11falsenonelisttrue

Compute Resources

Have questions or need help with compute, including activation or issues? Follow this link.

User Agreement

Docker Usage

The information on this page assumes that you have a knowledge base of using Docker to create images and push them to a repository for use. If you need to review that information, please see the links below.

https://washu.atlassian.net/wiki/spaces/RUD/pages/1705115761/Docker+and+the+RIS+Compute1+Platform?atlOrigin=eyJpIjoiNzc4YTZjNjIxYmQwNGI3OTk4M2Q0MV

https://washu.atlassian.net/wiki/spaces/RUD/pages/1864892726/Docker+Basics+Building+Tagging+Pushing+A+Custom+Docker+Image?atlOrigin=eyJpIjoiMTVjMjNlM

Overview

The purpose of this tutorial is to demonstrate usage of the Intel® Compiler Base image for use on the Scientific Compute Platform. Please refer to this page for more information on the Intel® Compiler Base image. In this tutorial, we will be compiling a sample MPI-enabled `Hello World` program. The tutorial uses material from this page.

Requirements

Docker Desktop (https://www.docker.com/products/docker-desktop)

Free Docker Hub Account (https://hub.docker.com)

Multi-Stage Build

This section of the tutorial will demonstrate how to build a multi-stage Docker image for use on the Scientific Compute Platform. An MPI-enabled `Hello World` program is compiled using the Intel® Compiler Base image.

MPI Hello World Code Example

Below is sample code for a simple MPI-enabled `Hello World` program.

#include int main(int argc, char** argv) { // Initialize the MPI environment MPI_Init(NULL, NULL); // Get the number of processes int world_size; MPI_Comm_size(MPI_COMM_WORLD, &world_size); // Get the rank of the process int world_rank; MPI_Comm_rank(MPI_COMM_WORLD, &world_rank); // Get the name of the processor char processor_name[MPI_MAX_PROCESSOR_NAME]; int name_len; MPI_Get_processor_name(processor_name, &name_len); // Print off a hello world message printf("Hello world from processor %s, rank %d out of %d processors\n", processor_name, world_rank, world_size); // Finalize the MPI environment. MPI_Finalize(); }]]>

Dockerfile for a Multi-Stage Build

Using your favorite text editor, save the above code to a file called `mpi_hello_world.c`. In the same folder as the `mpi_hello_world.c` file, create a file called `Dockerfile`. Add the following lines of code to the `Dockerfile` file:

Building and Pushing the Docker Image

Please see this section for more information on building and pushing the Docker image to Docker Hub.

Interactive Command-Line Session

Job Submission

The following command can be used to submit an interactive command-line job to run the MPI-enabled `Hello World` program. Make sure to replace `docker/image` with the name of the Docker image you created.

The job submission command uses environment variables required for parallel computing on RIS. The job submission is also requesting 20 vCPUs spread across 5 exec nodes. Please see the parallel computing documentation for more information.

Running the MPI-enabled `Hello World` program

Run the MPI-enabled `Hello World` program using 20 vCPUs with the following command:

The following output should be displayed after the image has finished downloading on all exec nodes:

Single-Stage Build

Should you require a single-stage build, the following Dockerfile can be used. Please be aware that this method results in a larger Docker image which may cause increased computing time/resources/cost. This method also caches the source code in build layers resulting in public exposure, which may be unwanted. When possible, it is advised to use the multi-stage build method.