Forestry Thematic Exploitation Platform

Service Developer Guide

v2025.04

Contents

1 Introduction

1.1 Overview

One of the key features of Forestry Thematic Exploitation Platform (F-TEP) is the ability for users to create new processing services on the platform. The users can shape the services that F-TEP already offers or develop brand new services. This feature has also a collaborative aspect, as the developer can optionally share the service with others or make it public to all, giving other users the chance to use novel algorithms and processors on a satellite data tailored for their needs.

This document aims to give service developers the information required to successfully implement new services. The reader is assumed to be familiar with the basic functionality of the platform that is described in the User Manual, available at https://f-tep.com/usermanual.

1.2 Platform infrastructure

For effective service development, it as advantageous to understand how the F-TEP platform manages jobs launched by users. Instead of an interactive command prompt for writing and executing scripts, services are internally implemented as Docker images that are instantiated when a user launches a service. The system builds the Docker image based on the specification made by the service developer in *Dockerfile* and in the *workflow script* and any associated files. The Docker image contains full specification of a Linux virtual machine, including Linux distribution, installed programs, libraries, scripts, and environment variables. Service inputs and expected outputs are also specified.

When a user launches a service, i.e. creates a processing job, the Docker image that contains the service implementation is executed on a worker virtual machine. Each worker machine hosts at most two concurrent jobs, and the system automatically spawns new worker machines as needed. The configuration of a worker virtual machine is 8 virtual CPUs, 32 GB of RAM and 128 GB of SSD-type temporary workspace. Output product storage is not currently limited.

Note that specific configurations are possible. If a processing service requires specific capacity for CPU, RAM or storage, dedicated resources can be arranged. Please contact the Forestry TEP team for details.

1.3 Developer interface

All users have access to the service developer interface shown in Figure 2 below.

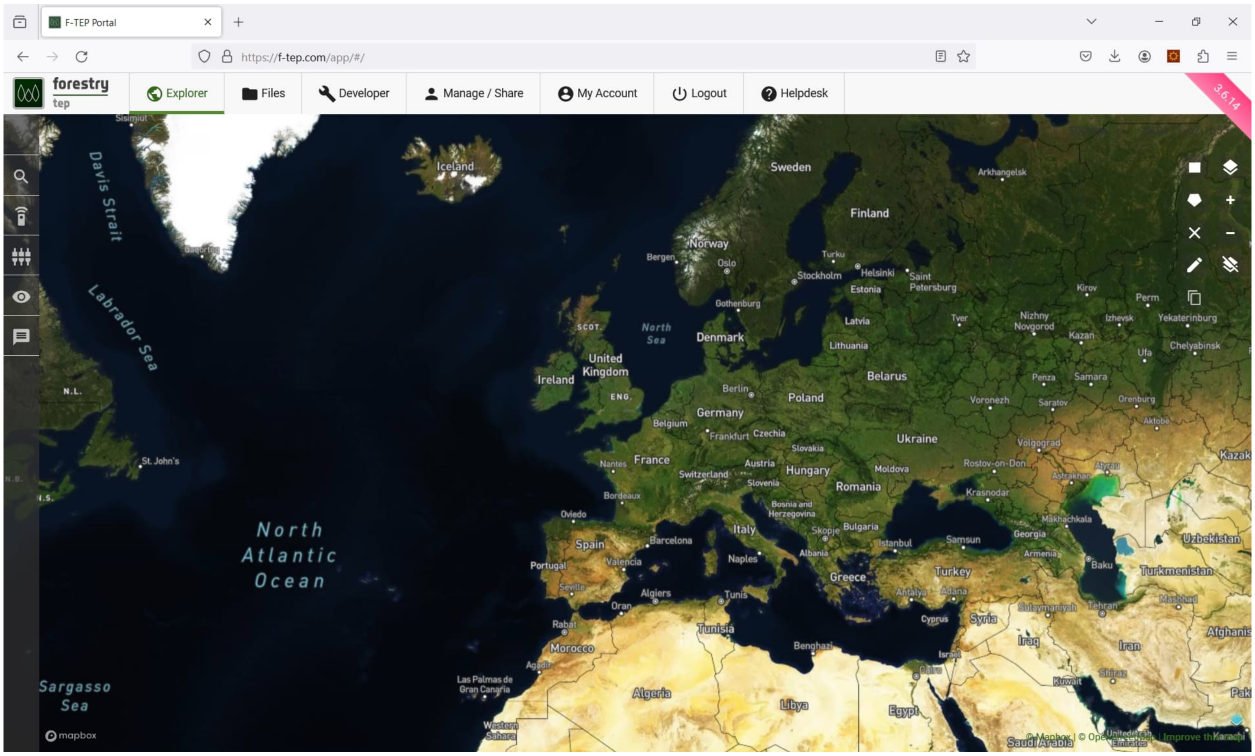

Figure 1 User view of F-TEP

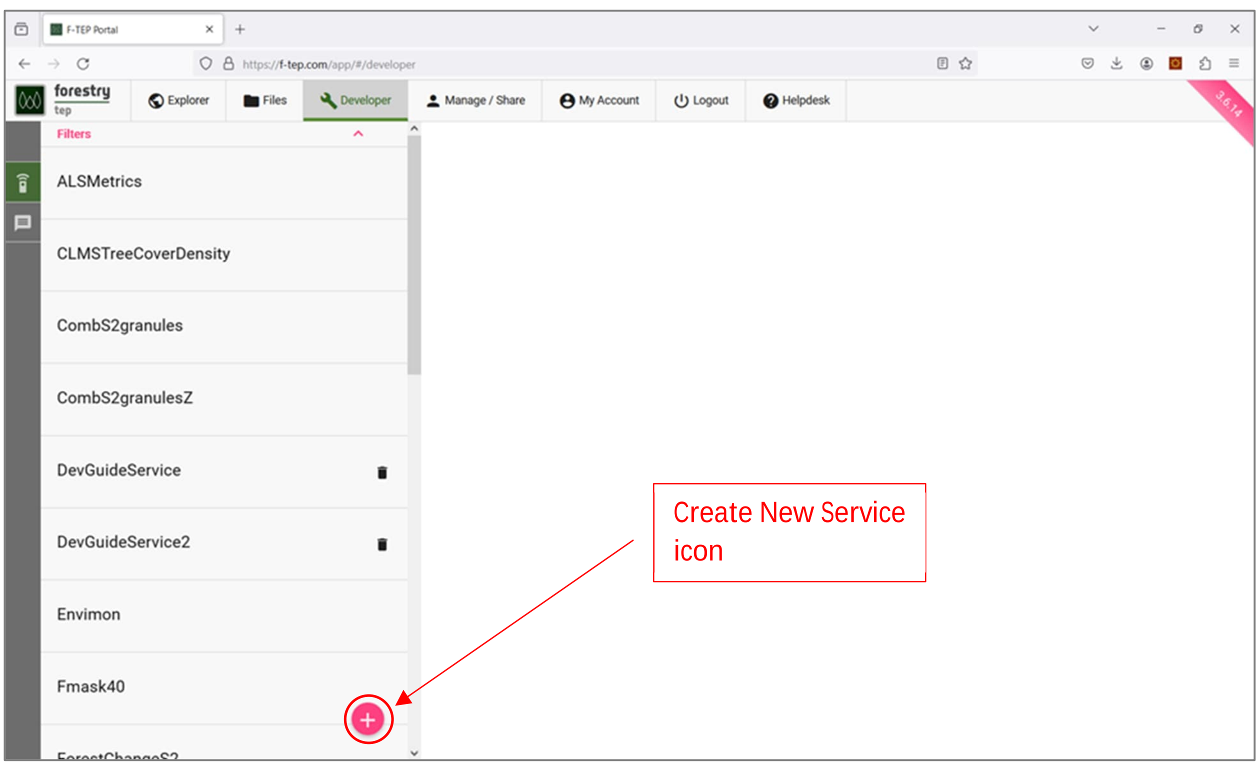

Figure 2 Developer view of F-TEP

The toolbar on the left side of the screen has two buttons:

 *Services*: toggles on and off the service list pane at the left side of the screen

 *Messages*: toggles on and off message list pane at the bottom of the screen

The service list in Figure 2 shows the services the user has access to. These will be made up of:

1. Core services, provided by F-TEP and made available to all
2. Own services, created by the user
3. Shared services, made visible to the user by other users

The definitions of existing services can be browsed by clicking on the service name on the list. Note that the by default service source files are not visible to other users, you can see the source files for your own services and for those for which source file visibility has been granted by the service developer.

The service list can be filtered by clicking on the *Filters* row on the top of the list, to open the options shown in Figure 3. The field *Search* is a case-insensitive free-text search from service names and descriptions. Other filtering options are service type and service ownership - either the services owned by the user, services shared with the user, or both.

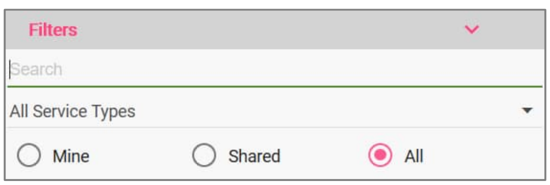

Figure 3 Service list filtering options

2 Service development

2.1 Creating a service

Development of a new service is started by clicking the *Create New Service* icon in the lower right corner of the service list (see Figure 2), which opens the dialog shown in Figure 4 below.

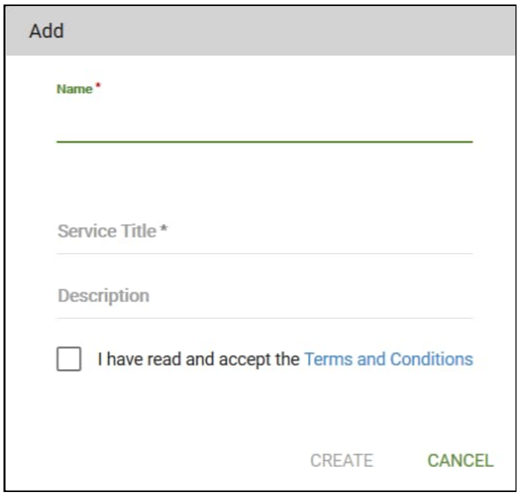

Figure 4 Add new service dialog

Service name, title and an optional description are specified in the dialog. Service name must start with a letter and only contain letters and numbers, no spaces or special characters. Service title and description are free format text fields. The maximum length of the description is 255 characters. If the length exceeds the limit, saving the service is not possible. The approval of Terms and Conditions checkbox needs to be checked to enable the *Create* button.

In case nothing appears to happen when clicking the *Create* button, please check the message list for an error message (*Could not create Service …*). A common cause of error (see Figure 5) is that service names need to be globally unique, i.e. another service may have reserved the name, even if the conflicting service is not visible to the user.

On successful creation of the service, the skeleton of the new service opens as shown in Figure 6.

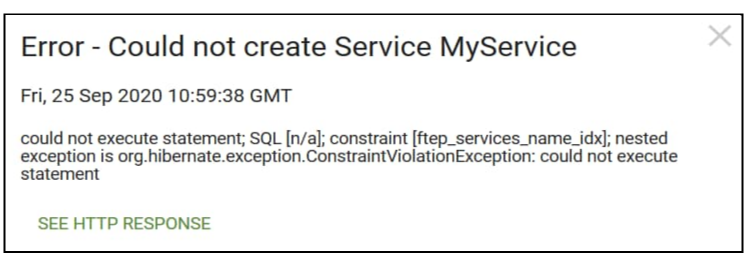

Figure 5 Error message when the service name is already in use

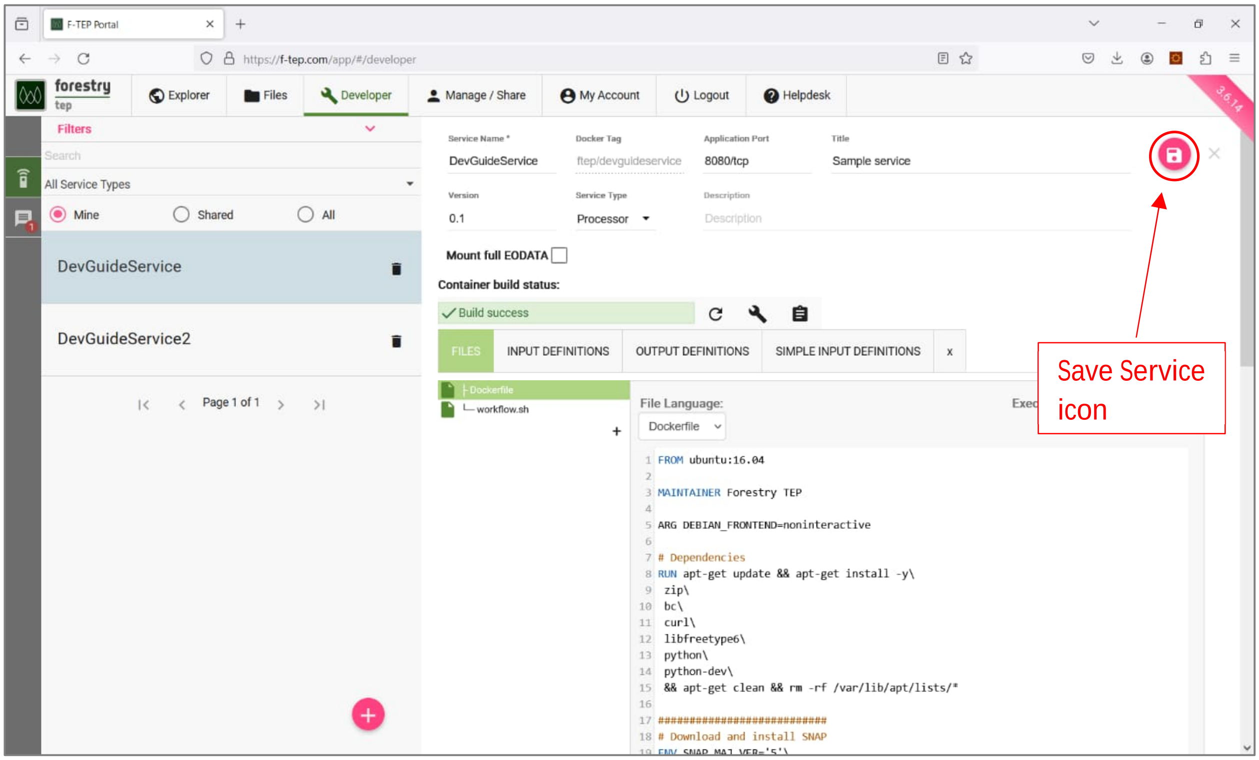

Figure 6 New service skeleton

On the top of the pane, there are fields for the service name, title and description, filled with values given in the creation dialog. The user can also specify a version number for the service and select the service type:

Processor	The default type for all services that do not have an interactive user interface.
Application	The type for interactive services with a graphical user interface.

Selecting the *Mount full EODATA* checkbox enables the service to access the CREODIAS EODATA archive as a directory /eodata from the code. With this option it is possible to create a service that gets as its inputs e.g. an area of interest and time span and the service can automatically retrieve and process all the available data (e.g. Sentinel-2 images). Please see the subsection on EODATA input for more details.

A service is defined with its files, input definitions and output definitions - see the following subsections.

Remember to save changes to the service after modifying any information by clicking the *Save Service* icon at the top right corner of the pane (see Figure 6).

2.2 Service files

The *FILES* tab lists all the files that form the service definition on the F-TEP server (Figure 7).

Figure 7 FILES tab

The skeleton for new services contains two generated files:

Dockerfile	The Dockerfile specifies the processing environment of the service, including the operating system and all installed programs and libraries. The default file provides SNAP and Orfeo toolbox on top of the Ubuntu 16.04 operating system, giving a solid basis for creating tailored EO data processing services. Other libraries can be installed as needed. It is possible to use the Docker image of service as the base image for another service by specifying the image in the FROM directive in the Dockerfile. The name of the image can be seen in the Docker tag field in the top row of the developer view of the service, e.g. ftep/gdalinfo. The tag is combined with F-TEP repository address, e.g. FROM ftep-docker-registry:5000/ftep/gdalinfo There are also some predefined base images available, please see chapter 5.
workflow.sh	This is the executable script that is run by default when the service is run, i.e. the service procedure is defined in this file. The default file demonstrates the expected directories for input and output data, and how to read the input parameter values. If the service developer prefers an alternative scripting language, workflow.sh can be updated to call another script, or the ENTRYPOINT parameter in the Dockerfile may be changed.

New text files can be added to the service via the plus sign below the file list.

In case the service requires binary files, commands to download them should be added in the Dockerfile. An example is given in Figure 20. Downloading of text files can also be specified in the Dockerfile, if e.g. an external software repository is used. The files are downloaded at the time of building the Docker image. The files that are added to the F-TEP service are visible to the owner of the service and to all the users the service source code is shared with.

Remember to save the service after modifying each file.

The service has to be built after its files have been modified for the changes to take effect. The container build status should show 'Build required' in this case. The build is initiated with the wrench icon next to the status field, after which the status is ‘Build requested’. The status field has to be refreshed manually with the arrow icon next to it. The build log opens in a new browser tab by clicking to notepad icon. The log tab can be refreshed manually with the web browser’s refresh function to follow the progress. The log icon is not visible it the service has not been built yet.

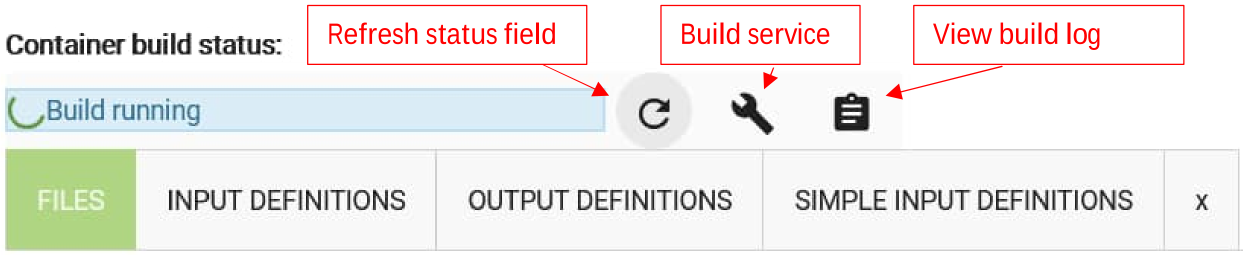

Figure 8 Service build controls

NOTE:

If the build process does not start in a few minutes please cancel the build are try again.

If the build process fails immediately and there is nothing in the log the most probable cause is a syntax error in the Dockerfile. The error messages from these are unfortunately not visible in the user interface so the best option is to try commenting Dockerfile commands to find the cause.

DEVELOPER HINT:

The virtual machine in which the service is run has a limited capacity of fast SSD disk. For large data sets the platform makes available a large network disk share that can be accessed from the services at

/home/worker/procDir

While this folder has very large capacity it is much slower to access, the speed difference to the local SSD can be 10-15x. It is thus always better to use the local disk if possible.

2.3 Input definitions

The service inputs are defined in the *INPUT DEFINITIONS* tab (Figure 9). New inputs are defined with the *Add* button. Remember to save the service after modifying inputs. These inputs are those that the service user defines when launching a job.

Figure 9 INPUT DEFINITIONS tab

The following fields can be defined for each input parameter:

Table 1 Fields for specifying service input parameters

Core	
ID	Required field. Tag used for the parameter in parameter file. It should not contain spaces or special characters.
Title	Required field. Name of the parameter shown to the user.
Description	Description of the parameter, shown in the user interface below the parameter value.
Data Type	
Field Type LITERAL	
Data Type	String/integer/double
Allowed Values	An enumeration of allowed values for the parameter, if applicable
Field Type COMPLEX	
Mime Type	Mime type of the input
Extension	File extension of the input
As Reference	When set to true, the parameter value is interpreted as an input data URL. On launching the service, attempt is made to retrieve the specified input data and provide it within the service execution environment.
Advanced	
Default value	The default value for the parameter for enumerated values.
Minimum Occurrences	Minimum number of values. Set to 0 for optional parameters and to 1 (or above) for required parameters.
Maximum Occurrences	Maximum number of values, set to 1 or above.
Data Reference	If True, enables the drag-and-drop behavior for input data.
Search Parameter	If True, enables to launch the service in systematic processing mode where a number of jobs is launched for the results from a data query that is executed periodically. Each job has one input file from the data query results as this parameter and all other parameters are the same for all the jobs.

	This option is visible only if Data Reference is True.
Enable Parallel Processing	If True and the number of inputs is greater than one, the job will be split to multiple parallel sub-jobs. Each sub-job processes one of the given parameter values.

Note that specification of the Data Type does not change the way the parameter values are passed to the service. The parameter file contains key-value pairs and all values are strings. The system attempts to download all parameter values that are in the URL format, regardless of the data type parameters.

All input parameters are available to the service in the parameter file:

/home/worker/workDir/FTEP-WPS-INPUT.properties

A sample parameter file with one data reference parameter and one string valued parameter is shown in Figure 10.

```
myparam="A string value"
image="sentinel2:///S2B_MSIL1C_20200928T042709_N0209_R133_T49WEN_20200928T053528"
```

Figure 10 A sample parameter file with one string valued parameter (myparam) and one data reference parameter (image)

The platform makes the data inputs available under this folder:

/home/worker/workDir/inDir

Each input is placed in a subfolder named with the ID of the parameter. The resulting input folder structure for the parameter file in Figure 10 is shown in Figure 11 below. (This sort of illustration can be viewed on the platform in the file dialog of a tool with a graphical user interface, such as SNAP or QGIS3.) Note that the folder with its name starting with S2B… is a symbolic link and that the input folder is in read-only mode. The Sentinel-2 input folders are in the .SAFE format, although the folder name does not contain the suffix.

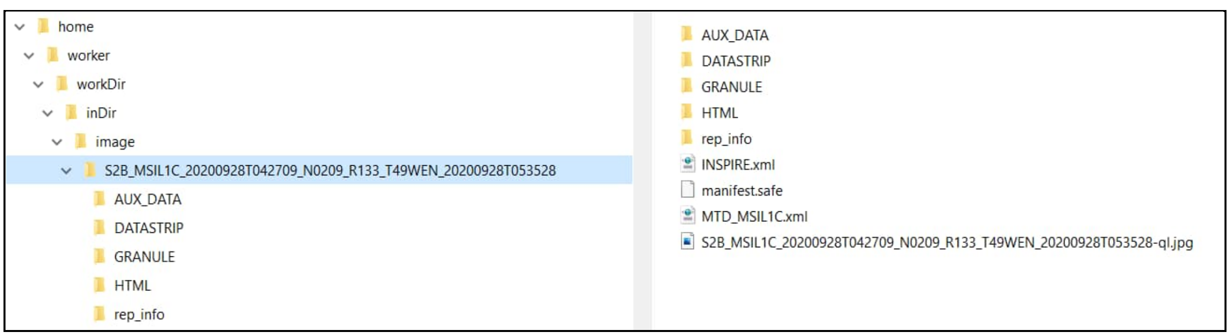

Figure 11 Input folder structure. Folder S2B_MSIL1C_20200928T042709_N0209_R133_T49WEN_20200928T053528 is a symbolic link.

Zipped input files are automatically unzipped to a subfolder with the name of the zip file (without .zip extension). Other compressed data formats are not automatically uncompressed.

If a parameter value is in the URL format, the system attempts to download the data into the input data folder. The job will fail if the download fails. The file will be downloaded into a subfolder named with the name of the file without the file type suffix, and if the file is a zip archive, it is unpacked into that folder.

2.4 EODATA input

If the service's *Mount full EODATA* checkbox is selected the CREODIAS data is accessible from the service code in directory /eodata. Two subdirectory levels of the directory are shown in Figure 12 below to give an idea of the data structure (situation 10.1.2024). The up-to-date directory structure can be browsed using the QGIS3 service.

auxdata - building-height-germany - CARD_BS_MC - CopDEM - CopDEM_COG - CSI-SRTM-DEM - ECOSTRESS - Elevation-Tiles - ESA_WORLD_COVER - GLC_Germany_2018 - MERIS - orbits - S2GLC - S2GLC-Poland - S2_L3_MOSAIC_120 - S2_L3_WASP - SRTMGL1 C3S - orders - static CAMS - GFAS - GLOBAL - GLOBAL_ADDITIONAL - WMO_ESSENTIAL CEMS - RM CLMS - Global - Imagery_and_reference_data - Local - Pan-European	CMEMS - NRT - REP Envisat - Envisat-Product-List.txt - Meris Envisat-ASAR - ASAR Global-Mosaics - ESA-WorldCover - Sentinel-2 Jason-3 - GDR - IGDR - OGDR Landsat-5 - TM Landsat-7 - ETM - Landsat-7-Product-List.txt Landsat-8 - Landsat-8-Product-List.txt - OLI - OLI_TIRS - TIRS	Sentinel-1 - AUX - SAR Sentinel-1-COG - SAR Sentinel-1-RTC - SAR Sentinel-2 - AUX - MSI Sentinel-3 - AUX - OLCI - SLSTR - SRAL - SYNERGY Sentinel-5P - AUX - TROPOMI Sentinel-6 - AMR-C - AUX - P4 SMOS - L1B - L1CL - L1CS - L2OS - L2SM

Figure 12 /eodata directory contents

The use of EODATA input is very useful if the service requires a lot of input data that could be automatically read from the /eodata directory in the service code. In this way the service user does not need to find and specify all the data as user inputs for the job. It also makes running the job faster as all user specified data inputs are first copied to a network disk for the job (as discussed in the Input definitions subsection above) and if there are hundreds of large raster files that takes time. Using EODATA input it is possible to create a service where the user specifies as inputs e.g. an area of interest and time span and the service developer takes care of reading and processing all the corresponding data (e.g. a set of Sentinel-2 images).

CREODIAS search interface can be used to locate the relevant data, e.g. by executing a query like the one shown in Figure 13. The query returns a JSON output which can be processed in the service code to locate the images. The returned metadata contains the directory of the data in the S3Path attribute. See https://creodias.docs.cloudferro.com/en/latest/eodata/EOData-Catalogue-API-Manual-on-Creodias.html for details. Python based services can also utilize the search method available in the VTT provided ftep-util Python module.

```
https://datahub.creodias.eu/odata/v1/Products?
$filter=(
(Attributes/OData.CSC.DoubleAttribute/any(i0:i0/Name eq 'cloudCover' and i0/Value le 90))
and (ContentDate/Start ge 2023-11-11T00:00:00.000Z
and ContentDate/Start le 2023-11-17T23:59:59.999Z)
and (Online eq true)
and (OData.CSC.Intersects(Footprint=geography'SRID=4326;POLYGON ((11.153066 48.499961, 12.318943
48.6006, 12.318943 47.891918, 11.051686 47.857926, 11.153066 48.499961))'))
and (Collection/Name eq 'SENTINEL-2'))
&$expand=Attributes&$expand=Assets
&$orderby=ContentDate/Start asc&$top=20
```

Figure 13 CREAODIAS search example (split to lines for readability)

2.5 Output definitions

The service outputs are defined in the *OUTPUT DEFINITIONS* tab (Figure 14). New outputs are defined with the *Add* button. Remember to save the service after modifying outputs.

Figure 14 OUTPUT DEFINITIONS tab

The following fields can be defined for each output:

Core			
	ID		This is the tag used for the output. It should not contains spaces or special characters.
	Title		This is the name of the output.
	Description		This is the description of the output.
Data Type			
	Field Type LITERAL		Not applicable, the services produce file outputs.
	Field Type COMPLEX		
		Mime Type	Mime type of the output
		Extension	Extension of the output

As Reference	Not applicable.
Advanced	
Default value	Not applicable.
Minimum Occurrences	Minimum number of values. Set to 0 for optional outputs and to 1 (or above) for required outputs.
Maximum Occurrences	Maximum number of outputs, set to 1 or above. It is an error if the service generates more files for this output.

Note that field type LITERAL or a Default value are not applicable with file outputs. However, the system does not enforce output type checks.

The service should place all its outputs under this folder:

/home/worker/workDir/outDir

Each output should be stored in a subfolder named with the ID of the output, e.g. output with ID myoutput should be placed in this folder:

/home/worker/workDir/outDir/myoutput

2.6 Templated service parameters

Service developers can optionally use an advanced service configuration to access powerful platform functionality. Via specifying templates, the service developer can define ways for the system to assist the eventual service user in e.g. automatic selection of the input data.

For a specific guidance on the feature, please see the online documentation:

https://github.com/vttresearch/f-tep/blob/development/docs/templated-service-parameters.md

As a simple usage scenario, the feature can be used to create a service with simplified parameters for basic users, while also providing an option to see the full parameter list when the user selects Advanced Mode in the service workspace. For this, follow these steps:

1. The *SIMPLE INPUT DEFINITIONS* tab is by default generated when the service is saved for the first time and will at first be empty. Remove the empty definitions with the X icon shown in Figure 15.
2. Click the + sign next to the *OUTPUT DEFINITIONS* tab to create the *SIMPLE INPUT DEFINITIONS* tab. This generates a template to fill all parameters specified in the *INPUT DEFINITIONS* tab. An example is shown in Figure 16.
3. Save the service by clicking the *Save Service* icon at the top right corner of the pane before continuing! If parameters are removed in the *SIMPLE INPUT DEFINITIONS* tab before saving the service, they will be removed from *INPUT DEFINITIONS* tab as well! (F-TEP version 3.5.7)
4. Remove the desired input parameters from the SIMPLE INPUT DEFINITIONS tab by clicking the *Remove Row* icon of the parameters.
5. Fill a default value for the removed parameters in the template, as shown in Figure 17.
6. Click the *VALIDATE TEMPLATE* button at the bottom of the pane. The service cannot be saved before the changes are validated.
7. Save the service.

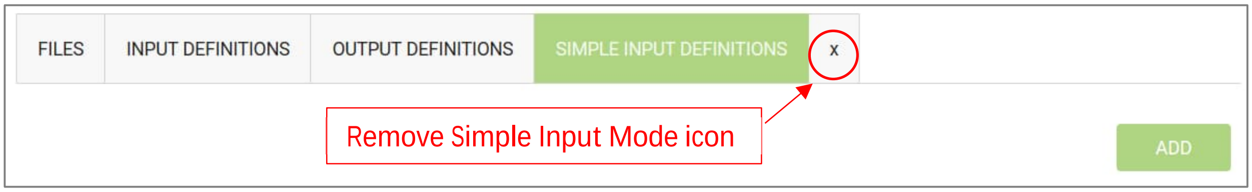

Figure 15 Simple Input Definitions tab

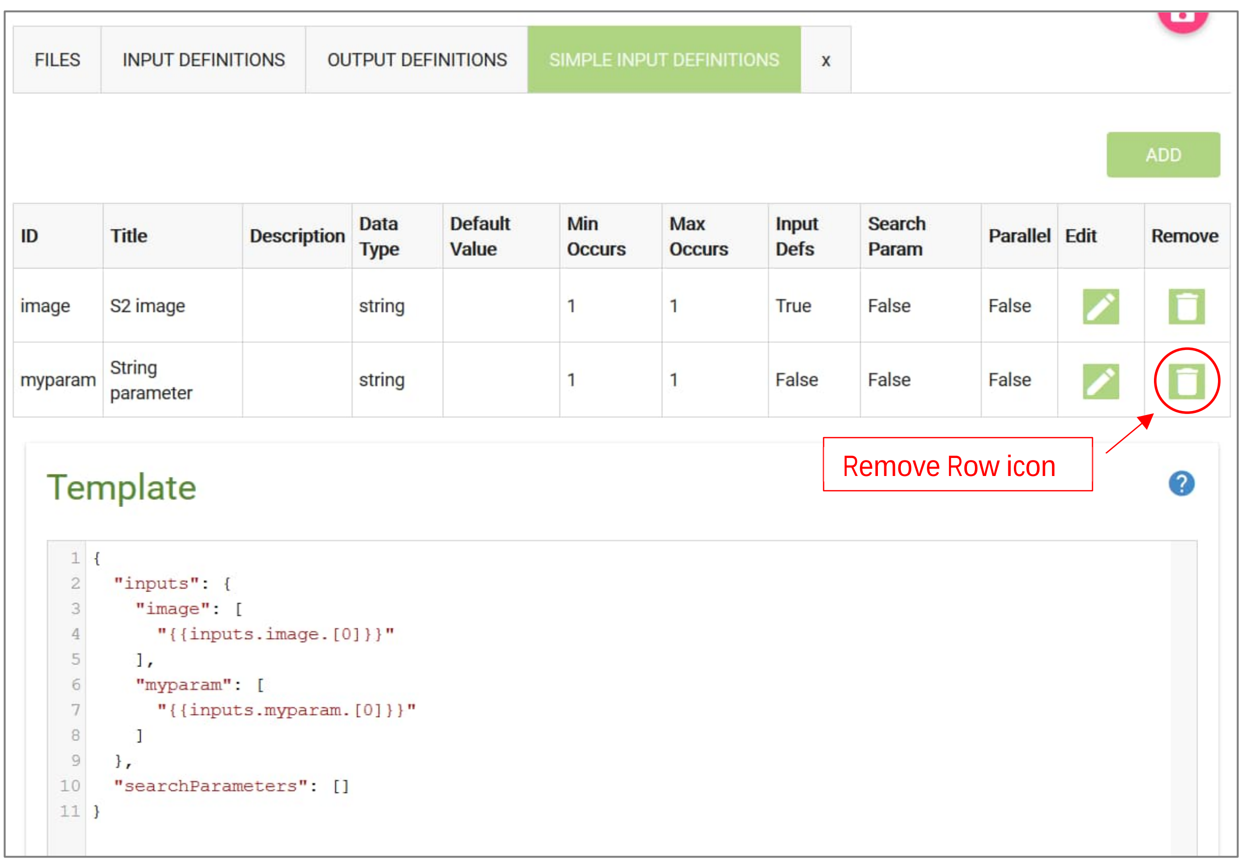

Figure 16 Simple input definitions example

Template

```
{
  "inputs": {
    "image": [
      "{{inputs.image.[0]}}"
    ],
    "myparam": [
      "20"
    ]
  },
  "searchParameters": []
}
```

Figure 17 Modified template

2.7 Systematic processing

All F-TEP services can optionally be configured to run in "Systematic" mode, allowing automated, scheduled detection of input data and generation of processing jobs. To learn about this feature, please see the online documentation:

https://github.com/vttresearch/f-tep/blob/development/docs/systematic-processing.md

To enable systematic processing for a service at least one input parameter must be specified as search parameter, see section 0

DEVELOPER HINT:

The virtual machine in which the service is run has a limited capacity of fast SSD disk. For large data sets the platform makes available a large network disk share that can be accessed from the services at

/home/worker/procDir

While this folder has very large capacity it is much slower to access, the speed difference to the local SSD can be 10-15x. It is thus always better to use the local disk if possible.

Input definitions.

2.8 Developer hints

This section provides miscellaneous hints for service developers.

DEVELOPER HINT:

While developing a service there might be a need to check the contents of the input folder. An easy way is to add the following line to workflow.sh, and define an output with ID *listing*:

```
ls -R -l -L ${IN_DIR} > ${OUT_DIR}/listing/listing.txt
```

The log lines in the user interface are not strictly in the correct order, thus using file output is easier.

The contents of the parameter file are shown in the service log, if the default behavior is not changed (note again that the lines are not in the correct order):

```
++ image=sentinel2:///S2B_MSIL1C_20200928T042709_N0209_R133_T49WEN_20200928T053528

++ myparam=String
```

DEVELOPER HINT:

The development process is faster when done on a local machine, which avoids the need to create Docker images. A quick solution is to create scripts or programs that take input folder, output folder, and parameter file path as command line arguments and set the required input and output folder structure on a local machine. Deploying such scripts and programs on the F-TEP platform is fast and straightforward.

DEVELOPER HINT:

It can be a good idea to print some information about the execution environment in the log, e.g. program and library versions - as in Figure 21 - as the default versions can be different from the development environment. If needed, particular versions can be specified in the Dockerfile installation commands.

3 Service sharing

3.1 Sharing a service to selected users

At any time, the developer can share the service to a defined group of users, such as a set of colleagues or customers. This allows the permitted users to see and to execute the service. Additionally, permission can be given to modify the service as well.

Groups are created via the Groups icon in the Manage/Share interface. Please see the User Manual for details.

Service sharing is done in the Manage/Share tab of the user interface, shown in Figure 18. First, select the service, with the help of the list filters if needed.

To share a service, click on the Share icon near the right edge of the screen, which opens the dialog shown in Figure 19. In this dialog, select the group with which to share the service. Also, define the permissions for the group regarding this service:

- SERVICE_USER: can run the service, does not see its source code
- SERVICE_READONLY_DEVELOPER: can read the source code
- SERVICE_DEVELOPER: can modify the source code
- SERVICE_OPERATOR: has full control of the service

Note that a user with a developer role cannot run the service. Access to both source code and service execution can be given by sharing the service to two groups with different permissions.

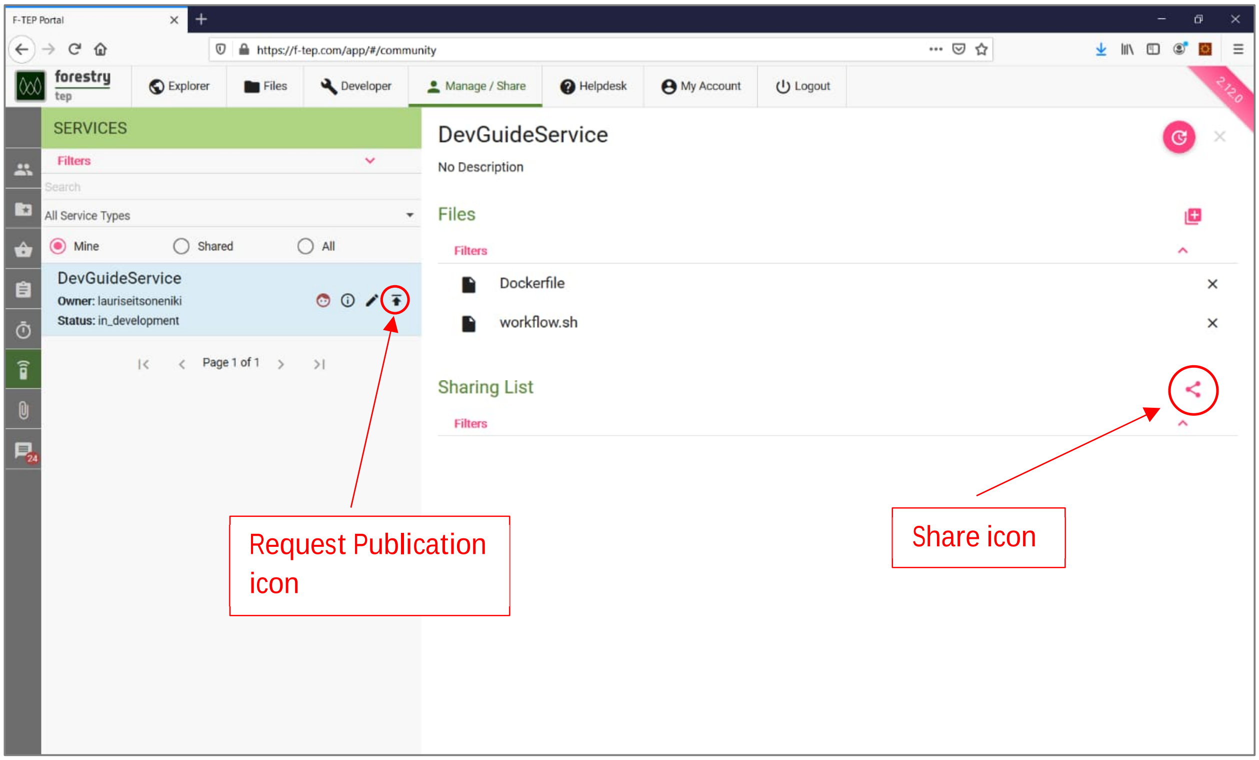

Figure 18 Service sharing interface

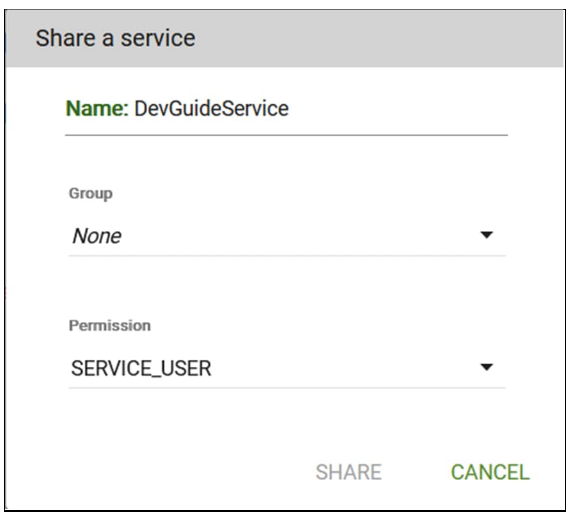

Figure 19 Service sharing dialog

3.2 Publishing a service

When a service is ready to be used by a wide audience, the developer can opt to publish it to the benefit of all users, among the other services on Forestry TEP. As opposed to service sharing, published services are visible to all users. Publishing a service does not make its source code visible to other users.

To request publication, enter the Manage/Share interface and click on the Request Publican icon in the left pane of the service list, shown in Figure 18. Publishing is not instantaneous, as new services will be reviewed by the F-TEP team to verify that they are safe and appropriate. After the review, the new service will be made visible to all users in the Explorer interface and via the API.

4 Examples

4.1 Simple service with a downloaded binary file and Python scripts

This service uses a Python script that could call a downloaded binary file during processing. It has one output definition (*result*) and it produces a zip file of its processing outputs as the final output. The script file my_script.py that is called in workflow.sh has to be defined in the *FILES* tab of the service.

```
FROM ubuntu:18.04

LABEL maintainer lauri.seitsonen@vtt.fi

# Install required programs
RUN apt-get update && apt-get install -y\
 zip\
 curl\
 python3\
 python3-pip\
 python3-dev\
 python3-gdal\
 gdal-bin\
 libgdal-dev

# Prepare processor script
RUN mkdir -p /home/worker/processor

# Download binary files. Note that curl needs to be installed before this.
RUN curl -L https://www.dropbox.com/s/sq2wwekmdwhz/my_binary?dl=1 > /home/worker/processor/my_binary

# Add execute rights to the downloaded binary file
RUN chmod a+x /home/worker/processor/*

# Copy all files defined in the FILES tab of the developer interface to the image
COPY * /home/worker/processor/

# Configure environment
ENV PATH=/home/worker/processor:${PATH}

# The script that is executed when the Docker image is instantiated
ENTRYPOINT ["/home/worker/processor/workflow.sh"]
```

Figure 20 Dockerfile with binary file download

```
#!/usr/bin/env bash

set -x

# F-TEP service environment
WORKFLOW=$(dirname $(readlink -f "$0"))
WORKER_DIR="/home/worker"
IN_DIR="${WORKER_DIR}/workDir/inDir"
OUT_DIR="${WORKER_DIR}/workDir/outDir"
WPS_PROPS="${WORKER_DIR}/workDir/FTEP-WPS-INPUT.properties"
PROC_DIR="${WORKER_DIR}/procDir"
TIMESTAMP=$(date --utc +%Y%m%d_%H%M%SZ)

# Temporary file storage
mkdir -p ${PROC_DIR}

# Set GDAL_DATA variable
export GDAL_DATA=$(gdal-config --datadir)

# Input files available under ${IN_DIR}
# Output files to be written to ${OUT_DIR}/<parameter>/<outputfilename>
mkdir -p ${OUT_DIR}/result

# Print information useful for debugging
cat ${WPS_PROPS}
python3 --version
pip3 list
printenv
gdal_translate --version

# Execute the Python script
python3 ${WORKFLOW}/my_script.py ${IN_DIR} ${PROC_DIR} ${WPS_PROPS}
rc=$?; if [[ $rc != 0 ]]; then
  echo "Workflow failed (exit code $rc)"
  exit $rc
fi

# Set read access to the generated files in ${PROC_DIR}
chmod -R a+r ${PROC_DIR}/*

# Make a zip file of all the files the script created in ${PROC_DIR}
cd ${PROC_DIR}
zip -r -fz ${OUT_DIR}/result/my_result_zip.zip *
```

Figure 21 workflow.sh for a service that uses Python scripts

4.2 A service using Python and GDAL

```
FROM ubuntu:18.04

LABEL maintainer lauri.seitsonen@vtt.fi

# Dependencies
RUN apt-get update && apt-get install -y\
 zip\
 curl\
 bc\
 libfreetype6\
 gdal-bin\
 libgdal-dev\
 python3\
 python3-dev\
 python3-pip\
 python3-gdal\
 software-properties-common\
 && apt-get clean && rm -rf /var/lib/apt/lists/*

RUN pip3 install GDAL numpy rasterio fiona pyproj shapely geopandas weightedstats sklearn scipy

RUN mkdir -p /home/worker/processor

# Prepare processor script
COPY * /home/worker/processor/

# Configure environment
ENV PATH=/home/worker/processor:${PATH}

ENTRYPOINT ["/home/worker/processor/workflow.sh"]
```

Figure 22 Dockerfile for a service that uses Python and GDAL

```
#!/usr/bin/env bash

set -x -e

# F-TEP service environment
WORKFLOW=$(dirname $(readlink -f "$0"))
WORKER_DIR="/home/worker"
IN_DIR="${WORKER_DIR}/workDir/inDir"
OUT_DIR="${WORKER_DIR}/workDir/outDir"
WPS_PROPS="${WORKER_DIR}/workDir/FTEP-WPS-INPUT.properties"
PROC_DIR="${WORKER_DIR}/procDir"
TIMESTAMP=$(date --utc +%Y%m%d_%H%M%SZ)

export GDAL_DATA=$(gdal-config --datadir)

# Input parameters available as shell variables
source ${WPS_PROPS}

export PYTHONIOENCODING=utf8

# Temporary file storage
mkdir -p ${PROC_DIR}
# Service output
mkdir -p ${OUT_DIR}/output_shapefile

# Input files available under ${IN_DIR}
# Output files to be written to ${OUT_DIR}/<parameter>/<outputfilename>
cd ${WORKFLOW}
python3 sample_images_by_polygons_v3_mt.py ${IN_DIR} ${PROC_DIR} ${WPS_PROPS}
rc=$?; if [[ $rc != 0 ]]; then
  echo "Workflow failed (exit code $rc)"
  exit $rc
fi

chmod -R a+r ${PROC_DIR}/*
cd ${PROC_DIR}

# Make a zip file
if [ -z "${output_shapefile_name}" ]
then
  output_shapefile_name="shape_${TIMESTAMP}.zip"
else
  output_shapefile_name="${output_shapefile_name}_${TIMESTAMP}.zip"
fi
zip -r -fz ${OUT_DIR}/output_shapefile/${output_shapefile_name} *
```

Figure 23 workflow.sh

4.3 A service using Python and R

```
FROM ubuntu:18.04

LABEL maintainer lauri.seitsonen@vtt.fi

ENV DEBIAN_FRONTEND=noninteractive

# Dependencies
RUN apt-get update && apt-get install -y\
 software-properties-common\
 && apt-get clean && rm -rf /var/lib/apt/lists/*

RUN add-apt-repository ppa:ubuntugis/ubuntugis-unstable

RUN apt-get update && apt-get install -y\
 zip\
 curl\
 bc\
 libfreetype6\
 python3\
 python3-dev\
 python3-pip\
 python3-gdal\
 gdal-bin\
 libgdal-dev\
 libgeos++-dev\
 libudunits2-dev\
 libproj-dev\
 libx11-dev\
 libgl1-mesa-dev\
 libglu1-mesa-dev\
 libfreetype6-dev\
 libxt-dev\
 libfftw3-dev\
 software-properties-common\
 && apt-get clean && rm -rf /var/lib/apt/lists/*

# Install Python libraries depending on the needs
RUN pip3 install GDAL numpy==1.18.1 pyproj==2.6.1.post1 pandas==1.1.1

# Install R
RUN gpg --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
E298A3A825C0D65DFD57CBB651716619E084DAB9 &&\
 gpg -a --export E298A3A825C0D65DFD57CBB651716619E084DAB9 | apt-key add - &&\
 add-apt-repository 'deb https://cloud.r-project.org/bin/linux/ubuntu bionic-cran35/' &&\
 apt-get update && apt-get install -y\
 r-base\
 r-base-core\
 r-recommended
# Install remotes to enable easy library installation
RUN Rscript -e 'install.packages("remotes")'
```

```
# Install required R libraries
RUN Rscript -e 'library(remotes); install_version("raster", version="3.0.12", upgrade="never")'
RUN Rscript -e 'library(remotes); install_version("sp", version="1.4.1", upgrade="never")'
RUN Rscript -e 'library(remotes); install_version("data.table", version="1.12.8", upgrade="never")'
RUN Rscript -e 'library(remotes); install_version("future", version="1.16.0", upgrade="never")'
RUN Rscript -e 'library(remotes); install_version("gdalUtils", version="2.0.3.2", upgrade="never")'
RUN Rscript -e 'library(remotes); install_version("glue", version="1.3.2", upgrade="never")'
RUN Rscript -e 'library(remotes); install_version("lazyeval", version="0.2.2", upgrade="never")'
RUN Rscript -e 'library(remotes); install_version("Rcpp", version="1.0.4", upgrade="never")'
RUN Rscript -e 'library(remotes); install_version("RCSF", version="1.0.2", upgrade="never")'
RUN Rscript -e 'library(remotes); install_version("rgeos", version="0.5.2", upgrade="never")'
RUN Rscript -e 'library(remotes); install_version("rgdal", version="1.4.8", upgrade="never")'
RUN Rscript -e 'library(remotes); install_version("rgl", version="0.100.50", upgrade="never")'
RUN Rscript -e 'library(remotes); install_version("sf", version="0.9.0", upgrade="never")'
RUN Rscript -e 'library(remotes); install_version("BH", version="1.72.0.3", upgrade="never")'
RUN Rscript -e 'library(remotes); install_version("RcppArmadillo", version="0.9.850.1.0", upgrade="never")'
RUN Rscript -e 'library(remotes); install_version("rlas", version="1.3.5", upgrade="never")'
RUN Rscript -e 'library(remotes); install_version("lidR", version="2.2.3", upgrade="never")'

RUN Rscript -e 'install.packages("unix")'

RUN mkdir -p /home/worker/processor

# Copy all needed files
COPY *.csv /home/worker/processor/
COPY *.py /home/worker/processor/
COPY *.R /home/worker/processor/
COPY *.sh /home/worker/processor/

# Download and install ftep_util wheel to use its Python functions
RUN curl -L https://f-tep.com/sites/default/files/ftep_util-0.0.1-py3-none-any.whl > /home/worker/ftep_util-0.0.1-
py3-none-any.whl
RUN pip3 install /home/worker/ftep_util-0.0.1-py3-none-any.whl

########################
# Configure environment
ENV PATH=/home/worker/processor:${PATH}

ENTRYPOINT ["/home/worker/processor/workflow.sh"]
```

Figure 24 Dockerfile

```
#!/usr/bin/env bash

set -x -e

# F-TEP service environment
WORKFLOW=$(dirname $(readlink -f "$0"))
WORKER_DIR="/home/worker"
IN_DIR="${WORKER_DIR}/workDir/inDir"
OUT_DIR="${WORKER_DIR}/workDir/outDir"
WPS_PROPS="${WORKER_DIR}/workDir/FTEP-WPS-INPUT.properties"
PROC_DIR="${WORKER_DIR}/procDir"
TIMESTAMP=$(date --utc +%Y%m%d_%H%M%SZ)

export GDAL_DATA=$(gdal-config --datadir)

# Input parameters available as shell variables
source ${WPS_PROPS}

export PYTHONIOENCODING=utf8

# Print information useful for debugging
cat ${WPS_PROPS}
python3 --version
pip3 list
printenv
gdal_translate --version
Rscript --version

# Service output
mkdir -p ${OUT_DIR}/output

# Input files available under ${IN_DIR}
# Output files to be written to ${OUT_DIR}/<parameter>/<outputfilename>

cd ${WORKFLOW}

python3 main.py ${IN_DIR} ${PROC_DIR} ${OUT_DIR}/output ${WPS_PROPS}
rc=$?; if [[ $rc != 0 ]]; then
  echo "ERROR: python3 main.py failed (exit code $rc)"
  exit $rc
fi
```

Figure 25 workflow.sh. Here the handling of parameters is done within the Python script and Rscript is called from Python.

```
retcode = os.system('Rscript my_script.R')
if retcode != 0:
    print('ERROR: Rscript my_script.R failed (exit code ' + str(retcode) + ')')
    exit(2)
```

Figure 26 Snippet to call Rscript from Python

4.4 Python utility library for parameter handling and input file detection

The names for input files that are listed in the service input parameters are not necessarily the names of the files on the disk. We provide ftep_util library to make parameter and input management in Python easier. The library can be taken into use with the following snippet in Dockerfile

```
# Download and install ftep_util wheel to use its Python functions
RUN curl -L https://f-tep.com/sites/default/files/ftep_util-0.0.11-py3-none-any.whl > /home/worker/ftep_util-0.0.10-py3-none-any.whl
RUN pip3 install /home/worker/ftep_util-0.0.11-py3-none-any.whl
```

Figure 27 Dockerfile snippet to install ftep_util library

Documentation of the library is available at https://f-tep.com/sites/default/files/ftep_util.html. Sample usage is given below in Figure 28.

The ftep_util library is preinstalled in the following service base images:

- ftep-docker-registry:5000/ftep/ubuntu_python:20.04_3.10.13
- ftep-docker-registry:5000/ftep/ubuntu_python:20.04_3.9.18

```
import argparse
import os
import ftep_util as ftep

if __name__ == '__main__':
	parser = argparse.ArgumentParser(description='My script')
	parser.add_argument('indir', help='Input data directory')
	parser.add_argument('procdir', help='Processing directory')
	parser.add_argument('outdir', help='Output directory')
	parser.add_argument('input', help='Input parameter file')
	args = parser.parse_args()
	if (not os.path.isdir(args.indir)):
		print('ERROR: Invalid input data directory ' + args.indir)
		exit(1)
	if (not os.path.isdir(args.procdir)):
		print('ERROR: Invalid processing directory ' + args.procdir)
		exit(1)
	if (not os.path.isdir(args.outdir)):
		print('ERROR: Invalid output directory ' + args.procdir)
		exit(1)
	if (not os.path.isfile(args.input)):
		print('ERROR: Invalid input parameter file ' + args.input)
		exit(1)

	# Parameters. The keys are those that have been specified in F-TEP service inputs
	inputDataKey = 'input_data'
	# Used to limit which image files will be included
	inputDataPatternKey = 'input_data_pattern'
	windowSizeKey = 'windowsize'
	outputNameKey = 'outputname'
	# Default parameter values
	params = ftep.Params()
	params.setValue(windowSizeKey, 10)
	# Add parameters from file
	params.readFile(args.input)

	# Find input files
	dataDir = os.path.join(args.indir, inputDataKey)
	if (not os.path.isdir(dataDir)):
		print('ERROR: No data directory')
		exit(1)
	files, filteredFiles = ftep.detectFiles(dataDir, supported_extensions=['.las', '.laz'],
		pattern=params.getString(inputDataPatternKey))
	if len(files) == 0:
		print('ERROR: No ALS data detected')
		exit(1)
	if len(filteredFiles) == 0:
		print('ERROR: No ALS data after filtering')
		exit(1)
```

Figure 28 Sample ftep_util usage

4.5 openEO interface to Copernicuc Data Space Ecosystem

The ftep_util Python package (discussed in section 4.4) has since version 0.10 enabled access to Copernicus Data Space Ecosystem (CDSE, https://dataspace.copernicus.eu/) via the openEO interface to exploit the data cube processing on the CDSE platform. To use the openEO interface the user running the job must have a valid CDSE account on the same name as the F-TEP account. The service developer can also decide to use a fixed CDSE user account for the service. The F-TEP implementation for the CDSE access utilizes a token storage which reduces the number of authorization requests to the CDSE account. If the token in the storage is not available or valid the system sends an email to the CDSE account used in the job and the job resumes when the user authenticates via the link provided in the email. Note that the CDSE account name must be a valid email address for this to work. The link should be visible in the job log as well.

TIP: Run Python scripts with the -u switch to enable seeing the log messages unbuffered. Otherwise all log rows may appear in one big chunk when the Python process finishes. E.g.

```
python3 -u myscript.py
```

An example script that does simple data cube processing and stores an output GeoTiff is shown in Figure 29.

```
import ftep_util
import logging
logger = logging.getLogger(__name__)
if __name__ == '__main__':
	logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s - %(message)s')
	con = ftep_util.getOpeneoConnectionCDSE()

	# Now that we are connected, we can initialize our datacube object with the area of interest
	# and the time range of interest using Sentinel 1 data.
	datacube = con.load_collection(
		"SENTINEL2_L2A",
		spatial_extent={"west": 5.14, "south": 51.17, "east": 5.17, "north": 51.19},
		temporal_extent = ["2021-02-01", "2021-04-30"],
		bands=["B02", "B04", "B08", "SCL"],
		max_cloud_cover=85,
	)
	# By filtering as early as possible (directly in load_collection() in this case),
	# we make sure the back-end only loads the data we are interested in and
	# avoid incurring unneeded costs.

	#From this data cube, we can now select the individual bands with the DataCube.band() method
	# and rescale the digital number values to physical reflectances:
	blue = datacube.band("B02") * 0.0001
	red = datacube.band("B04") * 0.0001
	nir = datacube.band("B08") * 0.0001

	# We now want to compute the enhanced vegetation index and can do that directly
	# with these band variables:
	evi_cube = 2.5 * (nir - red) / (nir + 6.0 * red - 7.5 * blue + 1.0)

	# Now we can use the compact “band math” feature again to build a binary mask
	# with a simple comparison operation:
	# Select the "SCL" band from the data cube
	scl_band = datacube.band("SCL")
	# Build mask to mask out everything but class 4 (vegetation)
	mask = (scl_band != 4)

	# Before we can apply this mask to the EVI cube we have to resample it, as the “SCL” layer has
	# a “ground sample distance” of 20 meter, while it is 10 meter for the “B02”, “B04” and “B08” bands.
	# We can easily do the resampling by referring directly to the EVI cube.
	mask_resampled = mask.resample_cube_spatial(evi_cube)

	# Apply the mask to the `evi_cube`
	evi_cube_masked = evi_cube.mask(mask_resampled)

	# Because GeoTIFF does not support a temporal dimension, we first eliminate it by taking
	# the temporal maximum value for each pixel:
	evi_composite = evi_cube.max_time()

	# Now we can download this to a local file:
	evi_composite.download("/home/worker/workDir/outDir/output/evi-composite.tiff")
```

Figure 29 CDSE openEO data cube example

5 Available Docker base images

In the platform there are prebuilt images that can be used as base images for services. The currently available base images are the following:

- ftep-docker-registry:5000/ftep/ubuntu_python:20.04_3.10.13
  - Ubuntu 20.04
  - Python 3.10.13
  - GDAL 3.0.4
  - ftep_util Python module version 0.0.11
- ftep-docker-registry:5000/ftep/ubuntu_python:20.04_3.9.18
  - Ubuntu 20.04
  - Python 3.9.18
  - GDAL 3.0.4
  - ftep_util Python module version 0.0.11
- ftep-docker-registry:5000/ftep/r_terra:4.3.1_1.7
  - Ubuntu 20.04
  - Python 3.8.10
  - GDAL 3.0.4
  - R 4.3.1 with packages
    - remotes
    - codetool
    - Rcpp 1.0.11
    - terra 1.7-39
    - fs 1.6.3
    - tictoc 1.2
- ftep-docker-registry:5000/ftep/ubuntu_r_lidr:20.04_4.3.1_4.0.4
  - Ubuntu 20.04
  - Python 3.8.10
  - GDAL 3.0.4
  - R 4.3.1 with packages
    - remotes 2.4.2.1
    - codetools 0.2-19
    - lidR 4.0.4
    - unix 1.5.6

The full Dockerfile definition of each image is given in the following subsections.

5.1 ftep-docker-registry:5000/ftep/ubuntu_python:20.04_3.10.13

```
FROM ubuntu:20.04
LABEL maintainer lauri.seitsonen@vtt.fi
ENV DEBIAN_FRONTEND=noninteractive

# Dependencies
RUN apt-get update && apt-get install -y \
 software-properties-common curl zip git gdal-bin libgdal-dev wget build-essential \
 zlib1g-dev libncurses5-dev libgdbm-dev libnss3-dev libssl-dev libreadline-dev \
 libffi-dev libsqlite3-dev libbz2-dev \
 && apt-get clean && rm -rf /var/lib/apt/lists/*

# Install Python 3.10
# https://linuxize.com/post/how-to-install-python-3-9-on-ubuntu-20-04/
WORKDIR /tmp
RUN wget https://www.python.org/ftp/python/3.10.13/Python-3.10.13.tgz
RUN tar -xf Python-3.10.13.tgz
WORKDIR /tmp/Python-3.10.13
RUN ./configure --enable-optimizations
RUN make
RUN make altinstall

# Make Python 3.10 the default option for python3
RUN update-alternatives --install /usr/bin/python3 python3 /usr/local/bin/python3.10 1

# Install pip
RUN curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
RUN python3 get-pip.py

# Force install of an older setuptools version.
# GDAL 3.0.4 build with setuptools>58.0.0 fails
RUN pip install build setuptools==57.5.0
# Force the same GDAL version as the binaries
RUN pip install GDAL==3.0.4
# Install ftep_util (which requires GDAL)
RUN curl -LO https://f-tep.com/sites/default/files/ftep_util-0.0.11-py3-none-any.whl
RUN pip install ftep_util-0.0.11-py3-none-any.whl
```

5.2 ftep-docker-registry:5000/ftep/ubuntu_python:20.04_3.9.18

```
FROM ubuntu:20.04
LABEL maintainer lauri.seitsonen@vtt.fi
ENV DEBIAN_FRONTEND=noninteractive

# Dependencies
RUN apt-get update && apt-get install -y \
software-properties-common curl zip git gdal-bin libgdal-dev wget build-essential \
 zlib1g-dev libncurses5-dev libgdbm-dev libnss3-dev libssl-dev libreadline-dev \
 libffi-dev libsqlite3-dev libbz2-dev \
 && apt-get clean && rm -rf /var/lib/apt/lists/*

# Install Python 3.9
# https://linuxize.com/post/how-to-install-python-3-9-on-ubuntu-20-04/
WORKDIR /tmp
RUN wget https://www.python.org/ftp/python/3.9.18/Python-3.9.18.tgz
RUN tar -xf Python-3.9.18.tgz
WORKDIR /tmp/Python-3.9.18
RUN ./configure --enable-optimizations
RUN make
RUN make altinstall

# Make Python 3.9 the default option for python3
RUN update-alternatives --install /usr/bin/python3 python3 /usr/local/bin/python3.9 1

# Install pip
RUN curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
RUN python3 get-pip.py

# Force install of an older setuptools version.
# GDAL 3.0.4 build with setuptools>58.0.0 fails
RUN pip install build setuptools==57.5.0
# Force the same GDAL version as the binaries
RUN pip install GDAL==3.0.4
# Install ftep_util (which requires GDAL)
RUN curl -LO https://f-tep.com/sites/default/files/ftep_util-0.0.11-py3-none-any.whl
RUN pip install ftep_util-0.0.11-py3-none-any.whl
```

5.3 ftep-docker-registry:5000/ftep/r_terra:4.3.1_1.7

```
FROM ubuntu:20.04
ENV DEBIAN_FRONTEND=noninteractive
# Dependencies
# Install 'software-properties-common' and 'apt-transport-https' to manage software repositories
RUN apt-get update && apt-get install -y \
build-essential zip curl bc gdal-bin libgdal-dev libgeos++-dev \
software-properties-common apt-transport-https \
&& apt-get clean && rm -rf /var/lib/apt/lists/*

# Install R
# From https://www.digitalocean.com/community/tutorials/how-to-install-r-on-ubuntu-20-04
RUN apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
E298A3A825C0D65DFD57CBB651716619E084DAB9 && \
  add-apt-repository 'deb https://cloud.r-project.org/bin/linux/ubuntu focal-cran40/' && \
  apt update && \
  apt install -y --no-install-recommends \
    r-base-core=4.3.1-4.2004.0

# Install R packages depending on your needs
RUN Rscript -e 'install.packages("remotes")'
RUN Rscript -e 'install.packages("codetools")'
RUN Rscript -e 'library(remotes); install_version("Rcpp", "1.0.11", repos="https://cloud.r-project.org/")'
RUN Rscript -e 'library(remotes); install_version("terra", "1.7-39", repos="https://cloud.r-project.org/")'
RUN Rscript -e 'library(remotes); install_version("fs", "1.6.3", repos="https://cloud.r-project.org/")'
RUN Rscript -e 'library(remotes); install_version("tictoc", "1.2", repos="https://cloud.r-project.org/")'
```

5.4 ftep-docker-registry:5000/ftep/ubuntu_r_lidr:20.04_4.3.1_4.0.4

```
FROM ubuntu:20.04
LABEL maintainer lauri.seitsonen@vtt.fi
ENV DEBIAN_FRONTEND=noninteractive
# Dependencies
# Install 'software-properties-common' and 'apt-transport-https' to manage software repositories
RUN apt-get update && apt-get install -y \
 python3\
 python3-dev\
 python3-pip\
 python3-gdal\
 build-essential\
 zip\
 curl\
 bc\
 gdal-bin\
 software-properties-common\
 apt-transport-https \
 gfortran\
 libgdal-dev\
 libgeos++-dev\
 libudunits2-dev\
 libproj-dev\
 libx11-dev\
 libgl1-mesa-dev\
 libglu1-mesa-dev\
 libfreetype6-dev\
 libxt-dev\
 libfftw3-dev\
 && apt-get clean && rm -rf /var/lib/apt/lists/*

# Install R
# From https://www.digitalocean.com/community/tutorials/how-to-install-r-on-ubuntu-20-04
RUN apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
E298A3A825C0D65DFD57CBB651716619E084DAB9 && \
 add-apt-repository 'deb https://cloud.r-project.org/bin/linux/ubuntu focal-cran40/' && \
 apt update && \
 apt install -y --no-install-recommends \
 r-base-core=4.3.1-4.2004.0

# Install R packages depending on your needs
RUN Rscript -e 'install.packages("remotes")'
RUN Rscript -e 'install.packages("codetools")'
RUN Rscript -e 'install.packages("lidR")'
RUN Rscript -e 'install.packages("unix")'
```