Developing complex workflows that integrate HPC, Artificial Intelligence and Data Analytics

Rosa M Badia – BSC

ACM Europe Summer School, Barcelona

31st August 2022

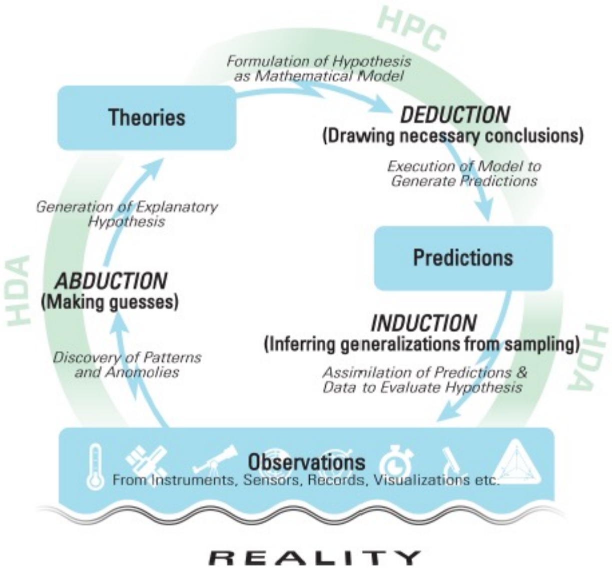
This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955558. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Germany, France, Italy, Poland, Switzerland, Norway.

Single inference cycle – scientific inquiry process

- Large amounts of data, coming from multiple sources
- Cycle of scientific inquiry process:
  1. Pre-processing steps for data curation and preparation (HDA)
  2. Computational steps (HPC)
  3. Analysis and analytics (HDA)
- Steps executed separately
- Fragmentation of workflows into separated components

HDA: High-end Data Analytics

HPC: High-Performance Computing

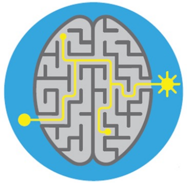


From "Big data and extreme-scale computing: Pathways to convergence-toward a shaping strategy for a future software and data ecosystem for scientific inquiry"

Computing infrastructure evolution

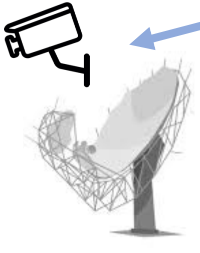
- Large HPC and Cloud Systems
  - Exascale computing providing extremely large computational power
  - With specialized architectures: GPUs, FPGAs
- Large data sources in instruments and sensors
- Edge devices providing parallel processing for data processes: to filter, reduce, compress data
- Artificial Intelligence Everywhere
  - HPC enabling Deep learning training
  - Edge devices performing inference... and training!

HPC and Cloud
Exascale computing
GPUs, FPGAs, EPI



AI

Edge devices



Sensors
Instruments
Actuators

Computing continuum

European context

- EuroHPC aims at developing a World Class Supercomputing Ecosystem in Europe
  - Procuring and deploying pre-exascale and petascale systems in Europe
- These systems will be capable of running large and complex applications
- Applications composing HPC, artificial intelligence and data analytics

EuroHPC systems

eFlows4HPC

First European Exascale Supercomputer announced to be installed in Jülich

Pre-Exascale

	Status	Country	Peak performance	Architecture
LUMI	Operational	Finland	552 petaflops	64-core AMD EPYC™ CPUs + AMD Instinct™ GPU
Leonardo	Under construction	Italy	322.6 petaflops	Intel Ice-Lake, Intel Sapphire Rapids + NVIDIA Ampere
MareNostrum 5	Contract signed	Spain	314 petaflops	…e Rapids, NVIDIA … Emerald Rapids, … and Intel Rialto

Petascale

	Status	Country	Peak performance	Architecture
Meluxi)na	O…		…flops	AMD EPYC + NVIDIA Ampere A100
Vega	O…tional	Slovenia	10.1 petaflops	AMD Epyc 7H12 + Nvidia A100
Karolina	Operational	Czech Republic	15.7 petaflops	AMD + Nvidia A100
Discoverer	Operational	Bulgaria	6 petaflops	AMD EPYC
Deucalion	Under construction	Portugal	10 petaflops	A64FX, AMD EPYC +Nvidia Ampere

https://eurohpc-ju.europa.eu/about/our-supercomputers_en

EuroHPC and its projects

- EuroHPC support for research and innovation activities
  - Call on Jan 2020: EuroHPC-02-2019: HPC and data-centric environments and application platforms
  - High Performance Computing (HPC) and data driven HPC software environments and application oriented platforms

ACROSS

MICROCARD

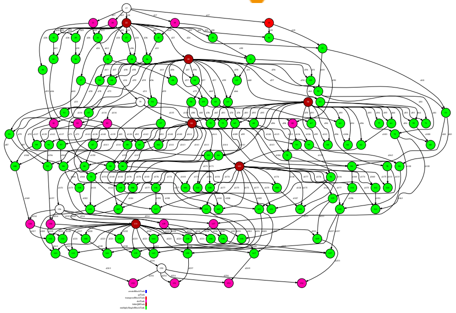
HEROES

REGALE
Open Architecture for Exascale Supercomputers

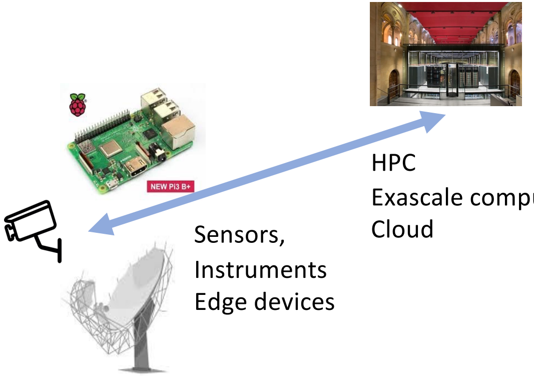
eFlows4HPC

PyCOMPSs/COMPSs ambition

eFlows4HPC



- Complex infrastructures with multiple and heterogeneous components
- Complex applications, composed of multiple components and pieces of software
- How to describe the workflows in such environment?
- Holistic approach where both data and computing are integrated in a single flow built on simple, high-level interfaces
  - Integration of computational workloads, with machine learning and data analytics
  - Intelligent runtime that can make scheduling and allocation, data-transfer, and other decisions



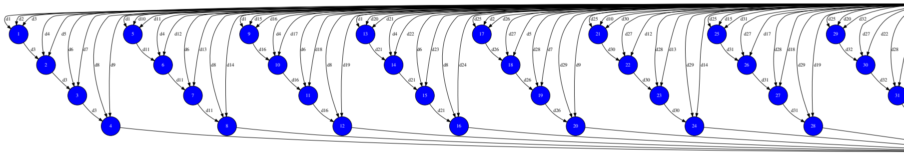
Talk overview

eFlows4HPC

- PyCOMPSs introduction
- eFlows4HPC project overview
- HPC Workflows as a Service

PYCOMPSS INTRODUCTION

Programming with PyCOMPSs/COMPSs



- 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
- Offers a shared memory illusion to applications in a distributed system
  - The application can address larger data storage space: support for Big Data apps
- 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
```

```
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
```

Some interesting features

- Task constraints: enable to define HW or SW requirements

```
@constraint (MemorySize=6.0, processorArchitecture="ARM")
@task (c=INOUT)
def myfunc(a, b, c):
    ...
```

- Linking with other programming models:

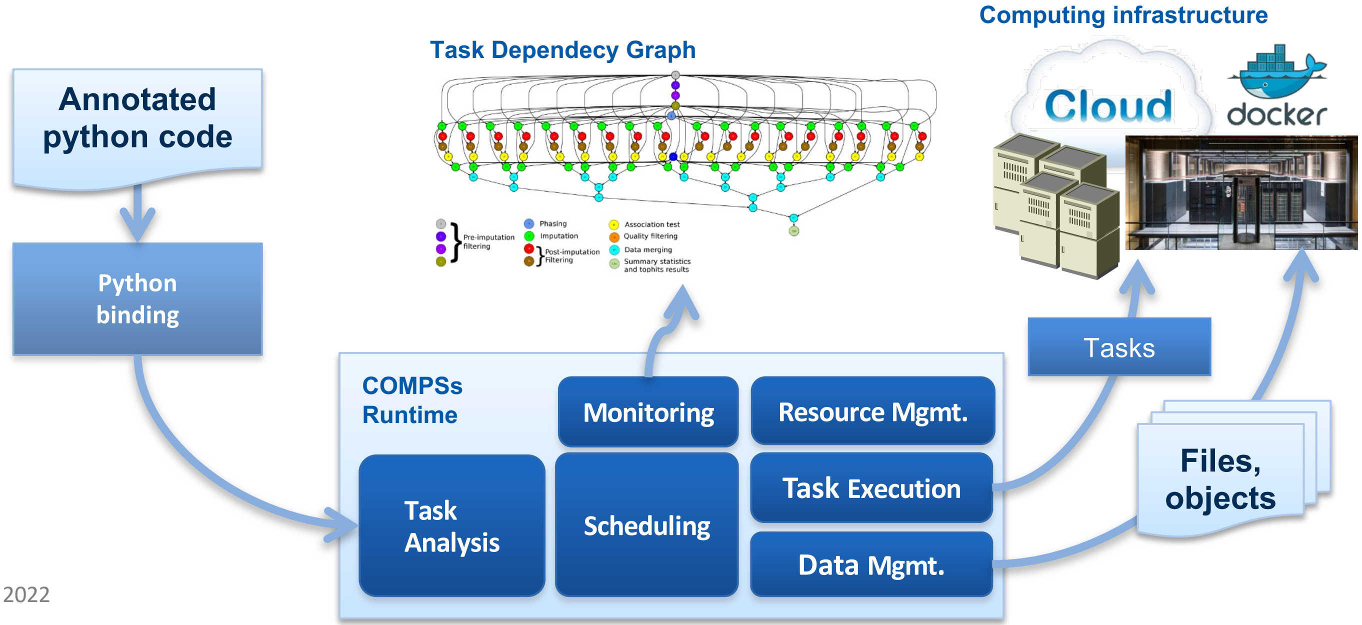
```
@mpi (runner="mpirun", processes=32,processes_per_node=8,...)
@task (returns=int, stdOutFile=FILE_OUT_STDOUT, ...)
def nems(stdOutFile, stdErrFile):
    pass
```

- Task failure management

```
@task(file_path=FILE_INOUT, on_failure='CANCEL_SUCCESSORS')
def task(file_path):
    ...
    if cond :
        raise Exception()
```

PyCOMPSs/COMPSs runtime

- PyCOMPSs/COMPSs applications executed in distributed mode following the master-worker paradigm
  - Description of computational infrastructure in an XML file
- Sequential execution starts in master node and tasks are offloaded to worker nodes
- All data scheduling decisions and data transfers are performed by the runtime



Support for http tasks

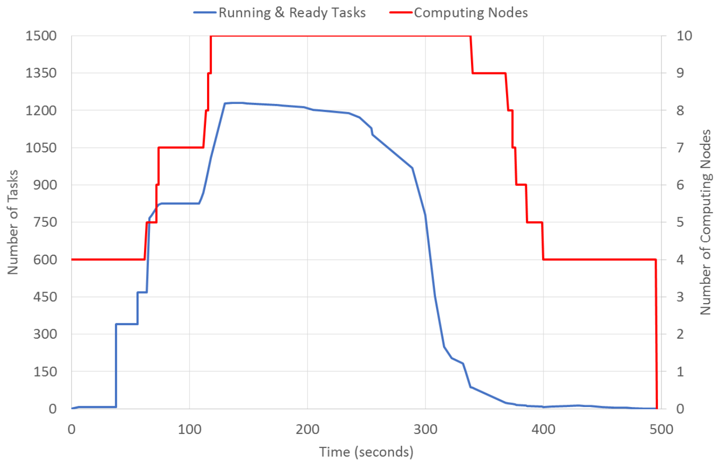
- Tasks can be executed on a remote Web Service via HTTP requests
- HTTP resource(s) need to be defined in the infrastructure definition (xml)
- At execution the runtime will search for the HTTP resource which allows the execution of the service and send a request to its base url
- Python parameters can be added to the request query
- This feature supports the execution of workflows that combine regular tasks and http tasks

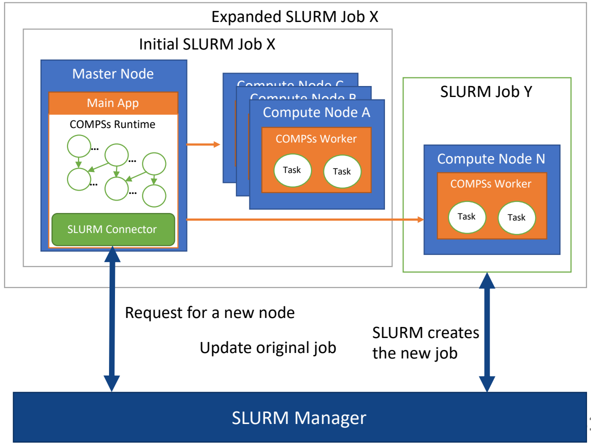
```
@http(service_name="service_1", request="GET",
      resource="get_length/{{message}}")
@task(returns=int)
def an_example(message):
    pass
```

```
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?>
<ResourcesList>
    <ComputeNode Name="localhost">
      ...
    </ComputeNode>

    <Http BaseUrl="http://remotehost:1992/test/"
        <ServiceName>service_1</ServiceName>
        <ServiceName>service_2</ServiceName>
    </Http>
       ...
</ResourcesList>
```

COMPSs runtime: support for elasticity

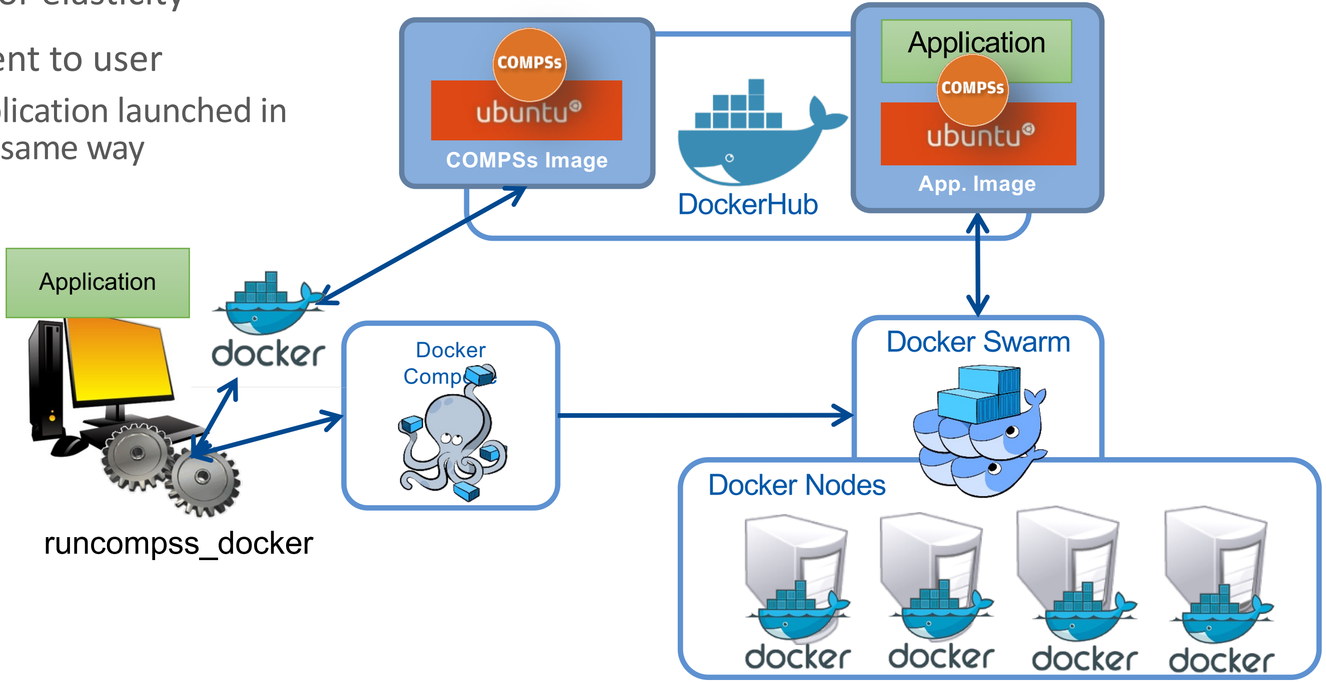
- Possibility to adapt the computing infrastructure depending on the actual workload
- Now also for SLURM managed systems
- Feature that contributes to a more effective use of resources





COMPSs runtime: containers

- Whole application deployed as a containers
  - Support for Docker, Singularity and other container engines
- Support for elasticity
- Transparent to user
  - Application launched in the same way



Tasks in container images

- Enablement of tasks embedded in container images
- Binaries and 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"
```

Support for data streams

- Interface to support streaming data in tasks
  - I.e: incoming video stream from a camera
- Task-flow and data-flow tasks live together in PyCOMPSs/COMPSs workflows
- Data-flow tasks persist while streams are not closed
  - Parameters can be one/multiple streams and non-streamed
- Runtime implementation based on Kafka

```
@task(fds=STREAM_OUT)
def sensor(fds):
        ...
    while not end():
        data = get_data_from_sensor()
        f.write(data)
    fds.close()
```

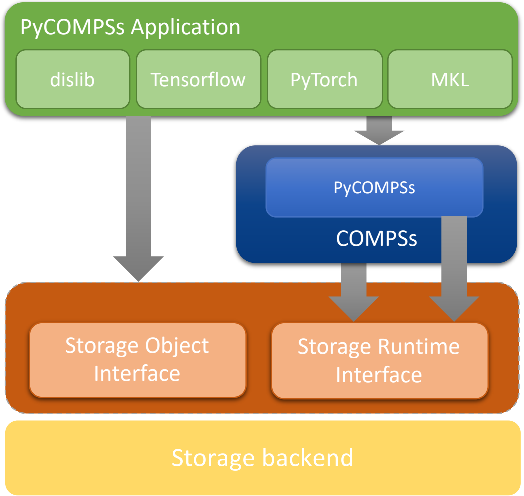
```
@task(fds_sensor=STREAM_IN, filtered=OUT)
def filter(fds_sensor, filtered):
        ...
    while not fds_sensor.is_closed():
        get_and_filter(fds_sensor, filtered)
```

Integration with persistent memory

- Programmer may decide to make persistent specific objects in its code
  - Can leverage new NVRAM or SDD disks
- Persistent objects are managed same way as regular objects
- Tasks can operate with them

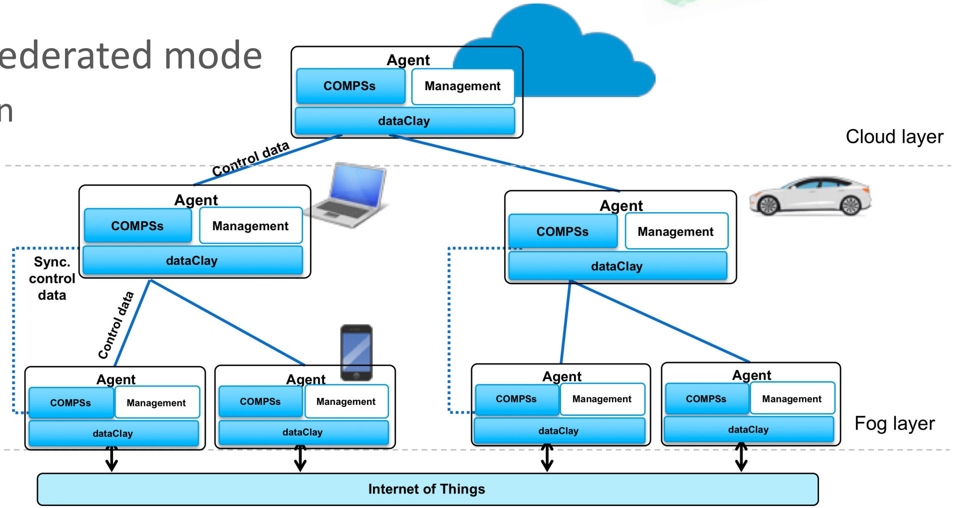
```
a = SampleClass ()
a.make_persistent()
Print a.func (3, 4)

a.mytask()
compss_barrier()

o = a.another_object
```

- Objects can be accessed/shared transparently in a distributed computing platform



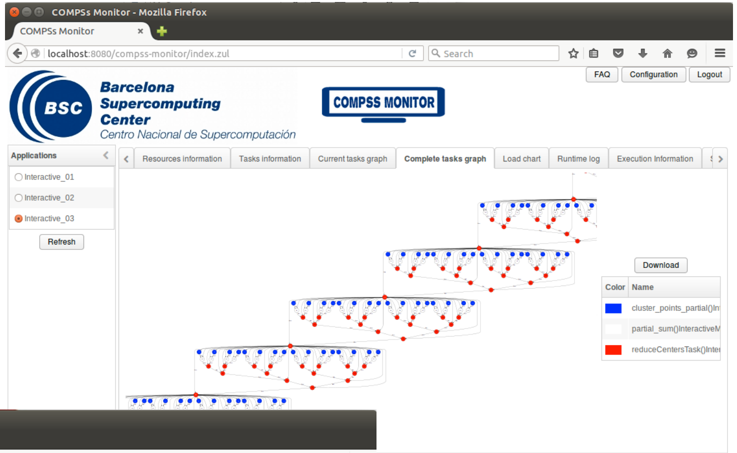
COMPSs in a fog-to-cloud architecture

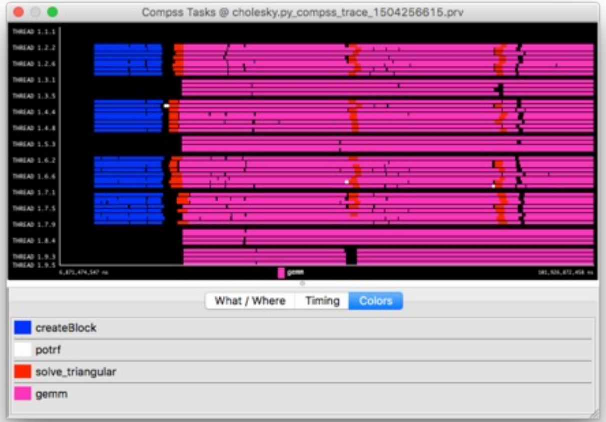
- **Decentralized** approach to deal with large amounts of data
- New COMPSs runtime handles distribution, parallelism and heterogeneity
- Runtime deployed as a microservice in an agent:
  - Agents are independent, can act as master or worker in an application execution, agents interact between them
  - Hierarchical structure
- Data managed by dataClay, in a federated mode
  - Support for data recovery when fog nodes disappear
- Fog-to-fog and Fog-to-cloud

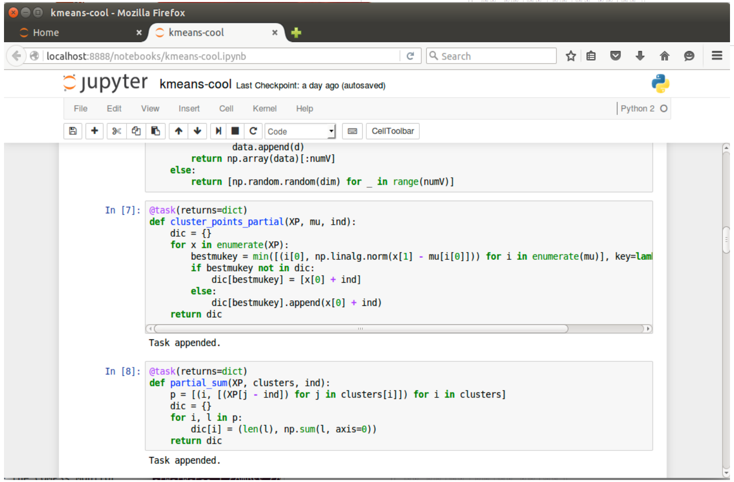


PyCOMPSs development environment

- Runtime monitor
- Paraver traces
- Jupyter-notebooks integration

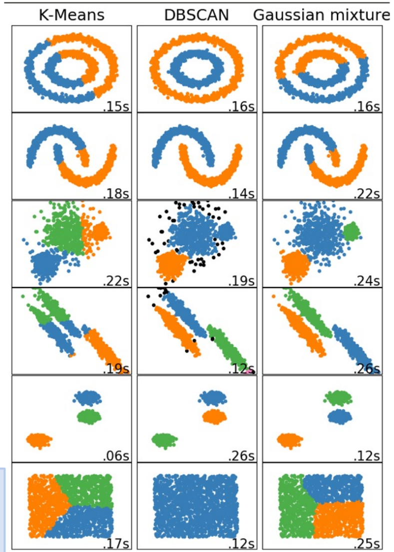






Dislib: parallel machine learning

- dislib: Collection of machine learning algorithms developed on top of PyCOMPSs
  - Unified interface, inspired in scikit-learn (fit-predict)
  - Based on a distributed data structure (ds-array)
  - Unified data acquisition methods
  - Parallelism transparent to the user – PyCOMPSs parallelism hidden
  - Open source, available to the community
- Provides multiple methods:
  - Data initialization
  - Clustering
  - Classification
  - Model selection, …

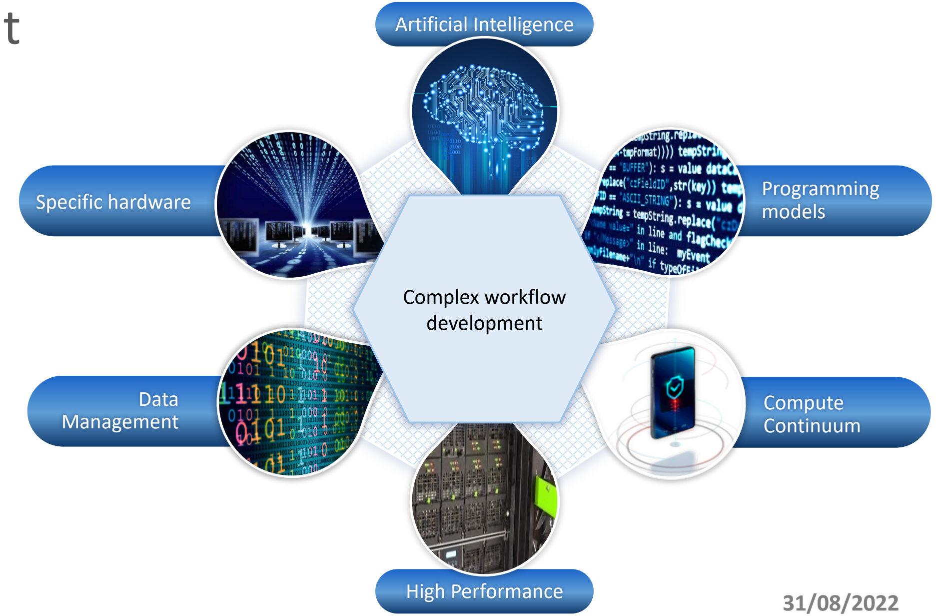


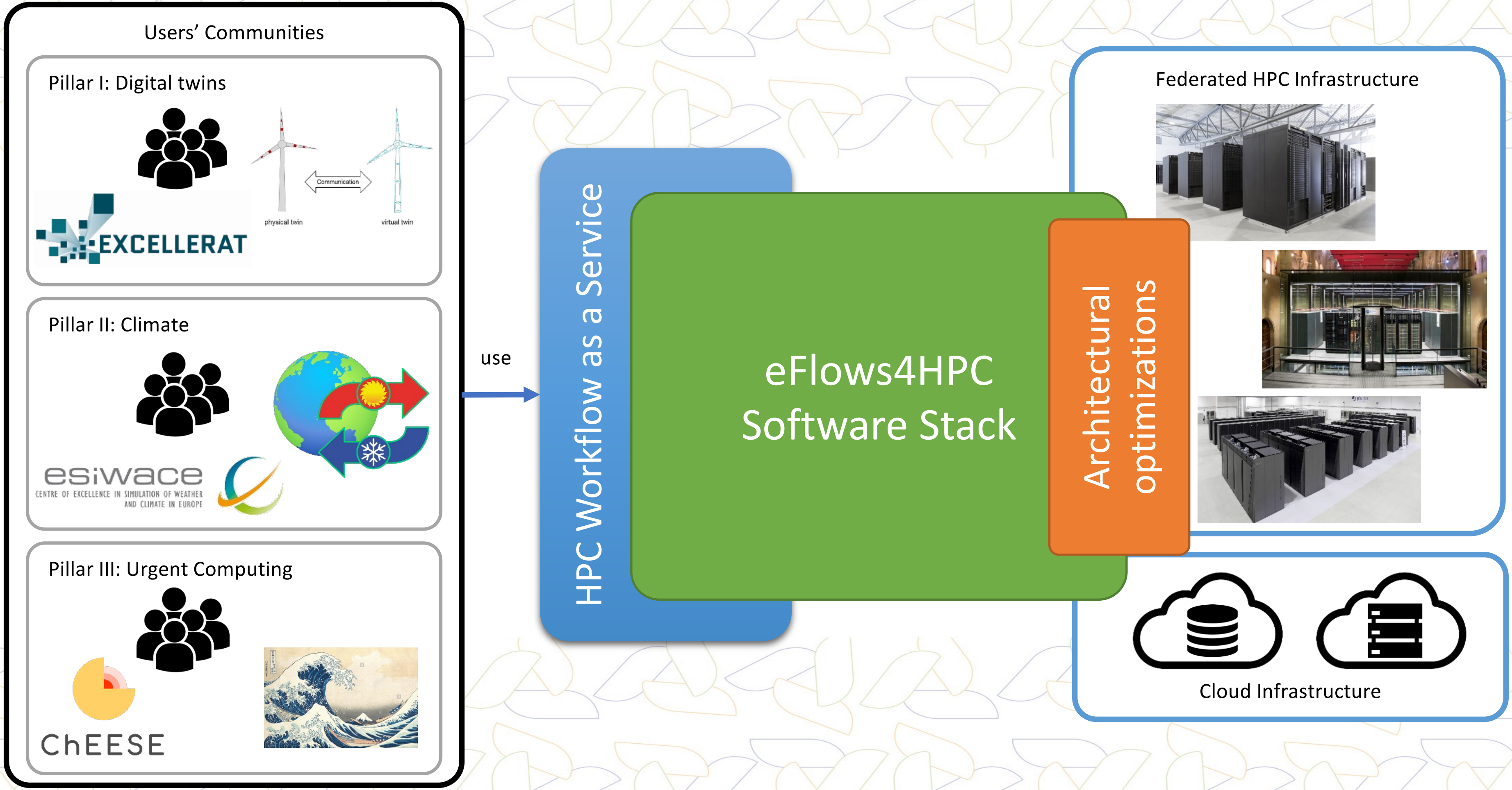

```
x = load_txt_file("train.csv", (10, 780))
x_test = load_txt_file("test.csv", (10, 780))

kmeans = KMeans(n_clusters=10)

kmeans.fit(x)

kmeans.predict(x_test)
```

EFLOWS4HPC OVERVIEW

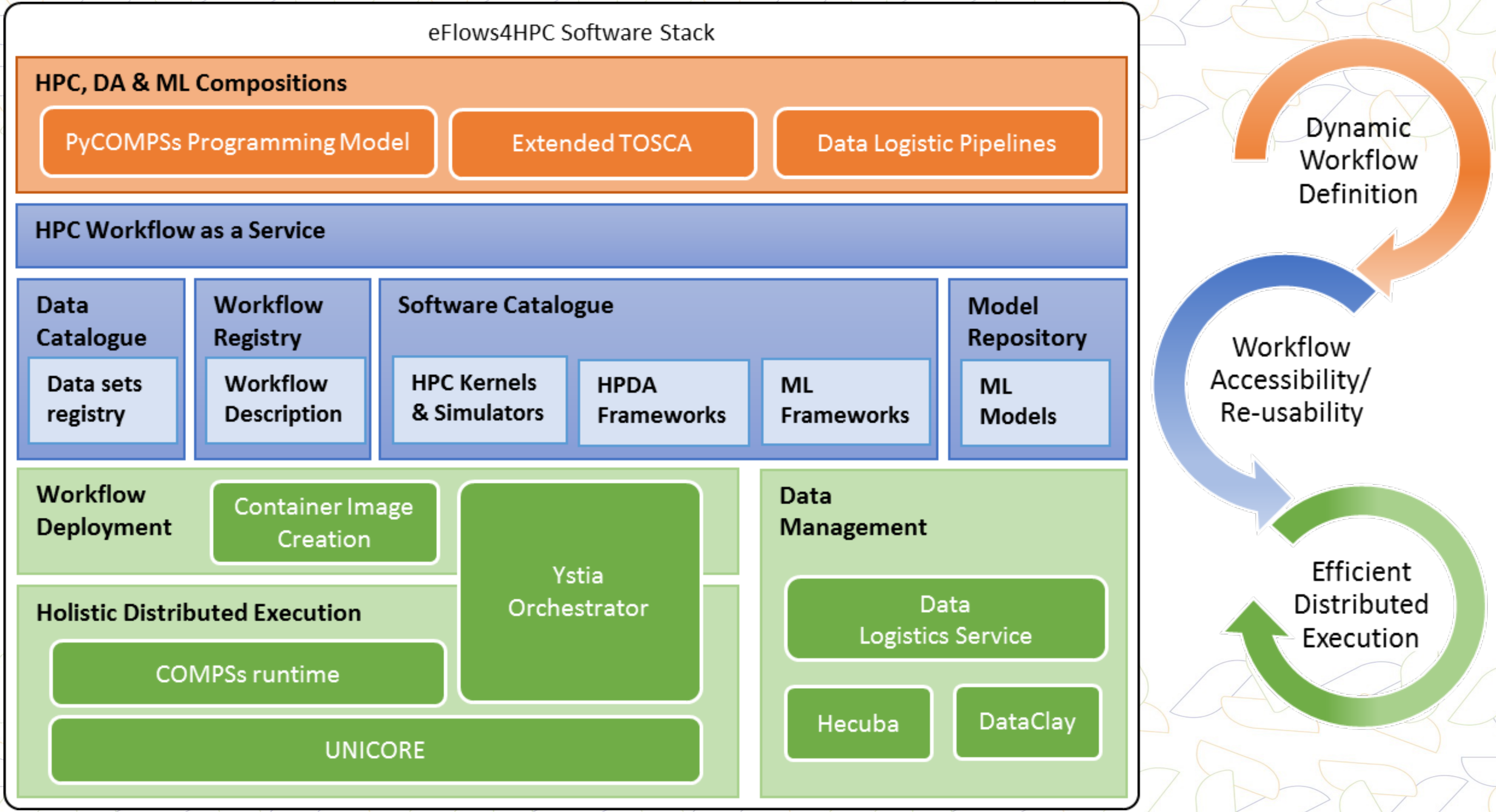
Main objectives

- Software tools stack that make it easier the development of workflows
  - HPC, AI + data analytics
  - Reactive and dynamic workflows
  - Efficient resource management
- HPC Workflows as a Service:
  - Mechanisms to make it easier the use and reuse of HPC by wider communities





Users' Communities
Pillar I: Digital twins
Communication
physical twin
virtual twin
EXCELLERAT
Pillar II: Climate
esiwace
CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE
Pillar III: Urgent Computing
ChEESE
use
HPC Workflow as a Service
eFlows4HPC Software Stack
Architectural optimizations
Federated HPC Infrastructure
Cloud Infrastructure


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

Pillar I: Manufacturing

CIMNE

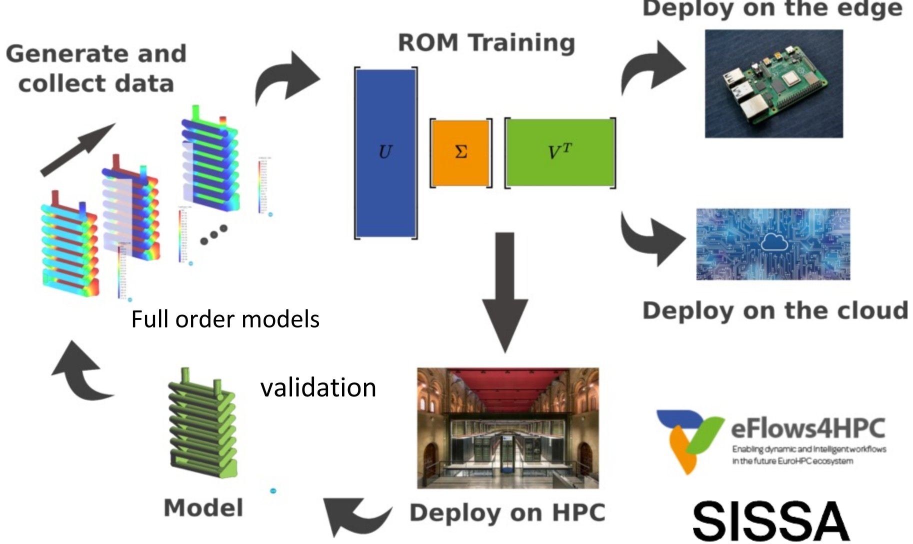
KRATOS MULTI-PHYSICS

Inria

Upgrade your meshes

ParMMg

SIEMENS

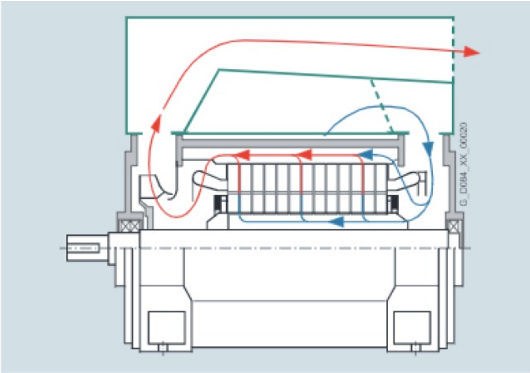


eFlows4HPC
Enabling dynamic and Intelligent workflows in the future EuroHPC ecosystem

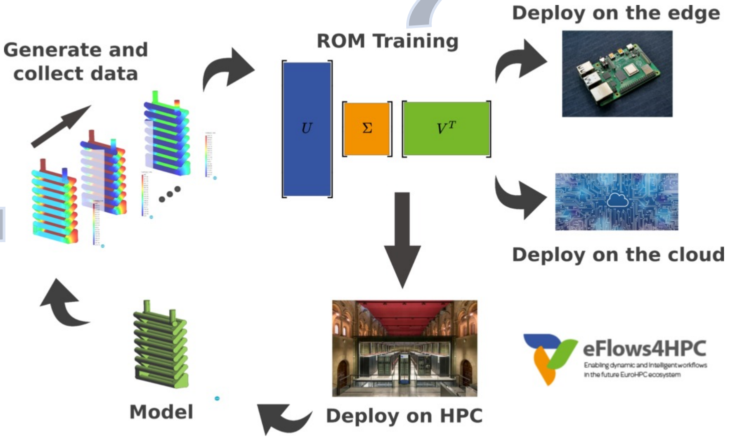
SISSA
Scuola Internazionale Superiore di Studi Avanzati

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



Integration of HPC and data analytics in a single workflow




```
import dislib as ds

@constraint(computingUnits=8)
@task(Y_blocks={Type: COLLECTION_IN, Depth: 2},
def my_qr(Y_blocks):
    Y = np.block(Y_blocks)
    Q,R = np.linalg.qr(Y, mode='reduced')
    return Q,R

def rsvd(A, desired_rank):
    k = desired_rank
    ...
    Y = A @ Omega
    Q,R = my_qr(Y._blocks)
    Q=load_blocks_rechunk([Q], ...)
    ...
```

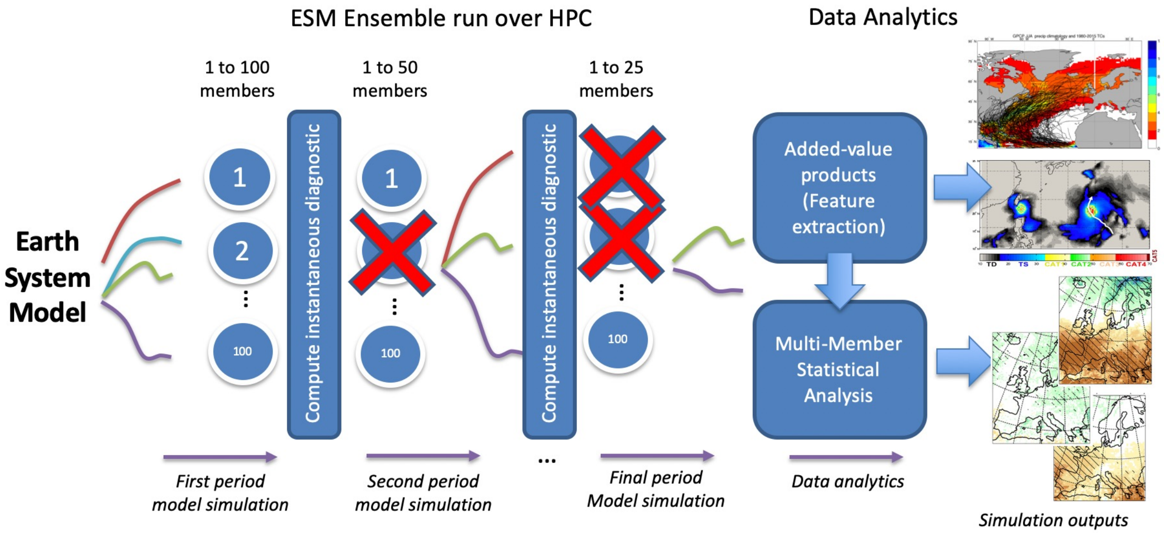
```
@constraint(computing_units="8")
@mpi(runner="mpirun", processes="16")
@task(returns = np.array)
def ExecuteInstance_Task(
    ...
    current_parameters =
    ...
    simulation = GetTrai
    simulation.Run()
    return simulation.GetSnapshotsMatrix()
```

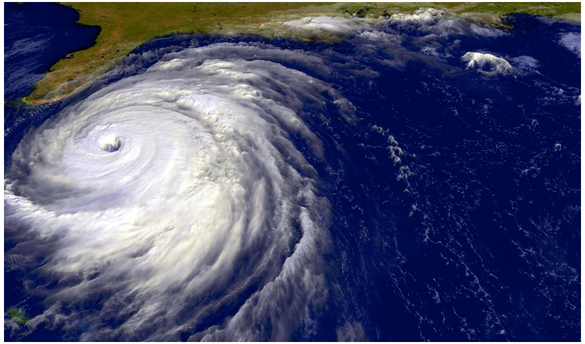
```
....
for instance in range (0,TotalNumberOFCases):
    blocks.append(ExecuteInstance_Task(model, parameters, pars, inst
...
U, s = rsvd(A, desired_rank)
...
```

Pillar II: Climate

cmcc Centro Euro-Mediterraneo sui Cambiamenti Climatici

eFlows4HPC





Dynamic (AI-assisted) workflow

ECMWF OpenIFS

FESOM2 Finite volumE Sea ice-Ocean Model

BSC Barcelona Supercomputing Center Centro Nacional de Supercomputación

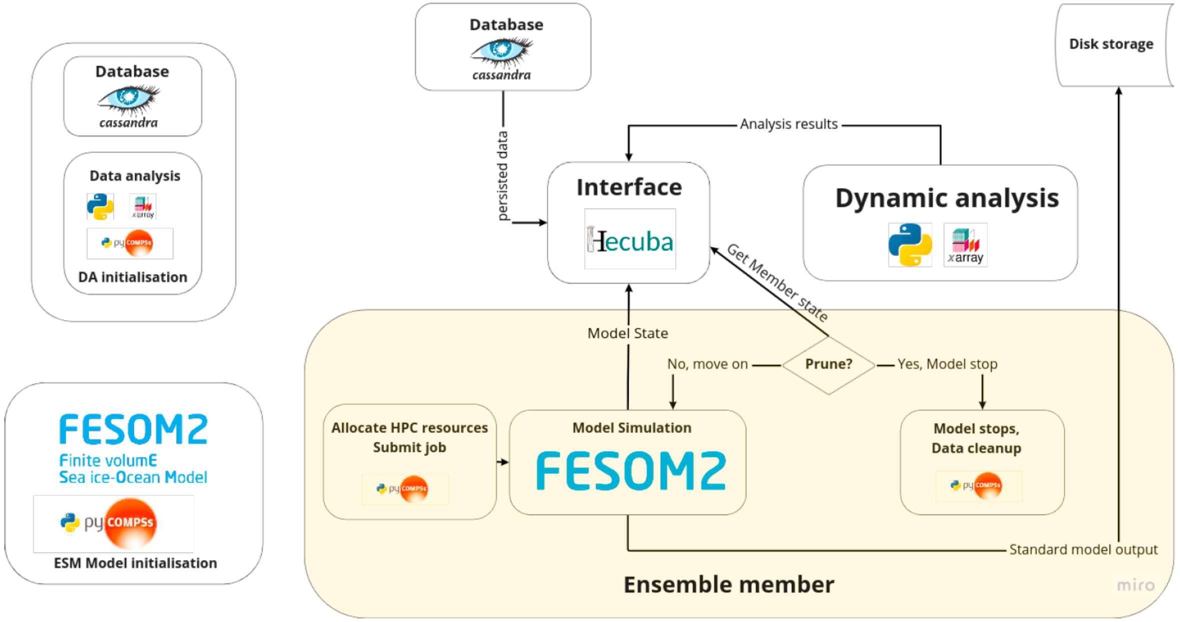
AWI ALFRED-WEGENER-INSTITUT HELMHOLTZ-ZENTRUM FÜR POLAR- UND MEERESFORSCHUNG

HPDA & ML/DL

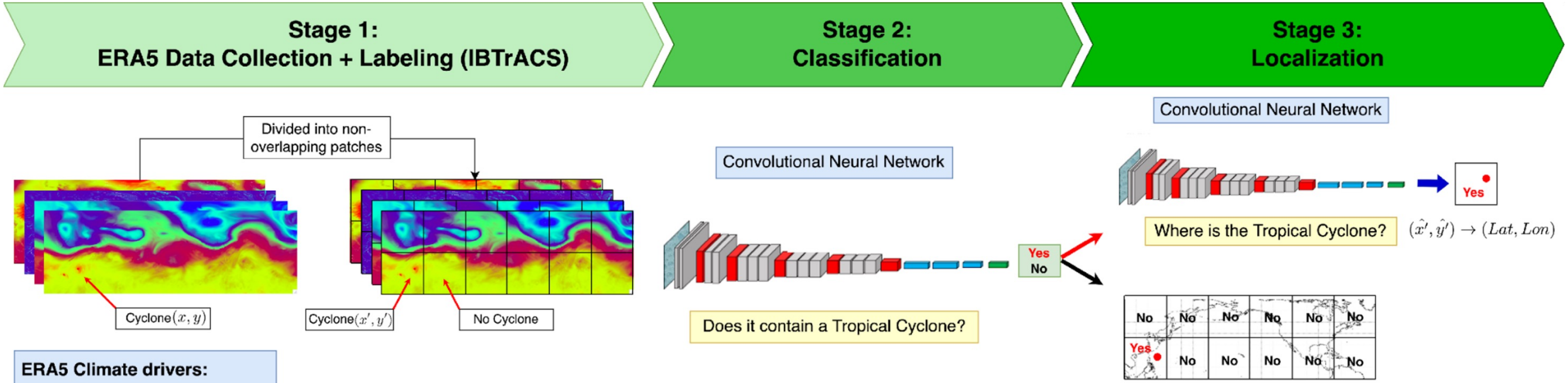
- Perform climate predictions: temperature, precipitation or wind speed
- AI-assisted pruning of the ESM workflow
- Study of Tropical Cyclones (TC) in the North Pacific, with in-situ analytics

ESM Dynamic Workflow

- PyCOMPSs workflow running an ensemble of FESOM simulations
- Intermediate simulation results stored persistently to Hecuba
- Pruning mechanism cancels given simulations based on dynamic analysis of data



Tropical Cyclones Detection ML Workflow: Training Phase



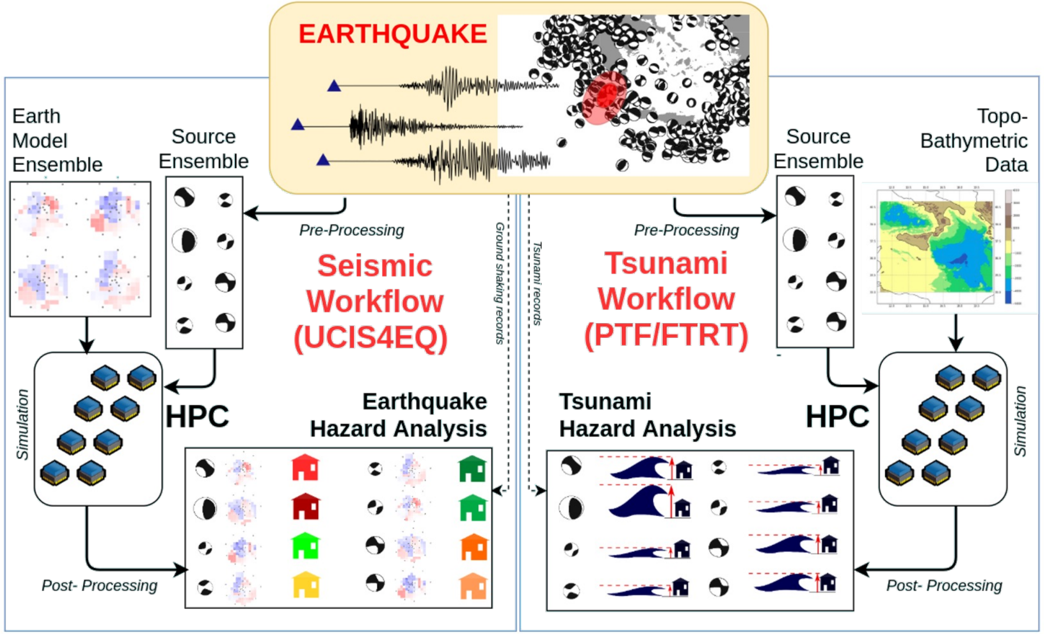
ERA5 Climate drivers:

- 10 m wind gust
- Temperature at 500 hPa
- Temperature at 300 hPa
- Mean sea level pressure

- Three main stages:
  - STAGE 1: Gathering of ERA5 climatic maps and generation of patches containing at most 1 tropical Cyclone (TC) each
  - STAGE 2: Classification of TC presence/absence inside the patch
  - STAGE 3: Localization of TC center coordinates in the patches in which the TC was previously classified as present

Pillar III: Urgent computing for natural hazards

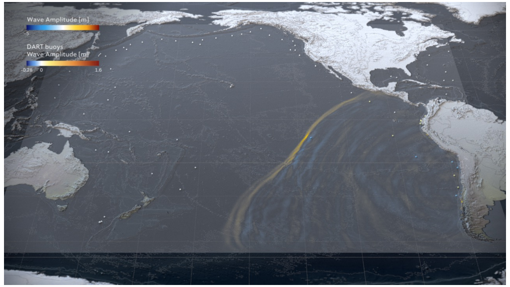
eFlows4HPC



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



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

ETH zürich

ISTITUTO NAZIONALE DI GEOFISICA E VULCANOLOGIA

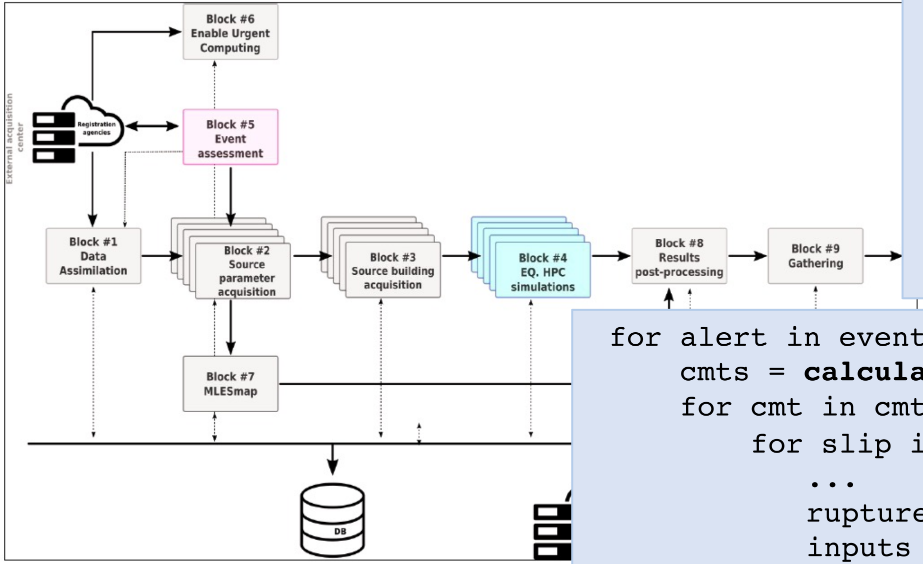
PSNC

UNIVERSIDAD DE MÁLAGA

NGI

UCIS4EQ workflow: http services as tasks

Urgent Computing Integrated Services for Earthquakes: UCIS4EQ



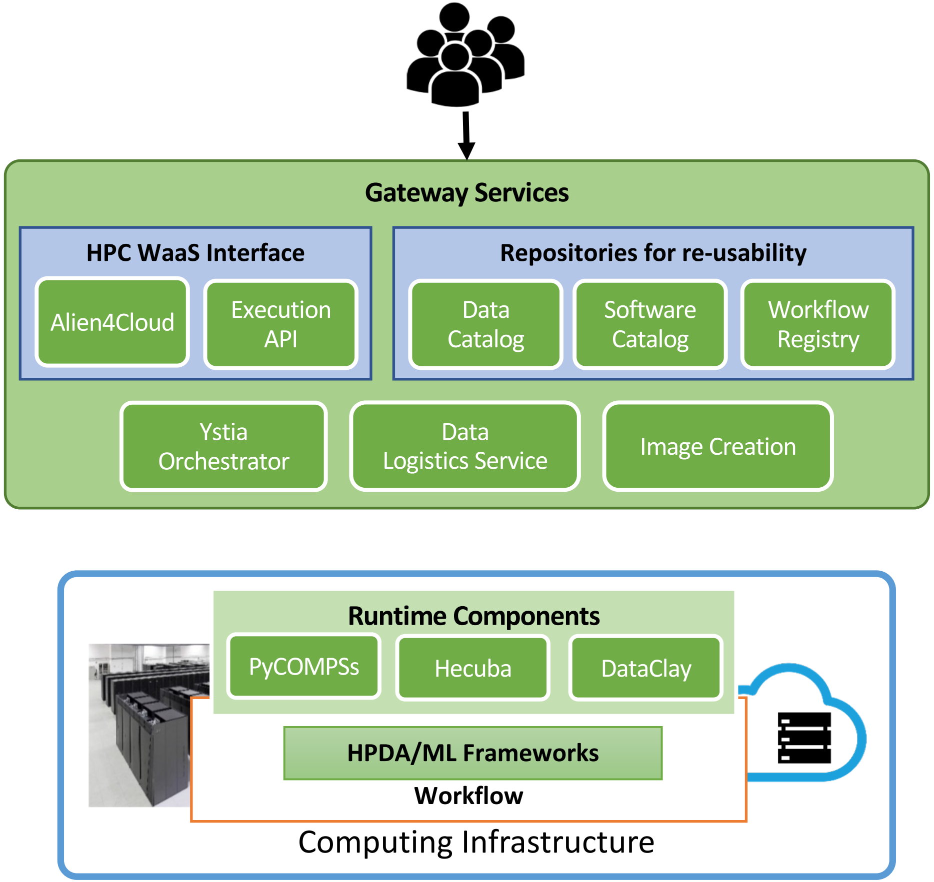

```
@http(request="POST", resource="SalvusRun", ...')
@task(returns=1)
def run_salvus(event_id, trial, input, resources):
    """
    """
    pass
@http(request="POST", resource="cmt",...)
@task(returns=1)
def calculate_cmt(alert, event_id, domain, precmt):
    """
    """
    pass
...
```

```
for alert in event['alerts’]:
    cmts = calculate_cmt(alert, eid, domain, precmt)
    for cmt in cmts.keys():
        for slip in range(1, region['GPSetup']['trials']+1)
            ...
            rupture = compute_graves_pitarka(eid, alert, ...)
            inputs = build_input_parameters( eid, alert, ...)
            salvus_inputs = build_salvus_parameters( eid, path, ...)
            result = run_salvus( eid, path, ...)
            all_results.append(run_salvus_post(eid, result, ...))
result = run_salvus_plots(eid, basename, domain, resources)
```

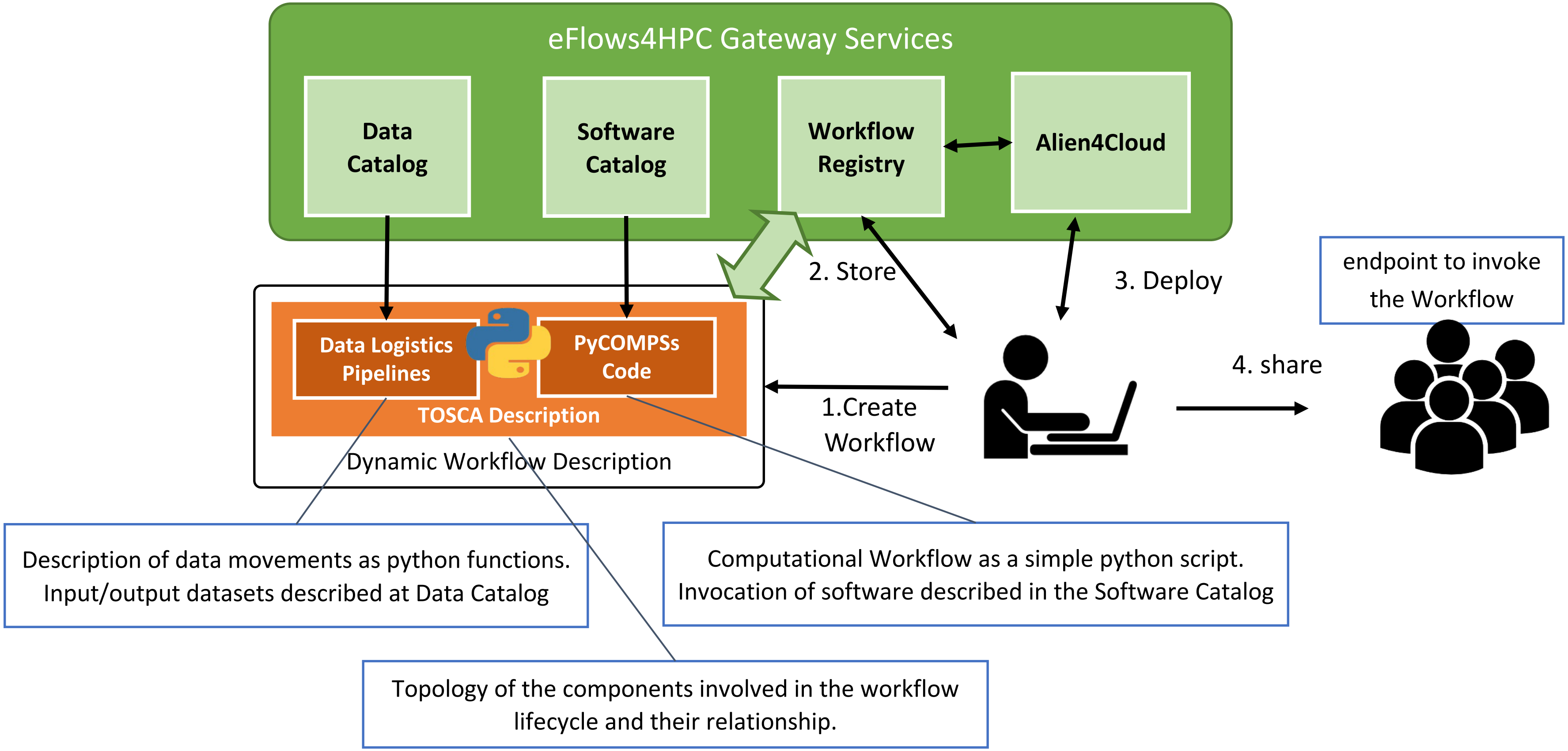
HPC WORKFLOWS AS A SERVICE

eFlows4HPC software stack and HPCWaaS

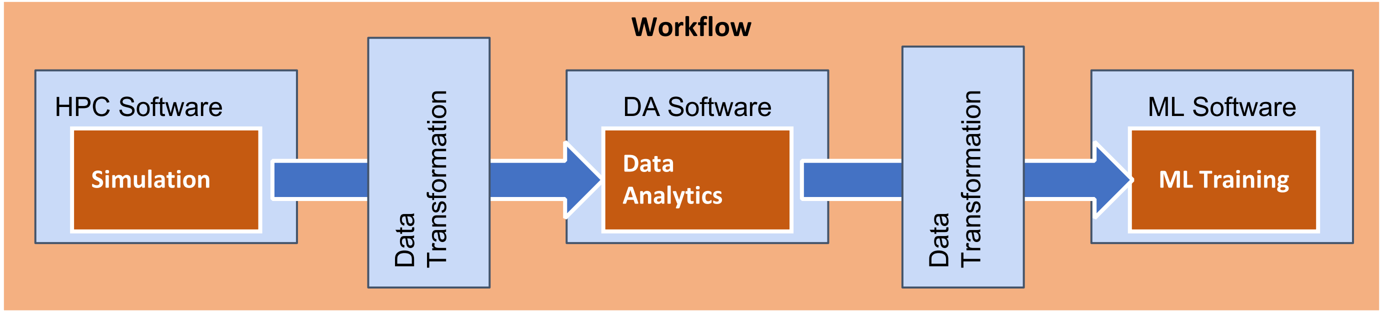
- Gateway services
  - Components deployed outside the computing infrastructure.
  - Managing external interactions and workflow lifecycle
- Runtime Components
  - Deployed inside the computing infrastructure to manage the workflow execution



Workflow development overview



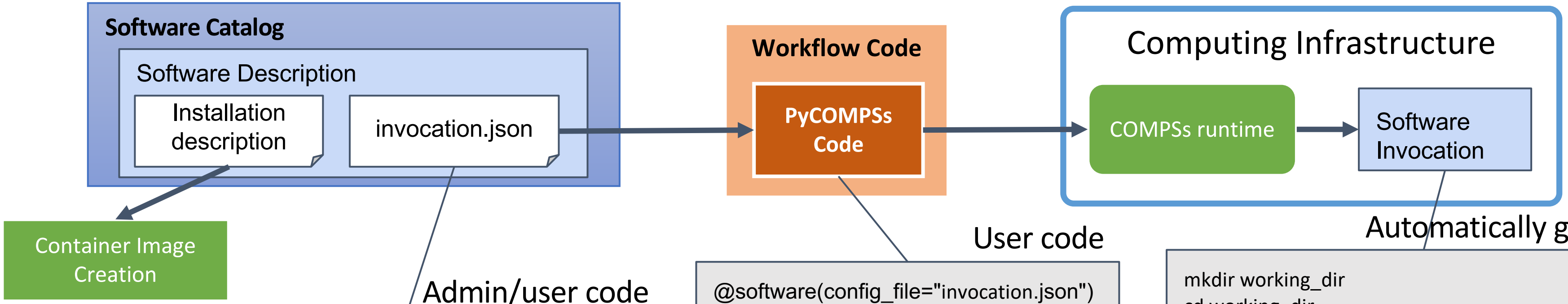
Interfaces to integrate HPC/DA/ML



- Goal:
  - Reduce glue code
  - Developer focuses in the functionality, not in the integration
  - Reusability
- Two paradigms:
  - Software integration
  - Data transformations

```
@data_tranformation(input_data, transformation description)
@software(invocation description)
def data_analytics (input_data, result):
    pass

#Workflow

simulation(input_cfg, sim_out)
data_analytics(sim_out, analysis_result)
ml_training(analysis_result, ml_model)
```

Software Invocation description



Admin/user code

```
{
 "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}
}
```

User code

```
@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')
```

Automatically generated

```
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 of a software invocation to a PyCOMPSs task
- Reused in different workflows

Data Catalogue and Data Logistics Service

Data Catalogue:

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

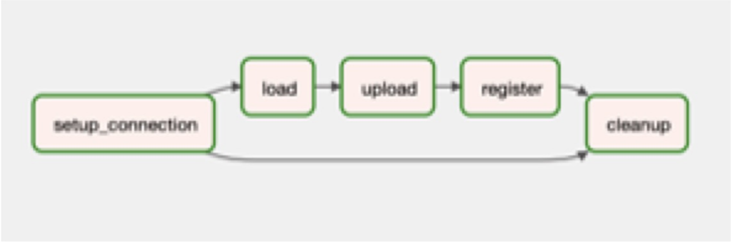
Data Pipelines:

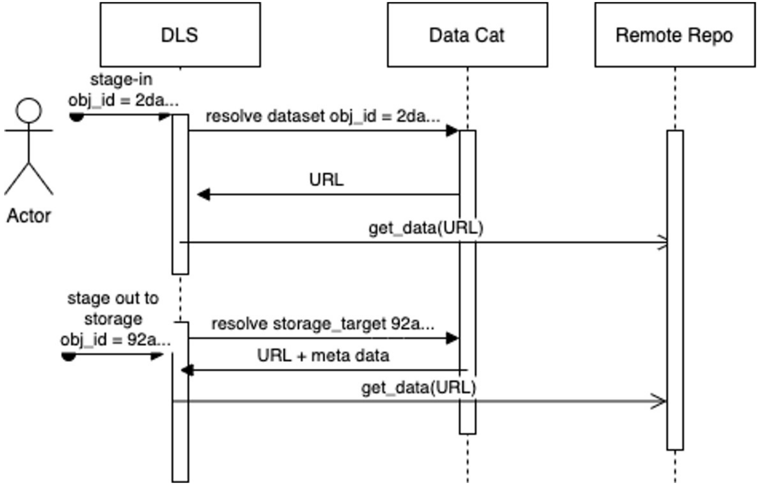
- Formalization of data movements for transparency and reusability
- Stage-in/out, image transfer

Data Logistics Services (DLS):

- Performs the execution of data pipelines to fuel Pillars' computations

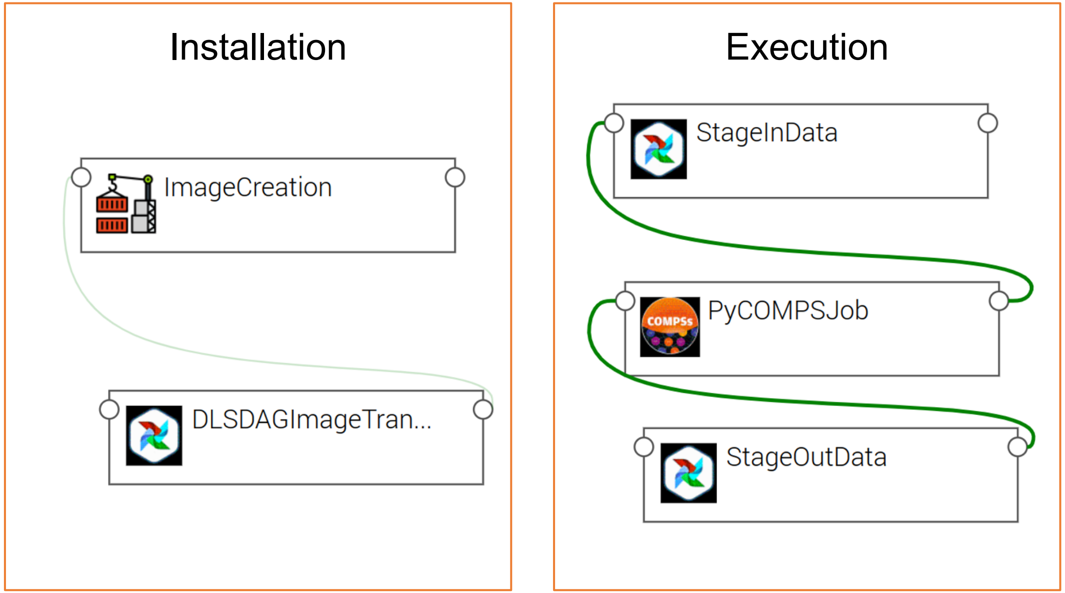
Data pipeline





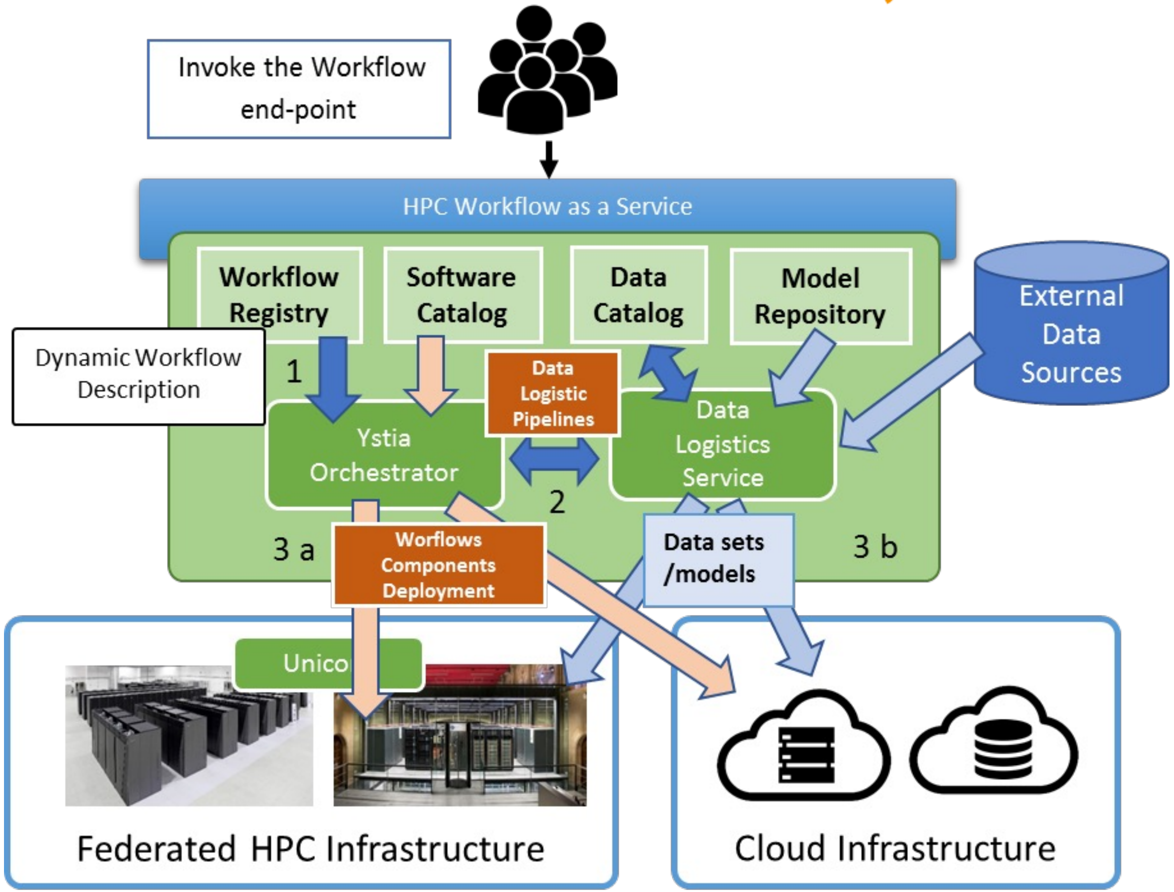
TOSCA Modelization

Topology of the different components involved in the Workflow lifecycle



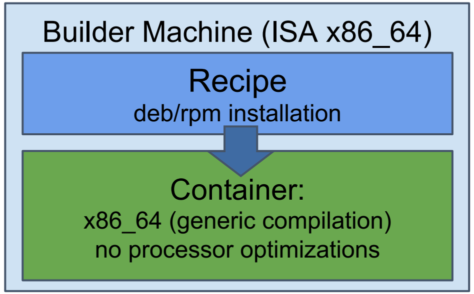
Deployment

- Invocation of a workflow
- Deployment orchestrated by Ystia Orchestrator (Yorc)
- Workflow retrieved from registry
- Data Logistic Service – data stage-in and stage-out
  - Periodical transfers of data outside HPC systems
- Deployment of workflow components in the computing infrastructures
  - HPC containers built with easybuild/Spack

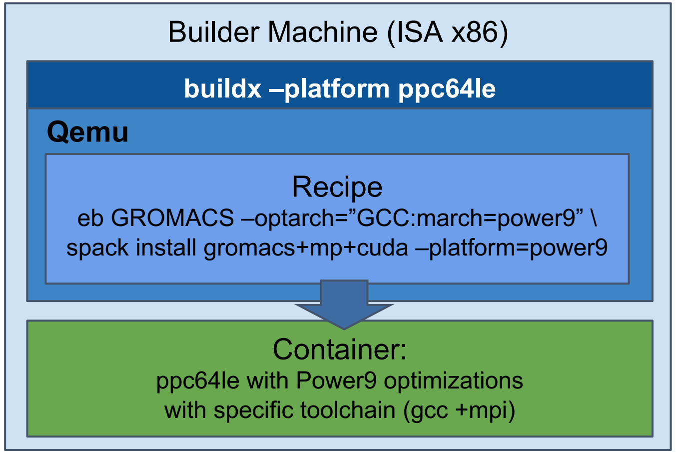


HPC Ready Containers

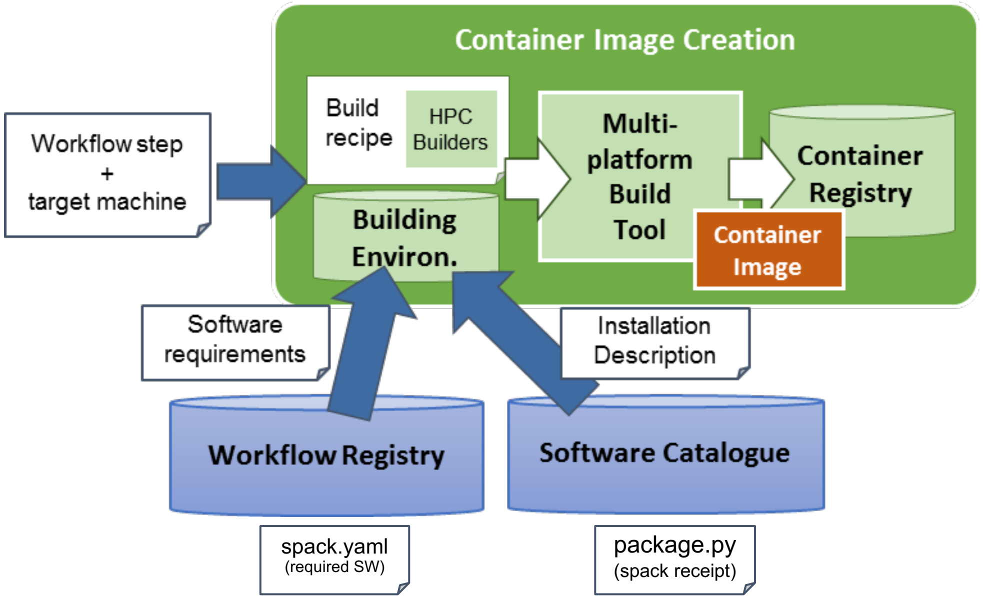
Standard container image creation



eFlows4HPC approach

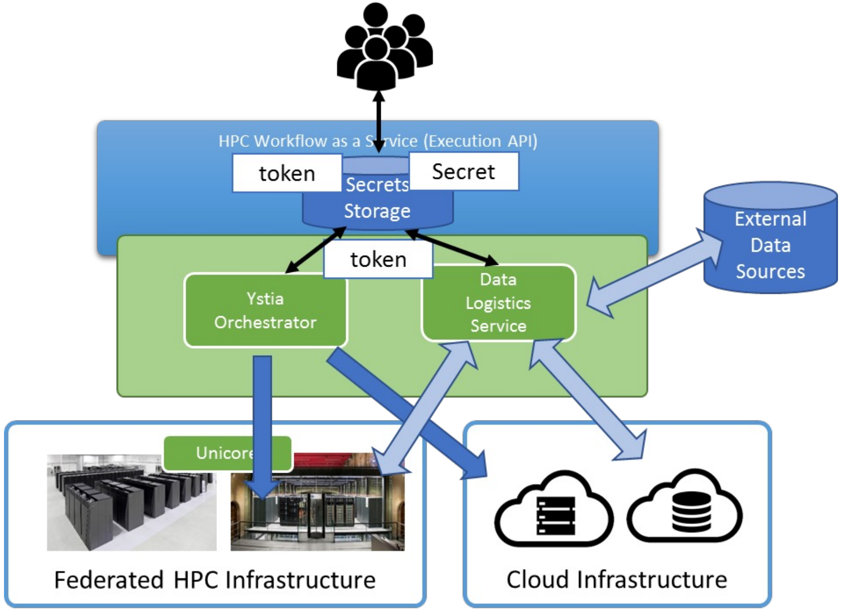


Service to automate the Container Image Creation



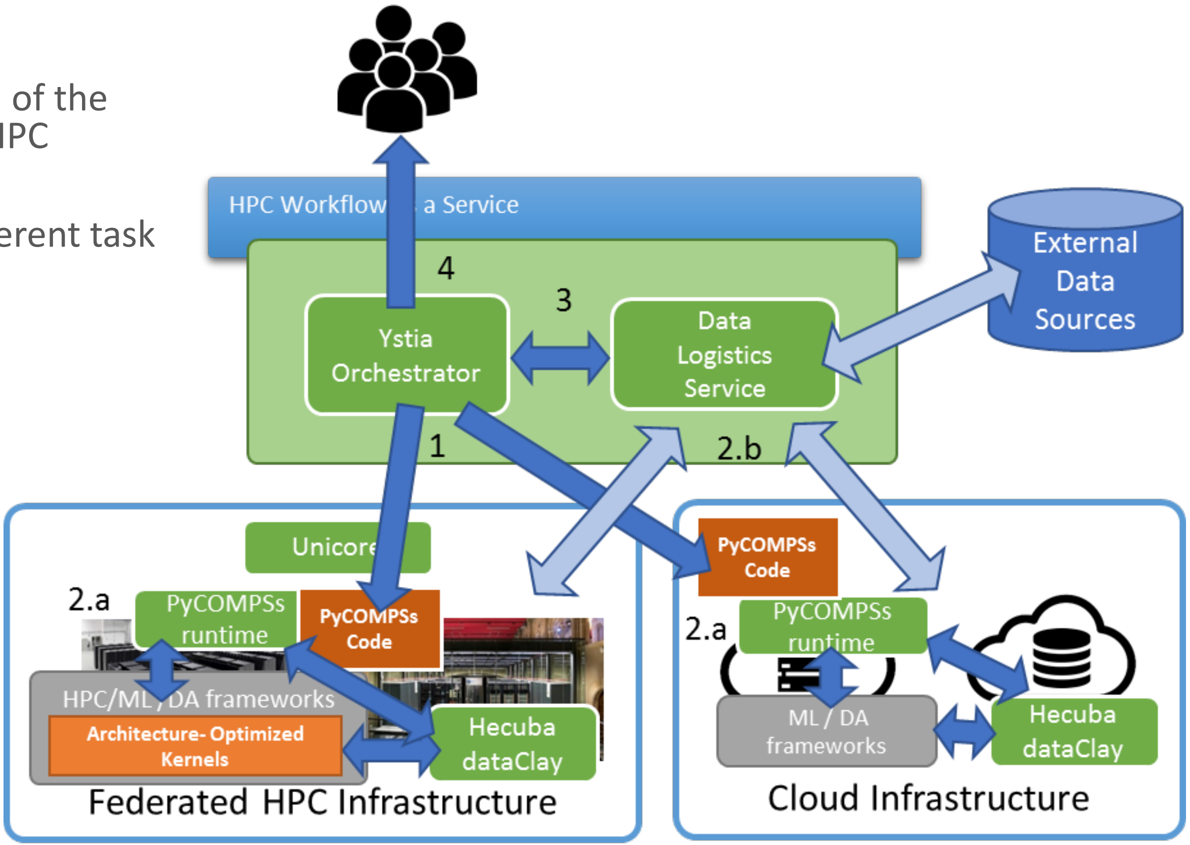
Credential management

- Prior to executing the registered workflows, the users have to configure the infrastructure access credentials
- Usernames, public-key certificates, passwords
- Users' certificates managed by an Execution API
  - Provides a few methods to register and access credentials or generate a new secret
  - A token is generated and returned to the user
  - HashiCorp Vault for secret (SSH keys) management
- User authorizes adding credentials in the HPC cluster
- Credentials identified by the token attached to the user's workflow invocation.



Operation- Workflow Execution

- Submission of the execution of the workflow processes to the HPC infrastructure
- PyCOMPSs orchestrates different task types
  - HPC (MPI), ML, DA
- Dynamic execution
  - Runtime task-graph
  - Task-level FT
  - Exceptions
- Data management
  - Persistent storage
- Optimized kernels
  - EPI, GPU, FPGA



Conclusions

- There is a need for providing tools for the development of complex workflows
- Complex workflows involve HPC aspects, artificial intelligence components and big data
- PyCOMPSs is a good alternative to define the behaviour of such workflows
- eFlows4HPC aims at providing a software stack that supports the development, deployment and execution of complex and dynamic workflows
- The HPCWaaS aims to provide a functionality similar for FaaS in cloud for complex workflows in HPC to make it easier the adoption of HPC technologies

Project partners

eFlows4HPC

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

Further Information

- eFlows4HPC webpage: https://eflows4hpc.eu
- COMPSs webpage page: http://www.bsc.es/compss
    - Documentation
    - Virtual Appliance for testing & sample applications
    - Tutorials
  - Source Code

  https://github.com/bsc-wdc/compss
  - Docker Image

  https://hub.docker.com/r/compss/compss
  - Applications

  https://github.com/bsc-wdc/apps

  https://github.com/bsc-wdc/dislib

Projects where COMPSs is involved

eFlows4HPC

Enabling dynamic and Intelligent workflows in the future EuroHPC ecosystem

Per Med CoE

HPC/Exascale Centre of Excellence in Personalised Medicine

AI SPRINT

AI IN SECURE PRIVACY-PRESERVING COMPUTING CONTINUUM

CAELESTIS

MEEP | MareNostrum Experimental Exascale Platform

CECH Cluster Emergente del Cerebro Humano

Barcelona Supercomputing Center

Centro Nacional de Supercomputación

Joint Laboratory for Extreme-Scale Computing

Inria NCSA Argonne BSC JÜLICH RIKEN

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.