Data Pipelines and Data Logistics Service

Jedrzej Rybicki – JSC

SPONSORED BY THE Federal Ministry of Education and Research

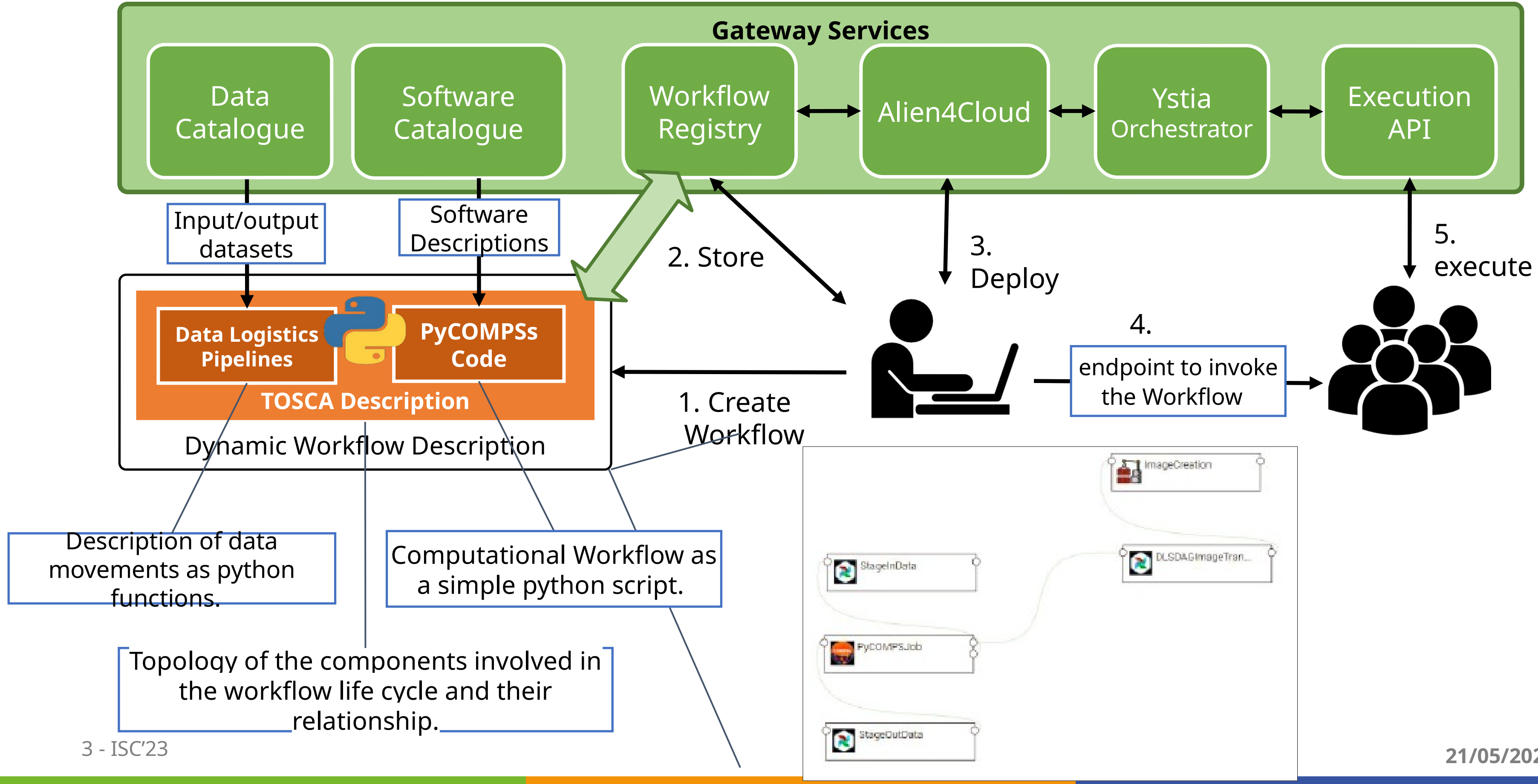
This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955558. The JU receives support from the European Union’s Horizon 2020 research and innovation programme and Spain, Germany, France, Italy, Poland, Switzerland, Norway. The project has received funding from German Federal Ministry of Education and Research agreement no. 16GPC016K.

Data Pipelines: Motivation

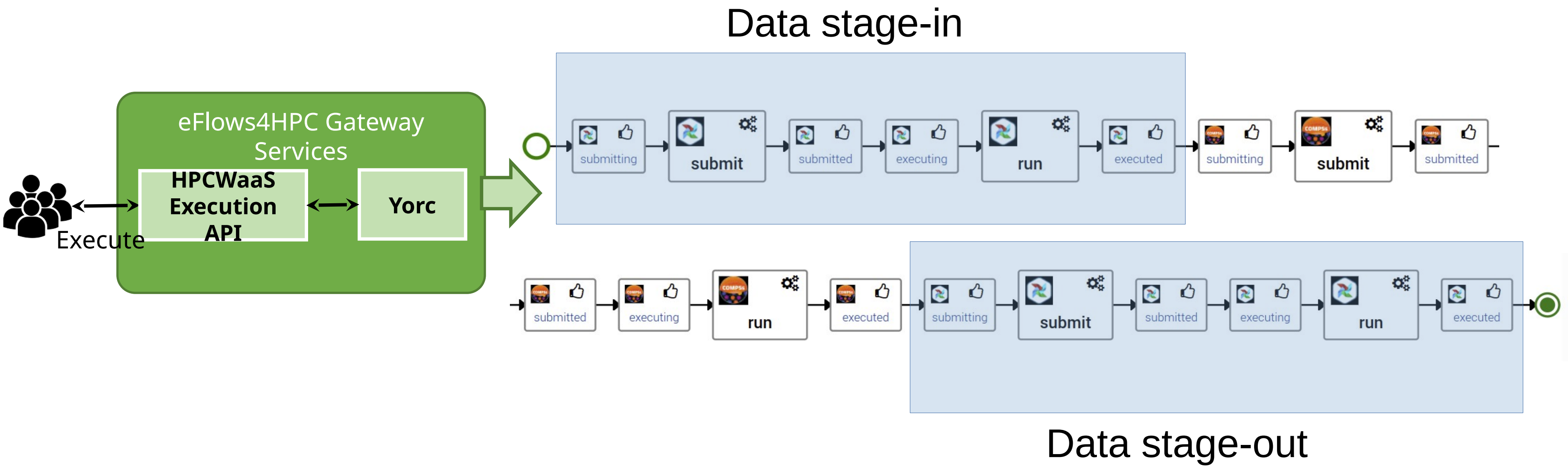
eFlows4HPC aims to deliver a workflow software stack and an additional set of services to enable the integration of HPC simulations and modelling with big data analytics and machine learning in scientific and industrial applications. The software stack will **allow for the creation of innovative adaptive workflows** that efficiently use computing resources considering novel storage solutions.

- Computations require (lots) of data
- Data Logistics Service (DLS): fuel the scientific calculations with required data
- DLS pipelines describe how the data are moved

eFlows4HPC Workflow life-cycle



End-user workflow execution



Data Pipelines: Motivation

- Assumptions about data:
  - Right data
  - Right place
  - Right time
  - → Often proven wrong! (hacks! “legacy solutions”)
- Requirement to publish the data
  - → Often forgotten
- Data Pipelines (as part of scientific workflows):
  - Formalization
  - Reproducibility (FAIR)
  - Portability of the workflows

Apache Airflow

Airflow Features

- Pipelines as code
- Extensible (ready to use operators)
- Flexible
- Powerful scheduling, restarting, etc
- Templating
- Various deployment options (standalone → Kubernetes)
- Dependency injections (connections, etc)
- Monitoring
- UI, Rest API

Apache Airflow

Airflow is a platform created by the community to programmatically author, schedule and monitor workflows

- Workflows → Pipelines (Pipelines as Code (Python))
- Defined as DAG (Directed Acyclic Graph), and contains individual pieces of work called Tasks, arranged with dependencies and data flows taken into account
- DAG specifies the dependencies between Tasks
- Tasks themselves describe what to do
- Airflow Scheduler and Executor execute your tasks on an array of Workers
- User interface to visualize pipelines, monitor progress, and troubleshoot issues when needed
- created by: Maxime Beauchemin at Airbnb
- Apache incubation program (2016), Apache Top Level (2019)

```python
from datetime import datetimefrom airflow import DAG

from airflow.operators.bash import BashOperator

from airflow.operators.python import PythonOperator



def myfunc():

   print("And hello from Python!")



with DAG( "simple", description="simple DAG example", schedule="@daily", start_date=datetime(2023, 4, 1), catchup=False) as dag:

       task1 = BashOperator(task_id="print", bash_command="echo 'Hallo world!'")

       task2 = PythonOperator(task_id="myfunc", python_callable=myfunc)

       task1 >> task2
```

```
from airflow.decorators import dag, task
import pendulum


@dag(dag_id="simpleft", schedule="@daily", catchup=False, start_date=pendulum.datetime(2023, 4, 1, tz="UTC"))
def simpletf():
        @task
        def task1():
                print("Hello world")
                return "Prompt"
        @task
        def task2(prompt):
                print(prompt)


        prompt = task1()
        task2(prompt)
simpletf()
```

DAG

- `start_date` and `schedule` required

  → series of intervals for which DAG will be ran

  – manual trigger is also possible)
- By default (`catchup=True`) a run for each interval
- Backfill: rerun for all intervals
- Trigger rules: `all_success` (default), `all_failed`, `all_done`, etc
- Arguments: inputs (with defaults)

```
@dag(
        start_date=pendulum.datetime(2023,1,4, tz='UTC'),
        schedule="@daily", # timedelta(days=1), "0 * * * *"
        end_date=pendulum.datetime(2023, 1, 20),
        depends_on_past=True,
        catchup=True,
        tags = [‘isc’, ‘tutorial’]
        params={
                'name': Param('John', type='string'),
                'age': Param(16, type='integer', minimum=16) })
def mydag():
 . ...
```

Tasks

- Task: basic unit of work
- Kinds:
  - Operators (Task templates)
  - Sensors (Task waiting for something)
  - Decorated Python functions
- Task has up- and down-stream
- Communication with XComs (small messages)

Operators

- `BashOperator`, `PythonOperator`
- `DockerOperator`, `HiveOperator`
- DB: `MySqlOperator`, `PostgresOperator`, `MsSqlOperator`, `OracleOperator`, `JdbcOperator`
- `SimpleHttpOperator`, `S3FileTransformOperator`
- Messaging: `EmailOperator`, `SlackAPIOperator`
- `BaseOperator`: use to build your own
- More:
  - https://airflow.apache.org/docs/apache-airflow/stable/operators-and-hooks-ref.html

Sensors

- Examples:
  - `BashSensor` (arbitrary command for sensing)
  - `FileSensor`
  - `TimeSensor`
- Modes: poke vs. reschedule

TaskFlow Tasks

- Decorated Python functions
- Internally: `PythonOperator` + XComs
- New feature in Airflow 2.0

```
from airflow.decorators import dag, task
import pendulum
from airflow.sensors.base import PokeReturnValue

@dag( dag_id="simpletf", schedule="@daily", catchup=False, start_date=pendulum.datetime(2023, 4, 1, tz="UTC"))
def simpletf():
    @task.sensor(poke_interval=60, timeout=3600, mode="reschedule")
    def waitfor():
        return PokeReturnValue(is_done=True, xcom_value="xcom_value")

    @task.virtualenv(task_id='python_in_venv',
        requirements=['scikit-learn'],
        system_site_packages=False)
    def task1():
        import sklearn
        return "Prompt"

    prompt = task1()
    waitfor() >> prompt
simpletf()
```

Tasks attributes

- `execution_timeout` – define maximal running time
- `timeout` – maximal tries for sensor
- `slas` and `sla_miss_callback` – define SLA
- `params` – execution context

Differentiate:

- Dag vs DagRun
- Operator vs Task vs TaskRun

“Dependency injection”

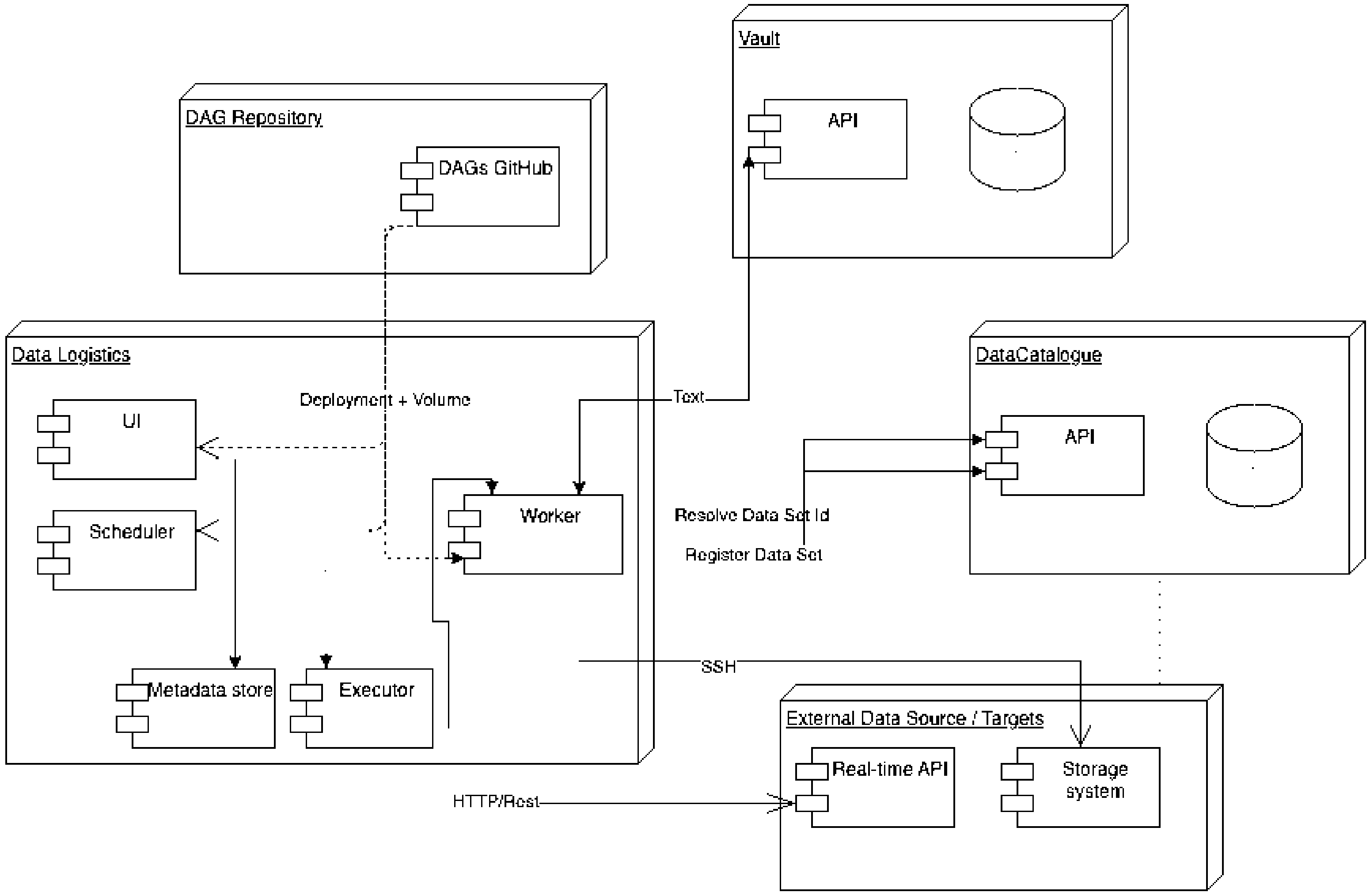
eFlows4HPC

- Parameters (for particular DagRun)
- Variables (global constants)
- Hooks (interact with external entities) and Connections (credentials, etc)

```
@dag(schedule=None, catchup=False, start_date=pendulum.datetime(2023,1,4, tz='UTC'),
params={ 'name': Param('John', type='string'), 'age': Param(16, type='integer', minimum=16) })
def mydag():
    @task
    def setup_db():
        ….
    @task
    def insert_data(**context):
        hook = SqliteHook(conn_name_attr='sqlite_default') # goto: admin/connections to see/add connections
        parms = context['params']
        name = parms['name']
        age = parms['age']
        rows = [(name, age)]
        hook.insert_rows(table="Customers", rows=rows, target_fields=["first_name", "age"])
```

Architecture

- **Scheduler:** handles both triggering scheduled workflows, and submitting **Tasks** to the executor to run
- **Executor:** responsible for running tasks (part of the **Scheduler** or separate instance)
- **Worker(s):** execute **Tasks**, communicate with **Executor**
- **Webserver:** User interface to inspect, trigger and debug the behavior of DAGs and tasks, and API for integration
- A folder of DAG files, read by the **Scheduler** and **Executor**
- **Metadata database:** used by the scheduler, executor and web server to store state


DAG Repository
DAGs GitHub
Vault
API
Data Logistics
Deployment + Volume
UI
Scheduler
Worker
Metadata store
Executor
Text
DataCatalogue
API
Resolve Data Set Id
Register Data Set
SSH
External Data Source / Targets
Real-time API
Storage system
HTTP/Rest

eFlows4HPC examples

Thanks

eFlows4HPC

Enabling dynamic and Intelligent workflows in the future EuroHPC ecosystem

www.eFlows4HPC.eu

@eFlows4HPC eFlows4HPC Project

SPONSORED BY THE Federal Ministry of Education and Research

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955558. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Germany, France, Italy, Poland, Switzerland, Norway. The project has received funding from German Federal Ministry of Education and Research agreement no. 16GPC016K.