A FAIR Digital Object-Based Data Lake Architecture

Hendrik Nolte, Piotr Kasprzak, Julian Kunkel, Philipp Wieder, Ramin Yahyapour
Motivation
Data Lake

• Typical requirements from researchers within such a distributed institute are:
  - Central place to store all data
    • Unstructured data with millions of files and PB of volume
  - Search capabilities over all collected data sets
  - Globally enforced governance
  - High-Level user interface
  - Easy way to share data and developed methods
  - Reproducible processing
    • Built-in provenance auditing

Often research data management on HPC systems is far from this
Motivation
Data Silos

Introduction: Data Lake

- With cheap storage costs, people promote the concept of data lake
  - Combines data from many sources and of any type
  - Allows for conducting future analysis and not miss any opportunity

- Attributes of the data lake
  - **Dump** everything: all time all data: raw sources and processed data
    - Decide during analysis which data is important, e.g., no “schema“ until read
  - **Dive** in anywhere: enable users across multiple business units to
    - Refine, explore and enrich data on their terms
  - **Fishing** for data, i.e., flexible access
    - Shared infrastructure supports various patterns
    - Usage: Batch, interactive, online, search
Concept of a Data Lake
General User Workflow on a Data Lake
Data Lake Functionalities

- Data repository for all raw, processed and associated data
- Start remote compute jobs, i.e. perform data analytic tasks
  - Automatic provenance auditing
  - Automatic artifact management
- Management of project/experiment/task metadata
- Search and inspection of data
- Sharing of data and developed methods
  - Portable packages of performed workflows
- Enforced governance
- Guided workflows
  - Python-Notebooks
  - Web-UI’s
Reference Architecture: Zone Architecture I

- Zone Architecture most common, but elusive
- Differences:
  - User Groups
  - Number and focus of Zones
- Fundamental idea:
  - Different Zones contain data in different degrees of processing

Giebler et al. “A Zone Reference Model for Enterprise-Grade Data Lake Management” (2020) IEEE Xplore
Reference Architecture: Zone Architecture II

- Each zone has a different backend, i.e. storage, metadata and processing
- Not consistency safe
- Complete pipeline not possible in self service for users
- No overarching data lineage
- And no reproducibility
- Hard to scale access rights
- No global view

Giebler et al. "A Zone Reference Model for Enterprise-Grade Data Lake Management" (2020) IEEE Xplore
Data Lake Based on FAIR Digital Objects

- Unified and flat space
- Data refinement based on the type
  - Raw Date or Processed Date
- Data type needs to be registered
  - Define metadata attributes
- Governance based on types and their typed attributes
- User does not interact with backend services directly
- Instead user call functions of these DO’s
  - Guarantees a global consistent state
Schematic Architecture
Ingestion

DO Definition:

birthday_present

recipient:
address:
content:
date_of_delivery:
gifted_by:

Images from Flaticon.com
Processing
```python
import data_lake

conn = data_lake.Connection(username='', password='', url='data-lake.gwdg.de')

job = data_lake.Job()
job.comment = 'Beispiel Job'
job.container_name = 'bart-DeepDeepLearning'
job.compute.append('cd /program/hpc-statustagung-rep/
job.compute.append('python3 example_python_script.py')

job.git = [
    {
        "uri" : "git@gitlab.gwdg.de:hnoltel/hpc-statustagung-rep.git",
        "type" : "build",
        "bsh" : "nkdir /program/output/"
    }
]

job.job_name = 'Output-Test'
job.data_category = 'kspace'
job.env_var = {
    "OUTPUT_FOLDER" : '/program/output/',
    "NUM_REPETITIONS" : '2'
}

job.output_dir = '/program/output'

res = job.submit(conn)

res = data_lake.show_jobs(conn, 'DataOutput', 'output_file_1.txt', format='GitRepositories,GitCommits,ContainerName,DataOutput,Comment,NUM_REPETITIONS,CreateDate')

print(res.content.decode('utf-8'))
```

<table>
<thead>
<tr>
<th>GitRepositories</th>
<th>GitCommits</th>
<th>ContainerName</th>
<th>DataOutput</th>
<th>Comment</th>
<th>NUM_REPETITIONS</th>
<th>CreateDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpc-statustagung-rep.</td>
<td>a1877ca42de7e8c4bc27e6d2265dd07627588115</td>
<td>bart-DeepDeepLearning.sif</td>
<td>output_file_1.txt</td>
<td>Beispiel Job</td>
<td>2</td>
<td>2021-04-10T15:24:363800</td>
</tr>
<tr>
<td>hpc-statustagung-rep.</td>
<td>a1877ca42de7e8c4bc27e6d2265dd07627588115</td>
<td>bart-ddl.sif</td>
<td>output_file_1.txt</td>
<td>Beispiel Job</td>
<td>3</td>
<td>2021-04-06T22:36:43.716102</td>
</tr>
<tr>
<td>hpc-statustagung-rep.</td>
<td>a1877ca42de7e8c4bc27e6d2265dd07627588115</td>
<td>bart-ddl.sif</td>
<td>output_file_1.txt</td>
<td>Beispiel Job 3</td>
<td>2</td>
<td>2021-04-07T13:11:48.496271</td>
</tr>
</tbody>
</table>
Remote Job Execution
HPCSerA
Outlook: Automated Processing

- Users can create, configure and share those processing steps
  - Published under specific API
  - Possible in self-service
- Only few adaptions are necessary
  - Change the explicit input data definition to conditions of the input type and, if necessary, define conditions for the typed attributes
- By introducing conditions for the execution of a task, automation using a workflow engine is/seems possible
Conclusion

- Typical requirements from researchers within such a distributed institute are:
  - Central place to store all data
    - Particularly unstructured research data
  - Globally enforced governance
  - High-Level user interface
  - Easy way to share data and developed methods
  - Search capabilities over all collected data sets
  - Reproducible processing
    - Built-in provenance auditing

- Our Data Lake can fulfill these requirements with the following set of features:
  - WebApp offers this single central contact point
  - Data-Governances can be enforced on the data lake layer
  - Web-Interfaces or libraries
  - Access to data and methods can be managed by users
  - The data catalog offers search capabilities over all entities
  - Built-in reproducibility and provenance auditing using Job Manifests and HPCSerA

Our data lake abstracts the complicated storage architecture and combines a research data management system with an HPC cluster.
Motivation

Data Warehouse

- Data Source A
- Data Source B
- Data Source C

ETL → Data Warehouse

- Dimension Table: time (time_key, day, day_of_the_week, month, quarter, year)
- Dimension Table: branch (branch_key, branch_name, branch_type)
- Dimension Table: location (location_key, location, street, city, state_or_province, country)
- Dimension Table: supplier
- Dimension Table: item
- Dimension Table: item_name
- Dimension Table: brand
- Dimension Table: type
- Dimension Table: supplier_type

Sales Fact Table:
- time_key
- item_key
- branch_key
- location_key
- unit_sold
- dollars_sold

https://www.javatpoint.com/data-warehouse-what-is-star-schema
HPCSerA
Security Aspects

![Diagram showing security aspects of HPCSerA]

- **Manage Tokens**: User manages tokens.
- **2FA**: User uses 2FA for authentication.
- **WebUI**: Access to the Web UI.
- **Auth app**: Authentication application.
- **Store**: Token store.
- **Validate**: Token validation.
- **HPCSerA**: Core component.
- **HPC agent**: Agent component.
- **Batch system**: Batch system component.
- **CLI**: Command line interface.
- **Client**: Client component.

### Table: Roles and Descriptions

<table>
<thead>
<tr>
<th>Role Number</th>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GET_JobStatus</td>
<td>Client can retrieve information about a submitted job. Used by client/agent to update the job status</td>
</tr>
<tr>
<td>2</td>
<td>UPDATE_JobStatus</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>GET_Job</td>
<td>Endpoint used by the agent to retrieve job information.</td>
</tr>
<tr>
<td>4</td>
<td>POST_Code</td>
<td>Client to ingest new code to the HPC system.</td>
</tr>
<tr>
<td>5</td>
<td>GET_Code</td>
<td>Agent pulls new code. Might be necessary to run new job.</td>
</tr>
<tr>
<td>6</td>
<td>POST_Job</td>
<td>Client triggers parameterized job.</td>
</tr>
<tr>
<td>7</td>
<td>UPDATE_Job</td>
<td>Client updates already triggered job.</td>
</tr>
<tr>
<td>8</td>
<td>DELETE_Job</td>
<td>Client deletes already triggered job.</td>
</tr>
</tbody>
</table>

*In scope of HPCSerA development*