ERDC/ITL TN-24-1
January 2024

Dockerization of the Coastal Model Test Bed Toolkit

By Michael A. Clement, Spicer Bak, and Tyler Hesser

PURPOSE: The purpose of this technical note is to document and describe changes made to the Coastal Model Test Bed (CMTB) suite of software in conjunction with the version 2 (V2) update.

BACKGROUND: The CMTB project incorporates numerical models and facilitates researching environmental conditions that motivate morphological changes in coastal zones (Young et al. 2019). The CMTB project provides a platform for evaluating the capabilities and weaknesses of coastal numerical models. This is done by making comparisons of model results to data captured at the Coastal and Hydraulics Laboratory (CHL) Field Research Facility (FRF) (Bak et al. 2017). CMTB was written in the Python programming language. As such, it offers a high degree of portability and usability for developers looking to make use of the tools or extend existing functionality. In an effort to improve the developer experience as well as build toward a more sustainable development future for the project, it was determined that the CMTB codebase should be placed in a Docker environment. This document discusses the reasoning behind this decision, how it was carried out, and new workflows that result from the so-called "Dockerization" of the application code.

OVERVIEW OF EXISTING CODEBASE: This section provides a brief overview of the components that make up the CMTB software.

The CMTB provides scaffolding for examination and verification of numerical models leveraged by the US Army Corps of Engineers (Bak et al. 2017), and the nearshore scientific community. CMTB allows near real-time evaluation of these numerical models while providing access to substantial, high-quality data harvested from the CHL FRF. With this data in hand, model initialization is significantly simplified (Young et al. 2019). The core CMTB code is contained in a repository that then references other repositories (submodules) that contain supporting code. This separation of concerns allows specific functionality to be developed independently while preserving the overall structure and behavior of the main CMTB application code.

The submodules used in CMTB are as follows:

1. **getdatatestbed**—This submodule offers a means of interacting with the CHL public Thematic Real-time Environmental Distributed Data Services (THREDDS) server and local FRF server for fetching pertinent coastal data.

2. **prepdata**—This submodule provides functionality for cleaning and processing various input and output data formats used within CMTB.

3. **testbedutils**—This submodule holds general purpose FRF tools that are used for processing wave, model, and geoprocessing data.

Prior to 2023, researchers wanting to use the CMTB tools needed to pull them from a public git repository and set up a development environment that worked in their specific computing environment before they could start to leverage the project. The process of setting up the development environment was potentially prone to errors, which could lead to a delay in researchers being able to perform actual work. Using the project on different operating systems posed a similar challenge as certain aspects of the project had different requirements depending on the operating system of the development environment. While the use of the Python programming language provided significant cross-platform support, there was room for improvements when it came to package management, development environment consistency, and ease of onboarding.

As part of a larger effort to bolster and streamline CMTB code assets, a new repository was created on the Engineer Research and Development Center (ERDC) public GitLab instance. In addition to relocating the code, significant changes were also planned to bring the functionality of the project to a more advanced state. This set of enhancements is what makes up the latest version of the CMTB project, known as V2.

OVERVIEW OF DOCKER: A key component of this effort was the creation of a containerized environment in which to run the project. This was accomplished using Docker. Docker is a cross-platform containerization solution that allows developers to create customized environments (called images) that will run the same on any machine that supports Docker. This means developers on the CMTB project can share a common and predictable computing environment despite using different operating systems or having different system configurations. The use of a containerized environment in conjunction with the portability of Python adds an extra layer of stability and consistency to the CMTB development experience.

By containerizing the CMTB development environment, developers working on the CMTB project gain a higher degree of simplicity and safety with respect to running the CMTB code and any changes they might add. This also paves the way for potential future deployments of the CMTB project on web servers or in other environments.

OVERVIEW OF HARBOR: The Docker images for this project are stored in a container registry. A container registry is an online platform that provides a central location where Docker images may be stored (pushed) and retrieved (pulled). It allows developers to host prebuilt Docker images that other users can take advantage of without having to build them manually. This saves time and ensures developers always have access to critical Docker images.

The container registry used for the CMTB project is called Harbor (https://harbor.erdc.dren.mil). To interact with Harbor, a user must have an account. To make an account, the user can simply navigate to the URL, and the account creation process will begin automatically. Once the account is set up, the user's ID must be added to the CMTB project by another developer with access. Once access has been granted, the new developer can sign into Harbor from the command line of his work station and then run the Docker pull command to retrieve the image from harbor.

DOCKERIZED WORKFLOW: The workflow for developing new features of the CMTB project in the Docker environment is very straightforward. The primary change to the developer's workflow is that he first must install Docker (instructions for installing docker are easily located on the web and are out of scope for this document). With Docker installed on the developer's machine he can clone the repository and begin interacting with the CMTB.

After cloning the repository, the developer must then either pull the pre-built Docker image for the project or build it locally. This Docker image contains the working environment and all necessary software libraries to interface with the CMTB code. Pulling the pre-built image offers a time-savings benefit for the developer as it eliminates the need to build and install all dependencies for the project (an automated, but time-consuming process).

Pulling the Prebuilt Image

Prerequisites:

- A user account with access to the CMTB group on the ERDC Information Technology Laboratory's Harbor container registry (https://harbor.erdc.dren.mil)
- The client secret, found in the user profile screen on Harbor

To pull the prebuilt Docker image follow the proceeding steps:

1. First log into Harbor from a command line. Here is the command for Mac/Linux (note that "<USERNAME>" must be replaced with the current user's Harbor username):

   ```
   docker login --username <USERNAME> https://harbor.erdc.dren.mil
   ```

   Then, enter the client secret when prompted for a password.
2. From the same terminal window where the login was performed, change directory to the root of the CMTB code repository
3. Run the following command to pull the image and start the project:

   ```
   docker compose up
   ```

Building the Image Locally. If there are issues with logging in or pulling the image directly from Harbor it can be built locally. To do this the developer must open a terminal/command prompt and change directory to the root of the project. Once there, he must run the following commands:

1. From a terminal window in the root of the project (this may take several minutes to complete):

   ```
   docker compose build
   ```

2. When the build has completed, the project can be started by running:

   ```
   docker compose up
   ```

After completing the aforementioned steps, the project will be running locally, and the developer can navigate to the Jupyter notebooks in their web browser.

Additionally, a shell into the containerized environment may be opened by launching a new terminal window and running the following command from the project's root directory:

```
docker compose exec cmtb bash
```

This allows the developer to run any necessary command line tools in the project environment.

Note: Anything done in a shell session within the container (e.g., packages installed, environment variables set, etc.) will not persist after the docker containers are shut down. To make persistent changes to the environment, the Dockerfile must be adjusted and the image rebuilt.

OVERVIEW OF FEATURE DEVELOPMENT WORKFLOW: To standardize the development of new features for the CMTB project, a Gitlab workflow has been adopted. There are four steps in the process of developing a new feature.

The first step is to create an issue in Gitlab with a descriptive name matching the feature to be developed. Using Gitlab's issue tracker when developing features provides a convenient place to store pertinent documents, have discussions and code reviews, and track requirements details.

The second step is to create a git branch from the issue. This will associate all code that is pushed with the existing issue.

The third step is to develop the feature on the new git branch.

The fourth step is to open a merge request on Gitlab. A merge request is a request to take the code from the feature branch and combine it with the project's main branch. This happens when all development on the feature has completed.

This process ensures that feature development happens in a controlled and predictable way. It also promotes the development of features in small, discrete installments rather than as more drawn-out endeavors.

SUMMARY: The containerization of the CMTB codebase and subsequent enhancements to the GitLab workflow serve to provide a more unified, predictable, and structured development environment and improvement process for the CMTB project code. The overarching goal of this and future efforts in this vein is to make the CMTB codebase more robust, extendable, and deployable in order to heighten impact.

ADDITIONAL INFORMATION: This technical note was prepared by the US Army Engineer Research and Development Center, Information Technology Laboratory. Questions about this technical note can be addressed to Mr. Tyler J. Hesser (Tyler.J.Hesser@usace.army.mil). This technical note should be cited as follows:

> Clement, Michael A., Spicer Bak, and Tyler J. Hesser. 2024. *Dockerization of the Coastal Model Test Bed Toolkit*. ERDC/ITL TN-24-1. Vicksburg, MS: US Army Engineer Research and Development Center.

REFERENCES

Bak, A. S., T. Hesser, J. Smith, and M. Bryant. 2017. *Initialization and Setup of the Coastal Model Test Bed: STWAVE*. ERDC/CHL CHETN-I-93. Vicksburg, MS: US Army Engineer Research and Development Center.

Young, L. D., A. S. Bak, and B. D. Johnson. 2019. *Initialization and Setup of the Coastal Model Test Bed: CSHORE*. ERDC/CHL CHETN-IV-115. Vicksburg, MS: US Army Engineer Research and Development Center. http://dx.doi.org/10.21079/11681/32387.