Data Lake Ready to Use

User Manual, version 1.2

This page intentionally left blank

Table of Contents

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. EuroCC2: Objectives

EuroCC2 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 Data Lake (DL) 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

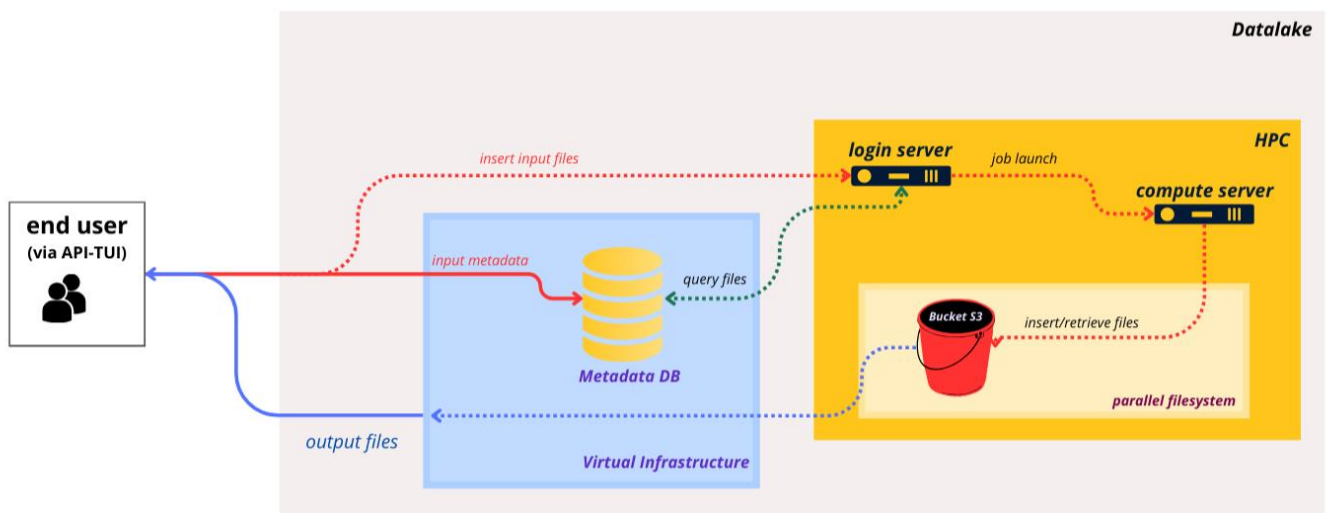Here is represented the graphical diagram of the service described in the previous sections.

Figure 2: Graphical Representation of service Architecture

- **End User:** It represents individuals or systems that interface with the Data Lake 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 Data Lake 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 Data Lake, 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. Text User Interface

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 credentials:
- Follow instructions 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 Data Lake'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
```

13. Log in to the API VM and generate authentication tokens (these are user-specific, generate one for each user and set an appropriate duration):

```
ssh -i ~/dtaas_sshkey.key datalake@<FLOATING_IP_ADDRESS>

source /opt/dtaas/bin/activate

dl_generate_token --user username --duration 86400
```

14. Save the generated token to a text file and provide it to the user, who can store it at the `~/.config/dlaas/api-token.txt` path on their local device, where it will automatically be detected by the TUI when launching commands.

6. API usage

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

6.1.   Upload (POST)

```
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
```

6.2.   Download (GET)

```
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
```

6.3.   Delete (DELETE)

```
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
```

6.4.   Query (POST)

```
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
```

6.5.   Replace (PUT)

```
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
```

6.6.   Update (PATCH)

```
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
```

7. Text User Interface

It is also possible to interact with the API server on the VM using the `dl_tui` executable for uploading, downloading, replacing, and updating files, as well as launching queries for processing data and browsing the contents of the Data Lake.

The general syntax for command-line calls is:

```
dl_tui --action --option1=value1 --option2=value2 ...
```

The API interface can be called via the `dl_tui` executable. The following actions are supported:

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

The IP address of the API server will be taken by the `config_hpc.json` configuration file (see the configuration section for more details). 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 executable. Otherwise, the token can be sent directly via the `--token=...` option.

7.1. Upload

To upload files to the Data Lake, use the `--upload` action. The following options are available:

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

Example:

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

NOTE: existing files cannot be replaced with this action. To replace an existing file or its metadata, use the `--replace` or `--update` actions.

7.2. Download

To download files from the Data Lake, use the `--download` action. The following options are available:

- `--key=...`: S3 key (equal to the filename) corresponding to the file to be downloaded

Example:

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

7.3. Delete

To delete files from the Data Lake, use the `--delete` action. The following options are available:

- `--key=...`: S3 key (equal to the filename) corresponding to the file to be deleted

Example:

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

7.4.   Replace

To replace a Data Lake file and its metadata, use the `--replace` action. The following options are available:

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

NOTE: only existing files can be replaced. To upload a new file, use the `--upload` action.

Example:

```
dl_tui --replace --file=/path/to/updated/file.csv –metadata /path/to/updated_metadata.json
```

7.5.   Update

To only update the metadata of a Data Lake file (without replacing the file itself), use the `--update` action. The following options are available:

- `--key=...:` S3 key of the file which needs to be updated
- `--metadata=...:` path to the .json file containing the metadata of the file to be updated in the Data Lake

Example:

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

7.6.   High-performance analytics

The library enables high-performance analytics on the Data Lake files via the `--query` action. The TUI will fetch the list of files matching the given SQL query, run the analysis on these files and upload the results back to the Data Lake. The following modes are currently supported:

- Python scripts
- Docker/Singularity containers

Analytics are run on Data Lake files using a query system. Users can select the files to be analyzed by writing a SQL query in a text file and providing it via the `--query_file` option.

7.6.1.   Simple query

If only the query file is provided, the corresponding files are zipped to an archive and uploaded to the Data Lake, ready for download. A job ID will be provided after the command launch. This will be the unique identifier for the query, and results will be provided in a .zip file named `results_<JOB_ID>.zip`.

Example:

```
$ dl_tui --query --query_file=/path/to/query.txt
Successfully launched query SELECT * FROM Data Lake WHERE category = cat

Job ID: ddb66778cd8649f599498e5334126f9d
$ dl_tui --download --key=results_ddb66778cd8649f599498e5334126f9d.zip
```

7.6.2. Python scripts

It is possible to provide a Python script for analysis and have the Data Lake run the script on the files matching the query. The path to the Python file should be provided using the `--python_file` option. The script needs to satisfy the following requirements:

- It must feature a `main` function, which will be **all** that is actually run on HPC (*i.e.*, any piece of code not explicitly present in the `main` function will not be executed). Helper functions can be declared anywhere, but must be explicitly called in `main`
- The `main` function must accept a list of file paths as input, which will be populated automatically with the matches of the SQL query
- The `main` function should return a list of file paths as output, corresponding to the files which the user wants to save from the analysis. The interface will then take this list of paths, save the corresponding files (generated in situ on HPC) into an archive which is then uploaded to the S3 bucket and made available to the user for download via the API

Below is an example of a valid Python script to be passed to the interface, with a `main` function taking a list of paths as input and returning a list of paths as output.

The script uses the scikit-image library to rotate the images given in input and save them alongside the original, using matplotlib to generate the comparison "graphs" and imageio to open the actual image files. The comparison images are saved in a temporary folder (`rotated`) whose contents are returned by the main function in form of file paths. The script also uses the Python multiprocessing libary to run the job in parallel, exploiting the HPC performance for the analysis.

```python
import imageio as iio
import skimage as ski
import matplotlib.pyplot as plt
import os, sys
from multiprocessing import Pool

def rotate_image(img_path):  # "worker" function, expecting a file path as input
    os.makedirs('rotated', exist_ok=True)  # Create temporary folder if doesn't exist

    image = iio.imread(uri=img_path)  # Open file with image
    rotated = ski.transform.rotate(image=image, angle=45)  # Rotate image

    fig, ax = plt.subplots(1,2)  # Initialize comparison plot
    ax[0].imshow(image)  # Print original image to the left
    ax[1].imshow(rotated)   # Print rotated image to the right

    path = f'rotated/{os.path.basename(img_path)}'  # Generate output file path name
    fig.savefig(path)  # Save image

    return path  # Return the path of the newly saved image

def main(files_in):  # main function, expecting a list of file paths as input
    with Pool() as p:  # Use all available cores
        files_out = p.map(rotate_image, files_in)  # Run rotate_image on all files

    return files_out  # Return list of file paths
```

To launch the analysis, use the following command:

```
$ dl_tui --query --query_file /path/to/query.txt --python_file /path/to/script.py
Successfully launched analysis script /path/to/script.py on query SELECT * FROM Data
Lake WHERE category = cat

Job ID: a2b8355051940f6cf09ff225d8340389
```

7.6.3. Docker/Singularity containers

It is also possible to run a Docker/Singularity container to analyze the files matching the query. Users can either provide the path to the container image via the `--container_path` option, or alternatively provide the URL of the image to be used via the `--container_url` option. By default, containers are launched in "run" mode (equivalent to `docker/singularity run image.sif`). It is possible to provide a specific executable to be used via the `--exec_command` option; in this case the container is launched in "exec" mode (equivalent to `docker/singularity exec /path/to/executable image.sif`).

Input files will be automatically provided by the Data Lake API to the container executable based on the query matches. An `/input` folder will be bound to the container corresponding to the folder on the parallel filesystem where Data Lake files are stored (in read-only mode).

The executable should expect input file paths as CLI arguments (equivalent to `docker/singularity run /input/file1.png /input/file2.png /input/file3.png ...`)

The container should save all results that should be uploaded to the Data Lake to the `/output` folder, which will automatically be created and bound by the Data Lake infrastructure at runtime

7.7. Extra utilities

7.7.1. Browse files

The `--browse` action allows users to browse the Data Lake content. The option `--filter=...` is available, which accepts an SQL-like query string for listing the requested files, removing the `SELECT * FROM Data Lake WHERE` part of the query itself and only leaving the filters. For example, `SELECT * FROM Data Lake WHERE category = dog OR category = cat` becomes `filter="category = dog OR category = cat"`.

Example without filter:

```
$ dl_tui --browse
Filter: None
Files:
  - cat_1.png
  - dog_1.png
  - cat_2.png
  - dog_2.png
  - cat_3.png
  - dog_3.png
```

Example with filter:

```
$ dl_tui --browse --filter="category = cat"
Filter: category = cat
Files:
  - cat_1.png
  - cat_2.png
  - cat_3.png
```

7.7.2. Check job status

The `--job_status` action allows users to check the status of jobs running on HPC.

```
$ dl_tui --job_status
                       JOB ID   SLURM JOB   STATUS REASON
  42684ce4c6d2440b8f9ad6647581a52d    14673377  PENDING Dependency
```

7.8.   Configuration

The library first loads the default options written in the JSON files located in the `dlaas/etc/default` folder (which can be taken as a template to understand the kind of options which can be configured).

If you wish to overwrite these defaults and customize your configuration, it is recommended to save a personalized version of the `config_hpc.json` and `config_server.json` files in the `~/.config/dlaas` folder. The options indicated here will take precedence over the defaults. Missing keys will be left at the default values.

If you wish to send custom configuration keys on the fly, it is also possible to pass configuration options to the `dl_tui_<hpc/server>` executables via the `config_hpc` and `config_server` keys in the input JSON file, also in JSON format. These will take precedence over both defaults and what is found in `~/.config/dlaas/config_<hpc/server>.json`.

For the hpc version, the configurable options are the following:

- `user`: the user name in the MongoDB server
- `password`: the password of the MongoDB server
- `ip`: network address of the machine running the MongoDB server
- `port`: the port to access the MongoDB server
- `database`: the name of the MongoDB database
- `collection`: the name of the MongoDB collection within the database
- `s3_endpoint_url`: URL at which the S3 bucket can be found
- `s3_bucket`: name of the S3 bucket storing the Data Lake files
- `pfs_prefix_path`: path at which the Data Lake files are stored on the parallel filesystem`modules`: list of system modules to be loaded on HPC

For the server version, the configurable options are the following:

- `user`: username of the HPC account
- `host`: address of the HPC login node
- `venv_path`: path of the virtual environment in which the libraries are installed
- `ssh_key`: path to the SSH key used for authentication on the HPC login node
- `compute_partition`: SLURM partition for running the HPC job
- `upload_partition`: SLURM partition for uploading the files to the S3 bucket
- `account`: SLURM account for the HPC job
- `qos`: SLURM QoS for the HPC job
- `mail`: email address to which the notifications for job start/end are sent
- `walltime`: maximum walltime for the HPC job
- `nodes`: number of nodes requested for the HPC job
- `tasks_per_node`: number of proesses to spawn on each node for the HPC job
- `cpus_per_task`: number of CPU cores per process requested for the HPC job
- `gpus`: number of GPUs requested for the HPC job

Application information and contributions

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

This manual, `version 1.2`, refers to tag `v1.2` 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.