This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 777533.

PROviding Computing solutions for ExaScale ChallengeS

D5.2	Alpha release of the Data service		
Project:	PROCESS H2020 – 777533	**Start / Duration:**	01 November 2017 36 Months
Dissemination[1]:	Public	**Nature[2]:**	**R**
Due Date:	31 January 2019	**Work Package:**	**WP 5**
Filename[3]	PROCESS_D5.2_Alpha_release_of_the_Data_service_v1.0.docx		

ABSTRACT

During the first 15 months of its implementation, PROCESS has progressed from architecture design based on use cases' requirements (D4.1) and through architecture validation again based on use cases (D4.2) towards initial implementations of computing services (D6.1) and data services - effort presented here in the deliverable D5.2.
In D5.2 we provide an initial demonstrator of the data services, which works in cooperation with the computation services demonstrated in D6.1. The demonstrator is based on the design of the PROCESS data infrastructure described in D5.1. The implementation and initial integration of the infrastructure are based on use case requirements, formulated here as custom application-specific services which are part of the infrastructure. The central, connecting component of the data infrastructure is LOBCDER. It implements a micro-infrastructure of data services, based on dynamically provisioned Docker containers. Additionally to LOBCDER and use case-specific services, the data infrastructure contains generic data and metadata-handling services (DISPEL, DataNet). Finally, Cloudify integrates the micro-infrastructure and the orchestration components of WP7.

[1] PU = Public; CO = Confidential, only for members of the Consortium (including the EC services).
[2] R = Report; R+O = Report plus Other. Note: all "O" deliverables must be accompanied by a deliverable report.
[3] eg DX.Y_name to the deliverable_v0xx. v1 corresponds to the final release submitted to the EC.

Deliverable Contributors:	Name	Organisation	Role / Title
Deliverable Leader[4]	Ladislav Hluchý	UISAV	Coordinator
Contributing Authors[5]	Maximilian Höb, Jan Schmidt	LMU	Writers
	Adam Belloum, Reggie Cushing	UvA	Writers
	Hanno Spreeuw, Jason Maassen	NLESC	Writers
	Mara Graziani, Henning Müller	HES-SO	Writers
	Balazs Somoskoi, Jörg Pancake-Steeg	LSY	Writers
	Martin Bobák, Ondrej Habala, Martin Šeleng	UISAV	Writers
	Jan Meizner	AGH	Writers
Reviewer(s)[6]	Tobias Guggemos	LMU	Reviewer
	Matti Heikkurinen	LMU	Reviewer
Final review and approval	Maximilian Höb	LMU	Reviewer

Document History

Release	Date	Reasons for Change	Status[7]	Distribution
0.0	2018-11-20	Structure of the deliverable fixed	Draft	
0.1	2018-11-30	First version of texts in chapters 2-4	Draft	
0.2	2018-12-07	Version for discussion	Draft	
0.3	2018-12-16	Major changes to text	Draft	
0.6	2019-01-11	Updates to demonstration scenarios	Draft	
0.7	2019-01-18	Restructured for readability	Draft	
0.8	2019-01-24	Draft completed for internal review	In Review	
0.9	2019-01-28	Reviewed Draft	In Review	
1.0	2019-01-31	Final Version	Released	Public

[4] Person from the lead beneficiary that is responsible for the deliverable.
[5] Person(s) from contributing partners for the deliverable.
[6] Typically, person(s) with appropriate expertise to assess the deliverable quality.
[7] Status = “Draft”; “In Review”; “Released”.

Table of Contents

Executive Summary

Following the architecture design described in D4.1 and the three-step approach described in D4.2, we have provided a design in D5.1 and now continue in D5.2 to describe our progress in developing and integrating the PROCESS data services. In D4.2 we were mostly focused on describing the design of the data services (D4.2-Section 2 page 10-12), evaluating the technologies (D4.2-Section 2.3.1 page 15-17), and presenting the first proof-of-concept implementation of a *micro-data infrastructure* for storage federation (D4.2-Section 2.3.2 page 17-19). In D5.2, we focus on the implementation of the data services and, more importantly, we describe the interaction of these data services with the various PROCESS use cases. We also show how the PROCESS use cases benefit from the support of the PROCESS data services.

First, Section 1 of this deliverable describes the main data services: LOBCDER, DataNet, and DISPEL. As a second step, we describe all the mechanisms and APIs needed for the interactions in Section 2. Section 3 describes the applicability to the different PROCESS use cases and derive scenarios for the demonstration at the project review in M18. Even though our demonstration scenarios show a full integration of the data services with the other components developed in PROCESS, the modular approach followed allows us to integrate the data service in other existing and well-established data processing frameworks. This multi-purpose nature of the project outputs eases the dissemination and promotion of the use of the PROCESS data services beyond the scope of the project.

D5.2 describes the alpha release of a limited number of data services, but we expect more data services being developed based on the new use case requirements planned in D4.3 in M18.

List of Figures

List of Tables

1 Overview

The PROCESS data service environment is composed of the three main components LOBCDER, DataNet and DISPEL for a virtual file system, meta-data environment and data (pre)processing respectively. They are connected to data sources and managed by a service orchestration environment as shown in Figure 1. A core component of the environment is a distributed virtual file system driven by LOBCDER. The features of this tool have evolved according to requirements of the use cases (D4.1[8], D4.2[9], D5.1[10]). The current version follows a micro-infrastructure approach that allows creating a use case specific data service infrastructure in a container. The PROCESS data service environment has a dedicated set of tools for a meta-data management and data (pre)processing. The metadata environment (driven by DataNet) is implemented as swarm clusters of metadata repositories. The metadata manager communicates with the swarm clusters using JSON messages sent via a REST API and DataNet interacts with LOBCDER with another REST API. It also has access to data sources via dedicated data adapters. The (pre)processing environment driven by DISPEL offers several processing elements (e.g. data access, data filtering, and data integration). DISPEL accesses the data sources via dedicated data adapters. The DISPEL Gateway allows communication via WebDAV protocol or a REST API. The data service environment is administered by LOBCDER which connects to the service orchestration environment by REST API and WebDAV.

The PROCESS data infrastructure is meant to be programmable and customizable for every use case. Thus every application has its own set of data services deployed at runtime on the available storage resources. In this architecture, LOBCDER takes the role of the manager that is responsible for instantiating the data infrastructure for each application workflow. The other data services can be instantiated as containers on-demand to create *Kubernetes pods*. Service data containers instantiated by LOBCDER have different capabilities such as accessing remote storage (e.g. HPC file systems) or federated access to distributed storage using a dedicated interface.

[8] D4.1: Initial state of the art and requirements analysis, PROCESS architecture. PROCESS project report, 2018.

[9] D4.2: Report on architecture evaluation and Dissemination. PROCESS project report, 2018.

[10] D5.1: Design of a data infrastructure for extreme-large datasets. PROCESS project report, 2018.

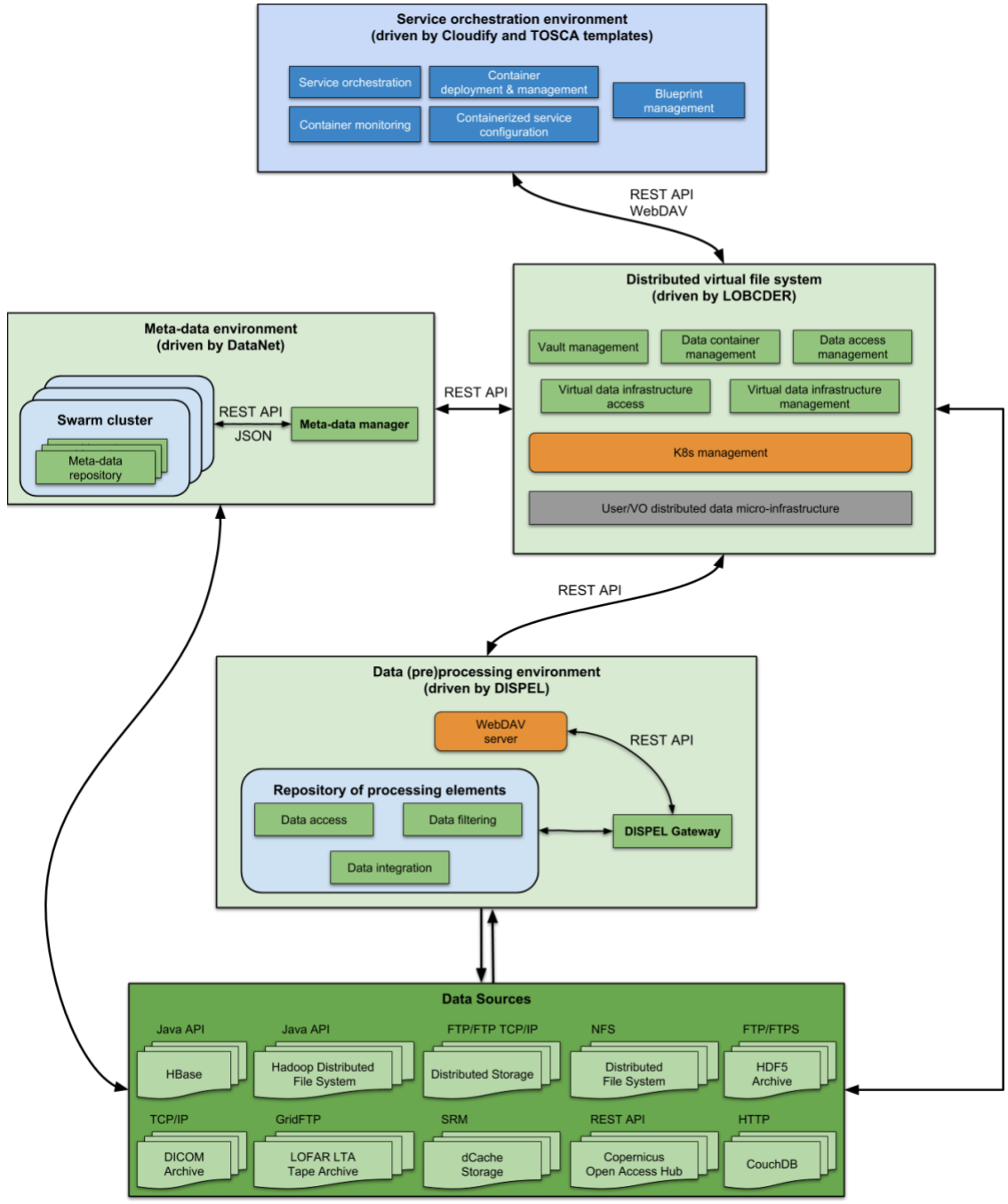

Figure 1 PROCESS data service environment (light green) with its interconnection to data sources (dark green) and the Service Orchestration Environment (light blue)

1.1 LOBCDER/Micro-Infrastructure

With LOBCDER we implement the micro-infrastructure approach to develop the PROCESS data platform. The idea of a micro-infrastructure is to decompose large, monolith infrastructures into more scalable and manageable infrastructures. This decomposition allows better scalability since it divides state management, such as indices, between many infrastructures. Furthermore, the increasing complexity of data requirements necessitates a programmable approach, optimisable for each application without interfering with the other ones. Leveraging the power of containers, we created a platform using Kubernetes allowing users to create an infrastructure with their dedicated data services. Typical data services include data store adaptors to connect to remote data such as HPC file systems, native cloud

storage using Ceph block storage, runtime services that have access to the storage such as WebDAV points, Jupyter notebooks and data staging services.

The LOBCDER constitutes a hyper-converged infrastructure that provides a virtualised distributed programmable data layer. The infrastructure is a Kubernetes cluster using VMS and physical nodes distributed amongst PROCESS partners. The Kubernetes cluster servers offer a programmable layer to abstract data services and storage. Data sources can be of two types. The first is a *Cloud-native storage*, where the storage is managed directly by the Kubernetes cluster through Ceph. In this scenario, a container has persistent storage in the cluster used to store or cache data for an application. The second type is a *HPC storage*, where the data service containers mount remote storages in HPC clusters. The sequence to access and make use of the data services is described in Figure 2, the first two steps of the sequence shown in Figure 2 are dedicated to the creation of the micro-infrastructure:

- **Request a token**: All the LOBCDER API calls are token protected. A token needs to be requested from the LOBCDER administration partner.
- **Create infrastructure**: After getting a token a user needs to create his own data infrastructure through API calls with the header x-access-token set to the received token

Once the information about the created data infrastructure is available, the execution environment can create and start the execution of the application data processing pipeline.

A REST API allows users to create their infrastructure as a set of pods and expose multiple WebDAV endpoints to access their data (see Appendix A).

An important point to mention here is the integration of LOBCDER with the Execution Environment (EE), which has to use the data services at several points during the application processing pipeline:

- The users' micro-infrastructure is dynamic and services and their corresponding ports may change. For this reason, the first step of integration with EE is to discover the user's endpoints. This is done through the management API, which provides a description of the endpoints by calling ***/api/v1/infrastructure***.
- Every micro-infrastructure exposes a WebDAV endpoint, which is accessible with a valid EE token. The endpoint provides access to all user's local and remote data through WebDAV.
- **Query and data staging service**: every micro-infrastructure implements a data query and staging service which lists the physical location of files and stage data onto HPC sites. The service can be used by the EE to identify the location of files on different HPC sites and to describe a staging pipeline with webhooks. These will asynchronously stage data (Figure 3) onto the HPC file system and use a webhook as a call-back to notify about staging progress.
- **Pre-processing workflows**: UCs such as UC#1 will have a pre-processing and staging workflow defined on the data services. They will be exposed as endpoints whereby the EE can call and register a "callback webhook" to be notified when pre-processing and staging has finished so that computation can commence.

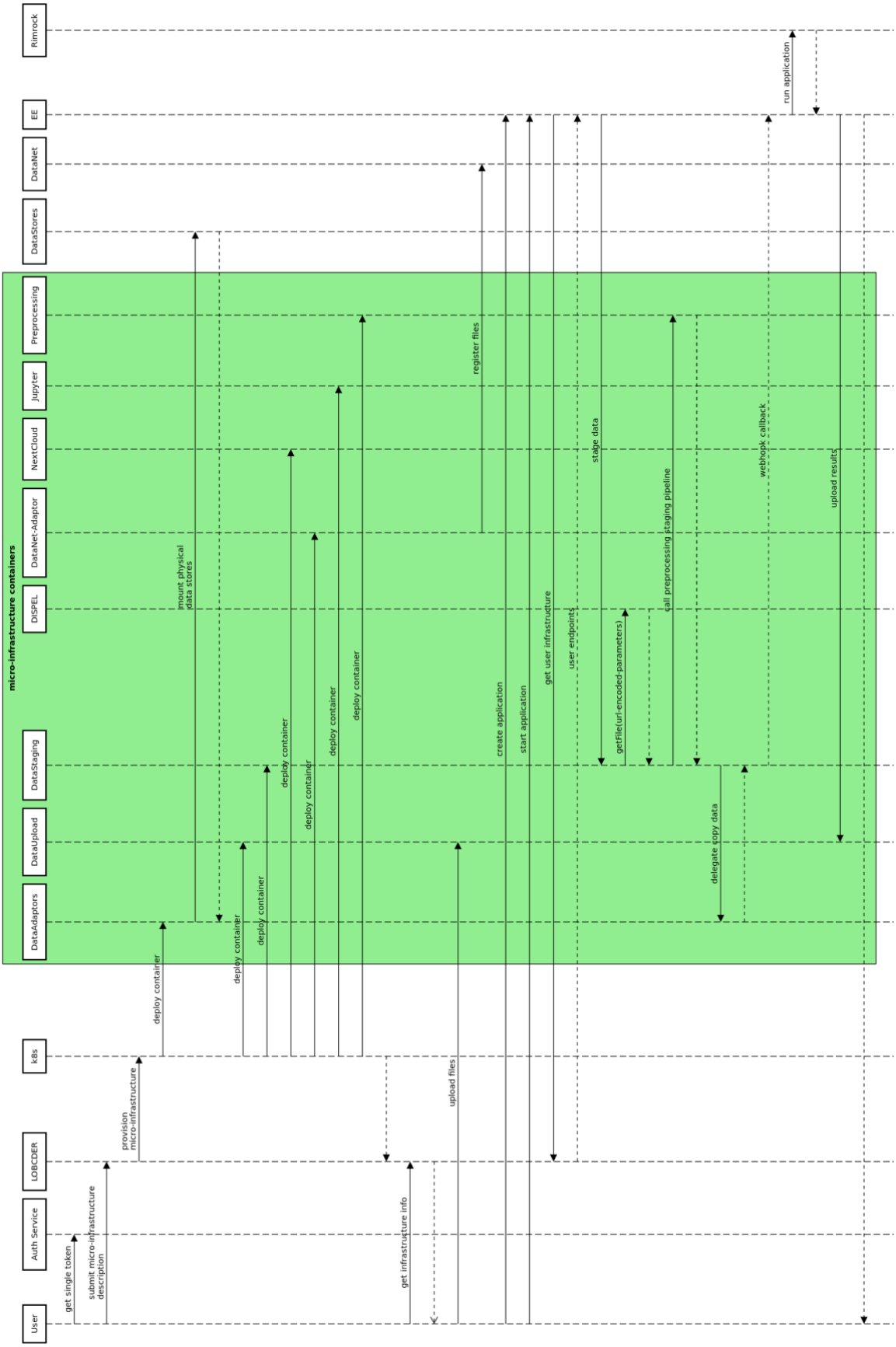

Figure 2 Sequence of interacting components from the user perspective. The green block is a dynamically created virtual infrastructure per use-case. The infrastructure encapsulates use-cases' data management, credentials, distributed resources and pre-processing routines.

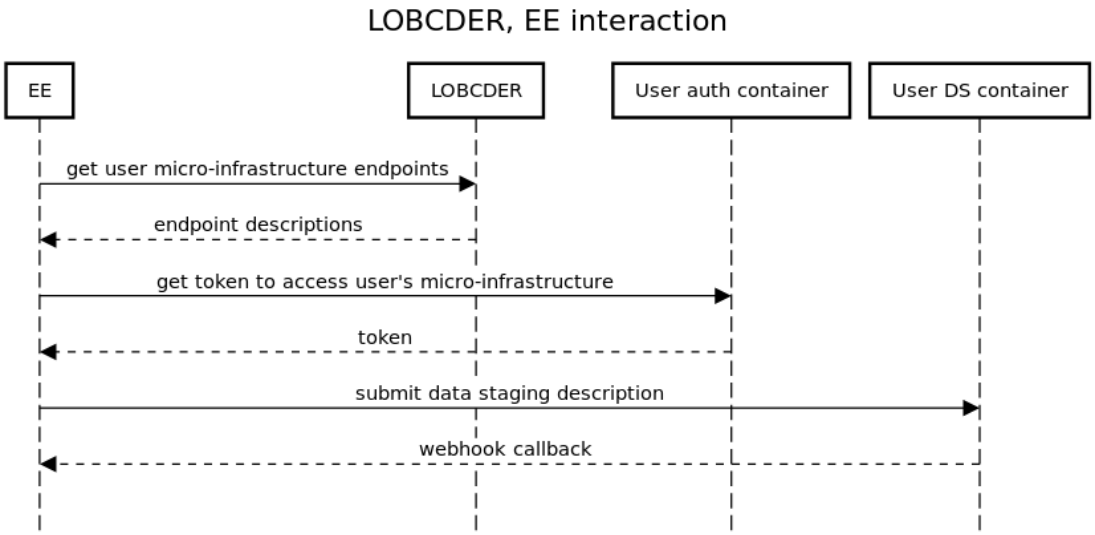

Figure 3 Simplified EE to LOBCDER interaction. EE queries LOBCDER to retrieve the user's dynamic infrastructure. EE can then access user's data services e.g. data staging.

The LOBCDER micro-infrastructure approach revolves around containers. For this purpose, several template containers are developed for the use in PROCESS. We categorize these containers into different groups depending on their capabilities. All the data containers identified from the requirement analysis fit two categories: the *logic containers* and the *Storage adaptors containers*.

Table 1 The categories of containers to create any micro-infrastructure

Storage adaptor containers	Provide access to remote storage such as HPC filesystems Examples: sshfs, GridFTP, Cloud-native-storage, etc.
Logic containers	Provide functionality on top of the storage adaptors. Examples: token-based WebDAV, Jupyter service, DISPEL service.

1.2 Storage Adaptor Containers

For this category, we are considering three *storage adaptor* containers (see Table 2). The project plans on developing additional atorage adaptors at the later stages of the implementation.

Table 2 Three storage adaptor containers

sshfs adaptor container[11]	The container can mount a remote folder through ssh credentials. When creating an infrastructure through the API, a user supplies his credentials to the remote server. The credentials are used to copy ssh keys to the remote storage and discarded after the operation has finished. This approach makes password-less authentication possible. A user can revoke access from LOBCDER at any time by removing the key entry in authourized_keys in his .ssh home directory.

[11] available on docker hub recap/process-sshfs:v0.1

gridFTP container adaptor	PROCESS uses gridFTP delegation service for high-performance data transfers between sites.
cloud-native adaptor	Storage is provisioned directly in the Kubernetes cluster using Rook/Ceph storage manager. The storage is mounted into a container and exposed alongside the other adaptors using WebDAV.

1.3 Logic Containers

Logic container help to develop and offer new services on top of three basic storage adaptors (see Table 2). User can access logic container services after the micro-infrastructure is deployed (see Figure 4):.

1. A user first needs a token to interact with the management API.
2. The user submits a JSON description of the infrastructure to the */api/v1/infrastructure* URL.
3. LOBCDER contacts the kubernetes API to initialise a micro-infrastructure.
4. The user queries the API to get the endpoint descriptions which include URL and ports for the dynamically running services.
5. The user can access the running services.

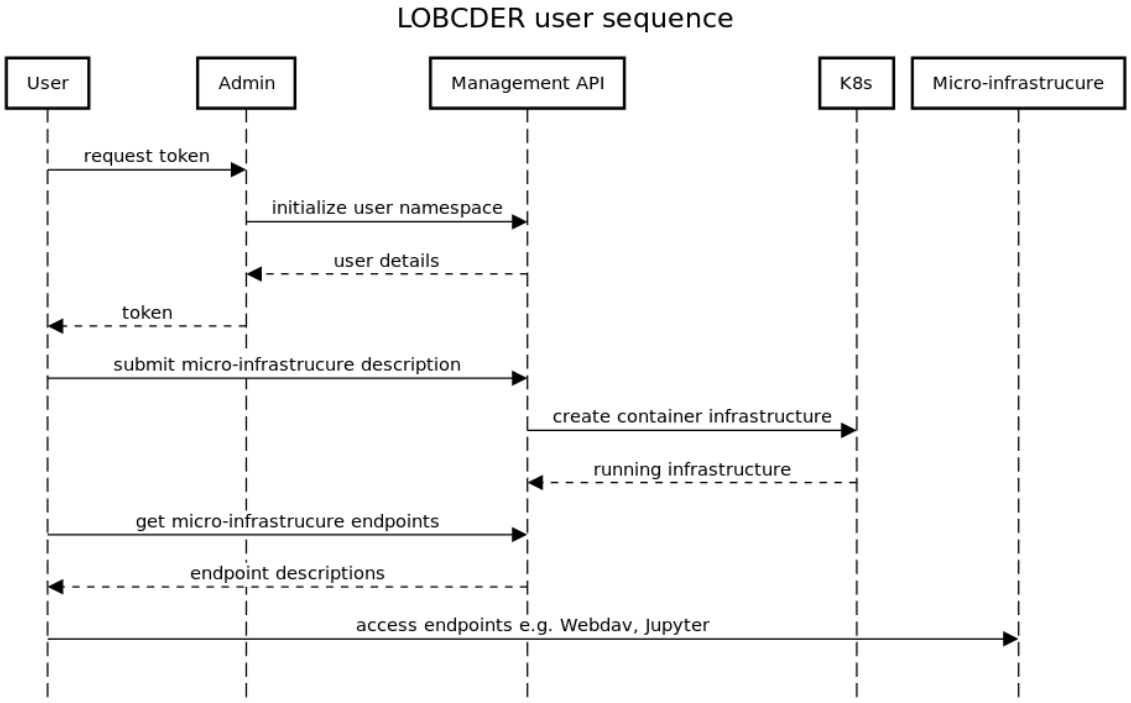

Figure 4 Simplified user to LOBCDER interaction. A user requests credentials to access LOBCDER API. The user can then submit requests to create a virtual infrastructure and access services

Table 3 List of Logic containers

WebDAV server[12]	Through the API infrastructure description, users supply a username and password for protecting the WebDAV point since this will we exposed publicly.
token based WebDAV server[9]	The API provides access by computing services. We modified a standard WebDAV server to authenticate using web tokens needed by the execution environment. When supplying the infrastructure description, a user also supplies the list of authorised users with their public keys through the WebDAV endpoint. This WebDAV implementation is expecting the WebDAV calls to have a header 'authorization' filled with a token provided by an external entity (in/out case the execution environment). Upon access, the WebDAV server decodes the header token check the user email is in the list of users and check the signature by decrypting using the public key provided when setting up the infrastructure.
Jupyter service container	Jupyter service container allows the user to access data through a processing environment whereby they can perform lightweight processing inside the data infrastructure. The adaptor data is mounted in /data folder on the container. The future releases extend this into a general user interface container for PROCESS with PROCESS-specific Python modules. This general UI forms the basis of use case-specific UIs with Python modules to handle the different data and pipelines needed.
Query/Staging service[13]	This service lists the files and their location on the adaptor containers. The purpose of this service is to incorporate also staging capabilities for integration with EE where EE can request data staging between adaptors so that applications would have just-in-time data on the HPC file systems (API calls see Appendix D). These containers are meant to optimise the execution of application workflows, e.g. by scheduling data transfers between sites. Caching containers, with cloud-native data storage, will enable frequently accessed data remaining easily accessible (to be developed).
DISPEL	DISPEL container provide access to an entire data (pre)processing environment. The DISPEL data processing environment is currently available as a Debian-based virtual machine with a complete deployment of all tools, manuals and a tutorial with example data processes. The VM contains a graphical development environment based on Eclipse, shown in Figure 7. DISPEL is accessed via standard WebDAV interface (HTTP protocol) since it is part of the LOBCDER distributed data infrastructure. The parameters for data processing are encoded in the provided URL, as described previously in D5.1. The URL encodes the following parameters: (1) Selection of DISPEL template (in the DISPEL language). (2) Zero or more parameters required by the template.

[12] available on docker hub recap/process-webdav:v0.3

[13] available as an image on docker hub recap/process-core-query

	Example of URL-encoded parameters: **http://lobcder.process-project.eu/dispel/tiffstore/312/20181129/12/0-1200-0-400** Components of the URL are: **http://lobcder.process-project.eu/**: the URL of the LOBCDER WebDAV server which provides the data **dispel:** a prefix which LOBCDER uses to recognise the sub-repository to contact (the DISPEL service) **tiffstore:** selection of the DISPEL data process template to execute **312:** subject designation (application-specific metadata) **20181129:** data creation time (application-specific metadata) **12:** layer in a multi-layer TIFF file (application-specific metadata) **0-1200-0-400**: grid selection (application-specific metadata)
DataNet-adaptor	DataNet-adaptor container allows pushing of metadata to the Datanet service. Datanet allows performing operations on the metadata sets such as: (1) Creating/ Updating/ Querying/ Deleting entities. DataNet offers straightforward user access via the REST API as well as GUI HAL browser. DataNet is available in the form of the Java source code under the OSI approved license as well as a Docker Container for the convenient deployment. DataNet Rest API is described in Appendix C.
NextCloud[14]	Next Cloud container allows the user to view their data in dropbox fashion.

Following, we show the user interfaces of four *logic containers*: the Jupyter service (Figure 6), Querying files (Figure 5), the DISPEL graphical authoring and execution environment (Figure 7), and NextCloud (Figure 8).

[14] available as *recap/process-nextcloud* image on docker hub

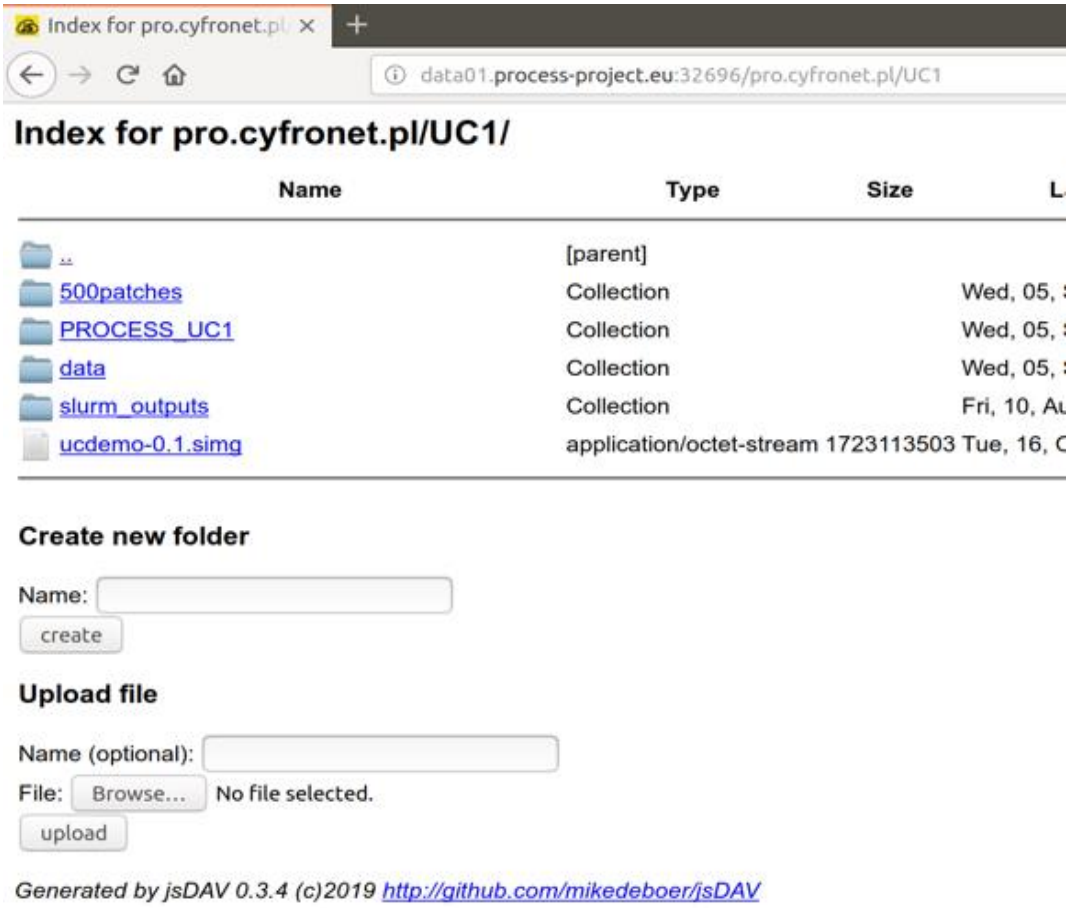

Figure 5 WebDAV data service deployed through the micro-infrastructure API. Through the API the user sets the username and password protecting the WebDAV point

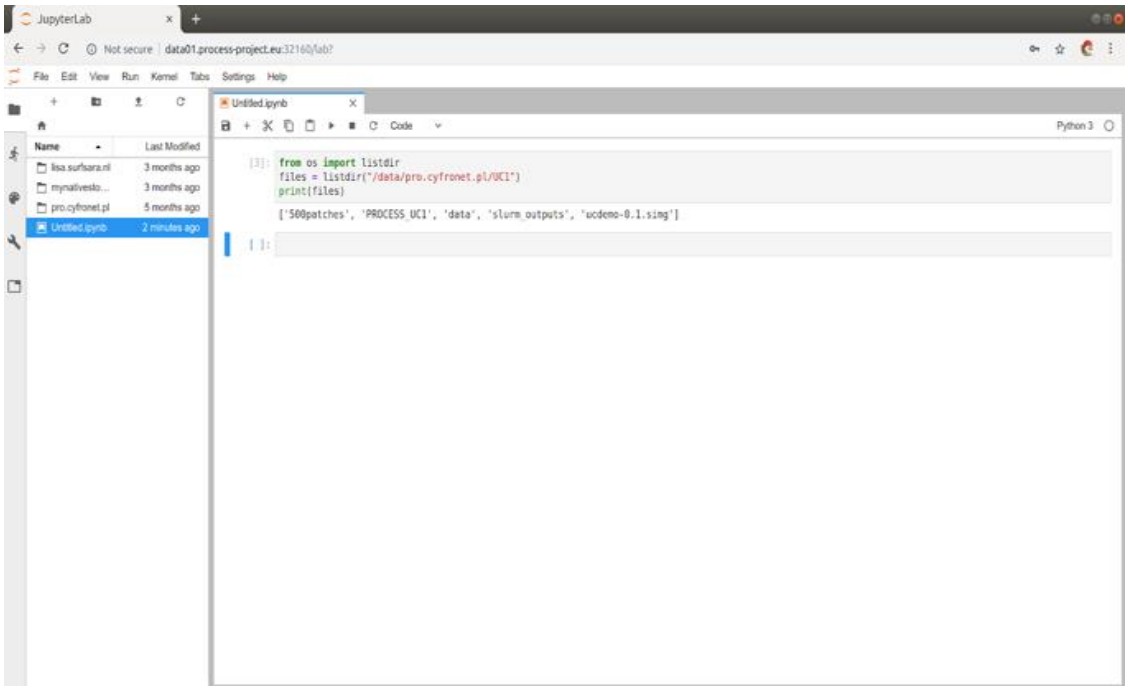

Figure 6 Jupyter service using the storage adaptor. Querying files from the Prometheus adaptor through WebDAV service

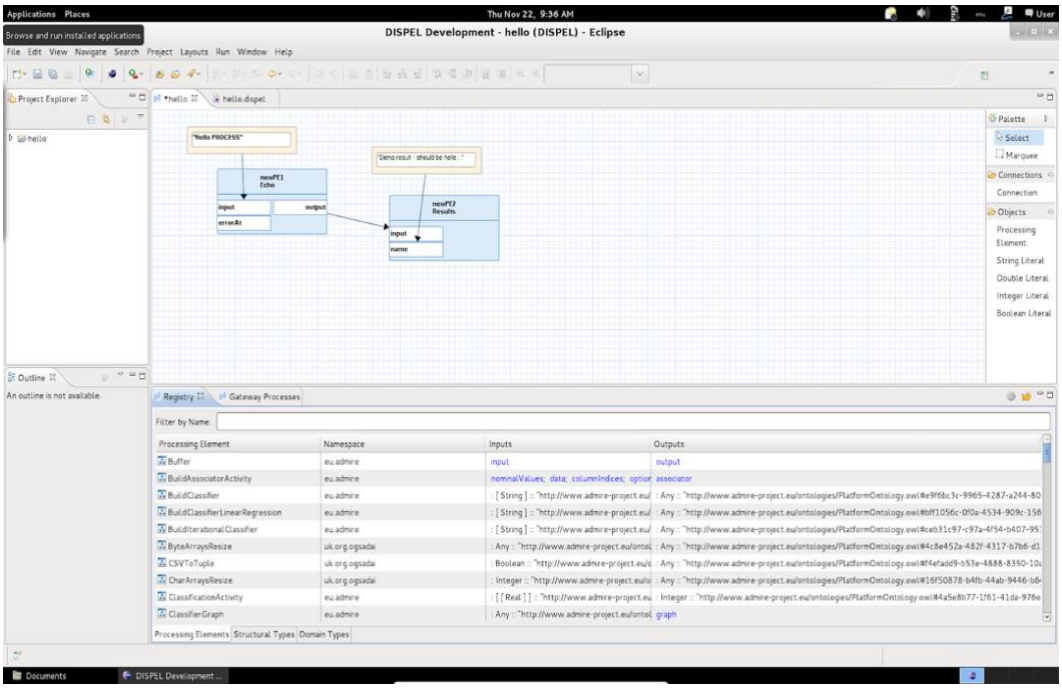

Figure 7 DISPEL graphical authoring and execution environment based on Eclipse

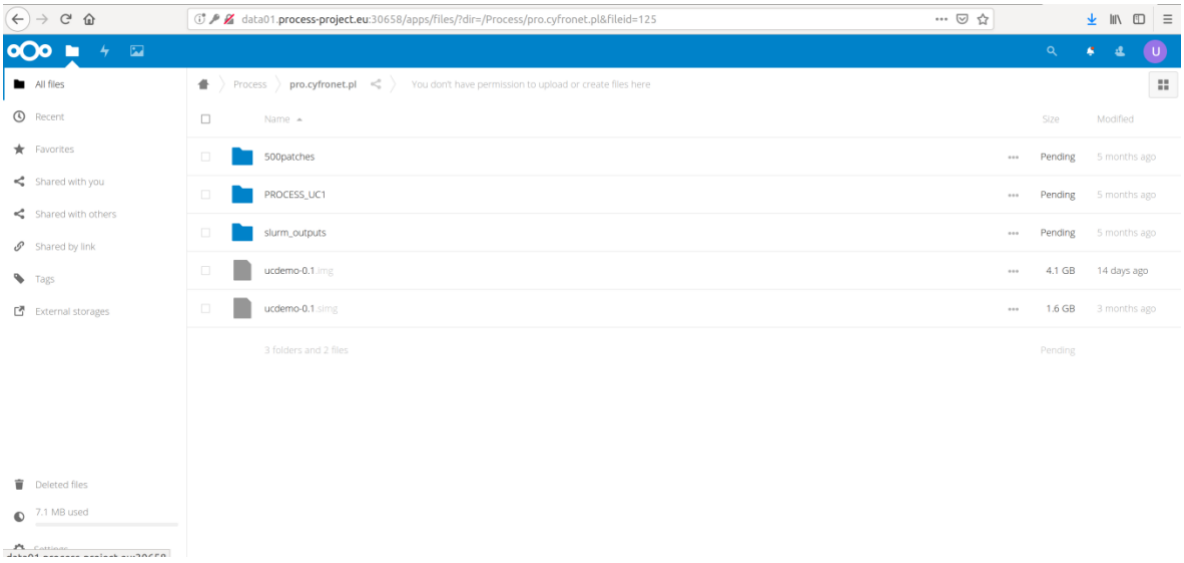

Figure 8 NextCloud service exposes a GUI to the user to work with user's files.

2 PROCESS data services from the user perspective

2.1 Data Services for Medical Use Case

The application setup of UC#1 can be split into two workflows. The first is about data staging and pre-processing, while the second is the neural network training which needs GPU compute nodes. Ideally, the compute workflow needs fast access to the pre-processed data, which means having the pre-processed data ready on the local filesystem before starting the computation. For this reason, we propose the pre-processing and staging workflow to be part of the data services that can pre-process and push the datasets directly onto the HPC file systems.

The pre-processing in UC#1 extracts patches from high-dimensional medical images. The runtime input requires a series of hyperparameters, such as the staging location of the data, the resolution level of the extracted the patch, the patch size and stride, and the patch sampling strategy (i.e. random sampling, importance sampling, dense coverage). The filesystem is scanned, and patient metadata and doctor annotations are retrieved for each image. Based on the physician's annotations, the system builds a binary mask of normal and tumour tissue. From each of the two tissue types, a set of image patches is extracted, by sampling locations in the high-dimensional image, according to the sampling strategy. The pixel values of the image patches, metadata about the patient, the lymph node, the hospital that handled the acquisitions, the resolution level of the patch, the doctor annotations and the patch location in the image are stored in an HDF5 database.

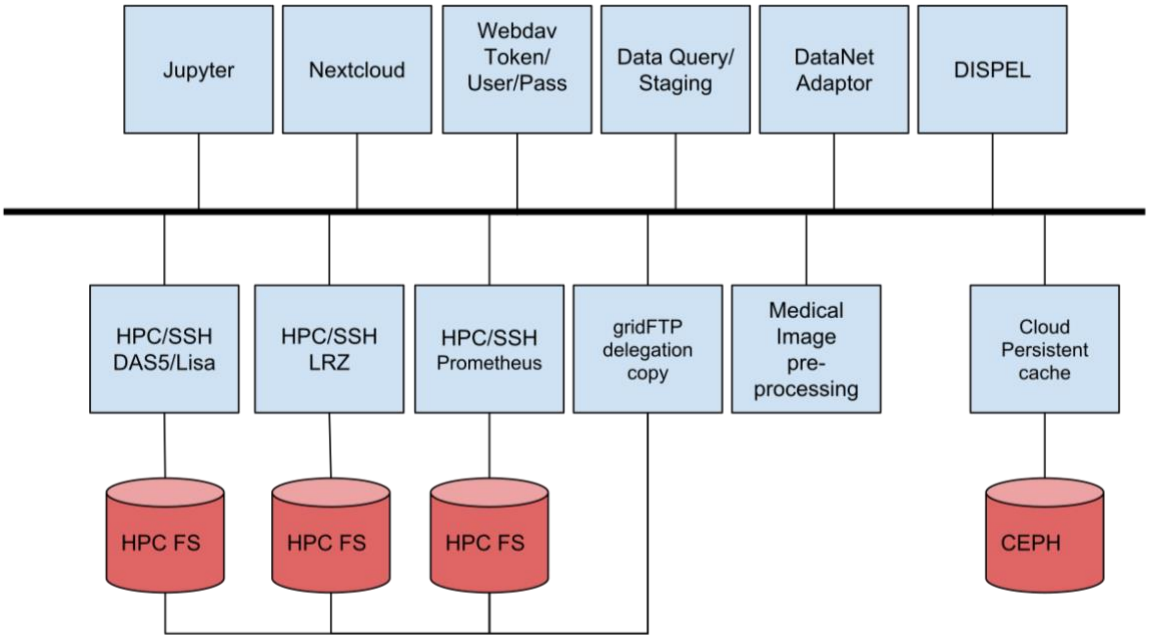

Figure 9 Micro-infrastructure for UC#1

In Figure 9, we illustrate the set of containers proposed for the data micro-infrastructure setup for UC#1.

- **WebDAV service**: two WebDAV containers are used to expose the data as a filesystem. For authentication, user/pass is available for standard WebDAV clients, while the Execution Environment uses the token-based mechanism.
- **Copy service**: The role of this container is to expose a REST API that handles copying files between sites or pull public files from the internet directly onto the HPC file systems.

- **Query service**: This REST service container queries all file systems to find he storage location of the physical file. The query service is a starting point for the integration with DataNet.
- **Pre-processing service**: This REST service container allows users to define input raw data, input hyper-parameters to generate new pre-processed datasets and HPC output locations so that the pre-processed datasets are pushed directly onto the HPC filesystems. It also keeps track of these generated datasets using a local database.
- **Pre-processing runtime**: This container encapsulates the logic of pre-processing.
- **Cloud persistent storage**: This container exposes a storage block hosted directly inside the Kubernetes cluster. This storage is used to host raw data that is needed by the pre-processing pipeline and acts as a cache for the generated datasets.
- **HPC SSHFS**: These are standard rudimental containers acting as adaptors to the HPC file systems. Through these adaptors, the copying service can push/pull data from the HPC clusters.
- **Key/Value DB**: A container that maintains state such as indexes for the generated datasets and location of the files.

2.2 Data Services for LOFAR Use Case

The LOFAR use case presents some specific conditions for efficient scaling of PROCESS data services that need to be met and which influence the design of the data services:

- The pipelines should be executed by containers, which allows efficient increasing the number of LOFAR archival observations by increasing the number of containers respectively - assuming these observations are of the same size.
- Simultaneous or quasi-simultaneous processing of multiple observations has the benefit of reducing latencies induced by data transfers -i.e. staging of observational data, from tape to dCache and from dCache to a compute cluster - and by compute bottlenecks.
- Data transfers may take significant time due to the data sizes and distances involved. Even with a speed of 10 GBit/s, a 16 TB dataset requires about four hours to transfer. Fortunately, copying data from a temporary disk to the processing location can be split based on the observational sub-band. Thus, staging and copying can overlap.
- Compute bottlenecks can occur in between the two subsequent calibration steps. The first step is direction independent and is embarrassingly parallel, by distributing the different sub-bands of a single observation (typically 244 sub-bands per observation) over the different nodes, with one sub-band per node. Processing can start as soon as a sub-band has been copied to a node disk - usually a four-hour operation. The next step is direction dependent calibration with an algorithm that needs a unified memory space to compute the calibration solutions. Typically, this step takes four days on a single fat node with hundreds of GB of RAM, leaving the remaining nodes idle when processing a single observation.
- Processing of multiple LOFAR archival observations simultaneously by many containers reduces latencies on the compute nodes after the first calibration step of the first observation has been completed. Also, it is crucial that the reservation of the compute nodes is made intelligently, i.e. avoiding situations where the nodes wait for data to arrive at the cluster.

PROCESS can offer access to the three sites storing LOFAR Long Term Archive data – Amsterdam, Jülich and Poznan – making simultaneous combined staging and processing of observations possible. Durrently processing of data from different sites requires separate user interfaces. Figure 10 presents the services required for the LOFAR Use Case pipeline.

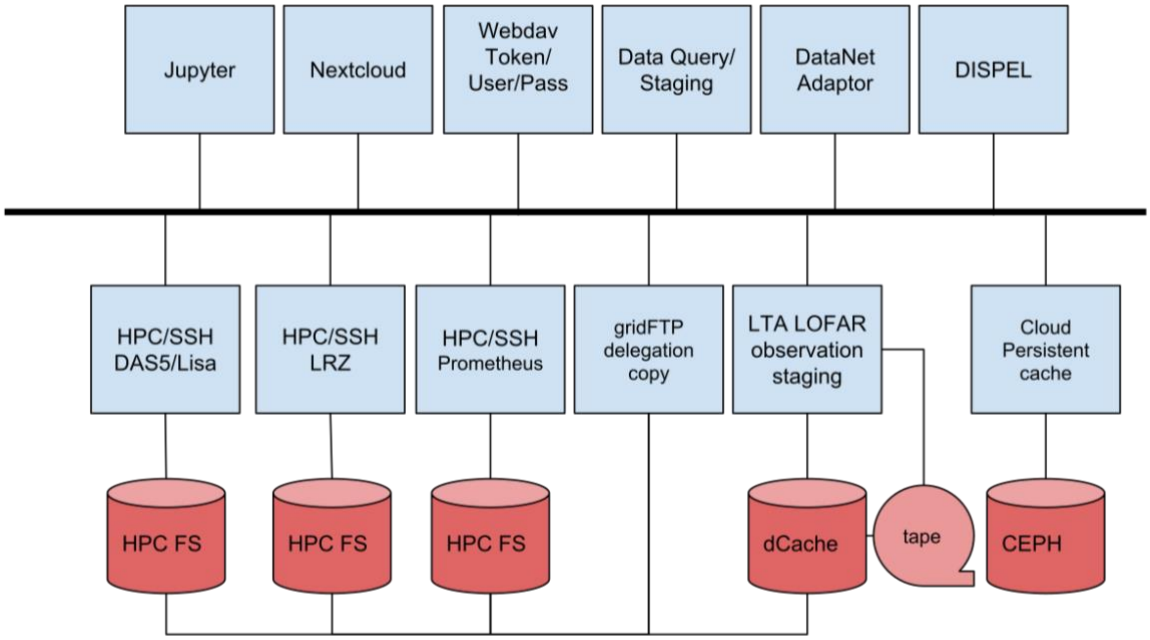

Figure 10 UC#2: data infrastructure including data adaptors as well as long-term-archive staging service.

2.3 Data Services for UNISDR Use Case

The role of the UC3 is to act as a channel to promote new PROCESS services to the users of the UNISDR portal. In the initial phase, it is sufficient to have a solution where the basic LAMP software stack can access a moderately sized data set through a PROCESS service that presents a standard file system interface.

The technical roadmap for determining the additional services is as follows:

- Integrate the current UNISDR service as a component of the PROCESS architecture - possibly also as a deployable service to provide web-based access to a read-only dataset in a restricted network.
- As advanced PROCESS functionalities become available, analysis of the usage patterns of the portal will help to identify the most relevant aspects to integrate into future releases.
- In parallel to this, the suitability of the PROCESS AAI solution as mechanisms to provide the users with an option to register (in addition to the current anonymous use) is analysed. The (self-) registration functionality is a prerequisite for offering functionalities such as workflows, collaborative metadata generation or derived datasets based on the official UNISDR data.

While the size of the original UNISDR dataset is fixed (~1.5TB), linking it with other data sources (either emerging from the preparations of the next UNISDR Global Assessment report or identified as relevant from the exploitation point of view) is being considered and may increase the overall storage requirements.

2.4 Data Services for Ancillary Pricing Use Case

The Ancillary Pricing use case consists of 3 main parts:

- The pricing service,
- The training environment based on available static data, and
- The stream data part incorporating streaming data into the model training.

We choose an approach to start with the modelling part based on the static historical data.

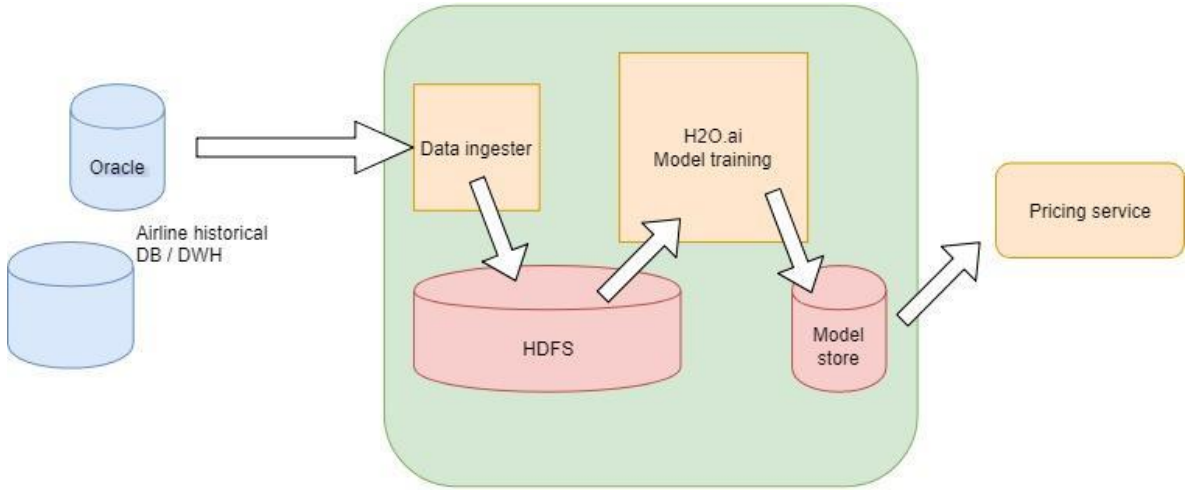

Figure 11 The pipeline of UC#4 Ancillary Pricing

From the data services perspective, we focus on the historical data part and the corresponding model training environment. This historical data is available in a diverse historical database or a warehouse solution at the airlines' data centres, stored in the form of relational data mostly in large Oracle instances. The challenge here is to ingest data from the existing different relational data sources like RDBMS.

After or even during the ingestion we need to consider the GDPR directives and mask or tokenise personal data - like names, date of birth - from the dataset. Data that could be used to identify an individual must be converted to broader demographically relevant information. As an example, the date of birth should be replaced by the age group. This kind of transformation is a preparatory or pre-processing step that should be part of the data services. Because the H2O.ai model training framework needs to work on massive amounts of data, it is stored on Hadoop/HDFS. Key architectural components of this UC are:

- HDFS/Hadoop/HBase
- Store model training results
- Services to use from PROCESS data service

2.5 Data Services for Copernicus Use Case

This use case uses datasets from the Copernicus Space Component Data Access (CSCDA) Services. PROCESS components are used to perform preparatory steps that are prerequisites for the in-depth analysis using the PROMET software.

The use case's workflow shown in Figure 12 triggers by the workflow definition in the PROCESS portal. During the pre-processing, the data is fetched and directly processed. The implementation of this processing will also show during testing if storing of input data withing the PROCESS infrastructure is needed. Afterwards, the data is to be transferred to the PROCESS data storage together with metadata information. The PROMET execution fetches this data and computes the raw result data files and formatted to a user-friendly output format during post-processing, including visualisations. The user can download this data through the PROCESS portal.

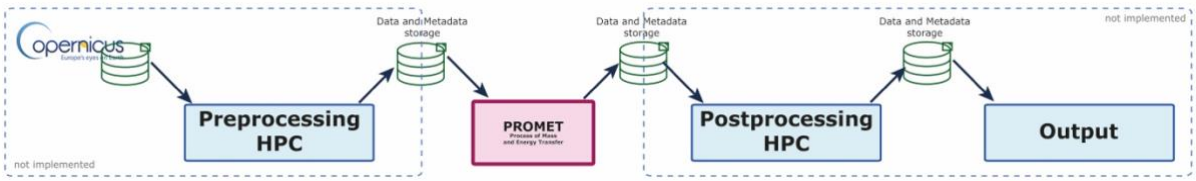

Figure 12 The pipeline of UC#5 Copernicus

In the ongoing development of the use case, the pre- and post-processing at this time of the project are not implemented to be portable outside the development infrastructure. However, it is work in progress to distribute this computation across the PROCESS computing resources. For this execution the PROCESS data services as shown in Figure 13 will be used to connect to the Copernicus data warehouse services and to transfer its data.

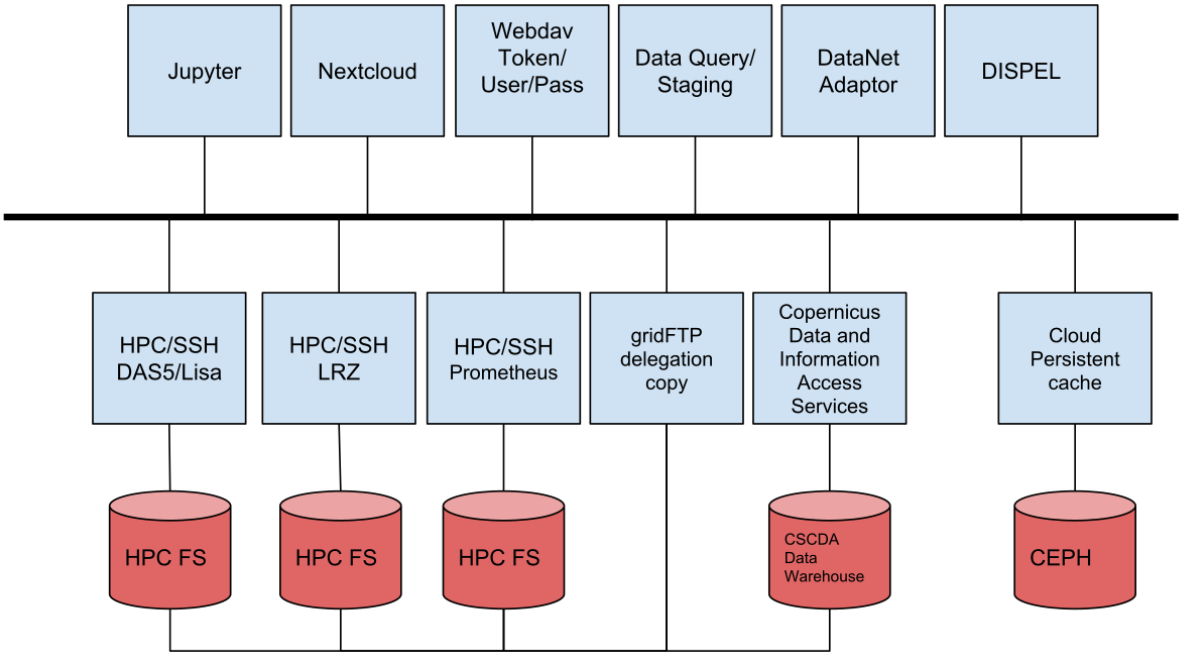

Figure 13 Data services for UC#5

3 Demonstration scenarios

During the first project review demonstrations will focus on two application scenarios derived from the mature use cases UC#2 and UC#5. Presenting PROCESS technology on UC#1 is also possible; therefore we include also UC#1 description in this chapter.

All PROCESS data services are accessible from the Interactive Execution Environment (IEE), demonstrated in D6.1[15]. It allows users to parameterise the application processing pipeline. After login into the EE portal, a user may create a new "Process" for each use case. In the first step, the user sees the same interface (Figure 14), through which they can select the application processing pipeline they want to work on. After this step, they are forwarded to the application specific portal to customise the application processing pipeline further.

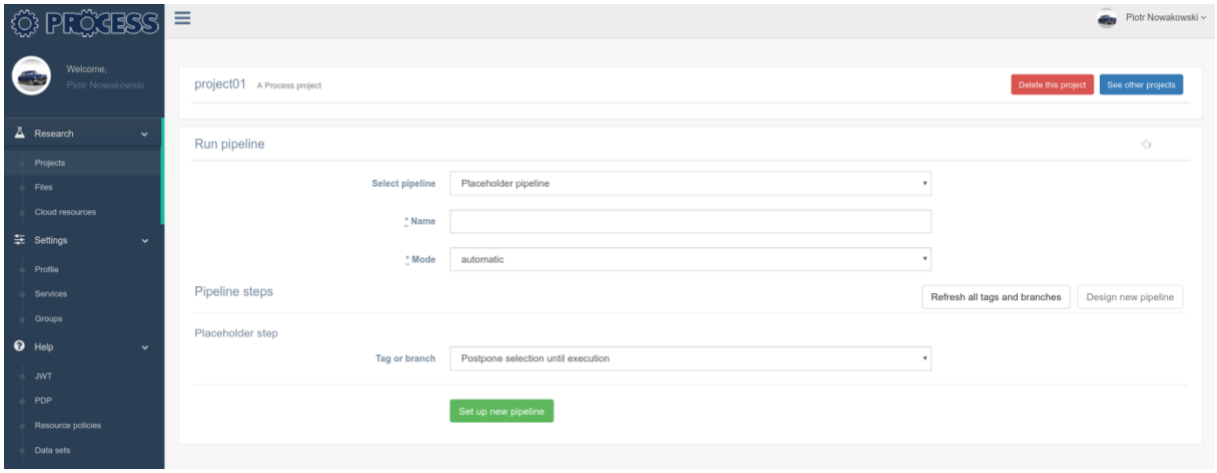

Figure 14 The Interactive Execution Environment UI modified to allow selection of UC parameter entry form

In the following text details for UC#1, UC#2 and UC#5 using the PROCESS alpha release demonstrator are shown.

3.1 UC#1 scenario

The scenario for UC1 is about training different models on different data centres. The goal is to train multiple models simultaneously on public datasets. The requirements of training very complex models on large scale datasets are met by PROCESS. In the following, we present the pipeline for the three phases of the use case, namely data staging, pre-processing and training.

3.1.1 Data staging

1. The user logs in with username and password.
2. From a menu of functionalities, the option Data Management is selected
3. The user specifies the location of the data on the local servers
4. The user selects one or more of the available data centres.
5. Upon confirmation of the user, the data is reliably transferred to the selected servers.
6. A confirmation message informs the user that the data has been successfully staged on each of the data centres.

[15] D6.1: First prototype. PROCESS project demonstrator, 2018.

3.1.2 Data pre-processing

1. The user logs in with username and password.
2. From a menu of functionalities, the option Data pre-processing is selected.
3. The user specifies the dataset of interest from the available options
4. The user selects a pre-processing pipeline. The pre-processing consists of extracting an arbitrary number of patches from the high-resolution images.
5. The user specifies a series of parameters, namely the number of patches to extract from each image, the patch resolution, the patch sampling strategy.
6. Upon confirmation of the user, the pre-processing pipeline is executed with the desired parameters.
7. An intermediate H5 database, obtained as result of the computations, is stored in the PROCESS infrastructure.
8. A confirmation message informs the user that the pre-processing of the data has been successful.

3.1.3 Network training

1. The user logs in with username and password
2. From a menu of functionalities, the option Network training is selected
3. The user specifies the intermediate database of interest from the available options.
4. The user specifies the model configuration. For instance, the model type, the loss function, the activation functions, the learning rate decay and momentum are specified together with the batch size and the number of training epochs.
5. The user confirms the configuration and launches the model training.
6. At the end of training, the model weights and training logs are stored and made available for download.

3.2 UC#2 scenario

The scenario for UC2 focuses the initial (direction independent) calibration of a LOFAR observation, through the “Prefactor” calibration package. This working example uses the observation of a calibrator as input. Its outputs are the calibration solutions for that calibrator.

1. The user creates a data infrastructure for the use case by submitting a description including credentials to access the distributed resources.
2. The user checks the status of the infrastructure and receives access credentials to the infrastructure.
3. The user logins into EE.
4. EE discovers the user’s data infrastructure endpoints, e.g. WebDAV and data staging.
5. The user creates an application pipeline and supplies the LOFAR visibility ID input and other relevant information (as shown in Figure 15).
6. The EE queries the data staging service with the LOFAR visibility ID and submits a staging request to stage in the 16TB of data to the HPC file system.
7. The EE registers a webhook with data staging service so that the pipeline can continue after data has been successfully staged-in.
8. The webhook is called, and EE continues executing the pipeline.
9. Depending on the output location, EE can stage-out the data using WebDAV or data-staging/upload services.

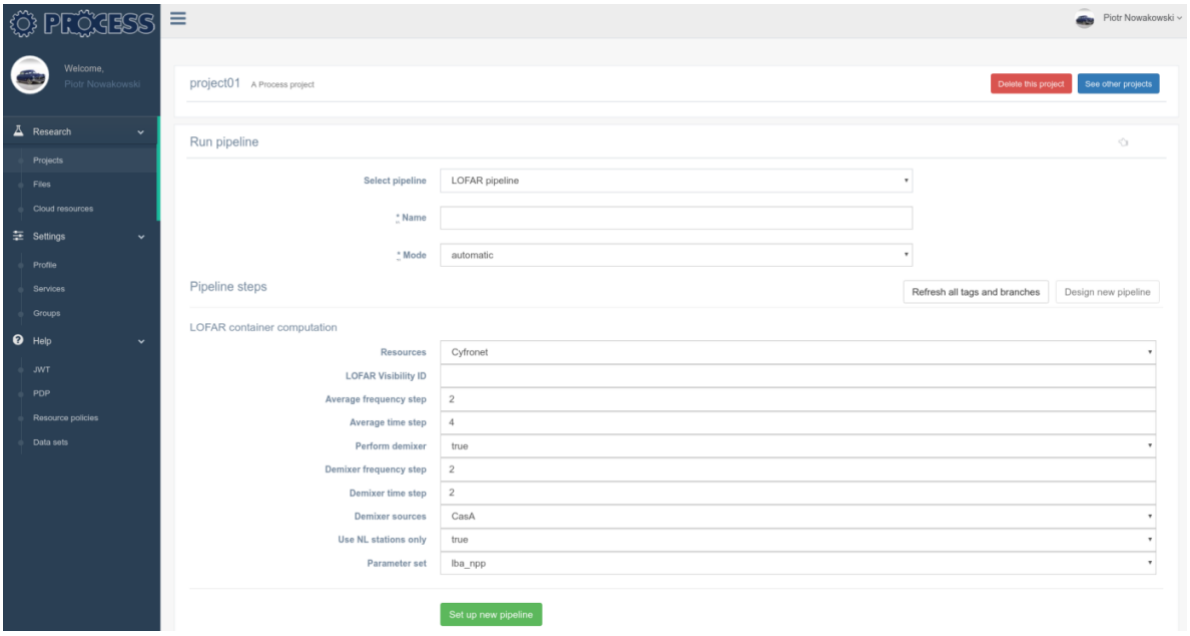

Figure 15 IEE interface with UC#2 parameter entry form displayed

3.3 UC#5 scenario

The UC#5 scenario focuses on a PROMET computation based on the configuration parameters supplied by the user in the EE. With this information, the input data is described and can be located. The kind of output depends on the user's configuration.

1. The user creates a data infrastructure for the use case by submitting a description including credentials to access the distributed resources.
2. The user checks the status of infrastructure and gets access credentials to the infrastructure.
3. The user logs into EE.
4. The EE discovers user's data infrastructure endpoints, e.g. WebDAV and data staging.
5. The user creates an application pipeline and supplies the relevant input parameters (as shown in Figure 16)
6. The EE queries the DataNet for the relevant metadata and stages in data through the data staging service.
7. The EE registers a webhook with data staging service so that the pipeline can continue after data has been successfully staged-in.
8. The webhook is called, and EE continues executing the pipeline.
9. Depending on the output location, EE can stage-out the data using WebDAV or data-staging/upload services.

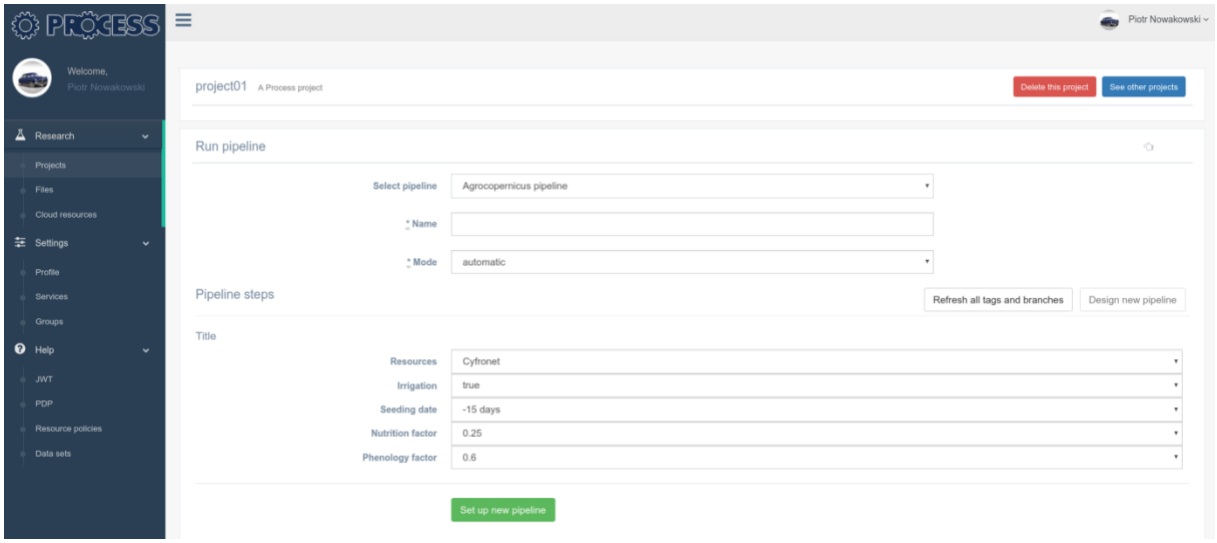

Figure 16 IEE user interface with the UC#5 parameter entry form displayed

3.4 Micro-infrastructure provisioning

As the first step towards the implementation of the different scenarios derived from the application use cases, we have created and a micro-infrastructure for each use case (see Figure 17 for general overview and Figure 18 for detailed list of all provisioned containers).

```
uc1    usecase-001-648b4f47f7-z698k    12/12    Running
uc2    usecase-002-58f67454b7-z599w    12/12    Running
uc3    usecase-003-8567bb5bb7-s49sz    11/11    Running
uc4    usecase-004-7bf8c887dd-f64kh    11/11    Running
uc5    usecase-005-8bc5ff45d-875v5     11/11    Running
```

Figure 17 Running micro-infrastructures for each use-case.

Each micro-infrastructure is composed of six data services, namely Datanet, DISPEL, Jupyter, Webdav with token authentication (jwt), Webdav with user/pass authentication (ht) and some application specific data services like the medical-pre-processing (UC#1) and the lta-lofar-prestaging (UC#2) (Figure 18).

uc1	usecase-001-datanet	NodePort	10.100.111.40	<none>	8003:31133/TCP
uc1	usecase-001-dispel	NodePort	10.98.191.146	<none>	8080:31925/TCP
uc1	usecase-001-ht	NodePort	10.107.120.46	<none>	8000:31478/TCP
uc1	usecase-001-jupyter	NodePort	10.108.215.81	<none>	8888:30548/TCP
uc1	usecase-001-jwt	NodePort	10.108.7.50	<none>	8001:30533/TCP
uc1	usecase-001-medical-preprocessing	NodePort	10.99.1.202	<none>	8888:31411/TCP
uc1	usecase-001-nextcloud	NodePort	10.98.211.254	<none>	80:30658/TCP
uc1	usecase-001-query	NodePort	10.104.199.161	<none>	8002:31901/TCP
uc2	usecase-002-datanet	NodePort	10.110.60.37	<none>	8003:32065/TCP
uc2	usecase-002-dispel	NodePort	10.103.209.41	<none>	8080:32450/TCP
uc2	usecase-002-ht	NodePort	10.107.185.13	<none>	8000:30685/TCP
uc2	usecase-002-jupyter	NodePort	10.108.171.227	<none>	8888:32402/TCP
uc2	usecase-002-jwt	NodePort	10.109.109.1	<none>	8001:32560/TCP
uc2	usecase-002-lta-lofar-prestaging	NodePort	10.111.38.122	<none>	8888:32095/TCP
uc2	usecase-002-nextcloud	NodePort	10.107.165.182	<none>	80:32535/TCP
uc2	usecase-002-query	NodePort	10.111.150.110	<none>	8002:32324/TCP
uc3	usecase-003-datanet	NodePort	10.110.88.114	<none>	8003:31540/TCP
uc3	usecase-003-dispel	NodePort	10.98.215.206	<none>	8080:30986/TCP
uc3	usecase-003-ht	NodePort	10.105.5.50	<none>	8000:31626/TCP
uc3	usecase-003-jupyter	NodePort	10.101.92.233	<none>	8888:31226/TCP
uc3	usecase-003-jwt	NodePort	10.102.26.45	<none>	8001:32603/TCP
uc3	usecase-003-nextcloud	NodePort	10.97.183.215	<none>	80:30044/TCP
uc3	usecase-003-query	NodePort	10.109.208.8	<none>	8002:32397/TCP
uc4	usecase-004-datanet	NodePort	10.110.57.63	<none>	8003:32338/TCP
uc4	usecase-004-dispel	NodePort	10.102.253.17	<none>	8080:31985/TCP
uc4	usecase-004-ht	NodePort	10.101.217.179	<none>	8000:30856/TCP
uc4	usecase-004-jupyter	NodePort	10.103.85.79	<none>	8888:30506/TCP
uc4	usecase-004-jwt	NodePort	10.97.39.242	<none>	8001:31880/TCP
uc4	usecase-004-nextcloud	NodePort	10.108.246.190	<none>	80:32168/TCP
uc4	usecase-004-query	NodePort	10.104.156.217	<none>	8002:30819/TCP
uc5	usecase-005-datanet	NodePort	10.101.230.94	<none>	8003:32475/TCP
uc5	usecase-005-dispel	NodePort	10.97.151.187	<none>	8080:31725/TCP
uc5	usecase-005-ht	NodePort	10.107.164.123	<none>	8000:32663/TCP
uc5	usecase-005-jupyter	NodePort	10.101.167.255	<none>	8888:30731/TCP
uc5	usecase-005-jwt	NodePort	10.110.158.108	<none>	8001:32743/TCP
uc5	usecase-005-nextcloud	NodePort	10.102.63.79	<none>	80:30817/TCP
uc5	usecase-005-query	NodePort	10.99.13.207	<none>	8002:32614/TCP

Figure 18 Provisioned services for use cases #1 through #5.

4 Conclusion

In this deliverable, we have focused on prototyping and presenting the PROCESS data infrastructure with a micro-infrastructure approach. The micro-infrastructure is created at runtime and composed of a set of data service containers, which are instantiated by the core of the infrastructure, LOBCDER. All the interactions among the software components composing the micro-infrastructure are clearly defined and used to create the initial implementations of PROCESS application scenarios' data handling pipelines, which are under development for demonstrations at the project review in M18.

All software components developed and reported in this deliverable are available in the PROCESS software repository[16]. Their development continues, and new versions will be available in the repository as soon as they become available.

4.1 Future work

This alpha release of the data services demonstrator will be used in the validation of PROCESS architecture and the production of its update (D4.3 in M18 and D8.1 in M21). We will continue to develop the micro-infrastructure management (LOBCDER) by improving the data sharing between containers and the connection to native cloud storage element using Rook/CEPH. Exploiting storage within LOBCDER as cache will be implemented and the integration with Cloud computing resources through Cloudify is going to be consolidated. We aim to improve the resource provisioning using TOSCA templates and provide more client tools to enables deployment and execution of new services. Besides consolidating the micro-infrastructure with Cloud resources, we plan to improve existing containers and to support further the UC application see Table 4:

Table 4 Planned further development of the general PROCESS data service containers

Jupyter container	Provide PROCESS-specific Python modules, which will allow interfacing with PROCESS components and services. Extend this container into a several, use case -specific versions with their respective use case-specific Python modules.
DISPEL container	Most immediate work to be done in DISPEL development is: • migration to a current base Docker image • implement caching of created data and tracking of it (via DataNet) • a virtual directory structure for existing data, based on configurable parameters • Advanced options for HTTP GET command (compression, continuation...)
DataNet container	• Providing mechanism for DataNet scalability inside each site (using mechanism for handling distributed containers such as Docker Swarm or Kubernetes) • Enable mechanism to distribute DataNet repositories to multiple sites • Provide better integration with the Data Infrastructure management platform esp. in the scope of auto-scaling • Allow simple storage of provenance data for computation tracking
GridFTP container	Necessary for p2p data transfers without the need to channel the data through LOBCDER. Currently, not all PROCESS computing sites accept the users' grid. It is important for the development of the use case scenarios that the credentials are accepted across the PROCESS data sites.

[16] https://software.process-project.eu

Finally, the application of the use cases will also be further developed, which might result in a new requirement for the next data services releases. Table 5 briefly summarises the planned development of the five applications use cases.

Table 5 Planned further development of the PROCESS use cases

UC#1	• Improving training and time performances of the algorithms • Increasing the dataset sizes and the number of models that is possible to train • Introducing visualisation techniques and model explainability
UC#2	This working example uses the observation of a calibrator as input. Its outputs are the calibration solutions for that calibrator. We are planning to add • The observation of a target field and apply the calibration solution for the calibrator to the target field. • The direction dependent calibration for the target field, through the "DDF" calibration package. Finally, we want to add imaging to the pipeline. • We want to calibrate a target field using a different package, e.g. SAGECal and compare ease of use and results.
UC#3	The work on UC3 will proceed on three tracks: 1) Identifying new users for the existing dataset and complementing it with related ones (e.g. emerging from projects, such as LEXIS) 2) Testing and integrating new services emerging from the PROCESS development to complement or replace the current UNISDR data portal 3) Actively searching new users who could use statistical disaster risk data in their research activities
UC#4	• Improve the training model, its performance; increase the number of possible models and introduce different modelling algorithms. • Create the pricing engine and incorporate the trained models. • Improve the test data generator and the creation of higher data volumes.
UC#5	• Containerized pre- and post-processing and include them into the Workflow • Enhance computations with higher resolution, and larger data set size • Allow a complete workflow configuration with all physical parameters

5 Appendices

5.1 Appendix A: LOBCDER REST API

A REST api allows users to create infrastructure as a set of pods and expose multiple WebDAV endpoints to access their data. The sequence to access and make use of the data services are:

- Requesting a token: All the LOBCDER API calls are token protected. A token needs to be requested from the UvA PROCESS partner.
- Creating infrastructure: After getting a token, a user needs to create his own data infrastructure through API calls with the header x-access-token set with the requested token. The virtual infrastructure is managed through a set of API calls on data01.process-project.eu.:

- **POST /api/v1/infrastructure** submits a description of the infrastructure needed by the user/use-case. The JSON description describes the components needed to access federated data. In this example a minimal infrastructure is described as having two ssh adaptors to different HPC sites (Cyfronet and Amsterdam) and two methods of exposing the data (normal WebDAV and token-based WebDAV). The backend takes care to map these descriptions to template containers created specifically for PROCESS.


```
 {"name":"test-001","namespace":"user-001","location":"uva",
  "storageAdaptorContainers":
  [{"name":"mynativestorage","type":"native","expose":"webdav",
  "volume":{"name":"pv-001", "size":"1Gi"}},
  {"name":"lisa.surfsara.nl","type":"sshfs","expose":"webdav","caches":
[{"enabled":true,"size":"1Gib","location":"uva","retnetion-policy":{}}],
"user":"user","host":"lisa.surfsara.nl","path":"/nfs/scratch/user",
"password":"******"},
{"name":"pro.cyfronet.pl","type":"sshfs","expose":"webdav","caches":
[{"enabled":true,"size":"1Gib","location":"uva","retnetion-policy":{}}],
"user":"user","host":"pro.cyfronet.pl","path":"/net/archive/groups/plggprocess",
"password":"*****"}],
"logicContainers":
[{"name":"mydataStaging","type":"query"},
 {“name”:”myJupyter”,“type”:”jupyter”,”pass”:”*****”},
 {"name":"myDav","type":"webdav","user":"user","pass":"*****"},
 {"name":"myDavWithTokens","type":"webdav-jwt","users":{"user@domain.nl":
  {"name":"name","publicKey":"base64 public key"}}
 }
]
}
```


- **GET /api/v1/infrastructure** returns the exposed service of the infrastructure in JSON format. These services are publicly accessible and meant to be used by users and compute alike. In this example we expose two types of WebDAV services, one protected with username/password and another with web tokens to for the execution environment.


```
[{"type":"webdav","name":"test-001ht","ports":[32696],
  "host":"data01.process-project.eu"},
 {"type":"jupyter","name":"test-001-jupyter","ports":[32160],
  "host":"data01.process-project.eu"},
 {"type":"webdav-jwt","name":"test-001-jwt","ports":[31708],
  "host":"data01.process-project.eu"},
 {"type":"query","name":"test-001-query",
  "ports":[31681],"host":"data01.process-project.eu"},
 {"type":"token","header":"x-access-token","value":"****"}
]
```

- **DELETE /api/v1/infrastructure/:id** where *id* is the name of the infrastructure. In this example it would be *test-001*. This call tears down an infrastructure, deleting all running instances of the containers.
- **PUT /api/v1/user** This call is only accessible by administrator tokens and is used to register new users and create tokens for the users.
- **GET /api/v1/endpoints/:[NAME]** This call is meant to integrate with other PROCESS services such as the EE. It will accept external tokens and return interested endpoints requested by the external service.

5.2 Appendix B: Data query service API

- **GET /api/v1/list**: Authentication to this service is through tokens, which are generated specifically for the user infrastructure. N.B. these url endpoints reside within the user infrastructure and are user specific. This method call generates a list of files and their locations on the different adaptors. E.g.


```
{"sample_script.py":[{
   "filename":"/UC1/sample_script.py","basename":"sample_script.py",
   "lastmod":"Fri, 28 Dec 2018 18:47:10 +0000","size":284,
   "type":"file","mime":"text/x-script.python",
   "location":{"name":"lisa.surfsara.nl","host":"localhost","port":3003,
            "type":"webdav","mount":"/nfs/scratch/cushing"}
   },
   {
    "filename":"/Mock/sample_script.py","basename":"sample_script.py",
    "lastmod":"Tue, 27 Nov 2018 20:25:40 +0000", "size":284, "type":"file",
    "mime":"text/x-script.python",
    "location":{"name":"pro.cyfronet.pl","host":"localhost","port":3002,
    "type":"webdav","mount":"/net/archive/groups/plggprocess"}
   }
 ],
 "ucdemo-0.1.simg":[{
   "filename":"/UC1/ucdemo-0.1.simg","basename":"ucdemo-0.1.simg",
   "lastmod":"Tue, 16 Oct 2018 12:41:37 +0000", "size":1723113503,
   "type":"file", "mime":"application/octet-stream",
   "location":{"name":"pro.cyfronet.pl","host":"localhost","port":3002,
            "type":"webdav", "mount":"/net/archive/groups/plggprocess"}
   },
   {"filename":"/Mock/ucdemo-0.1.simg","basename":"ucdemo-0.1.simg",
    "lastmod":"Wed, 12 Sep 2018 08:10:04 +0000", "size":1727819807,
    "type":"file","mime":"application/octet-stream",
    "location":{"name":"pro.cyfronet.pl","host":"localhost","port":3002,
             "type":"webdav","mount":"/net/archive/groups/plggprocess"}
    }],
 "tensorflow-gpu.img":[{
   "filename":"/Mock/tensorflow-gpu.img","basename":"tensorflow-gpu.img",
   "lastmod":"Sun, 23 Sep 2018 14:34:20 +0000", "size":3923771423,
   "type":"file","mime":"application/octet-stream",
  "location":{"name":"pro.cyfronet.pl","host":"localhost", "port":3002,
           "type":"webdav","mount":"/net/archive/groups/plggprocess"}
 }],
"tensorflow-gpu_2.img":[{
   "filename":"/Mock/tensorflow-gpu_2.img","basename":"tensorflow-gpu_2.img",
   "lastmod":"Sun, 23 Sep 2018 16:30:52 +0000", "size":3945791519,
   "type":"file","mime":"application/octet-stream",
   "location":{"name":"pro.cyfronet.pl","host":"localhost","port":3002,
            "type":"webdav","mount":"/net/archive/groups/plggprocess"}
  }]
}
```


- **GET /api/v1/find/:id** Where id is the filename. This url will list the locations of the file in all adaptors E.g. /api/v1/find/ucdemo-0.1.simg and return two location on the same adaptor


```
[{"filename":"/UC1/ucdemo-0.1.simg","basename":"ucdemo-0.1.simg",
  "lastmod":"Tue, 16 Oct 2018 12:41:37 +0000","size":1723113503,
```

"type":"file","mime":"application/octet-stream",
"location":{"name":"pro.cyfronet.pl","host":"localhost","port":3002,
"type":"webdav","mount":"/net/archive/groups/plggprocess"}
},
{"filename":"/Mock/ucdemo-0.1.simg","basename":"ucdemo-0.1.simg",
"lastmod":"Wed, 12 Sep 2018 08:10:04 +0000", "size":1727819807,
"type":"file", "mime":"application/octet-stream",
"location":{"name":"pro.cyfronet.pl","host":"localhost","port":3002,
"type":"webdav","mount":"/net/archive/groups/plggprocess"}
}]

- **POST /api/v1/copy**: This URL will stage data between sites asynchronously and will notify the caller through a registered webhook.

[{“id”: “copy-001”, “from”: “pro.cyfronet.pl:/UC1/ucdemo-0.1.simg”, “to”: “lisa.surfsara.nl:/UC1/ucdemo-0.1.simg”, “webhook”: “http://…”}]

5.3 Appendix C: DataNet API

The API may be used to access the repository in an automated fashion. Below, some sample feature operations on the datasets are shown:

Request: *CREATE METADATA ENTRY*

```
$ curl -i -H "Content-Type: application/json" -X PUT https://<repository>/<collection>/<entry_id> -d
'{"property_1":"property_value_1", "property_2":"property_value_2" }'
```

Response

HTTP/1.1 201 Created

Request: *GET METADATA ENTRY*

$ curl https://<repository>/<collection>/<entry_id>

Response

```
{
        "_id": "<entry_id>",
        "property_1": "property_value_1",
        "property_2": "property_value_2
}
```

Request: *QUERY METADATA COLLECTION*

$ curl -G --data-urlencode "filter={'property_1':'property_value_1'}" https://<repository>/<collection>

Response

```
[
   {
        "_id": "<entry_id>",
        "property_1": "property_value_1",
        "property_2": "property_value_2
        }, ...
]
```

5.4 Appendix D: PROCESS data services infrastructure

Table 6 List of available PROCESS services - available/used hardware, computer addresses

LOBCDER	The main entry point for LOBCDER is http://data01.process-project.eu:30000 • WebDAV and other service endpoints for each user/use-case will be on high order ports on http://data01.process-project.eu:3XXXX. • Local WebDAV repository at UISAV is available at https://147.213.75.208/process/dav/ A set of template containers are also made available on docker hub; • WebDAV container recap/process-webdav • sshfs container recap/process-sshfs • API server container recap/process-core-infra • LOBCDER makes use of mongodb container as a database. Note: The code base is currently being hosted on GitHub • infrastructure https://github.com/recap/MicroInfrastructure.git • modified WebDAV server https://github.com/recap/jsDAV.git
DISPEL	While intended for just-in-time deployment in the LOBCDER micro-infrastructure or services run in Cloudify-managed containers, there is a static DISPEL deployment available for testing at 147.213.75.208:8080.
DataNet	Deployed at Cyfronet. Available endpoints: • REST API: https://md-api.process.cyfronet.pl/ • HAL Browser (UI): https://md.process.cyfronet.pl/ Note: DataNet will be deployed in every computing center by the end of the project
Cloudify	Cloudify manager is installed at IISAS at 147.213.76.115 (cloudify.ui.sav.sk). • TOSCA templates for execution of generic Docker, as well as Jupyter portal and DISPEL service are uploaded to the server. • The plugins for deploying services on Openstack and Kubernetes are installed, too. Docker images of the mentioned services in TOSCA are downloaded on the fly from DockerHub.