Project Requirements

**Overview:** This document contains subjective technical and quantitative requirements for the project according to feedback from intended end users (MITgcm-ECCO researchers, scientists, climate science graduate students, and others interested in using MITgcm model data). This document will be continuously updated as the author sees fit as new requirements are introduced, existing requirements modified, and trade-offs explored throughout the project design process. Since the project’s objectives are to design a product that is robust enough to run a wide array of tasks in the MITgcm-ECCO workflow, it is reasonable to design a framework that divides the tasks across multiple units of software (containers) to prevent one unit of software from taking on a number or intensity of tasks that exceeds memory allocation and crashing. In turn, this necessitates a need for the units of software to communicate the output of these tasks with one another in real time which can be accomplished using a container orchestration platform. Since both the container orchestration platform and its containers are intended to run on user’s host machines, then, there will likely need to be host machine hardware and software considerations in the design of the product. **Therefore, the requirements are assigned to four sub-systems integral to the design process, namely, a portable unit of software (containerization) system, a container orchestration system, host machine hardware, and host machine software. In turn, each requirement will be denoted as either a subjective technical requirement (requirement that is technical in scope but has no tangible quantitative values ascribed to it), or as a quantitative requirement (requirement that has quantitative bounds on its operation).**

**Subjective Technical Requirements (In Progress)**

Each of the project’s subjective technical requirements will be enumerated in sections according to their associated subsystem in the final product design as follows:

1. ***Containerization Technology System-*** *This designation specifically refers to the open-source container platform itself that the containers will be built from (such as Docker, Ansible, Puppet, etc. and these packages' dependencies including engines, daemons, etc)*
2. Containerization system shall be compatible with most frequently used host machine platforms, architectures, and operating systems
3. Containerization system shall allow for the installation of ECCO and MITgcm libraries and their dependencies for building, running, and interpreting results from model problems

* Where automation is not possible, containerization system shall be able to be easily configured by end user to communicate with host machine for completing tasks in workflow (for example, through optfiles, mpi-tasks etc.)

1. Containerization system shall be documented with user guide that enables straightforward replication of steps necessary to execute tasks in workflow of final product(s) (CR)
2. Containerization system shall be open source with readily available documentation for the developer (CR)

***2. Container Orchestration Technology System***

1. Container orchestration system shall be compatible with most frequently used host machine platforms, architectures, and operating systems (SR)
2. Container orchestration system shall be easily configured to communicate with containerization platform for completing tasks in workflow (for example, through optfiles, mpi-tasks etc., exchange of information between containers) (CR)
3. Container orchestration system shall be documented with user guide that enables straightforward replication of steps necessary to execute tasks in workflow (CR)
4. Container orchestration system shall be open source with readily available documentation for the developer (CR)

***3. Containerization Software Stack -*** This designation refers to the software stack that the containers, their source code, their microservices, and the communication between containers will be built from.

**Smart Goal Product Release**

a) Containerization stack shall enable end users to build computationally inexpensive MITgcm model problems without MPI libraries (CR)

b) Containerization stack shall use job scheduler to schedule execution of computationally inexpensive model problems without MPI libraries (CR)

c) Containerization stack shall enable job scheduler to run computationally inexpensive model problems without MPI libraries (CR)

d) Containerization stack shall enable end users to build computationally inexpensive MITgcm model problems with MPI libraries (SR)

e) Containerization stack shall use job scheduler to schedule execution of computationally inexpensive model problems with MPI libraries (SR)

f) Containerization stack shall enable job scheduler to run computationally inexpensive model problems with MPI libraries (SR)

**Stretch Goal Product Release**

a) Containerization stack shall enable end users to implement model problem specifications through a GUI (CR)

b) Containerization stack shall enable end users to implement desired post-processing/data analysis tasks on output data through a GUI (CR)

c) Containerization stack shall enable end users to build and run more computationally expensive model problems without MPI libraries (CR)

d) Containerization stack shall enable end users to build, run more computationally expensive model problems with MPI libraries (SR)

e) Containerization stack shall offer support for building, and running computationally inexpensive ASTE regional domain model problems without MPI libraries (CR)

f) Containerization stack shall offer support for building, and running computationally inexpensive ASTE regional domain model problems with MPI libraries (CR)

g) Containerization stack shall offer support for building, and running computationally expensive ASTE regional domain model problems without MPI libraries (SR)

h) Containerization stack shall offer support for building, and running computationally expensive ASTE regional domain model problems with MPI libraries (SR)

\* During testing: Supplement documentation on what constitutes a 'computationally inexpensive model problem' using a combination of performance benchmarks and hardware resources such as CPU processor count, single vs. multithreading, parallelism, etc.

***4. Host Machine Hardware***

1. Containerization system shall be computationally inexpensive to run on end user host machines

* System shall reduce amount of RAM occupied by host machine when executing tasks in MITgcm-ECCO workflow (SR)
* System shall reduce CPU load of host machine when executing tasks in MITgcm-ECCO workflow (SR)

***5. Host Machine Software***

1. Containerization system shall be computationally inexpensive to run on end user host machines

* System shall reduce compile time necessary to execute tasks in MITgcm-ECCO workflow (SR)
* System shall reduce runtime time necessary to execute tasks in MITgcm-ECCO workflow (SR)

**Quantitative Requirements (In Progress)**

As test runs of building, running, and analyzing output from MITgcm model problems are performed, document statistics on computational costs (runtime, compile time, CPU load, RAM, etc.) and use these properties to gauge thresholds and objectives for design requirements to minimize these computational expenses.

**System: Containerization Technology System**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Requirement** | **Requirement Class** | **Thresholds** | **Objectives** | **Units** |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |
|  |  |  |  |  |  |

**System: Container-Orchestration Platform**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Requirement** | **Requirement Class** | **Thresholds** | **Objectives** | **Units** |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |

**System: Container-Software Stack (Smart Goal Product Release)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Requirement** | **Requirement Class** | **Thresholds** | **Objectives** | **Units** |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |

**System: Container-Software Stack (Stretch Goal Product Release)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Requirement** | **Requirement Class** | **Thresholds** | **Objectives** | **Units** |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |

**System: Host Machine Hardware**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Requirement** | **Requirement Class** | **Thresholds** | **Objectives** | **Units** |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |

**System: Host Machine Software**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Requirement** | **Requirement Class** | **Thresholds** | **Objectives** | **Units** |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |

\* The relevance of host machine hardware and software system requirements and their consideration in Feasibility Studies will be updated as the project evolves and will be dependent on the chosen containerization and container orchestration platforms. It is possible that the requirements for these two categories will be enumerated while not being considered as 'sub-systems' for which weighted scoring matrices for the Feasibility Analysis will be assembled. In other words, host machine hardware and software constraints in the production environment(s) on Sverdrup and/or TACC machines may be used to motivate the requirements that the resultant product in development must meet. In this way, there would not be any consideration of designs to implement in host machine hardware or software to optimize the product design. Requirements have been classified as either critical requirements ('CR' or requirements most crucial to the development and behavior of the end product and soft requirements ('SR' or requirements least crucial to the development and behavior of the end product). These definitions will be continuously updated throughout the project lifecycle as improved understanding of what requirements best constrain the successful development and deployment of a final product are obtained.

**Design Compromise Schematic(s) (In Progress)**

As test runs of building, running, and analyzing output from MITgcm model problems are performed use documented statistics to determine where trade-offs exist in performance to develop design compromise schematics and decision matrices

**Decision Matrices (In Progress)**