Resource Management
Operators support

EOX IT Services https://eox.at
EOFARM
EOfarm P.C. http://eofarm.com

EOEPCA Operators demo 2022-09-20

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.
Overview

- Team
- Resource Management Architecture
- Resource Catalogue
- Data Access Service & Ingestion
- User Workspace
- Summary
Resource Management Architecture

- Heavily based on **Python geospatial stack/ecosystem**
- Kubernetes mainly using helm charts
- Automation via flux
- System and per Workspace deployments
Resource Catalog Overview

- Discover/Search, Harvest, Distributed/Federated Search (user and system catalog)
- Metadata
- Standards
  - ISO 19115/19139
  - OGC CSW 2.0.2 / 3.0.0
    - Core Profile
    - ISO Application Profile
    - OGC OpenSearch Geo/Time, EO Profiles
  - OGC API - Records - Part 1: Core
  - STAC API (1.0.0-rc1), STAC Item (1.0.0)
Resource Catalog Technologies

- **pycsw**
  - Python OGC Reference Implementation Server
  - Multiple metadata formats (ISO, DC etc.)
  - Flexible backends (RDBMS)

- **OWSLib**
  - Swiss army knife Python client library for OGC Web Services and APIs

- **pygeometra**
  - Metadata composition

- **pygeoapi**
  - Python OGC Server Reference Implementation
  - Evolving OGC API standards
Resource Catalogue Technologies

- PostGIS
  - PostgreSQL Spatial extension
  - OGC SFSQL
  - Metadata Store for pycsw, EOxServer

- Elasticsearch (in Roadmap)
  - Full Text Search Engine
  - Highly Distributed
Resource Catalogue Configuration

- Helm Chart available: https://github.com/EOEPCA/helm-charts/releases/tag/rm-resource-catalogue-1.1.0
- pycsw
  - Can be scaled vertically and horizontally since it is a stateless service, just add instances.
  - Populate the catalogue through:
    - pycsw-admin CLI: https://docs.pycsw.org/en/latest/administration.html#loading-records
    - OGC CSW-T:
Resource Catalogue Configuration

- PostgreSQL
  - Can be highly optimized for millions of metadata records using spatial index and full text search support.
  - Can be scaled vertically
  - Horizontal scaling can be achieved in various scenarios: (i) Read Scalability through pgpool or pgbouncer by using PostgreSQL replication (ii) Read-Write Scalability using FDW and Partitioning or tools like Postgres-XL and Citus.
  - Replacing the default helm chart with a high-availability one, like Bitnami: [https://github.com/bitnami/charts/tree/master/bitnami/postgresql-ha](https://github.com/bitnami/charts/tree/master/bitnami/postgresql-ha)
Resource Catalogue Examples

Based on pycsw, a Python OGC CSW server implementation

Collections
OpenAPI
Swagger
JSON

Conformance
CSW 3.0.0
CSW 2.0.2
OpenSearch
STAC API
OAI-PMH
SRU

Powered by pycsw 3.0.dev0

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```xml
<ows:Ows12 Fence Keywords="catalogue"></ows:Ows12>
<ows:Ows12 Fence Keywords="discovery"></ows:Ows12>
<ows:Ows12 Fence Keywords="metadata"></ows:Ows12>
<ows:Ows12 Fence Keywords="ISO19115"></ows:Ows12>
<ows:Ows12 Fence Keywords="OGC"></ows:Ows12>
<ows:Ows12 Fence Keywords="CSW"></ows:Ows12>
<ows:Ows12 Fence Keywords="Version"></ows:Ows12>
<ows:Ows12 Fence Keywords="Provider"></ows:Ows12>
<ows:Ows12 Fence Keywords="Name"></ows:Ows12>
<ows:Ows12 Fence Keywords="Organization"></ows:Ows12>
<ows:Ows12 Fence Keywords="simple"></ows:Ows12>
<ows:Ows12 Fence Keywords="Contact"></ows:Ows12>
<ows:Ows12 Fence Keywords="Name"></ows:Ows12>
<ows:Ows12 Fence Keywords="Position"></ows:Ows12>
<ows:Ows12 Fence Keywords="Title"></ows:Ows12>
<ows:Ows12 Fence Keywords="Phone"></ows:Ows12>
<ows:Ows12 Fence Keywords="Address"></ows:Ows12>
<ows:Ows12 Fence Keywords="Mailing"></ows:Ows12>
<ows:Ows12 Fence Keywords="City"></ows:Ows12>
<ows:Ows12 Fence Keywords="Administrative"></ows:Ows12>
<ows:Ows12 Fence Keywords="Postal"></ows:Ows12>
```
Resource Catalogue Examples
Resource Catalogue Examples
Resource Catalogue Example
Resource Catalogue Examples

```python
bbox_query = BBox([37.8, 23.4, 38.8, 24.5])
bbox_query = BBox([37.13, 37.9, 15.1])

begin = PropertyIsGreaterThanOrEqualTo(propertyName='apiso:TempExtent_begin', literal='2021-04-02 00:00:00')
end = PropertyIsLessThanOrEqualTo(propertyName='apiso:TempExtent_end', literal='2021-04-03 00:00:00')
cloud = PropertyIsLessThanOrEqualTo(propertyName='apiso:CloudCover', literal='20')

filter_list = [
    And(
        bbox_query,  # bounding box
        begin, end,  # start and end date
        cloud        # cloud
    )
]

The filter is then applied to the GetRecords request and results are shown:

csw.getrecords2(constraints=filter_list, outputschema='http://www.isotc211.org/2005/gmd')
csw.results

{'matches': 1, 'returned': 1, 'nextrecord': 0}

selected_record = list(csw.records)[0]

for rec in csw.records:
    print(F'Identifier: {csw.records[rec].identifier}

Identifier: S2B_MSIL1C_20210402T095029_N0300_R079_T335V8_20210402T121737.SAFE
type: dataset
title: S2B_MSIL1C_20210402T095029_N0300_R079_T335V8_20210402T121737.SAFE
```

Finally we demonstrate how to show the record footprint on a map, using the Folium Python library:

```python
# folium.Map(locations=[38, 20], zoom_start=6, tiles='OpenStreetMap')
folium.Rectangle(bounds=[[float(rec.bbox.minx), float(rec.bbox.miny)], [float(rec.bbox.maxx), float(rec.bbox.maxy)]])
```
Resource Catalogue Examples
Resource Catalogue Examples
Data Access Service Overview

- **View Server** (*EOxServer* as core)
- Flexible WMS viewing interface
  - raster expressions, outlines, masking, property filtering
- Performance optimized WM(T)S cached viewing
**Data Access Service** Overview

- **Download interfaces**
  - WCS + EO Application Profile
  - DSEO
- **OGC OpenSearch catalog interface**
  - Geo Extension
  - Time Extension
  - EO Extension
  - CQL filtering
Data Access Ingestion

- Harvesting from System catalog
- Metadata enrichment
- Export to STAC Item
- Registration into Data Access and Resource Catalog
Data Access Service & Ingestion Components

- Database - PostgreSQL
- Renderer - EOxServer
- Cache - Mapcache
- Registrar - Python + EOxServer
- Harvester - Python
- Redis queues
- Webclient based on EOxC
- Elasticsearch/Fluentd/Kibana logging stack
Data Access Ingestion

Scheduler → Harvester → Registrar → Seeder

System Catalog → Harvester

Request tiles → Seeder

Cache bucket → Seeder

STAC Item → Preprocessor

Data bucket → Preprocessor

STAC Item → Preprocessor

Catalogue

Metadata enrichment → Catalogue

Data Access DB → Catalogue

Renderer

ATOM/JSON → Search

Search → System Catalog

Harvester

STAC Item → Harvester

STAC Item → Harvester

STAC Item → Harvester

STAC Item → Harvester

STAC Item → Harvester

STAC Item → Harvester
Data Access Ingestion - Workspace

Workspace API -> Harvester

Harvester -> Registrar

Registrar -> Seeder

Seeder -> Data Access DB

Data Access DB -> Catalogue

Catalogue -> Renderer

Renderer -> Cache Bucket

Cache Bucket -> Seeding Bucket

Seeding Bucket -> Data Access DB

Data Access DB -> Preprocessor

Preprocessor -> STAC Item

STAC Item -> Data Bucket

Data Bucket -> Stage-out Bucket

Stage-out Bucket -> Harvest

Harvest -> STAC Item

STAC Item -> STAC Item

STAC Item -> STAC Item

STAC Item -> STAC Item
Harvesting configuration

- Several harvesters can be configured

```json
harvesters:
  - name: Creodias-Opensearch
    resource:
      url: https://finder.creodias.eu/resto/api/collections/Sentinel2/describe.xml
      type: OpenSearch
      format_config:
        type: 'application/json'
        property_mapping:
          start_datetime: 'startDate'
          end_datetime: 'completionDate'
          productIdentifier: 'productIdentifier'

    query:
      time:
        property: sensed
        begin: 2019-09-10T00:00:00Z
        end: 2019-09-11T00:00:00Z
        collection: null
        bbox: 14.9,47.7,16.4,48.7
    filter: {}
    postprocess:
      - type: harvester_eoepca.postprocess.CREODIASOpenSearchSentinel2Postprocessor
    queue: register
```
Harvesting configuration

- **resource**: defines from where shall be harvested
- **filter**: filter harvested items using CQL2-JSON
- **postprocess**:
  - apply postprocessing (Python script)
  - transformation to STAC if not already
  - metadata enrichment

```python
class MyPostprocessor(Postprocessor):
    def postprocess(self, item: dict) -> dict:
        ...
```

- **queue**: output queue
Data Access configuration

- View Server can be used with rigid type system
  - Products
    - main Record item
  - Coverages
    - sub-item of Products (rel OGC Coverges)
  - Browses
    - visual representation of a Product
    - either materialized or virtual
  - Masks
  - Collections
Data Access configuration
Data Access configuration

- Helm chart values
- global values:
  - productTypes
  - collections
- and registrar specific values
Data Access ProductType configuration

- Product type definition:

```json
productTypes:
  - name: S2MSI1C
    filter:
      s2:product_type: S2MSI1C
    collections:
      - S2L1C
    metadata_assets: []
    coverages:
      S2L1C_B01:
        assets:
          - B01
      ...
```
Data Access ProductType configuration

● Browse setup:

```python
...  
defaultBrowse: TRUE_COLOR
browses:
  TRUE_COLOR:
    asset: visual
    red:
      expression: B04
      range: [0, 4000]
      nodata: 0
    green:
      expression: B03
      range: [0, 4000]
      nodata: 0
    blue:
      expression: B02
      range: [0, 4000]
      nodata: 0
  NDVI:
    grey:
      expression: (B08-B04)/(B08+B04)
      range: [-1, 1]
    masks:
      clouds:
        validity: false
...```

Data Access ProductType configuration

- **filter**: when registering only matching products use this product type
- **collections**: place this product in the collection
- **coverages**: all coverages to be created for the product and the asset name to find it in the STAC Item
- **browses**:
  - the asset name of the browse and/or
  - the red, green, blue or greyscale sources or expressions
  - value ranges to stretch
Data Access Collection configuration

- Collection definition:

```py
collections:
S2L1C:

    product_types:
    - S2MSI1C

    coverage_types:
    - S2L1C_B01
    - S2L1C_B02
    - S2L1C_B03
    - ...
```
Data Access Collection configuration

- `product_types`: product types that can be inserted into this collection
- `coverage_types`: coverage types that can be inserted into this collection
Registration configuration

```python
config:
    defaultBackends:
    - path: registrar_pycsw.backend.ItemBackend
      kwargs:
        repository_database_uri: postgresql://postgres:mypass@resource-catalogue-db/pycsw
        ows_url: https://data-access.develop.eoepca.org/ows
      defaultSuccessQueue: seed_queue
```
disableDefaultRoute: true
routes:
  items:
    path: registrar.route.stac.ItemRoute
    queue: register_queue
    replace: true
backends:
  - path: "registrar.backend.eoxserver.ItemBackend"
    kwargs:
      instance_base_path: "/var/www/pvs/dev"
      instance_name: "pvs_instance"
      product_types: []
      auto_create_product_types: True
  - path: "registrar_pycsw.backend.ItemBackend"
    kwargs:
      repository_database_uri: "postgresql://postgres:mypass@resource-catalogue-db/pycsw"
      ows_url: "https://data-access.{{ workspace_name }}.develop.eoepca.org/ows"
      public_s3_url: "https://cf2.cloudferro.com:8080/{projectid}:{bucket}"
Registration configuration

```
collections:
  path: registrar.route.stac.CollectionRoute
  queue: register_collection_queue
  replace: true
  backends:
    - path: registrar_pycsw.backend.CollectionBackend
      kwargs:
        repository_database_uri: postgresql://postgres:mypass@resource-catalogue-db/pycsw

ades:
  path: registrar.route.json.JSONRoute
  queue: register_ades_queue
  replace: true
  backends:
    - path: registrar_pycsw.backend.ADESBackend
      kwargs:
        repository_database_uri: postgresql://postgres:mypass@resource-catalogue-db/pycsw

application:
  path: registrar.route.json.JSONRoute
  queue: register_application_queue
  replace: true
  backends:
    - path: registrar_pycsw.backend.CMLBackend
      kwargs:
        repository_database_uri: "postgresql://postgres:mypass@resource-catalogue-db/pycsw"
        ows_url: "https://data-access.{{ workspace_name }}.develop.eoepca.org/ows"
        public_s3_url: "https://cf2.cloudferro.com:8080/{projectid}:{bucket}"
```
Registration configuration

• “Routes”
  ○ Listens on a specific queue
  ○ Parses to a specific type (e.g.: STAC Item)
  ○ Connected to backends (e.g.: pycsw)
  ○ Modes (register, replace, deregister)
  ○ Configurable output queue
  ○ Default Route (additional backends)
Seeder Configuration

- Re-uses cache configuration
- `minzoom` / `maxzoom`
Renderer Configuration

- Re-uses global ProductType setup
- EO-WCS:
  - Collections -> advertised DatasetSeries
  - Products -> non-advertised DatasetSeries
  - Coverages -> non-advertised Coverages
  - Typical flow:
    - GetCapabilities -> DatasetSeries
    - DescribeEOCoverageSet -> Coverages
    - GetCoverage
Renderer Configuration

- **EO-WMS:**
  - Collections: advertised layer structure
  - Products: non-advertised layer structure
  - Coverage: non-advertised layer structure

- **Basic layers:**
  - "<name>" -> default rendering
  - "<name>__outlines" -> rendered footprint
  - "<name>__outlined" -> default rendering with imposed footprint
Renderer Configuration

- Collection/Products:
  - "<name>__<browse>": specific visualization
    - e.g: “S2L1C__NDVI”
  - "<name>__<mask>": polygon mask
    - e.g: “S2L2A__clouds”
  - "<name>__masked_<mask>": default visualization with masks subtracted
Renderer Configuration

- **Coverages:**
  - `dim_bands` parameter for RGB composition
    - e.g: `...&dim_bands=B08,B04,B03&...`
  - `dim_ranges` parameter
    - e.g: `...&dim_range=0 4000&...`
Renderer Configuration

- Additional parameters:
  - **cql**: Allows to pass CQL filters
    - e.g: ...&cql=cloudCover < 10&...
  - **sortBy**: Allows to sort rendered images
    - e.g: ...&sortBy=cloudCover A&...
Renderer Configuration

- OpenSearch
- Automatically exposed as set of endpoints
- `/opensearch` -> root OSDD
- `/opensearch/<format>` -> collection level search
- `/opensearch/collections/<collection>` -> collection OSDD
- `/opensearch/collections/<collection>/<format>` -> product level search
Cache/Client Configuration

```json
layers:
  - id: S2L1C
    title: Sentinel-2 Level 1C True Color
    abstract: Sentinel-2 Level 2A True Color
    displayColor: '#eb3700'
    grids:
      - name: WGS84
        zoom: 13
    parentLayer: S2L1C
  search:
    histogramBinCount: 15
    histogramThreshold: 80
  - id: S2L1C__TRUE_COLOR
    title: Sentinel-2 Level 1C True Color
    abstract: Sentinel-2 Level 2A True Color
    grids:
      - name: WGS84
        zoom: 13
    parentLayer: S2L1C
```
Cache/Client Configuration

- `global.layers`
- **Metadata:** `id`, `title`, `abstract`
- `displayColor`: used in the client
- `grids`: what grids shall be cached and until which `zoomlevel`
- `parentLayer`: to build layer hierarchies
- `search`: search settings
- `global.storage.cache`:
  - `storage configuration`
  - `cache bucket credentials as secret`
Cache Configuration

- **Endpoints:**
  - `/cache/` -> main entry point
  - `/cache/wmts` -> WMTS
  - `/cache/wms` -> WMS endpoint
User Workspace

- **EOEPCA Model**
  - Object storage bucket
  - Data Access Service
  - Resource Catalog with federated system catalog
  - Deployment scenario(s) up to operator

- **Workspace API** based on FastAPI
  - create, delete, etc.
  - using “Bucket” claims via K8s CRD & K8s Secret
  - register data & processes

- **Bucket operator** for OpenStack
User Workspace

```
default

GET /probe Probe

POST /workspaces Create Workspace

GET /workspaces/{workspace_name} Get Workspace

DELETE /workspaces/{workspace_name} Delete Workspace

PATCH /workspaces/{workspace_name} Patch Workspace

POST /workspaces/{workspace_name}/redeploy Redeploy Workspace

POST /workspaces/{workspace_name}/register Register

POST /workspaces/{workspace_name}/deregister Deregister

POST /workspaces/{workspace_name}/register-collection Register Collection

POST /workspaces/{workspace_name}/create-container-registry-repository Create Container Registry Repository

POST /grant-access-to-container-registry-repository Grant Container Registry Access View
```
User Workspace Customization

- **Configuration options in detail here**
- Central configs:
  - `workspaceChartsConfigMap`
    - References a config map containing HelmReleases which define the user workspace functionality
  - `s3Endpoint, s3Region`
    - Bucket server location
  - `harborUrl, harborUsername, harborPassword`
    - Access for workspace api to harbor (container registry)
Harbor (container registry)

- Installed via **helm release**
- Central configs:
  - harborAdminPassword
    Change in interface after first install and configure in workspace api
  - storageClass
    Set an appropriate storage class for all harbor services
Summary

- Open Standards (OGC, ISO)
- Free and Open Source (FOSS)
  - Core components
  - Reference implementation
  - Geospatial (FOSS4G)
- Best of breed Python geospatial components
- Download, configure, and run
  - [https://github.com/EOEPCA/ eoepca](https://github.com/EOEPCA/ eoepca)
- Open communities
- Open to contributions
Thanks for your attention!