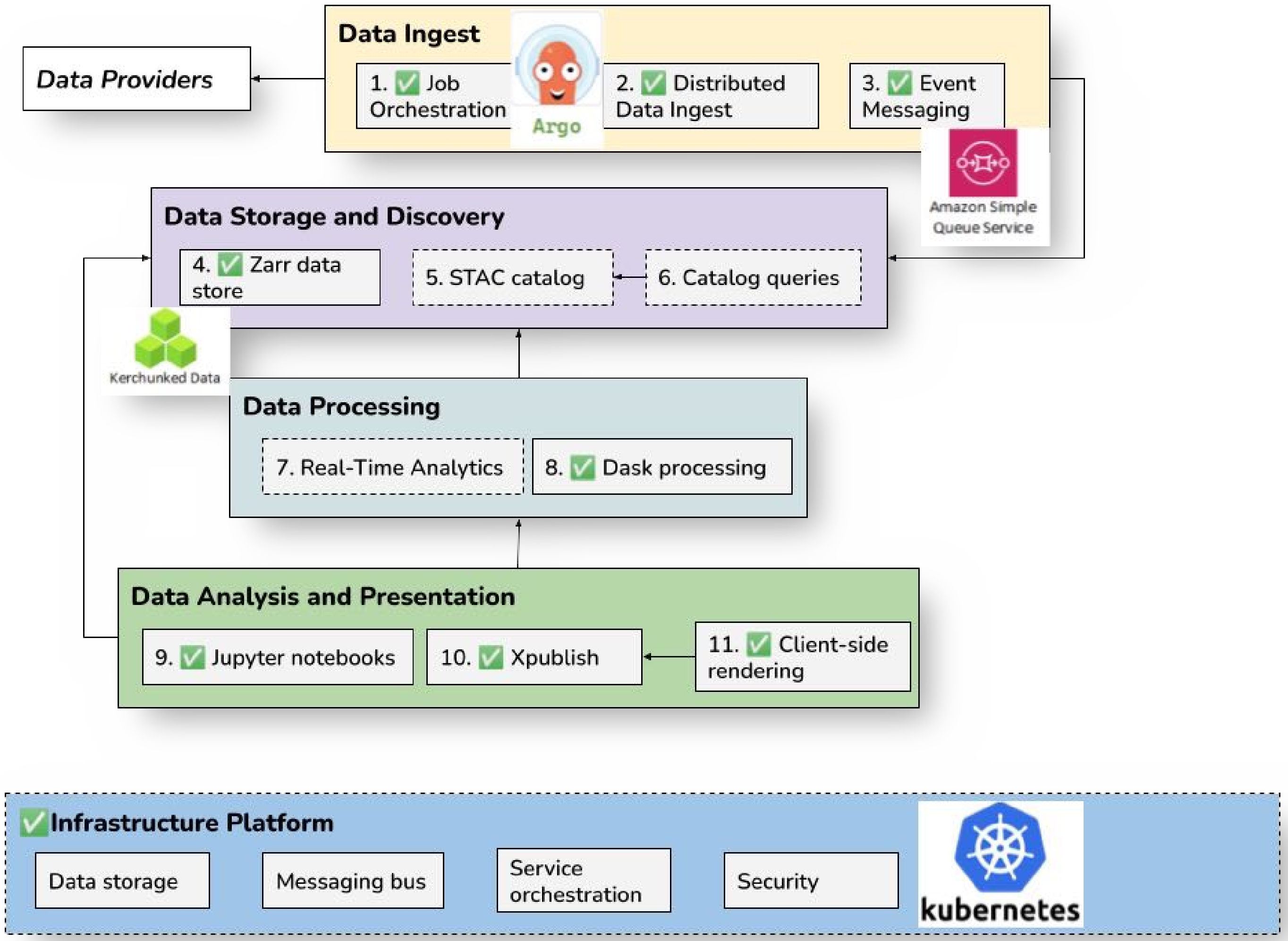
Architecting a Cloud-Native Service-Based Ecosystem for DMAC

- Goal: increase the use of IOOS data and promote connections to other disciplines **by lowering the barriers** for entry
- Objectives:
  - understand the current state, challenges, and opportunities
  - develop a roadmap and proposed architecture
  - prototypes and demonstration datasets

Prototype Roadmap

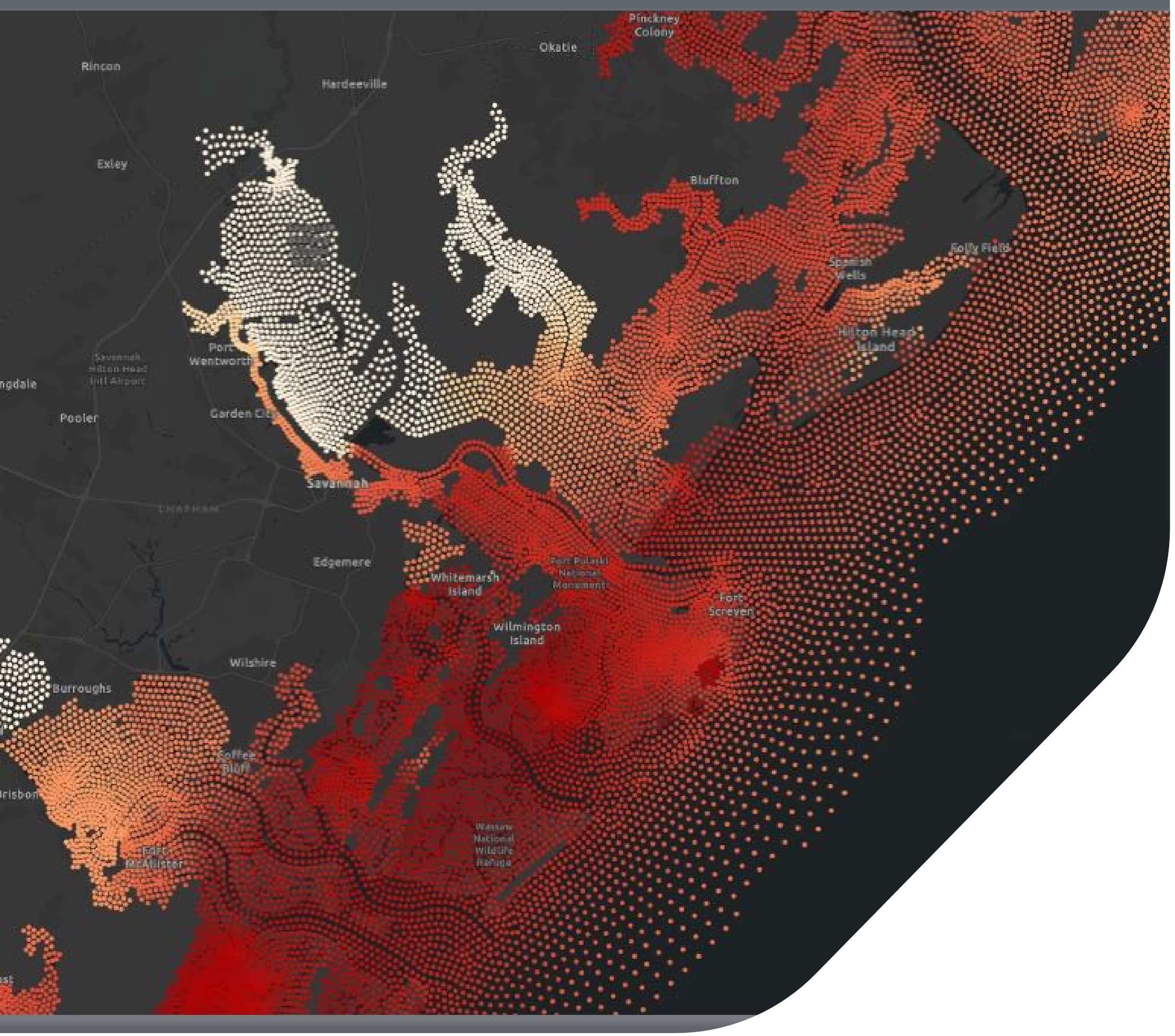
Workflow Components

- Ingest
- **Storage** and discovery
- **Processing**
- **Analysis** & Presentation



NCDIS 40-Year Reanalysis

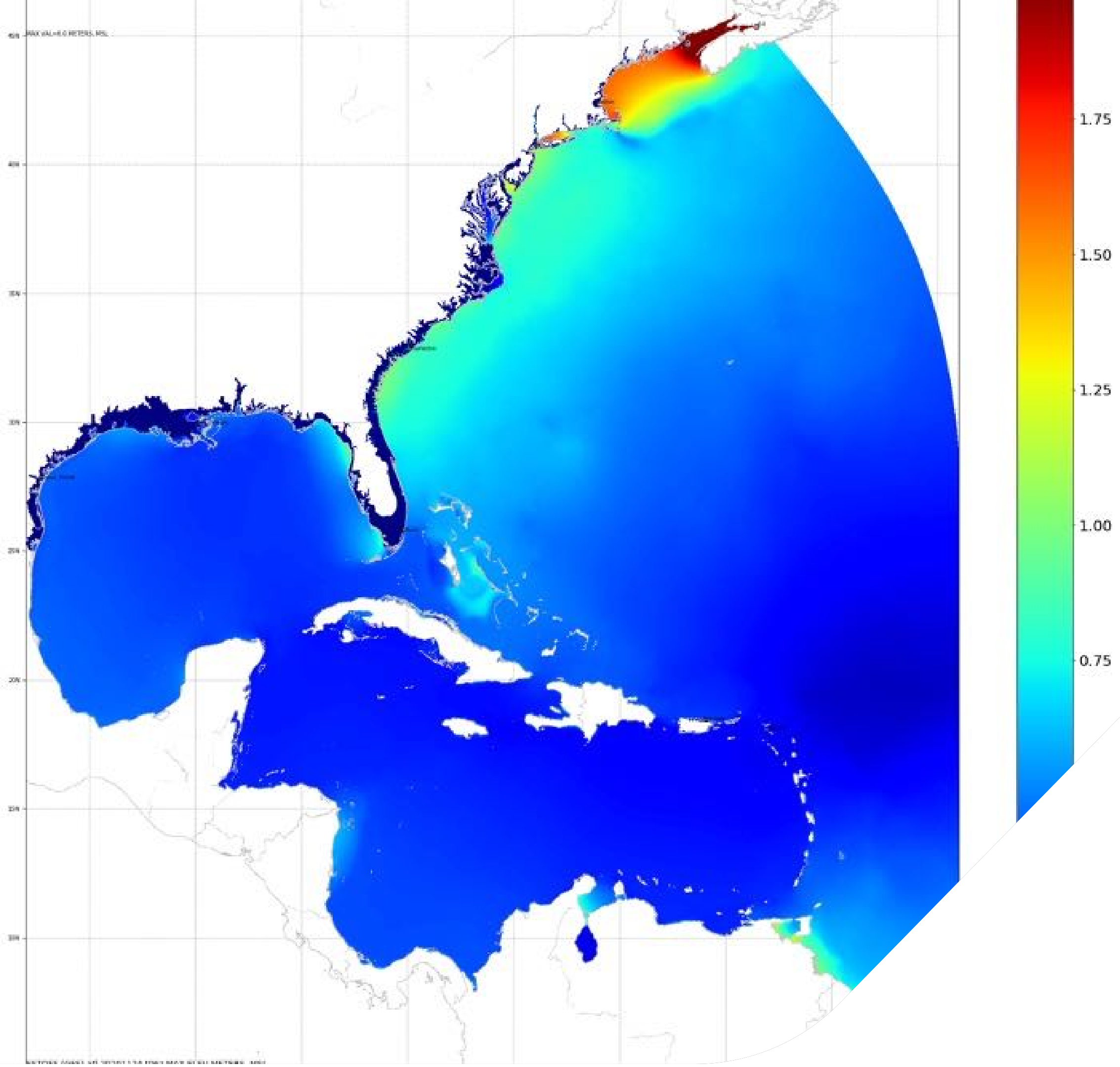
Case Study #1



CASE STUDY #1

National Coastal Data Information System (NCDIS) Multi-decadal Year Reanalysis

- Long-term Reanalysis of Water Levels & Waves
- Integrating models & observations to predict flooding between tide stations
- Fills extensive gaps between observation locations



CASE STUDY #1

National Coastal Data Information System (NCDIS)
40 Year Reanalysis

- Atlantic reanalysis uses the HSOFS mesh (~1.8 million nodes)
  - 40 years of hourly data every 500m along the coast, including within bays, estuaries, and coastal river mouth entrances
  - ~70 TB of storage
- Pacific reanalysis uses the GSTOFS mesh (~450k-2.2m nodes)
  - Resolution down to 80 m for Hawaii and US West Coast
  - 90-120 m for Pacific Islands
  - ~300 TB of storage

Case Study #1: NCDIS Reanalysis

- ADCIRC output for ~1.8 million nodes over a 40-year period - ~ 70 TB of data (5.5 TB for water level)
- Original data had a chunk size of ~1.8 GB which was leading to slow access times
- Rechunked the underlying netCDF to manageable chunk sizes
- Leveraging Kerchunk and intake catalogs
- Much faster access time for both extracting a timeseries and the entire grid

Intake Catalog

```yaml
sources:
  NCDIS-WaterLevel-1979-2021:
    driver: intake_xarray.xzarr.ZarrSource
    description: 'NCDIS Water Level, 1979 - 2021'
    args:
      consolidated: False
      chunks:
        time: 240
        node: 160000
      urlpath: "reference://"
      storage_options:
        fo: 's3://ncdis-ra/jsons/fort.63_post_1979-2021.json'
        remote_options:
          anon: false
        remote_protocol: s3
```

```python
catalog =  intake.open_catalog('s3://ncdis-ra/ncdis_intake.yml')
ds = catalog['NCDIS-WaterLevel-1979-2021'].to_dask()
ds.zeta
```

xarray.DataArray 'zeta' (**time**: 376943, **node**: 1813443)

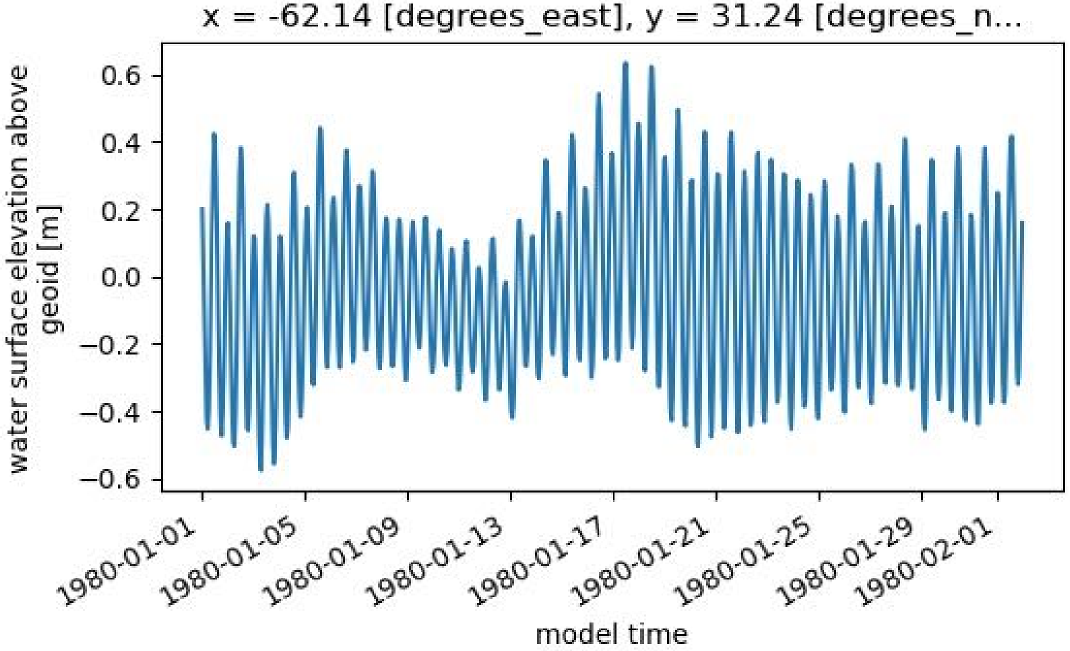
	Array	Chunk
Bytes	4.97 TiB	292.97 MiB
Shape	(376943, 1813443)	(240, 160000)
Count	18853 Tasks	18852 Chunks
Type	float64	numpy.ndarray

Case Study #1: NCDIS Reanalysis

2 s to extra a one-month time series
3 m to extract 141 annual time series
6 s to compute the daily max for the grid

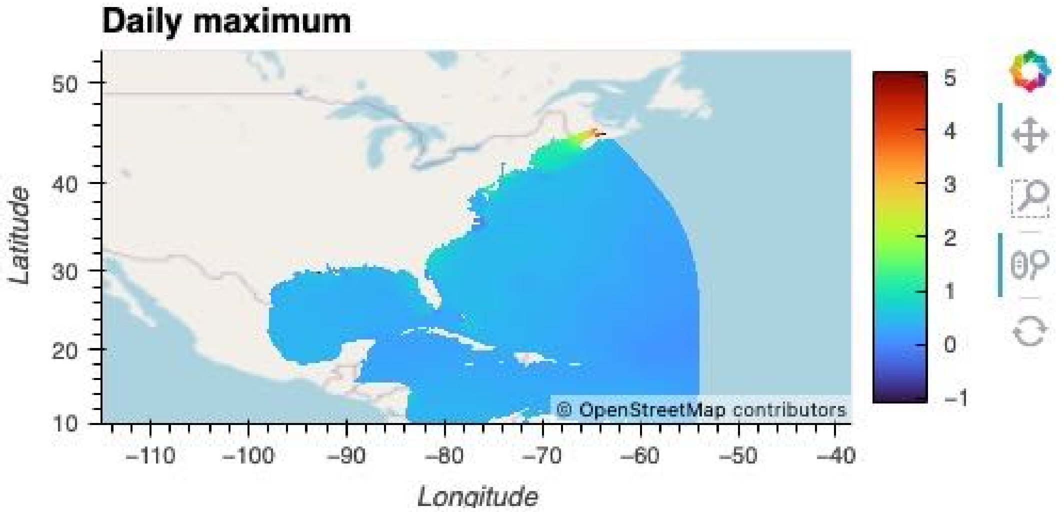
```
%%time
water_level = ds['zeta'].sel(time=slice('1980-01-01','1980-02-01'),
                             node=1477363).plot(aspect=2, size=3)
```

```
CPU times: user 300 ms, sys: 16.7 ms, total: 317 ms
Wall time: 2.24 s
```




```
%%time
zeta_sel =  ds['zeta'].sel(time=slice('1985-01-01','1985-01-01'))
grid_max = zeta_sel.max(dim='time').load()
plot_gridded_output("Daily maximum", grid_max, "turbo", 500,250)
```

```
CPU times: user 1.18 s, sys: 285 ms, total: 1.46 s
Wall time: 6.31 s
```



Case Study #1: NCDIS Reanalysis

- Requirements for post-processing and product generation
  - Pull 40-year variables (time series or max values) on the 500-m grid
  - Pull 40-year variables (time series or max values) on the nodes
  - Validate model output alongside NWLON, USGS, and USACE water level observations
  - Calculate daily maximums and monthly means
  - Regression analysis, histograms, trends,
- Dask and Nebari
  - Deployed Nebari in AWS
  - Gives access to a Kubernetes cluster without having to manually configure it
  - Utilizes Dask Gateway so work can be scaled across the cluster

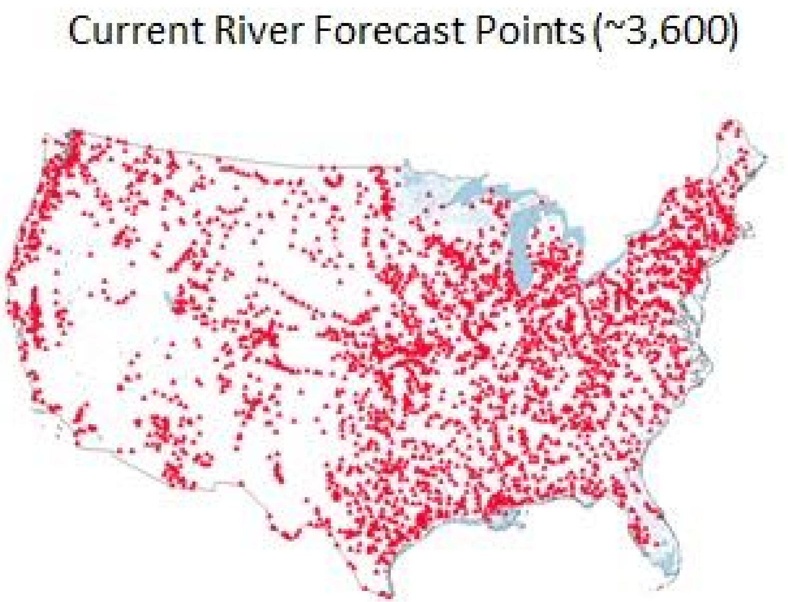
National Water Model

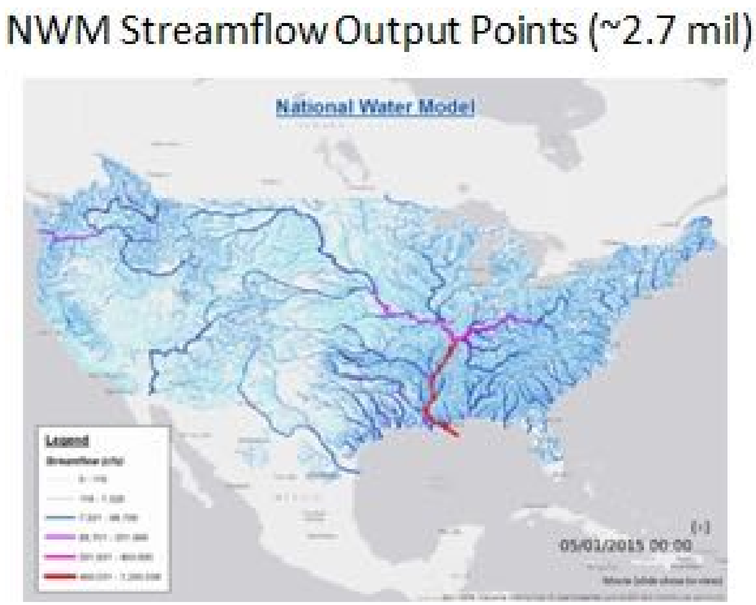
Case Study #2

Case Study #2: National Water Model

- Supplements traditional NWS river forecasts
- Streamflow and discharge forecast for 2.7 million reaches
- Additional hydrologic information on 1km, 250m, & 100m grids
- Updated hourly
- ~5 GB of data per day
- NOT gridded…networked

- Released original API c. 2019
  - Queries the NODD S3 bucket
  - Temporal, spatial, and variable subsetting
  - Terminal point and feature ID filters





CASE STUDY #2

National Water Model Forecast

- Make access easier and more performant
- Use the files that are already stored in the NODD AWS bucket
  - Avoid duplicating the data
  - Each forecast hour is written to a different netCDF file in the AWS bucket
  - Latitude / longitude information not stored with the model output
  - Network information not stored with the model output

CASE STUDY #2

National Water Model Forecast

Completed Tasks

- Transform model output in NODD bucket to cloud optimized format
- Create a net CDF file with the lat/lon info
- Create virtual dataset by merging Kerchunked output and lat/long data
- Create intake catalog to allow data access w/o knowledge of underlying data file format

Remaining Tasks

- Workflow to automatically kerchunk files when they arrive in the NODD AWS Bucket
- Directed graph to allow stream tracing

Case Study #2: National Water Model

Example 1:
- Open the dataset with python and view structure

```
catalog = intake.open_catalog('s3://nextgen-dmac/nwm/nwm_intake.yml')
ds = catalog['NWM_Data'].to_dask()
ds
```

xarray.Dataset

▸ Dimensions: (**time**: 18, **feature_id**: 2776738, **reference_time**: 1)

▾ Coordinates:

feature_id	(feature_id)	float64	101.0 179.0 ... 1.18e+09 1.18e+09
latitude	(feature_id)	float32	dask.array<chunksize=(2776738,), met...
longitude	(feature_id)	float32	dask.array<chunksize=(2776738,), met...
reference_time	(reference_time)	datetime64[ns]	2023-06-12
time	(time)	datetime64[ns]	2023-06-12T01:00:00 ... 2023-06-...

▾ Data variables:

crs	(time)	object	dask.array<chunksize=(1,), meta=np.nd...
nudge	(time, feature_id)	float64	dask.array<chunksize=(1, 2776738), m...
qBtmVertRunoff	(time, feature_id)	float64	dask.array<chunksize=(1, 2776738), m...
qBucket	(time, feature_id)	float64	dask.array<chunksize=(1, 2776738), m...
qSfcLatRunoff	(time, feature_id)	float64	dask.array<chunksize=(1, 2776738), m...
streamflow	(time, feature_id)	float64	dask.array<chunksize=(1, 2776738), m...
terminal	(feature_id)	float64	dask.array<chunksize=(2776738,), met...
velocity	(time, feature_id)	float64	dask.array<chunksize=(1, 2776738), m...

Example 2:
- Extract all data in an AOI at a specific time and compute the maximum value
- Cloud optimized data makes accessing the data very performant
- This example took less than 2 seconds to complete

```
%%time
lon_min= -90.5777
lat_min=35
lon_max=-75
lat_max=37
time_s='2023-06-12T15:00'

ids = np.where((ds.longitude>=lon_min) &(ds.longitude<=lon_max) &
        (ds.latitude>=lat_min) & (ds.latitude <= lat_max))[0]

s_subset = ds.streamflow.sel(time=time_s).isel(feature_id=ids)
max_val = s_subset.max().compute()
print("Maximum value: ",max_val.values)
```

```
Maximum value:  7701.459827858955
CPU times: user 184 ms, sys: 7.94 ms, total: 192 ms
Wall time: 1.68 s
```

Thanks for your time

For more information please contact:

Kelly Knee
Kelly.knee@rpsgroup.com

Cheryl Morse
Cheryl.morse@rpsgroup.com