HPC/Exascale Centre of Excellence in Personalised Medicine

D2.2 Midterm code release

Version 1.0

Contract Number	951773
Project Website	http://www.permedcoe.eu/
Contractual Deadline	M18, March 2022
Dissemination Level	PU
Nature	R
Author(s)	Jesse Harrison (CSC), Javier Conejero (BSC), Jesús Gorroñogoitia (ATOS), Pablo Rodríguez-Mier (UKHD)
Contributor(s)	-
Reviewer(s)	Miguel Vazquez (BSC), Sampo Sillanpää (CSC)
Keywords	Infrastructure tooling, building blocks, workflows, machine learning

Notice: *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No "951773".*

Version	Author	Date	Description of Change
V0.1	Jesse Harrison	01 March 2022	Initial draft
V0.2	Jesse Harrison	18 March 2022	Version incorporating feedback from reviewers
V1.0	Jesse Harrison	30 March 2022	Minor changes to repository access details
			(Final Change Log entries reserved for releases to the EC)

Table of contents

1. Executive Summary

This Deliverable presents an overview of PerMedCoE activities during Year 1 in relation to solutions for infrastructure tooling (including software containerisation, building block and workflow development), adherence of building blocks and workflows to best software practices identified in Deliverable 1.1, and available user documentation including a catalogue of existing building blocks and workflows. Further, the Deliverable discusses steps taken to explore machine learning methods as part of PerMedCoE during the first half of the project, and presents updated general design choice recommendations for building block and workflow development.

2. Introduction

Tasks undertaken within PerMedCoE Work Package 2 involve packaging HPC-optimised and containerised core software tools as building blocks designed to perform specific functionalities. By providing end-users with a standardised interface to execute individual building blocks and combine building blocks into workflows, it is possible to address diverse use cases of interest to researchers and practitioners within the field of personalised medicine. The optimisation and containerisation of core software tools take place as part of PerMedCoE Work Package 1, with example use cases addressed by different building blocks and workflows defined in Work Package 3. Further tasks addressed by Work Package 2 include formulating design choices for building blocks and the integration of core software tools as part of use-case workflows, improving the scalability and interoperability of workflows over different HPC platforms, and providing tools to improve the usability of and access to the PerMedCoE infrastructure.

In support of Work Package 2 activities during Year 1 of the PerMedCoE project, the present Deliverable provides:

- Summaries of infrastructure tooling solutions to date
- Details on the adherence of PerMedCoE building blocks and workflows to the software best-practice guidelines described in Deliverable 1.1 [1]
- A current catalogue of building blocks and use-case workflows with links to user documentation and development code
- A synthesis of recommendations for the construction and execution of building blocks and workflows (Annex I)

As part of the midterm code release and in support of information provided in the Deliverable report, general user documentation has been made available on the PerMedCoE readthedocs website:

https://permedcoe.readthedocs.io

Detailed user documentation and code specific to building blocks and use-case workflows are provided under the PerMedCoE GitHub organisation:

https://github.com/PerMedCoE

3. General infrastructure tooling for building blocks and workflows

This section summarises the infrastructure tooling solutions designed and employed as part of the building block and workflow activities during Year 1 of the PerMedCoE project. Wider documentation on building block design choices is available in Annex I of the Deliverable.

3.1. Software containerisation and access to core tool image files

Following design choices discussed in Section A2 of Annex I, existing PerMedCoE building blocks (Section 4.1) use Singularity containers for core software execution. Note that Singularity has recently been renamed to Apptainer (https://apptainer.org). For clarity, we still use the term Singularity in this document. Singularity definition files provide a transparent and reproducible format for documenting software installation and configuration steps, with public access to Singularity definition files for PerMedCoE core software being described in Deliverable D1.2 [2]. A survey of different HPC platforms involved in PerMedCoE, undertaken as part of Work Package 2 activities, has confirmed the availability of Singularity on most PerMedCoE-affiliated host systems (Section A2.2.3 of Annex I).

Access to pre-built core Singularity images during Year 1 of the PerMedCoE project has been limited to PerMedCoE tool developers, with image files hosted on the BSC B2DROP service that is restricted to PerMedCoE partners (b2drop.bsc.es). During Year 2, public access to image files on GitHub will be supported via ORAS (https://oras.land/cli). ORAS-enabled access to Singularity images is already supported by the COBREXA core software, with user documentation on pre-built images provided in the COBREXA GitHub repository:

https://github.com/LCSB-BioCore/COBREXA.jl#prebuilt-images

3.2. Building block development

General design choices for PerMedCoE building block development are outlined in Section A2 of Annex I. Further to using Singularity for core software containerisation (Section 3.1), implementing these choices in practice has involved the development and deployment of a Python API for constructing building block and workflow templates (Section 3.2.1), along with a unified command line interface for building block and workflow execution (Section 3.2.2). Public access to existing PerMedCoE building blocks is outlined in Section 4.1 of this Deliverable.

3.2.1. Python API for constructing building block and workflow templates

The ***permedcoe*** package provides the Python API necessary for the development of building blocks within the PerMedCoE scope. It also provides a command line tool that eases the creation of building blocks and workflows with templates. This package is publicly available via the Pypi repository (https://pypi.org/project/permedcoe) enabling its easy installation with pip, and in GitHub (https://github.com/PerMedCoE/permedcoe).

The Python API provided by the ***permedcoe*** package comprises a set of decorators, parameter type definitions and functions to be used in the building block implementation (https://permedcoe.readthedocs.io/en/latest/02_components/components.html#python-api):

- The decorators enable one to define the functionality of the building block and hide the container and execution complexities
- The parameter type definitions enable to declare the type of the building block input and output parameters (e.g. files or directories)
- The functions provide helper functionalities to improve its operation

The available decorators are: *@task*, *@mpi*, *@binary*, *@constraint* and *@container*. The @task decorator is used to specify that a function should be considered as a unit of work and serves as the placeholder for parameter type definitions. A building block usually comprises a single unit of work, but can be composed of more than one. The @mpi or the @binary decorator can be placed over the @task decorator to define that the unit of work is a MPI application or a binary application, and is the placeholder for a particular binary. The @container decorator is used to define the container where the unit of work will be executed. On top of all these decorators, @constraint can be used to define specific requirements for the binary/MPI/task. This decorator is specific to the PyCOMPSs workflow manager and is used to specify the task requirements on HPC resources, such as the number of cores or amount of memory. Other workflow managers specify this using their own mechanisms when calling the building blocks.

The ***permedcoe*** base package also provides a command line interface which is able to provide building block and workflow templates. User documentation for the command line interface is available via the PerMedCoE readthedocs website:

https://permedcoe.readthedocs.io/en/latest/02_components/components.html#template-creation

More specifically, the building block template is the skeleton of a blank building block with the structure of a Python package. It is ready to be deployed and used (with dummy code including all decorators and parameter type definitions), so that the only requirement from the building block developer is to complete its main file with the desired functionality. This skeleton and building block structure are described in detail in the documentation:

https://permedcoe.readthedocs.io/en/latest/04_creating/01_building_block_templates/building_block_templates.html

The application template additionally represents a blank application for a given workflow manager (currently for PyCOMPSs, Snakemake and NextFlow templates, which are the workflow managers considered within the scope of PerMedCoE), so that the workflow developer can start importing and using the desired building blocks. This either takes place from the command line interface using the Snakemake or Nextflow managers, or using the PyCOMPSs Python interface. The relevant documentation is provided in:

https://permedcoe.readthedocs.io/en/latest/04_creating/02_application_templates/application_templates.html

Finally, the entire permedcoe package usage has been condensed in a step-by-step tutorial that includes information ranging from building block creation to the workflow execution:

https://permedcoe.readthedocs.io/en/latest/04_creating/04_tutorial/tutorial.html

3.2.2. Unified command line interface for building block and workflow execution

A unified command line interface for building block execution has been implemented as part of the ***permedcoe*** base package, which is automatically exposed by all building blocks. This interface has been established following guidelines described in Annex I (Section A2.1.5). Consequently, a building block can be directly invoked from the command line after its deployment with the building block name.

The ***permedcoe*** base package command line interface is further able to invoke building blocks respecting the same interface as workflow applications:

https://permedcoe.readthedocs.io/en/latest/02_components/components.html#building-block-execution

https://permedcoe.readthedocs.io/en/latest/02_components/components.html#application-execution

3.3 Exploration of machine learning methods

3.3.1 Tasks targeted for machine learning toolbox development

Developing a machine learning toolbox for the PerMedCoE project is one of the tasks of Work Package 2 (Task 2.1). Analytical tools and use case workflows have been scoped to identify opportunities for using machine learning to improve their HPC compatibility. Tasks and tools that have been identified as potential targets for machine learning toolbox development during Years 2-3 of the PerMedCoE project are outlined in Table 1.

Analytical task	**Examples of AI/ML software or libraries**	**Potential PerMedCoE use cases**
Dimensionality reduction (using e.g. unsupervised ML)	Scvis (Ding et al. 2018 [3]); Python and/or R, UMAP	All use cases
Approximate solution for non-linear metabolic systems that model thermodynamics of reactions	COBREXA (specialised solver relying on ML methods, under development)	1, 4
Model trajectory analyses	DEAP (https://deap.readthedocs.io)	4
Use of surrogate models	PhysiCell	4, 5
Model parameter fitting	DEAP and general hyperparameter optimisation frameworks, e.g. ray tune (https://docs.ray.io/en/latest/tune/index.html), hyperopt (http://hyperopt.github.io/hyperopt) and optuna (https://optuna.org)	4, 5
Classification tasks (e.g. classifying metabolic fluxes to uncover possible cell phenotypes)	TensorFlow (https://www.tensorflow.org), Keras (https://keras.io), PyTorch (https://pytorch.org)	1, 2
Diverse unsupervised and supervised ML tasks for PyCOMPSs workflows	dislib (https://github.com/bsc-wdc/dislib)	All use cases
Matrix factorisation	TensorFlow, Macau (https://macau.readthedocs.io/en/latest/index.html), JAX (https://jax.readthedocs.io/en/latest)	2 (see Section 3.3.2 for details)

Table 1. Analysis tasks and their relation to AI/ML software or libraries with potential applications in PerMedCoE use cases.

3.3.2. Exploration of ML methods for analyses using CellNOpt / CARNIVAL

To date, the application of machine learning techniques has primarily been explored with reference to using CellNOpt / CARNIVAL as part of PerMedCoE workflows. The refactoring and/or extensions of CellNopt / CARNIVAL focus on the fast and scalable generation of mechanistic models, from which functional features (e.g active or inactive proteins, pathways or interactions) can be extracted from gene expression data or (phospho)proteomics data. However, one important issue that needs to be addressed is the translation of the output of these models to a relevant outcome of interest.

This issue could be addressed by adding a ML step for learning a mapping function from CellNOpt / CARNIVAL features to the response of interest. For this purpose, the CellNOpt / CARNIVAL development team has evaluated the use of different ML techniques and libraries to expand the capabilities of this software, focusing on three main aspects: 1) scalability, 2) interpretability and 3) simplicity (lightweight, minimal dependencies).

For preliminary evaluation, different systems were implemented based on probabilistic matrix factorisation (using Macau), neural networks (using TensorFlow), and matrix factorisation with gradient descent (using JAX). Macau [4] is a probabilistic matrix factorisation method with side features, which is able to predict particular responses with missing data (e.g. some drugs are only tested in some cells) and also incorporating side information (e.g. drug and cell features) in order to make predictions for unobserved drugs and cell lines. Although the performance is very good in this setting, current implementation requires the use of a Gibbs sampler for sampling the posterior distribution, which is slow, does not scale well and does not support the use of GPUs. Another line of investigation has involved using Tensorflow with Keras to build different neural network architectures. One issue with neural networks is that the typical high-dimensional nonlinear functions that are learnt are good for prediction but complex for interpretation. Although these libraries can also be used to learn linear models, the code needed to replicate some basic matrix factorisation and training with gradient descent is non-trivial. Further, the libraries are heavy in terms of dependencies.

JAX was selected for further exploration of ML methodologies, since it met the specific requirements set by PerMedCoE (see Table 1). Benefits of JAX in this context include:

- Use of XLA (a domain-specific compiler for linear algebra) to compile NumPy programs on GPUs.
- Easy-to-define linear models with NumPy-like syntax
- Auto-differentiation of NumPy operations and some Python syntax
- Relatively lightweight (e.g. in comparison with TensorFlow)

Using JAX, four different linear models of matrix factorisation for prediction of drug responses on cell lines have been implemented (with this topic being of relevance to PerMedCoE Use Case 2). The approach is similar to the Macau method but using a deterministic method instead of a probabilistic one for better scalability.

Matrix factorisation for IC_{50} imputation

Assuming drug responses have only been measured for certain cell lines, this problem is basically an imputation problem (predicting the missing drug responses). Given a matrix of $\log(IC_{50})$ values where rows are drugs and cells are columns, this matrix can be factorised as D^TC where D and C are two low-rank matrices. Each $\log(IC_{50})$ value is written as a linear combination of two latent vectors, one for drugs and one for cells. The model can be extended to add a bias term for drugs (a column vector) and a bias for cell lines (a row vector):

$\log(IC_{50}) = D^TC + \text{bias\_drugs} + \text{bias\_cells}$

This is implemented in JAX in the following manner:

```
@jax.jit
def mf(params):
      LD, LC, ld_bias, lc_bias, mu = params
      Dt = jnp.transpose(jnp.add(LD, ld_bias))
      C = jnp.add(LC, lc_bias)
      return jnp.dot(Dt, C) + mu
```

Matrix factorisation using cell features

A goal of using machine learning methods as part of Use Case 2 is to employ features from both drugs and cells. Using CARNIVAL / CellNopt, it is possible to identify interesting features concerning altered signalling pathways for cell lines. In order to incorporate this information, one can further decompose the latent cell matrix C, so that each cell latent vector is a function of the cell features. This can be written as:

$C = L_C * C_F^T + \text{bias\_cells}$

Where L_C is the latent matrix to be estimated and C_F is the matrix with cell features:

```
@jax.jit
def mf_with_col_features(params, col_features):
      LD, LC, ld_bias, lc_bias, mu = params
      Dt = jnp.transpose(jnp.add(LD, ld_bias))
      C = jnp.add(jnp.dot(LC, jnp.transpose(col_features)), lc_bias)
      return jnp.dot(Dt, C) + mu
```

Matrix factorisation using drug features

Something similar can be done in instances where drug features (e.g target genes of the drugs) are available, but cell features are unavailable:

```
@jax.jit
def mf_with_row_features(params, row_features):
    LD, LC, ld_bias, lc_bias, mu = params
    D = jnp.add(jnp.dot(LD, jnp.transpose(row_features)), ld_bias)
    Dt = jnp.transpose(D)
    C = jnp.add(LC, lc_bias)
    return jnp.dot(Dt, C) + mu
```

Matrix factorisation using both cell and drug features

If both drug and cell features are available, it is possible to extend the model as follows:

```
@jax.jit
def mf_with_features(params, row_features, col_features):
    LD, LC, ld_bias, lc_bias, mu = params
    Dt = jnp.transpose(jnp.add(jnp.dot(LD,
         jnp.transpose(row_features)), ld_bias))
    C = jnp.add(jnp.dot(LC, jnp.transpose(col_features)), lc_bias)
    return jnp.dot(Dt, C) + mu
```

After defining the different models, latent matrices can be estimated from the data, using JAX autodiff capabilities to differentiate through the model given a loss function (RMSE) between predicted log(IC50) values and observed ones. More details about how the training is done are available in the following notebook:

https://colab.research.google.com/drive/134kuHCpJ3kOTDHyDMCXhYq1XKYOhhN9h#scrollTo=MfL75o7cXaAz

A test container for exploiting this model is available via:

https://github.com/saezlab/permedcoe/tree/master/containers/ml-jax

Further, a building block was developed for using this approach together with other developed building blocks (e.g CARNIVAL and CellNopt), which is available at:

https://github.com/saezlab/permedcoe/tree/master/building-blocks/core/src/ml_jax_drug_prediction

The building block / Singularity container includes a way to test the performance of the method with different settings using GDSC data, explained here:

https://github.com/saezlab/permedcoe/tree/master/building-blocks/core

3.4 Adherence to software best practices

According to the software best-practice guidelines described in Deliverable D1.1 [1], software and code produced by PerMedCoE must, among other recommendations, adhere to FAIR (Findable, Accessible, Interoperable, Reusable) principles. Adherence of the PerMedCoE core software tools to these guidelines has been assessed based on a follow-up questionnaire circulated among core tool developers, with summaries of the questionnaire results provided in Deliverable D1.2 [2].

Actions to ensure the adherence of PerMedCoE building blocks and workflows to FAIR principles during Year 1 are summarised in Table 2. Further steps to ensure the compatibility of PerMedCoE building blocks and workflows with multiple HPC infrastructures are outlined in Section A2 of Annex I. To date, building blocks and workflows have been deployed on two platforms (BSC MareNostrum 4 and CSC Mahti), with several development results presented in Deliverable D2.1 [5]. Further cross-platform compatibility testing and follow-up work on ensuring adherence to software best practices is planned for Years 2-3 of the project (Section 5).

The underlying motivation for the building block and workflow designs implemented in PerMedCoE is to provide solutions that are:

- Portable and cross-compatible at multiple levels of organisation (ranging from the level of individual tools and building blocks to workflows and, ultimately, HPC platforms hosted by PerMedCoE partners)
- Expert-validated and error-free
- Composable and modular (facilitating the easy use and deployment of building blocks as part of complex workflows)

To date, validation of the existing PerMedCoE building blocks has primarily focused on those building blocks used in Use Case 5. Further validation work in relation to other use cases is currently in progress, with Use Case 2 workflows already having been successfully deployed on MareNostrum 4.

FAIR principle	**Actions concerning building blocks and workflows during Year 1**
F (Findable)	Provision of a general website for PerMedCoE user documentation Catalogue of building blocks and workflows in the general PerMedCoE user documentation Establishing linkages between user documentation and GitHub repositories for building blocks and workflows Workflows for end users’ consumption searchable in the Croupier frontend (marketplace)

A (Accessible)	Open licensing of building blocks and workflows (whenever possible depending on the tools used within the building blocks and building blocks used in the workflows) Usage of GitHub repositories for source code storage and distribution Development of a common CLI for building block and workflow deployment, and template creation Preparation of end-user documentation for building blocks and workflows Workflow execution for end users in the Croupier frontend
I (Interoperable)	Usage of Singularity (Apptainer) containers for core software tool packaging in building blocks Compatibility of core software building blocks with auxiliary applications (see definition of auxiliary applications in Section 2.1.3 of Annex I) and between PerMedCoE use cases Establishing support for multiple workflow managers (PyCOMPSs and SnakeMake; initial development of support for NextFlow) Development work to establish solutions for cross-infrastructure workflow deployments using Croupier
R (Reusable)	Open access to building block and workflow code under the PerMedCoE GitHub organisation (For details on access to core software-associated code, see Deliverable D1.2 [2]) Multiple (based on different inputs) workflow execution from Croupier’s frontend

Table 2. Steps taken during Year 1 of the PerMedCoE project to ensure adherence of building blocks and workflows to FAIR principles.

3.5 Croupier implementation

Croupier is a meta-orchestrator or workflow manager (implemented as a plugin of the Cloudify Cloud workflow manager[1]) that can distribute an application’s work tasks and data across heterogeneous infrastructures, and in particular across HPC clusters for execution (see Deliverable D2.1 [5]). Croupier enables *i*) application providers to register their applications as workflows into the Croupier marketplace, and *ii*) application consumers to browse existing applications, and execute them in target HPC clusters with the input data they supply. See the Application Execution with

1 https://cloudify.co/

Croupier section in PerMedCoE online documentation for additional information about Croupier's supported roles and their procedures:

https://permedcoe.readthedocs.io/en/latest/04_creating/03_croupier/croupier.html

Note: in the following, by application we mean any Croupier workflow whose tasks are executed by PyCOMPSs or Snakemake in the backend HPC infrastructures.

Since Deliverable D2.1 [5], some of Croupier's features have been improved, and others incorporated. Croupier's code for PerMedCoE is available at the Croupier GitHub repository:

https://github.com/ari-apc-lab/croupier/tree/permedcoe

The **data management support** for data transfer across infrastructures has been redesigned and reimplemented, aiming for flexibility. Croupier supports HTTP transfer between Cloud and HPC data sources and RSYNC transfer among HPC clusters. Croupier also acts as a data proxy between those HPC clusters with external Internet access disabled.

The Identity Access Management (IAM) has been improved. Croupier supports declaring multiple Vault servers in workflows and uses the same mechanism to retrieve access credentials for HPC clusters and data sources.

A **complete Croupier framework**, including KeyCloak for Single Sign On (SSO) user's authentication, Vault for secrets management, Cloudify/Croupier for workflow execution, and the monitoring framework based on Prometheus and Grafana, for HPC task monitoring, has been installed within a Kubernetes cluster in the Atos infrastructure and is available at http://<service>.croupier.permedcoe.eu. For instance, Cloudify/Croupier is available at:

http://cloudify.croupier.permedcoe.eu

In order to execute an application's workflow, consumers must log themselves in KeyCloak to get access to the Cloudify/Croupier frontend. Then, Croupier uses Vault instances registered in the workflow to retrieve their credentials to get access to the HPC infrastructures for task execution and data transfer, by using the token it gets from Vault, which requests the KeyCloak JWT token. This mechanism only enables Croupier to collect credentials for the logged user who has executed the workflow. The Vault token is deleted after the credentials are retrieved and they are kept only in memory for the time the workflow is being executed, reinforcing the credentials' safekeeping.

Croupier can be used by PerMedCoE users to run applications from its web frontend. Therefore, Croupier needs to be integrated with the PerMedCoE workflow managers that are being run in the HPC clusters, namely PyCOMPSs and SnakeMake to distribute an application's tasks.

Croupier has been integrated with PyCOMPSs by using remote ssh access to its CLI. For more information, see the PyCOMPSs usage documentation at:

https://pycompss.readthedocs.io/en/latest/Sections/08_PyCOMPSs_CLI/02_Usage.html

Croupier follows the PyCOMPSs standard procedure to:

1. Deploy an instance of each application's task in the user's workspace of its HPC cluster
2. Launch each application's task in the HPC cluster where it has been deployed
3. Monitor periodically each scheduled application's task inquiring for its execution state. This is done by using the SLURM scheduler CLI.
4. Collect monitored data about the queuing and execution timing and the resources consumed for each launched application's task, which is aggregated into a Grafana dashboard[2].

Croupier distributes an application's tasks according to the flow declared in the application's workflow, either sequentially or in parallel.

The usage of Croupier has been tested for launching workflows associated with Use Cases 5 (COVID-19) and 2 (Drug Synergies) in MareNostrum4 (BSC). Moreover, the Croupier task deployment support has been adopted to deploy both UCs in MN4 from GitHub sources. For this, the application provider has to include the application deployment script within the application blueprint artifacts. Setails on how the COVID-19 workflow is declared for Croupier using the TOSCA specification are provided in the Croupier section in the PerMedCoE online documentation.

The following code snippet (see next page) declares the main COVID-19 task:

```
job:
    type: croupier.nodes.PyCOMPSsJob
    properties:
        job_options:
            modules:
                - load singularity/3.5.2
                - use /apps/modules/modulefiles/tools/COMPSs/libraries
                - load permedcoe
            app_name: covid19
            app_source: permedcoe_apps/covid19/covid-19-workflow-
main/Workflow/PyCOMPSs/src
            env:
```

2 This feature will be integrated in the short term.

```
                    - PERMEDCOE_IMAGES: ${PERMEDCOE_IMAGES}
                    - PERMEDCOE_ASSETS: ${PERMEDCOE_ASSETS}
                    - dataset: $HOME/permedcoe_apps/covid19/covid-19-workflow-
main/Resources/data
                compss_args:
                    num_nodes: { get_input: num_nodes }
                    exec_time: { get_input: exec_time }
                    log_level: 'off
                    graph: true
                    tracing: 'false'
                    python_interpreter: python3
                    qos: debug
                app_file: '$(pwd)/covid19_pilot.py'
                app_args: { get_input: covid19_args }
            deployment:
                bootstrap: "scripts/deploy.sh"
                revert: "scripts/revert.sh"
                hpc_execution: false
        relationships:
            - type: task_managed_by_interface
              target: hpc
            - type: input
              target: data_small
            - type: output
              target: covid_results
            - type: deployment_source
              target: github_data_access_infra
```

This task, declared by the COVID-19 application provider, needs to be fed with the consumer's inputs (those obtained with the *get_inpu*t function in the task definition) for execution. The *job_options* property of the task (of type *PyCOMPSsJob*) collects the different inputs required by PyCOMPSs for task deployment and execution, including required modules, application sources, PyCOMPSs parameters and application parameters.

Plans for future work concerning the Croupier implementation for PerMedCoE are described in Section 5.

4. Available building blocks and use-case workflows

4.1 Existing building blocks

A list of available building blocks, along with general building block descriptions, is available on the PerMedCoE readthedocs website:

https://permedcoe.readthedocs.io/en/latest/03_existing/01_available_building_blocks/available_building_blocks.html

All PerMedCoE building blocks are hosted in the following GitHub repository, with sub-folders for individual building blocks:

https://github.com/PerMedCoE/BuildingBlocks

While the PerMedCoE readthedocs website provides high-level descriptions of each building block, detailed user documentation is made available in the corresponding building block folders on GitHub. Links to building block-specific folders in the PerMedCoE GitHub organisation are provided under each general building block description on readthedocs.

4.2 Existing use-case workflows

Workflows are currently available for two PerMedCoE Use Cases:

- COVID-19 multiscale modelling of the virus and patients' tissue (Use Case 5)
- Drug synergies for cancer treatment (Use Case 2)

Currently, PyCOMPSs implementations exist for both workflows, with a SnakeMake implementation having additionally been developed for Use Case 5. Information on available workflows, general workflow descriptions and lists of building blocks used by a given workflow is available on PerMedCoE readthedocs:

https://permedcoe.readthedocs.io/en/latest/03_existing/02_existing_workflows/existing_workflows.html

Further, the workflows and detailed user documentation are hosted in workflow-specific PerMedCoE GitHub repositories:

- https://github.com/PerMedCoE/covid-19-workflow
- https://github.com/PerMedCoE/drug-synergies-workflow

5. Conclusions and future tasks

The infrastructure tooling solutions and recommendations described in this Deliverable, along with development activities focusing on building block design, testing and deployment, have enabled the successful development and initial deployment of PerMedCoE workflows on MareNostrum4 (BSC) and Mahti (CSC). The building blocks associated with existing PerMedCoE workflows have also been successfully deployed on other HPC platforms for testing purposes. Activities undertaken during Year 1, including the exploration of machine learning approaches as part of Use Case 2 and work undertaken to integrate Croupier with PerMedCoE workflows, have further enabled us to target several key action points in Years 2 and 3. Further actions of relevance to this Deliverable that will be implemented during the second half of the PerMedCoE project include:

i) Consolidating and harmonising the installation and configuration steps employed by individual core software tool Singularity containers and building blocks. This will be achieved by a systematic cross-comparison of Singularity definition files and building block recipes.

ii) Further utilisation of machine learning tools as part of PerMedCoE building blocks and workflows. As part of this task, additional options for implementing GPU support as part of specific workflow components will be investigated (e.g. for rapid image analysis).

iii) Short- and medium-term feature development for the Croupier meta-orchestrator. The following features are planned for implementation by Month 24 of the PerMedCoE project:

- Assessment of work and data flow distribution (tasks, data) across HPC clusters (e.g. MN4 (BSC) and Mahti (CSC) for PerMedCoE use cases (e.g. UC2 and UC5)
- Integration with the Snakemake workflow manager
- Integration of the Croupier Web frontend
- Assessment of workflow monitoring and dashboard gathering of consumed resources for job queuing and execution prediction
- Evaluating opportunities for the monitoring of HPC cluster partitions (i.e. queues)

iv) Easier installation of and access to building blocks and ancillary files. The installation of building blocks and related files will ultimately be automated, with preliminary work into this topic having been commenced on MareNostrum 4. User-friendly access options for deploying simplified building blocks (e.g. via a web interface) will be explored during Years 2-3 within Work Package 2.

v) Improving the cross-platform portability of PerMedCoE building blocks and workflows. This could be done e.g. via employing Singularity containers using the bind model for MPI jobs (see Section A2.1.6 in Annex I).

Annex I: Roadmaps and design choices made during Year 1

A1. Introduction

This Annex details choices and recommendations concerning the delivery of PerMedCoE core and auxiliary applications as building blocks designed to meet the requirements of scientific use cases defined in Work Package 3. Guidelines are provided on the organisation of building blocks, building block configuration, workflow and user management processes, sensitive data handling, and steps taken to improve the cross-compatibility of PerMedCoE software containers on multiple HPC platforms. Design choices are also described with reference to monitoring building block performance and scalability. The Annex expands upon and provides an updated version of PerMedCoE Milestone MS07 [6].

A2. Design choices for building blocks

A2.1 Organisation of containers into building blocks

A2.1.1 Choice of container software

PerMedCoE containers rely on Singularity (to be renamed as Apptainer; https://github.com/apptainer/singularity), owing to the compatibility of Singularity containers with diverse HPC platforms. A key benefit is that using Singularity containers requires no root access on the host system, with container processes relying on user-level credentials without requiring access to a daemon. Singularity definition files provide a reproducible and modifiable format for documenting software installation and configuration steps undertaken as part of PerMedCoE (see Section A2.2.1 for information on converting between Singularity and Docker container file formats).

A2.1.2 Core software containers

PerMedCoE core software containers are built to enable use of the following applications:

- CellNOpt / CARNIVAL (signal transduction network modelling)
- Selected constraint-based modeling toolboxes (genome-scale simulation of cellular metabolism)
- MaBoSS (stochastic simulations of Boolean models)
- PhysiCell (agent-based modelling for simulating cell-cell interactions)
- New versions of the above applications (e.g. COBREXA, PhysiCell-X and PhysiBoSS)

To facilitate the use of these applications on HPC platforms, support for OpenMP threading and multi-node communication will be provided by incorporating the following features in the container image files:

- Mathematics library supporting threaded routines (e.g. Intel® oneAPI MKL)
- Message Passing Interface (MPI) and associated libraries required by multi-node job submissions

Due to the limited compatibility between different MPI implementations, it is recommended that separate core software containers be built to meet demands imposed by the MPI configurations present on different host environments (see Section A2.2.3 for information on MPI alternatives). In addition, for application-specific information on software dependencies and scaling-up for HPC use, see Section A2.2.2, Deliverables D1.1 [1] and D1.2 [2], and Milestone MS05 [7].

A2.1.3 Other containers

Where required, additional containers will be built to provide functionalities that either augment or expand upon the functionalities offered by the core PerMedCoE applications, for example by enabling specific steps required for data processing, management or analysis. The overall design of the auxiliary containers will be based on the same principles as the design of the core software containers. An example of an auxiliary container involves implementing a machine learning toolkit in support of specific scientific use cases defined in Work Package 3.

A2.1.4 Grouping of containers into building blocks

Containers will be grouped into building blocks based on functionalities delivered to end-users, with each functionality enabled by a core bioinformatics application(s) and/or supporting installations. The specific functionalities to be delivered are identified in Work Package 3. Each building block will invoke a single container to enable a single predetermined functionality. However, several building blocks may call upon the same container, depending on the task being executed.

A2.1.5 Building block interface and configuration

Similar to the BioExcel Building Blocks library (http://mmb.irbbarcelona.org/biobb), the user-facing side of the PerMedCoE building blocks has been designed so that no direct interaction with the underlying container software (such as using container software-specific launch commands) is required. A uniform set of Python wrapper scripts is used to provide standardised shell-level access to all building blocks (Table A1, Section A2.3.1; also see https://permedcoe.readthedocs.io). Building block configuration is based on flat .yaml files (or .json files) utilising a standard template, enabling the specification of software-specific options. Where feasible, the building blocks will be configured to use a set of default options for software execution where non-default options have not been specified.

Wrapper script argument	**Details**
--input	List of input paths
--output	List of output paths
--config	Path to config file (e.g. .yaml)
--tmpdir	Sets $TMPDIR and mounts /tmp inside the container
--help	Argument for displaying help information that shall list the interpretation of input and output files, and available non-standard command line arguments and configuration options
--processes	Argument required for MPI jobs
--gpus	Requirements for GPU jobs
--mem	Memory requirement

Table A1. Python wrapper arguments for creating a unified interface for the execution of PerMedCoE building blocks.

A2.1.6 Bundled versus external components for building block utilisation

The containerisation of PerMedCoE core applications (Section A2.1.2) requires the installation of several components within containers that are also available on the host system (e.g. Python). Where the containerised applications employ Python for data processing and/or analysis, completing these steps using a container-bundled Python installation is recommended to ensure version specificity and optimal I/O speeds. In contrast, the execution of building block wrapper scripts will rely on a Python installation available on the host system. Otherwise and in general, it is recommended that the building blocks should rely on no external dependencies (see Section A2.2.3).

By default, interactions between building blocks and workflow managers (Section A2.3.2) will rely on workflow manager installations on the host system, with no additional workflow manager processes launched by the containers. Launching workflow manager processes from within PerMedCoE containers will be made possible as a separate feature where required.

With reference to the execution of multi-node jobs requiring MPI, Singularity container image files can be configured in several ways, depending on the end purpose. For example, MPI configurations used by Singularity containers can rely on 'hybrid' or 'bind' models (see https://permedcoe.github.io/mpi-in-container for a detailed introduction to using MPI in Singularity image files). In the hybrid model, the host MPI acts in tandem with a MPI installation inside the container. The MPI version included in the container must match that on the host. In the bind model, the host system MPI (and/or elements related to it, such as relevant drivers) is bound to the container. Initially, PerMedCoE building blocks will utilise image files built using the hybrid model, with containers relying on the bind model being under investigation. Provided that their functionality can be ensured on multiple host systems, containers employing the bind model offer potential advantages over images based on the hybrid model, with container definition files using the bind model being comparatively robust to host system updates (see Section A2.1.7).

A2.1.7 Guidelines for container definition file preparation

In the following, recommendations are provided for the construction of container definition files to improve their adaptability and to facilitate version control. A suggested layout for Singularity definition files is provided in Table A2.

Software version specifications and operating system updates. It is recommended that software versions are specified at the beginning of the *%post* section in the Singularity container definition file, using environment variables. This enables the introduction of software updates without a need to modify subsequent installation commands. Where possible, operating system updates and library installations should also be introduced prior to other installation commands.

Software configuration. Where possible, PerMedCoE container definition files should employ a standardised shared set of environment variables to specify global configuration options. For example, for threaded applications (e.g. software installations using OpenMP threading), a single thread should be used by default. To enable users to modify threading settings depending on the analysis being performed, the number of threads will be made possible to specify in the building block configuration files (Section A2.1.5). Similarly, it is recommended that users be able to specify custom environment variables using building block configuration files.

Section / components	**Notes**
1. Container + building block titles	
2. Operating system specification	See Section A2.2.3 for information on operating system / distribution selection
3. Licensing and maintainer information	.txt file with summary of software licenses, maintainer name and contact details

4. External files copied into container	E.g. internal configuration files
5. Environment variables for installations	Separate from environment variables loaded upon container execution
6. General library installations	E.g. based on a list in a separate .txt file
7. General software installations	Software required by downstream installations and configuration steps
8. Core and auxiliary application installations	PerMedCoE core applications and software installations supporting them
9. Global environment variables	Default variables loaded at runtime

Table A2. Recommended layout of Singularity definition files for PerMedCoE containers.

A2.2 Software tool harmonisation and cross-platform compatibility

A2.2.1 Converting between container file types

While PerMedCoE building blocks will be based on Singularity containers (Section A2.1.1), Singularity Python (https://singularityhub.github.io/singularity-cli) can be used to convert Singularity definition files to a format compatible with Docker (https://www.docker.com). By default, Docker containers require root privileges from users and access to a daemon (Docker daemon). However, a rootless mode for running Docker containers is available in Docker v20.10 onward.

A2.2.2 Software dependencies

To ensure the mutual compatibility of PerMedCoE building blocks, installations of software dependencies will be matched across containers (e.g. via version-bound installation scripts). An outline of general requirements of the core applications (Section A2.1.1), as well as further dependencies introduced by their modification for deployment on HPC platforms, is provided in Table A3. For additional details for the refactoring and scaling-up of the applications for HPC use, see Milestone MS05 (Roadmap on core applications' needs for pre-exascale optimisation), Deliverable D1.1 (Roadmap of software scalability to pre-exascale, extension processes and best practices for software development) and Deliverable D1.2 (Software best practices and optimisation interim report).

Application	**General dependencies**	**Dependencies introduced by adaptation for HPC**
CellNOpt / CARNIVAL	C/C++, R, GraphViz > 2.2	OpenMP, MPI, HighFive

	CytoCopter plugin requires Cytoscape 3.5 and Java 11 (e.g. OpenJDK 11)	
COBREXA	Julia language runtime, JuMP.jl, DistributedData.jl, LP solver supported by JuMP.jl	(None)
MaBoSS	C++, Perl, Python	MPI
PhysiCell-X	C++, OpenMP	MPI, potentially libraries required for GPU support
Where employed, the COMP Superscalar (COMPSs) introduces several further dependencies. A full list of COMPSs dependencies is available in the COMPSs GitHub repository (https://github.com/bsc-wdc/compss).		

Table A3. Dependencies of HPC-adapted PerMedCoE core applications.

A2.2.3 Host environment characteristics and heterogeneity

This section provides an overview of HPC environments on which PerMedCoE tools are expected to be employed. Commonalities and differences between the environments are highlighted to support the development of practices aimed at ensuring the cross-platform compatibility of building blocks (Section A2.2.4). The contents of this section are based on a living document circulated between WP1-3 and are periodically updated.

Commonalities between host environments. Most HPC environments considered in PerMedCoE feature RHEL or CentOS as the operating system and Slurm as the batch job system. Most host systems have Singularity installations available, with support for OpenMPI and NVIDIA GPUs.

Differences between host environments. BSC CTE-Arm features PJM as the batch job system rather than Slurm. Several host systems presently lack PyCOMPsSs installations, with installations primarily available on BSC platforms and CSC Mahti. CSC LUMI and BSC CTE-AMD feature AMD (as opposed to NVIDIA) GPUs. Details concerning the network software stack require confirmation for most host environments.

A2.2.4 Steps to ensure compatibility between building blocks and HPC platforms

In the following, recommendations are made in order to ensure cross-compatibility between individual PerMedCoE Singularity image files, as well as building blocks and diverse HPC environments.

Internalisation of building block dependencies. With the exception of employing a host-specific Python installation to run wrapper scripts and job submissions requiring communication with the host MPI installation, dependencies needed to use PerMedCoE core and auxiliary applications should be included within the relevant container image files.

Options for threading. To ensure cross-container compatibility, options for threading support should be implemented in a standardised way in all PerMedCoE Singularity definition files. For example, the same Intel® oneAPI MKL Link Line Advisor settings should be used for Intel® oneAPI MKL installations in all definition files.

MPI alternatives. Depending on the host environment, the MPI installation may be based on either Open MPI or Intel® MPI (which implements MPICH). To enable their use on host environments with different MPI installations, it is recommended that separate implementations of the same PerMedCoE Singularity containers be designed with relevant MPI installations included, with building blocks providing the option to select the correct MPI version for a particular host environment. MPI version-specific differences in analysis code should be taken into account to support the implementation of PerMedCoE building blocks on multiple host environments. Solutions for automated switching between several MPI installations within a single container are subject to investigation.

Choice of Linux distribution. The choice of Linux distribution (e.g. Ubuntu versus CentOS) must be harmonised between individual building blocks because it has implications for job parallelisation using MPI. For example, OpenFabrics Enterprise Distribution (OFED) versions available via the Mellanox Technologies Ltd. public repository (https://www.mellanox.com/support/mlnx-ofed-public-repository) are distribution-specific.

GPU architectures. Similar to the implementation model concerning support for different host MPI installations, it is recommended that separate Singularity containers be developed for different GPU architectures (NVIDIA / CUDA or AMD / ROCm), with the relevant building blocks providing the opportunity to select the correct GPU architecture for a given host environment.

Optimal use of environment variables. Where any environment variables are set outside the container (Section A2.1.7), these should be prefaced by 'SINGULARITYENV_' to both separate them from variables already set on the host and to ensure that they are transposed into the container at runtime.

A2.3 Workflow overview and orchestration

This section details design choices with reference to the use of PerMedCoE building blocks as part of analytical workflows, including information on workflow management tools, job schedulers, and resource and user management.

A2.3.1 Utilisation of building blocks as part of workflows

Options for building block utilisation include:

- Manual execution
- Workflows created by end-users
- Using pre-built workflows
- Employing a meta-tool for workflow generation

Initially access to PerMedCoE building blocks will be provided via pre-built workflows specific to individual Use Cases, with options for workflow construction by end-users explored during the second half of the project. Current workflow managers employed for PerMedCoE workflow execution include PyCOMPSs, SnakeMake and NextFlow.

A2.3.2 Job schedulers and resource management

Building blocks are designed to be independent of the job scheduling method. Building blocks shall therefore not interact with any external resource manager or job scheduler. If any dynamic execution or scheduling is required (i.e. it is insufficient to assign a static pool of resources to the building block), building blocks shall request the required resources from the workflow manager, in a way configured by the workflow author. Building blocks should not attempt to independently probe what resources are available, but should work only with these explicitly specified by the workflow manager.

This restriction should make the building block execution easy to specify in many contemporary job schedulers and workflow specification frameworks, including PyCOMPSs, SnakeMake, NextFlow, HPC queueing systems such as Slurm and PBS, and even shell scripts and Jupyter notebooks useful for small-scale demonstration purposes.

A2.3.3 Federated user management

Support models for federated user management as part of PerMedCoE workflow management are currently under development. A framework for federated user management is required to enable building block and workflow execution on multiple HPC environments.

A2.4 Handling of sensitive data

To comply with regulations concerning the handling of sensitive data, no raw data or data generated by analytical software will be stored within PerMedCoE container images. Personally sensitive data will be processed to enable their anonymised use on diverse platforms and host systems. A centralised approach will be adopted for sensitive data processing, with work related to this topic carried out by the Barcelona Supercomputing Center (BSC).

A topic being investigated within PerMedCoE is the use of encrypted Singularity containers and their implementation as part of workflows on diverse host environments.

A2.5 Analytical diagnostics and performance monitoring

Analytical diagnostics. As part of designing PerMedCoE building blocks it is recommended that relevant analytical diagnostics (e.g. model diagnostics, warnings, and performance-relevant timing) be generated by default, in addition to output files produced by the analytical software being used. The diagnostics shall be stored in a line-oriented machine-readable text format, such as CSV or syslog-like records.

Performance monitoring. Monitoring of building block performance, scalability and adherence to PoP recommendations will be carried out using tools such as Extrae and Paraver. Wherever the comprehensive performance profiling is required within a building block, it is recommended that this be implemented separately (e.g. in a specialised building block version) or turned off by default, to avoid performance degradation during routine use.

Acronyms and Abbreviations

- CLI: Command Line Interface
- D: Deliverable
- GDSC: Genomics of Drug Sensitivity in Cancer
- HPC: High-performance computing
- HTTP: Hypertext Transfer Protocol
- IAM: Identity Access Management
- IC50: Half maximal inhibitory concentration
- JWT: JSON Web Token
- ML: Machine learning
- MN4: MareNostrum4
- MPI: Message Passing Interface
- ORAS: OCI Registry As Storage
- PoP: Performance Optimisation and Productivity
- RMSE: Root mean square error
- SSO: Single Sign-On
- UC: Use Case

References

1. PerMedCoE Deliverable D1.1: Roadmap of software scalability to pre-exascale, extension processes and best practices for software development
2. PerMedCoE Deliverable D1.2: Software best practices and optimisation interim report
3. Ding J, Condon A, Shah SP. (2018) Interpretable dimensionality reduction of single cell transcriptome data with deep generative models. Nat Commun 9: 2002. https://doi.org/10.1038/s41467-018-04368-5
4. Simm J, Arany A, Zakeri P, Haber T, Wegner JK, Chupakhin V, Ceulemans H, Moreau Y. (2017) Macau: Scalable Bayesian factorization with high-dimensional side information using MCMC. 2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP), 2017, pp. 1-6. DOI: 10.1109/MLSP.2017.8168143
5. PerMedCoE Deliverable D2.1: Pilot workflow to test infrastructure
6. PerMedCoE Milestone MS07: Design choices for building blocks
7. PerMedCoE Milestone MS05: Roadmap on core applications' needs for pre-exascale optimisation