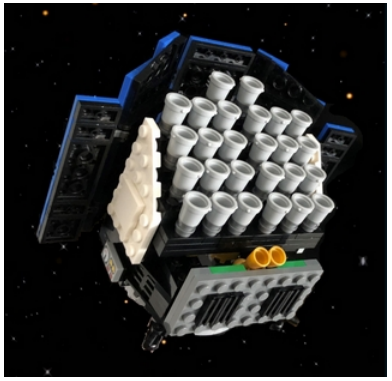
Utilisation d'Airflow pour le traitement de données d'observation de la mission spatiale PLATO – Épisode 2

SIST24 – Marseille

Hervé Ballans

5 juin 2024

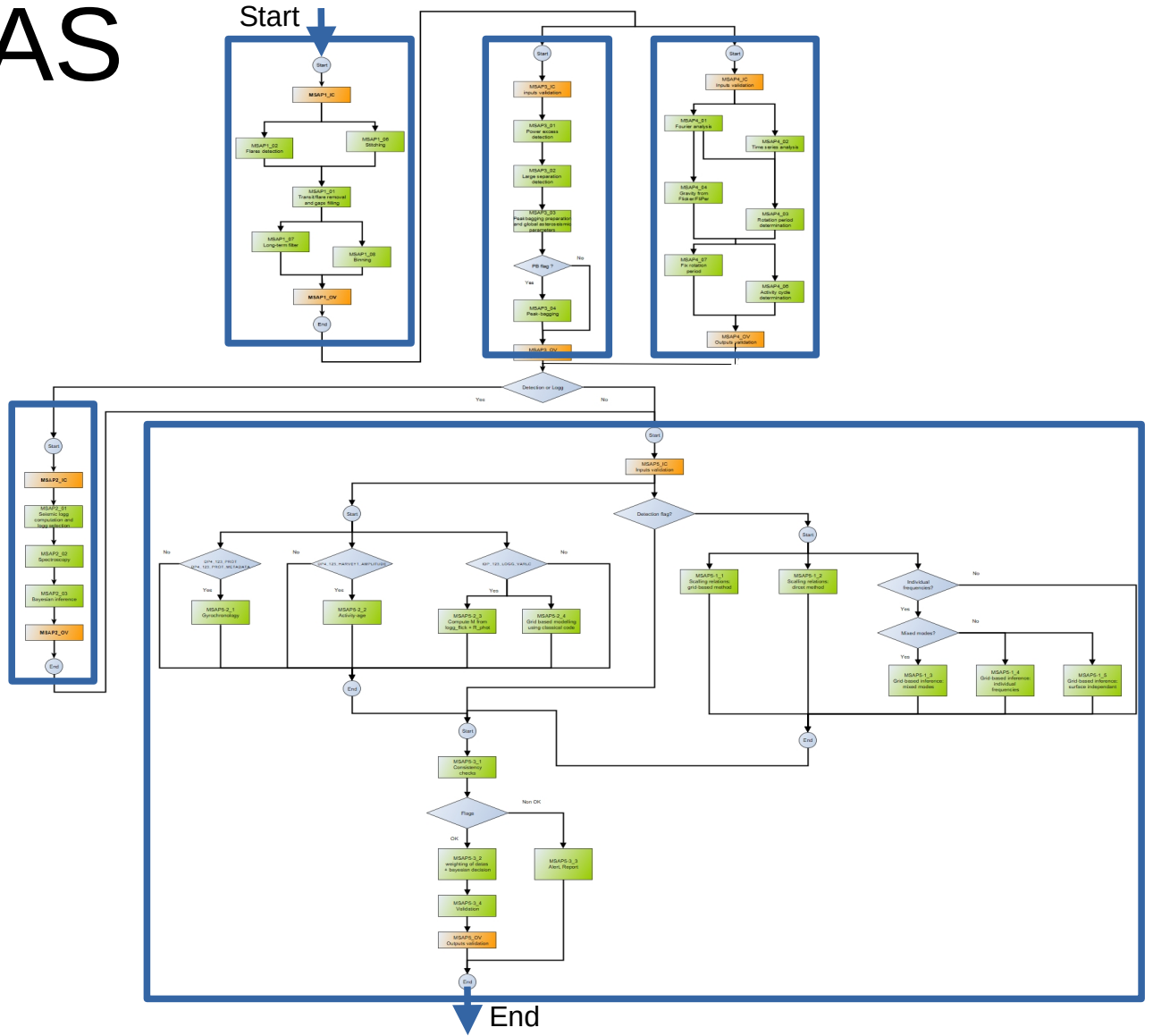
Résumé de l’épisode 1



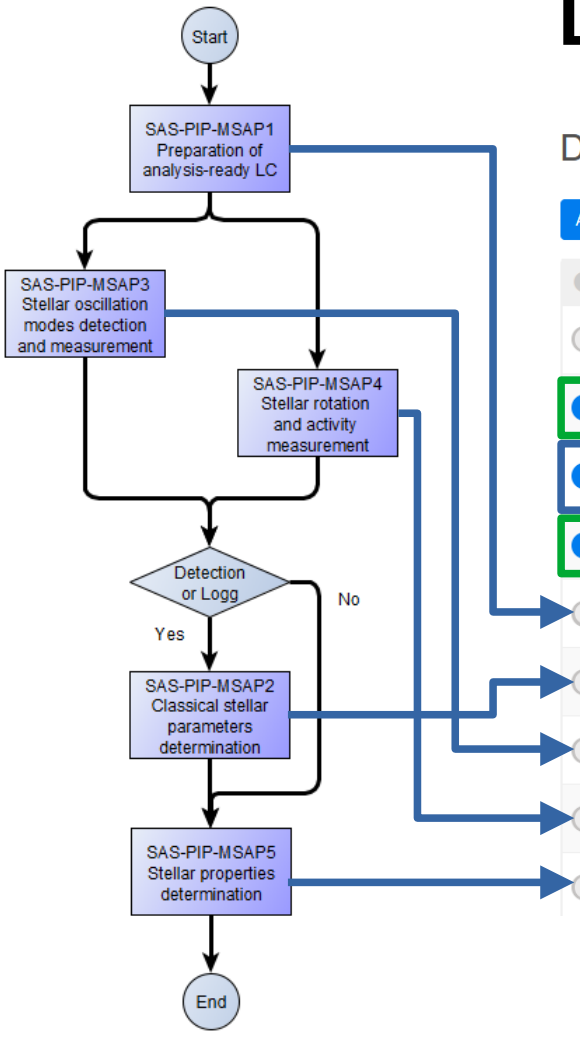
- Contexte : mission spatiale ESA
  - lancement fin 2026 (exploitation : 2027-2030, 2034 avec extension)
  - Observations longues de ~300 000 étoiles
- Responsabilité IAS : mise en place et exploitation d’une chaîne de traitements contenant environ 30 modules scientifiques (SAS – Stellar Analysis System)
- Environnement : 10 instituts différents pour le développement des modules
- Choix technologique :
  - Airflow pour la gestion du pipeline (nos workflows sont des DAGs!)
  - Docker pour la livraison des modules scientifiques à intégrer
  - Gitlab/Gitlab-CI pour la centralisation des codes sources et la configuration de l’intégration continue

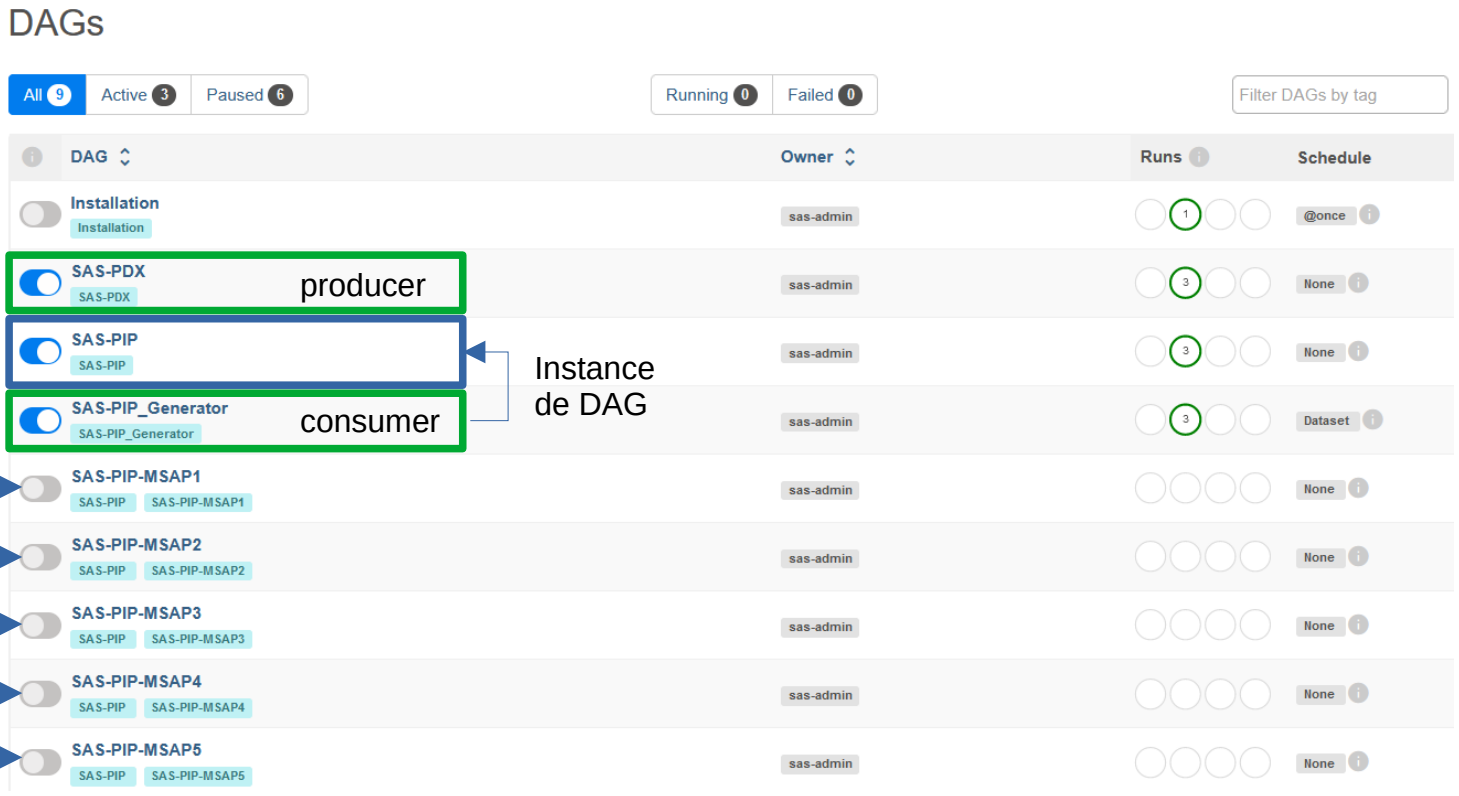
Le pipeline SAS

- 5 grandes étapes de traitements
- 1 boite verte → 1 module scientifique
- 1 étoile en entrée (courbe de lumière)
- ~200 paramètres en sortie (dont Masse, Rayon et Âge de l'étoile)
- Déclenchement du pipeline par événement de mise à jour d'un dataset (DAGs producer/consumer)



DAG modulaires

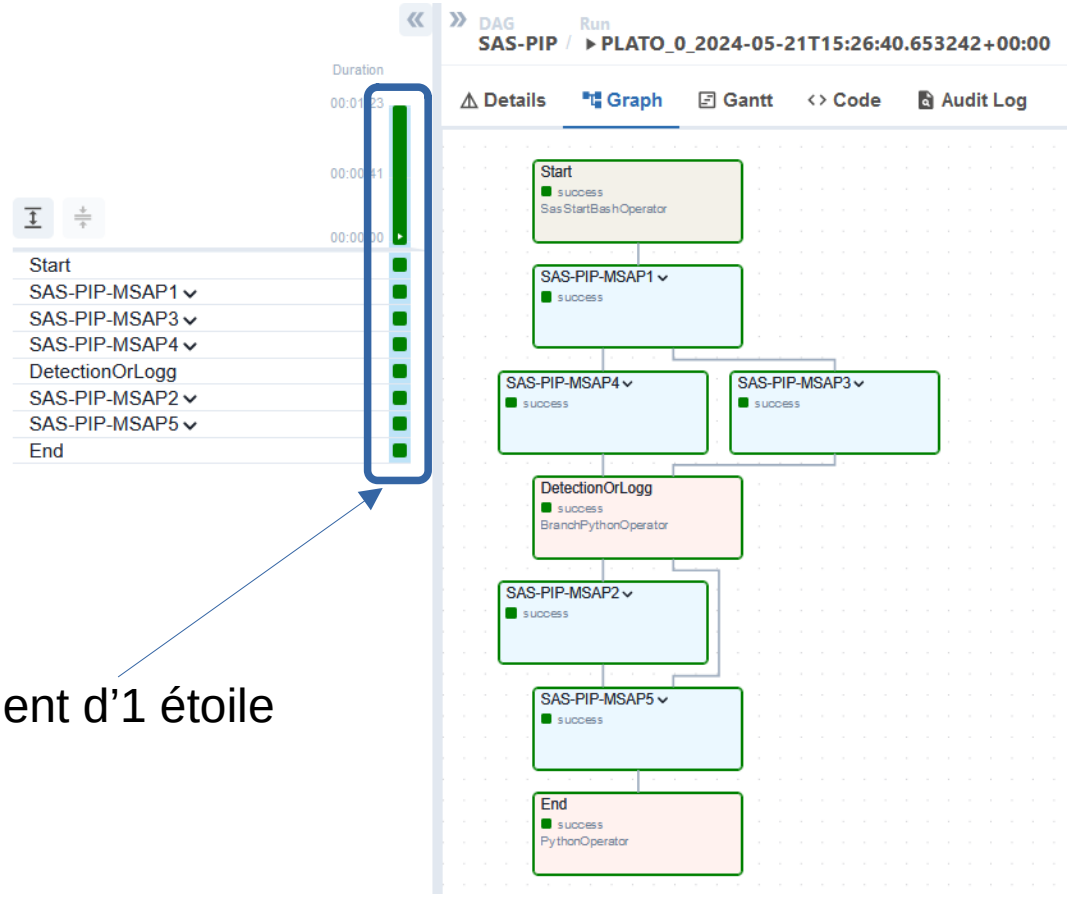




DAG SAS-PIP

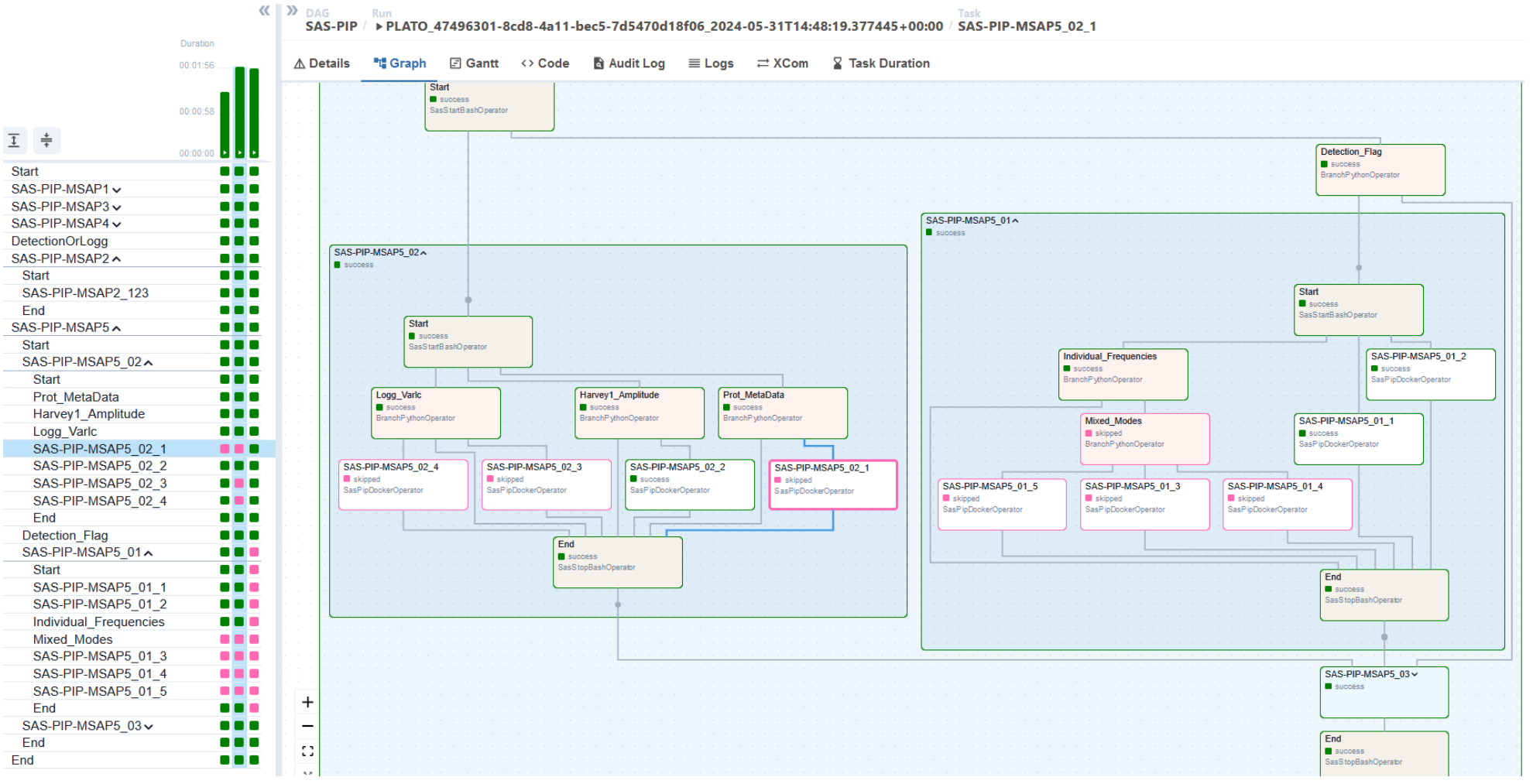
```
import h5py
from airflow import DAG
from airflow.operators.python import BranchPythonOperator
from airflow.operators.python import PythonOperator
from pdcdb_client.apis.sas_api import SasApi
from pdcdb_client.factory import PdcdbApiFactory
from pdcdb_client.interfaces.models.l2 import HarveyProfile
from pdcdb_client.interfaces.models.l2 import StellarParameters
from pdcdb_client.interfaces.models.meta import PlatoVersionMetadata
from sas_plugins.constants import H5MappingTable
from sas_plugins.constants import Msap3ModuleEnum
from sas_plugins.constants import Msap4ModuleEnum
from sas_plugins.constants import SasPipEnum
from sas_plugins.constants import SubSystemEnum
from sas_plugins.group_components.pip import msap1_taskgroup
from sas_plugins.group_components.pip import msap2_taskgroup
from sas_plugins.group_components.pip import msap3_taskgroup
from sas_plugins.group_components.pip import msap4_taskgroup
from sas_plugins.group_components.pip import msap5_taskgroup
from sas_plugins.hooks import SasDataStoreHook
from sas_plugins.operators import SasStartBashOperator
from sas_plugins.utils import retry
```

Instance du DAG SAS-PIP

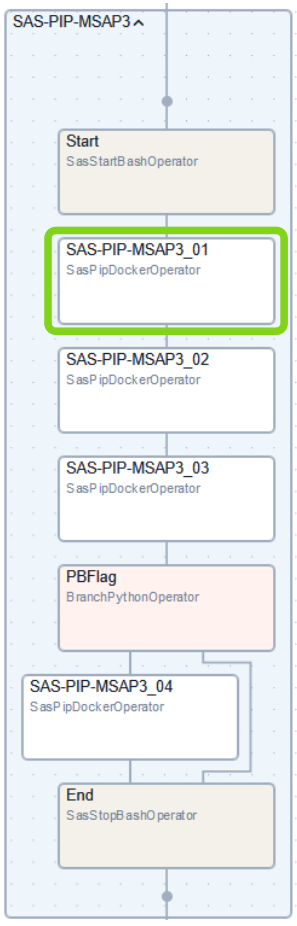


Traitement d’1 étoile

Instance du DAG SAS-PIP



DockerOperator « surchargé »




```python
from airflow.operators.python import BranchPythonOperator
from airflow.utils.task_group import TaskGroup
from airflow.utils.trigger_rule import TriggerRule
from sas_plugins.constants import H5MappingTable
from sas_plugins.constants import Msap3ModuleEnum
from sas_plugins.constants import SasPipEnum
from sas_plugins.operators import SasPipDockerOperator
from sas_plugins.operators import SasStartBashOperator
from sas_plugins.operators import SasStopBashOperator
from sas_plugins.utils import retry
```

```python
def msap3_taskgroup(storage_directory, group_id, plato_id):
    with TaskGroup(SasPipEnum.SAS_PIP_MSAP3.value) as group:
        start = SasStartBashOperator(
            pipeline_id=SasPipEnum.SAS_PIP_MSAP3.value
        )

        MSAP3_01 = SasPipDockerOperator(
            task_id=Msap3ModuleEnum.MSAP3_01.value,
            plato_id=plato_id,
            doc_md=Msap3ModuleEnum.MSAP3_01.__doc__,
        )

        MSAP3_02 = SasPipDockerOperator(
            task_id=Msap3ModuleEnum.MSAP3_02.value,
            plato_id=plato_id,
            doc_md=Msap3ModuleEnum.MSAP3_02.__doc__,
        )

        MSAP3_03 = SasPipDockerOperator(
            task_id=Msap3ModuleEnum.MSAP3_03.value,
            plato_id=plato_id,
            doc_md=Msap3ModuleEnum.MSAP3_03.__doc__,
        )

        pbflag = BranchPythonOperator(
            task_id="PBFlag",
            python_callable=test_msap3_03_output,
            op_args=[storage_directory, group_id, plato_id],
        )
```

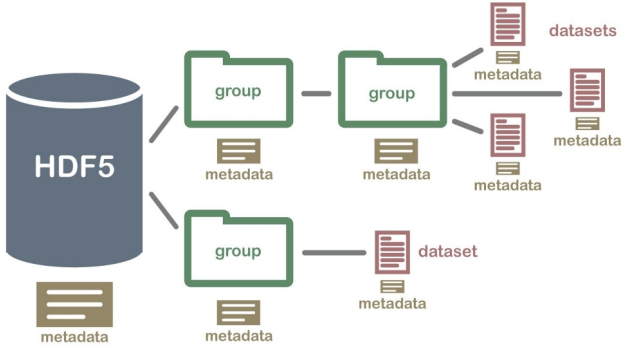
```python
from airflow.providers.docker.operators.docker import DockerOperator
from sas_plugins.constants import DataProductType
from sas_plugins.constants import ErrorMessage
from sas_plugins.constants import H5MappingTable
from sas_plugins.hooks import SasDataStoreHook
from sas_plugins.pattern import Observable
from sas_plugins.pattern import ProcessingTraceability
from sas_plugins.utils import retry

from docker.types import Mount

logger = logging.getLogger(__name__)

DOCKER_CONN_NAME: str = "test_registry_conn"

class SasPipDockerOperator(DockerOperator, Observable):
        "plato_id",
        "custom_command",
    ]

    def __init__(
        self,
        task_id: str,
        plato_id: str,
        custom_image: Optional[str] = None,
        custom_command: Optional[str] = None,
        custom_docker_url: Optional[str] = None,
        **kwargs,
    ) -> None:
        """Init the class.

        Args:
            task_id (str): task ID in the workflow
            plato_id (str): plato ID to compute
            custom_image (Optional[str], optional): image to execute. Defaults to None.
            custom_command (Optional[str], optional): command to execute. Defaults to None.
            custom_docker_url (Optional[str], optional): custom docker URL. Defaults to None.
        """
        # Possible jinja templates
        self.plato_id: str = plato_id
        self.custom_image: Optional[str] = custom_image
        self.custom_command: Optional[str] = custom_command
        self.custom_docker_url: Optional[str] = custom_docker_url

        # Init I/O and hooks
        self.__host_plato_id_directory: Optional[str] = None
        self.__host_input_data: Optional[str] = None
        self.__host_output_data: Optional[str] = None
        self.__docker_plato_id_directory: Optional[str] = None
        self.__docker_input_data: Optional[str] = None
        self.__docker_output_data: Optional[str] = None
        self.__pip_hook: Optional[SasDataStoreHook] = None
        self.__sto_dsm_hook: Optional[SasDataStoreHook] = None

        # Hook connectors
        self.__sas_pip_conn_id: str = (
            SasDataStoreHook.SasDataStoreUri.SAS_PIP.get_conn_airflow()
        )
        self.__sas_sto_dsm_conn_id: str = (
            SasDataStoreHook.SasDataStoreUri.SAS_STO_DSM.get_conn_airflow()
        )

        # Init super constructors
        DockerOperator.__init__(
            self,
            task_id=task_id,
            image=None,
```

Gestion des données

- Les données ne transitent pas d’un module à l’autre (Airflow n’est pas un gestionnaire de dataflow!)
- Tout passe par un backend de stockage centralisé (garantit la possibilité de rejeu
- Modèle de données strict !
  - Le module (l’image Docker) attend une liste stricte d’entrées (obligatoires et optionnelles)
  - Après l’exécution dans le worker (instance de l’image Docker), le module produit une liste stricte de données (obligatoires et optionnelles)
- Format unique utilisé sur tout le pipeline : choix de HDF5

Format HDF5

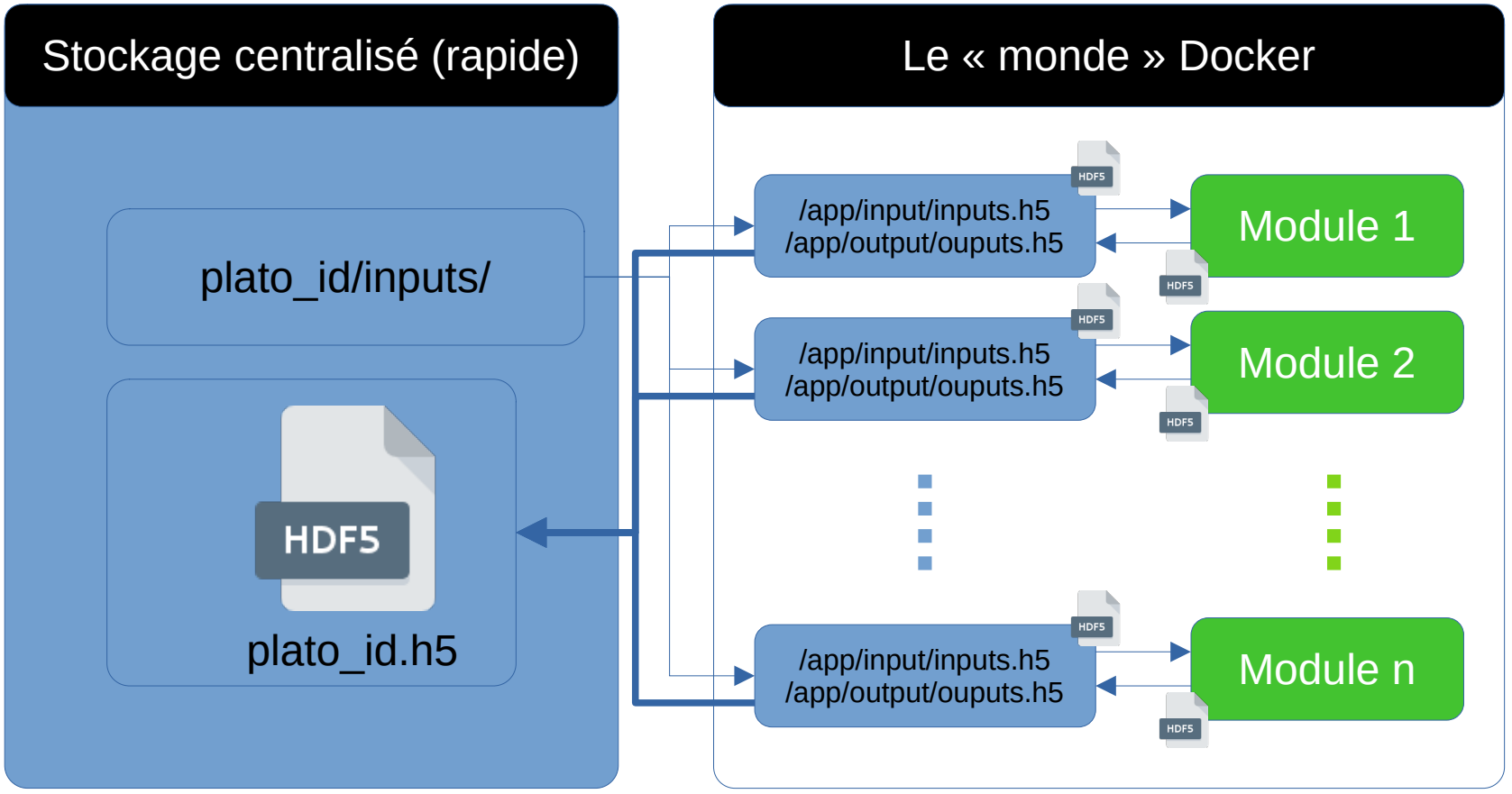
- Format hiérarchisé pour le stockage, la lecture, la visualisation, la manipulation et l'analyse des données scientifiques
- Format adapté pour les grosses volumétries et supportant les E/S parallèles et les « threads »
- Python : librairie h5py



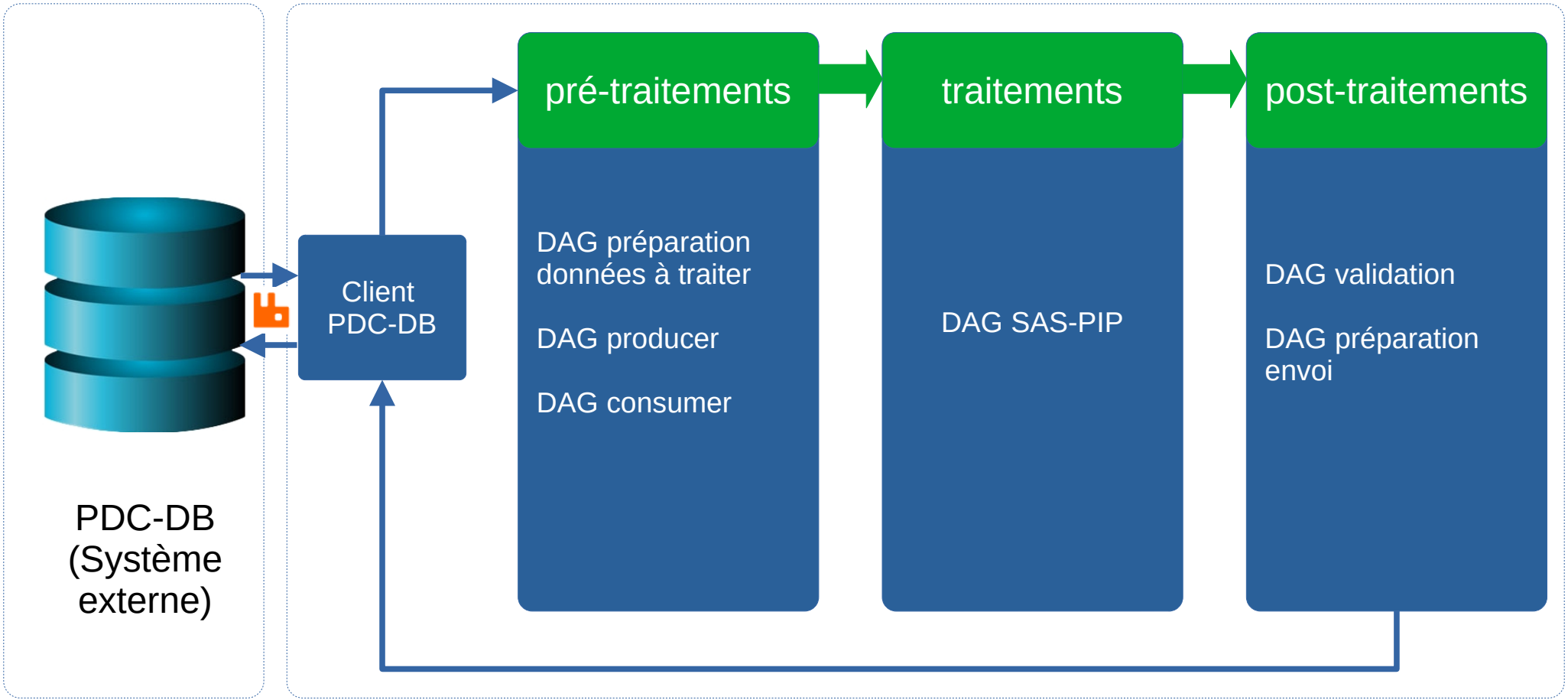

```
HDF5 "PLATO_ffece7e1-9748-4ee1-b00e-a575f2776a69.h5" {
GROUP "/" {
   ATTRIBUTE "OBS_END_DATE" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( 0 ) / ( 0 ) }
      DATA {
      }
   }
   ATTRIBUTE "OBS_START_DATE" {
      DATATYPE  H5T_IEEE_F64LE
      DATASPACE  SIMPLE { ( 0 ) / ( 0 ) }
      DATA {
      }
   }
   ATTRIBUTE "PLATO_ID" {
      DATATYPE  H5T_STRING {
         STRSIZE H5T_VARIABLE;
         STRPAD H5T_STR_NULLTERM;
         CSET H5T_CSET_UTF8;
         CTYPE H5T_C_S1;
      }
      DATASPACE  SCALAR
      DATA {
      (0): "PLATO_ffece7e1-9748-4ee1-b00e-a575f2776a69"
      }
   }
   ATTRIBUTE "QUARTER_NUMBER" {
      DATATYPE  H5T_STD_I64LE
      DATASPACE  SCALAR
      DATA {
      (0): 0
      }
   }
```

```
      GROUP "SAS-PIP-MSAP5_01_4" {
         ATTRIBUTE "outputs" {
            DATATYPE  H5T_STRING {
               STRSIZE H5T_VARIABLE;
               STRPAD H5T_STR_NULLTERM;
               CSET H5T_CSET_UTF8;
               CTYPE H5T_C_S1;
            }
            DATASPACE  SIMPLE { ( 8 ) / ( 8 ) }
            DATA {
            (0): "IDP_124_MASS_GRID_FREQS", "IDP_124_RADIUS_GRID_FREQS",
            (2): "IDP_124_AGE_GRID_FREQS", "IDP_124_LOGG_GRID_FREQS",
            (4): "IDP_124_TEFF_GRID_FREQS", "IDP_124_L_GRID_FREQS",
            (6): "IDP_124_DENSITY_GRID_FREQS",
            (7): "IDP_124_METADATA_REFMOD_GRID_FREQS"
            }
         }
         GROUP "inputs" {
            ATTRIBUTE "PDP" {
               DATATYPE  H5T_STRING {
                  STRSIZE H5T_VARIABLE;
                  STRPAD H5T_STR_NULLTERM;
                  CSET H5T_CSET_UTF8;
                  CTYPE H5T_C_S1;
               }
               DATASPACE  SIMPLE { ( 7 ) / ( 7 ) }
               DATA {
               (0): "PDP_D_122_TEFF_SAPP", "PDP_D_122_M_H_SAPP",
               (2): "PDP_D_122_ALPHA_FE_SPECTROSCOPY", "PDP_C_122_L_IRFM",
               (4): "PDP_B_121_MOD_EVOL", "PDP_B_121_MOD_OSC_FREQ",
               (6): "PDP_D_122_COVMAT_SAPP"
               }
            }
            ATTRIBUTE "SAS-PIP-MSAP3/SAS-PIP-MSAP3_03" {
               DATATYPE  H5T_STRING {
                  STRSIZE H5T_VARIABLE;
                  STRPAD H5T_STR_NULLTERM;
                  CSET H5T_CSET_UTF8;
                  CTYPE H5T_C_S1;
```

Interfaces internes



DAGs pre et post traitements



Conclusion

- Dans notre contexte (séparation des développements du pipeline et des modules scientifiques) :
  - Construire un pipeline modulaire
  - Uniformiser les interfaces (critique pour l'intégration)
  - Utiliser Docker pour les modules (tasks) du pipeline
  - Utiliser un backend de stockage pour gérer les entrées/sorties d'une tâche à l'autre
  - Séparer les DAGs de pre et post-processing de ceux du traitement
- Prochaine étape : test du pipeline sur une infrastructure Kubernetes (Épisode 3?)