Koninklijk Nederlands
Meteorologisch Instituut
Ministerie van Infrastructuur en Waterstaat

RODEO WP5 EDR API Workshop

Vienna

08-04-2025

Lukas Phaf

Jeffrey Vervoort

Welcome!

- Slides:
  - https://tinyurl.com/rodeo-wp5-workshop
- GitHub Repo
  - https://github.com/EUMETNET/ogc-edr-workshop

- Feedback welcome!

Schedule and outline – Day 1

Time	Item
09:00	Welcome
09:10	Introduction of participants
09:20	Introduction to RODEO
09:40	Develop your own EDR
10:50	Coffee break
11:05	Develop your own EDR
12:45	Lunch break
13:35	Develop your own EDR
15:15	Coffee break
15:30	Develop your own EDR
17:00	End of day

Step	Item
0	Setup environment
1	Landing page
2	Retrieve data for single location
3	Filtering (time and parameters)

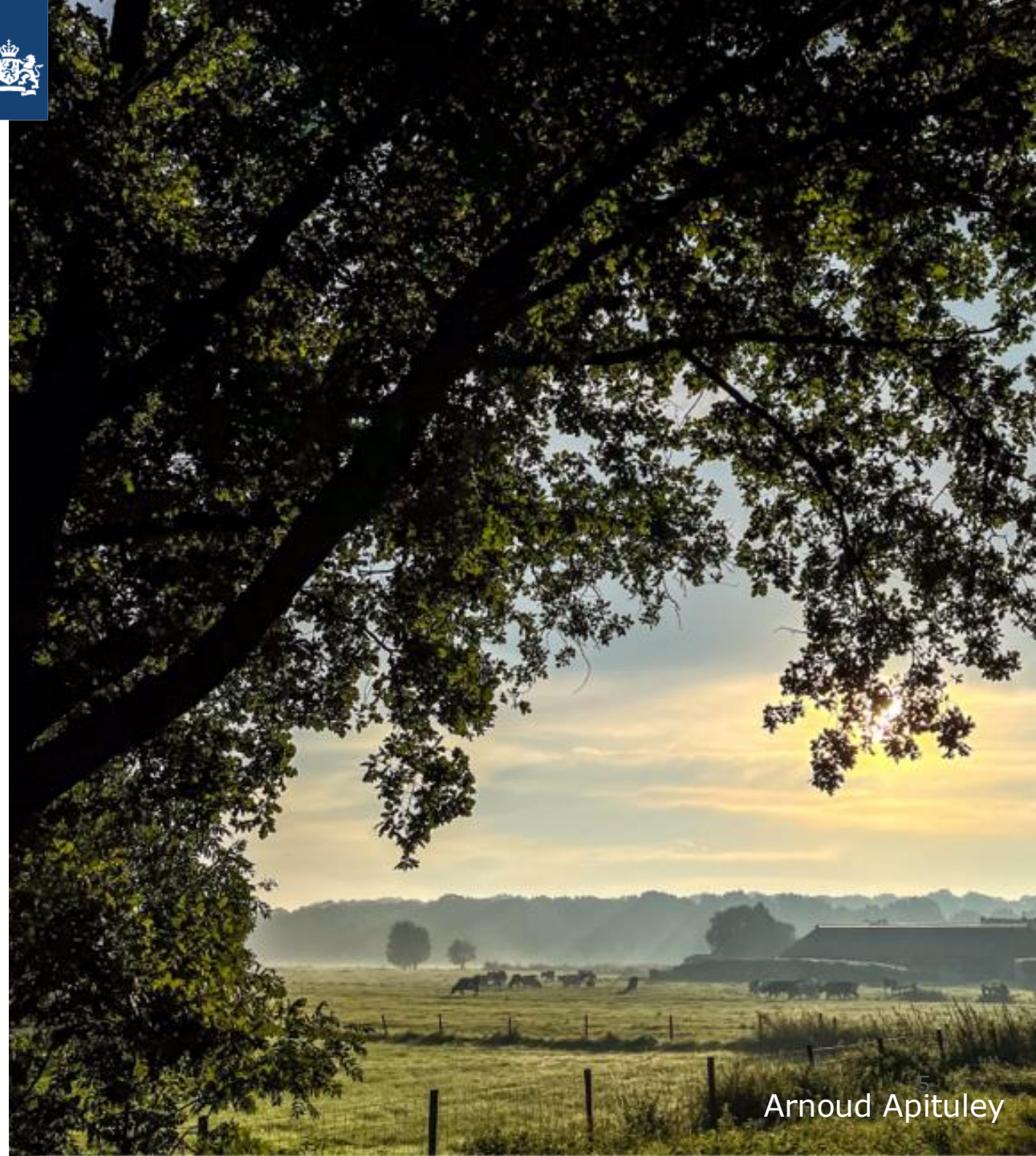
Schedule and outline – Day 2

Time	Item
09:00	Welcome
09:05	Develop your own EDR
10:45	Coffee break
11:00	Develop your own EDR
12:40	Lunch break
13:30	Develop your own EDR
15:10	Coffee break
15:25	Develop your own EDR
17:00	End of day

Step	Item
3	Filtering (time and parameters)
4	Collection metadata
5	(Bonus) List of locations
6	(Bonus) Area query
	Connecting your data

Why OGC EDR?

- Environmental Data Retrieval
- OGC standard
  - Open Geospatial Consortium
  - Since 2016
- Discoverable
- Filtering in space and time
- Multiple datasets
  - Collections and instances
- Used by RODEO E-SOH, WP5, ...



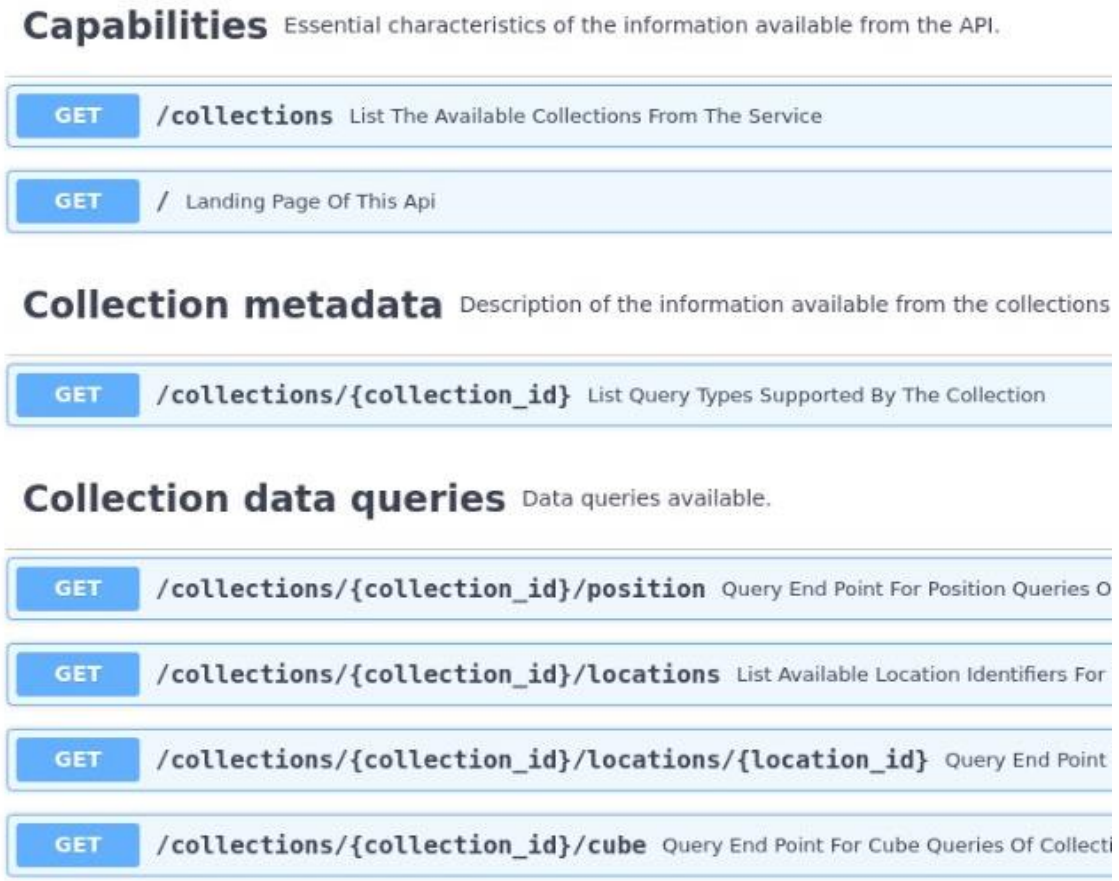
CoverageJSON

- Recommended EDR output
- A format for publishing geotemporal data (on the web)
  - Time series
  - Gridded
- Format: JSON
  - Easier than NetCDF?
  - Better than CSV
- More info:
  - https://covjson.org/


```
{
    "type": "CoverageCollection",
    "domainType": "PointSeries",
    "coverages": [
        {
            "type": "Coverage",
            "domain": {
                "type": "Domain",
                "domainType": "PointSeries",
                "axes": {
                    "x": {
                        "values": [
                            3.275
                        ]
                    },
                    "y": {
                        "values": [
                            51.9978
                        ]
                    },
                    "t": {
                        "values": [
                            "2023-01-22T11:10:00Z"
                        ]
                    }
                }
            },
            "ranges": {
                "dd": {
                    "type": "NdArray",
                    "dataType": "float",
                    "axisNames": [ 3 elements… ],
                    "shape": [ 3 elements… ],
                    "values": [
                        36.2
                    ]
                }
            },
            "eumetnet:locationId": "0-20000-0-06321"
        }
    ],
    "parameters": {
        "dd": {"type": "Parameter"...}
    },
    "referencing": [ 2 elements… ]
}
```

How to use an EDR API?

- Any HTTPS client
  - e.g. from web browsers to Python
  - Insomnia or Postman



Demo of KNMI EDR API Synoptic Observations

- https://tinyurl.com/rodeo-wp5-workshop
- Getting started yourself
  - KNMI Developer Portal
  - KNMI EDR Documentation
  - CovJSON playground

Step 0

Setting up

Code and environment

- Clone the repository and checkout step_0

```
git clone https://github.com/EURODEO/ogc-edr-workshop.git
git checkout step_0
```

- Python3 virtual environment

```
python3 -m venv venv/
source venv/bin/activate
```

- Install dependencies

```
pip3 install pip-tools
pip-sync
```

Check setup

```
python3 data/data.py
```

- NetCDF datastore
  - 10–min synoptic observations of a single day
- data.py is interface between data and EDR
- Should be replaced with your data backend (after workshop...)

```
python3 main.py
```

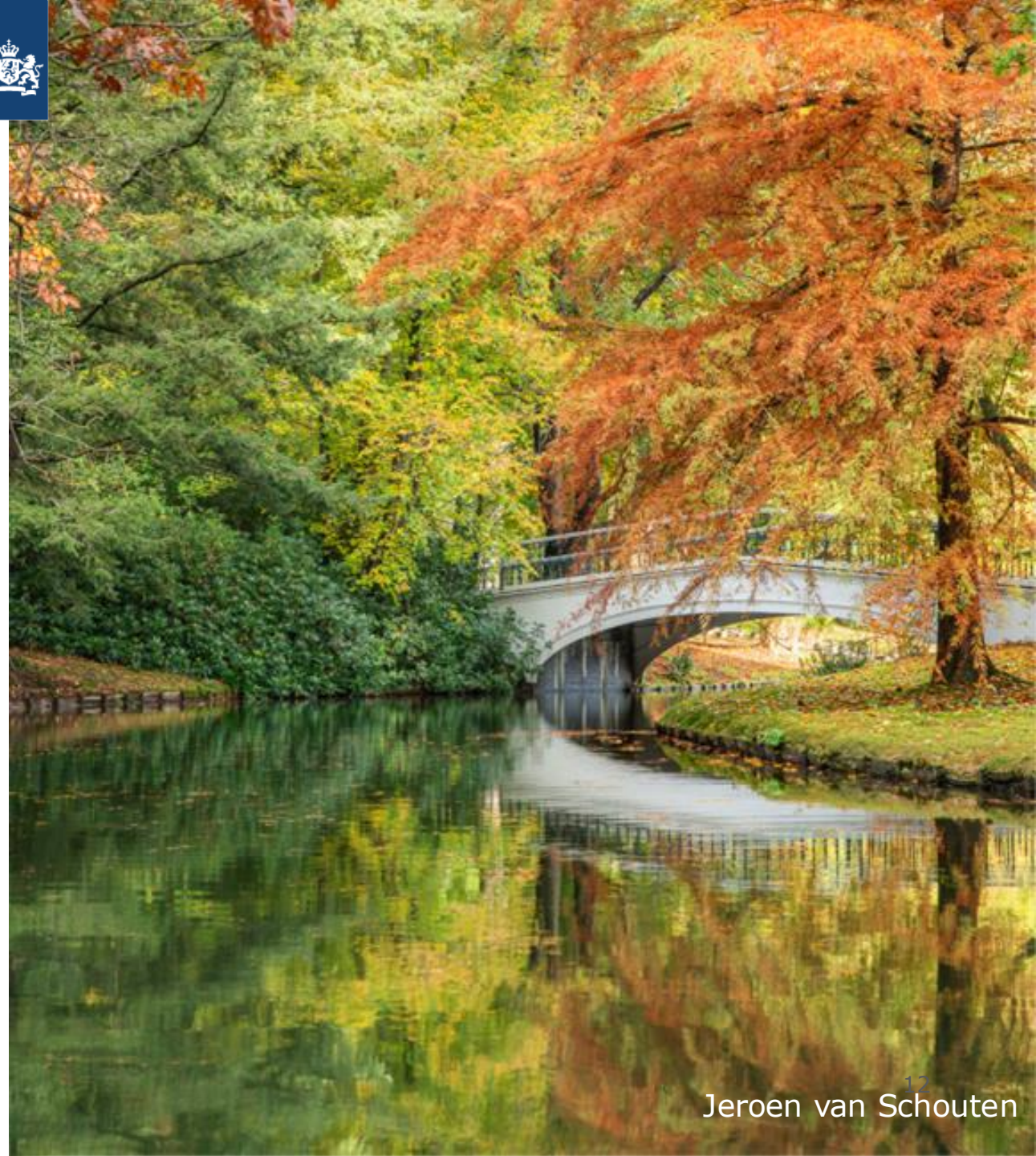
- Starts FastAPI using uvicorn as gateway
- Can be used for step debugging

```
uvicorn main:app --reload
```

- Starts uvicorn with auto reloading
- Test Swagger: http://localhost:8000/docs

How to build an EDR API?

- How ever you want!
  - OGC EDR specification
- Python packages for building EDR APIs
  - Pydantic models for CoverageJSON
  - Pydantic models for EDR
- WP5 Climate EDR:
  - OMSZ EDR API (Hugarian Meteo Service)
- Other examples:
  - EDR-isobaric (MetNorway)
  - RODEO E-SOH EDR API



Step 1

Landing Page

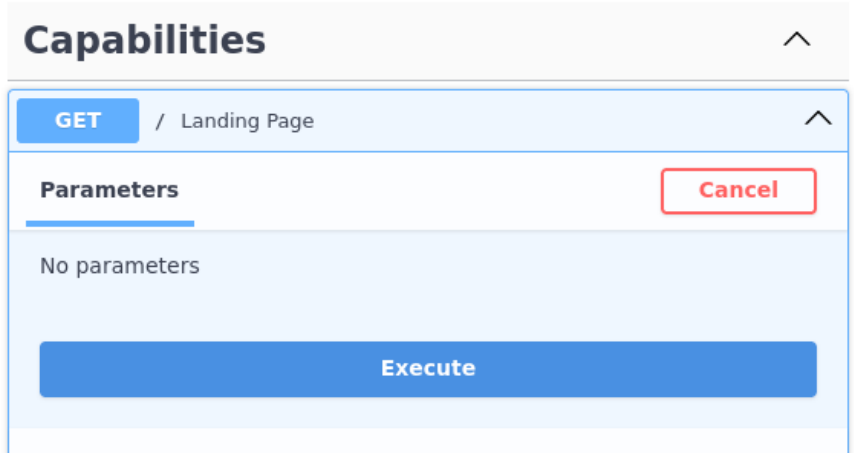
Landing page

- Specification
- Use LandingPageModel from edr_pydantic in main.py

```python
from edr_pydantic.capabilities import LandingPageModel


@app.get(
    "/",
    tags=["Capabilities"],
    response_model=LandingPageModel,
    response_model_exclude_none=True,
)
async def landing_page(request: Request) -> LandingPageModel:
    pass
```

- Test result http://localhost:8000/
- Or via Swagger



Result


```
{
    "title": "EDR tutorial",
    "description": "A simple example EDR implementation",
    "links": [
        {
            "href": "http://localhost:8000/",
            "rel": "self",
            "title": "Landing Page in JSON"
        },
        {
            "href": "http://localhost:8000/docs",
            "rel": "service-desc",
            "title": "API description in HTML"
        },
        {
            "href": "http://localhost:8000/openapi.json",
            "rel": "service-desc",
            "title": "API description in JSON"
        },
```


- Problems?

```
git checkout step_1
```

Step 2

Retrieve data for single location

CoverageJSON

- Specification
- Domain (axes)
  - Standard domains
  - We will use PointSeries
- Parameters
- Ranges


```
{
    "type": "Coverage",
    "domain": {
        "type": "Domain",
        "domainType": "PointSeries",
        "axes": {
            "x": {
                "values": [
                    5.5081
                ]
            },
            "y": {
                "values": [
                    52.4483
                ]
            },
            "t": {
                "values": [
                    "2023-10-20T09:10:00Z"
                ]
            }
        },
        "referencing": [ 2 elements… ]
    },
    "parameters": [ 1 element… ],
    "ranges": {
        "dd": {
            "type": "NdArray",
            "dataType": "float",
            "axisNames": [
                "t",
                "y",
                "x"
            ],
            "shape": [
                1,
                1,
                1
            ],
            "values": [
                76.2
            ]
        }
    },
    "eumetnet:locationId": "0-20000-0-06269"
}
```

Endpoint: /collections/observations/locations/{id}

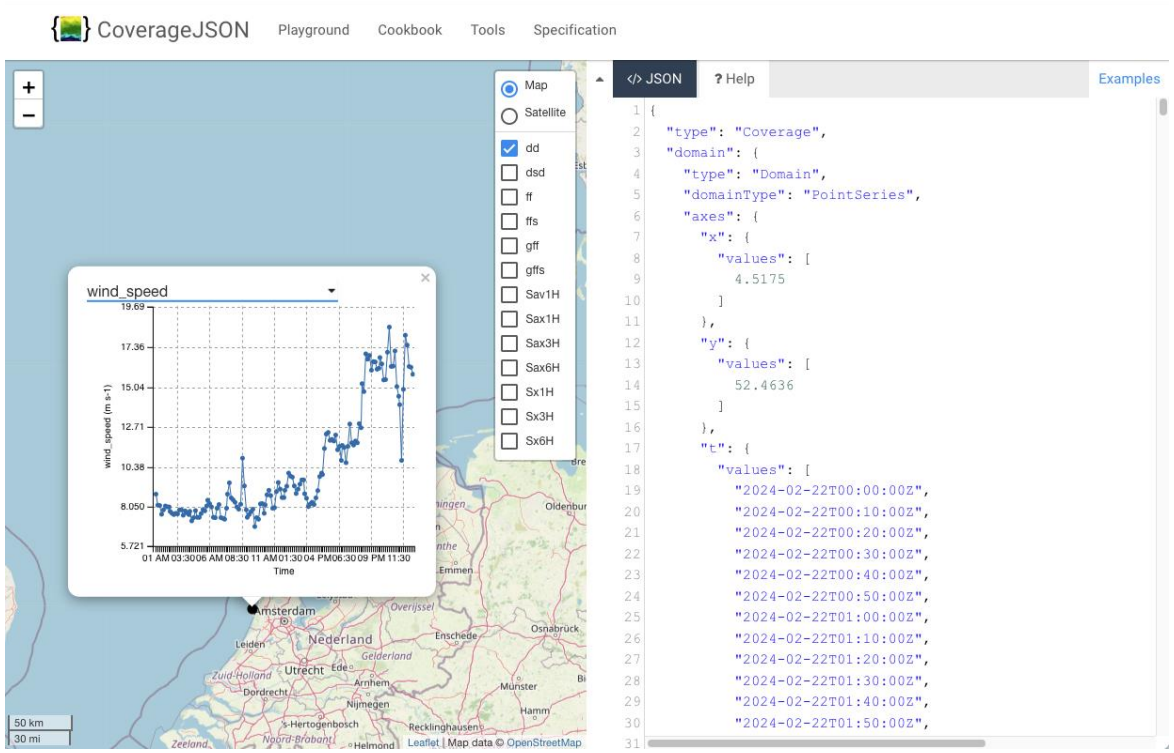
- In `api/observations.py`

```
@router.get(
    "/locations/{location_id}",
    tags=["Collection data queries"],
    response_model=CoverageCollection,
    response_model_exclude_none=True,
    response_class=CoverageJsonResponse,
)
async def get_data_location_id(
    location_id: Annotated[str, Path(example="0-20000-0-06260")],
    parameter_name: Annotated[
        str | None,
        Query(alias="parameter-name", description="Comma seperated list of parameter names.", example="ff, dd"),
    ] = None,
    datetime: Annotated[str | None, Query(example="2024-02-22T01:00:00Z/2024-02-22T02:00:00Z")] = None,
) -> CoverageCollection:
    pass
```

- Use
  - data.get_station()
  - data.get_variables()
  - data.get_data()
- Remarks:
  - No (or minimal) error handling yet
  - No filtering yet

Result

- Use CovJSON playground



- Problems?

```
git checkout step_2
```

Intermezzo 1

- Coverage vs CoverageCollection
- Metadata in database or as code
- RODEO parameter metadata

Coverage vs CoverageCollection


```
{
  "type": "Coverage",
  "domain": {
    "type": "Domain",
    "domainType": "PointSeries",
    "axes": { … 3 … },
    "referencing": [
      {
        "coordinates": [
          "y",
          "x"
        ],
        "system": {
          "type": "GeographicCRS",
          "id": "http://www.opengis.net/def/crs/EPSG/0/4326"
        }
      },
      {
        "coordinates": [
          "t"
        ],
        "system": {
          "type": "TemporalRS",
          "calendar": "Gregorian"
        }
      }
    ]
  },
  "parameters": {
    "dd": { … 5 … },
    "ff": { … 5 … }
  },
  "ranges": {
    "dd": { … 5 … },
    "ff": { … 5 … }
  },
  "eumetnet:locationId": "0-20000-0-06260"
}
```

```
{
  "type": "CoverageCollection",
  "coverages": [
    {
      "type": "Coverage",
      "domain": {
        "type": "Domain",
        "domainType": "PointSeries",
        "axes": { … 3 … }
      },
      "ranges": {
        "dd": { … 5 … },
        "ff": { … 5 … }
      },
      "eumetnet:locationId": "0-20000-0-06260"
    }
  ],
  "parameters": {
    "dd": { … 5 … },
    "ff": { … 5 … }
  },
  "referencing": [
    {
      "coordinates": [
        "y",
        "x"
      ],
      "system": {
        "type": "GeographicCRS",
        "id": "http://www.opengis.net/def/crs/EPSG/0/4326"
      }
    },
    {
      "coordinates": [
        "t"
      ],
      "system": {
        "type": "TemporalRS",
        "calendar": "Gregorian"
      }
    }
  ]
}
```

Metadata in Database or as Code

- Database
  - Already available?
  - Joins/relations
  - Consistency
- Traits of this metadata
  - Mostly static
  - Release cycles
  - Often requested (also in data queries)
- As code
  - Performance
  - Simple version control
  - Easier updates (no DB migrations)

Metadata in Database or as Code

- Database
  - Already available?
  - Joins/relations
  - Consistency
- Traits of this metadata
  - Mostly static
  - Release cycles
  - Often requested (also in data queries)

- As code
  - Performance
  - Simple version control
  - Easier updates (no DB migrations)

- We did not have an existing solution
- Best fit for the use case at KNMI

Parameter metadata (in EDR)


```
"tg": {
    "type": "Parameter",
    "label": {
        "en": "Air Temperature 10 cm Mean"
    },
    "description": {
        "en": "Past 10 minute mean air temperature, at 10 centimeters, in degrees Celcius"
    },
    "observedProperty": {
        "id": "https://vocab.nerc.ac.uk/standard_name/air_temperature/",
        "label": {
            "en": "Air temperature"
        }
    },
    "unit": {
        "label": {
            "en": "degrees celsius"
        },
        "symbol": {
            "value": "°C",
            "type": "https://qudt.org/vocab/unit/DEG_C"
        }
    },
    "measurementType": {
        "method": "mean",
        "duration": "-PT10M"
    },
    "eumetnet:standard_name": "air_temperature",
    "eumetnet:level": 0.1
}
```

CF conventions

More examples

id	label	description	unit	standard_name	method	duration	level
DDVEC	Mean wind direction	Daily vector mean wind direction, representative for 10 m, in degrees. Wind direction has been weighted with wind speed during aggregation.	degree	wind_from_direction	Mean	-P1D	10
FG	Mean windspeed	Daily mean windspeed, representative for 10 m, in m/s	m/s	wind_speed	Mean	-P1D	10
TG	Mean temperature	Daily mean air temperature, measured at 1.5 m, in degrees Celsius	°C	air_temperature	Mean	-P1D	1.5
TN	Minimum temperature	Daily minimum temperature, measured at 1.5 m, in degrees Celsius	°C	air_temperature	Min	-P1D	1.5
TX	Maximum temperature	Daily maximum temperature, measured at 1.5 m, in degrees Celsius	°C	air_temperature	Max	-P1D	1.5
Q	Global radiation	Daily summed global radiation, in J/cm²	J/cm²	TBD	Sum	-P1D	
RH	Precipitation amount	Daily precipitation amount, in mm; -1 for <0.05 mm	mm	precipitation_amount	Sum	-P1D	
PG	Mean sea level pressure	Daily mean sea level pressure , in hPa	hPa	air_pressure_at_mean_sea_level	Mean	-P1D	
UG	Mean relative atmospheric humidity	Daily mean relative atmospheric humidity	%	relative_humidity	Mean	-P1D	

QUDT

Step 3

Filtering (time and parameters)

Filtering (parameters and time)

- Extend `get_data_location_id()`
- Filter `parameter-name`
  - Comma separated list: `dd, ff`
- Filter time
  - start/end (closed interval)
  - ISO8601 string (with Z)

- Remarks:
  - Error handling: What about parameters that don't exist
  - What about parameters that don't exist for the requested station?
- Error responses:
  - 404 for non-existent path
  - 400 for mistake in query parameters
  - 400 (?) for no data (e.g. outside time interval)

Result

- Non-existent location_id:
  - `404: "detail": "Location not found"`
- Mistake in parameter-name:
  - `400: "detail": "The following parameters are not available: {'barbecue_weather'}"`
- Outside datetime:
  - `400: "detail": "No data available"`

- Problems?

```
git checkout step_3
```

Intermezzo 2

- Input query parameters

Input query parameters

- Multiple approaches are possible in Fastapi/Pydantic
- Aim: reusability
- We tried several
- Work in progress

Individual parameters

- Simple
- Not reusable
- Manual type conversion

```python
@router.get(
    "/locations",
    tags=["Collection data queries"],
    response_model=EDRFeatureCollection,
    response_model_exclude_none=True,
    response_class=GeoJsonResponse,
)
async def get_locations(
    bbox: Annotated[str | None, Query(description="Only features that have a geometry "
                                                  "that intersects the bounding box are selected.",
                                      example="5.0,52.0,6.0,52.1")] = None,
    datetime: Annotated[str | None, Query(description="Either a date-time or an interval, open or closed.",
                                          example="2024-02-22T01:00:00Z/2024-02-22T02:00:00Z")] = None,
) -> EDRFeatureCollection:

    # Handle bounding box
    if bbox:
        bbox_values = list(map(lambda x: float(str.strip(x)), bbox.split(",")))
        if len(bbox_values) != 4:
            raise HTTPException(status_code=400, detail="If provided, the bbox should have 4 values")
        left, bottom, right, top = bbox_values
        ...
```

Single Pydantic model

- Less duplication
- Doesn’t work!

Issues with:

- Default values
- Examples
- Mix & match

```python
class LocationsQueryModel(BaseModel):
    bbox: str | None = Field(None, description="Only features that have a geometry"
                                               " that intersects the bounding box are selected.",
                             examples=["5.0,52.0,6.0,52.1"]),
    datetime: str | None = Field(None, description="Either a date-time or an interval, open or closed.",
                                 examples=["2024-02-22T01:00:00Z/2024-02-22T02:00:00Z"])

@router.get(
    "/locations",
    tags=["Collection data queries"],
    response_model=EDRFeatureCollection,
    response_model_exclude_none=True,
    response_class=GeoJsonResponse,
)
async def get_locations(
       query: Annotated[LocationsQueryModel, Query()]
) -> EDRFeatureCollection:

    # Handle bounding box
    if query.bbox:
        bbox_values = list(map(lambda x: float(str.strip(x)), query.bbox.split(",")))
        if len(bbox_values) != 4:
            raise HTTPException(status_code=400, detail="If provided, the bbox should have 4 values")
        left, bottom, right, top = bbox_values
        ...
```

Custom types

- Automatic type conversion
- Complicated
- Reusable

```python
BBox = Tuple[float, float, float, float]

def _validate_str_to_bbox(value: str) -> BBox:
    if type(value) is str:
        value = tuple(float(x) for x in value.split(","))
    if len(value) != 4:
        raise ValueError("bbox expects 4 values")
    return TypeAdapter(BBox).validate_python(value)


BBoxFromString: TypeAlias = Annotated[BBox, PlainValidator(_validate_str_to_bbox,
                                                           json_schema_input_type=str)]

BBoxQueryOptional = Query(
    description="Only features that have a geometry that intersects the bounding box are selected.",
    openapi_examples={"4 numbers": Example(summary="Bounding box - 2 dimensional", value="5.1, 52.0, 5.2,
52.1")},
)

BBoxOptionalParam = Annotated[BBoxFromString | None, BBoxQueryOptional, WithJsonSchema({"type": "string"})]

@router.get(
    "/locations",
    tags=["Collection data queries"],
    response_model=EDRFeatureCollection,
    response_model_exclude_none=True,
    response_class=GeoJsonResponse,
)
async def get_locations(
        bbox: BBoxOptionalParam = None,
        datetime: DatetimeIntervalOptionalParam = None,
) -> EDRFeatureCollection:
```

Step 4

Collection metadata

Collection metadata

- EDR specification
  - Examples in spec
- Parameters in EDR vs Parameters in CovJSON
- Implement /collections/observations
  - Bonus: Implement /collections

Vasko Lozanov

Result


```
{
    "id": "observations",
    "links": [
        {
            "href": "http://localhost:8000/collections/observations",
            "rel": "self"
        }
    ],
    "extent": {
        "spatial": {
            "bbox": [ 1 element… ],
            "crs": "EPSG:4326"
        },
        "temporal": {
            "interval": [
                [
                    "2024-02-22T00:00:00Z",
                    "2024-02-22T23:50:00Z"
                ]
            ],
            "values": [
                "2024-02-22T00:00:00Z/2024-02-22T23:50:00Z"
            ],
            "trs": "datetime"
        }
    },
    "data_queries": {
        "area": [ 1 element… ],
        "locations": {
            "link": {"href": "http://localhost:8000/collections/observations/locations"...}
        }
    },
    "crs": [ 1 element… ],
    "output_formats": [ 1 element… ],
    "parameter_names": {
        "D1H": {
            "type": "Parameter",
            "id": "D1H",
            "label": "Rainfall Duration in last Hour",
            "unit": {"label": "min"...},
            "observedProperty": {"id": "https://vocab.nerc.ac.uk/standard_name/rainfall_duration"...}
        },
        "Q1H": {"id": "Q1H"...},
        "Q24H": {"id": "Q24H"...},
        "R12H": {"id": "R12H"...}
```


- Problems?

```
git checkout step_4
```

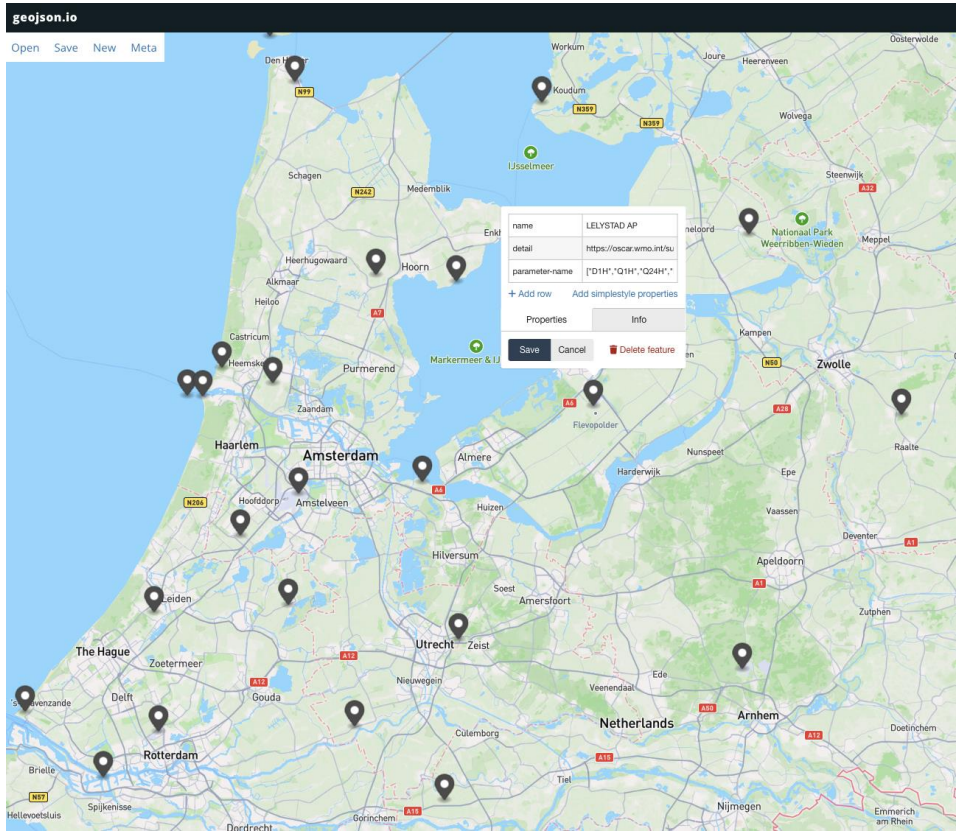
Step 5

List of locations

BONUS

List of locations

- /observations/locations
  - Return GeoJSON
  - geojson.io playground
- Query parameters
  - None in the spec
  - Suggestion: bbox, datetime and parameter-name
- EDRFeatureCollection
  - GeoJSON FeatureCollection with parameters



Step 6

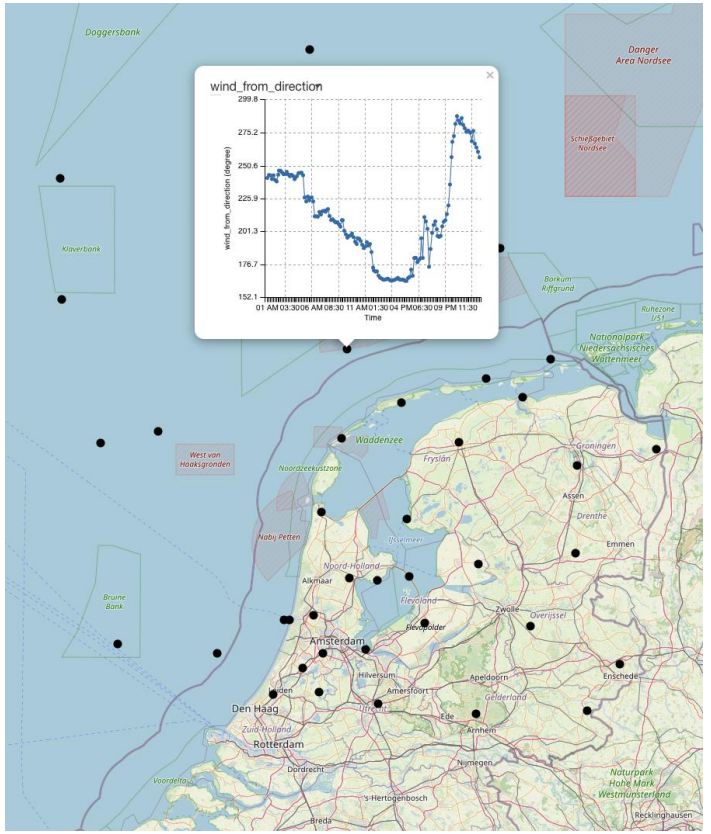
/area query

BONUS

/area query

- CoverageCollection
- Addition of station identifier to Coverage: "eumetnet:locationId": station.wsi
- Hint: calculate stations in polygon:

```
from shapely import geometry, wkt
poly = wkt.loads(coords)
stations_in_polygon = [s for s in get_stations()
                       if geometry.Point(s.longitude,
s.latitude).within(poly)]
```



Aggregator

Short intro

Work in Progress: EDR Aggregator

- Unify multiple EDRs into one
- (Proposed) functionality:
  - Output conversion: NetCDF, CSV
  - Caching
  - Queries spanning collections
- Benefits for NMHSs:
  - Simplify EDR implementation and deployment
  - No direct public access required
  - Implementation agnostic
- Benefits for users:
  - One endpoint
  - One query
- Work in progress: To be released as OSS