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 end users 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. The requirements are assigned to three sub-systems integral to the design process, namely, a portable unit of software (container technology) system, a container orchestration system, and a software stack system. 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). A Feasibility Analysis will be performed on the container technology and container orchestration system designs using subjective technical requirements that the developer deems important in the successful development and deployment of the product before the software stack for the products addressing the smart and stretch goals can be developed.

Before defining the subjective technical requirements and quantitative requirements for each system in the product design, a table depicting software constraints for at a minimum, reproducing the build and run steps of model verification experiments in a containerized environment will be provided. As the developer encounters additional software packages, libraries, and dependencies that are necessary for reproducing the MITgcm workflow for the model verification experiments, entries will be added, modified, and removed as the developer sees fit. Thus, this table will be an evolving form of documentation.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| No | Package Category | Package Release | Purpose of Use | Test Results | Alternative Packages |
| 1 | Developer Tools | which-2.20 | package installation locator tool | SIST | Not Necessary |
| 2 | Developer Tools | git-1.8.3.1 | version control | SIST | Not Necessary |
| 3 | Developer Tools | make-3.82 | Building Makefiles | SIST | Not Necessary |
| 4 | Developer Tools | vim | Linux Text Editor | SIST | Not Necessary |
| 5 | Compilers | gcc-4.8.5 | GNU C Compiler | SIUT | Other Versions? OpenHPC |
| 6 | Compilers | gcc-gfortran-4.8.5 | GNU Fortran Compiler | SIUT | Other Versions? OpenHPC |
| 7 | I/O | netcdf-4.3.3.1 | I/O for NetCDF data | SINT | Other Versions like 4.5.0? How to install? |
| 8 | I/O | netcdf-fortran-4.4.4 | I/O for NetCDF data w/Fortran | SINT | Other Versions? |
| 9 | MPI | openmpi-1.10.7 | OpenMPI build and run compatibility | UINT | Other Versions? |
| 10 | OpenMP |  |  |  |  |
| 11 | Code Repositories | MITgcm.git | Software stack for MITgcm models | SIST | Not Necessary |
| 12 |  |  |  |  |  |
| 13 |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Subjective Technical Requirements (Completed)**

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 and these packages' dependencies (including engines, daemons, etc)*
2. Containerization system shall be compatible with most frequently used host machine architectures (amd64 and arm64), and operating systems (Linux and MacOSX platforms) (CR)
3. Containerization system shall be open source with readily available documentation for the developer and end-user (CR)
4. Containerization system shall offer instant portability of containers from one platform to another within a fixed computing architecture (CR)
5. Containerization system shall allow for seamless installation and updates of ECCO and MITgcm libraries and their dependencies for building, running, and interpreting results from model problems (CR)
6. Containerization system shall use minimum system resources when running on end-user machines (CR)
7. Containerization system shall offer efficient CI/CD support for the developer (CR)
8. Containerization system shall provide useful performance monitoring and benchmarking capabilities (CR)
9. Containerization system shall generate meaningful logs for troubleshooting errors in development (CR)
10. Containerization system shall support persistent data storage for retaining useful model data (SR)
11. Containerization system shall permit the usage of large data volumes for storing model data (SR)
12. Containerization system shall have quick application performance (SR)
13. Containerization system shall have a comprehensive ecosystem and strong community (SR)
14. Containerization system shall provide good API and GUI support (SR)
15. Containerization system shall have regular updates and releases from developers (SR)
16. Containerization system shall have built-in protection from security vulnerabilities (SR)
17. Containerization system shall have strong adoption and popularity amongst companies and organizations (SR)

***2. Container Orchestration System (Completed)***

1. Containerization system shall be compatible with most frequently used host machine architectures (amd64 and arm64), and operating systems (Linux and MacOSX platforms) (CR)
2. Container orchestration system shall be open source with readily available documentation for the developer (CR)
3. Container orchestration system shall offer instant portability of containers from one platform to another within a fixed computing architecture (CR)

d ) Container orchestration system shall allow for seamless installation and updates of ECCO and MITgcm libraries and their dependencies for building, running, and interpreting results from model problems (CR)

e) Container orchestration system shall have effective load balancing functionality (CR)

e) Container orchestration system shall use minimum system resources when running on end-user machines (CR)

f) Container orchestration system shall provide useful performance monitoring and benchmarking capabilities (CR)

g) Container orchestration system shall generate meaningful logs for troubleshooting errors in development (CR)

h) Container orchestration system shall be easy to install, set up, and update (CR)

1. Container orchestration system shall support persistent data storage for retaining useful model data (SR)
2. Container orchestration system shall permit the usage of large data volumes for storing model data (SR)
3. Container orchestration system shall have a comprehensive ecosystem and strong community (SR)
4. Container orchestration system shall be tailored towards local development environments (SR)
5. Container orchestration system shall provide good API and GUI support (SR)
6. Container orchestration system shall have regular updates and releases from developers (SR)
7. Container orchestration system shall have effective lifecycle management (SR)
8. Container orchestration system shall have built-in protection from security vulnerabilities (SR)
9. Container orchestration system shall have flexible standards for development (SR)

***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.

**Feasibility Analysis**

**Overview:** Using the subjective technical requirements documented above, a Feasibility Analysis will be performed to study the opportunity costs of incorporating different Container Technology System and Container Orchestration System Designs into the stretch and smart goal product designs. Note that the subjective technical requirements for the stretch and smart goal product software stacks will not be incorporated into the Feasibility Analysis because their requirements cannot be tested until a container technology and orchestration system have been chosen and implemented into the development environment. The Feasibility Analysis will adopt a linear scoring methodology where the requirements will be sorted in descending order from most to least important and assigned a numeric value in linearly descending order with the most important being assigned a value equal to the number of requirements for that system design and the least important, a value equal to 1. For each design, if the requirement is met, a value equal to that requirement number is assigned to that design, otherwise, a value of 0 is assigned for not having met that requirement. A linear combination of values will then be taken for each design and color-coded according to the criteria defined at the top of the legend in the Feasibility Analysis spreadsheet. The design that meets the largest percentage of documented subjective technical requirements will be downselected for the Prototype and Testing Phase of the design.

**Results:**

According to the results from the Feasibility Analysis, the container technology and container orchestration designs that meet the largest percentage of criteria (requirements) and are thus downselected for detailed design are Docker and Docker-Compose. The next steps in the design of the products will be to install and set up these systems on the container and singularity development environments. Once set up is complete, requirements and criteria regarding successful development of source code to be tested and deployed in these products' distributed systems will be documented. Then, rudimentary design schematics including class diagrams, pseudocode, and flowcharts will be developed to characterize the behavior of the source code, as well as the testing, integration, and deployment of the source code inside of the container technology system (Docker) and container orchestration system (docker-compose). As an aside, it is possible that there may be too much overhead in configuring docker-compose to run docker containers using singularity in the singularity environment on Sverdrup and TACC compute nodes. If this is the case, an alternative container orchestration tool such as singularity-compose may be explored.

**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.

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

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

Design Compromises

1) Model Problem Build and Run Efficiency

a) Sophistication of Model Build Process

b) Load Balancing Efficiency

c) CPU Wall Clock Time

d) RAM of Host Machine

e) Computing Platform Compatibility

Sophistication of Model Build Process

Model Problem Build and Run Efficiency

Load Balancing Efficiency CPU Wall Clock Time

RAM of Host Machine Computing Platform Compatibility

2) Containerized GUI Functionality

a) User Experience/Navigability of GUI

b) CPU and RAM of Host Machine

d) Quantity and Quality of Features

Quality and Quantity of GUI Features

Containerized GUI Functionality

User Experience/Navigability CPU and RAM of Host. Machine

3) Ease of Product Operation for End User

a) Installation and Operation Documentation

b) Product Automation of Workflow

c) Custom Interaction/Modification of Workflow

Product Automation of Workflow

Ease of Product Operation for End User

Custom Modification of Workflow Installation and Operation Documentation

**Decision Matrices (In Progress)**

**A picture containing text, font, screenshot, line

Description automatically generated**

**A blue line with black text

Description automatically generated with low confidence**

|  |  |
| --- | --- |
| 1) Build and run computationally inexpensive model problems w/o MPI libraries  2) Document and release detailed product designs in code repository and image registry | 1) Extend support for products on computing platforms commonly used by CRIOS community that differ from development environments |
| 1) Implement GUIs for end users to input model specifications and select data-processing/analyses tasks  2) Introduce support for building and running ASTE regional domain problems | 1) Scale up support for more computationally expensive model problems  2) Configure support for products on broad range of computing platforms for general MITgcm-community use |