Python Reader for ADF Data Description

Dr. Aleksandar Jelenak

Allotrope Connect Workshop
September 2021

The HDF Group

1

Outline

The HDF Group

- *h5ld*, Python package for reading linked data in HDF5
- Working with ADF Data Description using *h5ld*
- Reading Data Description from ADF files in Amazon S3
- Storing ADF Data Description graph for reuse

h5ld: HDF5 Linked Data

h5ld: HDF5 Linked Data Reader

The HDF Group

- Open-source Python package from The HDF Group.
- Community code contributions are welcome.
- Dependencies: Python 3.7 or later; *h5py*, *rdflib packages;* and HDF5 library v1.10.6 or later.
- Goal is to provide independent readers for HDF5-based formats with Linked Data
- Currently supported: Allotrope Data Format (ADF).
- Command-line and programmatic interface.
- On GitHub: https://github.com/HDFGroup/h5ld

Working with ADF Data Description Using *h5ld*

Command-Line Interface

The HDF Group

Example:

```
python –m h5ld –f json–ld –o output.json example.adf
```

- Read linked data from the ADF file and write it out in the JSON-LD format to *output.json* file.
- Output RDF formats: Turtle, JSON-LD, N-Quads, TriG.
- Omitting an output file will print the RDF content out for ingest by another command-line tool.

- Full description:

```
python –m h5ld ––help
```

Programmatic Interface

The HDF Group

```
In [3]: with h5py.File("../../../Allotrope/examples/R180735_PQTEST_small.adf") as f:
            g = AllotropeDF(f).get_ld()

In [4]: g

Out[4]: <Graph identifier=Nc5781f4de4d445a59725c047f267fd48 (<class 'rdflib.graph.ConjunctiveGraph'
        >)>

In [5]: len(g)

Out[5]: 894
```

Allotrope RDF Namespaces

The HDF Group

```
In [6]: namespaces = dict((pre, str(iri)) for pre, iri in g.namespaces())
        pprint(namespaces)
```

```
{'adf-dc': 'http://purl.allotrope.org/ontologies/datacube#',
 'adf-dc-hdf5': 'http://purl.allotrope.org/ontologies/datacube-hdf-map#',
 'adf-dd': 'http://purl.allotrope.org/ontologies/datadescription#',
 'adf-dp': 'http://purl.allotrope.org/ontologies/datapackage#',
 'af-a': 'http://purl.allotrope.org/ontologies/audit#',
 'af-c': 'http://purl.allotrope.org/ontologies/common#',
 'af-e': 'http://purl.allotrope.org/ontologies/equipment#',
 'af-m': 'http://purl.allotrope.org/ontologies/material#',
 'af-p': 'http://purl.allotrope.org/ontologies/process#',
 'af-q': 'http://purl.allotrope.org/ontologies/quality#',
 'af-r': 'http://purl.allotrope.org/ontologies/result#',
 'af-s': 'http://purl.allotrope.org/shape/',
 'af-sh': 'http://purl.allotrope.org/ontologies/shapes/',
 'af-x': 'http://purl.allotrope.org/ontologies/property#',
 'afs-hdf5': 'http://purl.allotrope.org/shapes/hdf#',
 'dct': 'http://purl.org/dc/terms/',
 'hdf5': 'http://purl.allotrope.org/ontologies/hdf5/1.8#',
 'obo': 'http://purl.obolibrary.org/obo/',
 'owl': 'http://www.w3.org/2002/07/owl#',
 'pav': 'http://purl.org/pav/',
 'qb': 'http://purl.org/linked-data/cube#',
 'qudt': 'http://qudt.org/schema/qudt#',
 'qudt-quantity': 'http://qudt.org/vocab/quantity#',
 'qudt-unit': 'http://qudt.org/vocab/unit#',
 'rdf': 'http://www.w3.org/1999/02/22-rdf-syntax-ns#',
 'rdfs': 'http://www.w3.org/2000/01/rdf-schema#',
 'sh': 'http://www.w3.org/ns/shacl#',
 'skos': 'http://www.w3.org/2004/02/skos/core#',
 'xml': 'http://www.w3.org/XML/1998/namespace',
 'xsd': 'http://www.w3.org/2001/XMLSchema#'}
```

ADF Data Description Graphs

The HDF Group

```
In [7]: list(g.contexts())

Out[7]: [<Graph identifier=urn:x-adf:TechnicalGraph (<class 'rdflib.graph.Graph'>)>,
         <Graph identifier=urn:x-adf:UserDataGraph (<class 'rdflib.graph.Graph'>)>,
         <Graph identifier=urn:x-adf:MetaGraph (<class 'rdflib.graph.Graph'>)>]
```

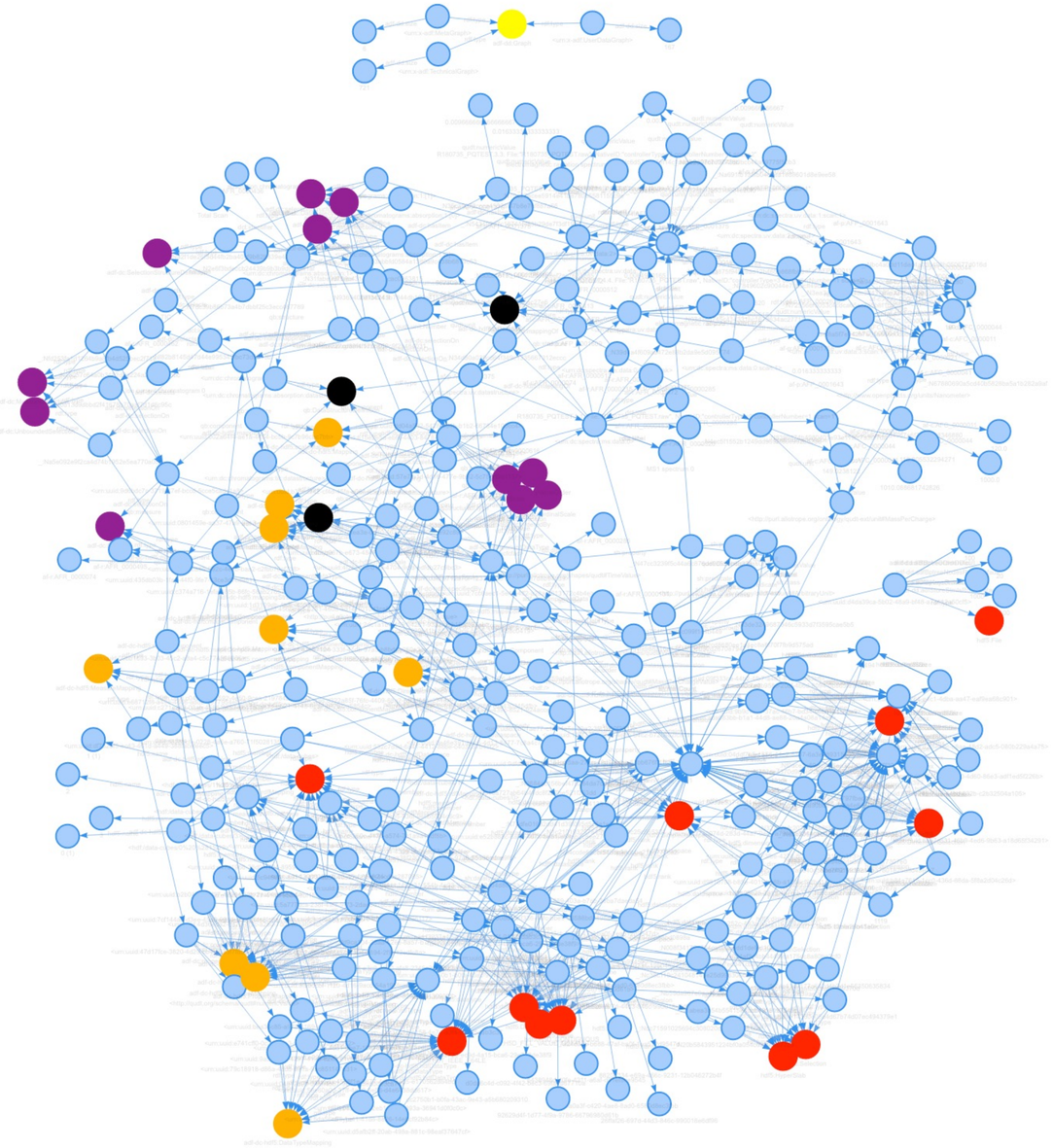
Visualization with *kglab* and PyVis

The HDF Group

```
kg = kglab.KnowledgeGraph(
    import_graph=g,
    namespaces=namespaces)
```

```
VIS_STYLE = {
    "hdf5": {
        "color": "red",
        "size": 30
    },
    "adf-dc-hdf5":{
        "color": "orange",
        "size": 30
    },
    "adf-dc": {
        "color": "purple",
        "size": 30
    },
    "adf-dp": {
        "color": "green",
        "size": 30
    },
    "adf-dd": {
        "color": "yellow",
        "size": 30
    },
    "qb": {
        "color": "black",
        "size": 30
    }
}

subgraph = kglab.SubgraphTensor(kg)
pyvis_graph = subgraph.build_pyvis_graph(notebook=True, style=VIS_STYLE)
pyvis_graph.force_atlas_2based(damping=2.0)
pyvis_graph.show("graph-vis.html")
```



SPARQL with *kglab*

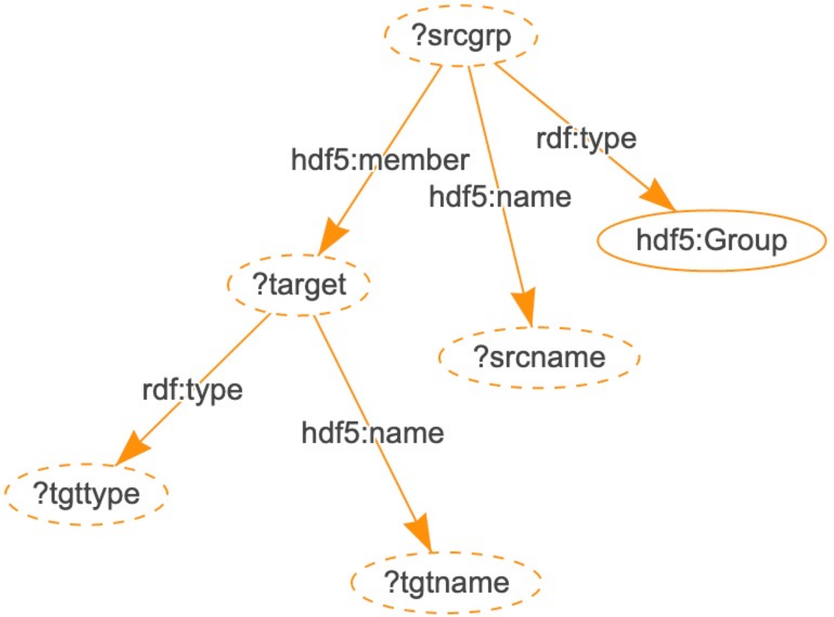
The HDF Group

In [14]:
```
sparql = SPARQL_PRFX + """
SELECT ?srcgrp ?srcname ?target ?tgttype ?tgtname
{
    ?srcgrp a hdf5:Group ;
            hdf5:member ?target ;
            hdf5:name ?srcname .
    ?target hdf5:name ?tgtname .
    ?target a ?tgttype .
}
"""
kg.query_as_df(sparql)
```

Out[14]:

	srcgrp	srcname	target	tgttype
0	<hdf:/>		<hdf:/data-cubes>	hdf5:Group
1	<urn:uuid:5c948658-8915-4473-ad88-40a27a23bd54>	measures	<urn:uuid:0be48729-ec9d-4a15-bca6-29e778de38f9>	hdf5:Dataset
2	<hdf:/data-cubes/1>	1	<urn:uuid:23c3067c-260c-4438-a574-7c1a9a44df8b>	hdf5:Group
3	<hdf:/data-cubes/1>	1	<urn:uuid:aa38f230-7b34-4425-97a6-55696adce9f6>	hdf5:Group
4	<urn:uuid:1306633b-7b02-45cf-85ef-74cd0c287085>	scales	<urn:uuid:6438948d-d0fe-4211-a6af-d01ef9ce9545>	hdf5:Dataset
5	<hdf:/data-cubes>	data-cubes	<hdf:/data-cubes/2>	hdf5:Group
6	<hdf:/data-cubes>	data-cubes	<hdf:/data-cubes/0%20%281%29>	hdf5:Group
7	<hdf:/data-cubes>	data-cubes	<hdf:/data-cubes/1>	hdf5:Group
8	<hdf:/data-cubes>	data-cubes	<hdf:/data-cubes/1%20%281%29>	hdf5:Group
9	<hdf:/data-cubes>	data-cubes	<hdf:/data-cubes/0>	hdf5:Group
			<hdf:/data-/3>	hdf5:Group

```
query_graph = kg.visualize_query(sparql, notebook=True)
query_graph.force_atlas_2based()
query_graph.show("query-vis.html")
```



Reading Data Description from ADF files in Amazon S3

Local File

The HDF Group

- This is the baseline, traditional file access.
- Using an ADF file with dummy Data Description.
- File: QueryTest.adf; size: 63.89 MB; number of RDF triples: 588,082
- Operations: (1) Read HDF5 datasets holding RDF triple data; (2) Decode RDF subject, predicate, and object information; (3) Form N-Quads statements; (4) Create an *rdflib* graph object by parsing the N-Quads statements.

```
In [23]: %%time

         with h5py.File("../../../Allotrope/examples/QueryTest.adf") as f:
             g = AllotropeDF(f).get_ld()

         CPU times: user 1min 7s, sys: 547 ms, total: 1min 7s
         Wall time: 1min 8s
```

S3 Object with HDF5 ROS3 VFD

The HDF Group

- Read-only S3 Virtual File Driver (VFD) is available with the HDF5 library (build time option).
- Its performance is greatly influenced by the creation properties of the HDF5 file and its datasets.

```
In [29]: %%time

         s3url = ("https://s3.us-west-2.amazonaws.com/"
                  "hdf5.sample/data/Allotrope/QueryTest.adf")
         with h5py.File(s3url, mode="r", driver="ros3",
                        aws_region=b"us-west-2",
                        secret_id=os.environ["aws_access_key_id"].encode(),
                        secret_key=os.environ["aws_secret_access_key"].encode()) as f:
             g_vfd = AllotropeDF(f).get_ld()

         CPU times: user 1min 8s, sys: 1.5 s, total: 1min 10s
         Wall time: 4min 48s
```

S3 Object with *fsspec* as HDF5 VFL

The HDF Group

- *h5py* can use a Python file-like object as an HDF5 virtual file layer
- *fsspec* is a Python package with support for many local, remote and embedded file systems and object stores.

```
In [16]: s3fs = fsspec.filesystem(protocol='s3')

In [24]: %%time

         with s3fs.open("s3://hdf5.sample/data/Allotrope/QueryTest.adf",
                        mode="rb") as adf_s3:
             with h5py.File(adf_s3, mode="r") as f:
                 g_s3 = AllotropeDF(f).get_ld()

         CPU times: user 1min 11s, sys: 2.23 s, total: 1min 13s
         Wall time: 2min 10s
```

Downloaded S3 Object as Local File

The HDF Group

- Download from object store, then use as a local file.
- Specific to ADF files with reasonable download times.

```
In [30]: %%time

         s3fs.get("s3://hdf5.sample/data/Allotrope/QueryTest.adf", "./")
         with h5py.File("QueryTest.adf", mode="r") as f:
             g_dwnld = AllotropeDF(f).get_ld()

         CPU times: user 1min 7s, sys: 866 ms, total: 1min 8s
         Wall time: 1min 13s
```

HSDS with HDF cloud-native file format

The HDF Group

- The original ADF file was ingested into an HDF Highly Scalable Data Service (HSDS) system.
- HSDS ingest converts an HDF5 file into the HDF cloud-native format and creates HDF5 datasets with larger chunks.
- This access method requires the *h5pyd* package, and a user account on the HSDS instance.

```
In [33]: %%time

         with h5pyd.File("/home/ajelenak/Allotrope/hsds/QueryTest.adf", mode="r") as f:
             g_hsds = AllotropeDF(f).get_ld()

         CPU times: user 1min 7s, sys: 591 ms, total: 1min 7s
         Wall time: 1min 13s
```

HSDS with Original ADF File

The HDF Group

- HSDS can also work with the original ADF file.
- HSDS ingest in this case only derives information about file locations of the HDF5 dataset chunks.

```
In [36]: %%time

         with h5pyd.File("/home/ajelenak/Allotrope/file/QueryTest.adf", mode="r") as f:
             g_hsds = AllotropeDF(f).get_ld()

         CPU times: user 1min 7s, sys: 626 ms, total: 1min 8s
         Wall time: 1min 23s
```

Storing ADF Data Description Graph for Reuse

Parquet File

The HDF Group

- *kglab* supports storing and loading graph data in Parquet.
- Parquet file size: 31.4 MB.
- Downside: The Parquet file holds only N-Triple statements.

```
In [39]: %%time

         kg_s3.save_parquet("s3://hdf5.sample/Trash/graph.parquet")

         CPU times: user 14.4 s, sys: 538 ms, total: 14.9 s
         Wall time: 37.9 s
```

```
In [44]: %%time

         kg_s3 = kglab.KnowledgeGraph()
         kg_s3.load_parquet("s3://hdf5.sample/Trash/graph.parquet")

         CPU times: user 2min 26s, sys: 17.8 s, total: 2min 44s
         Wall time: 2min 52s

Out[44]: <kglab.kglab.KnowledgeGraph at 0x16f64a610>
```

Compressed N-Quads Text File

The HDF Group

- Text file with the ADF file's Data Description N-Quads statements.
- File size: 20.1 MB.

In [41]:
```
%%time

nquads = io.BytesIO()
g_s3.serialize(nquads, format="nquads")
nquads.seek(0, io.SEEK_SET)

with s3fs.open("s3://hdf5.sample/Trash/QueryTest.nquads.deflate", mode="wb") as outf:
    outf.write(
        zlib.compress(
            nquads.getbuffer(),
            zlib.Z_DEFAULT_COMPRESSION
        )
    )

nquads.close()
```

```
CPU times: user 17.4 s, sys: 366 ms, total: 17 7 -
Wall time: 34.9 s
```

In [46]:
```
%%time

g_s3 = rdflib.ConjunctiveGraph()
with s3fs.open("s3://hdf5.sample/Trash/QueryTest.nquads.deflate", mode="rb") as outf:
    buf = io.BytesIO(
        zlib.decompress(
            outf.read()
        )
    )
    g_s3.parse(buf, format="nquads")
    buf.close()
```

```
CPU times: user 44.8 s, sys: 520 ms, total: 45.3 s
Wall time: 48.7 s
```

THANK YOU!

Questions & Comments?

ajelenak@hdfgroup.org

help@hdfgroup.org