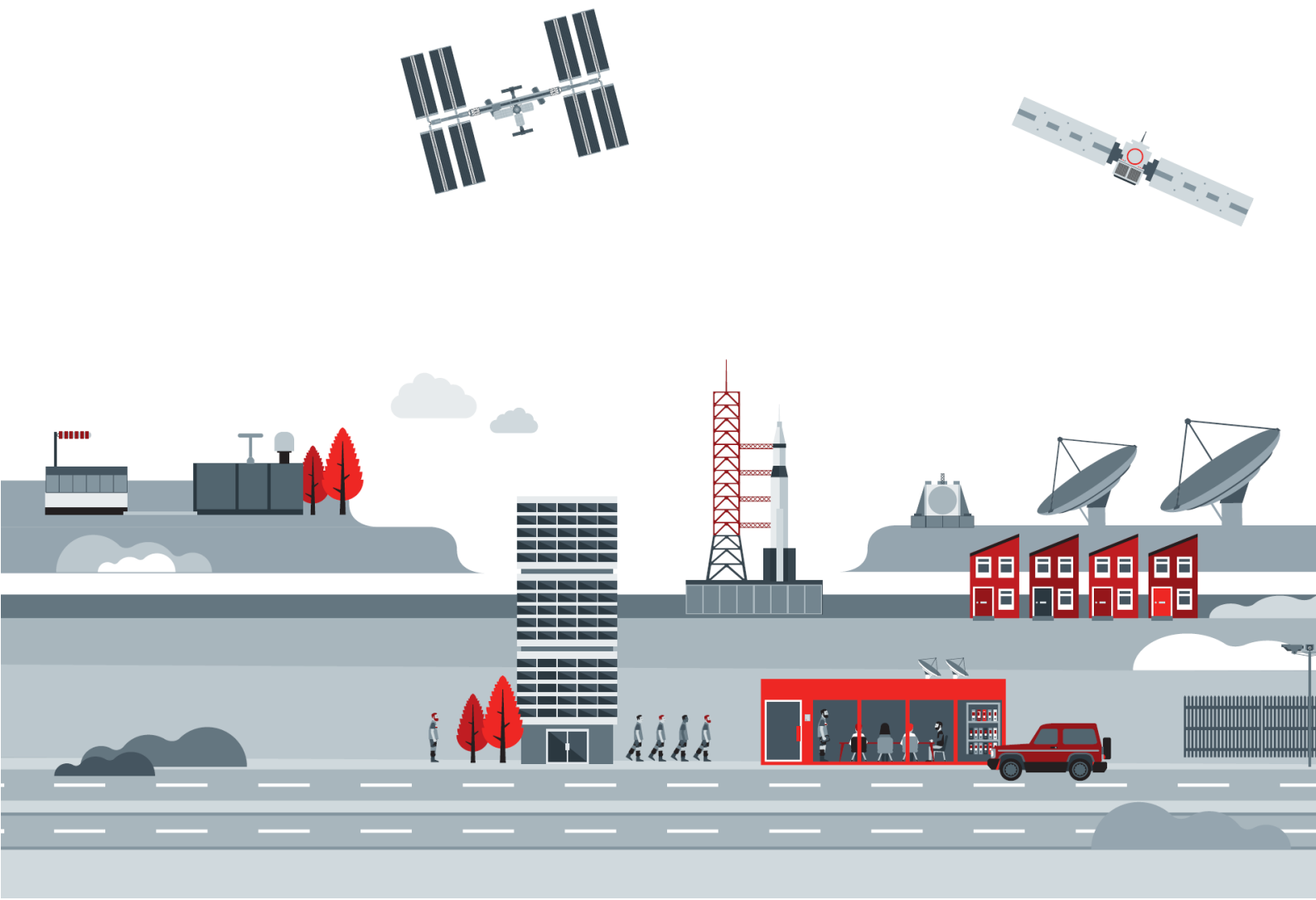
Collaborative Data Hub Software - Maintenance and Evolution Services - Ready for Digital Twin Earth

Transformation Framework ICD



Reference: COPE-SERCO-IF-21-1178
Date: 16/01/2026
Version: 5.0

Role/Title	Name	Signature	Date
Author	Giuseppe Le Voci, Aureliana Barghini, Luca Fabbri, Marco Di Bari		23/08/2023
Verified	Barbara Borgia		23/08/2023
Approved	Andrea Tesseri		23/08/2023

Change register

Version/Rev.	Date	Change	Reason
1.0	16/09/2021		First release of the document
1.1	25/11/2021	Fixed details regarding sen2cor workflow. Updated the Plugin Interface section.	Align to esa_tf v0.8 release
1.2	15/12/2021	Updated minor parameter details	Include latest updates prior to esa_tf v0.8 release
1.3	04/02/2022	Documenting details on the download of TF order outputs in section 3.3, step 7. Updated response in section 5.3.2 accordingly.	Implemented download of order outputs via NGINX
1.4	18/02/2022	Added: In section 3, added 2 sub-sections - 3.1 Service Installation Requirements - 3.2 Transformation Framework Configuration In section 3.5: removed Traceability Service In section 5.3: added sub-section 5.3.3 Filters	Update with new features of release esa_tf 0.9

		Small fixes in restapi responses	
1.5	03/03/2022	In section 3.4: Change role of the Actor allowed to install new plugins	Fix error
1.6	08/04/2022	Add TF configuration and plugins installation	Release esa_tf v0.9
1.7	14/04/2022	add definition of roles in config file add configuration of the owner and group of the output fieles	Release esa_tf v0.9.1
1.8	11/05/2022	Add small section on plugins logging Add output_dir in pluings main function interface description	Plugins interface improvement
1.9	21/06/2022	Add traceability configuration	Release esa_tf v1.0
2.1	03/02/2023	Add Monitoring configuration	Release esa_tf v1.3.0-osf
3.0	11/04/2023	Add configuration of csc-api data-sources	Release esa_tf v1.4.0-osf
4.0	06/09/2023	Add parallel processing configuration	Release esa_tf v1.5.2-osf
4.2	23/12/2024	Update version of plugins sen2cor and eopf	Release esa_tf v1.5.5-osf
4.3	23/12/2024	Fix error in sen2cor plugin	Release esa_tf v1.5.6-osf
5.0	16/12/2026	Remove Traceability Update data-sources	Release esa_tf v1.5.9-osf

Table of Contents

Index of Figures

1. Introduction

1.1 Scope

This document portrays the interfaces of the Transformation Framework within the Collaborative Data Hub Software Maintenance and Evolution Services for Digital Twin Earth. As such, it may serve both Data Hub administrators, its users and eventual developers who may wish to make use of, or expand on, the framework.

1.2 Purpose

This Interface Control Document [ICD] is devoted to the description of interfaces involved in the Transformation Framework. It aims to provide an overview of the architecture of the framework and, in this context, to define the workflow which allows data transformation into suitable output formats. In providing those details, this document also addresses the objective to inform on the pluggable nature of the easily expandable framework and its accessibility.

Please note that this document is referring to the DHS#9 (esa_tf v1.5.9-osf) release.

1.3 Document structure

The document is structured as follows:

- Section 1 (this section): Introduction, providing document structure, reference documents and definitions/acronyms.
- Section 2 contains an overview of the Transformation Framework and its external interfaces.
- Section 3 provides a description of the interfaces and a nominal scenario of product transformation.
- Section 4 details the Entity Model, as well as Transformation Order properties and applicable attributes.
- Section 5 contains examples of Transformation and Workflow Order requests.

1.4 Applicable Documents

Ref.	Title	Reference and Version
AD-1	[AD-SOW] Statement of Work: COLLABORATIVE DATA HUB SOFTWARE - MAINTENANCE AND EVOLUTION SERVICES - READY FOR DIGITAL TWIN EARTH	ESA-EOPG-EOPGC-SOW-12, v1.0
AD-2	DHS System Design Document	COPE-SERCO-TN-21-1171
AD-3	[AD-PRIP] Production Interface Delivery Point Specification - ICD	ESA-EOPG-EOPGC-IF-3, v1.6

1.5 Reference Documents

Ref.	Title	Reference and Version
RD-1.	HTTP Status Codes	<https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html>
RD-2.	[AD-TR-TN] Technical Note for the Interfacing of the Traceability Service by Production Services in RFP-1	ESA-EOPG-EOPGC-TN-23, v.TBD
RD-3.	[AD-TR-ICD] Traceability Service Interface Control Document	ESA-EOPG-EOPGC-IF-9, Issue 1/2
RD-4.	Dask.distributed lightweight library for distributed computing in Python	<https://distributed.dask.org>
RD-5.	Python entry-points specifications	<https://packaging.python.org/specifications/entry-points/>
RD-6.	Sen2Cor 2.12.03 Configuration and User Manual	<https://step.esa.int/thirdparties/sen2cor/2.12.0/docs/OMPC.TPZG.SUM.002%20-%20i1r0%20-%20Sen2Cor%202.12.03%20Configuration%20and%20User%20Manual.pdf>
RD-7.	Traceability Service User Manual	[AS-TR-UM] GAEL-P299-SUM-001-01-10, v1.10

1.6 Acronyms and Abbreviations

Acronym	Definition
AD	Applicable Document
API	Application Programming Interface
DHS	Data Hub Software
EO	Earth Observation
ESA	European Space Agency
GS	Ground Segment
HTTP	Hypertext Transfer Protocol Secure
HTTPS	Hypertext Transfer Protocol Secure
ICD	Interface Control Document
OData	Open Data Protocol
RD	Reference Document
SOW	Statement of Work
TF	Transformation Framework
UUID	Universally unique identifier

2. Transformation Framework overview

The framework for data transformation is a new component of the DHS intended to provide data transformation capabilities via the integration of processing elements applied on-demand to Copernicus Sentinel products, prior to delivery to the users.

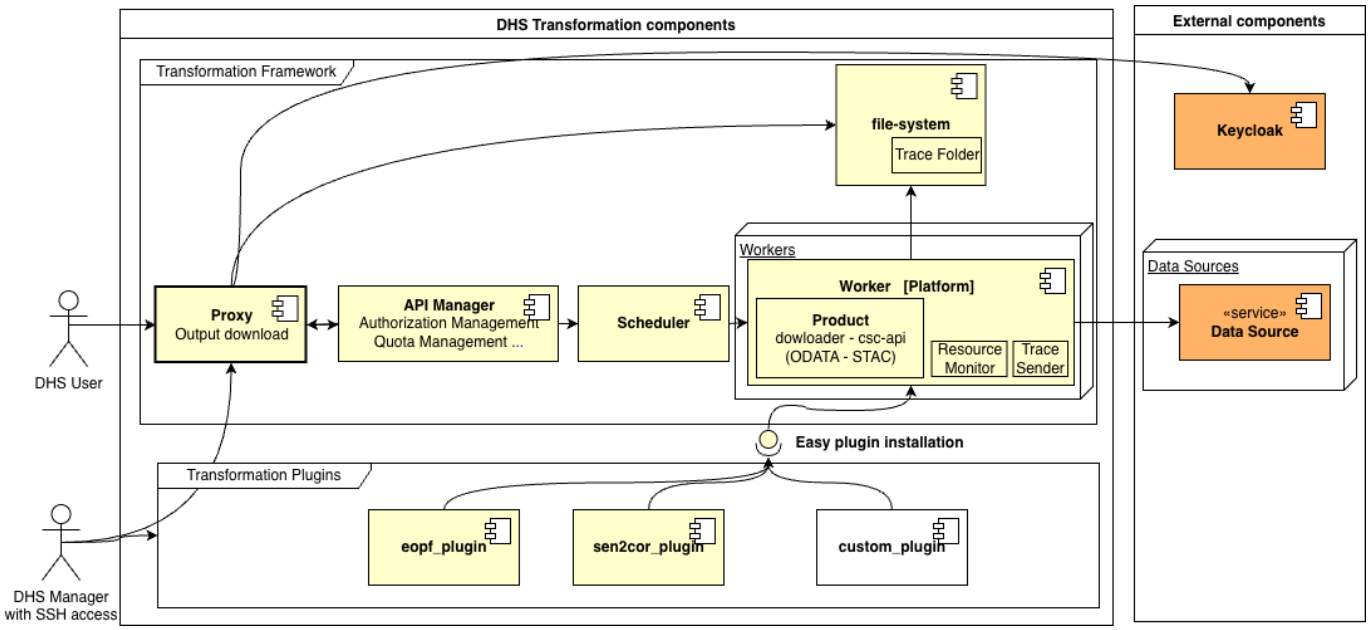


Figure 1: Component diagram of the Transformation Framework and its interfaces.

As depicted in Figure 1, the TF exposes a REST API, that is the main external interface with DHS sub-system, which stores and catalogues data and metrics. Through this interface, the framework allows:

- the retrieval of an input product to be transformed;
- the submission of request for data transformation, including user defined parameters;
- monitoring the progress of transformation processes.

The framework is composed of four components that are independent form the main DHS sub-systems namely:

- The API manager:
  - receives the request and submit them to the scheduler
  - implements the quota management and the authorization enforcement
  - keep in memory all the transformation submitted
- The scheduler and one or more workers that can be either deployed on the same machine, or on a remote processing facility. In particular, the scheduler and workers are provided by the dask.distributed package (see [RD-4.]) and include functionalities to implement queuing/load balancing of requests, and their asynchronous execution. In particular, with dask it is possible to prevent eventual duplicate requests from being processed multiple times, when those are already queued or in processing stage.
- The worker(s) exposes a plugin interface which allows the expansion of processing elements in charge of requested data transformation. This translates in the possibility to add or remove a processing element (workflows) from a given installation without changing the code. Note: Trace Sender is no more supported.
- The proxy that interacts with Keycloak and allows the download of the output.

Internally, the TF also acts as a client for:

- An Interface to the DHS Staging Storage (internal) which allows for packaging and uploading the output products of transformation processes.

3. Transformation Framework interface description

3.1 Service Installation Requirements

The TF core is designed to be locally used with simple commands and minimal requirements. When installed as a service on a production environment, an interaction with external services such as Web proxy and IAM services is required.

3.1.1 Publish the TF behind a proxy

Although the TF provides an internal proxy layer, it is intended for internal use only (logging and download management). Other tasks, such as using an official domain or adding SSL protection, should be provided by an external front-end proxy.

There are two tasks that should be properly configured:

- Publish the TF on an official domain (with HTTPS/SSL)
- Publish the TF in a sub-path of the same domain

As an example, the following sections will explain how to publish the TF installation on `https://mynetwork.net/transformation-framework`.

3.1.1.1 Configuring the domain (and SSL)

The internal proxy layer is already configured to receive and pass through proper HTTP headers from the outer front-end proxy.

The minimal configuration required by the front-end proxy is to pass a `Host` header, as it will be used to compose the hostname of generated URLs. In the example, this Host header will be `mynetwork.net`.
If the front-end proxy provides SSL (which is strongly recommended), the `X-Forwarded-Proto` header should also be set.
In the example HTTPS/SSL is used, therefore `X-Forwarded-Proto` should feature `https`.

Other `X- Forwarded -*` headers can also be useful for improving HTTP logging.

In case changes to the configuration of the front-end proxy are not applicable, it is possible to forcibly set two environment variables:

- `APPLICATION_HOSTNAME`, default: empty string. This is the hostname where the TF will be published. Instead of setting this value, the front-end proxy can be configured to properly set the `Host HTTP` header. In the example it will be again "`mynetwork.net`"
- `APPLICATION_PROTO`, default: empty string. This is the HTTP protocol used to serve the TF. Instead of setting this value, the front-end proxy can be configured to properly set the `X-Forwarded-Proto HTTP` header. In the example: "`https`"

Please note: using any APPLICATION_* environment variables above will override any headers received by the external proxy.

3.1.1.2 Configuring a subpath

The internal proxy is configured to serve the TF at root level.
For example: access to the list of transformations is performed by calling:

```
GET http://mynetwork.net/TransformationOrders.
```

When publishing the TF on the network an entry prefix path may be used, such as:

```
GET http://mynetwork.net/transformation-framework/TransformationOrders.
```

In this case the TF must be instructed accordingly by passing a ROOT_PATH environment variable.
In the example:

```
ROOT_PATH=/transformation-framework
```

3.1.1.3 Network Security

By default, the entry point of the TF is an HTTP service running on port 8080, but various other ports are available for debugging purpose:

- OData RESTful API endpoint
- 8786: Dask scheduler
- tea87mi8
- 7: Dask diagnostic service

3.1.2 Keycloak integration

Integration with Keycloak is performed from the internal NGINX proxy.

Keycloak should provide a client configured as follows:

- Client protocol: openid-connect
- Access Type: bearer-only

This first client will only validate the access token, it is not allowed to generate one. For this reason, a second Keycloak client is required with the following configuration:

- Client protocol: openid-connect
- Access Type: public

As the TF is not providing any user interface, no redirection to the keycloak portal for login can be done.

Authentication will work as follows:

- The user must obtain an access_token from Keycloak via the public client (using curl, postman or any other REST application)
- With this access_token, the user will be able to query the TF, providing an Authorization header:

  ```
  curl http://<host-and-port>/Workflows -H “Authorization: Bearer <access_token>”
  ```

- The NGINX proxy will intercept every request, and will query the Keycloak openid-connect service to introspect the token.

The internal call can be reproduced as follows:

```
curl --data "client_secret=<keycloak-client-secret>&client_id=<keycloak-client-id>&token=<access-token>" https://<keycloak-host>/auth/realms/<realm>/protocol/openid-connect/token/introspect
```

The response must be a standard introspection JSON response:

```
{
  …
  "iss": "https://<keycloak-host>/auth/realms/<realm>",
  "sub": "936ed039-cc94-4fe8-b60d-88e142c132a4",
  "typ": "Bearer",
  "name": "John Smith",
  "given_name": "John",
  "family_name": "Smith",
  "preferred_username": "user1",
  "email": "user1@asdf.com",
  "email_verified": true,
  "resource_access": {
      "<client-id>": {
      "roles": [
        "tf-user",
        "tf-superuser"
      ]
      },
      …
  },
  …
  "username": "user1",
  "active": true
}
```

From this response, NGINX will be able to intercept and pass through the following data:

- username or sub - this information is used to identify the user both on NGINX access log and on the TF operational logs. Although username data is more friendly, using sub data will improve the user's privacy (in case of GDPR requirements);
- resources.<client-id>.roles - this will be a list of roles (both default roles and assigned roles);
- active - if false, the token is valid but expired, so NGINX will treat this as a 401 error.

3.2 Transformation Framework Configuration

3.2.1 Configuration Of The Output Owner

The output owner and group can be configured in a dedicated `.env` file in the `esa_tf` folder, using the following variables:

- `OUTPUT_OWNER_ID`: the id of the owner of the output files
- `OUTPUT GROUP_OWNER_ID:` the group id of the owner of the output files

3.2.2 Docker Volumes Configuration

In `.env` file in the `esa_tf` folder, it is possible to configure the environment variables that define the paths to the docker volumes: output folder, config folder and source folder for external plugins:

- `OUTPUT_DIR`, default: "`./output`". This is the directory where the TF stores the output files.
- `CONFIG_DIR`, default: "`./config`". This is the directory containing the configuration files.
- `PLUGINS_DIR`, default: "`./plugins`". This is a folder where users must load the Python wheels corresponding to any additional external plugins they may want to install. The plugins will be installed automatically at container startup.

3.2.3 Data sources configuration

The data source shall be configured in the `${CONFIG_DIR}/hubs_credentials.yaml` file, where the `${CONFIG_DIR}` can be defined in esa_tf/.env file.

In `${CONFIG_DIR}/hubs_credentials.yam`, it shall be defined:

- api_type: that can be `csc-api`
- the credentials

in case the api_type is `csc-api`, it shall be defined also:

- `auth`: define the authentication type, it can be oauth2 or basic
- `query_auth:` define if the authentication is needed or not for querying the data source, it can be True or False
- `download_auth:` define if the authentication is needed or not for downloading the data from the source, it can be True or False
- `query_auth:` can be `stac` or `odata`

The following snippet represents an example of configuration file:

```
# example for: config/hubs_credentials.yaml
cda:
  api_type: csc-api
  auth: oauth2
  query_api: odata
  query_auth: False
  download_auth: True
  credentials:
    api_url: https://cda.copernicus.eu/cda/
    user: my-cda-username
    password: my-cda-password
    token_endpoint: https://auth.cda.copernicus.eu/auth/realms/CollaborativeDataAccess/protocol/openid-connect/token
    client_id: cda-gss


cdse:
  api_type: csc-api
  auth: oauth2
  query_api: stac
  query_auth: False
  download_auth: True
  credentials:
    api_url: https://catalogue.dataspace.copernicus.eu
    user: my-cdse-username
    password: my-cdse-password
    token_endpoint: https://identity.dataspace.copernicus.eu/auth/realms/CDSE/protocol/openid-connect/token
    client_id: cdse-public
gss:
  api_type: csc-api
  auth: oauth2
  query_api: stac
  query_auth: True
  download_auth: True
  credentials:
    api_url: https://dhs-test.onda-dias.eu/stac-02/stac/
    user: my-gss-username
    password: my-gss-password
    token_endpoint:https://collaborative-dhs.onda-dias.eu/auth/realms/collaborative-dhs/protocol/openid-connect/token
    client_id: ivv-gss1-client
```

The TF will generate an example of `${CONFIG_DIR}/hubs_credentials.yaml`, running in the folder `esa_tf` the command:
`make setup`

3.2.4 Roles configuration

Quotas and profiles shall be configured in the YAML file, `esa_tf/config/esa_tf.config`.
The TF will generate an example of configuration file by running command `make setup` in folder `esa_tf`.

The quota is the maximum number of TransformationOrders in queued status or in processing status for each user.
The profiles are related to the authorizations. Currently, two profiles are available: `user` and `manager`.

The user profile will be able to:

- request the configuration of all available workflows or one specific configuration
- submit a TransformationOrder
- request the list of her TransformationOrder or one specific transformation

The manager profile will be able also to request the list of all the transformations submitted.

The check on the quota and on the profile can be disabled in the configuration file.

The following snippet represents an example of configuration. Note that “role” inside the TF shall be always a couple composed as “`<client_id>:<role_name>`”:

```
# enable authorization check
enable_authorization_check: false

# enable quota configuration
enable_quota_check: true

# role configuration
# - quota is maximum number of transformation orders in progress or queued per user
# - profile can ben "manager", "user"

# default role configuration
default_role:
  quota: 2
  profile: user

roles:
  “client_id:tf_manager”:
    quota: 10
    profile: manager
```

In this example, the `tf_manager` role has a `manager profile` and `quota` 10.

Since `tf_manager` is the only role defined in the configuration file, for all the users that do not have the `tf_manager` role, the TF will use the `default_role`. Note that `default_role` can be customized and, if removed from the configuration file, the TF will use its internal defaults: `profile user` and `quota` 2.

3.2.5 Keeping period and excluded workflows

The keeping period and the list of workflows to be excluded shall be defined in the YAML file `esa_tf/config/esa_tf.config`.

The TF will generate an example of configuration file, running in the folder `esa_tf` the command: `make setup`

The `keeping_period` shall be an integer. It represents the minimum number of minutes from the CompletedDate that a TransformationOrder will be kept in memory.

The following snippet is an example of configuration file:

```
keeping-period: 14400
exclude-workflow:
  - workflow1_id
  - workflow2_id
```

3.2.6 Resource Monitor configuration

The Resource Monitor component is in charge of computing and logging the resources used by each Transformation order.

The Resource Monitor can be activated or deactivated using the key `enable_monitoring` in `esa_tf.config` YAML file in `esa_tf/config` folder. In `esa_tf.config`, you can also configure the `monitoring_polling_period_s`, i.e. the time interval used for sampling disk usage, subprocess CPU usage and RAM usage.

3.2.7 Parallel processing configuration

Every Transformation Order will be submitted to the Dask scheduler and executed within a container named "esa_tf_worker." By default, each worker can concurrently process up to 8 Transformation Orders. This value can be adjusted within the "esa_tf/.env" file using the NPROCESSES environment variable.

3.3 REST API Interface

The REST API interface provides DHS Actors with a "Transformation User" user role the capability to perform three distinct types of requests:

- Workflow Query Requests: these allow the retrieval of supported workflows and available transformation options which are applicable to the input product types of choice.

- Transformation Order Requests: these are processes which perform data transformation. They take as input the `InputProductReference` structure, which includes a combination of "`Reference; ContentDate; DataSourceName`" and, given that information, the product is searched within the pool of configured Data Sources instances (note that in this context, `ContentDate` and `DataSourceName` are optional input configurations). In the event the search is successful, a unique "transformation identifier"

is returned which corresponds to the specific order. The REST API interface does not allow to specify the function execution order in a given transformation process, nor output naming convention.

- Transformation Order Monitoring Requests: in this case, the requests allow Transformation Users to view the status of the ongoing transformation.

Requests can be sent through HTTP/HTTPS calls to the interface entry-points. The RESTful nature of the API allows supporting GET and POST requests, respectively to read and create content.

The content of the response is returned in JSON. The success of a request is returned according to the relevant HTTP status codes [RD.1.].

At a high level, the entry points to the REST API would provide access to the TF functionalities. In particular:

- GET requests to retrieve a list of available workflows
- POST requests to submit Transformation Orders, with the following options:
  - "InputProductReference" (Reference L1C Name, ContentDate Start and Stop, DataSourceName),
  - "WorkflowId",
  - "WorkflowOptions".
- GET requests to query information about an already submitted Transformation Order, filtering by TransformationOrder identifier or Status.

3.4 Plugin Interface

The whole Transformation Framework can be extended by adding custom plugins, which allow to expand the set of available workflows. The interface system is based on entry-points, a feature of the standard package setup tools [RD-5.]. In this context, the ability to modify the configuration in such a way to either activate or deactivate new workflows provided by custom plugins is limited to DHS Actors with SSH access to the VM where the Transformation Framework runs. Three steps are necessary to add a new workflow to the Transformation Framework:

- creating a Python package that implements or wraps the processing;
- describing workflow interface through a dictionary;
- registering the workflow using entry-points interface.

3.4.1 Workflow description dictionary

The description of the workflow consists of a Python dictionary defined in the workflow package. It is the main interface with the Transformation Framework and it will be referenced in the entry-point (see Workflow registration).

An example is the Sen2Cor_L1C_L2A workflow description dictionary:

```
sen2cor_l1c_l2a = {
    "WorkflowName": "Sen2Cor_L1C_L2A",
    "Description": "Product processing from Sentinel-2 L1C to L2A using Sen2Cor v2.12.03, supporting
Level-1C product version 14.2 - 14.9",
    "Execute": "esa_tf_platform.esa_tf_plugin_sen2cor.run_processing",
    "InputProductType": "S2MSI1C",
    "OutputProductType": "S2MSI2A",
    "WorkflowVersion": "0.3",
```

```
    "WorkflowOptions": {
        "Aerosol_Type": {
            "Description": "Default processing via configuration is the rural (continental) aerosol type
with mid latitude summer and an ozone concentration of 331 Dobson Units",
            "Type": "string",
            "Default": "RURAL",
            "Enum": ["MARITIME", "RURAL"],
        },
        "Mid_Latitude": {
            "Description": "If 'AUTO' the atmosphere profile will be determined automatically by the
processor, selecting WINTER or SUMMER atmosphere profile based on the acquisition date and geographic
location of the tile",
            "Type": "string",
            "Default": "SUMMER",
            "Enum": ["SUMMER", "WINTER", "AUTO"],
        },
        "Ozone_Content": {
            "Description": "0: to get the best approximation from metadata (this is the smallest
difference between metadata and column DU), else select for midlatitude summer (MS) atmosphere: 250,
290, 331 (standard MS), 370, 410, 450; for midlatitude winter (MW) atmosphere: 250, 290, 330, 377
(standard MW), 420, 460",
            "Type": "integer",
            "Default": 331,
            "Enum": [0, 250, 290, 330, 331, 370, 377, 410, 420, 450, 460],
        },
        "Cirrus_Correction": {
            "Description": "FALSE: no cirrus correction applied, TRUE: cirrus correction applied",
            "Type": "boolean",
            "Default": False,
        },
        "DEM_Terrain_Correction": {
            "Description": "Use DEM for Terrain Correction, otherwise only used for WVP and AOT",
            "Type": "boolean",
            "Default": True,
        },
        "Resolution": {
            "Description": "Target resolution, can be 10, 20 or 60m. If omitted, 10, 20 and 60m
resolutions will be processed",
            "Type": "integer",
            "Enum": [10, 20, 60],
            "Default": None,
        },
    },
    "ProcessorName":                                                    "Sen2Cor_L1C_L2A",
    "ProcessorVersion":                                                         "v2.12.3",

}
```

The workflow description dictionary shall fulfil multiple requirements.

It shall contain the following set of mandatory keys:

- "WorkflowName"
- "Description"
- "Execute": value shall be a string containing the reference to the Python function that executes the workflow;
- "InputProductType": shall be a valid Sentinel-1 or Sentinel-2 or Sentinel-3 product type;
- "OutputProductType"
- "WorkflowVersion"
- "WorkflowOptions": shall be a dictionary containing the definition of the workflow input options; each workflow option shall contain the following keys;

- "Description" (mandatory)
- "Type" (mandatory)
- "Default" (optional, "Default" type shall be "Type" or None)

The function defined in "Execute" shall comply with the following interface:

```
def run_processing(
    product_path,
    *,
    workflow_options,
    processing_dir,
    output_dir,
   )
```

The first argument shall be the input "product_path", the second keyword argument is the dictionary of the "workflow_options", the third one is the "processing_dir" where data will be downloaded and processed, "output_dir" is the directory where to store the output file. The output of the "Execute" function shall be the absolute path of the output product.

3.4.2 Plugins installation

The installation of a new plugin requires two steps:

- Create the plugin wheel
- Load the wheel in the {PLUGINS_DIR}

The plugins inside the {PLUGINS_DIR} will be installed automatically at container startup.

3.4.3 Workflow registration

The registration of the workflow is performed via the entry-point system [RD-5.]. The workflow description dictionary shall be registered in the `setup.cfg` file of the package, using a naming convention which defines the *group* as `esa_tf_plugin`.

The *name* shall be equal to the workflow_id, i.e. the workflow identifier to be used in the interface to trigger the processing. Please note that the definition of multiple plugins with the same workflow_id will lead to a conflict and only one will be loaded.

The *obje03t reference* of the entry point shall refer to the workflow description dictionary.

For example, the sen2cor plugin is defined as follows:

```
entry_points="""
[esa_tf.plugin]
sen2cor = esa_tf_plugin_sen2cor:sen2cor_l1c_l2a
"""
```

3.4.4 How to access to the order processing folder for debug

The order processing folder, `/opt/esa-tf-platform/working_dir/${order_id}`, is inside the workflow container.

At the end of Transformation Order processing the folder is deleted. In order to avoid this behaviour, before starting docker-compose please set in the file esa_tf/esa_tf/.env the environment variable:

TF_DEBUG=1

The processing folder will be not deleted and the log level of the logger will be set to DEBUG level.

The container can be accessed by following these commands.

- List the docker containers:
```
$ docker ps
1bbf48244f22   collaborativedhs/esa_tf_worker:1.5.9-osf  …
6b58d01fe5e    collaborativedhs/esa_tf_restapi:1.5.9-osf …
d77cfaa604a5   daskdev/dask:2021.8.1-py3.9 …
c17e4cdc8eb4   nginx:1.21.6 …
```

- Select the container id of `collaborativedhs/esa_tf_worker` and run the following command to enter in the container:
```
docker exec -it ${worker_id} bash
```

- Enter in the processing directory:
```
cd working_dir/${order_id}
```

3.4.5 Workflow Logging

The Transformation Framework defines the python root logging with the following format:

"esa_tf-%(tf_version)s - %(name)s – order_id %(order_id)s - %(asctime)s.%(msecs)03d - %(levelname)s - %(message)s",

where:

- %(tf_version)s is the transformation framework version
- %(name)s name of the logger
- %(order_id)s is the order_id
- %(asctime)s.%(msecs)03d is the date time in UTC
- %(levelname)s is the logging level
- %(message)s" is the log message

In order to inherit the same format, the plugins should:

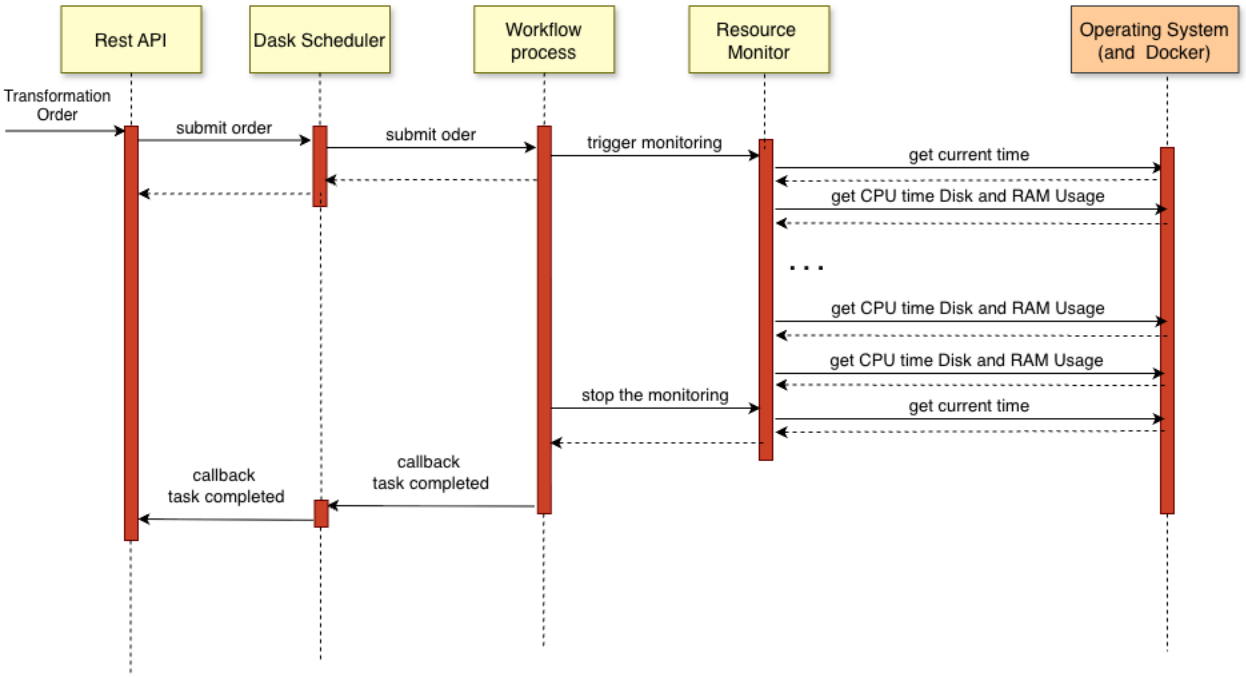
- use the python standard logging library https://docs.python.org/3/library/logging.html
- not override the root logging

In order to clearly identify logs coming from each plugin, it is suggested to group plugin logs always using plugin name as logger name:

```
logger = logging.getLogger(plugin_name)
```

3.5 Resource Monitor Interface

The Resource Monitor is installed on each worker, and it is triggered at the beginning of the Transformation



At the end of the processing the Resource Monitor will log the consumption information in the following formats:

CPU usage

```
esa_tf-1.5.9-osf - esa_tf_platform.monitoring - order_id {order_id} – 18/05/2022 18:00:13.101 – INFO – total CPU Time: 1600 s
```

RAM usage

```
esa_tf-1.5.9-osf - esa_tf_platform.monitoring - order_id {order_id} – 18/05/2022 18:00:13.101 – INFO – peak RAM usage: 1.1 Gb
```

Disk usage

```
esa_tf-1.5.9-osf - esa_tf_platform.monitoring - order_id {order_id} – 18/05/2022 18:00:13.101 – INFO – peak disk usage: 2.1 Gb
```

Processing time and download time from the Data Source

```
esa_tf-1.5.9-osf - esa_tf_platform.monitoring - order_id {order_id} -18/05/2022 18:00:13.101 - INFO - wall time: 3600
s
```

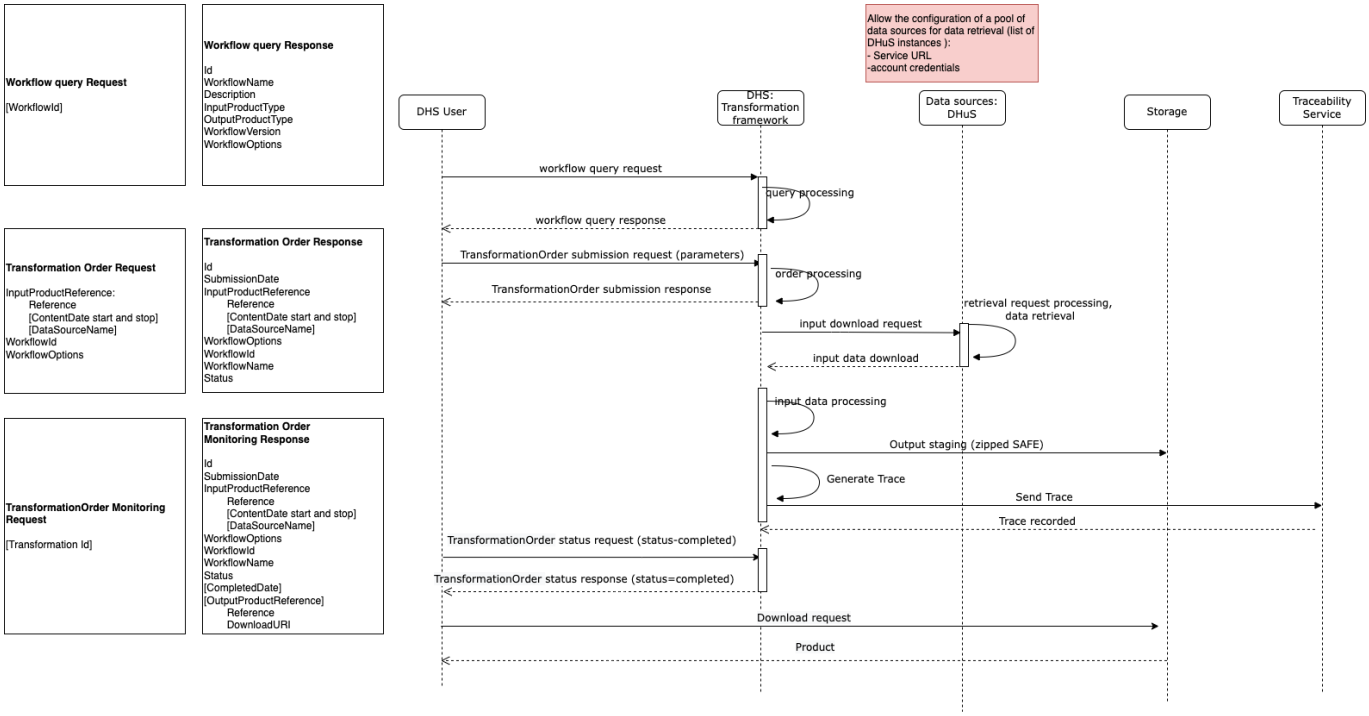


Figure 2: Sequence diagram of the Transformation Framework workflow.

3.6 Nominal Scenario Description

The workflow in a nominal scenario can be decomposed in the following steps:

- **STEP 1: Workflow Query**

A DHS Actor with "Transformation User" user role submits a query to the Transformation Framework to inspect which Workflows are available to process an input product of given type. The query response provides information on the identified Workflows, including the unique identifier, name, textual description and input and output product types. A set of user options within a Workflow may constrain how the requested processing can be modified and override the default options. The default set of options applied are also provided.

- **STEP 2: TransformationOrder Submission**

Upon selection of the desired input product residing in the Data Sources database, to which on-demand processing is to be applied, the User can submit a TransformationOrder request which triggers a dedicated Workflow, defined by the options which are passed in as argument of the request. Once received, the TF takes the TransformationOrder in charge and returns a response for confirmation to the User. In addition to the input parameters, the response includes a unique identifier for the specific TransformationOrder, a status (initially, *queued*), and a submission date.

In this context, please note that for the first definition of this interface it is assumed (and modelled) that a single TransformationOrder contains a single input product, which gives rise to a single output product. A TransformationOrder identifies a single Workflow to be applied.

- **STEP 3: Input Product Download**

When the TF receives a TransformationOrder request:

- if TransformationOrder is not jet in the list of the submitted orders, then the TF will submit the new order and in the response, the order status will be *in_progress.*
- if TransformationOrder is already in the list of the submitted orders, then:
  - if it is in *progress* or if it is *completed* and the output product is available, then the TF does not resubmit the order
  - if it is *completed*, but the output product has been deleted, then the TF resubmit the order
  - if it is *failed* the TF resubmit the order, its status changes to *in_progress.*

Thereafter, the TF initially extracts the information corresponding to the input product. This information is sent to the DHS Data Sources API to search for the product. This retrieval request will follow the OpenSearch syntax, compatible with that required by the Data Sources API. Elaboration of the request by the DHS results in the download of the input data which is sent back to the TF.

- **STEP 4: Input Data Processing**

Upon retrieval of the input product, the TF applies the processing and options identified in the TransformationOrder. The output product will be moved in the staging area, file system based.

- **STEP 5: Process Monitoring**

A DHS Actor with "Transformation User" user role submits a query to the Transformation Framework to inspect the status of the submitted TransformationOrder. The query response provides information on the TransformationOrder submitted, inclucind WorkflowId, WorkflowsOptions, SubmissionDate, Status and, if the order is completed, the response will include the CompletedDate and the OutputProductReference.

- **STEP 6: TransformationOrder Status Update**

The TransformationOrder entity is now updated. Its status turns to *completed* and it is supplemented by the following additional properties: the output product name and identifier, and the completed date. A User who submits a TransformationOrder request corresponding to this TransformationOrder Id, will now receive a response which contains that information.

- **STEP 7: Output Product Retrieval**

The output product is uploaded in .zip format to a NAS file-system configured within the TF. An API for the download of the Transformation Output will be available via a NGINX proxy. It can be either retrieved from the output folder, or downloaded via URL.

Once the processing is complete, the TF has added the field `OutputProductReference` in the TransformationOrder monitoring response. The `OutputProductReference` field is an array of JSON objects, one for each output product (in this context, note that workflows with more than a single output product are not supported, therefore the array has length 1). Each JSON object contains:

- `Reference`: the output filename
- `DownloadURI`: the output URI

The output URI complies with the following convention:

`http://<hostname>:<port>/download/<transformation_order_id>/<filename>.zip`

where:

`<transformation_order_id>` is the Transformation Order Identifier

`<filename>` is the output filename defined by the Workflow Plugin

4. Transformation Framework Entity Model and Properties

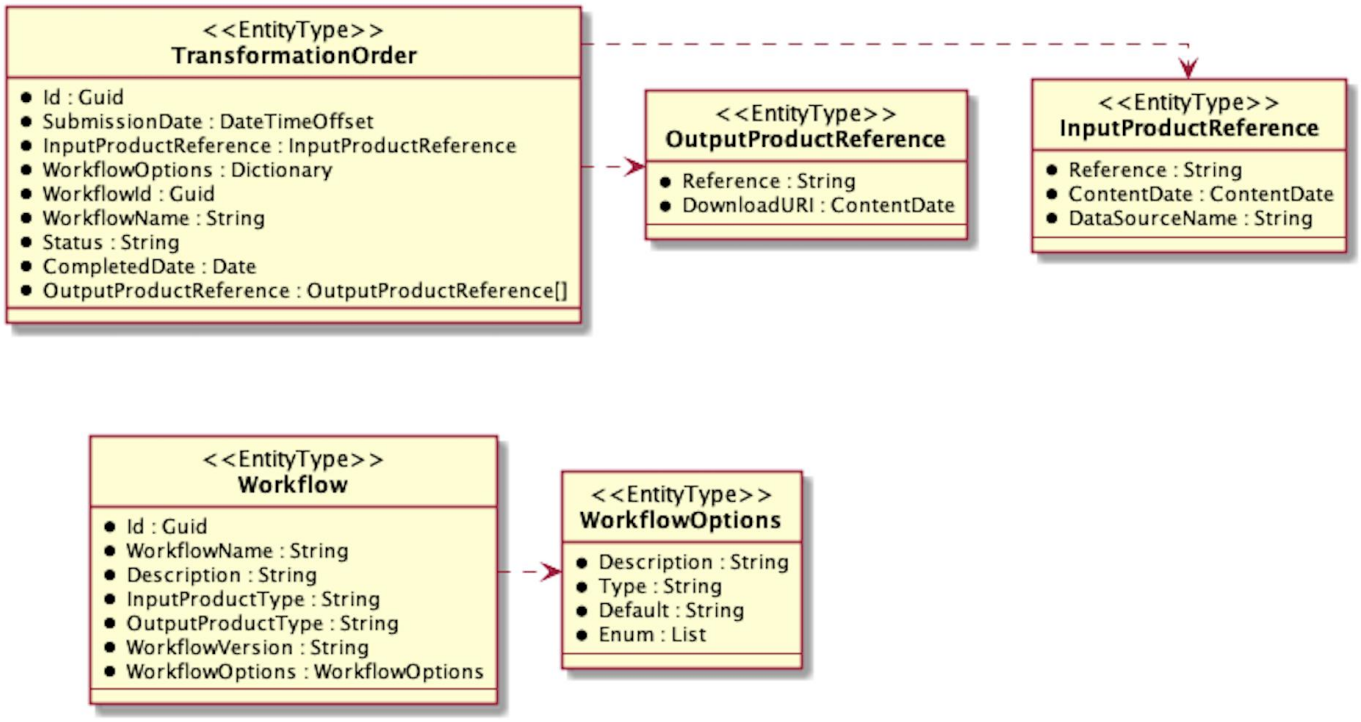


Figure 3: Entity Model for the Transformation Framework.

For more information on the Workflow and Transformation Order Properties, please refer to Tables 5-8 in [AD-3.], keeping in mind that the exact details will be tailored in future versions.

5. Transformation Order Examples

5.1 Workflow Query

5.1.1 Request

A Workflow Query GET Request for the retrieval of an array containing the available workflows could be:

```
GET http://<IP>:<PORT>/Workflows
```

5.1.2 Response

In this case, a use case response would be similar to the following:


```
{
  "value": [
    {
      "Id": "eopf_convert_to_cog",
      "WorkflowName": "eopf_convert_to_cog",
      "Description": "EOPF plugin for converting Sentinel-1, Sentinel-2 and Sentinel-3 SAFE in COG
format",
      "InputProductType": [
        "WV_SLC__1S",
        "IW_SLC__1S",
        "EW_SLC__1S",
        "IW_GRDH_1S",
        "EW_GRDH_1S",
        "IW_GRDM_1S",
        "EW_GRDM_1S",
        "IW_OCN__2S",
        "EW_OCN__2S",
        "S2MSI1C",
        "S2MSI2A",
        "SR_1_SRA_BS",
        "SL_1_RBT___",
        "OL_1_EFR___",
        "OL_1_ERR___",
        "OL_2_LRR___",
        "SL_2_LST___",
        "SL_2_FRP___",
        "SY_2_SYN___",
        "SR_2_LAN___"
      ],
      "OutputProductType": null,
      "WorkflowVersion": "0.3",
      "WorkflowOptions": {
        "cog_compression": {
          "Description": "Type of compression",
          "Type": "string",
          "Default": "DEFLATE",
          "Enum": [
            "NONE",
            "LZW",
            "JPEG",
            "DEFLATE",
            "ZSTD",
            "WEBP",
            "LERC",
            "LERC_DEFLATE",
            "LERC_ZSTD",
            "LZMA"
```

```
          ]
        }
      }
    },
    {
      "Id": "eopf_convert_to_netcdf",
      "WorkflowName": "eopf_convert_to_netcdf",
      "Description": "EOPF plugin for converting Sentinel-1, Sentinel-2 and Sentinel-3 SAFE in netcdf
format",
      "InputProductType": [
        "IW_OCN__2S",
        "EW_OCN__2S",
        "S2MSI1C",
        "S2MSI2A",
        "SR_1_SRA_BS",
        "SL_1_RBT___",
        "OL_1_EFR___",
        "OL_1_ERR___",
        "OL_2_LRR___",
        "SL_2_LST___",
        "SL_2_FRP___",
        "SY_2_SYN___",
        "SR_2_LAN___"
      ],
      "OutputProductType": null,
      "WorkflowVersion": "0.3",
      "WorkflowOptions": {
        "netcdf_compression": {
          "Description": "Activate the compression",
          "Type": "boolean",
          "Default": true,
          "Enum": [
            true,
            false
          ]
        },
        "netcdf_comp_level": {
          "Description": "Compression level",
          "Type": "integer",
          "Default": 1,
          "Enum": [
            1,
            2,
            3,
            4,
            5,
            6,
            7,
            8,
            9
          ]
        },
        "netcdf_shuffle": {
          "Description": "Activate the shuffle",
          "Type": "string",
          "Default": "YES",
          "Enum": [
            "YES",
            "NO"
          ]
        }
      }
    },
```

```
    {
      "Id": "eopf_convert_to_zarr",
      "WorkflowName": "eopf_convert_to_zarr",
      "Description": "EOPF plugin for converting Sentinel-1, Sentinel-2 and Sentinel-3 SAFE in zarr
format",
      "InputProductType": [
        "WV_SLC__1S",
        "IW_SLC__1S",
        "EW_SLC__1S",
        "IW_GRDH_1S",
        "EW_GRDH_1S",
        "IW_GRDM_1S",
        "EW_GRDM_1S",
        "IW_OCN__2S",
        "EW_OCN__2S",
        "S2MSI1C",
        "S2MSI2A",
        "SR_1_SRA_BS",
        "SL_1_RBT___",
        "OL_1_EFR___",
        "OL_1_ERR___",
        "OL_2_LRR___",
        "SL_2_LST___",
        "SL_2_FRP___",
        "SY_2_SYN___",
        "SR_2_LAN___"
      ],
      "OutputProductType": null,
      "WorkflowVersion": "0.3",
      "WorkflowOptions": {
        "dask_compression": {
          "Description": "Type of compression",
          "Type": "string",
          "Default": "zstd",
          "Enum": [
            "zstd",
            "blosclz",
            "lz4",
            "lz4hc",
            "zlib",
            "snappy"
          ]
        },
        "dask_comp_level": {
          "Description": "Compression level",
          "Type": "integer",
          "Default": 1,
          "Enum": [
            1,
            2,
            3,
            4,
            5,
            6,
            7,
            8,
            9
          ]
        },
        "dask_shuffle": {
          "Description": "Shuffle: NOSHUFFLE (0), SHUFFLE (1), BITSHUFFLE (2) or AUTOSHUFFLE (-1)",
          "Type": "integer",
          "Default": 2,
```

```
          "Enum": [
            0,
            1,
            2,
            -1
          ]
        }
      }
    },
    {
      "Id": "sen2cor_l1c_l2a",
      "WorkflowName": "Sen2Cor_L1C_L2A",
      "Description": "Product processing from Sentinel-2 L1C to L2A using Sen2Cor v2.12.03, supporting
Level-1C product version 14.2 - 14.9",
      "InputProductType": "S2MSI1C",
      "OutputProductType": "S2MSI2A",
      "WorkflowVersion": "0.3",
      "WorkflowOptions": {
        "Aerosol_Type": {
          "Description": "Default processing via configuration is the rural (continental) aerosol type
with mid latitude summer and an ozone concentration of 331 Dobson Units",
          "Type": "string",
          "Default": "RURAL",
          "Enum": [
            "MARITIME",
            "RURAL"
          ]
        },
        "Mid_Latitude": {
          "Description": "If 'AUTO' the atmosphere profile will be determined automatically by the
processor, selecting WINTER or SUMMER atmosphere profile based on the acquisition date and geographic
location of the tile",
          "Type": "string",
          "Default": "SUMMER",
          "Enum": [
            "SUMMER",
            "WINTER",
            "AUTO"
          ]
        },
        "Ozone_Content": {
          "Description": "0: to get the best approximation from metadata (this is the smallest
difference between metadata and column DU), else select for midlatitude summer (MS) atmosphere: 250,
290, 331 (standard MS), 370, 410, 450; for midlatitude winter (MW) atmosphere: 250, 290, 330, 377
(standard MW), 420, 460",
          "Type": "integer",
          "Default": 331,
          "Enum": [
            0,
            250,
            290,
            330,
            331,
            370,
            377,
            410,
            420,
            450,
            460
          ]
        },
        "Cirrus_Correction": {
          "Description": "FALSE: no cirrus correction applied, TRUE: cirrus correction applied",
```

```
          "Type": "boolean",
          "Default": false,
          "Enum": [
            true,
            false
          ]
        },
        "DEM_Terrain_Correction": {
          "Description": "Use DEM for Terrain Correction, otherwise only used for WVP and AOT",
          "Type": "boolean",
          "Default": true,
          "Enum": [
            true,
            false
          ]
        },
        "Resolution": {
          "Description": "Target resolution, can be 10, 20 or 60m. If omitted, 10, 20 and 60m
resolutions will be processed",
          "Type": "integer",
          "Default": null,
          "Enum": [
            10,
            20,
            60
          ]
        }
      }
    }
  ]
}
```

5.2 Transformation Order

5.2.1 Request

An example of Transformation Order POST Request for the submission of a specific Transformation Order would be:

```
POST http://<IP>:<PORT>/TransformationOrders
```

with, for instance, the following JSON payload:

```
{
  "InputProductReference": {
    "Reference": "S2B_MSIL1C_20191025T085939_N0208_R007_T37VCC_20191025T112031.zip",
    "DataSourceName": "scihub"
  },
  "WorkflowId": "sen2cor_l1c_l2a",
  "WorkflowOptions": {
    "Aerosol_Type": "RURAL",
    "Mid_Latitude": "SUMMER",
    "Ozone_Content": 331,
    "Cirrus_Correction": false,
    "DEM_Terrain_Correction": true,
    "Resolution": 10
  }
}
```

Note that in this case, ContentDate and DataSourceName are optional parameters.

5.2.2 Response

If such an Order has been successfully submitted, its status would turn to `in_progress`. The following is an example of response sent back to the User in a scenario where a Sentinel L1C input product is converted to an L2A output product via the sen2cor transformation plugin [RD-6.]:

```
{
  "Id": "93582f9c5e9da4d5ed462e6762dc0879",
  "InputProductReference": {
    "Reference": "S2B_MSIL1C_20191025T085939_N0208_R007_T37VCC_20191025T112031.zip",
    "DataSourceName": "scihub”,
  },
  "WorkflowOptions": {
    "Aerosol_Type": "RURAL",
    "Mid_Latitude": "SUMMER",
    "Ozone_Content": 331,
    "Cirrus_Correction": false,
    "DEM_Terrain_Correction": true,
    "Resolution": 10
  },
  "WorkflowId": "sen2cor_l1c_l2a",
  "WorkflowName": "Sen2Cor_L1C_L2A",
  "SubmissionDate": "2021-12-14T10:12:40.113617",
  "Status": "in_progress"
}
```

5.3 Transformation Order Monitoring

5.3.1 Request

An example of Transformation Order Monitoring Request for the retrieval of details pertaining to a specific Transformation Order of given Id would be:

```
GET https://<IP>:<PORT>/TransformationOrders("<ID>")
```

with `ID: 9ad5f1264795f663f7d7a36daedf0635`

Note that the user with `manager profile` can request all the transformations submitted by all the users using a dedicate route:

```
POST http://<IP>:<PORT>/admin/TransformationOrders
```

The route is forbidden for the user without a manager profile.

5.3.2 Response

A successful response corresponding to such an Order with status `completed` would then be the following:

```
{
  "Id": "9ad5f1264795f663f7d7a36daedf0635",
  "InputProductReference": {
    "Reference": "S2B_MSIL1C_20191025T085939_N0208_R007_T37VCC_20191025T112031.zip",
    "DataSourceName": "scihub",
  },
  "WorkflowOptions": {
```

```
    "Aerosol_Type": "RURAL",
    "Mid_Latitude": "SUMMER",
    "Ozone_Content": 331,
    "Cirrus_Correction": false,
    "DEM_Terrain_Correction": true,
    "Resolution": 10
  },
  "WorkflowId": "sen2cor_l1c_l2a",
  "WorkflowName": "Sen2Cor_L1C_L2A",
  "Status": "completed",
  "SubmissionDate": "2021-12-14T10:12:40.113617",
  "CompletedDate": "2021-12-14T10:50:22.233514"
  "OutputProductReference": [
        {
            "Reference": "S2B_MSIL2A_20191025T085939_N9999_R007_T37VCC_20211214T093626.zip",
            "DownloadURI":
"http://<IP>:<PORT>/download/9ad5f1264795f663f7d7a36daedf0635/S2B_MSIL2A_20191025T085939_N99
99_R007_T37VCC_20211214T093626.zip"
        }
    ]
}
```


5.3.3 Filters

The Transformations Orders Monitoring support the following OData filters:

- "Status" with "eq" operator
- "InputProductReference" with "eq" operator
- "SubmissionDate" with "le", "lt", "ge", "gt", "eq", operators
- "CompletedDate" with "le", "lt", "ge", "gt", "eq", operators

For example to obtain all the transformations with status completed, it shall be used the following request:

```
GET https://<IP>:<PORT>/TransformationOrders?\filter=Status%20eq%20'completed'
```