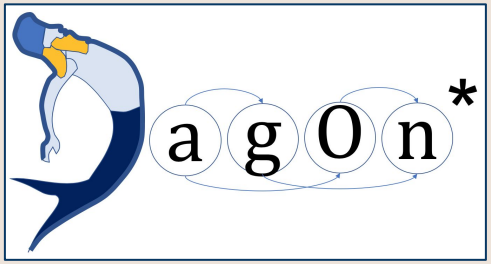
Tutorial: DAGonCAPIO and GLOBO-GLOWPP developing a computational workflow for orchestrating a global weather forecasting model

Gennaro Mellone, Raffaele Montella

University of Naples “Parthenope”, Napoli (Italy)

HIPES - The 2nd workshop about High-Performance e-Science

EURO-PAR
CONFERENCE 2025



Recap: DAGonStar

Direct Acyclic Graphs as parallel jobs on anything

DAGonStar is a production-oriented workflow engine:

- **Integrated** in the Python environment.
- **Minimal** footprint for external software components execution.
- **Avoiding** any workflow engine **centered data management**.
- **Checkpoints** for failover and execution recovery.
- **Straightforward** definition of tasks:
  - Python scripts.
  - Web interaction.
  - External software components.
- **Execution** sites **independence**:
  - Local / scheduler (SLURM).
  - Containers (Docker).
  - Clouds (AWS, OpenStack, DigitalOcean).

https://github.com/dagonstar/

The Programming model

Python Script: “DAGonStar Hello World App”

```
import dagon
…
workflow=Workflow(“myapp”,settings)
workflow.add_task(new Task(“a”,”...”))
workflow.add_task(new Task(“b”,”workflow:///a”))
workflow.add_task(new Task(“c”,”workflow:///a”))
workflow.add_task(new Task(“d”,”workflow:///b workflow:///
workflow.run()
sys.exit(0)
```

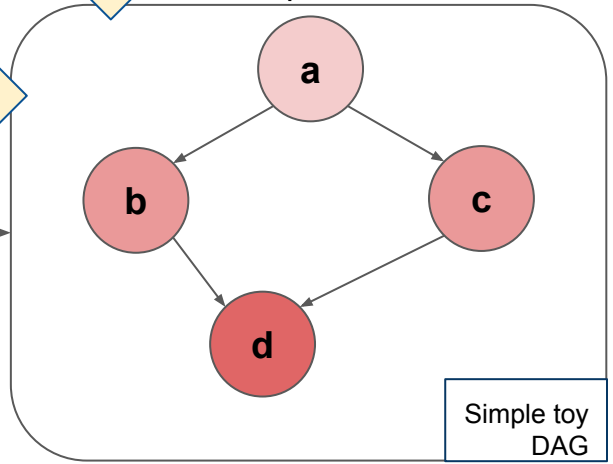
Task Flow

Defined by task dependencies.

Data Flow

Defined by data dependencies.

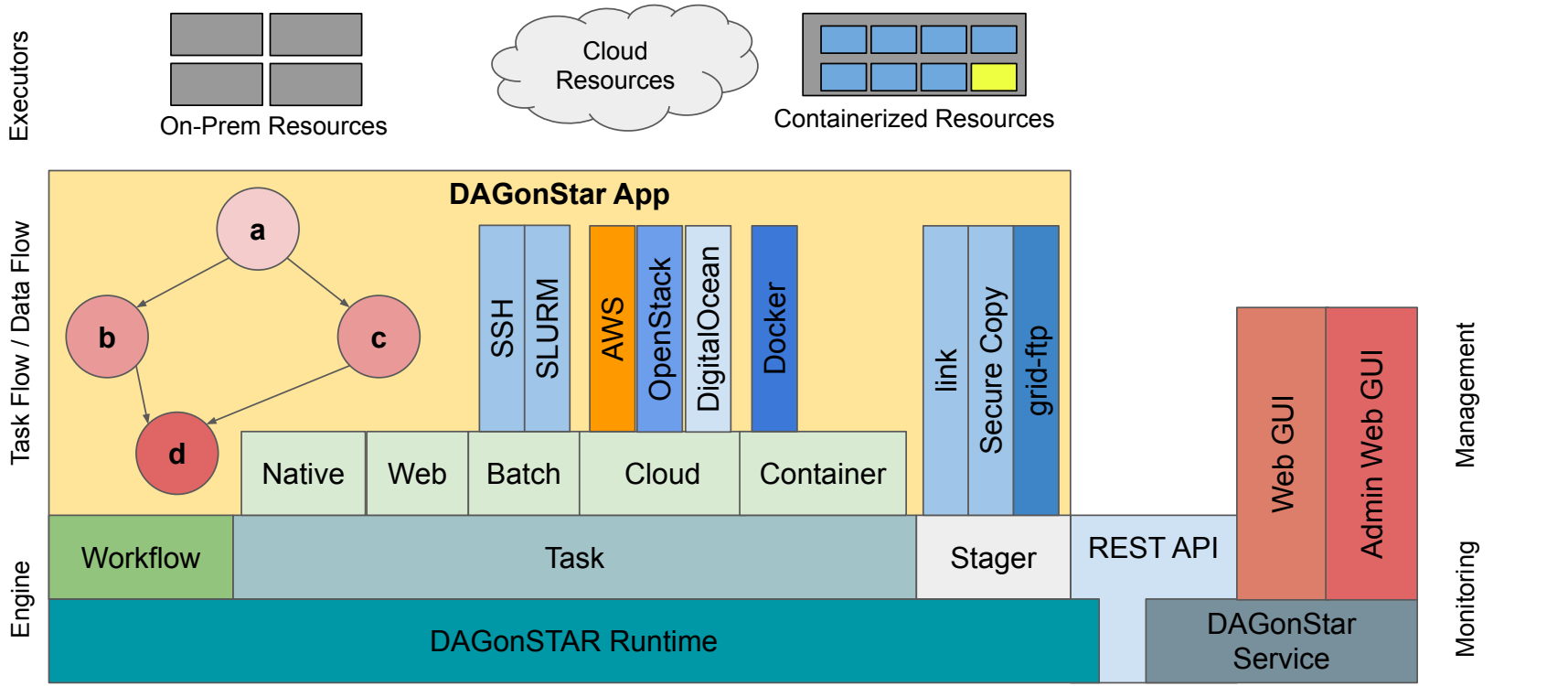
Data Flow



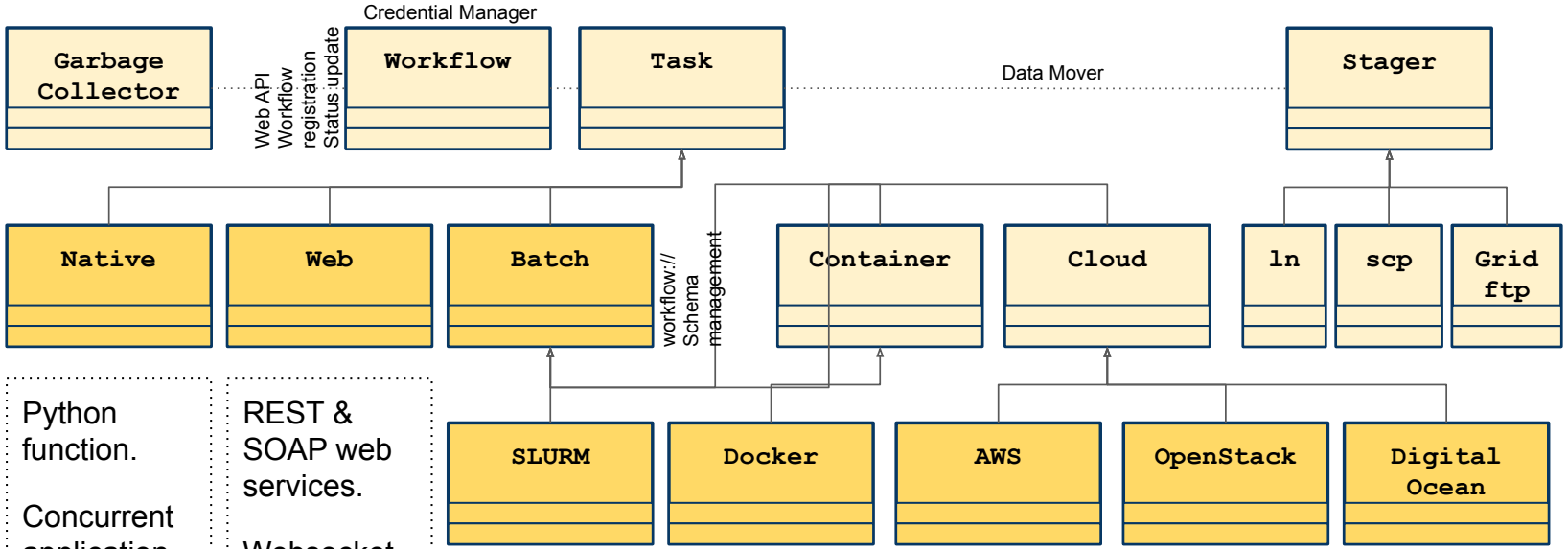
- Dealing with actual data files instead of high-level defined datasets.
- Performing backward data references in order to create dependencies.
- Having more Workflow instances in the same Python application.

DAonStar has been designed starting from the desired programming model.

Architecture



Components



Python function.

Concurrent application thread.

REST & SOAP web services.

Websocket subscribe & publish.

DagOn Service.

External software.

SSH or scheduler.

Task embedded in container.

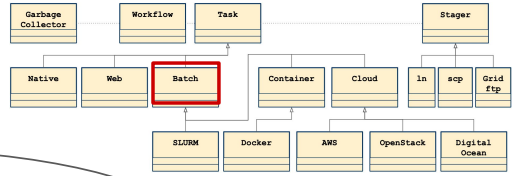
Task embedded in a virtual machine.

Instance status management.

Cloud independent.

The workflow:// schema

The **Batch** component takes charge of the management of data dependencies using the **workflow:// schema**.



workflow:// **workflow_unique_name**/**task_unique_name**/

The schema label

The workflow unique name
An UUID could be used
If empty means “current workflow”

The task unique name
Can be dynamically generated by the Python script when the workflow is created programmatically.

Task scratch directory root

Workflow

Stager

One Of...

ln

scp

Grid ftp

out

in

- Local
- Shared File System
- Remote scratch directory on physical machine, virtual instance or container

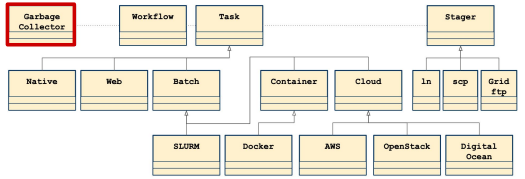
External software **sandbox**

workflow:///task_a/results.csv

Scratch directory

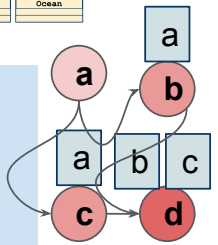
The garbage collector

- Tracks the storage and computational resources allocated during tasks execution.
- Proceeds to dispose them when no longer needed.



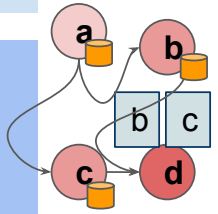
Make Dependencies

```
For each batch task in the <workflow> ...
      For each workflow://<workflow>/<task>/ reference in the task command line
...
            Increment the number of reference to <task>
```



On Task Finish

```
For each workflow://<workflow>/<task>/ reference in the task command line ...
      Decrement the number of reference to <task>
      If the number of reference to <task> is 0, clean up the involved resource
```

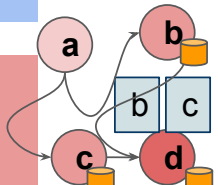


Clean Up

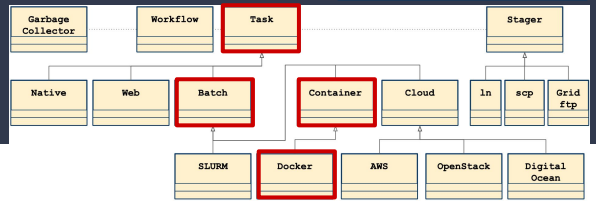
Local, remote or shared file system:
Remove the scratch directory.

Virtual machine instance:
Stop the instance.

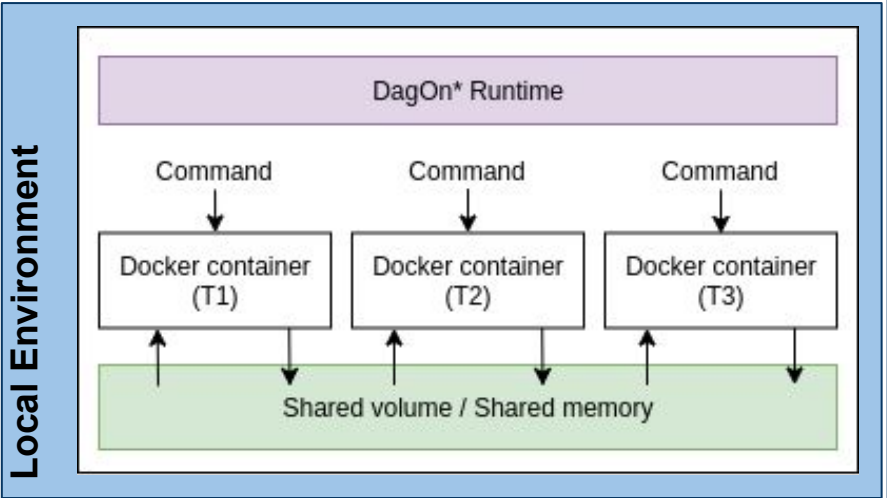
Container:
Stop the container.

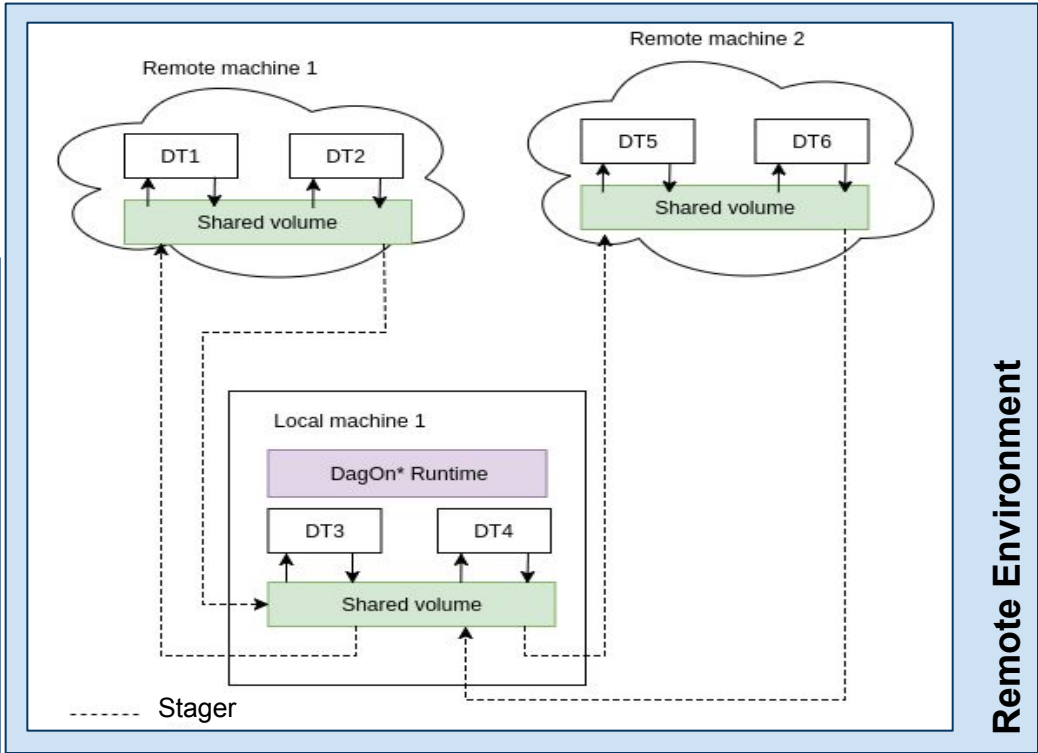


Container Tasks



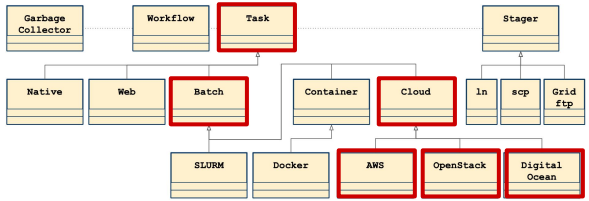
- Deployable in any Docker machine.
- Share a volume with the operating system host file system.
- If the tasks are on the same machine, the data transfer is done using shared memory.
- In a remote environment, data is copied to the volume shared between containers.

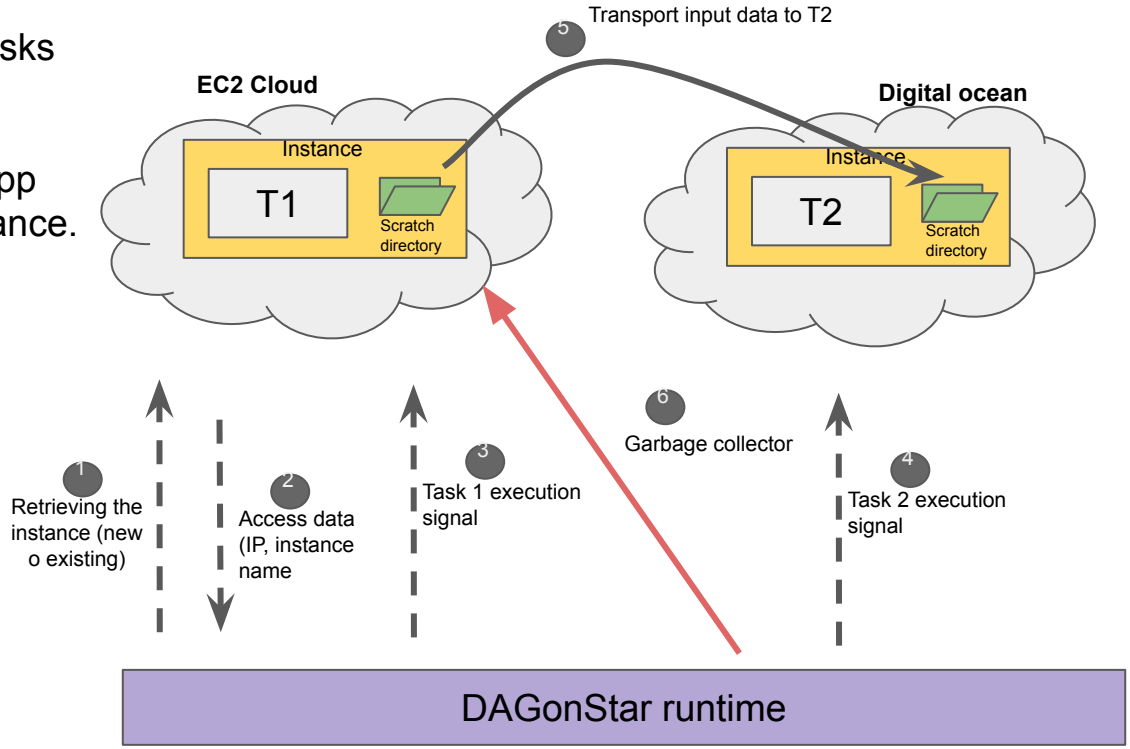




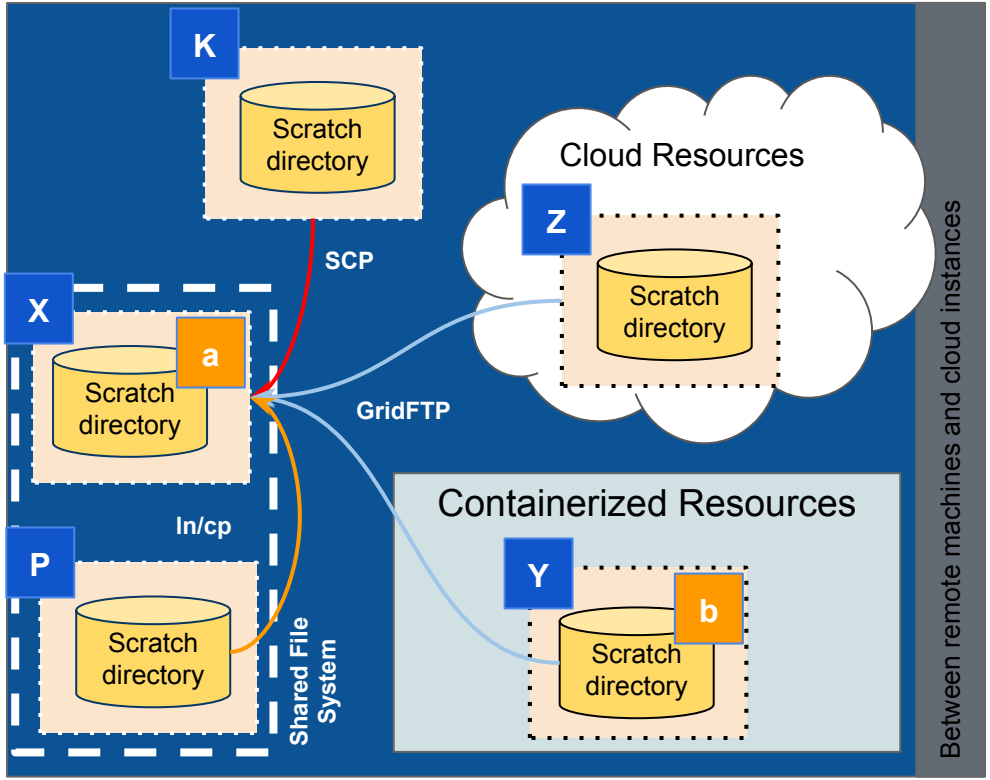
Cloud Tasks

- Deployable in private, public and hybrid clouds.
- Define programmatically the flavour and configuration of the instance.
- Interoperable with other types of tasks (batch, containers, etc).
- SSH is used to make the DagOn app controlling the virtual machine instance.
- Data is transferred between tasks using the Stager component.
- Leverage on Apache Libcloud
- Tested with:
  - AWS
  - OpenStack
  - Digital Ocean
  - **Google Cloud**



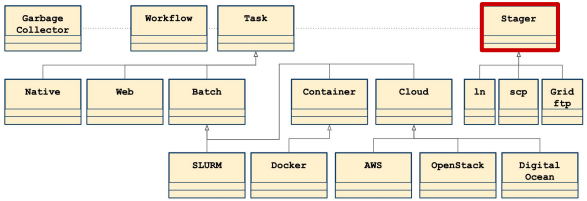


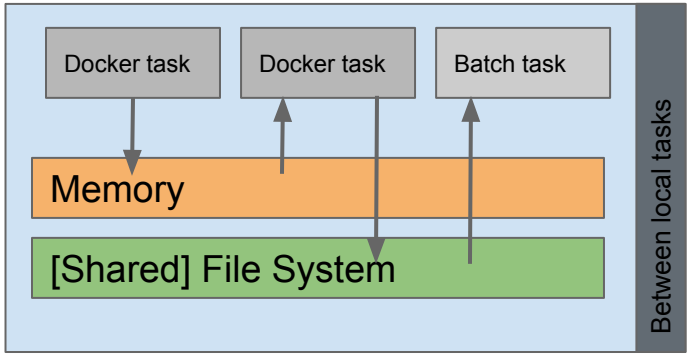
Staging




```
globus-url-copy -vb -p 4 gsiftp://X/tmp/a/f1 gsiftp://Y/tmp/b/f2
```

Globus Connect Server



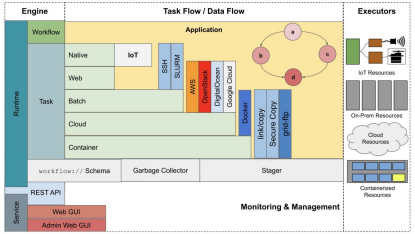


- Manages the data movement between all type of tasks.
- Fallback strategy:
  a. GridFTP
  b. Secure Copy
- Local tasks: memory, [shared] file system.

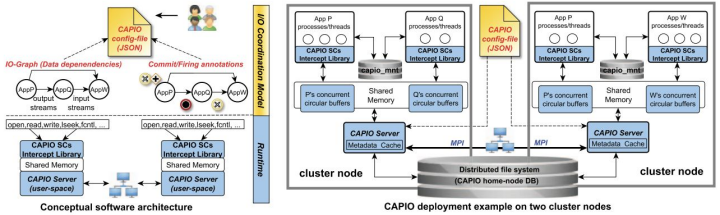
Staging

EURO-PAR CONFERENCE 2025

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE



+



=

- DAGonStar batch tasks generate the JSON file based on the dependencies between tasks, identified thanks to the workflow:// Schema;
- This JSON file is used by the CAPIO server for configuration;
- Tasks A and B make up a pipeline in which A produces files and B reads them;
- Posix calls made on these output files will be intercepted by the CAPIO server, allowing it to process this data in RAM.

Partners

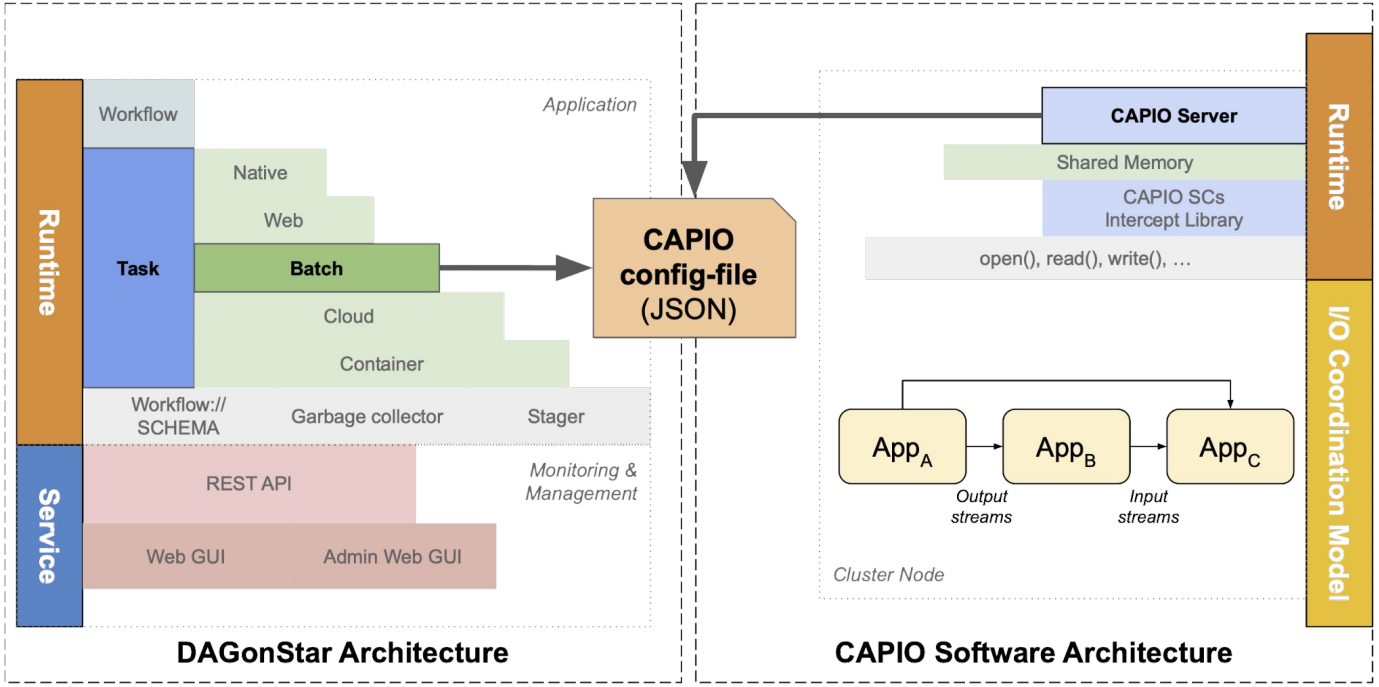
Università degli Studi di Napoli Parthenope

fides information technology

HYBRID CLOUD INTEGRATOR ITDM GROUP

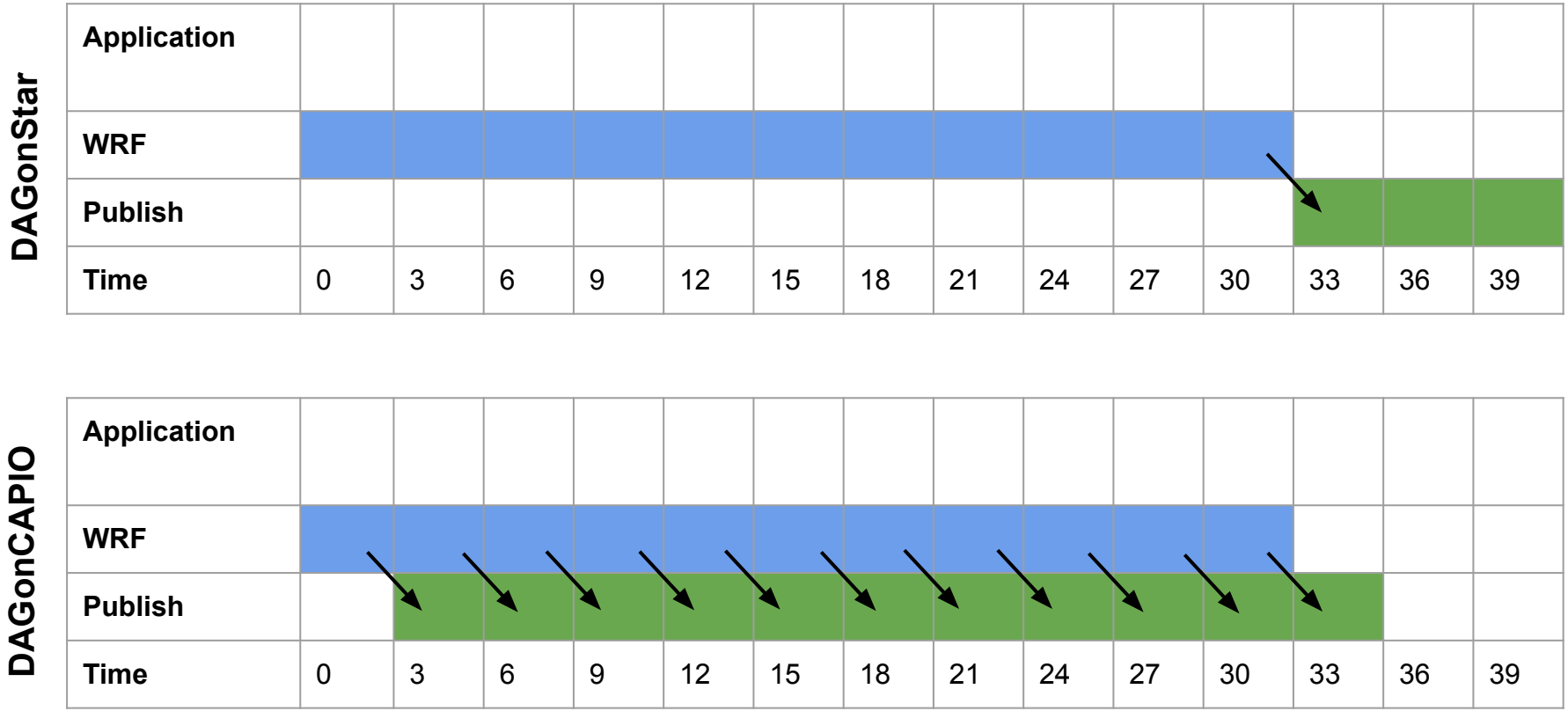
Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing
ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

DAGonStar and CAPIO Integration



Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing
ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

DAGonStar and CAPIO Integration



Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing
ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

Recap: GLOWPP

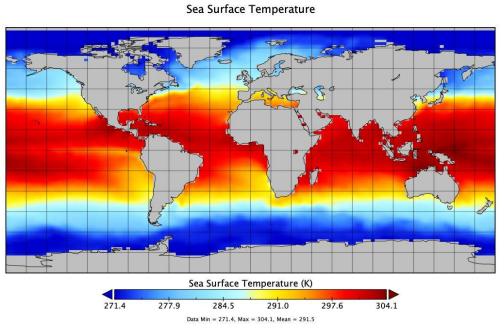
GLOWPP

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA | UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

https://glowpp-project.org

Refactoring, Optimization, and "Production-level" of GLOBO model developed by 'ISAC-CNR as an evolution of BOLAM (BOlogna Limited-Area Model).

- Overall code refactoring
- Moving statically allocated data structures to dynamical allocation
- Changing MPI point to point communication to collective communications (ghost boundaries management)
- Using shared memory techniques to parallel iterative sections
- Using OpenAcc to GPU-accelerate computing demanding solvers (i.e. radiance)



Global Weather model Prototype now and Production in the future

Hands On !

Our Objective:

Developing a computational workflow for orchestrating a global weather forecasting model.

Using DagonStar to manage **3 main tasks**:

1. **Simulation preparation** - script that saves parameters to a file (yaml file with pyglobo parameters)
2. **PyGLOBO Run**: running the PyGLOBO container that will run forecasts and saves a NetCDF file.
3. **Showing results**: Converting NetCDF file into a PNG image.

Preliminary Steps

Clone the project:

https://github.com/gennaromellone/hipes2025-tutorial

Requirements:

- **Docker** framework installed
- **Bash console** and related tools
- **Python 3.8+**

Environment Configuration:

- Create and enable virtualenv (not necessary, but highly recommended)

```
>python3 -m venv venv
>source venv/bin/activate
```

Preliminary Steps

Installing DAGonStar:

- Use pip to install the library:
  ```
  >pip install
  git+https://github.com/DagOnStar/dagonstar.git
  ```
- Create a folder for task execution.
  ```
  >mkdir runs/
  ```
- set the path into `dagon.ini` file:

  ```
  [batch]
  scratch_dir_base=/<your-path>/runs
  ```

- Test DAGonStar with a sample project
  ```
  >cd 0-preliminary-steps
  >python perliminary-test.py
  ```

1. Generate configuration data

1. Create a simulation script file

   `>cd 1-config/`

- Let's define the first task: we want to generate configuration data required by PyGlobo for the next steps
- Create or edit a `hipes-workflow.py`
- `> python hipes-workflow.py`

```
29  # --- Task A: Generate configuration data ---
30  # In this task we are going to generate configuration data useful for the next task.
31  # We will set "procx" and "procx" for setting the right number of cores required by PyGlobo, and "date"
32  # a date to forecast.
33  taskA = DagonTask(TaskType.BATCH, "A",
34                    " echo 2 > procx.txt; \
35                      echo 3 > procy.txt; \
36                      echo 2021-11-17 > date.txt; \
37                      echo Data created!"
38                    )
```

2. Launch GLOBO docker image

```
>cd 2-pyglobo/
```

- Clone the repository with inside the Dockerfile:
  ```
  >git clone https://git.isac.cnr.it/montella/globone-glowpp
  >cd globone-glowpp
  ```
- Download initial/boundary conditions (IC/BC):
  ```
  > wget
  http://wilma.to.isac.cnr.it/diss/globo/globo-ic-bc-20241016.tar.gz
  >mkdir -p input_data
  >tar -xzf globo-ic-bc-20241016.tar.gz -C input_data
  ```

!! You will need at least 30Gb of free space to support container's image and boundary conditions !!

2. Launch GLOBO docker image

- Access root if you have compiling problems

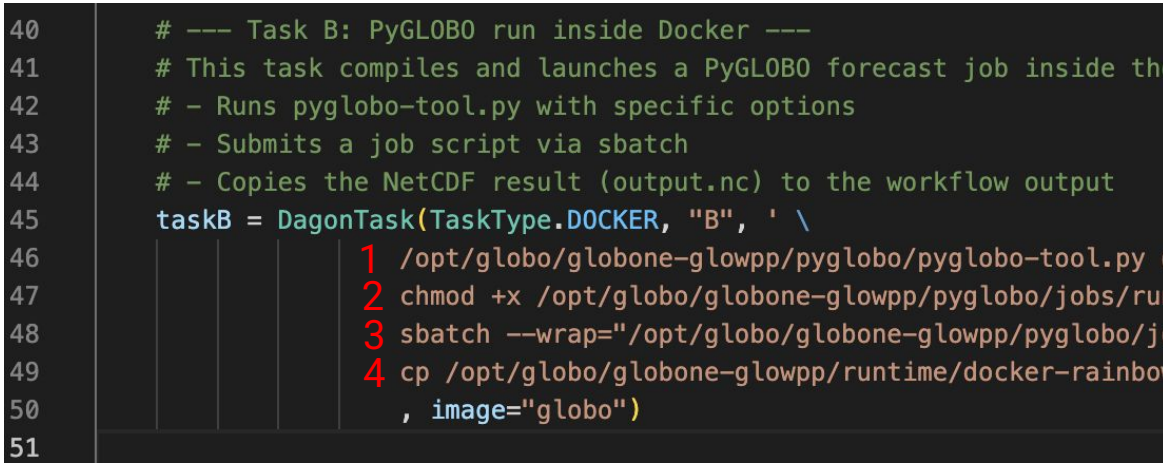
  ```
  > sudo su
  ```

- Build “globo” container

  ```
  > docker build -t globo .
  ```

2. Analyze the Task

1. Runs pyglobo tools in **compilers** mode and generates a slurm job. Using the previous settings.
2. **Set user permission** to the slurm job.
3. **Launch the slurm job** and wait for its completion (about 110 sec).
4. **Copy the forecasting** NetCDF file in a sample file.



2. Launch GLOBO docker image

- Test the new task
  `> python hipes-workflow.py`
- Check the outputs in the `run/` folder

- Go to parents folder:
  `> cd ..`

3. Build map2png container image

> cd 3-map2png/

- Build the Docker `map2png` image (remember to be root):

  > docker build -t map2png .

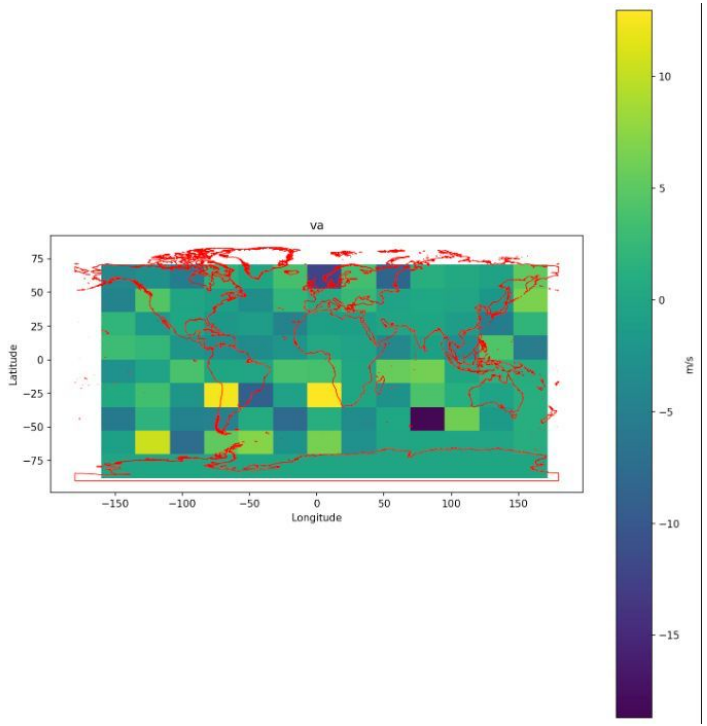
For any map2png documentation see:

> 3-map2png/README.md

```
52      # --- Task C: Visualization step ---
53      # Converts the NetCDF file produced in Task B into a PNG image using the "convert-png" Docker image.
54      # The input is read from Task B's output (workflow:///B/output.nc).
55      taskC = DagonTask(
56          TaskType.DOCKER,
57          "C",
58          "/app/map2png.py --in workflow:///B/output.nc --out result.png --var va --shp shapes/world-continents.shp ",
59          image="map2png"
60      )
61
```

3. Visualize our output file!

- Test the new task
  `>python hipes-workflow.py`
- Check the outputs in the `run/` folder



Conclusion

You have built a computational workflow with DagonStar.

Three main tasks successfully orchestrated:

- **Configuration** → YAML parameters for PyGLOBO
- **Forecasting** → Run PyGLOBO in Docker (NetCDF output)
- **Visualization** → Convert NetCDF into PNG

What You Learned:

- How to set up and configure DagonStar.
- How to integrate Docker containers inside a workflow.
- How to chain tasks to build a reproducible pipeline.

Stay in touch:

gennaro.mellone1@studenti.uniparthenope.it

raffaele.montella@uniparthenope.it