INTRODUCTION TO HPC WORKFLOWS AS A SERVICE AND SOFTWARE STACK (Session 1)

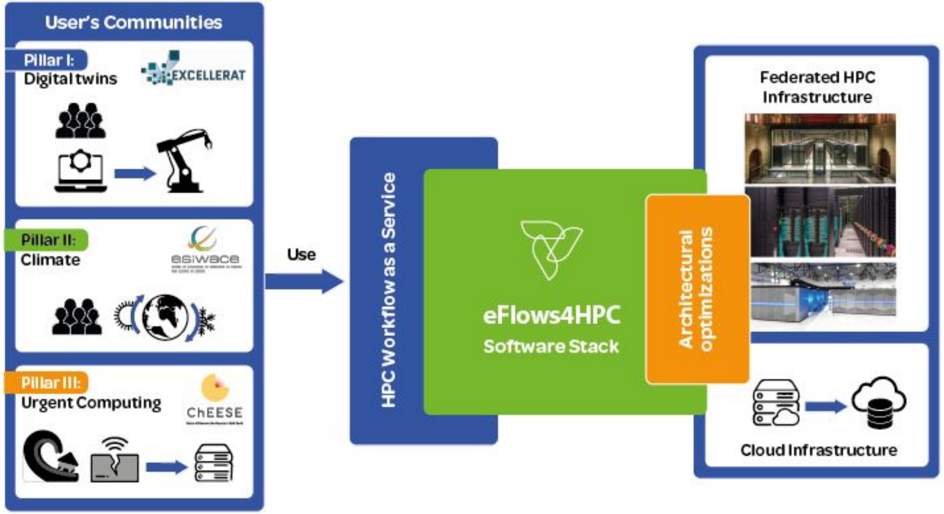
14th September 2022

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.

Outline

- **Overview**
- **Session 1: eFlows4HPC Software stack and HPCWaaS (20 min each)**
    - Part 1: Integrating different computations in PyCOMPSs
    - Part 2: HPC ready container images
    - Part 3: Data Pipelines and Data Logistics Service
    - Part 4: TOSCA Orchestration and HPCWaaS
- **Session 2: Other Software Components (15 min each)**
    - EDDL for ML in Project Pillars
    - Ophidia in Project Pillars
    - dataClay split
- **Hand-on session (45 min)**

Project Overview



A European **workflow platform** that enables the design of complex applications:

- integrating HPC processes, data analytics and artificial intelligence
- enabling the accessibility and reusability of applications to reduce the time to solution

NUMBER OF PARTNERS

16

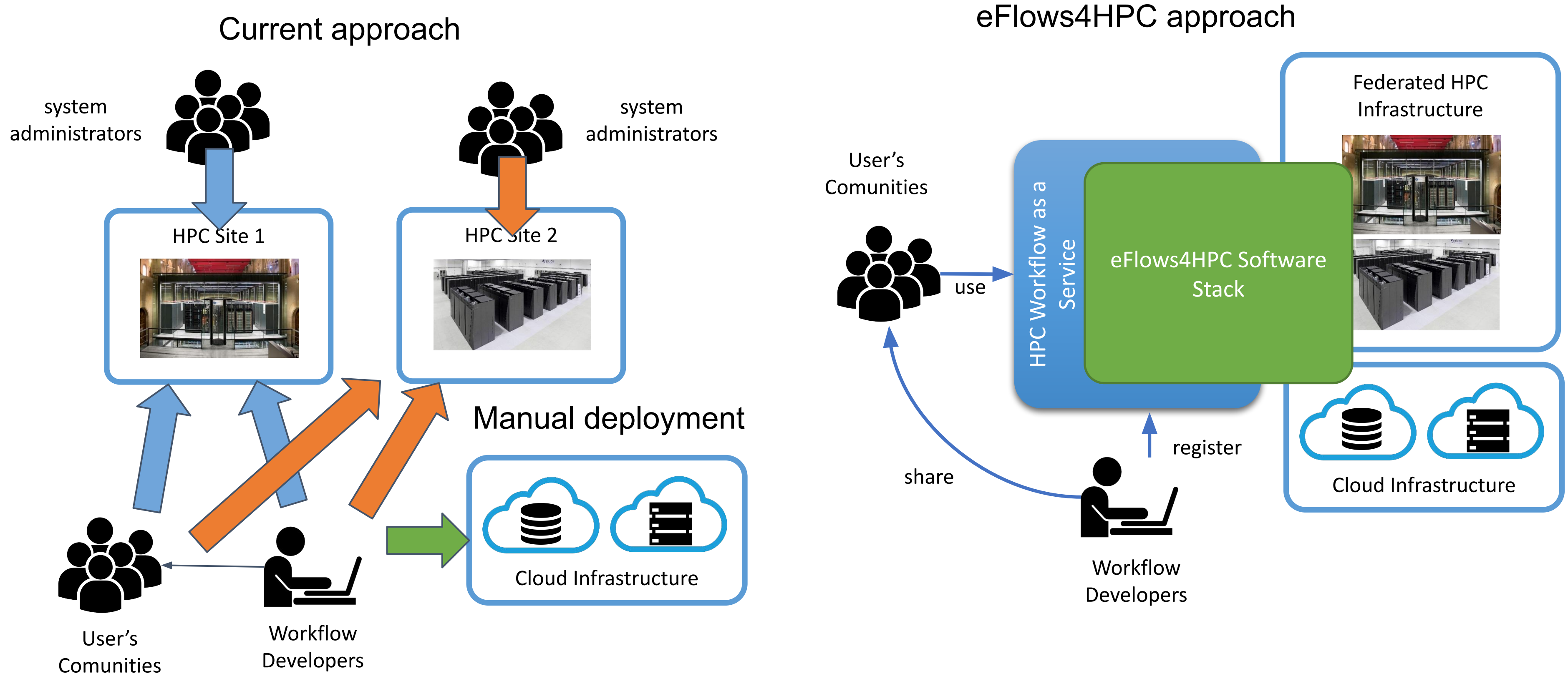
START DATE / END DATE

1 January 2020 / 31 December 2023

GRANT AGREEMENT ID

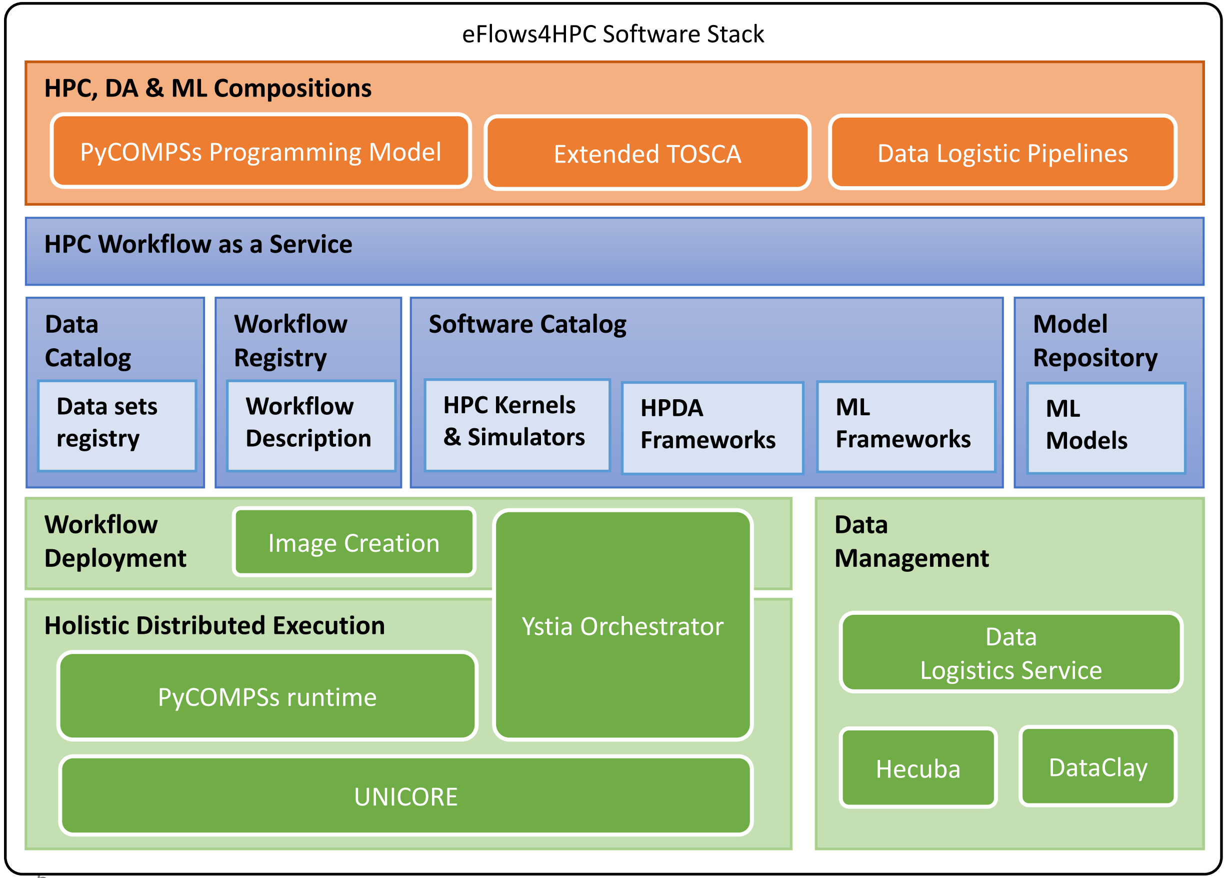
955558

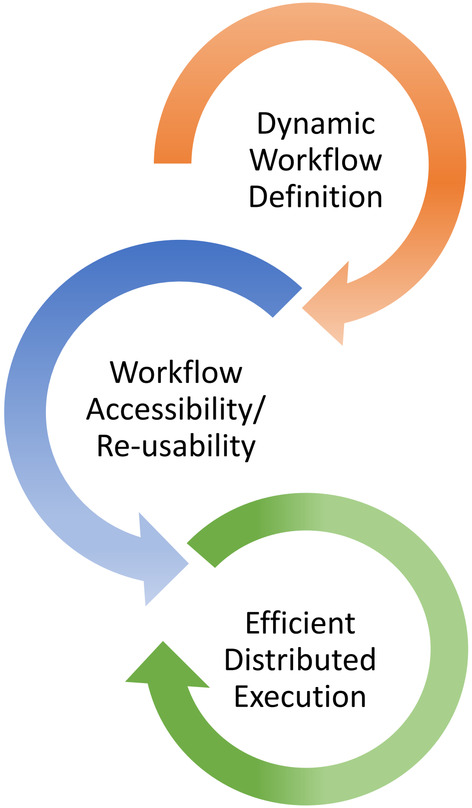
Motivation



Current approach
system administrators
system administrators
HPC Site 1
HPC Site 2
Manual deployment
Cloud Infrastructure
User's Comunities
Workflow Developers
eFlows4HPC approach
Federated HPC Infrastructure
User's Comunities
use
HPC Workflow as a Service
eFlows4HPC Software Stack
register
share
Cloud Infrastructure
Workflow Developers

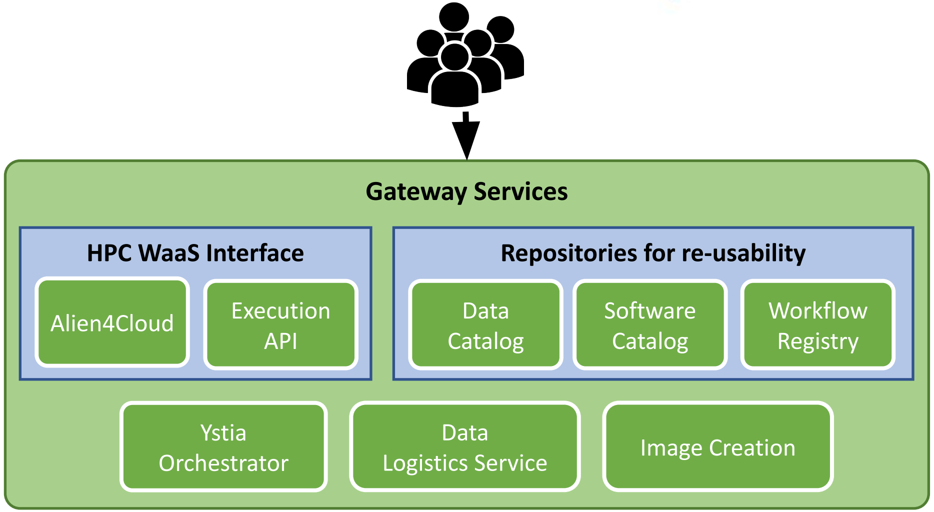
Software Stack overview

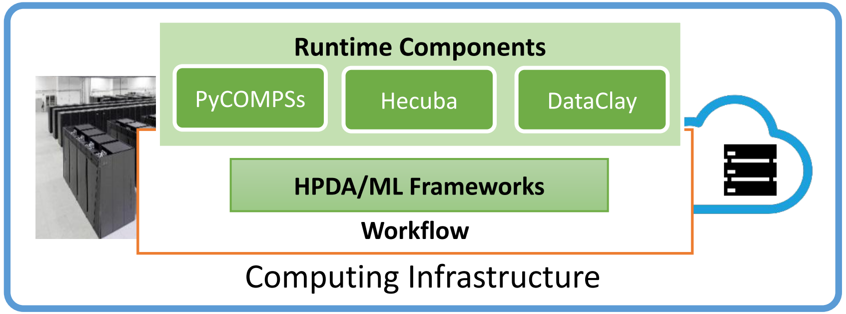




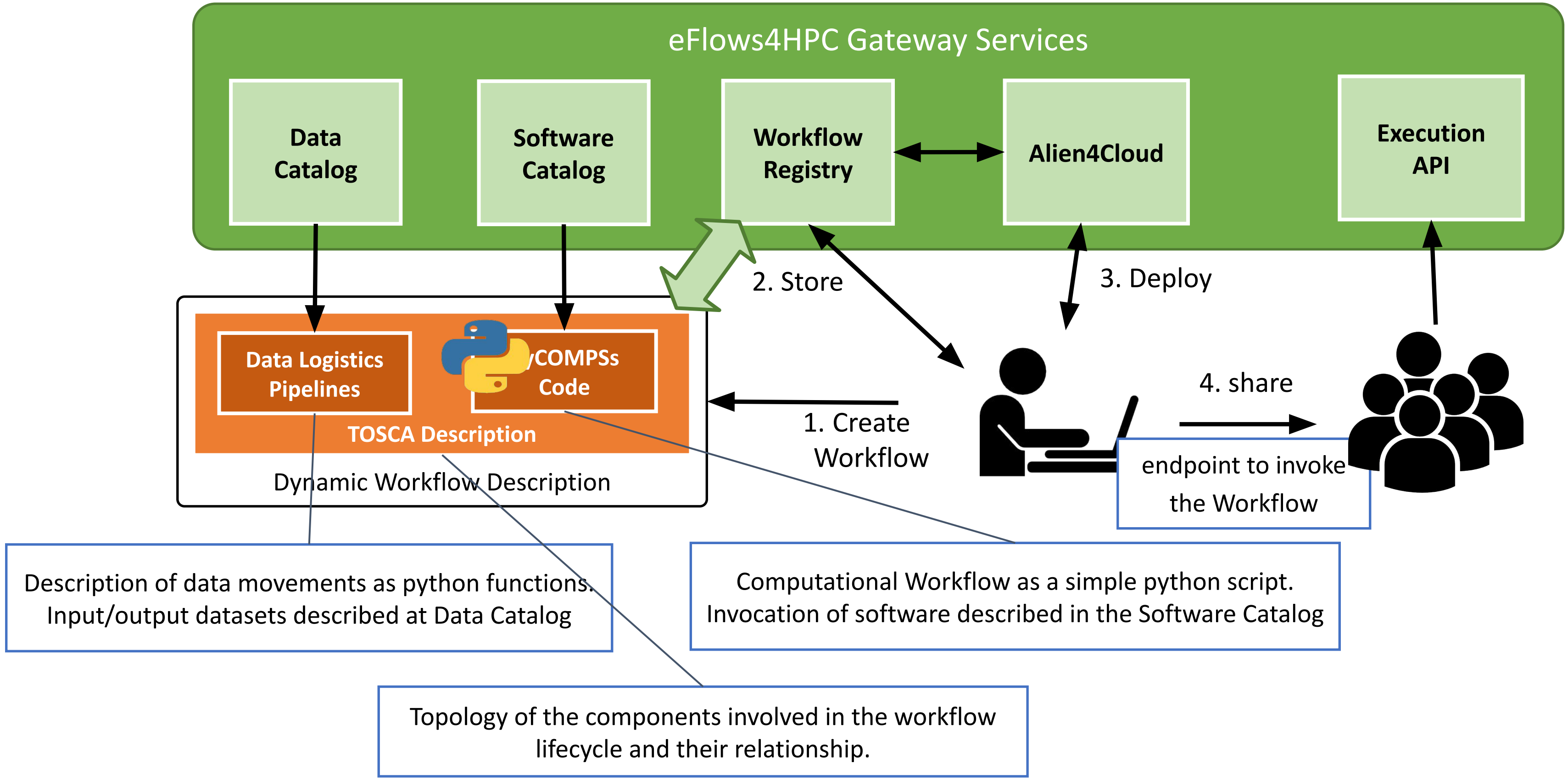
eFlows4HPC software stack and HPCWaaS

- **Gateway services**
  - Components deployed outside the computing infrastructure.
  - Managing external interactions and workflow lifecycle
- **Runtime Components**
  - Deployed inside the computing infrastructure to manage the workflow execution



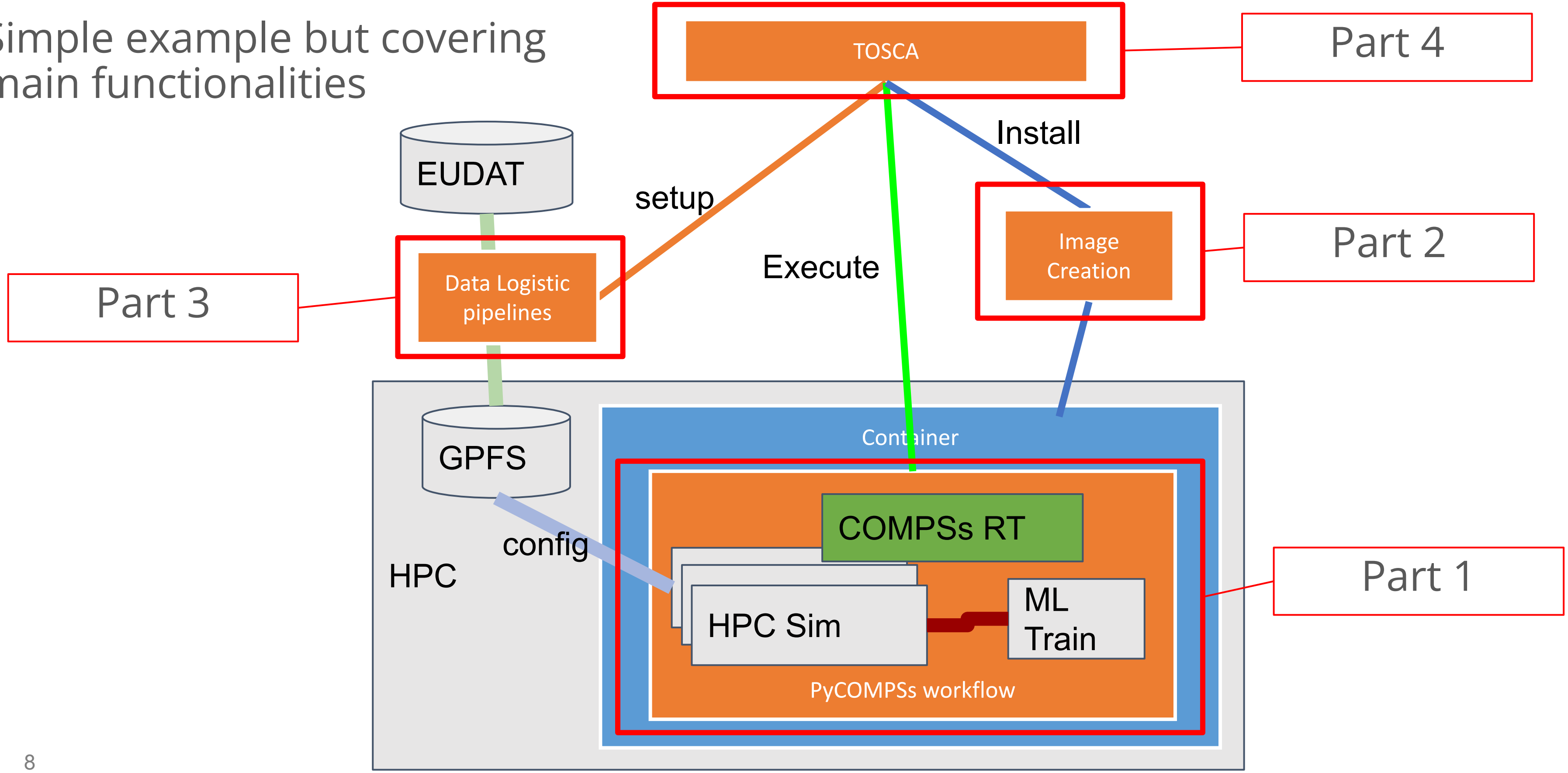


Workflow development overview



Minimal workflow

Simple example but covering main functionalities



Part 1: Integrating different computations in PyCOMPSs

Jorge Ejarque

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.

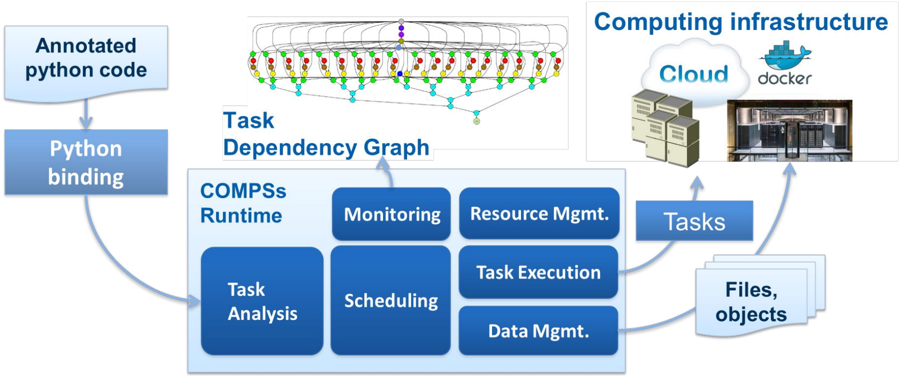
Motivation

eFlows4HPC

- **Complex workflows are composed by the execution of different software**
- **Workflow Developers dedicating time to develop glue code to integrate different software**
- **Reusable way to describe software executions**

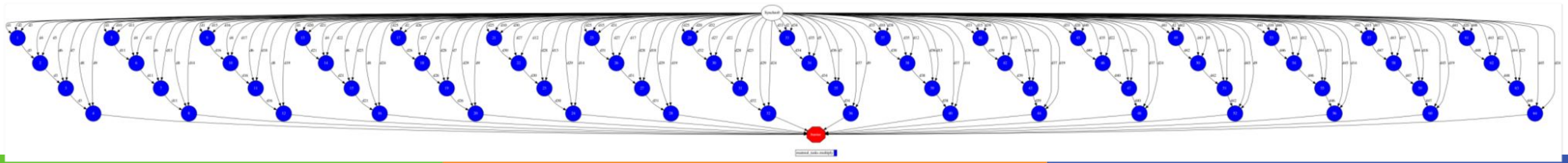
PyCOMPSs Overview

- **Sequential programming with simple linear address space**
- **Standard python + annotations/hints**
  - To identify tasks and directionality of data
- **Builds a dynamic task graph at runtime to infer potential concurrency**
- **Agnostic of computing platform**
  - Enabled by the runtime for clusters, clouds and containers

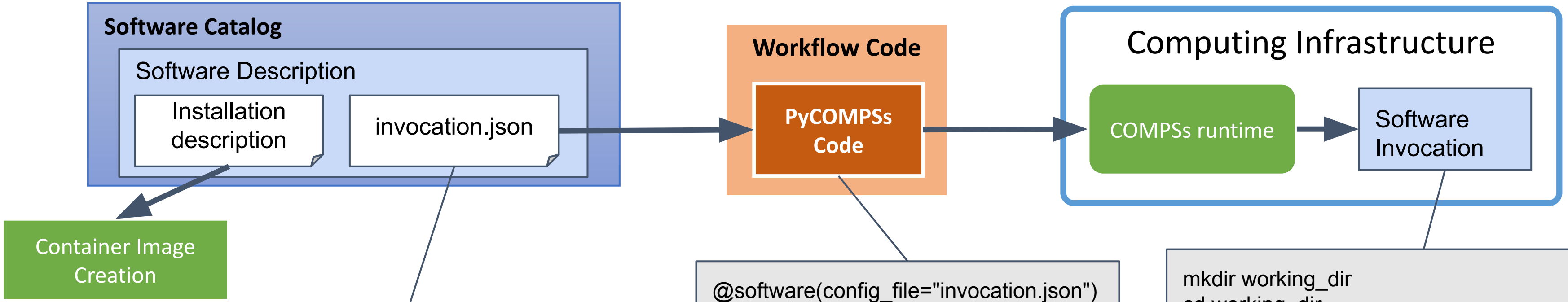



```python
initialize_variables()
startMulTime = time.time()
for i in range(MSIZE):
    for j in range(MSIZE):
        for k in range(MSIZE):
            multiply (A[i][k], B[k][j], C[i][j])
compss_barrier()
mulTime = time.time() - startMulTime
```

```python
@task(c=INOUT)
def multiply(a, b, c):
    c += a*b
```



Software Invocation description




```
{
  "type":"mpi",
  "properties":{
     "runner": "mpirun",
     “processes”: “$SW_PROCS”
     "binary": "mpi_sofware.x",
     "params": "-d {{param}}",
     “working_dir”: “{{working_dir}}”},
  "prolog":{
       "binary":"mkdir",
       "params":"{{working_dir}}"},
   "epilog":{
       "binary":"tar",
       "params":"zcvf {{out_tgz}}" {{working_dir}}},
  "constraints":{
     "computing_units": $SW_THREADS}
}
```

```
@software(config_file="invocation.json")
def mpi_exec(work_dir, param, out_tgz):
    pass

#Call to the task function:

mpi_exec('my_folder', 'hello_world')
```

```
mkdir working_dir
cd working_dir
export OMP_NUM_THREADS=$SW_THREADS
mpirun -n $SW_PROCS mpi_software.x -d param
tar zcvf out_tgz working_dir
```

- **Converts a python function to software invocation as a PyCOMPSs task**
- **Reused in different workflows**

Interfaces to integrate HPC/DA/ML

eFlows4HPC

```
@software(config_file="mpi_w_params.json")
@task()
def task_mpi_w_params(work_dir, param_d):
     pass


#Call to the task function:

task_mpi_w_params('my_folder', 'hello_world')
```

```
{
  "type": …,
  "properties":{
     … },

  "prolog":{
       …   },

   "epilog":{
        …   },

  "constraints":{
      ….  },

  “container:”{
      … }
}
```

Mandatory

Execution type and properties

Optional

Command to run before and after the execution.

resource required by the execution

Container containing the required software for the execution

Supported Types

Execution Type	Implementation	Properties
binary	pass	binary, params, working_dir
mpi	pass	runner, binary, params, processes, ppn, working_dir
	python	runner, processes, ppn, working_dir
mpmd_mpi	pass	runner, ppn, programs":[{binary, params, processes}]

Other properties

Property	Argument
constraints	computing_units, memory,...
prolog/ epilog	binary, params working_dir
container	image, engine

Using task parameter or environment variables

- **Task parameters:**
  - can be referred as {{param_name}} in:
    - execution type properties
    - epilog/prolog parameters

- **Environment Variables:**
  - can be referred as $ENV_VAR_NAME in:
    - execution type properties
    - epilog/prolog parameters
    - constraints
    - container

Examples

Examples

```
{
    "type":"mpmd_mpi",
    "properties":{
            "ppn":$CPUS_PER_NODE,
            "runner":"mpirun",
            “working_dir”: “{{working_dir}}”
            "programs":[
                    {
                            "processes": “$FESOM_PROCS”,
                            "binary":"$FESOM_EXEC",
                            "params":"{{fesom_param}}"
                    },
                    {
                            "processes": “$OIFS_PROCS”,
                            "binary":"$OIFS_EXEC",
                            "params":"{{oifs_param}}"
                    }
            ]
    }
}
```

```
@software(config=fesom_oifs_mpmd.json')
@task(log=FILE_OUT_STDOUT)
def simulation(working_dir, fesom_param, oifs_param, log):
  #mpirun -n X -ppn Y fesom.x fesom_param : -n M -ppn  oifs oifs_param > log
  pass
```

```
sim_cfgs=generate_simulation_cfg()
results = []
for cfg in sim_cfgs:
        simulation (cfg.wdir, cfg.fesom_param, cfg, oifs_param, “out.txt”)
        result = post_process(“out.txt”)
        results.append(result)
evaluate(results)
```

Examples

```
{
  "type":"mpi",
  "properties":{
     "runner": "mpirun",
     “processes”: “$MPI_PROCS”
  },
  "prolog":{
       "binary":"ln",
       "params":"-s {{rom}} RomParameters.json"
   },
   "epilog":{
       "binary":"rm",
       "params":"RomParameters.json"
   },
  "constraints":{
     "computing_units": $OMP_THREADS
  }
}
```

```
@software(config=kratos_rom_mpi.json')
@task(rom=FILE_IN, returns=1)
def execute_rom(parameters, rom):
   “”” kratos python mpi code “””
   return result
```

```
{
  "type":"binary",
  "properties":{
     “binary”: “gmx”
     "params": "mdrun -s {em}} -e {{em_energy}}",
  },

  "constraints":{
     "processors": [{“processorType”: “GPU”,
“computingUnits”:1}]
  }

  “container”:{
     “engine” : ”SINGULARITY”
     “image” : “/path/to/gromacs.sif”
  }
}
```

```
@software(config=gromacs_mdrun_gpu.json')
@task(em=FILE_IN,   em_energy=FILE_OUT)
def energy_minimization(em, em_energy):
          pass
```

```
singularity exec -nv /path/to/gromacs.sif  \
     gmx mdrun -s /path/to/em -e /path/to/em_energy
```

Questions

eFlows4HPC

Enabling dynamic and Intelligent workflows
in the future EuroHPC ecosystem

www.eFlows4HPC.eu

@eFlows4HPC eFlows4HPC Project

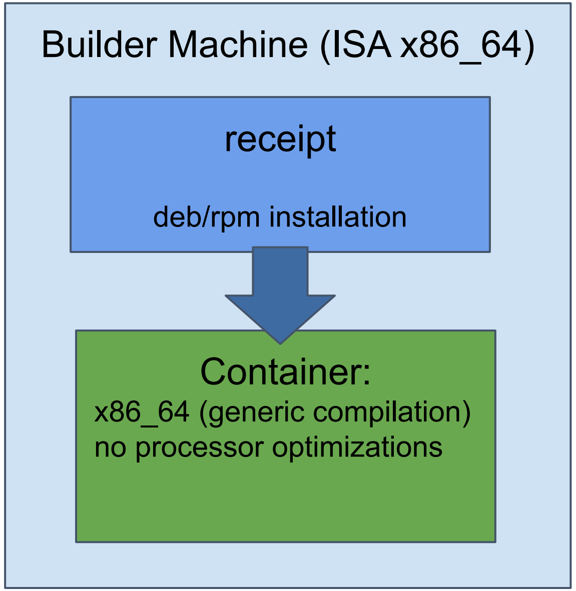
This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955558. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Germany, France, Italy, Poland, Switzerland, Norway.

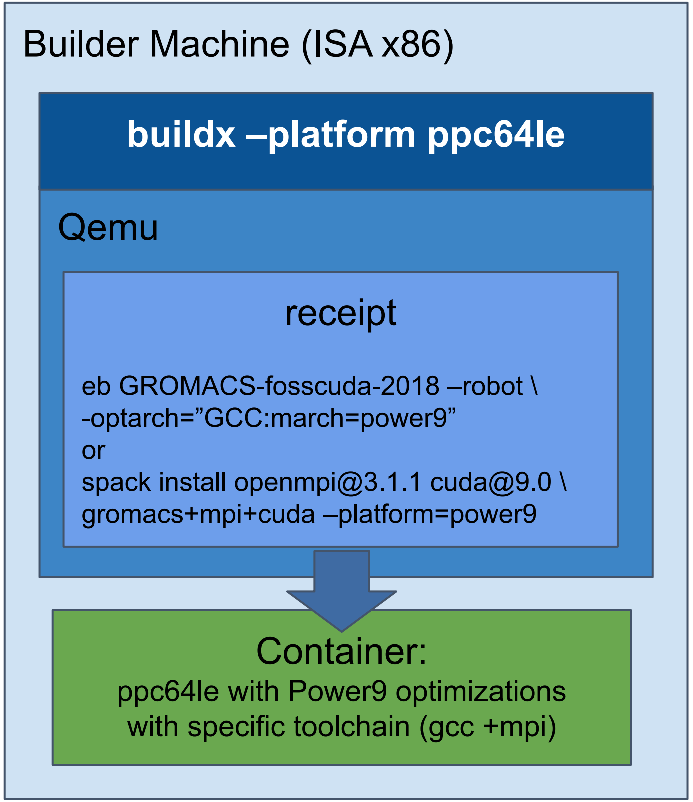
Part 2: HPC ready containers

Jorge Ejarque

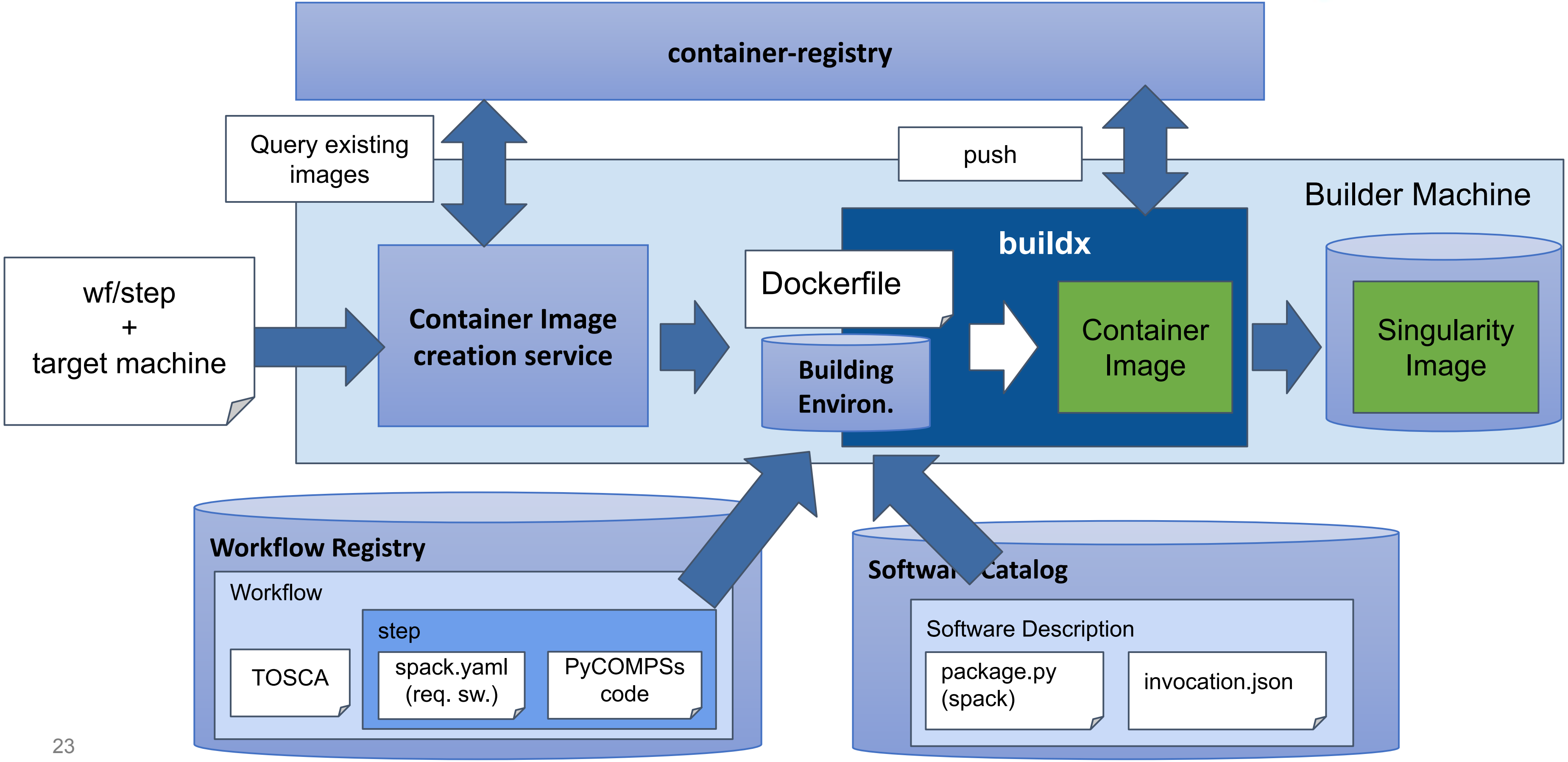
This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955558. The JU receives support from the European Union’s Horizon 2020 research and innovation programme and Spain, Germany, France, Italy, Poland, Switzerland, Norway.

HPC ready containers





HPC Software Deployment



HPC Software Deployment

eFlows4HPC

Software Catalog

Software Description

package.py

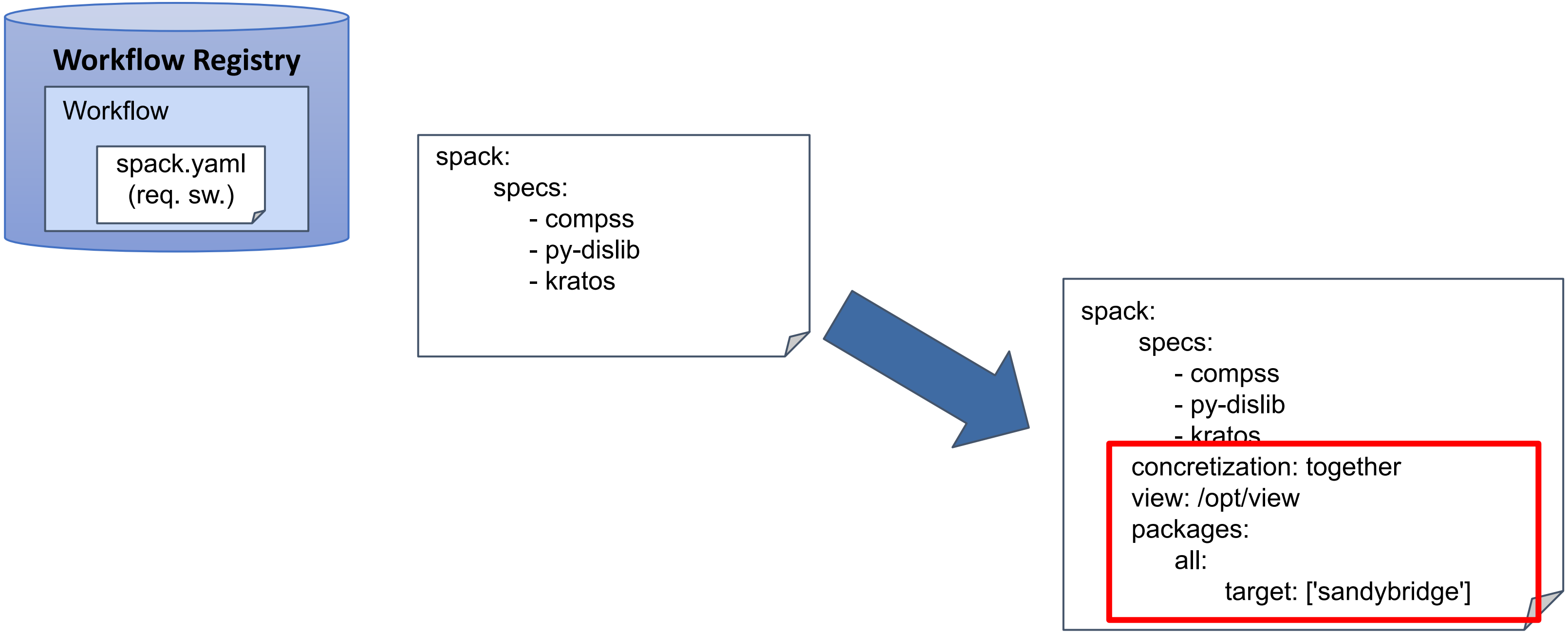
```
class Compss(Package):
    url = "https://compss.bsc.es/repo/sc/stable/COMPSs_2.10
    version('2.10', sha256='...', preferred=True)
     ....
    # dependencies.
    depends_on('python')
    depends_on('openjdk')
    depends_on('boost')
    …
    def install(self, spec, prefix):
        install_script = Executable('./install')
        install_script('-A', '--only-python-3', prefix.compss)

    def setup_run_environment(self, env):
        env.set('COMPSS_HOME', self.prefix.compss)
        env.prepend_path('PATH', self.prefix.compss + '/Runtime/scripts/user')

    def setup_build_environment(self, env):
            ….
```

```
class Kratos(CMakePackage):
        …
        #variant
        variant('mpi', default=False, description='Builds a MPI version of the library')
        depends_on('mpi', when='+mpi')
        …
        def cmake_args(self):
                args = []
                if self.spec.variants['mpi'].value == True:
                        args.append('USE_MPI=ON')
                else:
                        args.append('USE_MPI=OFF')
                return args
        ….
```

```
class PyDislib(PythonPackage):
        …
        # if you need specific versions.
        depends_on('python@3:', type=('build', 'run'))
        depends_on('py-scikit-learn@0.23^py-scipy@1.5,type=('run'))
        …
```

https://spack-tutorial.readthedocs.io/en/latest/tutorial_packaging.html#creating-the-package-file

HPC Software Deployment



Container Creation API

- **Request the image creation**

Request

```
POST /build/
```

```
{
  "machine": {
    "platform": "linux/amd64",
    "architecture": "rome",
    "container_engine": "singularity"},
  "workflow":"minimal_workflow",
  "step_id" :"wordcount",
  "force": False
}
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "id": "<creation_id>"
}
```

```
jorgee@localhost:~/eFlows4HPC/git/image_creation> ./client.sh <user> <passwd>
https://<image_creation_url> build <request.json>
Response:
{"id":"f1f4699b-9048-4ecc-aff3-1c689b855adc"}
```

Container Creation API

- **Check build status**

Request

```
GET /build/<creation_id>
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "status": "< PENDING | STARTED | BUILDING | CONVERTING | FINISHED | FAILED >",
  "message": "< Error message in case of failure >",
  "image_id": "< Generated docker image id >",
  "filename": "< Generated singularity image filename >"
}
```

```
jorgee@localhost:~/eFlows4HPC/git/image_creation> ./client.sh <user> <passwd>
https://<image_creation_url> status f1f4699b-9048-4ecc-aff3-1c689b855adc
Response:
{"filename":"image_sandybridge.sif","image_id":"ghcr.io/eflows4hpc/image_sandybridge","message":null,"
status":"FINISHED"}
```

Container Creation API

- ## Get generated Image

Request

```
GET /images/download/<Generated singularity image filename>
```

Response

```
HTTP/1.1 200 OK
Content-Disposition: attachment
Content-Type: application/binary
```

```
jorgee@localhost:~/eFlows4HPC/git/image_creation> ./client.sh <user> <passwd>
https://<image_creation_url> download <image_file.sif>

HTTP request sent, awaiting response... 200 OK
Length: 2339000320 (2.2G) [application/octet-stream]
Saving to: ‘image_file.sif’

image_file.sif          0%[                           ]   4.35M   550KB/s     eta 79m 0s
```

Demo

Image creation simple CLI

```
jorgee@localhost:~/eFlows4HPC/git/image_creation> ./client.sh test T3st22 https://bscgrid20.bsc.es
build test_request.json
Response:
{"id":"f1f4699b-9048-4ecc-aff3-1c689b855adc"}

jorgee@localhost:~/eFlows4HPC/git/image_creation> ./client.sh test T3st22 https://bscgrid20.bsc.es
status f1f4699b-9048-4ecc-aff3-1c689b855adc
Response:
{"filename":"reduce_order_model_sandybridge.sif","image_id":"ghcr.io/eflows4hpc/reduce_order_model_san
dybridge","message":null,"status":"FINISHED"}

jorgee@localhost:~/eFlows4HPC/git/image_creation> ./client.sh test T3st22 https://bscgrid20.bsc.es
download reduce_order_model_sandybridge.sif
--2022-05-24 16:01:28--
https://bscgrid20.bsc.es/image_creation/images/download/reduce_order_model_sandybridge.sif
Resolving bscgrid20.bsc.es (bscgrid20.bsc.es)... 84.88.52.251
Connecting to bscgrid20.bsc.es (bscgrid20.bsc.es)|84.88.52.251|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2339000320 (2.2G) [application/octet-stream]
Saving to: ‘reduce_order_model_sandybridge.sif’

reduce_order_model_sandybridge.sif       0%[                           ]   4.35M   550KB/s    eta
79m 0s
```

Next Steps

- **Software Integration**
  - Introduce data transformations
  - Other task types if required

- **Image Creation**
  - Include versioning
  - Improve CLI
  - Test with MPI versions
    - concretize with an specific MPI version
  - Include other devices GPUs
  - Populate repositories

Questions

eFlows4HPC

Enabling dynamic and Intelligent workflows in the future EuroHPC ecosystem

www.eFlows4HPC.eu

@eFlows4HPC eFlows4HPC Project

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.

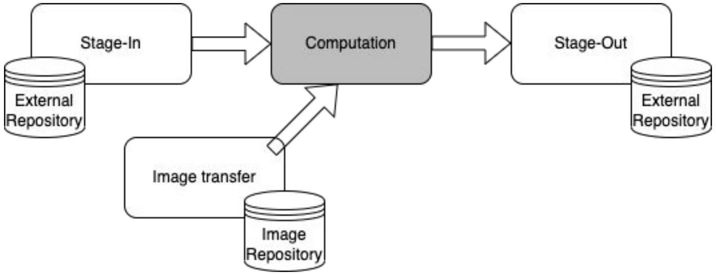
Part 3: Data Pipelines and Data Logistic Service

Jedrzej Rybicki (j.rybicki@fz-juelich.de)

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.

Data Logistics Service Backgrounds

- **Computations require (lots) of data**
  - DLS: fuel the scientific calculations with required data
  - DLS pipelines describe how the data are moved
- **Formalization of the data movements**
  - Reproducibility (FAIR)
  - Integrated with Data Catalogue
- **DLS is part of eFlows4HPC Workflow-as-a-Service**
- **Minimal Workflow**
  - Stage-in and –out
  - Singularity image transfer



Apache Airflow

- **Apache Airflow: a platform to programmatically author, schedule and monitor workflows**
- **Workflows (pipelines) as directed acyclic graphs (DAGs) of tasks**
  - ⇒ i.e. tasks depend on each other
- **Airflow Scheduler executes your tasks on an array of Workers**
- **User interface to visualize pipelines, monitor progress, and troubleshoot issues when needed**

Tasks

- **Task: unit of execution in Airflow**
- **Types:**
  - Operators
  - Sensors
- **Task instances (for each DAG run)**
- **Task relationships (classical):**
  - `Task1 >> Task2 >> Task3`
  - `TaskFlow`
  - or mixture of both

```python
@dag(default_args=default_args)
def my_pipeline():
    @task
    def get_url(**kwargs):
        hook = DataCatalogHook()
        return hook.get_entry(id='foo')

    @task
    def move(url, **kwargs):
        print(f"Moving {url}")

    url = get_url()
    move(url)

dag = my_pipeline()
```

Tasks: Operators

- BashOperator - executes a bash command
- PythonOperator - calls an arbitrary Python function
- EmailOperator - sends an email

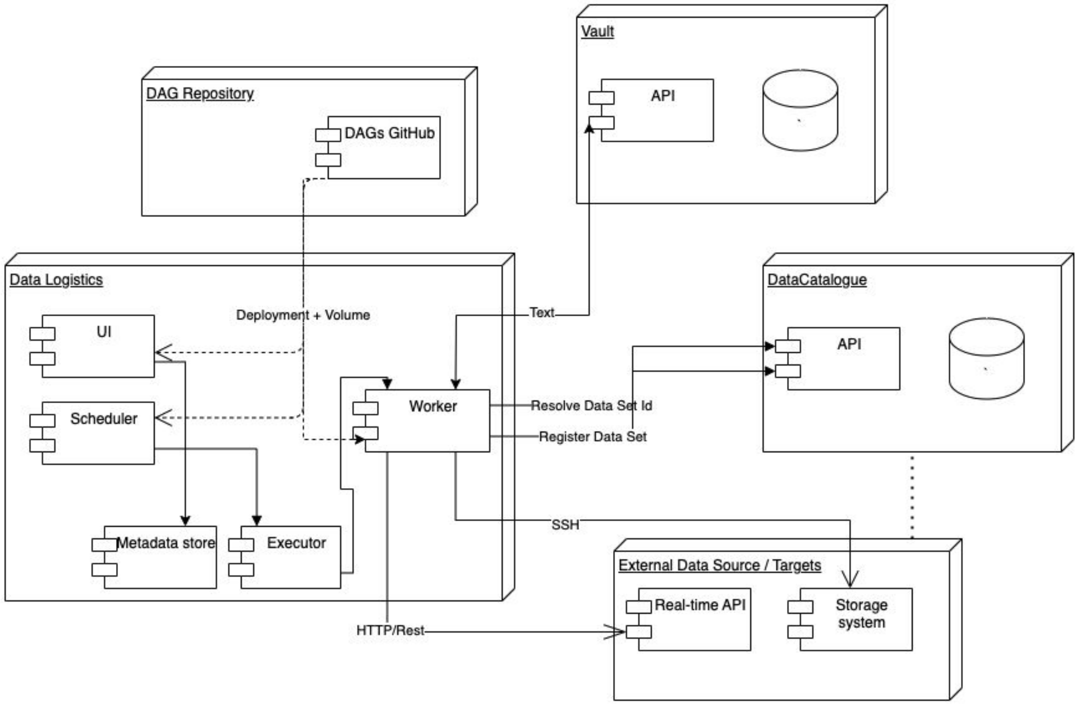
Contributed:

- SimpleHttpOperator, SSHOperator, (S)FTPOperator
- MySqlOperator, PostgresOperator, MsSqlOperator, OracleOperator
- DockerOperator, HiveOperator, SingularityOperator, Kubernetes
- SlackAPIOperator, DiscordOperator

Scheduling

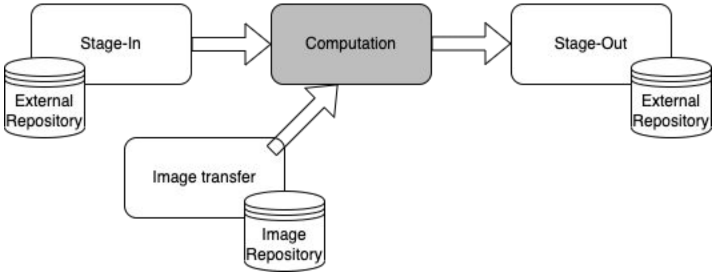
- `schedule_interval` - e.g. `@hourly`, `timedelta(days=1)`, or cron expressions 0 0 1 * *
- `start_date` (and optionally `end_date`) - defines series of intervals
- `catchup=True` - indicates if the scheduler should create a DAG run for each interval that has not been run
- `execution_date` - injected into task is not the current time but rather a indication of the interval
- SLAs: new feature allows to define maximum time a task should take
- *Pause/unpause*

DLS Architecture



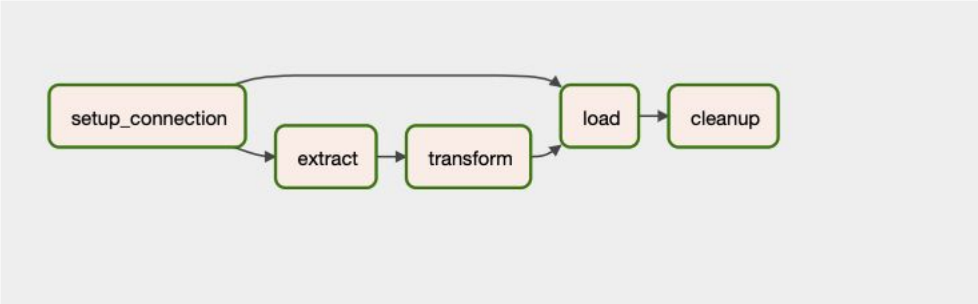
Intro: Minimal Workflow

- **Idea: model a typical computation workflow across Pillars**
  - serves as a guinea pig for the solution and infrastructure
  - basis for concrete Pillars’ implementations
- **Overview**



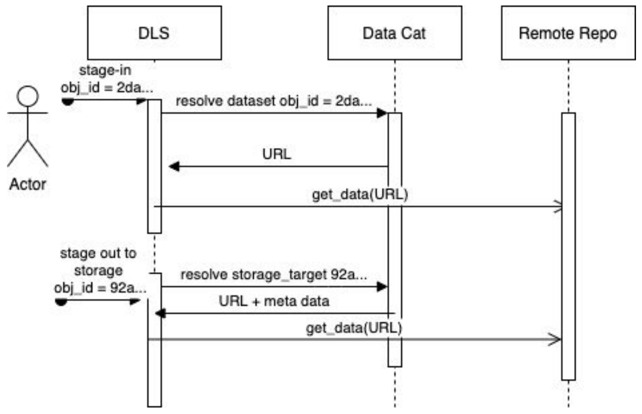
Stage-In Pipeline

- **Move data into processing facility**
  - Source: B2SHARE
  - Target: BSC (SSH)
- **Phases**
  - Extract list of files from a repository object
  - Upload to target
- **Challenges:**
  - Credentials management
  - Genericity



Challenge: Generic pipelines

- **Genericity**
  - same data stage-in workflow could be used to transfer different data
  - Using Data Catalogue
  - Pipeline input is id of a data set from Data Cat



https://datacatalogue.eflows4hpc.eu/storage.html?type=dataset&oid=23a1ed6b-682a-4682-887e-e3c17b9d69ea

https://datacatalogue.eflows4hpc.eu/dataset/23a1ed6b-682a-4682-887e-e3c17b9d69ea

- **Metadata management (in a later step)**

DEMO

HTTPS://DATALOGISTICS.EFLOWS4HPC.EU/

Data Logistics Service

Summary:

- Minimal workflow
- Reproducible data pipelines: Improvements in transparency and turn-around times
- Help in tedious tasks ⇒ more time for actual science
- Towards self-service:
  - Automatic deployment + deps management
  - PythonOperator + BashOperator (reuse of existing solutions)

Outlook:

- Pillars’ workflows
- Integrations: Model Repository + WP2 Storages
- Cloud-based processing
- Prefect?
  - High-reaching compatibility on dag/pipeline-level

Questions

eFlows4HPC

Enabling dynamic and Intelligent workflows in the future EuroHPC ecosystem

www.eFlows4HPC.eu

@eFlows4HPC

eFlows4HPC Project

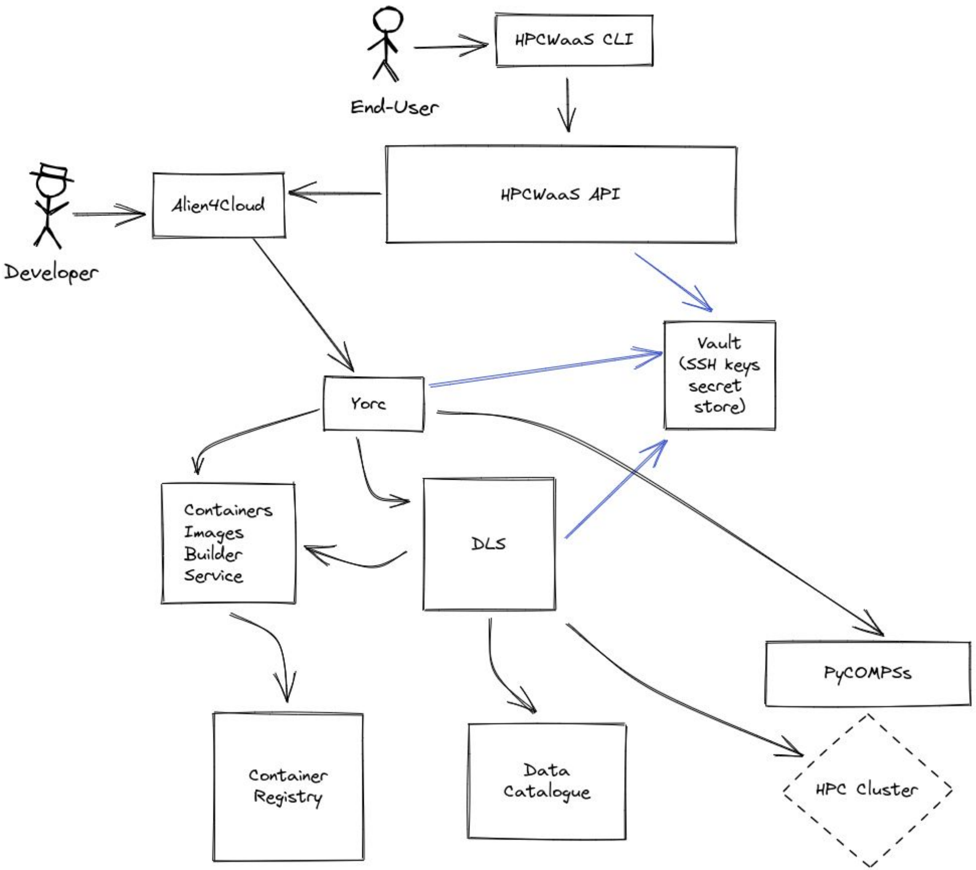
This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955558. The JU receives support from the European Union’s Horizon 2020 research and innovation programme and Spain, Germany, France, Italy, Poland, Switzerland, Norway.

Part 4: TOSCA Orchestration and HPCWaaS

Albertin, Loïc (loic.albertin@atos.net)

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.

Minimal workflow - The big picture



Minimal workflow - components

- HPCWaaS API/CLI: interface for the end-user that allows to control workflow executions
- Vault: Secret store used to securely store SSH credentials
- Alien4Cloud: interface for workflows developers to design & deploy workflows / also integrated with HPCWaaS API to manage executions
- Yorc: high level orchestration engine driven by Alien4Cloud
- DLS: orchestration engine for data movements (Datasets, images)
- PyCOMPSs: orchestration engine for computations
- Container Image Builder service: build container images fitting infrastructure constraints
- Container registry: store container images
- Data Catalogue: Store for datasets and computation results metadata

Minimal workflow - users

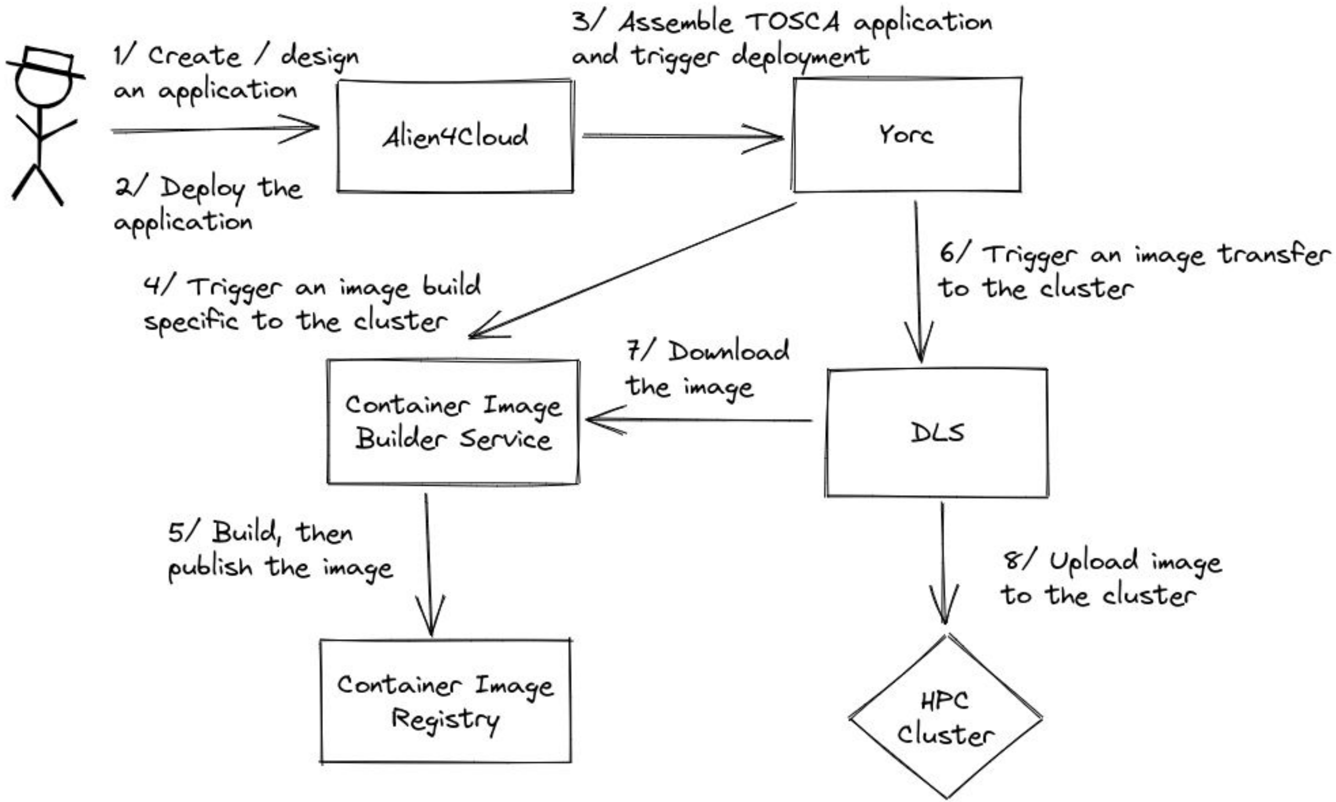
2 main users:

- The Developer is responsible for creating applications / high level workflows, to deploy and expose them to the end-user

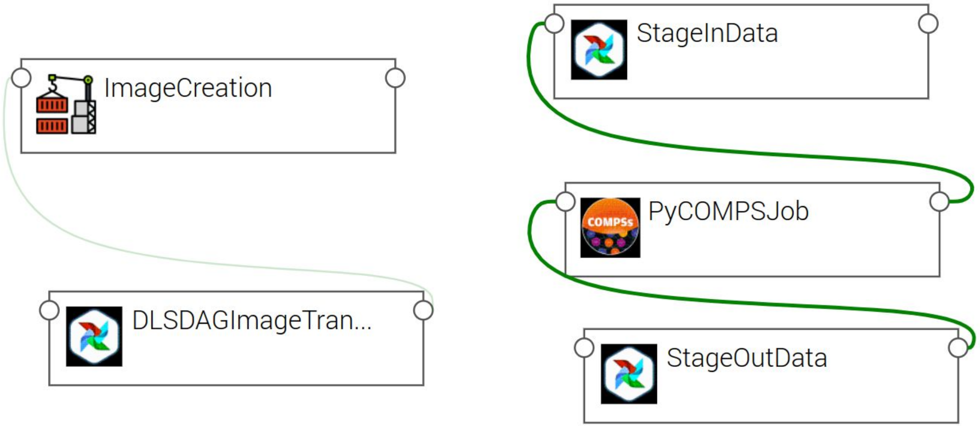
- End User essentially trigger and monitor executions of the workflows deployed by the developer.
  To do that he uses the HPCWaaS API which is specifically designed to abstract the complexity of using the orchestration stack

Developer point of view

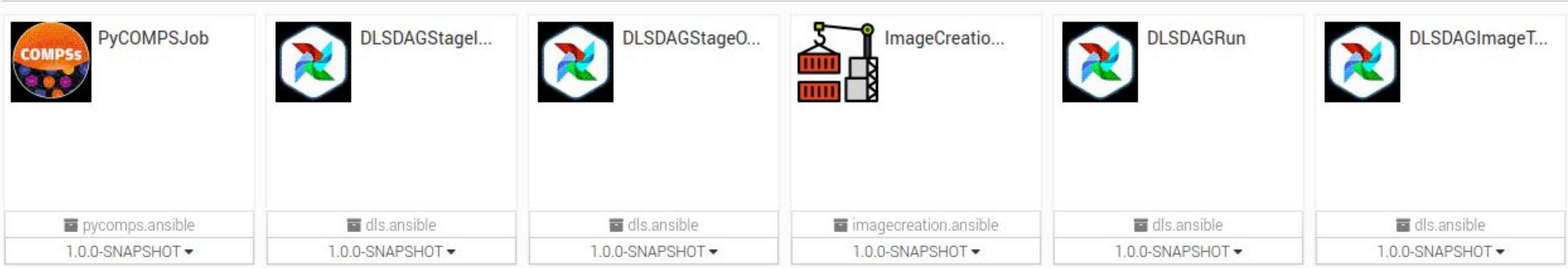
Minimal Workflow - seen as Developer



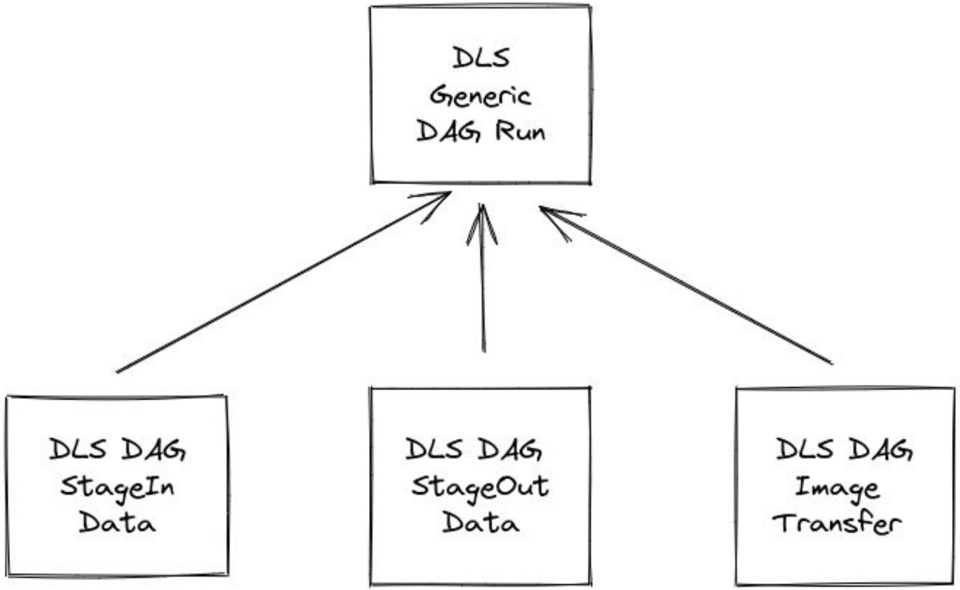
Minimal Workflow - TOSCA Modelization



eFlows4HPC TOSCA Components

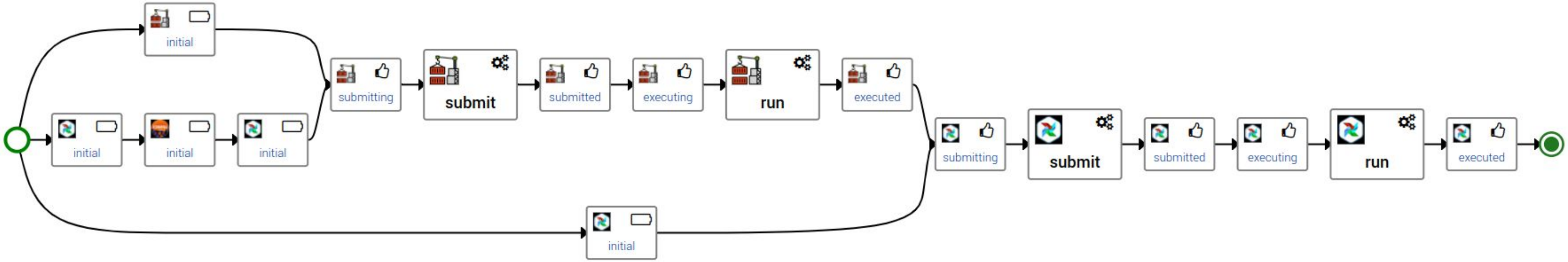


TOSCA Components - DLS hierarchy



Minimal Workflow - application deployment

Application deployment workflow (done once)



Minimal Workflow - resources for developers

TOSCA components:

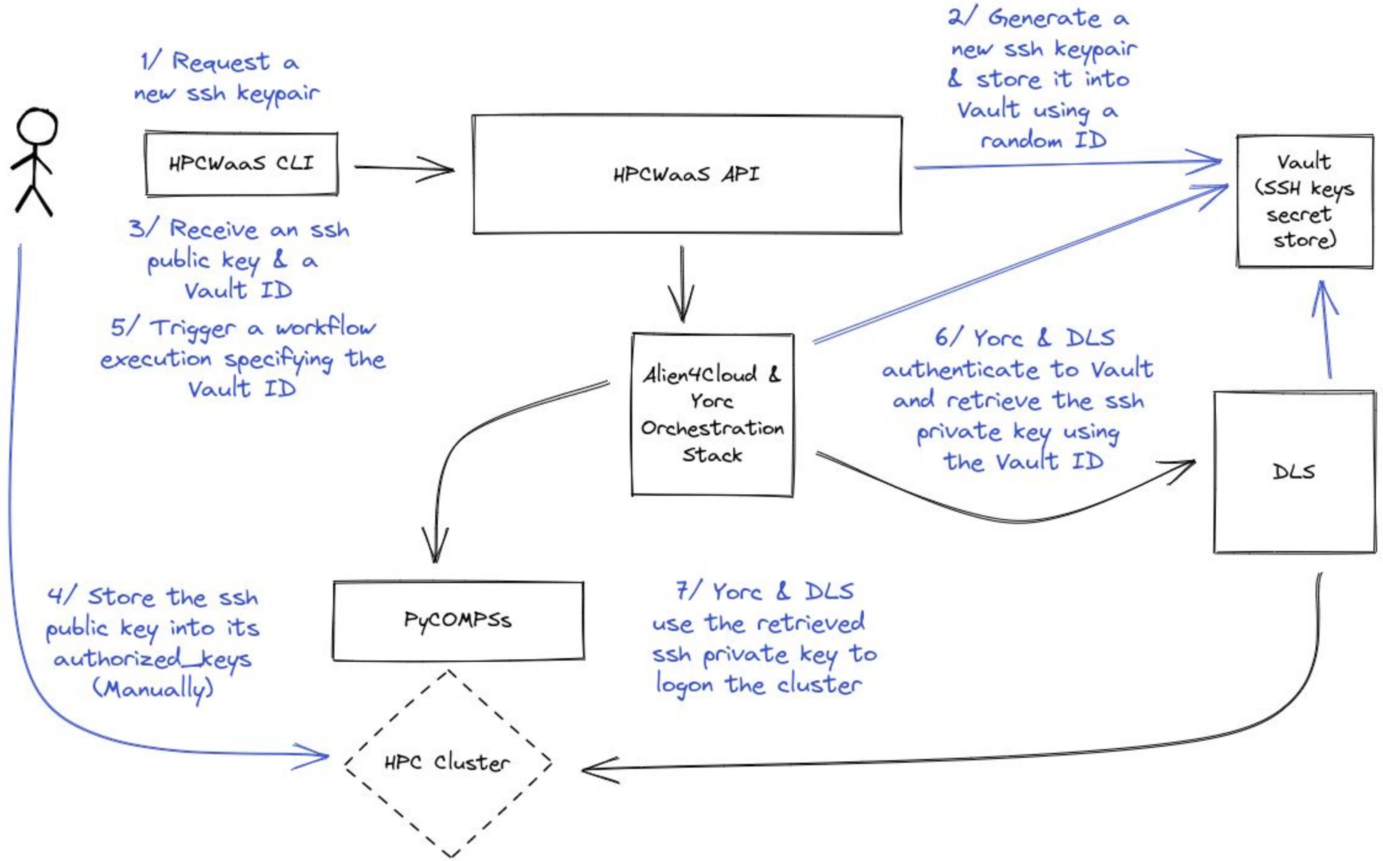
- DLS: https://github.com/eflows4hpc/dls-tosca
- PyCOMPSs: https://github.com/eflows4hpc/pycomps-tosca
- Image Creation: https://github.com/eflows4hpc/image_creation (tosca sub-folder)
- Minimal Workflow template: https://github.com/eflows4hpc/workflow-registry (minimal_workflow/tosca sub-folder)

TOSCA resources:

- https://alien4cloud.github.io/#/documentation/3.5.0/devops_guide/dev_ops_guide.html
- https://github.com/eflows4hpc/tosca-tutorial

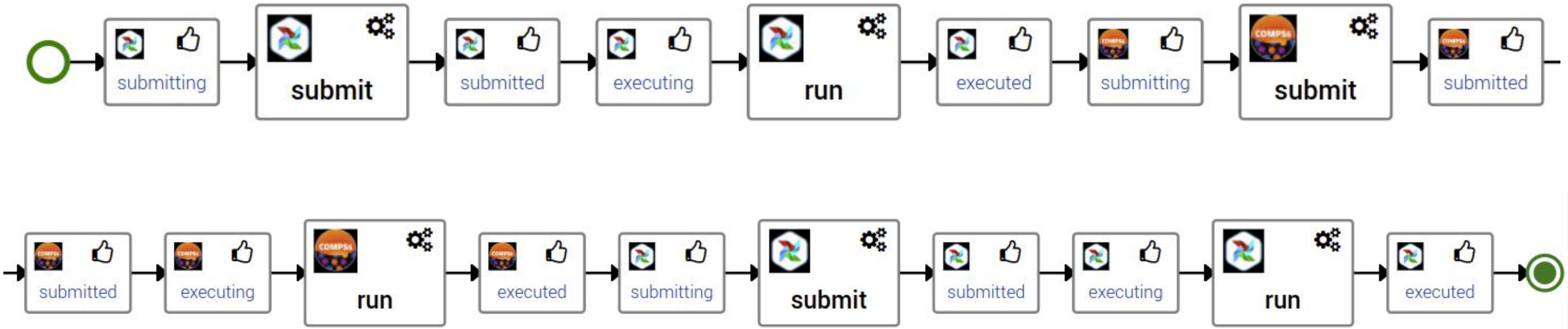
End-User point of view

Minimal workflow - seen as End user



Minimal Workflow - business workflow

End-User workflow (multiple executions)



HPCWaaS main CLI commands

- waas workflows list
- waas workflows trigger
- waas executions status
- waas executions cancel
- waas ssh_keys key-gen

Demo of the end-user workload

Minimal Workflow - resources for end-users

HPCWaas code/binaries:
https://github.com/eflows4hpc/hpcwaas-api

Thank you

eFlows4HPC

Enabling dynamic and Intelligent workflows in the future EuroHPC ecosystem

www.eFlows4HPC.eu

@eFlows4HPC

eFlows4HPC Project

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.