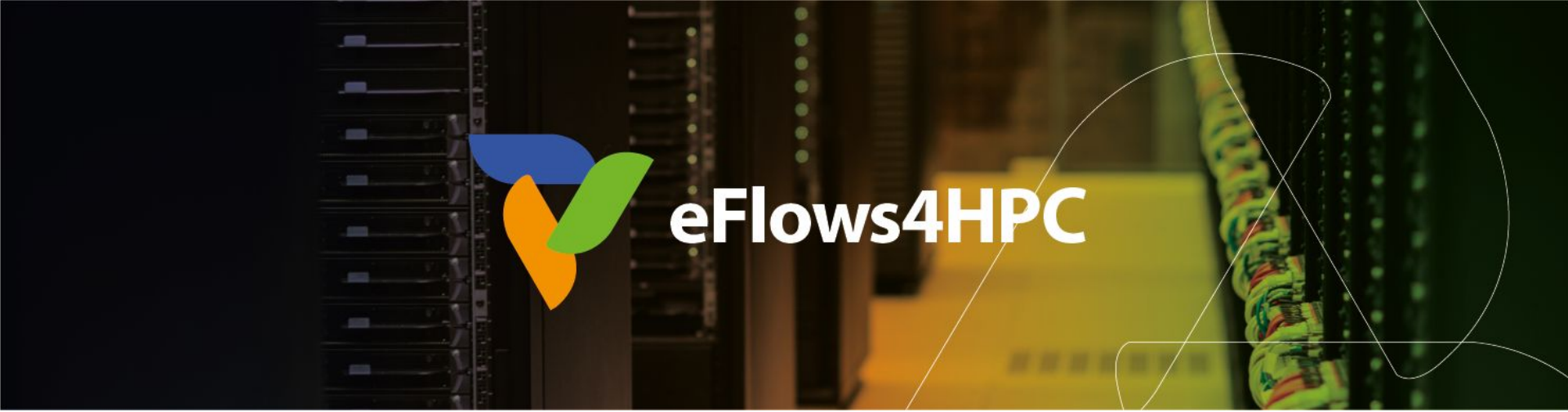


ESM Dynamic Workflow for Climate Simulations

Suvarchal K. Cheedela (AWI)

Nikolay Koldunov (AWI), Yolanda Becerra (BSC) , Juanjo Costa (UPC), Bruno Kinoshita(BSC), Rohan Ahmed(BSC), Miguel Castrillo(BSC), Julian R. Berlin (BSC) and many others

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.

Why workflows for ESM simulations?

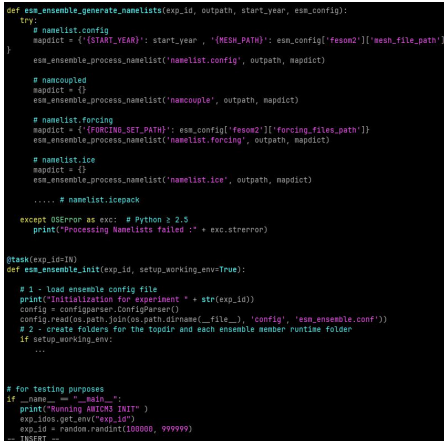
There are lot of things that are common for how climate models and their data are used across HPC centers.

Typical patterns:

- Build the model, optionally with needed dependencies.
- Prepare input data for the simulations: could be remote and on demand.
- Run the model with a certain configuration and resources.
- Analyze the model output data.
- Transfer the results: data or images

A workflow is a pipeline of any combination of these patterns.

How to workflows for ESM



```
def esm_ensemble_generate_namelists(exp_id, outpath, start_year, esm_config):
   try:
      # namelist.config
      mapdict = {'{START_YEAR}': start_year , '{MESH_PATH}': esm_config['fesom2']['mesh_file_path']
}
      esm_ensemble_process_namelist('namelist.config', outpath, mapdict)

      # namcoupled
      mapdict = {}
      esm_ensemble_process_namelist('namcouple', outpath, mapdict)

      # namelist.forcing
      mapdict = {'{FORCING_SET_PATH}': esm_config['fesom2']['forcing_files_path']}
      esm_ensemble_process_namelist('namelist.forcing', outpath, mapdict)

      # namelist.ice
      mapdict = {}
      esm_ensemble_process_namelist('namelist.ice', outpath, mapdict)

      ..... # namelist.icepack

   except OSError as exc:  # Python ≥ 2.5
      print("Processing Namelists failed :" + exc.strerror)


@task(exp_id=IN)
def esm_ensemble_init(exp_id, setup_working_env=True):

   # 1 - load ensemble config file
   print("Initialization for experiment " + str(exp_id))
   config = configparser.ConfigParser()
   config.read(os.path.join(os.path.dirname(__file__), 'config', 'esm_ensemble.conf'))
   # 2 - create folders for the topdir and each ensemble member runtime folder
   if setup_working_env:
      ...



# for testing purposes
if __name__ == "__main__":
   print("Running AWICM3 INIT" )
   exp_ids.get_env("exp_id")
   exp_id = random.randint(100000, 999999)
   INSERT
```

PyCOMPSS tasks

- Modularize each component of a workflow and make it composable: enables to have a clean, versionable, reusable workflows.
- Make the components interact and with job scheduler, enables modern, efficient workflows. – PyCOMPSS

How to workflows for ESM

```
def esm_ensemble_generate_namelists(exp_id, outpath, start_year, esm_config):
   try:
      # namelist.config
      mapdict = {'{START_YEAR}': start_year , '{MESH_PATH}': esm_config['fesom2']['mesh_file_path']
}
      esm_ensemble_process_namelist('namelist.config', outpath, mapdict)

      # namcoupled
      mapdict = {}
      esm_ensemble_process_namelist('namcouple', outpath, mapdict)

      # namelist.forcing
      mapdict = {'{FORCING_SET_PATH}': esm_config['fesom2']['forcing_files_path']}
      esm_ensemble_process_namelist('namelist.forcing', outpath, mapdict)

      # namelist.ice
      mapdict = {}
      esm_ensemble_process_namelist('namelist.ice', outpath, mapdict)

      ..... # namelist.icepack

   except OSError as exc:  # Python ≥ 2.5
      print("Processing Namelists failed :" + exc.strerror)


@task(exp_id=IN)
def esm_ensemble_init(exp_id, setup_working_env=True):

   # 1 - load ensemble config file
   print("Initialization for experiment " + str(exp_id))
   config = configparser.ConfigParser()
   config.read(os.path.join(os.path.dirname(__file__), 'config', 'esm_ensemble.conf'))
   # 2 - create folders for the topdir and each ensemble member runtime folder
   if setup_working_env:
      ...


# for testing purposes
if __name__ == "__main__":
   print("Running ANICMS INIT" )
   exp_ids.get_env("exp_id")
   exp_id = random.randint(100000, 999999)
   INSERT ...
```

PyCOMPSS tasks

Tosca topology

```
tosca_definitions_version: tosca_2_0
metadata:
  template_version: 0.1.0-SNAPSHOT
  template_author: julian

description: ""

imports:
  - yorc-types:1.1.0
  - tosca-normative-types:1.0.0-ALIEN20
  - alien-base-types:3.0.0
  - pycomps.ansible:1.2.0-SNAPSHOT
  - dls.ansible:1.1.0-SNAPSHOT

topology_template:
  inputs:
    debug:
      type: boolean
      required: true
      default: false
      description: "Do not redact sensible information on logs"
    target_host:
      type: string
      required: true
      description: "the remote host"
  node_templates:
    ESM_Workflow:
      metadata:
        a4c_edit_x: 5
        a4c_edit_y: "-46"
      type: pycomps.ansible.nodes.PyCOMPSJob
      properties:
        pycomps_endpoint: { get_input: target_host }
        compss_module_version: eflows4hpc
        num_nodes: 3
        qos: debug
        input_data_path: "/home/bsc32/bsc32044/fesom_ensemble_alien4cloud/input"
        output_data_path: "/home/bsc32/bsc32044/fesom_ensemble_alien4cloud/output"
        command: "/home/bsc32/bsc32044/fesom_ensemble_alien4cloud/esm_simulation_v5.py"
        arguments:
          - a100001
        container_image: ""
        container_compss_path: ""
        container_opts: ""
        python_interpreter: python3
        extra_compss_opts: "--qos=debug --exec_time=120 --keep_workingdir --worker_working_dir=/hom
4cloud:/apps/HECUBA/1.2/compss --env_script=/home/bsc32/bsc32044/fesom_ensemble_alien4cloud/set_es
.2/compss"
  workflows:
    exec_job:
```

- Modularize each component of a workflow and make it composable: enables to have a clean, versionable, reusable workflows.
- Make the components interact and with job scheduler, enables modern, efficient workflows. – PyCOMPSS

- Abstract machine topology, building software dependencies - containerization enables workflows that are portable across HPCs - HPCWaaS.

Interest at AWI: A CMIP7 use case - Coupled HR simulations

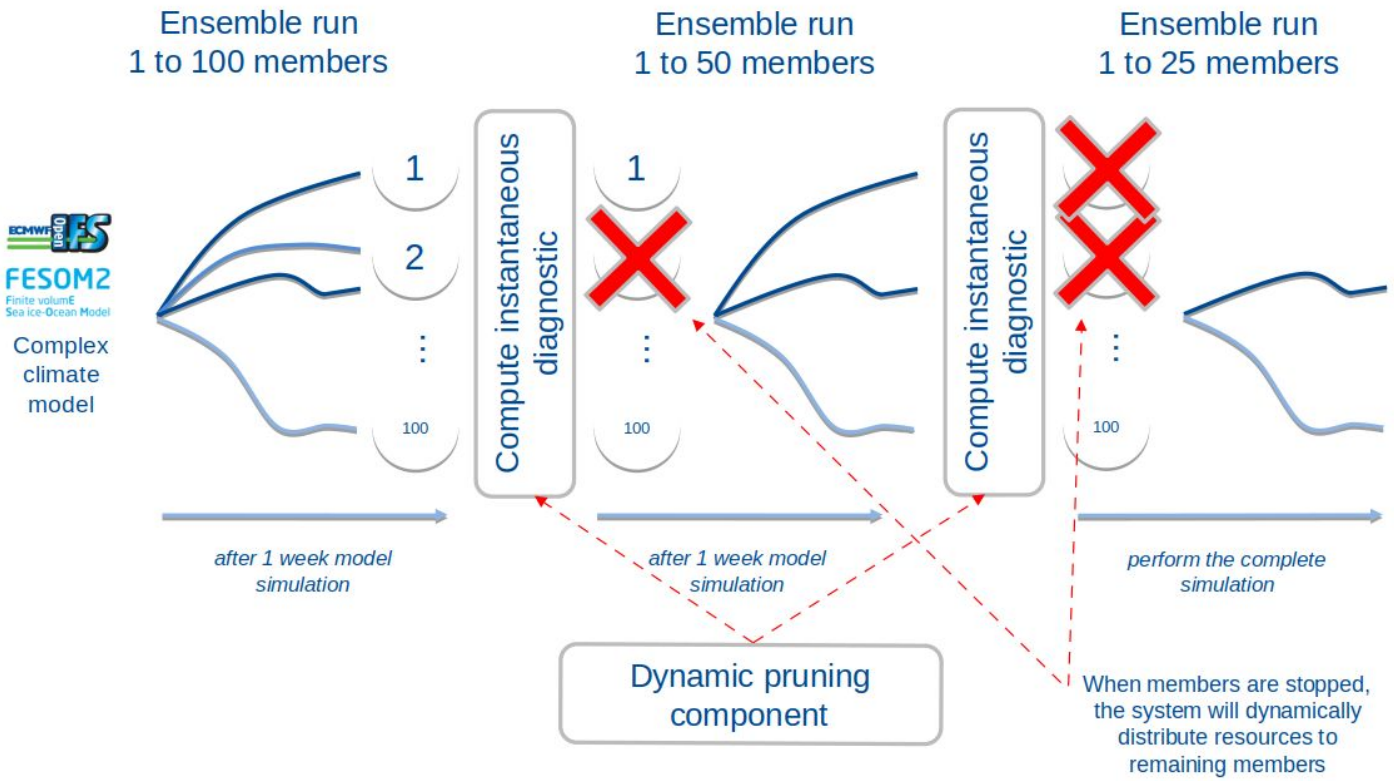
Interest at AWI: A CMIP7 use case - Coupled HR simulations

- **Long running coupled simulations are needed to spin-up the coupled model.**
  - Initially coupled models may “blowup”. An efficient dynamic workflow can be used to enable output only around the time of blowup for diagnosis, significantly reducing storage costs.

Interest at AWI: A CMIP7 use case - Coupled HR simulations

- **Long running coupled simulations are needed to spin-up the coupled model.**
  - Initially coupled models may “blowup”. An efficient dynamic workflow can be used to enable output only around the time of blowup for diagnosis, significantly reducing storage costs.
- **Tune the model for uncertain parameters for a reasonable climate.**
  - Every climate model needs tuning. It involves many simulation experiments and optimizing parameters. Most of them may be discarded (unrelated to blowup) early on. A dynamic workflow can save significant computational costs.

ESM Dynamic Workflow: example

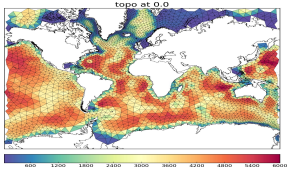


Interest at AWI: A CMIP7 use case - Coupled HR simulations

- **Long running coupled simulations are needed to spin-up the coupled model.**
  - Initially coupled models may “blowup”. An efficient dynamic workflow can be used to enable output only around the time of blowup for diagnosis, significantly reducing storage costs.
- **Tune the model for uncertain parameters for a reasonable climate.**
  - Every climate model needs tuning. It involves many simulation experiments and optimizing parameters. Most of them may be discarded (unrelated to blowup) early on. A dynamic workflow can save significant computational costs.
- **Perform multiple simulations according to protocol with necessary output.**
  - A workflow can be used to perform simulations that involve restarts.
  - A portable WaaS can be used to run simulations across multiple HPC centers.

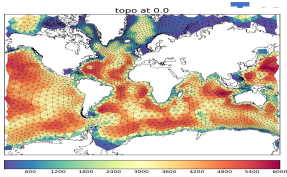
Interest at AWI: A CMIP7 use case - Coupled HR simulations

- **Long running coupled simulations are needed to spin-up the coupled model.**
  - Initially coupled models may “blowup”. An efficient dynamic workflow can be used to enable output only around the time of blowup for diagnosis, significantly reducing storage costs.
- **Tune the model for uncertain parameters for a reasonable climate.**
  - Every climate model needs tuning. It involves many simulation experiments and optimizing parameters. Most of them may be discarded (unrelated to blowup) early on. A dynamic workflow can save significant computational costs.
- **Perform multiple simulations according to protocol with necessary output.**
  - A workflow can be used to perform simulations that involve restarts.
  - A portable WaaS can be used to run simulations across multiple HPC centers.
- **Additional outputs to pursue research on specific interests.**
  - A dynamic workflow can be used to enable HR outputs at regions of interest, that are diagnosed effectively in Python.



Interest at AWI: A CMIP7 use case - Coupled HR simulations

- **Long running coupled simulations are needed to spin-up the coupled model.**
  - Initially coupled models may “blowup”. An efficient dynamic workflow can be used to enable output only around the time of blowup for diagnosis, significantly reducing storage costs.
- **Tune the model for uncertain parameters for a reasonable climate.**
  - Every climate model needs tuning. It involves many simulation experiments and optimizing parameters. Most of them may be discarded (unrelated to blowup) early on. A dynamic workflow can save significant computational costs.
- **Perform multiple simulations according to protocol with necessary output.**
  - A workflow can be used to perform simulations that involve restarts.
  - A portable WaaS can be used to run simulations across multiple HPC centers.
- **Additional outputs to pursue research on specific interests.**
  - A dynamic workflow can be used to enable HR outputs at regions of interest, that are diagnosed effectively in Python.
- **Transform the results and publish (output).**
  - A workflow can be used to standadize (cmor-ize) the output and regrid the data.
  - Automatically publish the data to data centers (ESGF or cloud).



Dynamic ESM Workflows: HECUBA

- **To enable efficient dynamic workflows that involve data, a database solution that can be used across workflow components is necessary.**
  - Choice of database depends on application.
    - Cassandra: A key-value store, NOSQL, distributed, fault-tolerant database is suitable for climate simulations in general.
  - HECUBA provides with “scientific” data API for Python, C++, C (and Fortran using an interface to CPP/C) API’s to interact with Cassandra.
    - HECUBA can be used to enable consistent interaction among components of ESM-workflows.
    - HECUBA provides asynchronous API interaction with Cassandra.
    - Having a Python API, is extremely useful for ML based workflows.

Sharing Data Models with HECUBA

- **If we want to share data among components consistently, they need to use a consistent data model. For instance to create a binary blob:**

Python: Numpy

```python
import numpy as np
import hecuba # connects to cassandra
var_data = np.arange(1.0,100.0)
storage_obj = hecuba.StorageNumpy(var_data)
storage_obj.make_persistent('variable1') # data is sent to Cassandra
```

CPP

```cpp
using StrKeyClass = KeyClass<std::string>;
using MyValueClass = ValueClass<StorageNumpy>;
class metricsDict:public StorageDict <StrKeyClass, MyValueClass>{};
StorageNumpy metrics_data(data,metadata); // instantiates a StorageNumpy with the specified info
metricsDict metrics;
StrKeyClass k = StrKeyClass("variable1");
MyValueClass v(metrics_data);
metrics[k]=v;                          //store asynchronously and sends data through the stream
```

Fortran: with C binding

```fortran
    subroutine write_array(varname, values, arr_size) &
                bind(c,name="hecuba_write_binary_array")
            use iso_c_binding, only: c_char, c_int, c_ptr, c_float, c_double
            character(kind=c_char) :: varname(*);
            real(kind=c_double) :: valueC(*);
            integer(kind=c_int) :: arr_size;
    end subroutine write_array
    call write_array("variable1", data, data_size)
```

- **Retrieve in Python**

```python
import numpy as np
import hecuba # connects to cassandra
hecuba.StorageNumpy.get_by_alias('variable1')
```

Similarly in CPP, Fortran

Composable data models with HECUBA

- **HECUBA supports variety of data types that are commonly used for scientific data: primary data types (ints, floats, strings, bools), dictionaries, binary blobs (numpy). These can also be nested to make complex data types.**

Simplified NetCDF data model

```
import numpy as np
from hecuba import StorageDict
from hecuba.storageobj import StorageObj

class Coordinates(StorageDict):
    '''
       @TypeSpec dict<<key:str>,values:ndarray>
    '''
    pass
class Coordinatevars(StorageDict):
    '''
    @TypeSpec dict<<var_name:str>,values:models.DataArray>
    '''
    pass
class DataArray(StorageObj):
    '''
    @ClassField dims list<str>
    @ClassField coords models.Coordinates
    @ClassField values ndarray
    '''
    pass
class Datavars(StorageDict):
    '''
       @TypeSpec dict<<var_name:str>,values:models.DataArray>
    '''
    pass
class Dataset(StorageObj):
    '''
    @ClassField dims list<str>
    @ClassField coords models.Coordinates
    @ClassField data_vars models.Datavars
    '''
    pass
~
~
```

- Initial implementation was based on NetCDF data model. Good for analysis, but not suitable for consistent performance comparison with fileIO and cloud storage.
- Parallel IO was more desirable.
- Initial implementation with HECUBA in FESOM2 was slower then using fileIO.

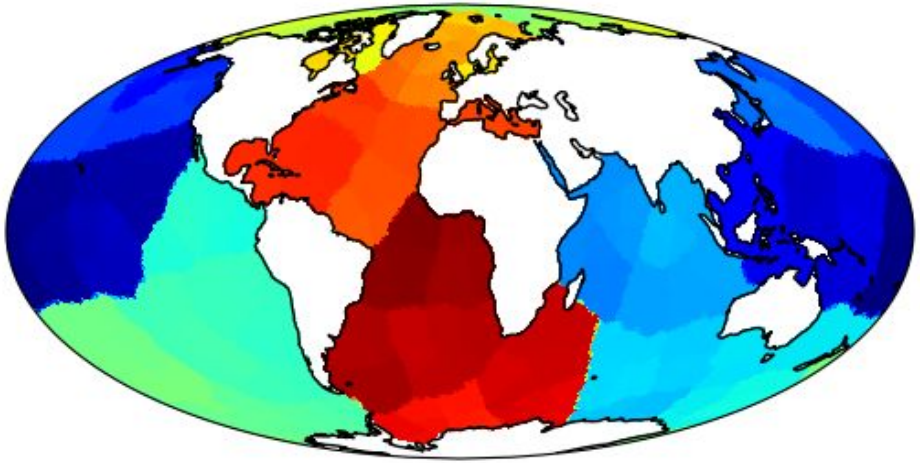
- Led to Improvements to HECUBA, simplified API in CPP.
- YAML is made the standard for data models.

Dynamic ESM workflows

- **Zarr data model is more suitable for parallel IO with little configuration on HPC.**

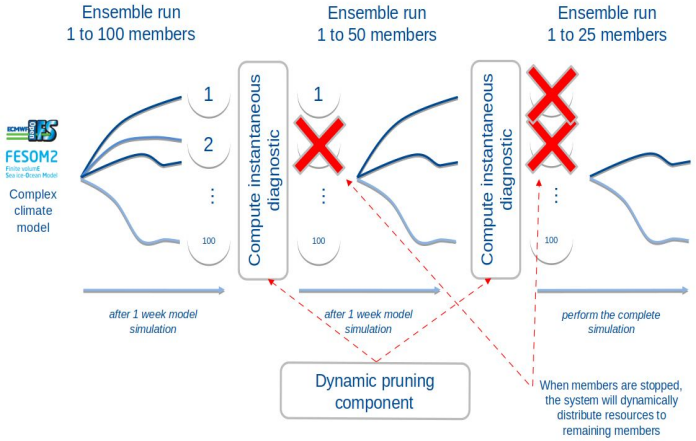
Simplified Zarr datamodel

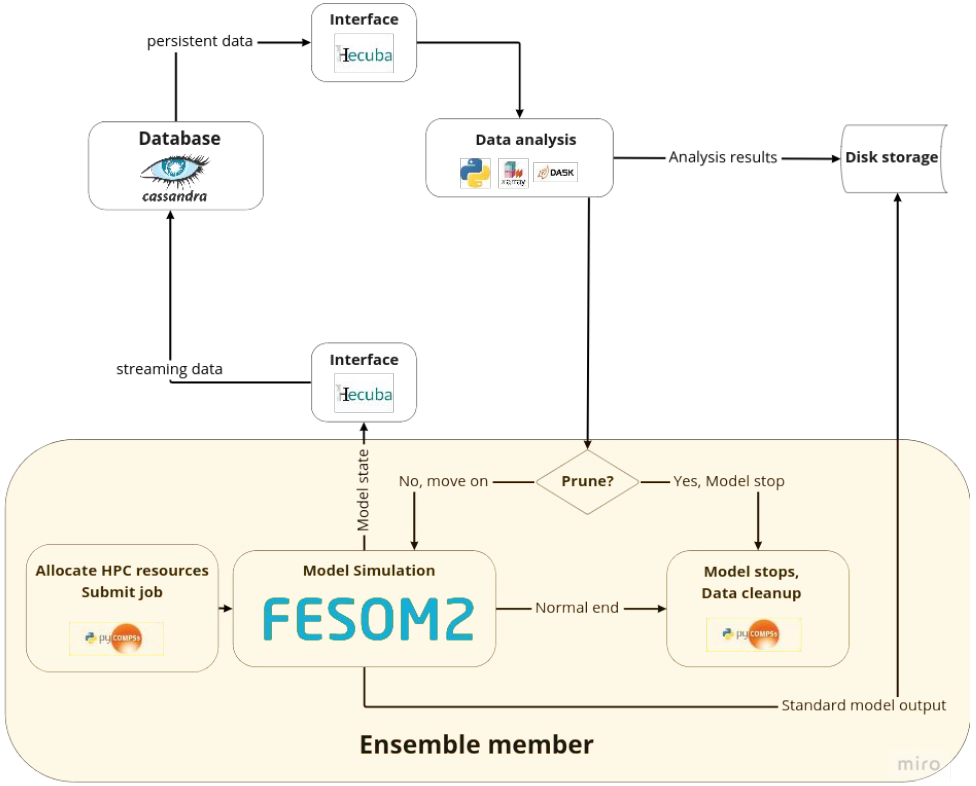
```
"TypeSpec": [ "midict", "StorageDict" ]
"KeySpec":
          - [ "time", "float"]     #1st is partiion key
          - [ "chunk", "int" ]       #Remaining are the clustering
"ValueSpec":
          - [ "metrics", "numpy.ndarray" ]
#
```



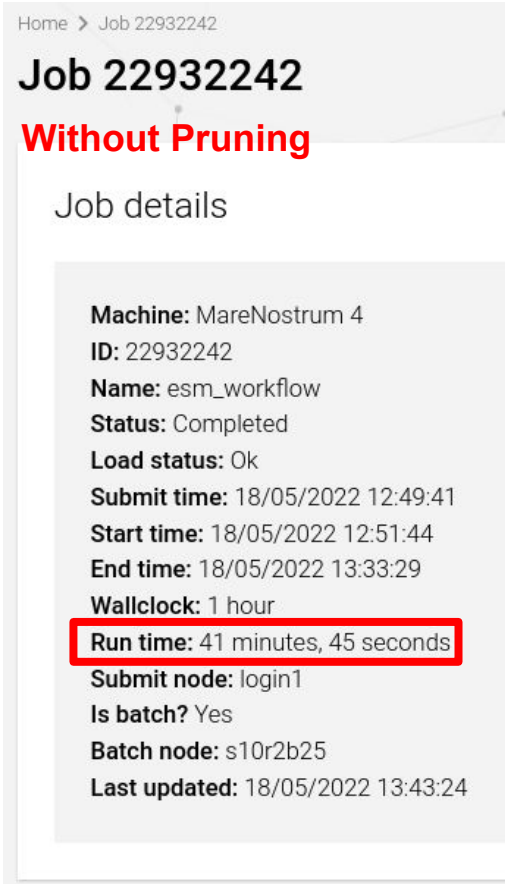
- Suitable for parallel IO.
- IO performance is independent of filesystem striping.
- Consistent data model comparison across backends: fileIO, database, cloud storage.

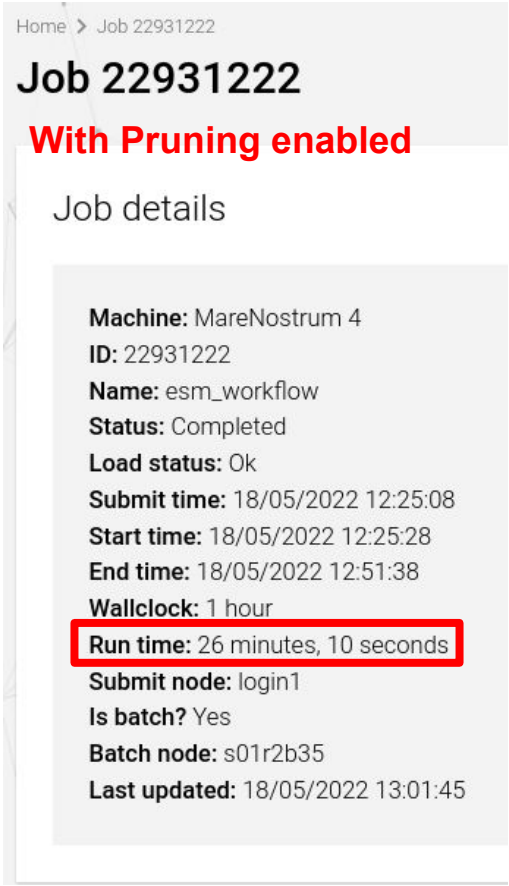
ESM Dynamic Workflow - Model tuning





ESM Dynamic Workflow - Pruning

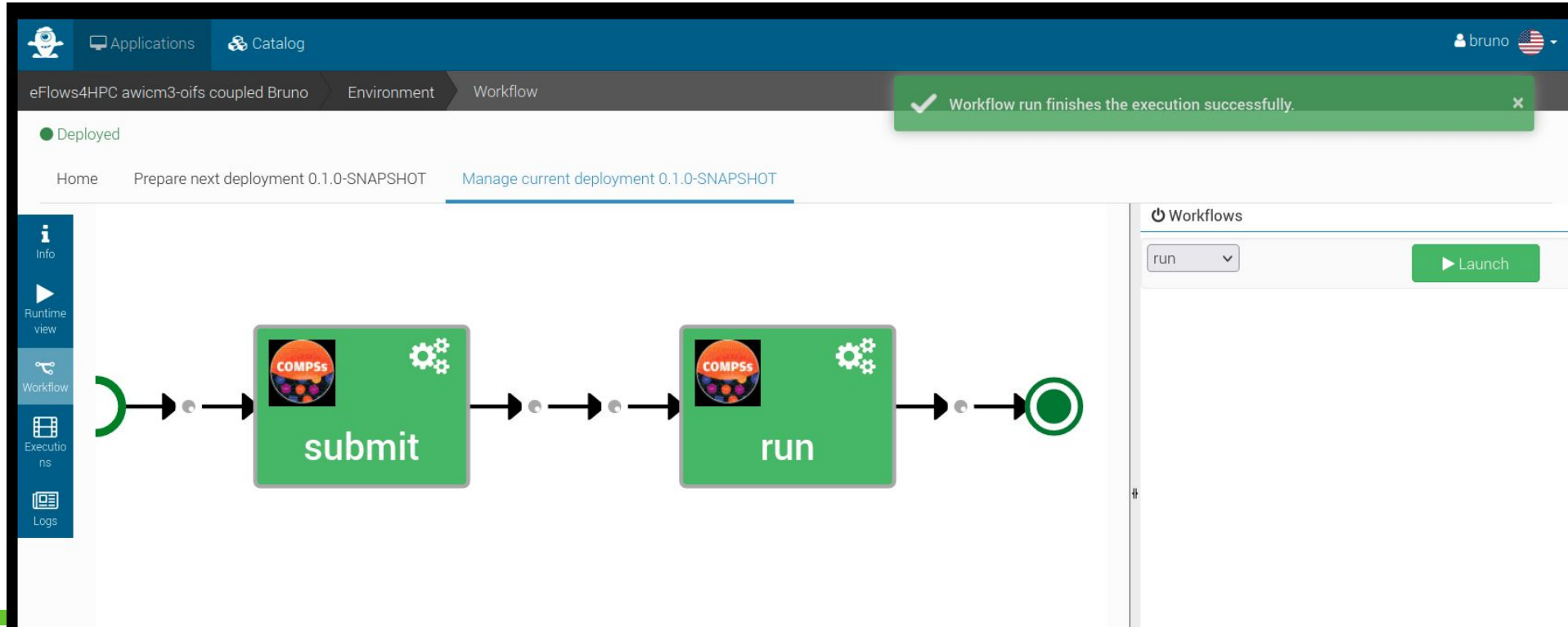




- Small tests conducted to check how is the behavior with and without pruning enabled in a basic and small **ensemble experiment consisting of 3 simulation members**
- We simulated a prune of 2 of its members with mocked analysis data at the very beginning (first chunks)
- Also the pruned members get all its generated data deleted from the output directories to save storage space
- * currently broken on MN4

ESM Dynamic Workflow - HPCWaaS

- Alien4Cloud Interface



Final remarks:

- **Workflows for climate simulation/analysis are good for you.**
  - Composable workflows: reusable, clean and versionable.
  - Dynamical workflows: Efficient and new applications. (Pycomps 👍) https://compss-doc.readthedocs.io
  - WaaS : Portable across HPC, efficient.

Final remarks:

- **Workflows for climate simulation/analysis are good for you.**
  - Composable workflows: reusable, clean and versionable.
  - Dynamical workflows: Efficient and new applications. (Pycomps 👍) https://compss-doc.readthedocs.io
  - WaaS : Portable across HPC, efficient.

- **Future work at AWI (with BSC)**
  - Benchmark HECUBA.
  - Model tuning for CMIP7 using Workflows.
  - Enable ML workflows.
  - Abstract IO backends and develop/use IO server.