Simplifying the life-cycle management of HPC, data analytics and AI workflows

Jorge Ejarque (BSC), Rosa M Badia (BSC)

ACM Europe Summer School on HPC Computer Architectures for AI and Dedicated Applications, Barcelona, July 5, 2023

MINISTERIO DE CIENCIA E INNOVACIÓN
Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
AGENCIA ESTATAL DE INVESTIGACIÓN

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. MCIN/AEI/10.13039/501100011033 and the European Union NextGenerationEU/PRTR (PCI2021-121957)

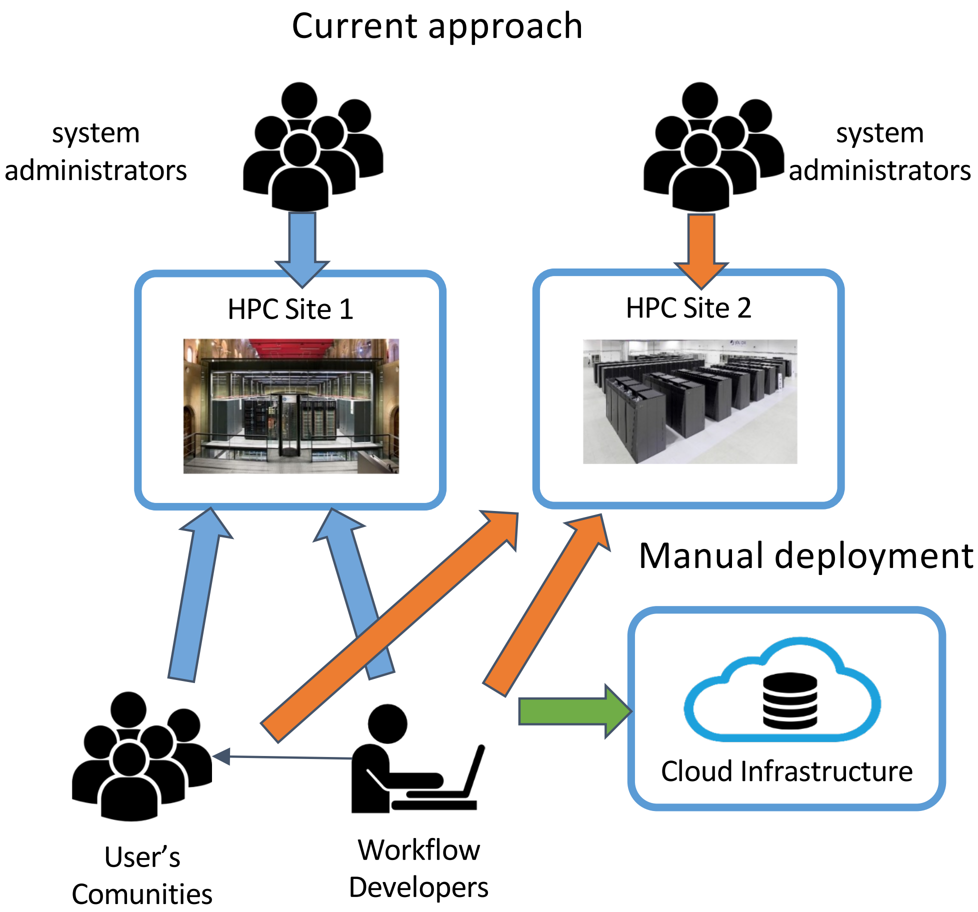
Agenda

eFlows4HPC

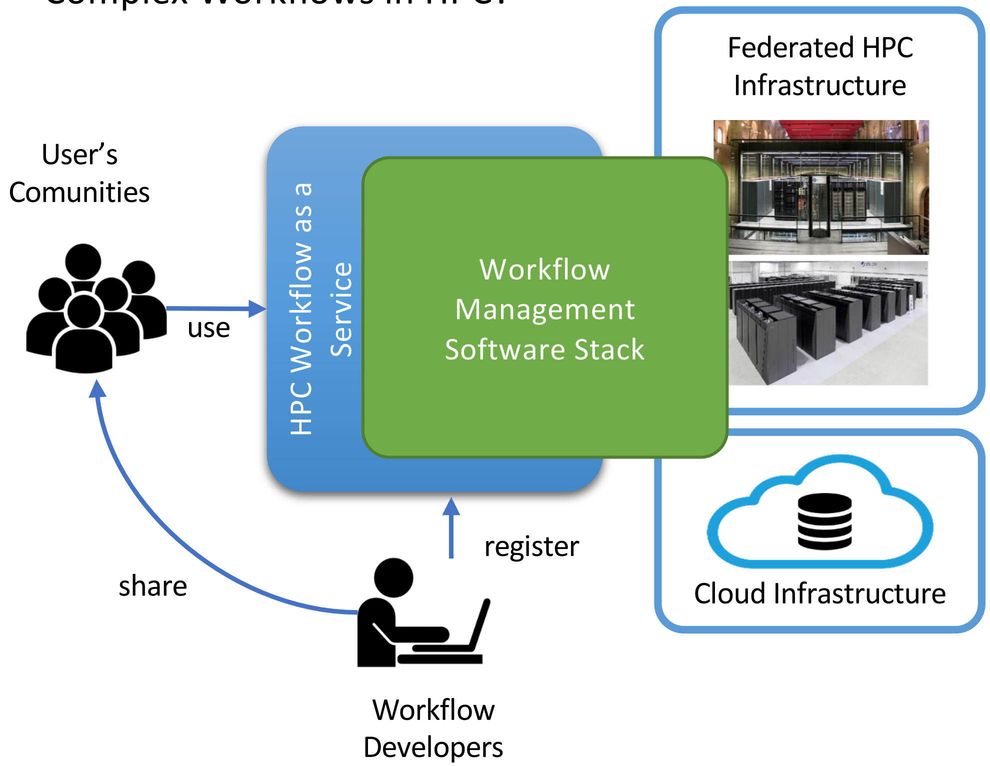
14:00 – 15:25	Lectures: - Overview of eFlows4HPC project - Integrating different computations in PyCOMPSs - HPC ready container images - Data pipelines and top-level workflow in TOSCA	Rosa M Badia
15:25 - 15:40	Bio Break	
15:40 - 16:55	Demo/ Hands-on session: How to build and deploy HPC Ready containers	Jorge Ejarque
16:55 - 17:00	Session conclusions	all presenters

OVERVIEW OF EFLOWS4HPC

HPC Environments

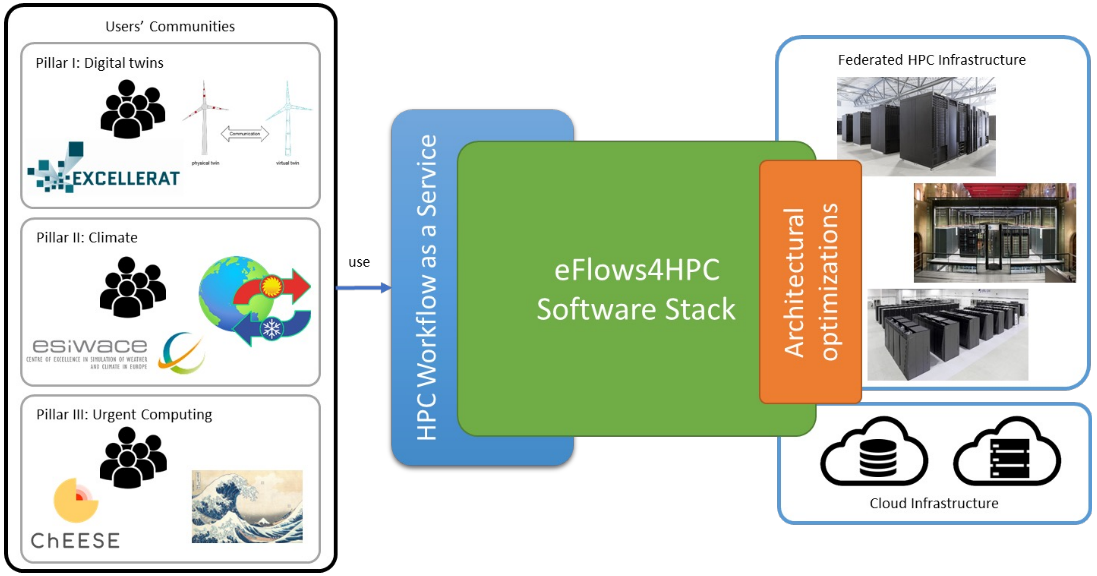


Can we apply something like FaaS for Complex Workflows in HPC?



eFlows4HPC in a nutshell

- **Software tools stack that makes easier the development and management of complex workflows:**
  - Combine different frameworks
    - HPC, AI, data analytics
  - Reactive and dynamic workflows
    - Autonomous workflow steering
  - Full lifecycle management
    - Not just execution
    - Data logistics and Deployment
- **HPC Workflows as a Service:**
  - Mechanisms to make easier the use and reuse of HPC by wider communities
- **Architectural Optimizations:**
  - Selected HPC – AI Kernels Optimized for GPUs, FPGA, EPI
- **Validation Pillar's**
  - End-user workflows linked to CoEs



Pillar I: Manufacturing

eFlows4HPC

CIMNE

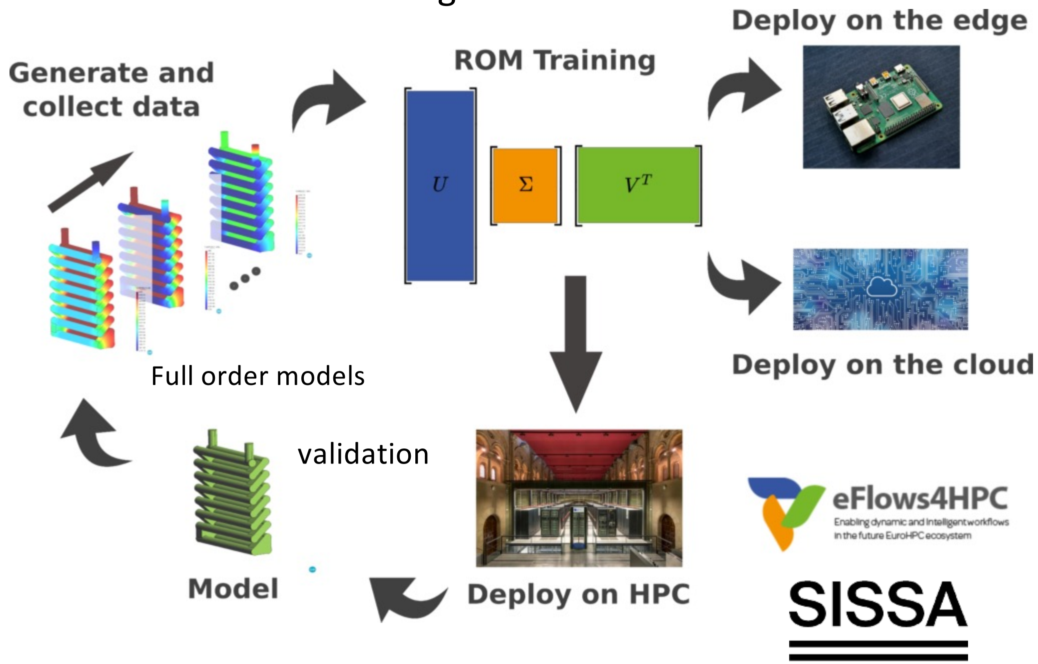
KRATOS MULTI-PHYSICS

Inria

upgrade your meshes

ParMMg

SIEMENS

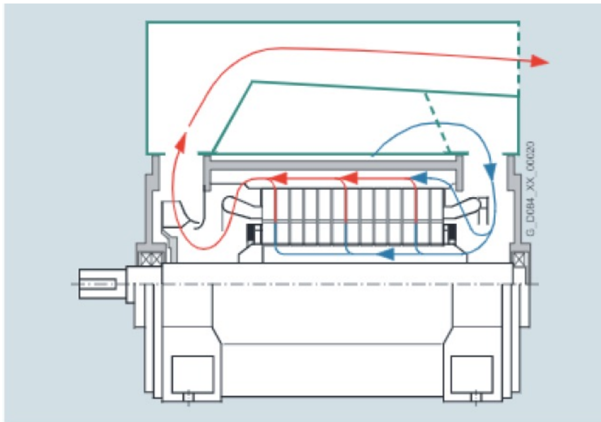


SISSA

Scuola Internazionale Superiore di Studi Avanzati

Pillar I focuses on the construction of DigitalTwins for the prototyping of complex manufactured objects:

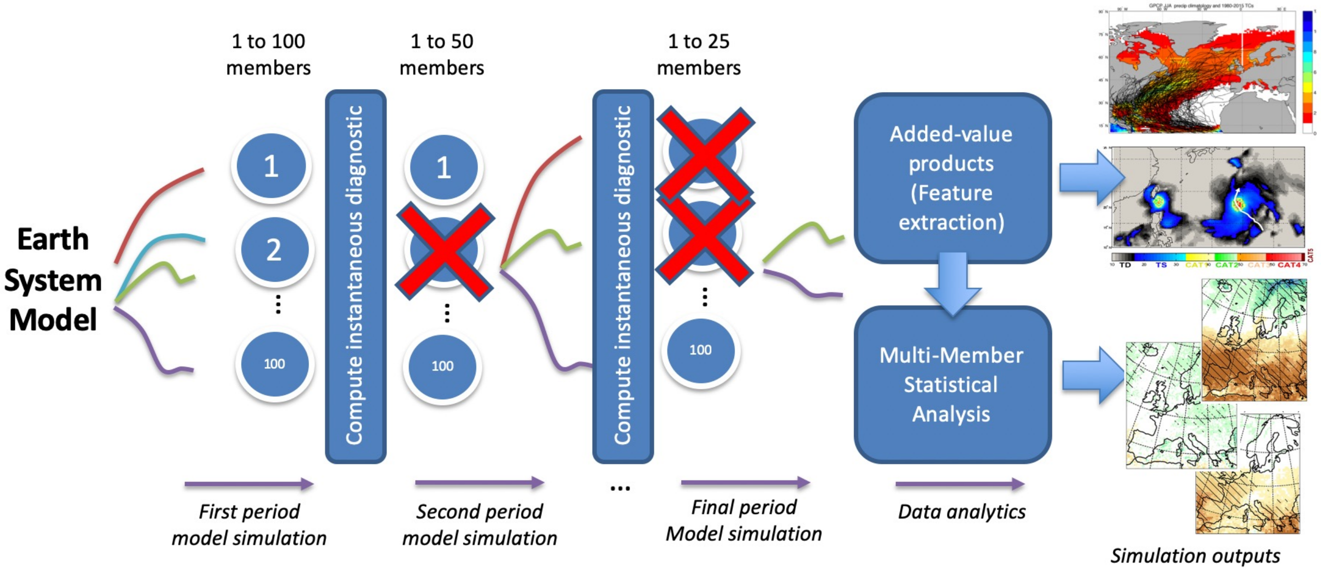
- Integrating state-of-the-art adaptive solvers with machine learning and data-mining
- Contributing to the Industry 4.0 vision

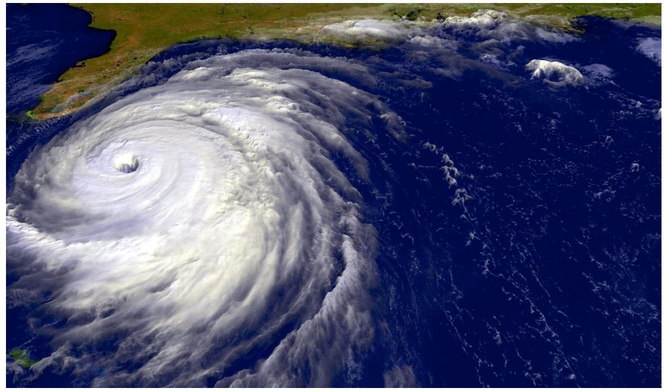


Pillar II: Climate

ESM Ensemble run over HPC

Data Analytics





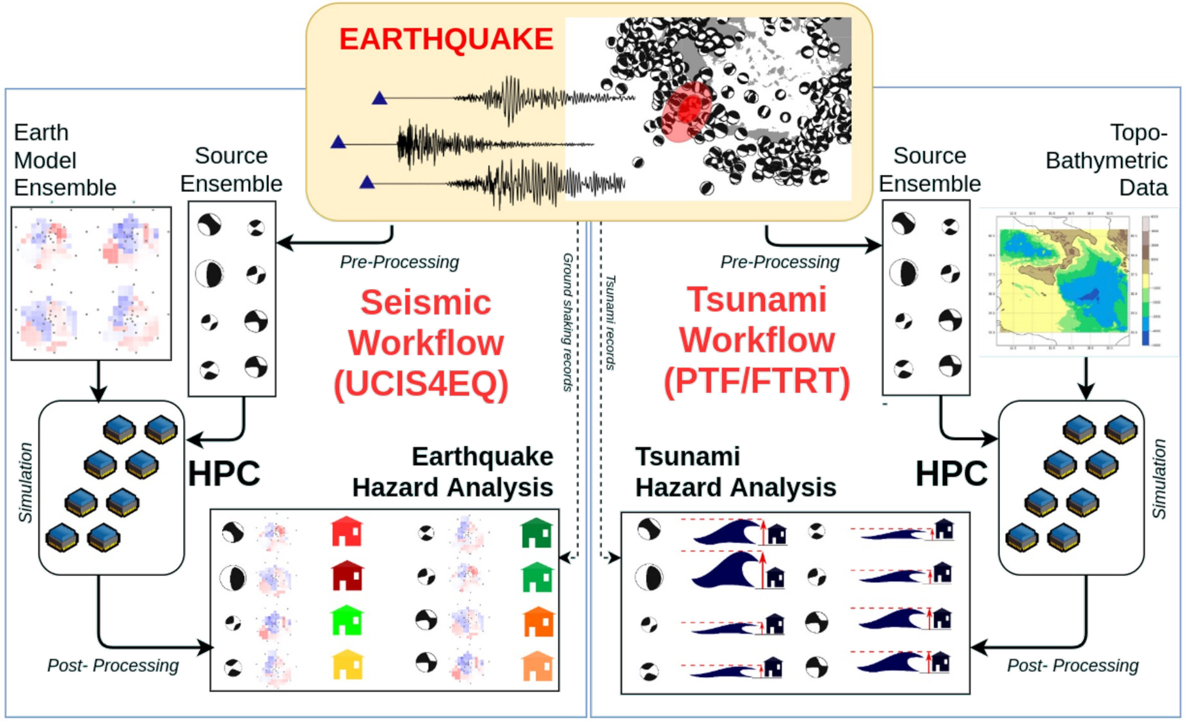
Dynamic (AI-assisted) workflow

HPDA & ML/DL

- **Perform climate predictions: temperature, precipitation or wind speed**
- **AI-assisted pruning of the ESM workflow**
- **Study of Tropical Cyclones (TC) in the North Pacific, with** in-situ analytics

Pillar III: Urgent computing for natural hazards

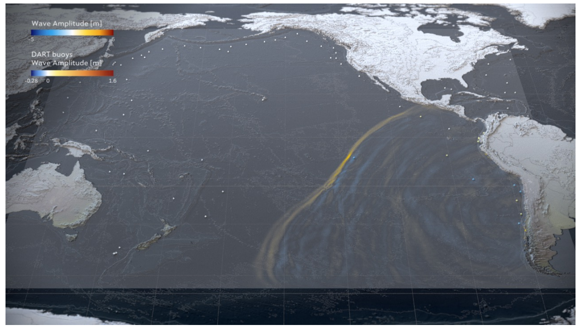
eFlows4HPC



Pillar III explores the modelling of natural catastrophes:

- Earthquakes and their associated tsunamis shortly after such an event is recorded
- **Use of AI to estimate intensity maps**
- **Use of DA and AI tools to enhance event diagnostics**
- **Areas: Mediterranean basin, Mexico, Iceland and Chile**

Tsunami-HySEA GPU-based code



BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

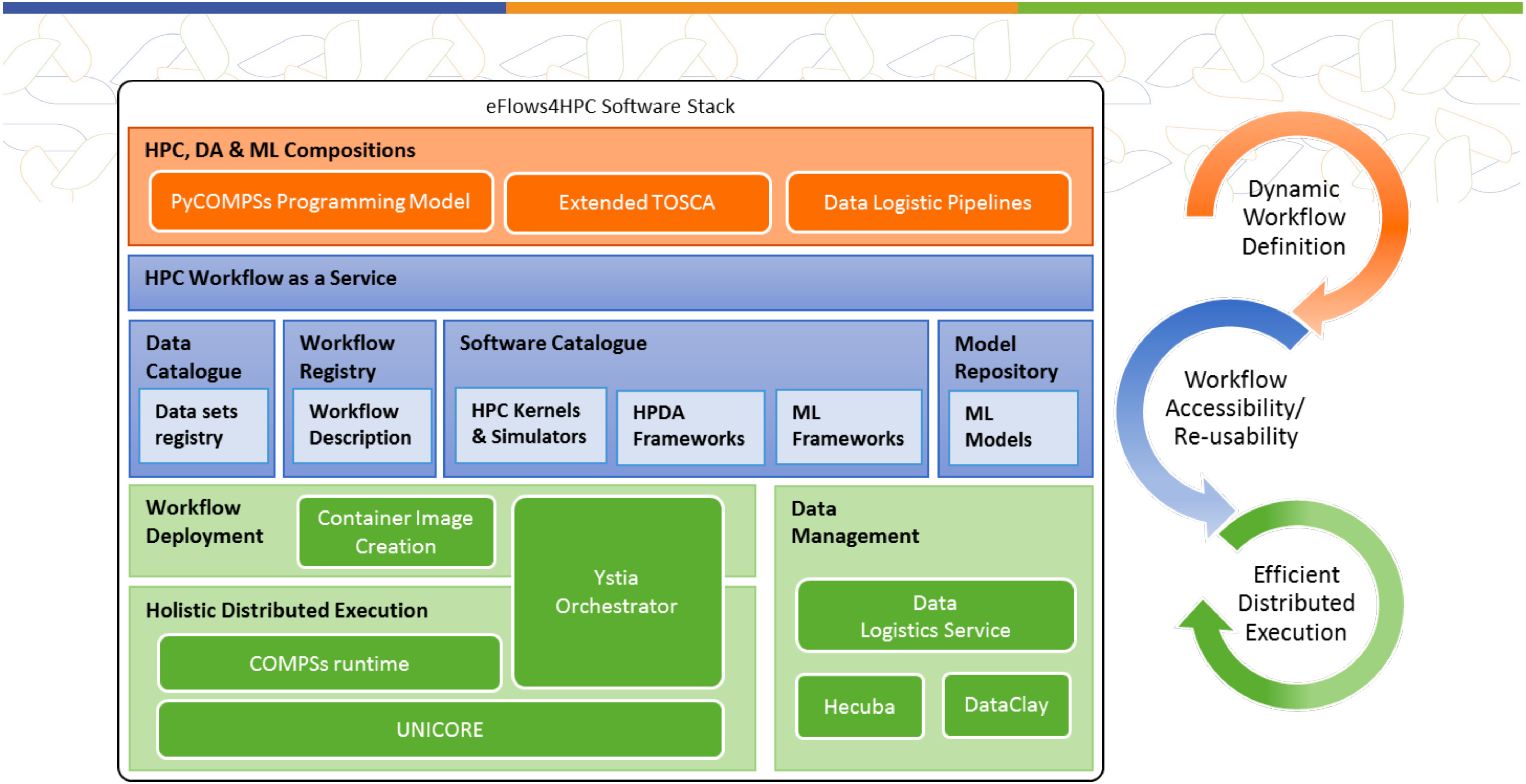
ETH zürich

ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

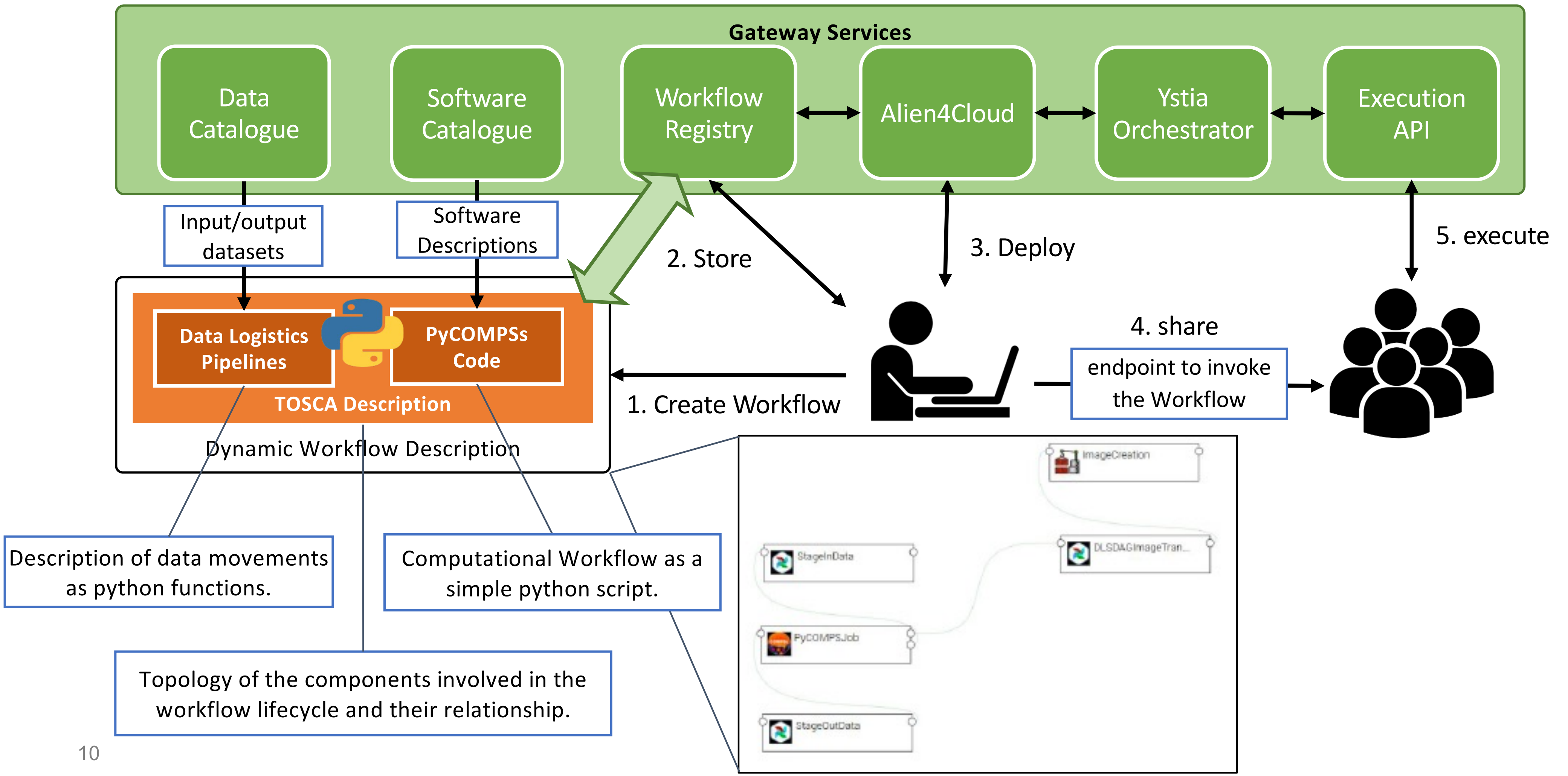
PSNC

UNIVERSIDAD DE MÁLAGA

NGI


eFlows4HPC Software Stack
HPC, DA & ML Compositions
PyCOMPSs Programming Model
Extended TOSCA
Data Logistic Pipelines
HPC Workflow as a Service
Data Catalogue
Data sets registry
Workflow Registry
Workflow Description
Software Catalogue
HPC Kernels & Simulators
HPDA Frameworks
ML Frameworks
Model Repository
ML Models
Workflow Deployment
Container Image Creation
Ystia Orchestrator
Holistic Distributed Execution
COMPSs runtime
UNICORE
Data Management
Data Logistics Service
Hecuba
DataClay
Dynamic Workflow Definition
Workflow Accessibility/ Re-usability
Efficient Distributed Execution

HPCWaaS Overview



Project partners

eFlows4HPC

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

CIMNE

JÜLICH Forschungszentrum

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Bull atos technologies

DtoK LAB

cmcc Centro Euro-Mediterraneo sui Cambiamenti Climatici

Inria

SISSA Scuola Internazionale Superiore di Studi Avanzati

INSTYTUT CHEMII BIOORGANICZNEJ Polskiej Akademii Nauk

UNIVERSIDAD DE MÁLAGA

AWI ALFRED-WEGENER-INSTITUT HELMHOLTZ-ZENTRUM FÜR POLAR- UND MEERESFORSCHUNG

ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

ETH zürich

SIEMENS

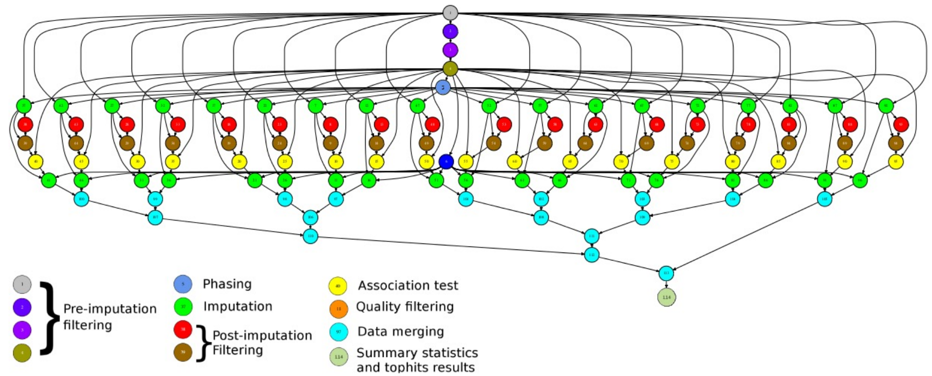
NGI

INTEGRATING DIFFERENT COMPUTATIONS IN PYCOMPSS

Main element: Workflows in PyCOMPSs

- **Sequential programming, parallel execution**
- **General purpose programming language + annotations/hints**
  - To identify tasks and directionality of data
- **Builds a task graph at runtime that express potential concurrency**
- **Tasks can be sequential and parallel (threaded or MPI)**
- **Offers to applications the illusion of a shared memory in distributed system**
  - The application can address larger data than storage space: support for Big Data apps
  - Support for persistent storage
- **Agnostic of computing platform**
  - Enabled by the runtime for clusters, clouds and container managed clusters

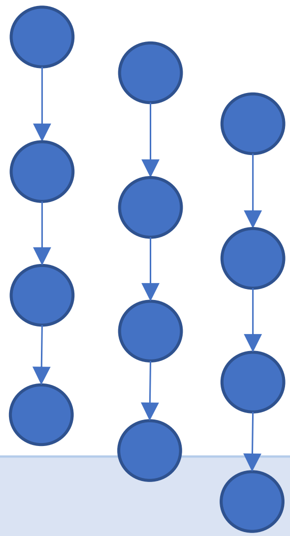
```
@task(c=INOUT)
def multiply(a, b, c):
    c += a*b
```



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

PyCOMPSs syntax

- Use of decorators to annotate tasks and to indicate arguments directionality
- Small API for data synchronization



Tasks definition

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

Main Program

```
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
```

Synchronization

eFlows4HPC

- **Main program and tasks do not share the same memory spaces**
- **The synchronization compss_wait_on waits for tasks generating the parameter to be finished and moves the data from the remote node to the node where the main program is executed:**

```
a = compute (b)
#compute is a task, here we can not check the value of a
...
a = compss_wait_on (a)
#here we can check the value of a
if a:
    ...
```

- **Tasks can be also synchronized with a barrier**

```
startMulTime = time.time()
for i in range(SIZE):
    compute (A[i], B[i])
compss_barrier()
mulTime = time.time() - startMulTime
```

Other decorators: Tasks’ constraints

eFlows4HPC

- **Constraints enable to define HW or SW features required to execute a task**
  - Runtime performs the match-making between the task and the computing nodes
  - Support for multi-core tasks and for tasks with memory constraints
  - **Support for heterogeneity on the devices in the platform**

```
@constraint (MemorySize=6.0, ProcessorPerformance=“5000”, ComputingUnits=“8”)
@task (c=INOUT)
def myfunc(a, b, c):
    ...
```

```
@constraint (MemorySize=1.0, ProcessorType =”ARM”, )
@task (c=INOUT)
def myfunc_other(a, b, c):
    ...
```

Failure Management

- Interface that enables the programmer to give hints about failure management

```
@task(file_path=FILE_INOUT, on_failure='CANCEL_SUCCESSORS’,
time_out=‘$task_timeout’)
def task(file_path):
    ...
    if cond :
        raise Exception()
```

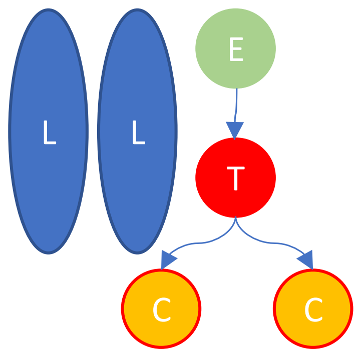
- Options: RETRY, CANCEL_SUCCESSORS, FAIL, IGNORE
- Implications on file management:
  - I.e., on IGNORE, output files: are generated empty
- **Possibility of ignoring part of the execution of the workflow, for example if a task fails in an unstable device**
- **Opens the possibility of dynamic workflow behaviour depending on the actual outcome of the tasks**

- **Tasks can raise exceptions**

```
@task(file_path=FILE_INOUT)
def comp_task(file_path):
    ...
    raise COMPSsException("Exception raised")
```

- **Combined with groups of tasks enables to cancel the group of tasks on the occurrence of an exception**

```
def test_cancellation(file_name):
    try:
        with TaskGroup('failedGroup’):
            long_task(file_name)
            long_task(file_name)
            executed_task(file_name)
            comp_task(file_name)
            cancelledTask(FILE_NAME);
            cancelledTask(FILE_NAME)

except COMPSsException:
        print("COMPSsException caught")
        write_two(file_name)
```



Other decorators: linking with other programming models

- **A task can be more than a sequential function**
  - A task in PyCOMPSs can be sequential, multicore or multi-node
  - External binary invocation: wrapper function generated automatically
  - Supports for alternative programming models: MPI and OmpSs
- **Additional decorators:**
  - @binary(binary="app.bin")
  - @mpi(binary="mpiApp.bin", runner="mpirun", processes=8)
  - @ompss(binary="ompssApp.bin")
- **Can be combined with the @constraint and @implement decorators**

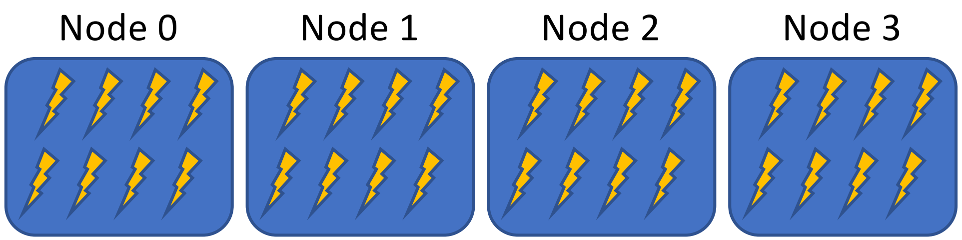
```
@binary(binary="app.bin", workingDir="/myApp")
@task()
def func(l):
    pass
```

Support for MPI tasks

- **Resource manager aware of multi-node tasks**

```
@mpi (runner="mpirun", processes= ”32”, processes_per_node=8)
@task (returns=int, stdOutFile=FILE_OUT_STDOUT, stdErrFile=FILE_OUT_STDERR)
def nems(stdOutFile, stdErrFile):
    pass
```

Launches MPI execution with
32 processes
8 processes per node



MPDM applications

- The @mpmd_mpi decorator can be used to define Multiple Program Multiple Data (MPMD) MPI tasks

```
@mpmd_mpi(runner="mpirun", working_dir = {{working_dir_exe}},
          programs=[{binary="fesom.x", processes = "$FESOM_PROCS" },
                    {binary="oifs", args="-v ecmwf -e awi3", processes = "$OIFS_PROCS" },
                    {binary="rnfma", processes = "$RNFMA_PROCS"}])
@task(log_file={Type:FILE_OUT, StdIOStream:STDOUT}, working_dir_exe=DIRECTORY_INOUT)
def esm_simulation(log_file, working_dir_exe):
   pass
```

- As a result of the @mpmd_mpi annotation, the following commands will be generated:

```
> cd working_dir_exe; mpirun -n $FESOM_PROCS fesom.x : \
      -n $OIFS_PROCS oifs -v ecmwf -e awi3 : -n $RNFMA_PROCS rnfma
```

Tasks in container images

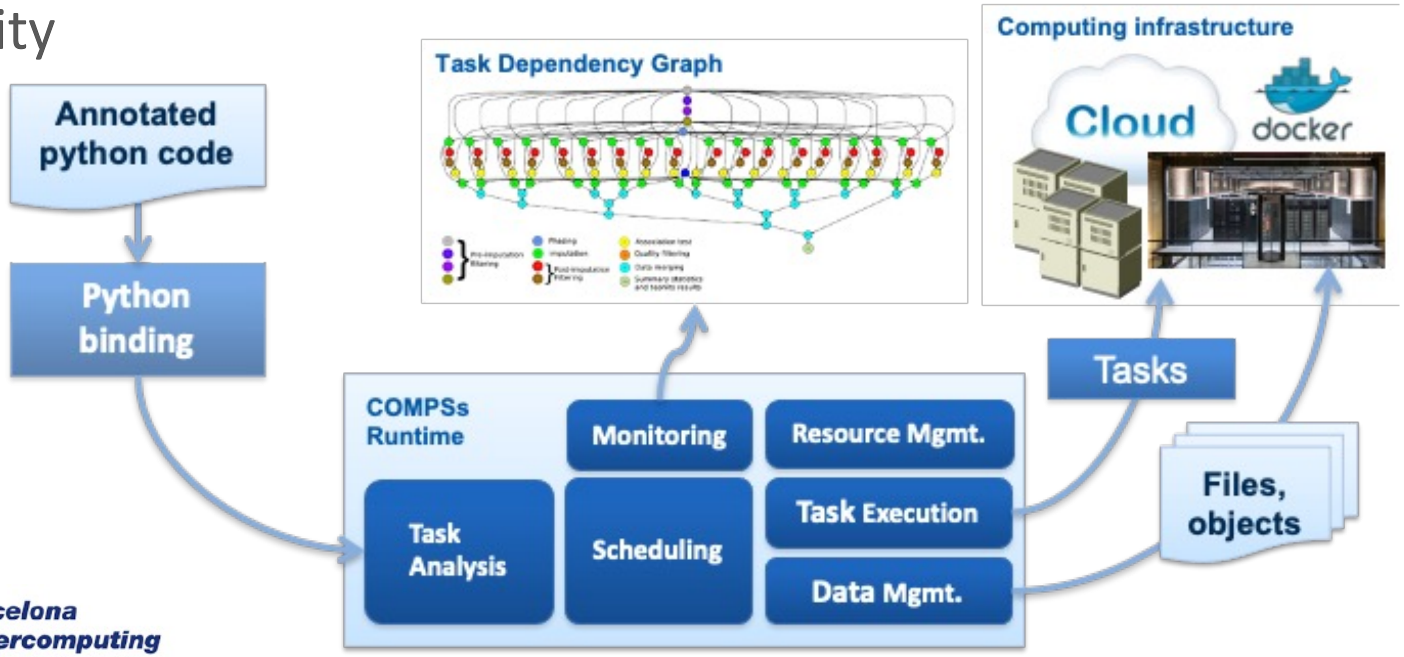
- **Goal: enable tasks embedded in container images**
  - @container decorator can be used together with the task annotation
  - Also support for user-defined tasks

```
@container(engine="DOCKER", image="ubuntu")
@binary(binary="ls")
@task()
def task_binary_empty():
    pass
```

```
@container(engine="DOCKER", image="compss/compss")
@task(returns=1, num=IN, in_str=IN, fin=FILE_IN)
def task_python_return_str(num, in_str, fin):
    print("Hello from Task Python RETURN")
    print("- Arg 1: num -- " + str(num))
    print("- Arg 1: str -- " + str(in_str))
    print("- Arg 1: fin -- " + str(fin))
    return "Hello"
```

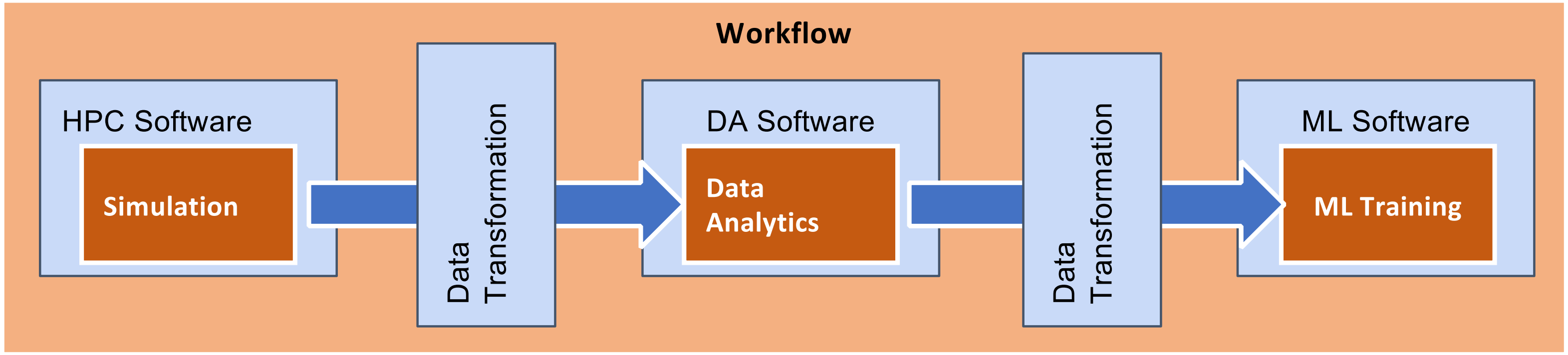
PyCOMPSs runtime

- Runtime deployed as a distributed master-worker
- All data scheduling decisions and data transfers are performed by the runtime
- Support for elasticity



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

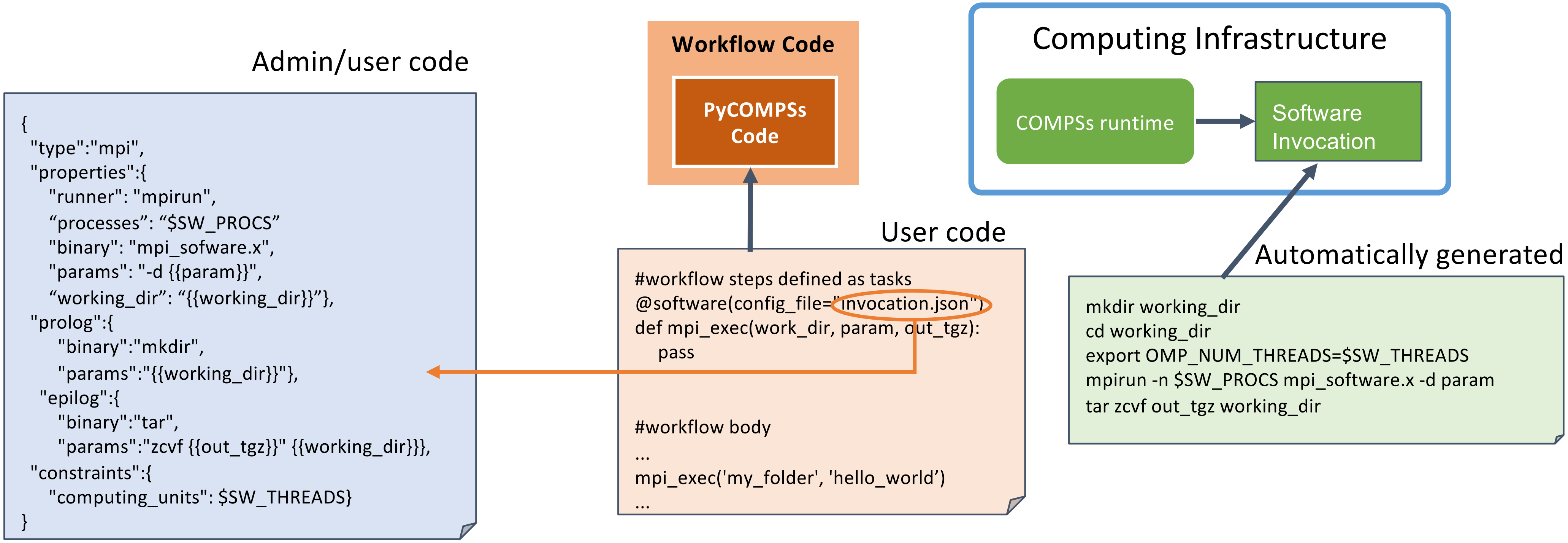
Interfaces to integrate HPC/DA/ML



- **Goal:**
  - Reduce the required glue code to invoke multiple complex software steps
  - Developer can focus in the functionality, not in the integration
  - Enables reusability
- **Two paradigms:**
  - Software invocation
  - Data transformations

```
#workflow steps defined as tasks
@data_transformation (input_data, transformation description)
@software (invocation description)
def data_analytics (input_data, result):
    pass

#workflow body
simulation (input_cfg, sim_out)
data_analytics (sim_out, analysis_result)
ml_training (analysis_result, ml_model)
```

Software Invocation description



Software invocation description
Stored in software catalog

- Converts a Python function of a software invocation to a PyCOMPSs task
- Takes information from the description in json
- Enables reuse in multiple workflows

Data transformations

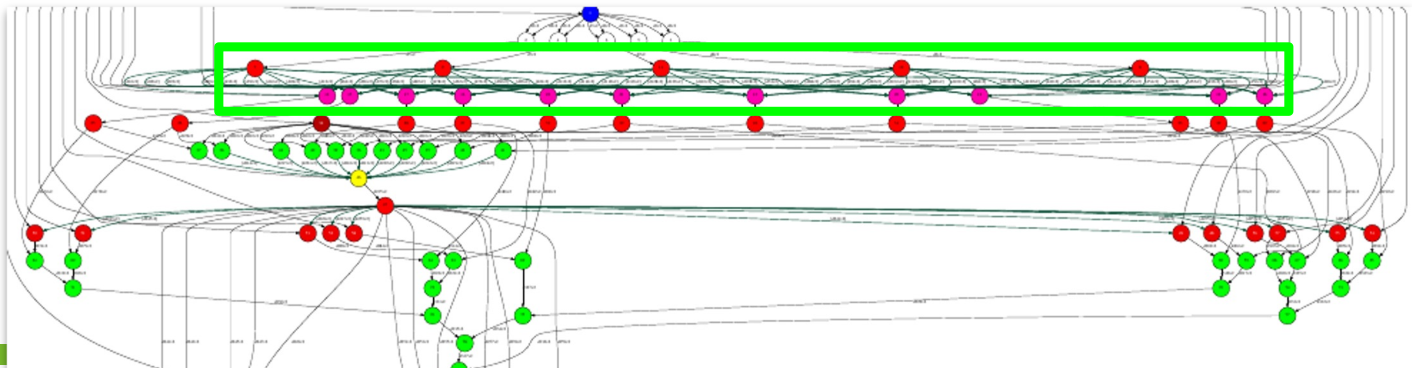
- **A data transformation changes the data without requiring extra programming from the developer**

Admin/user code

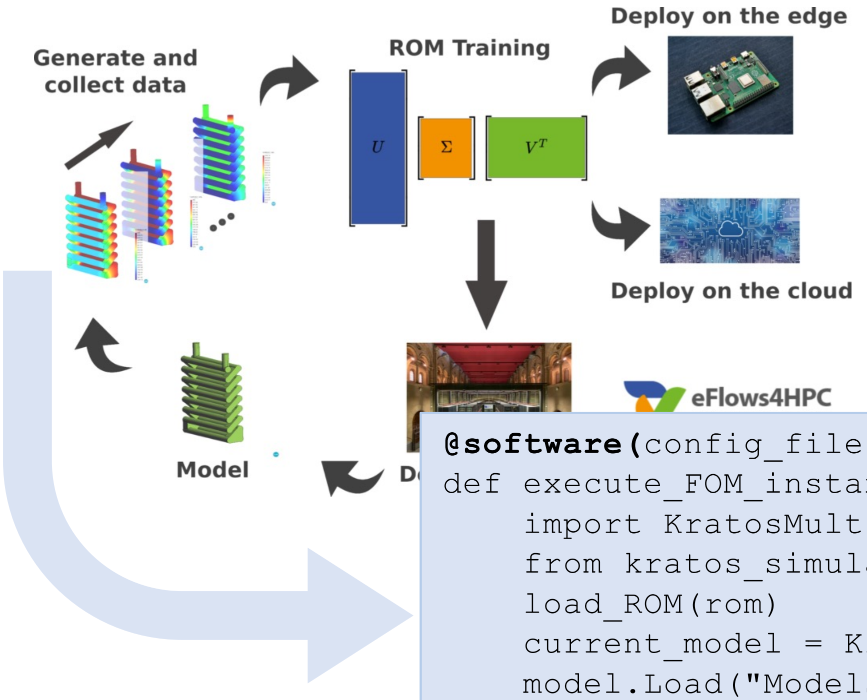
```
def load_blocks_rechunk(blocks, shape, block_size, new_block_size):
    ...
    SnapshotMatrix = load_blocks_array (final_blocks, shape, block_size);
    return SnapshotMatrix
```

```
@dt("blocks", load_blocks_rechunk, shape=expected_shape, block_size=simulation_block_size,
  new_block_size=desired_block_size, is_workflow=True)
@software(config_file = SW_CATALOG + "/dislib/dislib.json")
def rSVD(blocks, desired_rank=30):
    u,s = rsvd(blocks, desired_rank, A_row_chunk_size, A_column_chunk_size)
    return u
```

User code



Pillar I: Integration of HPC and data ana workflow

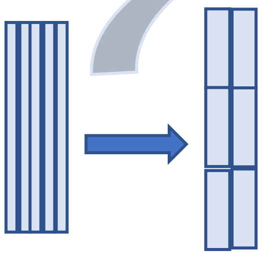


fom.json

```
{
    "execution" : {
        "type":"task"
    },
    "constraints" : {
        "computing_units": "$KRATOS_CUS"
    },
    "parameters" : {
        "returns" : 1,
        "model" : "IN",
        "parameters" : "IN",
        "sample" : "IN"
    }
}
```

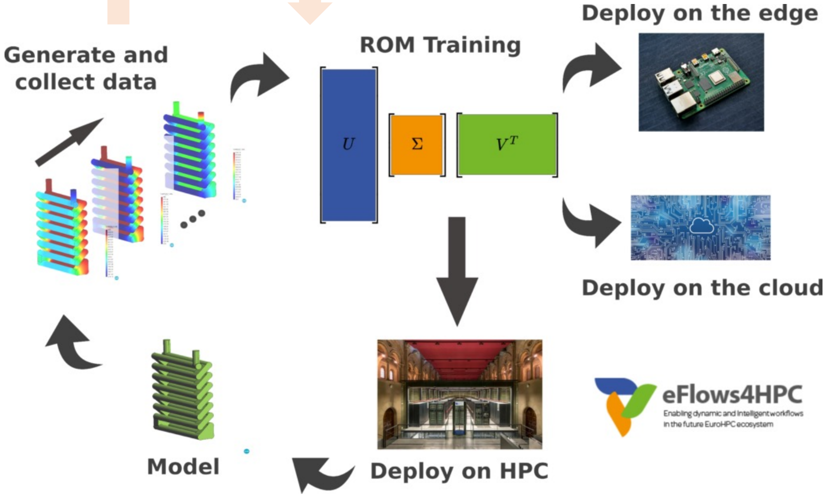
```
@software(config_file = SW_CATALOG+"/kratos/fom.json")
def execute_FOM_instance(model,parameters, sample):
    import KratosMultiphysics
    from kratos_simulations import RunROM_SavingData
    load_ROM(rom)
    current_model = KratosMultiphysics.Model()
    model.Load("ModelSerialization",current_model)
    del(model)
    current_parameters = KratosMultiphysics.Parameters()
    parameters.Load("ParametersSerialization",current_parameters)
    del(parameters)
    simulation = RunROM_SavingData(current_model,current_parameters,sample)
    simulation.Run()
    return simulation.GetSnapshotsMatrix()
```

Pillar I: Integration of HPC and data analytics in a single workflow

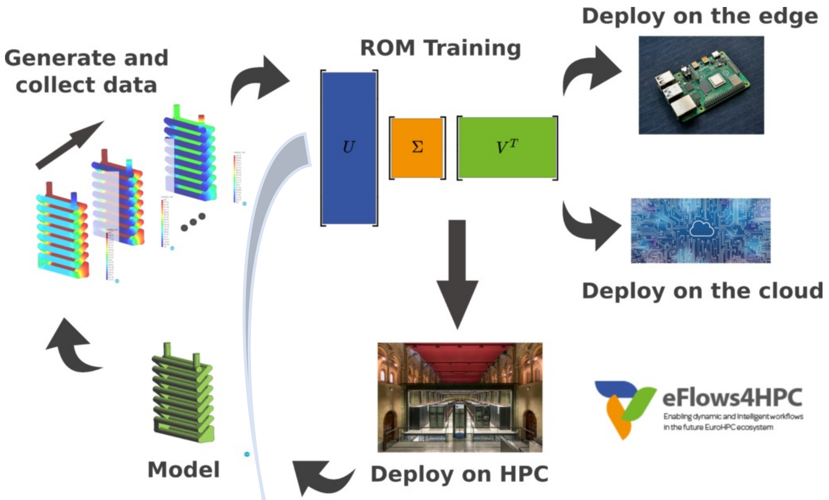


```
def load_blocks_rechunk(blocks, shape, block_size, new_block_size):
    ....
    arr = load_blocks_array(final_blocks, shape, block_size)
    return arr.rechunk(new_block_size)
```

Dislib method that re-partitions a set of blocks into a new ds-array of the given block size



Pillar I: Integration of HPC and data analytics in a single workflow



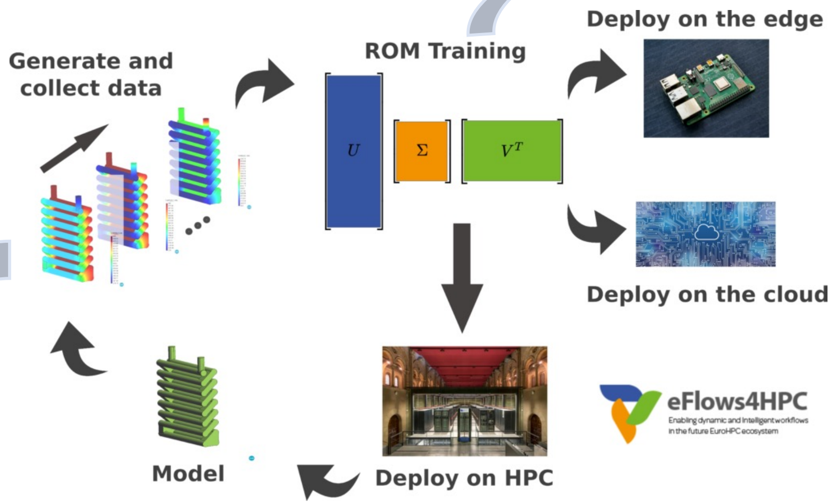
dislib.json

```
{
        "execution": {
                "type":"workflow"
        }
}
```

Method that invokes other PyCOMPSs tasks inside

```
@dt("blocks", load_blocks_rechunk, shape=expected_shape,
block_size=simulation_block_size, new_block_size=desired_block_size,
is_workflow=True)
@software(config_file = SW_CATALOG + "/py-dislib/dislib.json")
def rSVD(blocks, desired_rank=30):
    from dislib_randomized_svd import rsvd
    u,s = rsvd(blocks, desired_rank, A_row_chunk_size, A_column_chunk_size)
    return
```

Pillar I: Integration of HPC and data analytics in a single workflow



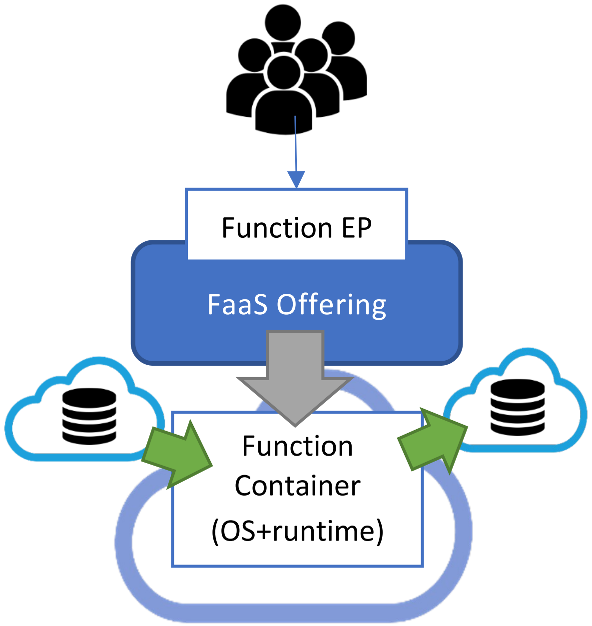

```
import dislib as ds

@dt("blocks", load_blocks_rechunk, shape=expected_shape,
block_size=simulation_block_size,new_block_size=desired_
is_workflow=True)
@software(config_file = SW_CATALOG + "/dislib/dislib.jso
def rsvd(blocks, desired_rank):
     k = desired_rank
     ...
     Y = A @ Omega
     Q,R = my_qr(Y._blocks)
     Q=load_blocks_rechunk([Q], ...)
     ...
```

```
@software(config_file=kratos.json)
def ExecuteInstance (model, parameters, Cases, instance):
    ...
    current_paramete
    ...
    simulation = Get
    simulation.Run()
    return simulatio
```

```
....
for instance in range (0,TotalNumberOFCases):
    blocks.append(ExecuteInstance(model, parameters, pars, instance))
...
U, s = rSVD(A, desired_rank)
...
```

HPC READY CONTAINER IMAGES

FaaS vs HPCWaaS

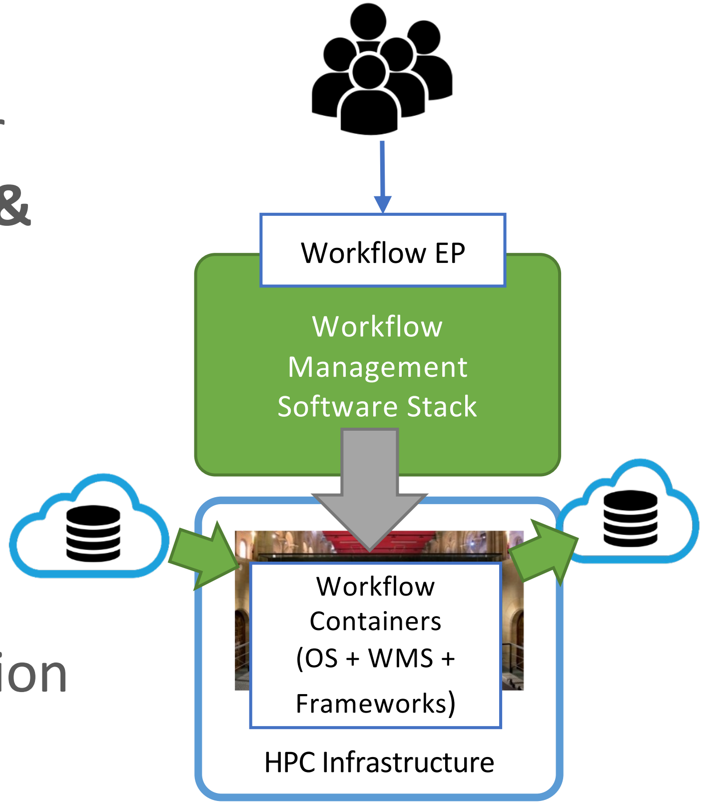


Similarities

- Easy to use for final user
- **Automate deployment & execution**
- Data integration
- **Containers**

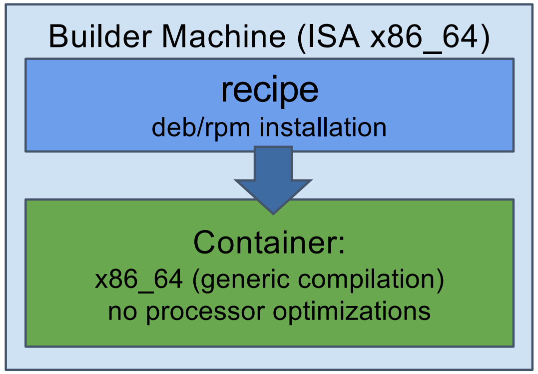
Differences

- Restrictions
- Deployment and Execution Complexity
- **Performance**



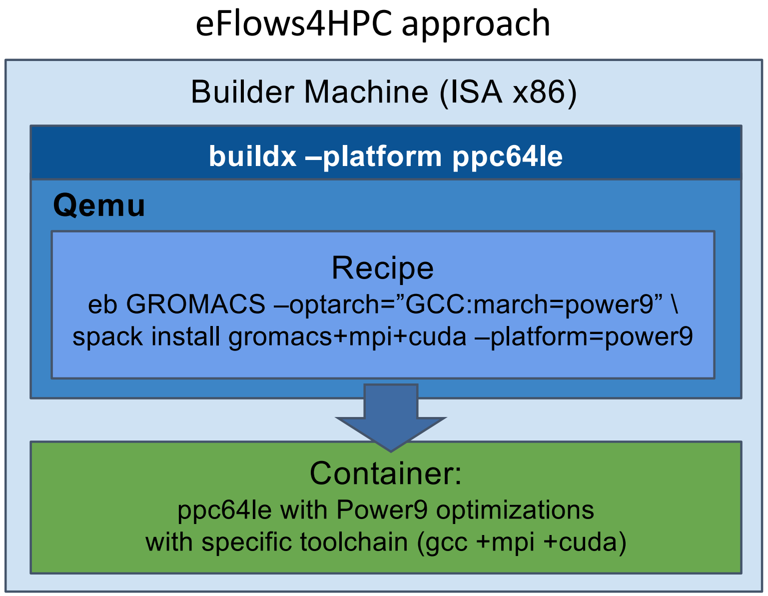
Containers and HPC

Standard container image creation



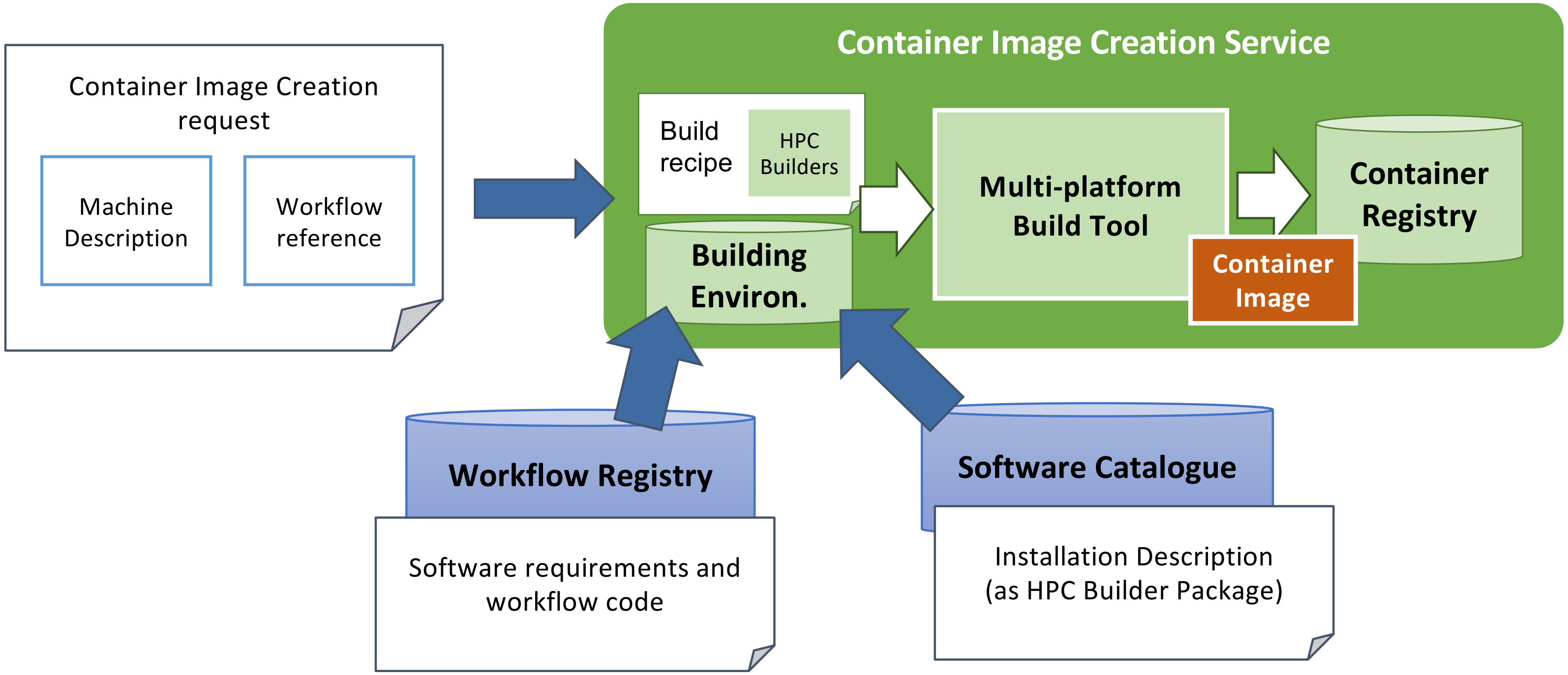
- **Simplicity for deployment**
  - Just pull or download the image
- **Trade-Off performance/portability**
  - Architecture Optimizations
- **Accessing Hardware from Containers**
  - MPI Fabric /GPUs
- **Host-Container Version Compatibility**

HPC Ready Containers



- **Methodology to allow the creation containers for specific HPC system**
  - Leverage HPC and Multi-platform container builders
- **It is tight to do by hand but let's automate!**

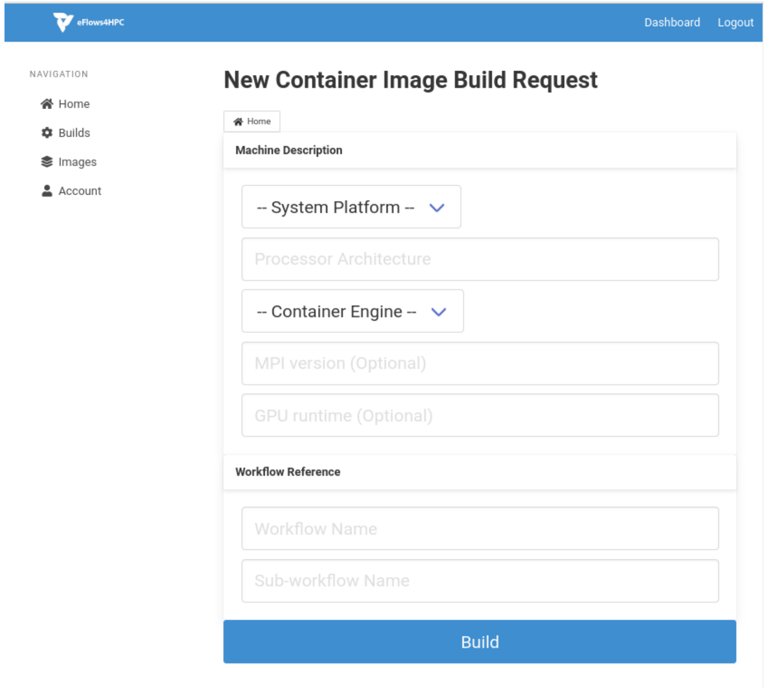
Container Image Creation Service



Container Image Creation Service

eFlows4HPC

- Web Interface



- REST Interface and CLI

```
POST /build/
```

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

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

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

```
localhost:~/image_creation> ./cic_cli <user> <token> https://<image_creation_url> build <request.json>
Response:
{"id":"f1f4699b-9048-4ecc-aff3-1c689b855adc"}
```

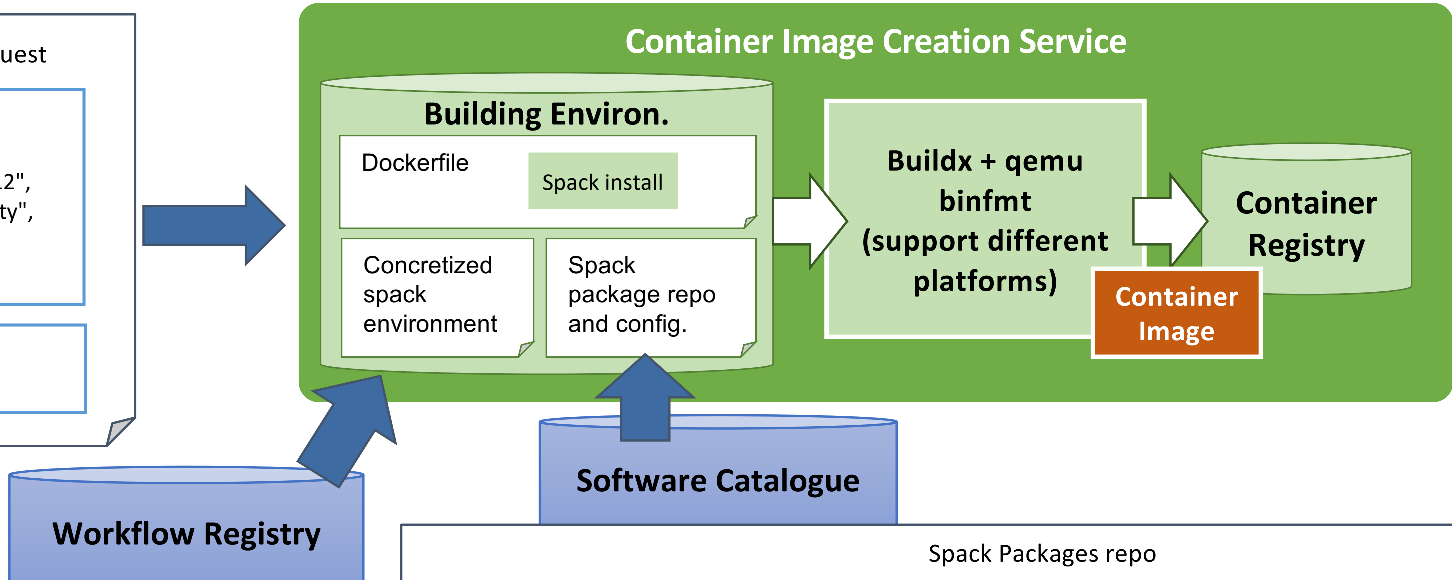
Example: Pillar I for MN4

eFlows4HPC

Container Image Creation request

```
"machine": {
    "platform": "linux/amd64",
    "architecture": "skylake_avx512",
    "container_engine": "singularity",
    "mpi" : intel-mpi@2018.4
    },
```

```
"workflow": "rom_pillar_I",
"step_id": "reduce_order_model"
```



Generic Spack environment

```
spack:
    specs:
        - compss
        - py-dislib
        - kratos@9.1.4 apps=LinearSolversApplication, \
                           FluidDynamicsApplication, \
                           StructuralMechanicsApplication, \
                           ConvectionDiffusionApplication
```

Spack Packages repo

```
class Compss(Package):
    url = ..
    version('2.10', sha256='...', preferred=True)
    ....
    depends_on('python')
    depends_on('openjdk')
    ...
    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 + '

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

```
class Kratos(CMakePackage):
    ...
    variant('mpi', default=False, description='..')
    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):
    ...
    depends_on('python@3:', type=('build', 'run'))
    depends_on('py-scikit-learn@0.23^pyscipy@1.5,type=('run'))
```

Spack Environment Concretization

eFlows4HPC

```
"machine": {
    "platform": "linux/amd64",
    "architecture": "skylake_avx512",
    "container_engine": "singularity",
    "mpi" : intel-mpi@2018.4
    },
```

```
spack:
    specs:
        - compss
        - py-dislib
        - kratos@9.1.4 apps=LinearSolversApplication, \
                             FluidDynamicsApplication, \
                             StructuralMechanicsApplication, \
                            ConvectionDiffusionApplication
```

```
spack:
    specs:
        - compss
        - py-dislib
        - kratos@9.1.4 apps=LinearSolversApplication, \
                             FluidDynamicsApplication, \
                             StructuralMechanicsApplication, \
                            ConvectionDiffusionApplication
        - intel-mpi@2018.4
    concretization: together
    view: /opt/view
    packages:
        all:
            target: [' skylake_avx512 ']
```

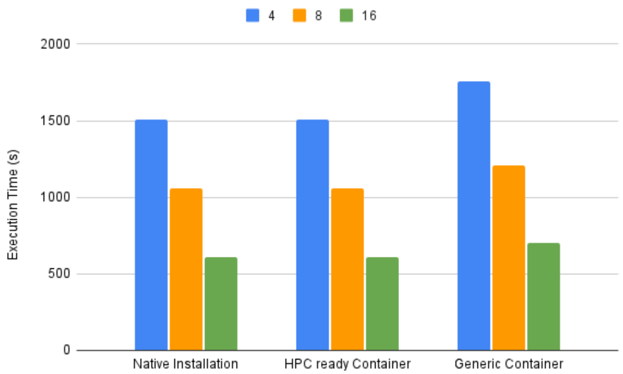
Performance

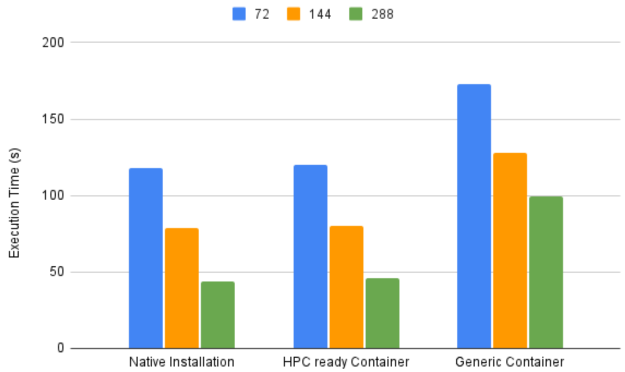
eFlows4HPC

Kratos Multiphysics (shared memory)

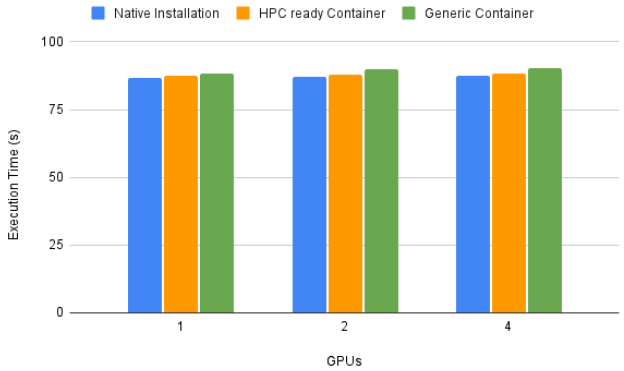
FESOM2 (MPI)

Nord3

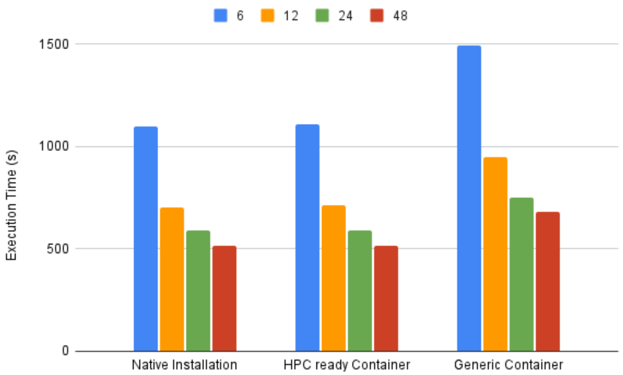


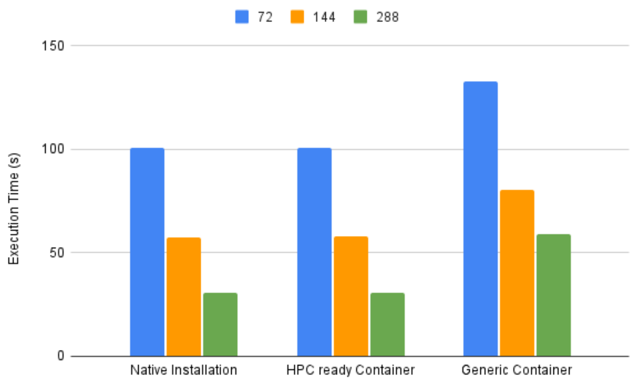


Tsunami-HySEA CTE-Power



MN4





Take away message

eFlows4HPC

- **HPC ready container images**
  - Advantages:
    - Reduce the deployment complexity
    - Performance close to bare metal
  - Drawbacks:
    - Larger building times compared to OS packages
      - Require more compilations
      - Configure spack binary caches
    - Less portable
      - CPU architecture, mpi, gpu versions

DATA PIPELINES AND TOP-LEVEL WORKFLOW IN TOSCA

Data pipelines: motivation

eFlows4HPC

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

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
- In eFlows4HPC, data pipelines implemented with Apache airflow

Data Catalogue and Data Logistics Service

eFlows4HPC

Data Catalogue:

- Lists datasets used and created by the workflow according to FAIR principles
- Provides metadata to make data movement pipelines more generic

Data Pipelines:

- Formalization of data movements for transparency and reusability
- Stage-in/out, image transfer

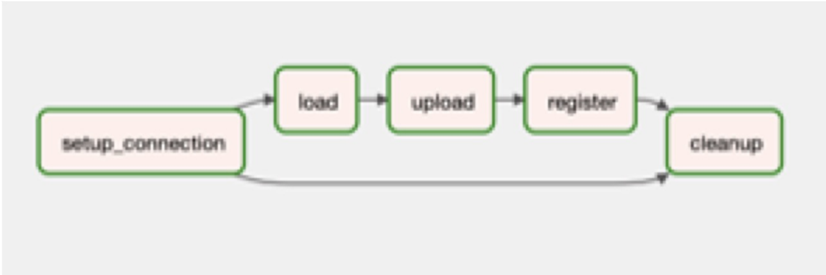
Data Logistics Services (DLS):

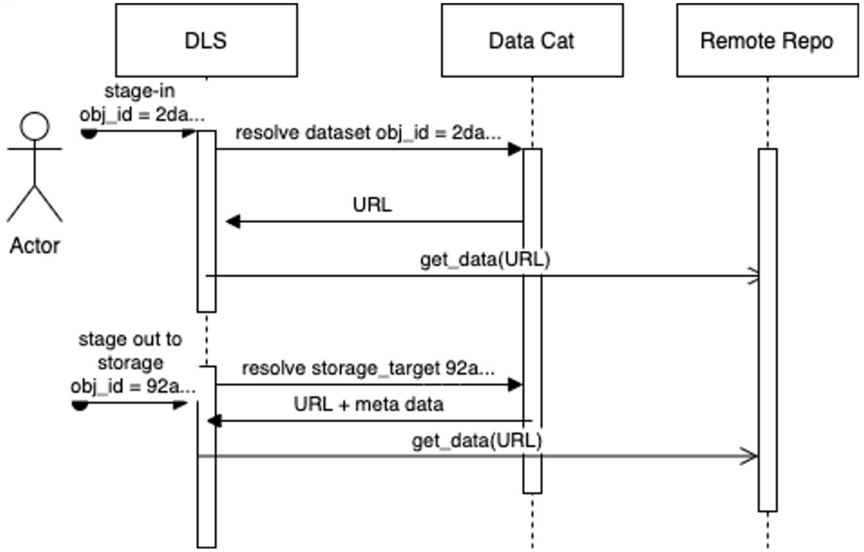
- Performs the execution of data pipelines at deployment and execution time

Production Ready Services:

- https://datacatalogue.eflows4hpc.eu
- https://datalogistics.eflows4hpc.eu/

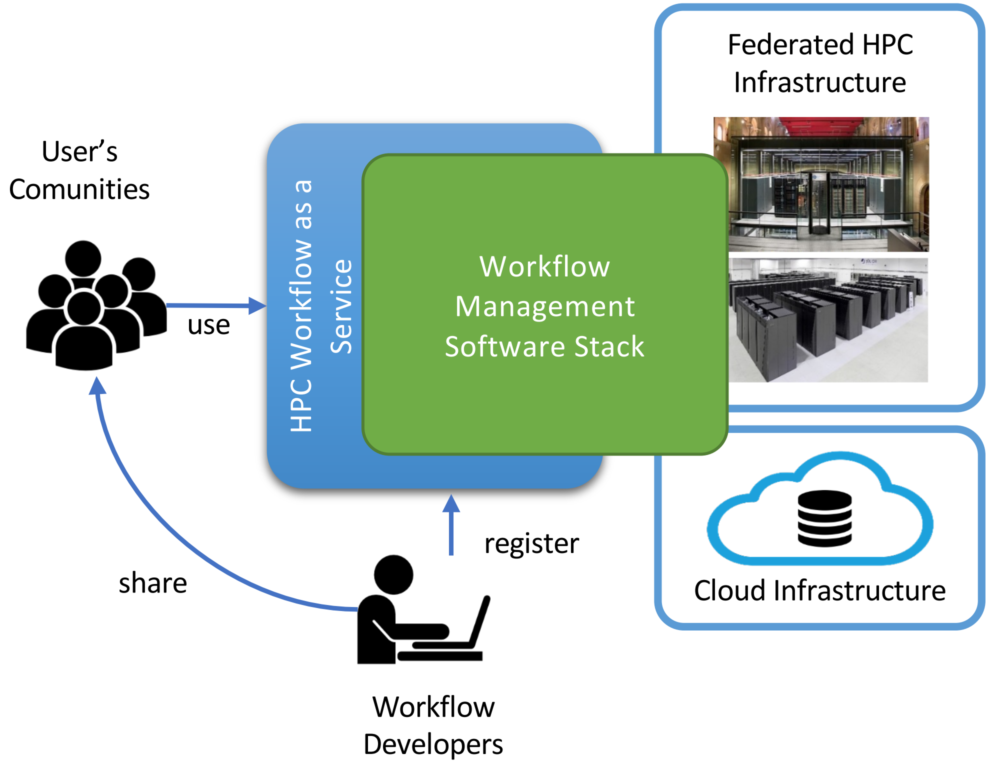
Data pipeline





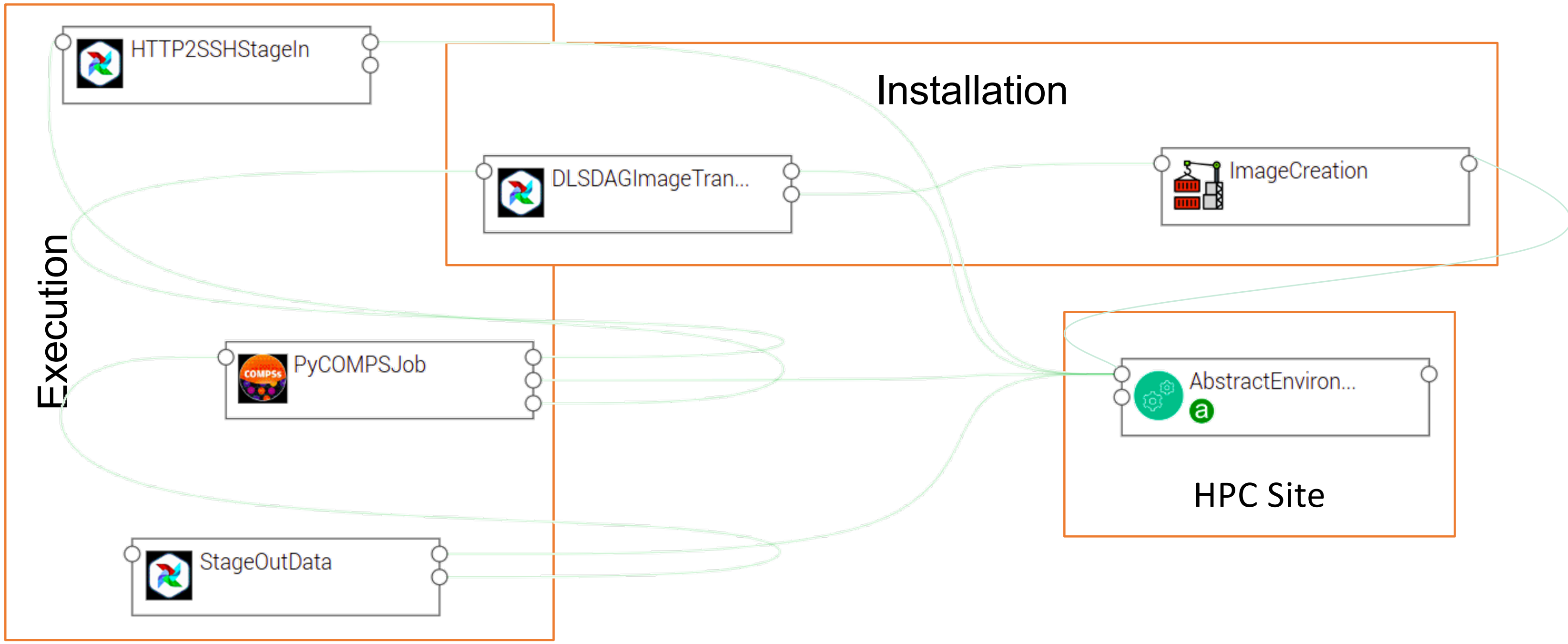
Top-level workflows approach

- **Requires a description for workflow lifecycle management**
  - TOSCA:
    - Model to describe cloud application topologies and its lifecycle orchestration
- **Interface for deploying and running the workflows**
  - HPCWaaS:
    - Development and Deployment (Alien4Cloud)
    - Execution (HPCWaaS API)

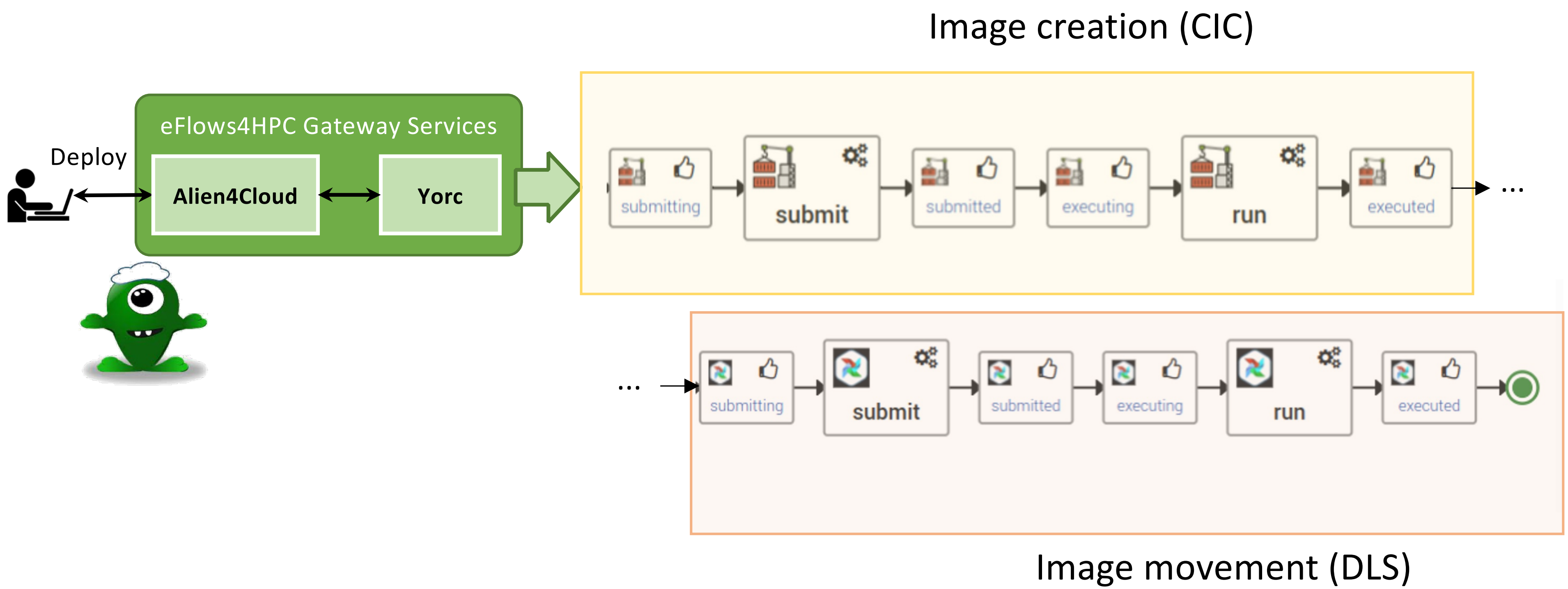


TOSCA Modelization

Topology of the different components involved in the Workflow lifecycle management

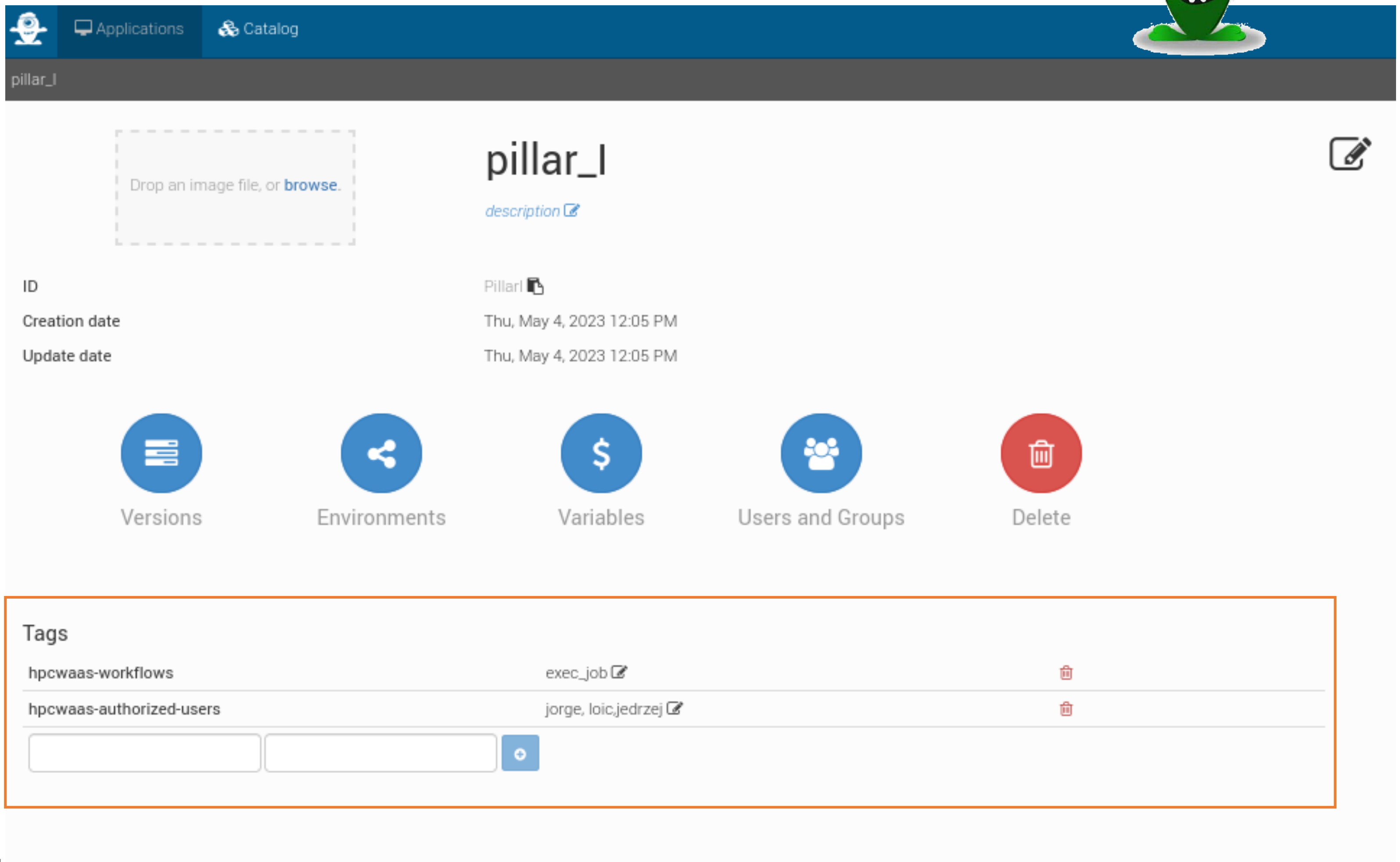


Application deployment (done once)

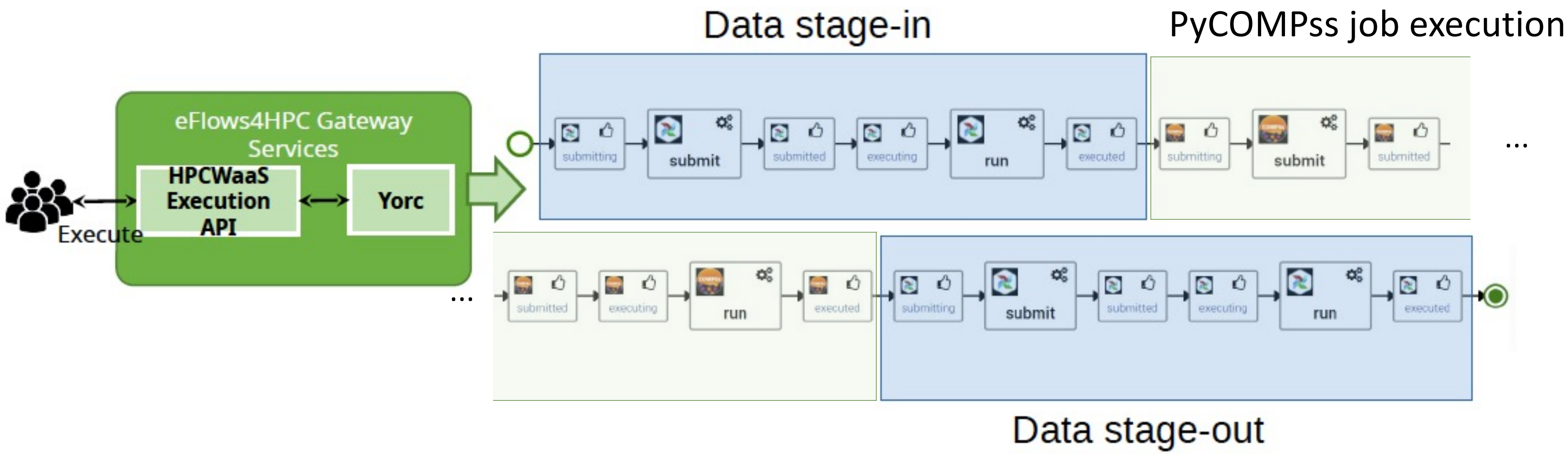


Workflow publication and users authorize

eFlows4HPC



End-user workflow execution (multiple executions)



Links

- Project website: eflows4hpc.eu
  - From software tab, access to github and documentation
- Project github: github.com/eflows4hpc
  - Specific repo for the ISC-HPC tutorial

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.