Data Lake Ready to Use

User Manual, version 1.0

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Bulgaria, Austria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Greece, Hungary, Ireland, Italy, Lithuania, Latvia, Poland, Portugal, Romania, Slovenia, Spain, Sweden, France, Netherlands, Belgium, Luxembourg, Slovakia, Norway, Türkiye, Republic of North Macedonia, Iceland, Montenegro, Serbia under grant agreement No 101101903.

This page intentionally blank

Summary

Table of Figures

Acronyms and Terminology

AI	Artificial Intelligence	Field of study in computer science that develops and studies intelligent machines.
API	Application Programming Interface	A way for two or more computer programs or components to communicate with each other. It is a type of software interface, offering a service to other pieces of software. In contrast to a User Interface, which connects a computer to a person, an application programming interface connects computers or pieces of software to each other.
DL	Data Lake	System or repository of data stored in its natural/raw format, usually object blobs or files
HPC	High-Performance Computing	Technology that uses clusters of powerful processors, working in parallel, to process massive multi-dimensional datasets (big data) and solve complex problems at extremely high speeds.
HPDA	High Performance Data Analytics	Process of quickly examining extremely large data sets to find insights. This is done by using the parallel processing of HPC to run powerful analytic software.
IoT	Internet of Things	Term describing devices with sensors, processing ability, SW and other technologies that connect and exchange data with other devices and systems over the Internet or other communications networks.
REST	Representational state transfer	Software Architectural Style created to guide the design and development of architecture for the World Wide Web. REST defines a set of constraints for how the architecture of a distributed, Internet-scale hypermedia system should behave.
SLAs	Service Level Agreements	Agreements between a service provider and a customer. Aspects of the service – quality, availability, responsibilities – are agreed between the service provider and the service user.
TUI	Text-based User Interface	Space where interactions between humans and machines occur. The goal of this interaction is to allow effective operation and control of the machine from the human end, while the machine simultaneously feeds back information that aids the operators' decision-making process.
VM	Virtual Machine	Virtualization or emulation of a computer system. Virtual machines are based on computer architectures and provide the functionality of a physical computer.

1. Introduction

Within the framework of the European Project EuroCC2, the national competence center EuroCC Italy developed the Data Lake Ready to Use application. This initiative aims to equip small and medium enterprises, as well as public infrastructure, with a service for easy storage and analytics on heterogeneous and multi-format raw and processed data. The service provides a fast deploy system for the core components of Data Lake technology and its interface. These components are chosen from free and open-source products, and both the installation scripts and this user manual will be released under an open license. This document aims to describe the product's requirements and functionalities, as well as providing instructions on installation and usage.

The service is designed to deliver a customizable Data Lake infrastructure according to the needs of the requesting user and integrate it with High Performance Computing (HPC) systems, enabling high-performance analytics on extensive datasets, increasing throughput and/or reducing time to solution.

The infrastructure is configured in dual parallel file system/object storage mode, benefitting from the speed of a parallel file system and from the ease of access and querying of an object storage system.

2. EuroCC: Objectives

EuroCC 2 is a project which works to identify and address the skills gaps in the European High Performance Computing (HPC) ecosystem and coordinate cooperation across Europe to ensure a consistent skills base. Mission of EuroCC is to improve the technological readiness of Europe. In particular, the role of EuroCC2 is to establish and run a network of more than 30 NCCs across the EuroHPC Participating States. The NCCs act as single points of access in each country between stakeholders and national and EuroHPC systems. They operate on a regional and national level to liaise with local communities, in particular SMEs, map HPC competencies and facilitate access to European HPC resources for users from the private and public sector.

Figure 1: NCCs across Europe

EuroCC2 delivers training, interacts with industry, develops competence mapping and communication materials and activities, and supports the adoption of HPC services in other related fields, such as quantum computing, artificial intelligence (AI), high performance data analytics (HPDA) to expand the HPC user base.

3. Service Description

The Datalake service primarily facilitates data management and querying. A REST API exposes the commands for data transfer and querying to the DL. This process employs S3 APIs for direct data transfer to the storage system and allows the data to be accessed in different modes from HPC. The service implements also a metadata database, assisting in cataloging multi-dimensional and multi-format data, capturing details, and can be customized as needed. The user can interact with the service either directly via the REST API (curl commands) or via commands provided by the DL-TUI (text-user interface). Finally, results from HPC can be retrieved both via API and DL-TUI.

4. Architectural Diagram

Here is represented the graphical diagram of the service described in the previous sections.

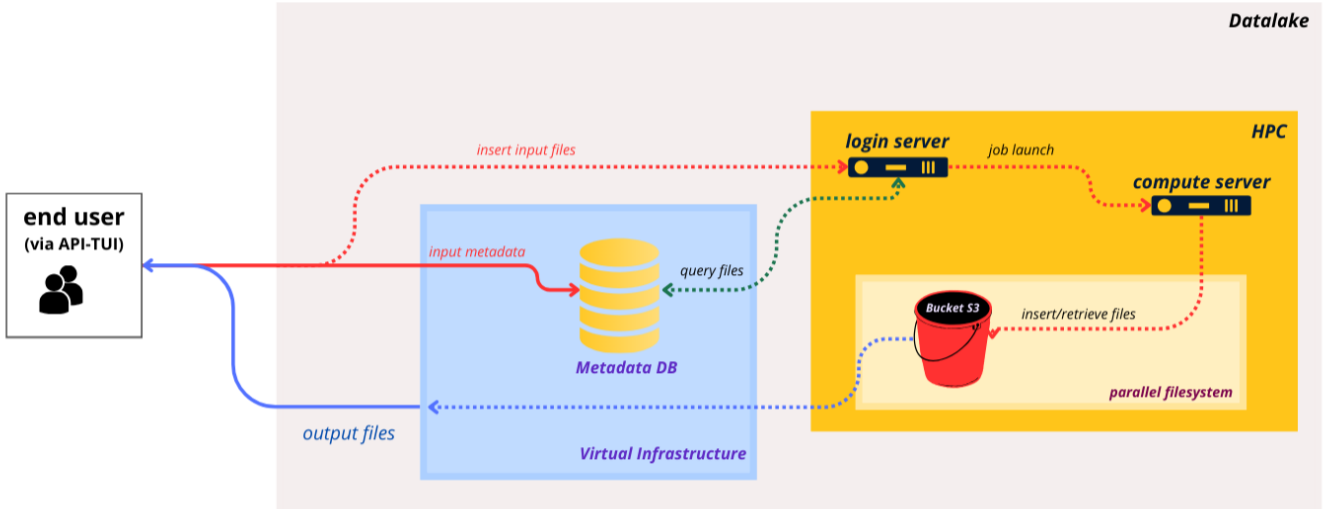

Figure 2: Graphical Representation of service Architecture

- **End User:** It represents individuals or systems that interface with the Datalake service.
- **API/TUI:** The primary access points for the end user, allowing data input/query/processing/retrieval by serving as an interface between the user and internal service components.
- **Virtual Infrastructure:** The Virtual Machine, instantiated by the self-deployment scripts, hosts the metadata database and ensures secure communication with the HPC environment.
- **Metadata DB:** A non-relational database which stores data in a non-tabular form. They are used when large quantities of complex and diverse data need to be organized and perform faster than traditional SQL databases because a query does not have to view several tables in order to deliver an answer. As a tool to implement this service, MongoDB was chosen.
- **HPC**: Infrastructure that provides High Performance Compute capability. It is composed by: login server(s), compute servers, and the parallel filesystem.
- **Parallel Filesystem**: An advanced storage mechanism designed for high-speed/ low latency I/O, playing a pivotal role in the overall data management within the service.
- **Object Storage System:** A storage solution interfaced with the parallel filesystem to provide an easy access mechanism through the S3 standard protocol.

4.1 Interfaces

In this section the various interfaces are presented in more detail.

4.1.1 API Overview

The DataLake API, developed using the Connexion framework, is designed to streamline and secure the interaction between users and DL environment. Below is reported an overview of its primary features and functionalities.

Features

- *Access Management:* The API ensures controlled access to resources, allowing users to interact with data stored in the S3 data lake based on their permissions.
- *User Authentication:* It implements robust authentication mechanisms to verify user identities, ensuring secure access to data and resources.

Functionalities

RESTful action	Purpose	Usage
GET (Download)	Facilitates the download of a single file from the data lake	Enables users to access and retrieve data efficiently
POST (Upload)	Supports the uploading of a single file to the S3 datalake, accompanied by a corresponding MongoDB entry	Simplifies the process of adding new data to the lake, ensuring a synchronized update of both file and metadata
POST (Query and Process)	Allows users to send queries and process files to HPC systems	Useful for complex data processing and analysis tasks requiring HPC resources
PATCH	Updates a MongoDB entry by adding new metadata, without replacing the entire entry	Enhances data annotation and cataloging by allowing incremental metadata updates
PUT	Updates (replaces) an existing MongoDB entry and the corresponding file in S3	Ensures data accuracy by allowing modifications and updates to existing data sets
DELETE	Deletes a single file from the S3 data lake along with its associated entry in MongoDB	Ideal for removing outdated or unnecessary data, ensuring data relevancy and storage optimization
GET (Browse)	Shows the contents of the data lake, allowing for filtering based on the files metadata.	Allows users to quickly search the data lake for relevant content

4.1.2 DL-TUI

The Text User Interface is a vital component for querying the data lake and running processing scripts on matching files.

The library consists of 3 executables:

- dl_tui_hpc:

  Purpose: The client version is intended to be run on the machine with direct access to the data lake files.

  Function: Performs querying and processing operations.

- dl_tui_server:

  Purpose: The server version is intended to be run on a cloud machine with access to an HPC infrastructure running Slurm.

  Function: Parses user input and launches a Slurm job on HPC, which calls the client version.

- dl_tui:

  Purpose: The following version is intended to be to provide the user an interface to interact with DL.

  Function: Translate RESTful commands in command-line interface ones.

5. Self-Deployment Guide

The following provides a detailed description of how to use the script provided to deploy the service.

5.1 Prerequisites

The following prerequisites must be met before continuing:

- The machine used for the deployment must possess an allowed and identifable IP address from which both the HPC and Cloud resources can be accessed via SSH;
- Recommended Requirements (VM): 8 vCPUs, 32 GB RAM, 1 TB of storage;
- The Cloud infrastructure must support setting up virtual machines via the OpenStack CLI;
- The HPC storage must be configured in dual S3/parallel filesystem mode; in particular the S3 object storage must be accessible via a URL of the type <ENDPOINT_URL>/<S3_BUCKET_NAME>, and the files on the parallel filesystem must be available at a pre-determined path <PREFIX_PATH>/<FILENAME>;
- The system administrator setting up the service must have both access and secret keys to the S3 bucket on HPC, and must have a valid credential, ssh permanent key and compute hours budget;
- The system administrator setting up the service must have a Linux terminal type with Python 3. This guide will show commands using CentOS as a reference, ***beware to adapt them with respect to your machine.***
- Other requirements: python3-venv, python3-pip, git.

5.2 Step-by-step Installation & Deploy

To install the DL infrastructure follow the steps described below (the playbook was validated on the Cineca Cloud/HPC infrastructure with a CentOS 8 VM).

1. Clone the repository containing the Ansible codebase:

```
git clone https://github.com/Eurocc-Italy/dl_deploy.git
```

2. Create a Python virtual environment (either with conda or venv) and activate said environment:

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

3. Run:

```
cd dl-deploy/
pip install -r ./requirements.txt
```

4. Copy /ansible/hosts.example in /ansible/inventory/hosts;

```
cp /ansible/hosts.example /ansible/inventory/hosts
```

5. Set python-openstackclient environment credential:

- Following instruction at https://docs.openstack.org/pythonopenstackclient/latest/cli/authentication.html
- if necessary, in the file `/ansible/roles/openstack-infra/defaults/main.yml` set **`dl_cloud`** to point to your cloud credentials file;

- Note: the roles doesn't read OS_* environment variables, it needs a `clouds.yml` in `~/.config/openstack` or `/etc/openstack` directory

6. Set in `/ansible/inventory/group_vars/all.yaml` the following variables as necessary for your specific configuration:
   - `openstack_external_network`
   - `dl_cloud_image`

7. Set in `/ansible/roles/openstack-infra/defaults/main.yml` the following variables, needed to set up Datalake's Security Group in Openstack Cloud:
   - `login_nodes` (IP range of the login nodes on HPC)
   - `compute_nodes` (IP range of the compute nodes on HPC)
   - `setup_ip` (IP range of the machine(s) carrying out deploy and administration)
   - `user_ips` (IP/range of the machine(s) for user accounts)

8. Set in `/ansible/roles/dlaas/defaults/main.yml` the following variable to the path of the SSH key for access to HPC: `hpc_operator_ssh_prv_key`

9. Edit `/ansible/inventory/hosts` changing `ansible_user` variable to match the operator's HPC facilities account.

10. Edit `/ansible/roles/dlaas/vars/s3.yml` and configure S3 settings.

11. Update `dtaas-digitaltwinasaservice/ansible/roles/dlaas/vars/certbot.yml` with a valid email account. This email is used to register an account with Let's Encrypt: `certbot_email_account: user@domain.TLD`

12. Launch the Ansible Playbook from ansible directory:

```
cd ./ansible
ansible-playbook -i inventory/ site.yml
```

6. Self-Deployment Guide

6.1 API USAGE

In this section examples are provided in order to launch API calls from the local machine via curl commands.

1. **Upload data via POST request:**

```
 curl -k -X 'POST' \
# Specifies the request type as POST
  'http://131.175.205.87/v1/upload' \
# URL to which the request is made
  -H 'accept: text/plain' \
# Sets the expected type of response to plain text
  -H 'Authorization: Bearer [Token] \
# Includes the authorization token for access control
  -H 'Content-Type: multipart/form-data' \
# Indicates that the data being sent is in multipart form,suitable for uploading files
  -F 'file=@"/home/username/datalake_deploy/UPLOAD/FILES/file.jpg";type=image/jpeg'
# file to upload with its MIME type specified
  -F ‘json_data=@”/home/username/datalake_deploy/UPLOAD/METADATA/file.json”; type=
application/json’
# metadata file for the upload, with its MIME type
```

2. **Download a file using a GET request:**

```
curl -k -X 'GET' \
# Specifies the request type as GET
  'http://131.175.205.87/v1/download?file_name=file.jpg' \
# URL from which the file will be downloaded
  -H 'Authorization: Bearer [Token] \
# Includes the authorization token for access control
  -H 'accept: application/octet-stream' > ../DOWNLOAD/ file.jpg
# Sets the expected response type as a binary stream and redirects the downloaded file
to the specified path
```

3. **Delete a file using the DELETE request:**

```
curl -k -X 'DELETE' \
# Specifies the request type as DELETE
  'http://131.175.205.87/v1/delete?file_name=file.jpg' \
# URL to which the request is made
  -H 'Authorization: Bearer [Token] \
# Includes the authorization token for access control
  -H 'accept: */*'
# Accepts any type of response, usually indicating success or failure
```

4. **Make a query using a POST request:**

```
curl -k -X 'POST' \
# Specifies the request type as POST; it sends data for processing, including a
configuration JSON, a Python script and a query file
  'http://131.175.205.87/v1/query_and_process' \
# URL to which the request is made
  -H 'accept: text/plain' \
# Sets the expected type of response to plain text
  -H 'Authorization: Bearer [Token] \
# Includes the authorization token for access control
  -H 'Content-Type: multipart/form-data' \
# Indicates that the data being sent is in multipart form, suitable for uploading
files
  -F 'config_json=' \
# Placeholder for sending a configuration in JSON format
  -F 'python_file=@"/home/username/datalake_deploy/QUERY/script.py"' \
# Python script to be processed
  -F 'query_file=@"/home/username/datalake_deploy/QUERY/query.txt";type=text/plain' |
cut -d " " -f 5 | tee 7-download-query-results/id
# Processes the response to extract specific data and save it
```

5. **Replace a file using a PUT request:**

```
curl -k -X 'PUT' \
# Specifies the request type as PUT
  'http://vm.ip.address/v1/replace' \
#  URL to which the request is made
  -H 'accept: text/plain' \
# Sets the expected type of response to plain text
  -H 'Authorization: Bearer <TOKEN>' \
# Includes the authorization token for access control
  -H 'Content-Type: multipart/form-data' \
# Indicates that the data being sent is in multipart form,suitable for uploading files
  -F 'file=@"path/to/file.jpg";type=image/jpeg' \
#  new file to upload, replacing the existing one
  -F 'json_data=@"path/to/metadata.json";type=application/json'
#  new metadata for the replaced file
```

6. **Update a file using a PATCH request:**

```
curl -k -X 'PATCH' \
# Specifies the request type as PATCH
  'http://vm.ip.address/v1/update’ \
# URL to which the request is made
  -H 'accept: text/plain' \
# Sets the expected type of response to plain text
  -H 'Authorization: Bearer <TOKEN>' \
# Includes the authorization token for access control
  -H 'Content-Type: multipart/form-data' \
# Indicates that the data being sent is in multipart form, suitable for uploading
files
  -F 'file="file_to_update.jpg"’ \
# Indicates the file to update
  -F 'json_data=@"path/to/metadata.json";type=application/json'
# new or updated metadata for the file
```

6.2DL_TUI

Alternatively, it is also possible to interact with the API server using **`dl_tui`** interface for uploading, downloading, replacing, and updating files, as well as launching queries for processing data.

The wrapper can be called via a third executable, **`dl_tui`**, with one of the following actions:

- `upload`
- `replace`
- `update`
- `download`
- `delete`
- `query`

The IP address of the API server will be taken by the **`config_hpc.json`** configuration file. Alternatively, it is possible to overwrite the default via the **`ip =...`** option.

A valid authentication token is required. If saved in the **`~/.config/dlaas/api-token.txt`** file, it will automatically be read by the wrapper. Otherwise, the token can be sent directly via the **`token=...`** option.

The **`upload`** and **`replace`** actions require the following additional options:

- `file=...`: path to the file to be uploaded to the Data Lake
- `json_data=...`: path to the .json file containing the metadata of the file to be uploaded to the Data Lake

The update action also requires the **`file=...`** and **`json_data=...`** options, but in this case the **`file=...`** should be the S3 key corresponding to the file (i.e., the filename).

The download and delete actions require a **`file=...`** option, which similarly to the update action should be the S3 key corresponding to the file to be downloaded/deleted.

The query action requires the following additional options:

- **`query_file=...:`** path to the text file containing the SQL query to be ran on the Data Lake.

- **`python_file=...:`** path to the Python file containing the processing to be ran on the files matching the query.

For installing **`dl_tui`** application followinf instruction at: `https://github.com/Eurocc-Italy/dl_tui.git`

In this section examples are provided in order to launch **`dl_tui`** application

1. **Upload**:

```
dl_tui --upload --file=path/to/file.jpg --metadata=path/to/metadata.json
```

2. **Download:**

```
dl_tui --download --key=file.jpg
```

3. **Replace:**

```
dl_tui --replace --file=path/to/file.jpg --metadata=path/to/metadata.json
```

4. **Update:**

```
dl_tui --update --key=file.jpg --metadata=path/to/metadata.json
```

5. **Delete:**

```
dl_tui --delete --key=file.jpg
```

6. **Query:**

```
dl_tui --query --query_file=/path/to/query.txt [--python_file=/path/to/script.py] [--
config_json=/path/to/config.json]
```

7. **Browse:**

```
dl_tui --browse [--filter="category = dog"]
```

Application information and contributions

The application and this manual are released under the MIT open source license.

This manual, *version 1.0*, refers to tag *v1.0* of repos:

- https://github.com/Eurocc-Italy/dl_deploy.git
- https://github.com/Eurocc-Italy/dl_api.git
- https://github.com/Eurocc-Italy/dl_tui.git

The **Data Lake Ready to use** application was developed by **EuroCC Italy** national competence center, with the specific contribution of **CINECA** and **IFAB**.

The project was realized thanks to:

Benedetta Baldini, Luca Babetto, Domitilla Brandoni, Francesco Cola, Ivan Gentile and Eric Pascolo.