

# “Grote databestanden gebruiken als data lake”

*Robin Kooyman, Peter Thijssen (MARIS)*  
- [robin@maris.nl](mailto:robin@maris.nl) [peter@maris.nl](mailto:peter@maris.nl) -

# Introductie

10-15 min

## Vragen aan de zaal

- Wat verstaan jullie onder “grote” databestanden? Wat is groot?
- Noem een aantal voorbeelden
- Waar lopen jullie tegenaan als problemen voor publicatie en gebruik?

# MARIS visie en ervaring

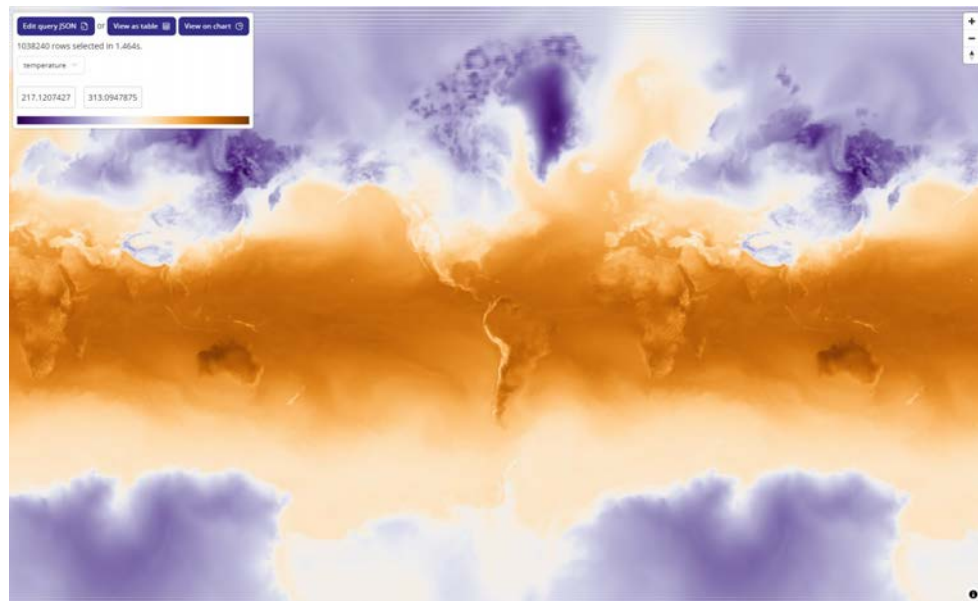
10-15 min

# Wat is voor ons grote data?

- Miljoenen kleine databestanden met miljarden observaties/datapunten
  - WADAR, Aquadesk, SeaDataNet, Euro-Argo, WOD, ..
  - Hoe (her)gebruik je deze bestanden? als geharmoniseerd data lake vanuit een notebook, model, AI?  
=> zie ons eerdere webinar op [Digishape.nl](http://Digishape.nl)
- Dataproducten en aggregaties in hoge resolutie (bathymetry, climatologie, ..)
  - EMODnet, Copernicus, C3S, CMEMS, EDITO
  - Hoe gebruik je deze grote bestanden via een service, met voldoende performance?  
=> Focus van vandaag.

# Use case: Gebruik van ERA5 data voor C3S Climate change attribution

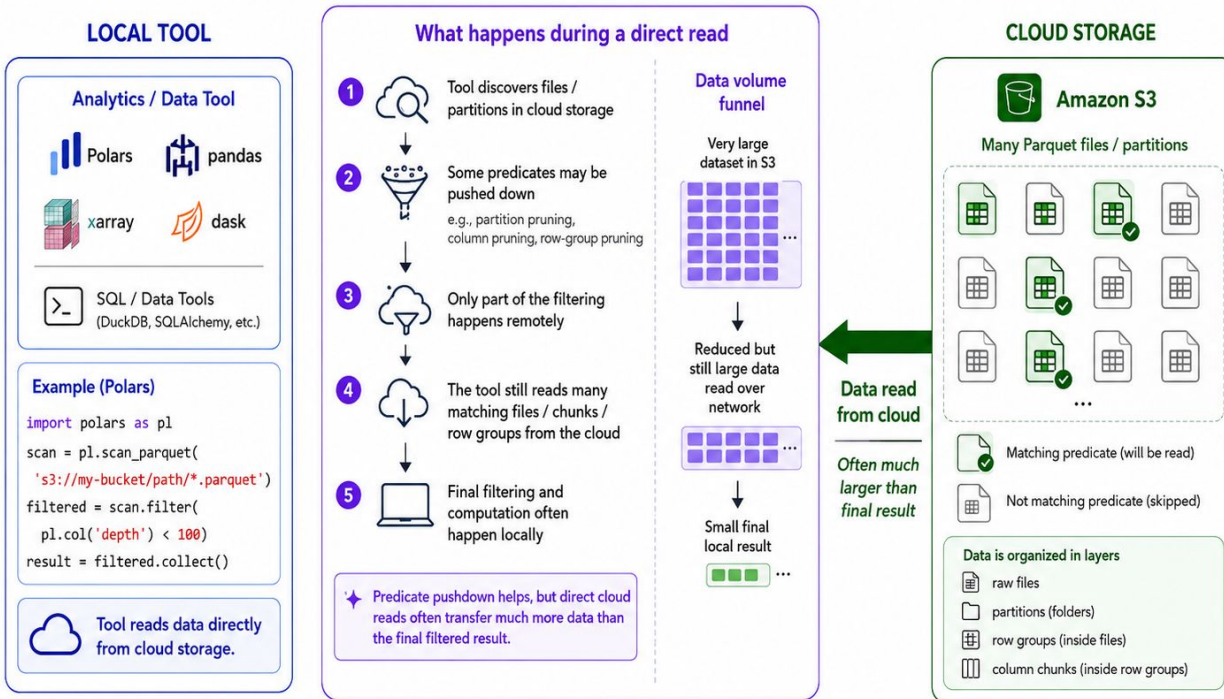
- Stored in Zarr
- ERA5 daily temperature since 1950
- 500GB+ across 75 yearly Zarr files
- ACCESS slow, not usable.
- Beacon Data Lake on top of the collection of Zarr files for:
  - Management
  - Subsetting Performance
  - Reduce Network Transfer
  - Harmonization
  - Common Query Framework
    - SQL
    - JSON
  - Real Time Data Manipulation
  - Product Creation



# Bestandsformaten een deel van de oplossing

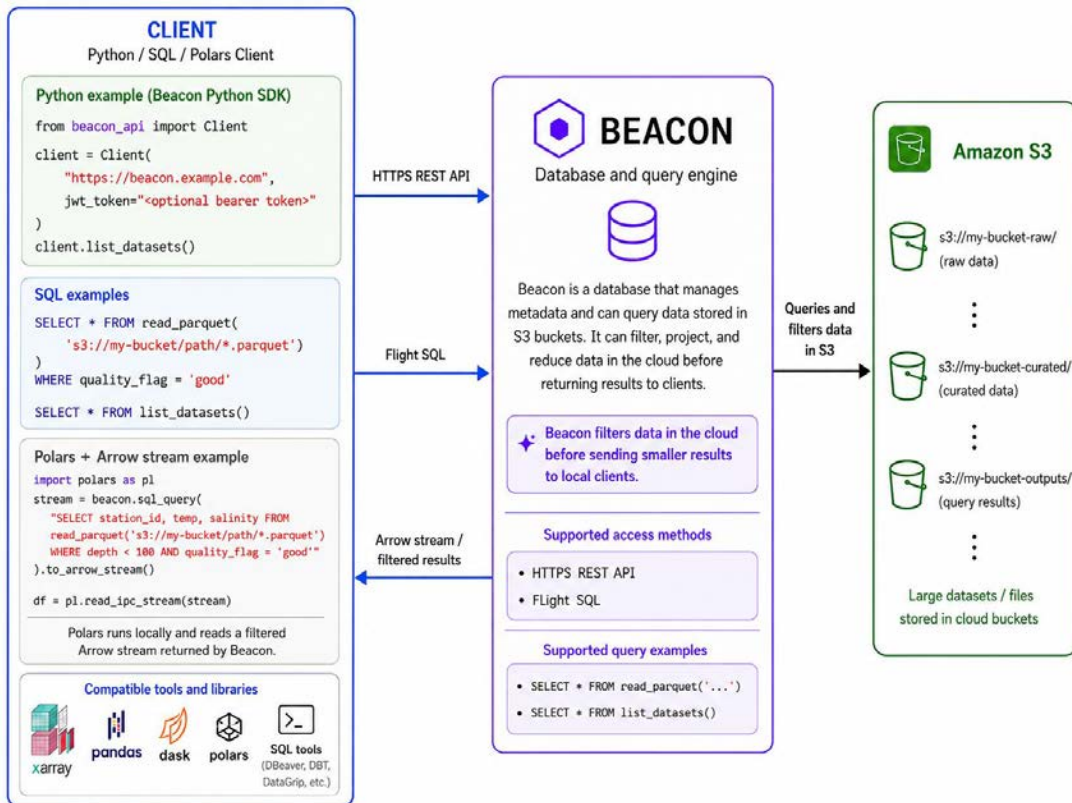
- Cloud Native Storage Formats
  - ZARR
  - Parquet
  - ATLAS
- Network Latency + Network Bandwidth
- Depends on being able to filter chunks/rowgroups upfront
  - Called “Predicate Pushdown”
  - If unable to filter up front, you download entire dataset (Could be 100 of GB’s)
- Compute Distance
  - Local Compute -> Network (1GB of chunks) -> Cloud Storage (S3,..)
  - Local Compute -> Network (100MB of chunks) -> Compute System (20ms latency) -> Cloud Storage (S3,..)
- Compute system next to data would always enable efficient filtering

# Direct Cloud Read with Partial Predicate Pushdown



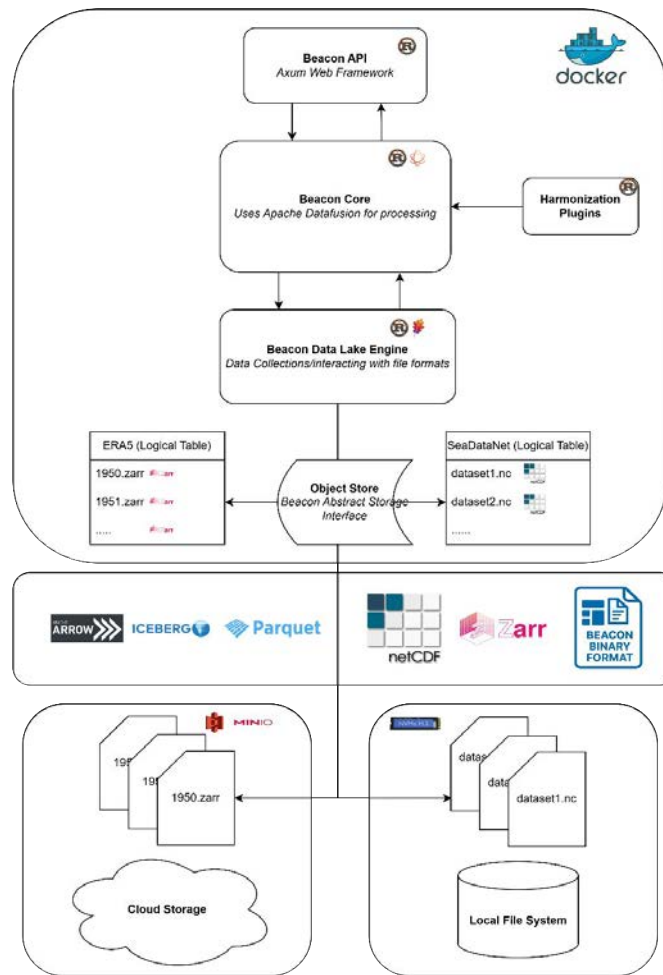
- Pushdown can reduce columns, partitions, and row groups
  - Not every predicate can be executed remotely
  - More data may be read from cloud than the size of the final filtered dataset

# Mogelijkheden met Beacon software



# BEACON IN A NUTSHELL

- **Open-Source High Performance ARCO Data Lake**
- **Written in Rust + C**
- **Runs on any platform using Docker**
- **Consists of:**
  - Rest API (if needed with authentication!)
  - Core Libraries (heavy lifting)
  - Data Harmonization Libraries (single output file)
- **Supports Reading & Writing data from/to**
  - Any S3 Compatible Storage
    - AWS S3
    - MinIO
    - Any other S3 bucket
  - Local File System
- **Supported Data Formats (s3, local, nas)**
  - CSV
  - Zarr (any structure, timeseries, gridded, cruises & supports parallel chunked reading)
  - Parquet
  - Apache Iceberg
  - Arrow
  - NetCDF (any structure, timeseries, gridded, cruises & supports chunked reading)
  - Beacon Binary Format (also ARCO)
  - Icechunk (Work in Progress)
- **Creating Real Time Collections (Logical Tables)**
  - On top of existing files:
    - Eg: seadatanet/\*.nc
    - Eg: era5/\*.zarr



More info:  
<https://beacon.maris.nl>

Open source  
 Free  
 Fast  
 Easy to deploy  
 Easy to use

# BEACON IN A NUTSHELL

- **Query using JSON**
  - Select relevant columns
  - Filter using:
    - Value Ranges / Equalities
    - Metadata
    - Polygons
- **Query using Python Library**
  - PyPi: <https://pypi.org/project/beacon-api/>
  - GitHub: <https://github.com/maris-development/beacon-py>
  - Documentation: <https://maris-development.github.io/beacon-py/getting-started/>
- **Query using SQL**
  - Full SQL support (read-only)
  - Write for admins only
- **Output formats:**
  - ODV ASCII
  - CSV
  - NetCDF
  - Arrow
  - Parquet
  - GeoParquet
  - Zarr

## Create and execute a query

To create and execute a query on a specific table, you can use the `query` method of the `Table` object. Here's an example of how to create a simple query that selects specific columns and applies filters:

```
df = (
    tables['default'] # Select the 'default' table as our data source
    .query() # Create a new query on the selected table
    .add_select_column("LONGITUDE") # Select the LONGITUDE column
    .add_select_column("LATITUDE") # Select the LATITUDE column
    .add_select_column("JULD")
    .add_select_column("PRES")
    .add_select_column("TEMP")
    .add_select_column("PSAL")
    .add_select_column(".featureType") # Select the .featureType column
    .add_select_column("DATA_TYPE")
    .add_range_filter("JULD", "2020-01-01T00:00:00", "2021-01-01T00:00:00") # Filter
    .add_range_filter("PRES", 0, 10) # Filter for pressure between 0 and 10 dbar for
    .to_pandas_dataframe() # Execute the query and return the results as a Pandas DataFrame
)
df
```

### Note

The `to_pandas_dataframe` method executes the query and returns the results as a Pandas DataFrame.

## What is Atlas File Format

- Directory-based store for thousands of named datasets
- Each dataset contains:
  - N-dimensional typed arrays + typed attributes
  - Global typed attributes
- Built on the .af binary array format
- Uses a variable-first layout:
  - One file per array/variable across all datasets
- Designed for workloads like:
  - Time steps
  - Station based data
  - Large analysis
- Works with:
  - Local disk
  - S3

```

1 store = pyatlas.Atlas.open("bgc")
2 store.list_datasets()

```

0ds

```

['files\\GL_PR_BO_DBBH.nc',
 'files\\GL_PR_BO_SHIP.nc',
 'files\\GL_PR_CT_2FGX5.nc',
 'files\\GL_PR_CT_FNFP.nc']

```

```

1 ds = store.to_xarray("files\\GL_PR_BO_SHIP.nc")
2 ds

```

1.7s Open 'ds' in Data Wrangler

xarray.Dataset

Dimensions: (TIME: 506248, DEPTH: 414)

Coordinates:

TIME	(TIME)	datetime64[ns]	1841-03-21 ... 2018-03-22T12:04...
LONGITUDE	(TIME)	float32	disk.array<chunksiz... (1000), meta=np.ndarr...
LATITUDE	(TIME)	float32	disk.array<chunksiz... (1000), meta=np.ndarr...
DEPTH	(TIME, DEPTH)	float32	disk.array<chunksiz... (1000, 414), meta=np.n...
TRAJECTORY	()	object	"SHIP"

Data variables: (35)

Attributes: (50)

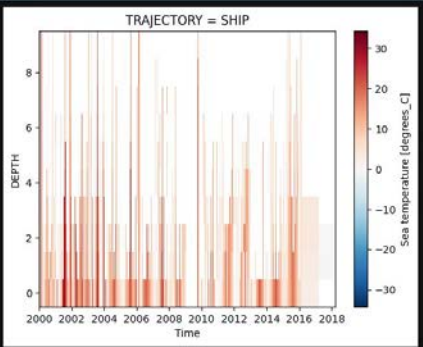
```

1 ds2 = ds.sel(TIME=slice("2000-01-01", "2020-12-31"), DEPTH=slice(0,10))
2 import matplotlib.pyplot as plt
3 ds2["TEMP"].plot(x="TIME", y="DEPTH")
4 plt.show()

```

Argument missing for parameter "self"

0.5s



## Why Atlas?

- Optimized for queries like:
  - Read variable X across many datasets
  - Subset variable X,Y,Z across many datasets
- Avoids opening thousands of files by storing all datasets for a variable in one .af file
- Supports chunked I/O for efficient partial reads and writes
  - Reducing memory consumption
  - Dask compatible!
- Configurable compression:
  - Zstd
  - LZ4
  - Uncompressed
- Persists min, max, null count, and row count statistics on flush
- Human-readable atlas.json metadata registry
- Rust-first, async-friendly design with shared caches and thread-safe access
- Python bindings available

Feature	NetCDF-4	Zarr v3	ATLAS
Layout	Dataset-first	Dataset-first	Variable-first
Compression	Deflate / Zstd / ...	Any codec plugin	Zstd / LZ4 / None
Chunking	Yes	Yes	Yes
Cloud object store	No (needs FUSE/etc)	Yes (native)	Yes (via <code>object_store</code> )
Multiple datasets in one file	No	No	Yes (all datasets per variable)
Metadata format	Binary (HDF5)	JSON	JSON
Cross-dataset column scan	Slow (N file opens)	Slow (N directory opens)	Fast (1 file open)
Partial reads	Yes	Yes	Yes
Statistics (min/max/nulls)	No	No	Yes (persisted on flush)
Self-describing metadata	Yes	Yes	Yes ( <code>atlas.json</code> )
Language support	C/Python/Julia/...	Python/Java/...	Rust
Mutable after write	Limited	Yes	Yes (chunked overwrites + compact)

# Vraag en antwoord

## 10-15 min