NOAA Technical Memorandum NOS CS 52

On-demand Automated Storm Surge Modeling Including Inland Hydrology Effects

Silver Spring, Maryland
December 2022

noaa National Oceanic and Atmospheric Administration

U.S. DEPARTMENT OF COMMERCE
National Ocean Service
Coast Survey Development Laboratory

Office of Coast
Survey National
Ocean Service
National Oceanic and Atmospheric
Administration
U.S. Department of Commerce

The Office of Coast Survey (OCS) is the Nation's only official chartmaker. As the oldest United States scientific organization, dating from 1807, this office has a long history. Today it promotes safe navigation by managing the National Oceanic and Atmospheric Administration's (NOAA) nautical chart and oceanographic data collection and information programs.

There are four components of OCS:

The Coast Survey Development Laboratory develops new and efficient techniques to accomplish Coast Survey missions and to produce new and improved products and services for the maritime community and other coastal users.

The Marine Chart Division acquires marine navigational data to construct and maintain nautical charts, Coast Pilots, and related marine products for the United States.

The Hydrographic Surveys Division directs programs for ship and shore-based hydrographic survey units and conducts general hydrographic survey operations.

The Navigational Services Division is the focal point for Coast Survey customer service activities, concentrating predominately on charting issues, fast-response hydrographic surveys, and Coast Pilot updates.

NOAA Technical Memorandum NOS CS 52

On-demand Automated Storm Surge Modeling Including Inland Hydrology Effects

Soroosh Mani[1, 2], Saeed Moghimi[1], Linlin Cui[3], Zhengui Wang[3], Y. Joseph Zhang[3], Jesse Lopez[5], Edward Myers[1], Tim Cockerill[6], Shachak Pe'eri[1]

1. Office of Coast Survey, Coast Survey Development Laboratory, Silver Spring, Maryland
2. Spatial Front Inc., Bethesda, Maryland
3. Virginia Institute of Marine Science, Gloucester Point, Virginia
4. University Corporation for Atmospheric Research, Boulder, Colorado
5. Axiom Data Science, Portland, Oregon
6. Texas Advanced Computing Center, University of Texas, Austin, Texas

December 2022

noaa National Oceanic and Atmospheric Administration

U. S. DEPARTMENT OF COMMERCE
Gina Raimondo, Secretary

National Oceanic and Atmospheric Administration
Richard Spinrad, Under Secretary

National Ocean Service
Nicole LeBoeuf, Assistant Administrator

Office of Coast Survey
Rear Admiral Benjamin Evans
Director

Coast Survey Development Laboratory
Corey Allen
Acting Division Chief

NOTICE

Mention of a commercial company or product does not constitute an endorsement by NOAA. Use for publicity or advertising purposes of information from this publication concerning proprietary products or the tests of such products is not authorized.

TABLE OF CONTENTS

TABLE OF FIGURES

Abstract

This project is a multi-agency (National Oceanic and Atmospheric Administration, US Geological Survey, and Office of Naval Research) collaboration with academia (Virginia Institute of Marine Science) and industry (Axiom Data Science) to improve storm surge modeling workflow. It is a part of the NOAA effort to develop disaster risk assessments tools and practical technical applications to reduce and mitigate coastal vulnerability to natural disasters. Efforts such as this will also support products and services that are vital for: 1) maritime navigation, 2) geodetic infrastructure, and 3) responsible use of ocean resources for the benefit of the economy.

The main goal of this project is the development of a cloud-based system to automatically prepare, mesh, and run a numerical coastal ocean modeling system and to post-process the system's predictions. This includes creating necessary tools or improving existing packages used within the system. The current report describes the implementation of the system and its components. From a high-level, the system is:

- Fully automated: no requirement for human intervention during simulation
- End-to-end: starts from elevation data and ends with generating simulation result maps
- Cloud based and hybrid: the system can fully run on cloud resources or use a combination of cloud and other available computational resources
- Scalable: the system can be easily scaled on the cloud

In this system, the computational mesh is generated dynamically for each specific storm, based on its track. Then, the elevation values are automatically interpolated from the elevation data so that the discretized domain realistically represent the actual bathymetry.

Inland hydrology and atmospheric forces are set up on the discretized domain to account for their effects on coastal flooding. There are two options for atmospheric forcing: either using numerical weather prediction model products, or an idealized parametric tropical cyclone model.

This project provides an automation framework for scalable simulation. Different components of the system are loosely coupled, such that each can be modified independently from the others. This makes modifying different steps, or adding additional steps, logic, and features relatively easy.

Keywords: cloud computing, data-driven workflow, AWS, Prefect, SCHISM, National Water Model, storm surge modeling, compound flooding

Terminology

The following terms are used throughout this manuscript. This section helps clarifying the meaning and context in which they are used.

- **Alpine Linux:** A lightweight minimal Linux distribution aimed at creating secure container images.
- **Amazon Web Services (AWS):** The cloud service provider used for this project. The backend of the workflow runs on this platform, including the workflow agent.
    - **Access Control List (ACL):** Access control settings for S3 buckets
    - **boto3:** Software development kit for interacting with AWS services in Python scripts.
    - **Elastic Compute Cloud (EC2):** An instance of a virtual machine provided by AWS.
    - **Elastic Container Service (ECS):** AWS container orchestration service. This service can run Docker containers and manage scaling up.
    - **Elastic Container Registry (ECR):** AWS container registry for storing generated Docker images
    - **Elastic Files System (EFS):** A Network File System (NFS) that can be mounted on multiple EC2 instances or for multiple ECS tasks within a subnet.
    - **Fargate:** AWS serverless compute engines
    - **FSx:** AWS high performance files system for providing capabilities such as Lustre on HPC.
    - **Identity & Access Management (IAM):** AWS service for authenticating users, accounts and services.
    - **Simple Storage Service (S3) Bucket:** Object storage provided by AWS, which is usually used for long term storage.
    - **Virtual Private Cloud (VPC):** AWS virtual networking infrastructure unit used for isolating/controlling traffic
- **Ansible:** A package used to configure provisioned infrastructure in a reproducible manner. This includes installing required packages and running necessary services.

- **Application Programming Interface (API):** Set of conventions and functions defined for utilizing a service or program.
- **Axiom Data Science/Axiom:** The private company that partnered with NOAA in a NOPP project, for developing the system described in this report.
- **Conda/Miniconda:** A package manager for setting up environment for developing or running other packages
- **Digital Elevation Model (DEM):** Computer representation of elevation data for a given region.
- **Docker:** The technology used for containerizing workflow components to run on AWS ECS service.
    - **Docker images:** Containerized environment with all the dependencies to be used for running different workflow tasks
    - **Docker compose:** A script that facilitates declaring how Dockerfiles are be built and the resulting images are started (arguments, tags, etc.)
    - **Docker containers:** A running environment created from a Docker image
- **EC-JRC:** European Commission Joint Research Centre
- **ECMWF:** European Centre for Medium-Range Weather Forecasts
- **ECMWF Reanalysis (ERA5):** data on atmospheric parameters
- **GIS:** Geographic Information Systems
- **GitHub:** The platform used for code and project management.
- **Global Forecast System (GFS):** NOAA's global weather forecast model for atmosphere, ocean, land and sea ice.
- **High Performance Computing (HPC):** Cluster systems where many nodes are connected on a fast network for running distributed parallel jobs along with dedicated storage (usually Lustre) shared between those nodes. Usually a job scheduler, (e.g. Slurm, PBS, etc.) is used to balance jobs on the nodes of these clusters.
- **High Resolution Rapid Refresh (HRRR):** NOAA's real-time 3-km resolution, hourly updated, cloud-resolving, convection-allowing atmospheric model for continental US and Alaska

- **HTML:** Hyper Text Markup Language. A language used for writing webpages
- **HTTP:** Hyper Text Transfer Protocol.
- **Input/output (IO):** The term used for describing the interaction of a running program with storage devices for writing and reading data.
- **MAE:** Mean Absolute Error
- **MFA:** Multi-Factor Authentication
- **MPI/OpenMP:** Different technologies for executing distributed parallel programs
- **National Water Model (NWM):** Numerical hydrology model for riverine flow in the US.
- **Network File System (NFS):** File system storage capability provided remotely over a local or wide-area network.
- **NOAA:** National Oceanic and Atmospheric Administration
    - **CO-OPS:** Center for Operational Oceanographic Products and Services
    - **NHC:** National Hurricane Center in NOAA National Weather Service office
    - **OCS:** Office of Coast Survey in NOAA National Ocean Service office
- **NOPP:** National Oceanographic Partnership Program
- **OCSMesh:** A Python package for coastal ocean model mesh preparation.
- **ONR:** Office of Naval Research in the US
- **ParallelWorks (PW):** A managed cloud environment provided through NOAA-RDHPCS. It is used for running computationally expensive segments of the workflow.
- **Prefect:** The workflow management tool. The workflow is defined in Prefect's Python API. Prefect Cloud is used for managing tasks while multiple Prefect agents with different credentials are used to execute tasks on different platforms.
    - **Prefect Cloud:** A workflow server-on-cloud service provided by the Prefect company to make deploying workflows more convenient.
    - **Prefect task:** The unit of job definition in the implementation of Prefect

    - **Prefect flow:** A series of tasks or flows bundled together to be executed in a single environment.
    - **Prefect run (flow-run):** The execution of a predefined Prefect flow
- **PySCHISM:** A Python package for preparing SCHISM model inputs.
- **RDHPCS:** NOAA's Research and Development High Performance Computing System providing HPC service on different platforms.
- **SCHISM:** Semi-implicit Cross-scale Hydro-science Integrated System Model. The coastal ocean model used by the system described in this manuscript.
- **SSH:** Secure Shell. A technology for connecting to a remote machine securely.
- **StormEvents:** A Python package for gathering data about storms tracked by NOAA.
- **Terraform:** A software used for provisioning AWS services.
- **Thalassa:** A Python package for visualizing coastal ocean model results through a dashboard on the browser.
- **TPXO:** A series of global models of ocean tides. Its copyright belongs to Oregon State University.
- **USGS:** United States Geological Survey
- **Virginia Institute of Marine Science (VIMS):** NOAA's academic partner in implementing the workflow as a part of the NOPP project.
- **Workflow/System:** The implemented automated end-to-end storm surge modeling workflow.
- **YAML:** Yet Another Markup Language. It's a file format for storing structured multi-level key-value pairs.
-

Introduction

Numerical modeling of physical phenomena, such as weather or ocean conditions, consists of three main steps:

1. Preparing the model inputs and configuration (preprocessing)
2. Running the solver (or multiple solvers in case of coupled models)
3. Processing the results of the simulation to generate human readable and comprehensible outputs (post processing).

This process as well as each of its steps brings with it some challenges for the modelers; many of the modelers are only interested in the final results in the form of reports and visualizations. Various tools (open source or proprietary) for numerical modeling exist, but usually the modeler is responsible for finding these tools and gluing them together to establish the full simulation process (the three steps described above). This usually involves writing custom scripts as well as manual transfer of data between storage infrastructure. Another challenge the modeler needs to overcome is finding resources to run these computationally expensive numerical models. The current solution for the resource issue is using traditional HPC platforms; these platforms require constant maintenance.

This work presents a solution to these problems, in the form of an automated on-demand system, developed by the authors, to address the two issues mentioned above: the resource requirement and workflow management. The system utilizes cloud technologies to address the need for computational resources while avoiding the maintenance cost (on-demand system), and uses a workflow manager to handle the logic for orderly execution of all simulation steps, from preparation all the way to post processing.

In specific, this project created an automated workflow for simulating storm surge in the event of major Atlantic hurricanes. The workflow is orchestrated by Prefect and is using AWS cloud services to implement the on-demand system. This workflow uses open source packages, namely StormEvents (Burnett, 2021), OCSMesh (Mani, et al., 2021), and PySCHISM (Calzada, Cui, & Wang, 2020), to:

1. Gather data about the storm
2. Generate the simulation domain
3. Discretize the domain
4. Set up the forcing and boundary conditions for the model
5. Run the solver - SCHISM (Zhang Y. , Ye, Stanev, & Grashorn, 2016)
6. Generate visualization from the results

Note that this system is not currently operationalized and is demonstrated here as a proof of concept. The code and configuration files used for this system can be found on GitHub at: https://github.com/SorooshMani-NOAA/odssm-infra.

Features of the System

The system can be described by the following highlights:

- Event based: Each step in the workflow is executed based on the results of the previous steps. Each step raises a success or failure signal to notify the rest of the process.
- Hybrid: This system may run completely on a single cloud service provider. It can also be configured to use multiple cloud providers. Specifically, this work explores using the ParallelWorks platform to implement the hybrid workflow.
- Flexible: Each step of the workflow can be easily modified or be replaced in isolation. New steps can also be added to improve the functionality of the workflow.
- Automated: The workflow system doesn't require human intervention.
- Data driven: The entire process is data-driven. For example, the unstructured mesh generation relies on the data from storm track as well as DEMs.
- No lock-in: Most of the tools that are used to create this workflow are open source generic tools and services which helps avoiding lock-in with any technology or service providers.
- No infrastructure maintenance: The system is on-cloud; hence the limiting factor is cost management, not server maintenance.

Workflow Overview

The workflow can be seen as a superset of the modeling process. The latter involves dealing with numerical model execution, inputs, and outputs. Other than the modeling process, the workflow needs to handle tasks such as secret management, infrastructure management, data transfer, shared resource management, and other similar tasks. These tasks are defined in Python using Prefect. This segment first describes the modeling process and then describes how this process fits into and interacts with the full system.

Modeling Process

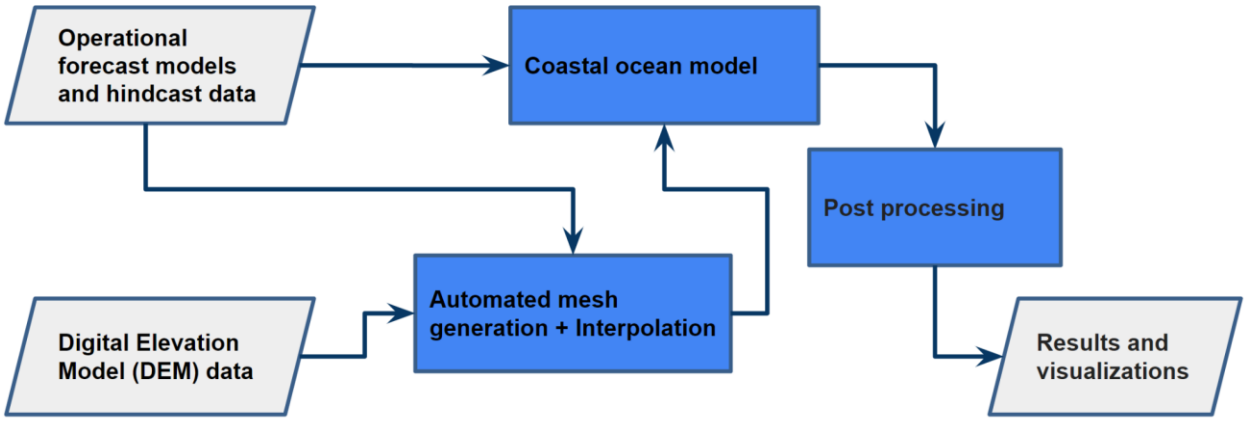

Figure 1 Overview of the modeling process

Figure 1 depicts an overview of the modeling process. It can be viewed through the lens of the three-step process described in the introduction.

The first step (preprocessing) consists of two parts:

a) Defining and discretizing the modeling domain
b) Gathering forcing and boundary conditions data for the model

The computational domain and the unstructured mesh are dynamically generated and are adjusted for each storm path; this would require information about the hurricane track. Then elevation data, usually available in the form of DEM tiles, is interpolated on the “raw” mesh. The workflow provides two options for mesh generation: one is generating the mesh from scratch, which is computationally expensive; another is combining existing mesh for the same domain. In the second part of preprocessing, forcing and boundary data are fetched from NOAA (and other similar organization) databases. For example, atmospheric data is obtained from NOAA GFS model outputs, and river flow data is downloaded from NWM dataset servers.

The second modeling step is running the numerical model, SCHISM. Similar to mesh generation, this step requires a lot of computational resources. SCHISM is parallelized through MPI and OpenMP technologies. This allows for distributed parallelization in both numerical solution and IO, which in turn benefits from high core-count environments, such as cloud or traditional HPC.

In the last step, after the model output is obtained, the results need to be properly gathered and analyzed to produce human readable outputs, such as maps, graphs and other forms of visualizations. Two independent solutions are provided as a part of this

project. The first is an integrated component in the workflow that generates interactive static HTML maps. The second solution is a fully dynamic interactive UI that can be used to explore different solution files. The second method is not an integral part of the process, but (as described later) is provided as a separate post processing solution.

System Management

Figure 2 shows how the modeling process fits into the workflow system. The workflow manager (Prefect - consisting of clients and a server) as well as computational and storage backend is provisioned using Ansible and Terraform (infrastructure managers). Then the system is ready to run simulation workflows.

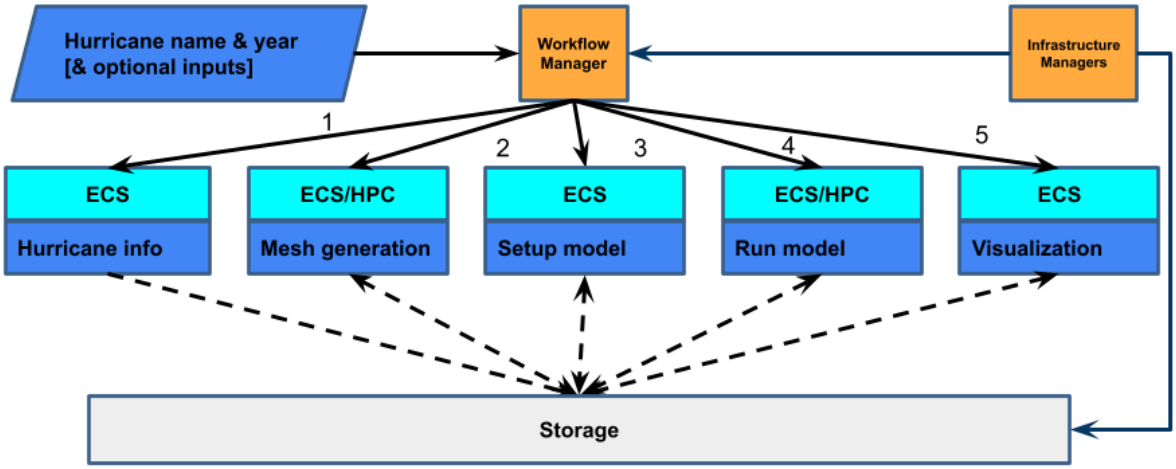

Figure 2 Simplified diagram of the workflow.

This system consists of different storage technologies for long-term and short-term storage. When executing an instance of the workflow (Prefect run), the data required by the run is synchronized between long-germ and short-term storage before the simulation process begins. Each step of the simulation process is executed on a dedicated machine, provisioned specifically for that task to make sure it has enough resources to run the task. Therefore, the workflow manager starts a dedicated machine for the task, run the task (e.g. mesh generation) and then shutdown and destroy the machine instance to avoid incurring unnecessary cost. Note that all of these three steps (provision, run, and destroy) need to be executed, even if the “run” step fails (e.g. failure to fetch the data for a hurricane due to network issues).

The workflow system is devised of 5 major steps:

1. Gathering hurricane information
2. Creating simulation domain and mesh generation
3. Setting up the numerical model inputs

4. Running the numerical model (simulation)
5. Post processing the results

Steps 1 to 3 together constitute the preprocessing step from the modeling process and steps 4 and 5 represent running solver and post processing, respectively.

After the simulation, the results are transferred from the short-term storage to the long-term storage dedicated for results. At this point if no other simulation is running, the short-term storage is completely wiped. Hence, the handling of the data that resides on the short-term storage requires synchronization between different Prefect runs, otherwise while one run is using the data, another might overwrite or remove it.

Modeling Tasks

As described in the introduction, the workflow is devised of 5 major tasks. In this section each of these tasks are described in details.

Gathering Hurricane Data

The first step in the workflow is to get the storm related data. This includes the storm track and wind swath information, as well as NOAA station measurements for validation of hindcasts. StormEvents is used in order to fetch this data from NOAA. StormEvents relies on NOAA NHC forecast or best-track data for obtaining storm lists, names and tracks, and on NOAA COOPS API to get the station measurements.

Mesh Generation

One of the highlights of this system is the dynamic generation of computation domain and unstructured mesh, tailored for a specific hurricane event. OCSMesh (Mani, et al., 2021) package, developed in NOAA OCS Coast Survey Development Lab, is utilized for this task. OCSMesh is a data-driven, automated coastal ocean modeling mesh preparation tool. As a part of implementing the system described in this manuscript, new features and capabilities were also added to OCSMesh.

The generation of domain and mesh is guided by the hurricane track and wind field radius. This ensures the regions impacted by that storm have higher resolution mesh (smaller elements) and are extended to the floodplain, while the rest of the domain has larger elements and is not extended beyond the coastline. Doing so reduces the computational resource requirements for running the solver (hydrodynamic model). In specific, the coastal region within the 34kt wind swath is taken as the region impacted by the storm (Pringle et al., 2022)

Mesh generation task processes many DEMs in order to generate size function for mesh generator, and after that to interpolate elevation on the mesh nodes. Because of this, it requires large amount of computer memory. This is provided through use of large memory nodes provided by the cloud services.

In the next section, the logic involved in creating the mesh from scratch is described. As mentioned in the introduction, there’s also an option to merge two existing mesh; the latter approach is much faster. The logic for identifying the region of storm impact is the same in the mesh-merge approach, the main difference is that it is executed on the mesh elements instead of shapes as described below. The details of merging the two mesh is not described in this manuscript.

Domain Definition

The computational domain is generated from the combination of 3 shapes:

1. Storm 34kt wind swath,
2. Full domain bounded by 0m contour (shape A), and
3. Full domain bounded by 10m contour (shape B).

All the three components are taken as input to the domain definition process. Note that items 2 & 3 are independent of the storm, which means they can be pre-computed. To get better results, the pre-computed shape bounded by the 10m contour uses more accurate and higher resolution DEMs. The two precomputed shapes are combined based on the region impacted by the storm (described next) to form the final domain. Figure 3 shows these three shapes for Hurricane Harvey (2017).

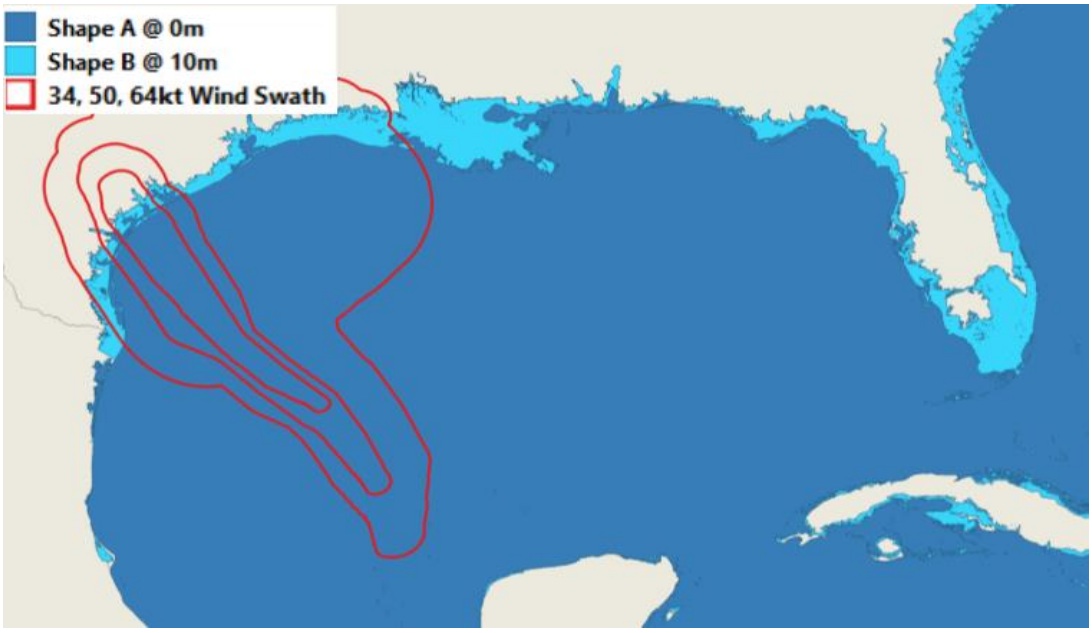

Figure 3 Depiction of the 3 shapes used to calculate the domain definition for Hurricane Harvey (2017)

Impacted Region

The impacted region is calculated based on the shapes described in the previous section. More concretely, shape B is intersected with the polygon formed by the exterior of 34kt wind swath (outer red ring in figures Figure 3 & Figure 4), then all the upstream basins of this shape are unioned with it. The impacted region will have higher resolution mesh, so in order to avoid small elements in open ocean, the calculated impacted region from the previous step is cut-off at a user-specified depth contour (e.g. 200m depth); this results in the region shown by the green dashed boundary in Figure 4. Then in the last step the final domain is calculated by executing union operation between the

impacted region and shape A. The final domain is shown by the yellow boundary in Figure 4.

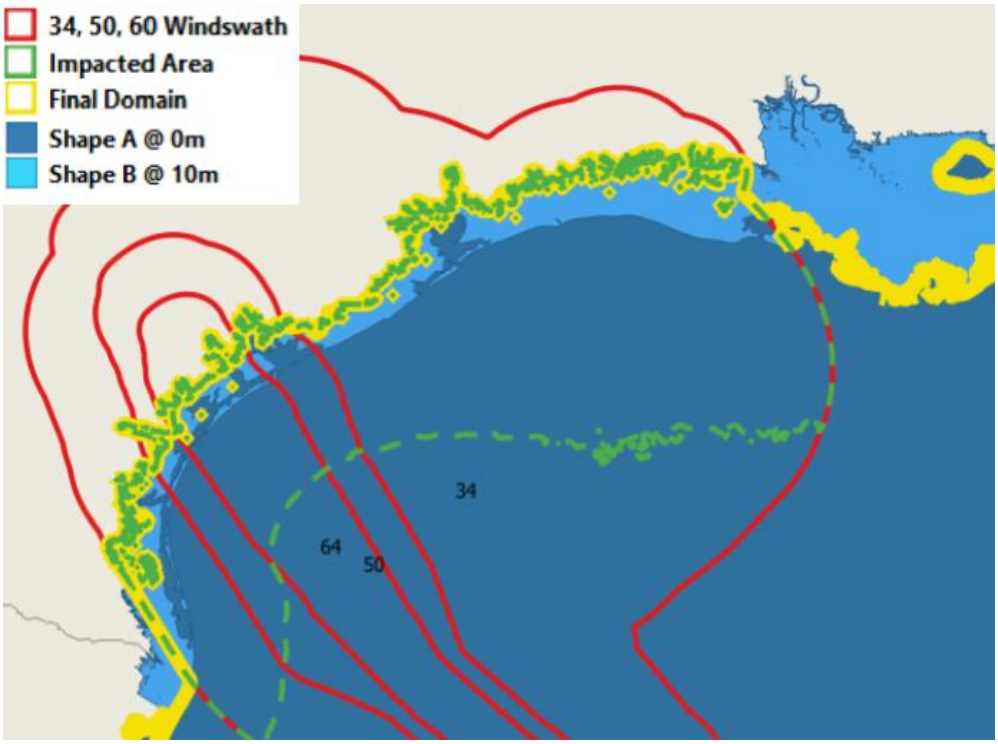

Figure 4 The impacted area of Hurricane Harvey (2017) as calculated based on 34kt wind swath. The green dashed boundary shows the calculated impacted area and the yellow boundary shows the final domain calculated based on shapes bounded at 0m (dark blue)

Size Function

Before proceeding to mesh generation, the element sizes must be specified. To achieve a smooth transition in element sizes, stepwise size refinement is applied both in the impacted region (high resolution region) and outside of it (low resolution region). The refinement is applied based on prespecified topography contours and expands based on user specified expansion rates. The maximum element size as well as minimum mesh sizes in both high- and low-resolution regions are also user specified. By default, the element edge maximum size is 20km and the minimums are set to 1500m and 300m in low- and high-resolution regions, respectively.

Figure 5 shows the result of the applied mesh size specification. The dark green region shows the smallest element size specification and red shows the largest size specified. In the impacted region (within the dark blue boundary) a size of 300m at coastline and all the way up on the floodplain is specified. For the low-resolution region, the coastline mesh size is set to 1500m.

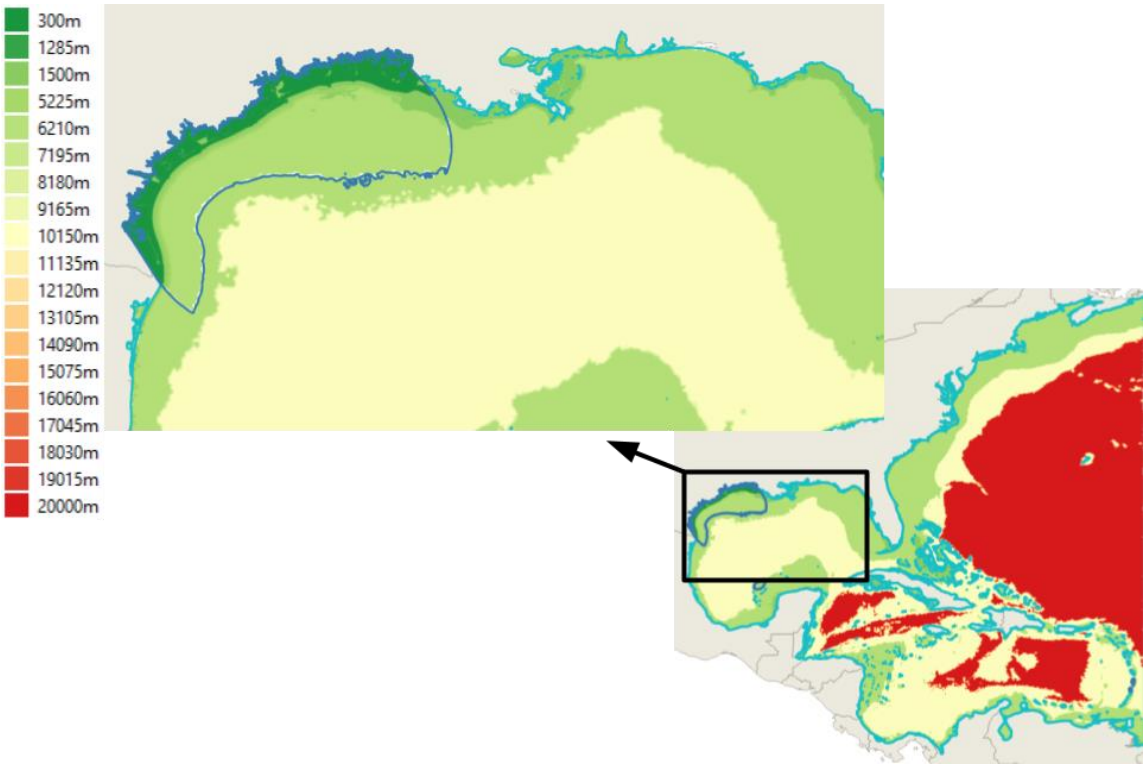

Figure 5 Mesh sizing function generated for Hurricane Harvey (2017)

Configuring Model Inputs

When the mesh is generated, the model parameters, forcing, boundary and initial conditions need to be specified for the solver. In this project, SCHISM is used in 2D mode for modeling storm surge. PySCHISM preprocessor is used to automated the model setup. It is a Python package developed for generating model inputs, such as tidal, atmospheric, and hydrologic forcing, as well as other inputs.

Boundary Conditions

Tidal Forcing

To generate input files for the tidal elevation and barotropic velocity of eight tidal constituents (S2, M2, N2, K2, K1, P1, O1, and Q1) TPXO dataset (Egbert & Erofeeva., 2002) is used; values are interpolated on the mesh's ocean boundary.

Bottom Friction

For the bottom surface roughness, a Manning's formulation is used with the coefficient *n* set to be 0.02 for open ocean and linearly increased to 0.05 as the bottom elevation changes from -1 m to -3 m into the watershed.

Initial Condition

In the current version of the workflow the initial condition for the water elevation is set to be zero in all areas with positive grid depths (i.e., water body) and equal to the bottom elevation in areas with negative grid depth (i.e., high ground over the watershed).

Atmospheric Forcing

With PySCHISM, the surface meteorological forcing (wind, air pressure and temperature, precipitation, humidity, shortwave and longwave fluxes) can be set to a combination of coarse resolution grid (sflux1 covering whole domain) and high-resolution grid (sflux2 covering a sub-domain of sflux1). ERA5 reanalysis product from the European Center for Medium-range Weather Forecast (ECMWF) with a horizontal resolution of about 9 km, and the Global Forecast System (GFS) with a horizontal resolution of about 28km can serve as the type 1 sflux. The NOAA High-Resolution Rapid Refresh (HRRR) model can serve as the type 2 sflux; HRRR is a real-time 3-km horizontal resolution, hourly updated, cloud-resolving, convection-allowing atmospheric model. For dates prior to year 2020 PySCHISM capability to download GFS & HRRR results is not implemented. Hence, ERA5 results are used for all hindcast runs (even for 2022 storm hindcasts) that simulate historic storm events; e.g. for Florence in 2018. However, for forecast (now and into the future) GFS & HRRR results are used.

Other than sflux type atmospheric forcing, SCHISM model supports parametric wind modeling by integrating PaHM (Velissariou, 2021). In case of a parametric wind run, the hurricane best track data (downloaded in the first workflow task) is used to set up the model.

Hydrology Forcing

In order to set up the hydrology forcing, river discharges from NWM are introduced into the model at its land boundary by PySCHISM. Intersection points between computational mesh land boundary and NWM river segments are identified and the river discharges are added as volume sources in the relevant cells.

Solver

SCHISM is the solver used for modeling storm surge in this workflow system. It is a highly flexible modeling system using hybrid unstructured meshes in the horizontal dimension and a versatile hybrid vertical system in the vertical dimension (Zhang Y. , Ye, Stanev, & Grashorn, 2016). Together with a semi-implicit solver, the model performs robustly and accurately across multiple scales from small creeks to deep oceans. The model has been successfully applied to compound flooding studies nationally (Zhang Y. , et al., 2020; Moghimi, Myers, Pe'eri, Zhang, & Ye, 2021; Ye, et

al., 2020; Ye, et al., 2021; Huang, et al., 2021) and also in Japan. SCHISM also incorporates a simple Holland parametric wind model (Velissariou, 2021), thus facilitating ensemble simulations.

Similar to mesh generation, running the solver also requires a lot of computational resources. Unlike mesh, this task requires more processing power, and not memory. The resources are provided by using compute optimized nodes from the cloud service provider.

Visualization

The last workflow major task is processing the results and generating visualizations from it. This project provides two ways of achieving this. The first uses a Python script as an integral step of the workflow, and the second is a fully interactive visualization environment that runs independent from the workflow system.

Static Report File

The first method uses a simple Python script to generate an HTML report in the form of an explorable map with result and station time series plots. This visualization provides the basic tool for glancing at the result and qualitatively comparing it with measurements to assess the accuracy of the simulation.

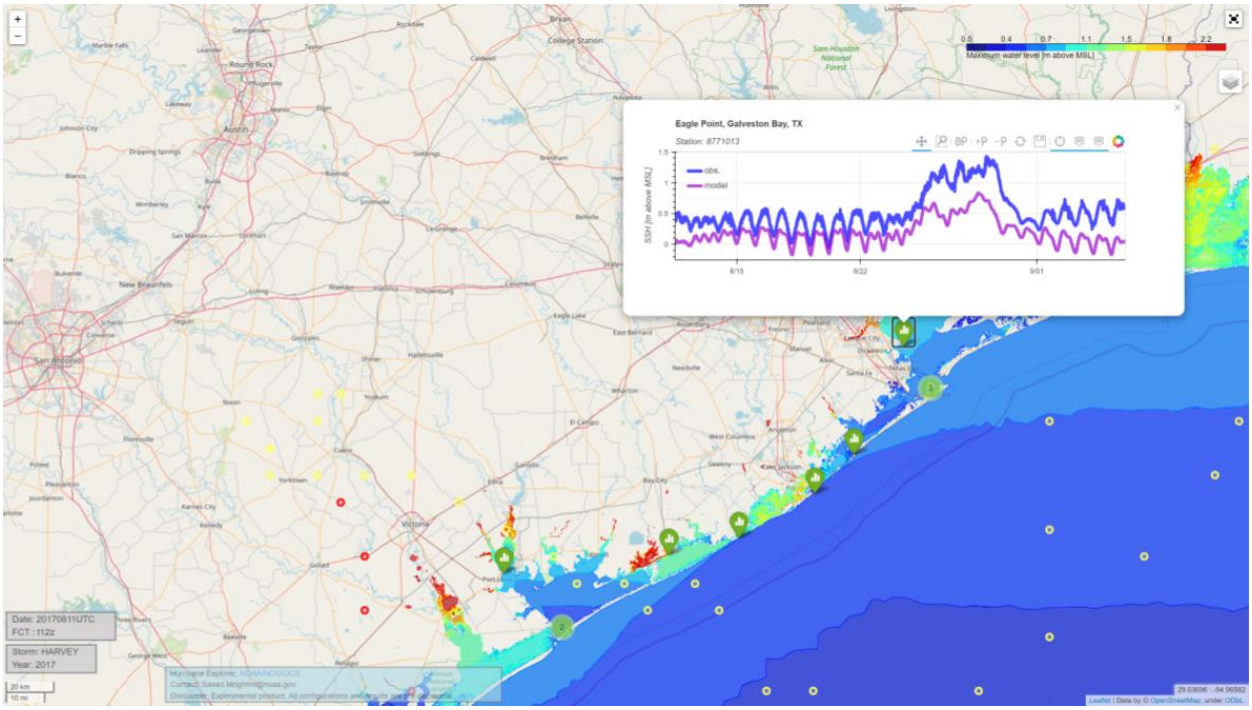

Figure 6 Interactive map showing maximum elevation contour and bias corrected time series plot for the simulation of Hurricane Harvey (2017)

Thalassa

Thalassa, developed by the EC-JRC and extended by VIMS developers, is a Python package for visualizations of large-scale hydrodynamic simulations. The interactive dashboard provides options to choose the dataset, variables and layers to render. At the moment, it is capable of viewing contour and time-series plots.

This package is not a part of the workflow; it runs separately on a server for exploring the final results from the workflow.

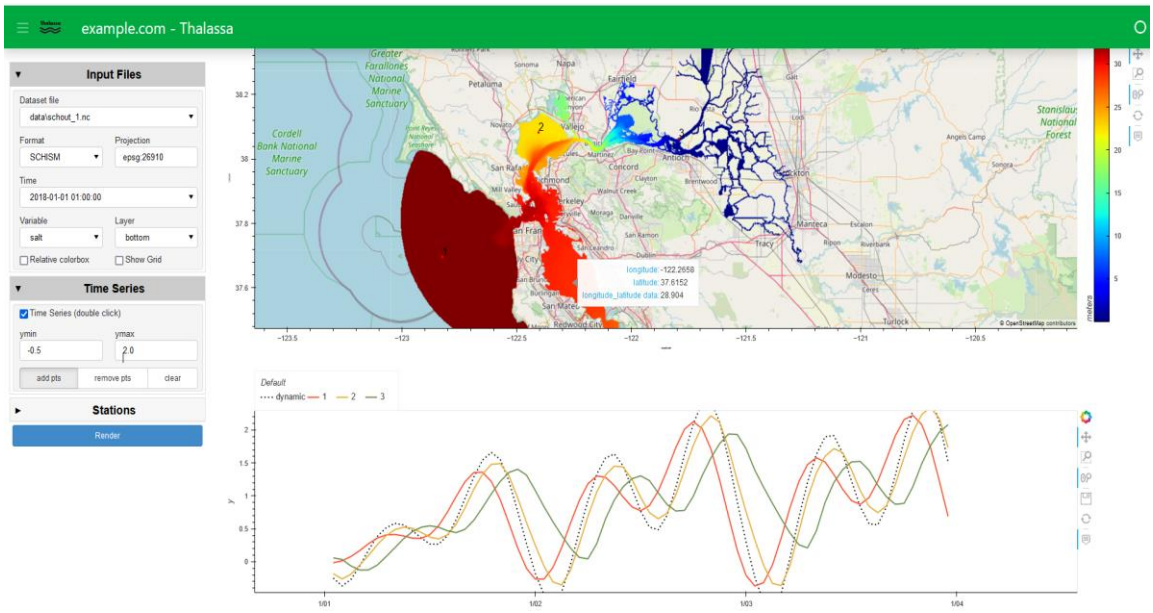

Figure 7 A snapshot of the Thalassa web UI showing SCHISM output variable contours on an interactive map as well as multiple station time series.

System Description

So far, a high-level description of how the system works has been provided. In this section, the technologies and tools that are utilized and the finer details of the system are discussed.

Tools and Packages

Terraform

One of the foci in this project is having the ability to reproduce the results. This led the project to use declarative infrastructure configurations and Infrastructure as Code (IaC) paradigm, so that it can be redeployed on some other account and get the same result. To that end, Terraform is used to manage the system backend. Terraform is a package developed by HashiCorp to declaratively define infrastructure in a reproducible way.

In this project Terraform sets up AWS services as well as local files required for configuring the backend and defining the Prefect workflow.

AWS Services

The system is implemented based on AWS cloud services. It includes use of different storage types and different types of compute instances and container services. AWS backend is defined using Terraform. This way the state of the system infrastructure is stored on a state-file accessible by Terraform, which makes tracking and modifying the infrastructure state much easier. In later sections specific AWS services that are used will be described in detail.

Local Files

Applying Terraform configuration results in creation of new resources, some of which need to be referred to or handled in subsequent stages of the workflow. Each of these resources have different types of identifiers, such as names, IP addresses, etc. Terraform already keeps track of all these identifiers and addresses. In order for other programs to have access to this information, these identifiers need to be written to a configuration file. Terraform's "LocalFile" provider allows for outputting tracked states. This capability is utilized to generate two local configuration files that pass the necessary information to Ansible and Prefect. This local file gets updated based on the state of other resources defined by Terraform.

Ansible

While Terraform is responsible for provisioning the infrastructure, it does not provide a modular way to configure the provisioned resources. For applications that require

additional configuration or libraries, Ansible provides the ability to do so in an idempotent and reproducible manner. Specifically, Ansible is used to automate the configuration, and orchestration of infrastructure and applications via human readable YAML text files.

In this workflow Ansible is used to configure the provisioned EC2 instances by installing libraries and running necessary services. Ansible is also used to configure the machine and deploy the Thalassa application on a separate server.

Prefect

From the onset, this project aimed at having the flexibility to be deployed on a range of compute infrastructure providers such as bare cloud environment, or managed cloud platforms. There are two general approaches available to facilitate such dynamic run-time flexibility: implement intersystem message passing to coordinate tasks and disparate orchestration systems, or use an orchestration service that has such a capability built-in. Given the complexity of the first option, we opted to use the cloud-hosted orchestration service and library Prefect.

Prefect can be thought of as the next generation of Apache Airflow. It has a number of features that align well with the constraints of the project. Prefect tasks and workflows are defined in pure Python code allowing for rapid implementation and iteration. Enabling features of the library include built-in support for cloud providers, container support, logging, a web-based UI, and the ability to define and launch ad-hoc workflows. The hosted Prefect service provides all of the features of the base library in addition to workflow monitoring, team management, and built-in multi-environment support.

How Prefect Works

Prefect workflows are defined in terms of **Tasks** and **Flows**. Each Flow is a directed acyclic graph. Nodes of this graph are different Tasks. Tasks could succeed, fail or be skipped based on the context of the Flow and results of the Task. The status of a Task then determines what subsequent Tasks must be executed or skipped. Finally, the **reference Task** decides what the status of the overall Flow is. Prefect has built-in Tasks for various kinds of user needs (https://docs.prefect.io/api/latest/); on a very basic level it supports executing shell commands on the machine handling the Task (more on Task placement later). On a higher level it has built-in Tasks for handling cloud infrastructure, executing Python code or talk to external databases. In more complex scenarios a Flow can be dependent on other Flows (https://docs.prefect.io/core/idioms/flow-to-flow.html) where an intermediate Task represents the success of the child Flow.

Prefect uses a client-server model. In Prefect 1.x, the Flows need to be registered with a server which manages the sequence of steps through Task and Flow metadata. Then

when a flow-run request is received by the server, it finds the right Prefect client (or agent) and passes the flow and its tasks to that agent to execute. Both the defined Flows and available agents can be labeled by the user; these labels are used by the server to match Flow execution to the agents. Utilizing this feature, one can configure Flows to run on different agents each of which could be running on different infrastructure and has different runtime environments and capabilities. This capability is used in this project to support **hybrid** workflows where sections of the workflow run on a platform separate from the main AWS account (figure Figure 8).

Docker

Docker is a containerization platform. It provides tools to package application runtime environments into container images. These images can be thought of as a frozen state of a system at the operating system (OS) layer. This provides the ability to develop, test and deploy applications independently from the machine they're running on. This capability makes containerization a perfect choice for running applications in the cloud environment.

Infrastructure Providers

One of the main benefits of this system is the on-demand nature of it; the infrastructure is provisioned as needed. Currently the infrastructure requirements are satisfied by two providers: AWS cloud and ParallelWorks managed platform (through NOAA RDHPCS cloud). Both of these platforms provide on-demand machines to execute the tasks. The workflow was originally implemented for a single AWS cloud, then ParallelWorks hybrid option was later added. In the case of hybrid workflow, the computationally expensive tasks, such as mesh generation or numerical model are executed on ParallelWorks. This section describes the services used from each provider and how that fits into the workflow implementation.

Amazon Web Services

AWS provides the backbone of the implemented system. It provides the long- and short-term data storage services for simulation data, registry for Docker images, and the machine instances for workflow tasks as well as for workflow management Agents. This section describes all the services used for the current implementation of the workflow system.

Using AWS services requires specific permissions for each service on the cloud provider account. The focus of this project has been to implement a working system; as a result, the permission requirements gathered for running the workflow is very broad

and needs to be revisited if this system is to be used in a production setting and on a production account.

The following sections itemize the services declared and used in this project.

S3 Storage

Simple Storage Service (S3) is an AWS service that provides object storage on the cloud, also known as buckets. S3 buckets have a variety of tiers for different archiving needs, but in general the basic tiers are very useful for long-term storage of data. Three S3 buckets are used for this project:

- The first bucket for storing the registered Prefect Flows as well as the metadata associated with the results of running the Flow.
- The second bucket for storing static data used by the workflow. The static data in this context means the data that is needed by all flow-runs and that doesn't change over time, or rarely does. Static data needs to be transferred to the bucket as a separate manual process for setting up the system.
    - Static data include DEMs, TPXO and NWM hydrofabric data as well as precomputed domain shapes and cached values from previous runs.
- The third bucket is used for storing the final results of the run, including the visualizations generated during the workflow run.

Elastic File System (EFS)

Elastic File System is another storage service provided by AWS. This type of storage is providing a file system as opposed to object storage (as in S3). This type of storage is a kind of Network File System that can be attached to a subnet and used by multiple machines. Because of this, it is a good candidate for the project's short-term storage during each workflow run. One EFS device is used in this system for storing transient data; the data that is generated and used during each flow run and is cleaned up after the run.

EFS is elastically resized based on the amount of data. The cost of EFS storage, then, is based on how much space is used. Hence it is good practice to remove temporary data from EFS when there's no need for that data after the flow run.

Another good alternative to EFS for transient data storage, is using the FSx as Lustre files system which has faster IO rate. The upgrade to Lustre file system is an optimization step which is outside the scope of the current project.

Elastic Compute Cloud (EC2)

Elastic Compute Cloud (EC2) is an AWS service that provides virtual machines (a.k.a. instances) of various sizes and computational powers to match the required computational needs. These instances can be launched with a variety of preconfigured operating systems, such as different flavors of Linux.

For this project EC2 instances are used for two different roles: 1) instances that run Prefect agent 2) instances for running simulation tasks Docker containers. The manager instance/node only needs to have enough resources to be able to handle running multiple Prefect Flows concurrently. The instances used for Docker tasks need to have larger memory and processing power based on the task being executed. For example, the task of meshing requires large memory, but the solver task requires more processing cores.

Currently for Prefect agent a `t3.small` instance type is used. This is large enough to handle 3 to 4 concurrent Flows. For larger number of concurrent flows, either a higher capacity instance needs to be used, or the Prefect Flows can be executed as ECS tasks on a Prefect ECS agent (details are beyond the scope of this report).

Launch Templates

Launch template is a capability for EC2 instances. As the name suggests, it is useful for launching multiple instances with the same predefined configuration. In this project launch templates are defined for different Docker tasks, so that the instance can be dynamically created by just referring to the template name.

At the time of this writing the following instance types are defined as template:

- `m5.8xlarge` for mesh generation
- `m5.xlarge` for gathering hurricane data and model configuration
- `c5.metal` for running the solver
- `c5.xlarge` for post-processing and visualization
- `c5.4xlarge` for ECS agents

The description of the instances can be found at: https://aws.amazon.com/ec2/instance-types/.

These templates are referred to by name in Prefect Flows to dynamically and automatically create and destroy instances as needed to achieve a basic scaling of the workflow to multiple concurrent runs.

Elastic Container Service (ECS)

One of the benefits of using a containerized environment is that usually cloud service providers can launch Docker containers from the images as a service. AWS Elastic Container Service is such a service.

This project uses ECS for running the containerized modeling tasks. Long-running jobs, such as mesh generation, cannot be executed on ECS Fargate provider. In these cases, EC2 instances need to be selected as “capacity providers”. Note that in this case although the EC2 instances are managed by Prefect, AWS sets up and configures the environment, so there’s no need for additional EC2 configuration.

To use ECS, one needs to define the following components:

- **Cluster:** Clusters are logical partitions for separating tasks and services in ECS. Capacity providers are assigned to clusters to run the tasks that are executed on a given cluster. Using clusters is especially useful when different capacity is needed for different tasks. For example, capacity needed for running mesh generation tasks are different from the capacity needed for fetching hurricane data.
- **Tasks:** ECS tasks (not to be confused with Prefect Tasks) are set of instructions to run Docker containers on ECS. This includes information such as what volumes to mount or what Linux capabilities to use. ECS tasks can also support overrides during launch to modify run-time parameters. In this project, one ECS task is defined for each Docker image; that means one ECS task for each of the five modeling steps.

Scaling

In this project the ECS auto scaling is **not** used. ECS auto scaling works by ensuring that there’s always enough capacity to run a new job; if a task is submitted and there’s not enough resources to run it, the task fails. So, there should always be more capacity than what is being used. In the context of high-performance computing, usually the reverse of this is implemented; jobs are queued and then based on all submitted jobs resources are allocated, if there’s not enough resources then the job waits in the queue. This queuing capability exists in the form of another service in AWS, called AWSBatch. However, exploring that technology is outside the scope of this project.

For supporting multiple concurrent jobs and resource management on ECS, a “manual” approach has been implemented. In this context manual means that the Prefect Flow embeds the logic of launching and destroying the instances; before an ECS task is started, Prefect creates an instance (from the template) exclusive to that task, then runs the task, and after that destroys the instance. Note that the instances that run ECS

tasks are exclusive, this is set up to avoid terminating an instance from one Flow-run when another Flow-run is executing its own task on it. In summary, Prefect manages instances as well as which task goes to which instance.

Task Definition

In this project ECS tasks are defined by Terraform; this has some downsides. Since the version of the Docker image used by the task is specified in the task definition, when images are updated the Terraform configuration needs to be reapplied. This makes switching between different versions of the workflow very cumbersome, where the developer needs to make sure configurations for Terraform and Prefect are applied in the right order. In later iterations versioning of tasks can be moved to workflow definition instead.

Elastic Container Registry (ECR)

Docker images that are used in the ECS tasks need to be uploaded to a registry accessible by ECS services. This can be done by either using a public registry like DockerHub or by using AWS ECR service. This project chose ECR because of the security requirements of scanning uploaded images for vulnerabilities.

Networking Components

AWS provides services to define the network and subnets on which other services (such as compute nodes, EFS devices, etc.) are placed. These services include access control lists and permission on using different ports. These networking components are a prerequisite for using any of the AWS services. In a production environment, these services are usually set up by account administrators and IT security personnel and are not up to the developer to decide how to structure it. In terms of the requirements for this project, the networking components should:

- Be able to satisfy requirements for AWS Elastic Container Service (ECS) which is used to run modeling tasks
- Provide public IP for manager node used to the main run Prefect Agent
- Allow EFS traffic and EFS mount target
- Allow SSH access to the manager node

Concretely, these components include Virtual Private Network (VPC), its subnets, security groups and network ACLs.

NOAA RDHPCS ParallelWorks

So far only the use of AWS services for the workflow have been discussed. ParallelWorks platform is a managed environment that provides virtual HPC like

environments on multiple cloud providers. Being managed means that the user of the platform doesn't need to worry about the details of setting up the HPC environment and the inner workings of the infrastructure. In contrast, using a cloud service provider directly requires more in-depth knowledge of how infrastructure needs to be set up. In ParallelWorks, using standardized configurations, one can set up similar environments with different cloud service providers. Moreover, this platform provides an HTTP API which can be accessed via user-specific tokens to control the state of the virtual clusters and running jobs. These capabilities make ParallelWorks a very good option for multiplatform (or hybrid) workflows. The implemented hybrid workflow passes the computationally expensive parts of the workflow to ParallelWorks, but executes the rest of the workflow directly on the AWS infrastructure.

Clusters in ParallelWorks are "project clusters", meaning that its resources are not shared with others outside the project. In the current implementation of the hybrid workflow, a Prefect agent is started on the head node of the ParallelWorks cluster. This Agent will handle starting Slurm jobs and waiting for their final status.

Two separate ParallelWorks clusters are used in this project; one for mesh generation jobs and one for SCHISM runs. The reason behind using two separate clusters was to keep environments separate. This keeps the environment separation similar to ECS clusters.

ParallelWorks provides a project specific S3 bucket. This means the workflow system is actually using four S3 buckets. Currently the static data is duplicated in a ParallelWorks bucket; this data is copied to the cluster during cluster initialization. Later this must be revisited to make better use of storage space and data sharing capabilities between the platforms. Workflow-run specific data is shared between the AWS part of the workflow and ParallelWorks through the ParallelWorks S3 bucket.

The hybrid workflow is invoked by setting the proper flag parameter in Prefect when running a new simulation Flow run. Currently the HTTP API doesn't support defining a cluster, the clusters need to be predefined from ParallelWorks web portal and then HTTP API can be used to access and manage them.

In current iteration of the workflow development, the run environment for tasks is not containerized in ParallelWorks. The environment binaries are manually packaged in persistent storage and fetched during cluster initialization.

Workflow Management

Now that all the components of the workflow system are described, let's dive a little deeper into the details of how the workflow is executed. To set up the system, first

Terraform and Ansible provision and set up the backend on the AWS account. The details of what services need to be set up are described in the infrastructure providers section. For hybrid workflows, ParallelWorks clusters need to be defined as well.

At this stage (assuming ParallelWorks clusters have been started as well) five Prefect agents should be ready for executing Flows; three agents on the AWS EC2 and one on each of the two ParallelWorks cluster head nodes (figure Figure 8). Different Agents on the EC2 instance serve different purposes; first one is the main workflow manager agent, the second one is responsible for handing off data and instructions to ParallelWorks clusters, and the third one is responsible for executing Flow-runs as ECS tasks. The agents on the cluster head nodes, as described before, are responsible for execution and management of Slurm jobs. One thing to note is that ParallelWorks clusters don't actually need to be on from the start; Prefect flow starts them automatically if they are off. In that case the agents on those head nodes are started as soon as the cluster is started and initialized.

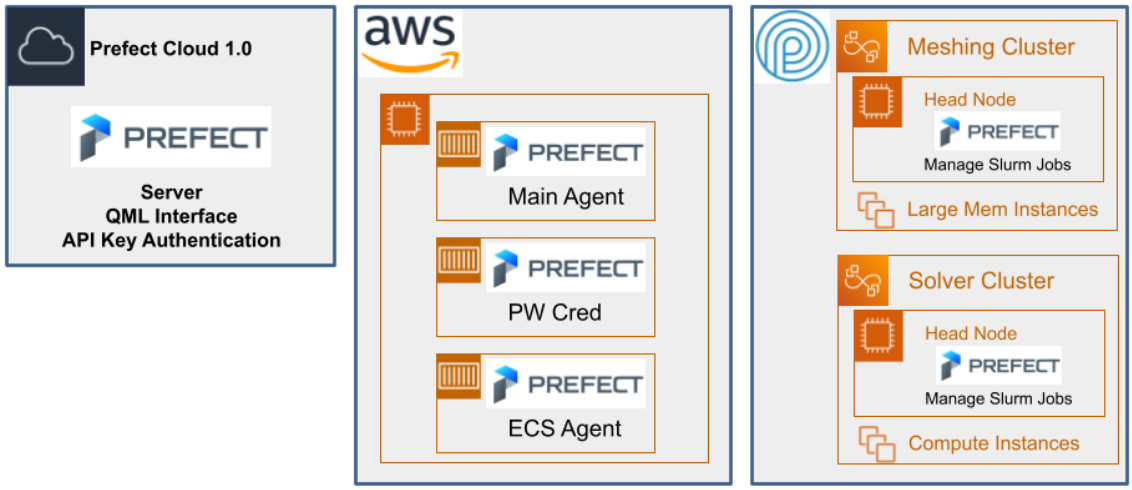

Figure 8 Arrangement of Prefect agents for the workflow system

With the system initialized and ready for running the workflow, this is how it handles the simulation instances:

1. The user specifies the simulation parameters, such as hurricane name and year. These parameters include exposed mesh parameters discussed earlier, as well as workflow control flags such as whether to use hybrid workflow or whether to use parametric wind or not.

2. The shared static data required for all the flow-runs is synchronized between static S3 bucket and EFS. Note that all simulation tasks have shared access to the EFS device. The EFS is also shared between different flow-runs. Because of

this shared access, a mechanism is put in place to avoid data corruption when multiple concurrent Flows are running at the same time.

3. After static data is copied, a directory is created in the EFS to store the data associated and specific to the current run.

4. An EC2 instance is created and the first modeling task is executed on it. The first modeling task is gathering hurricane data, including track, wind swath and station measurements. This data is copied to the EFS storage. After the completion of this simulation task, Prefect terminates the EC2 instance.

5. The next simulation task is the mesh generation. There are two ways to execute this task; through ECS task on a large memory instance on AWS or through ParallelWorks cluster. This decision is made by the input parameters specified by the user at the beginning of the process.

    a. For hybrid workflow, first the run data is copied from EFS to a ParallelWorks S3 bucket. This requires the transfer agent to have credentials to for both EFS on the AWS account as well as the bucket provisioned by ParallelWorks. After that, the agent on the head node of ParallelWorks cluster copies this data from the S3 to the cluster's FSx file system and starts the Slurm job for mesh generation. After mesh generation the results are copied to the same S3 bucket which is then copied to EFS (figure Figure 10).

    b. For non-hybrid workflow, similar to step 4, an instance is created exclusively for the task, the task is executed on that instance and then the instance is destroyed (figure Figure 11).

6. After mesh generation the ECS task to execute model set up is started. Similar to previous ECS tasks, this task will run on its Prefect managed exclusive EC2 instance which is destroyed when the task is completed.

7. Next is the solver task. Similar to mesh generation, this task can take one of the two routes depending on user input.

    a. For hybrid mode, the task is handed off to ParallelWorks cluster after data transfer similar to that of step 5. Since ParallelWorks provides an HPC like environment, the solver can benefit from running the MPI job on many nodes in the cluster (figure Figure 10).

    b. For cloud-only mode the ECS task for solver is executed. Note that due to technical issues and restrictions, unlike running on ParallelWorks, the

ECS task cannot run using multiple nodes. That means the best way to make use of MPI parallelization of the solver is to use an EC2 instance with many cores. Similar to other tasks, this EC2 instance is created and destroyed by Prefect exclusively for the use by this ECS task (figure Figure 11).

8. In the last modeling task, the raw results from the solver are used to generate a dynamic map for comparing station data and model results as well as contours for maximum water elevation. This step runs as an ECS task similar to the previous ones.

9. If all the steps of the modeling are successful, the result files are copied from the EFS storage to the results S3 bucket.

10. The last step is to clean up the EFS storage. The run specific folder is first cleaned up, then if there are no other run directories in EFS structure the static data is also removed.

Figures Figure 9, Figure 10 and Figure 11 show the process described above.

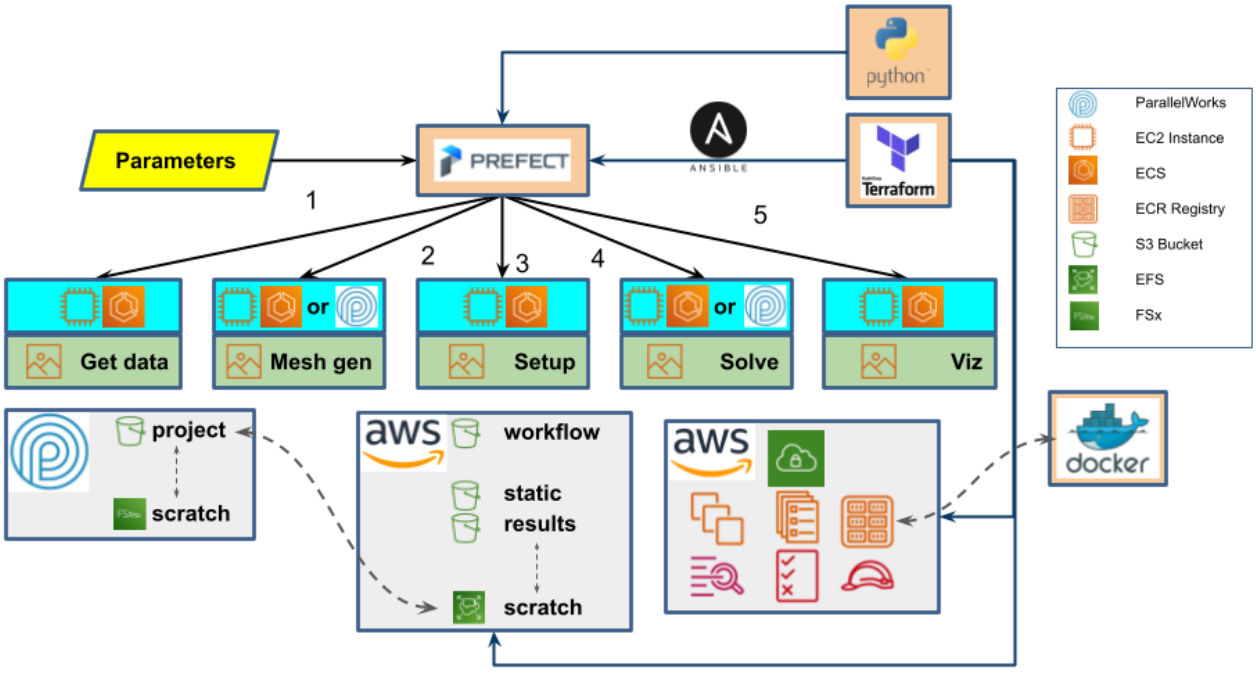

Figure 9 Detailed overview of the workflow

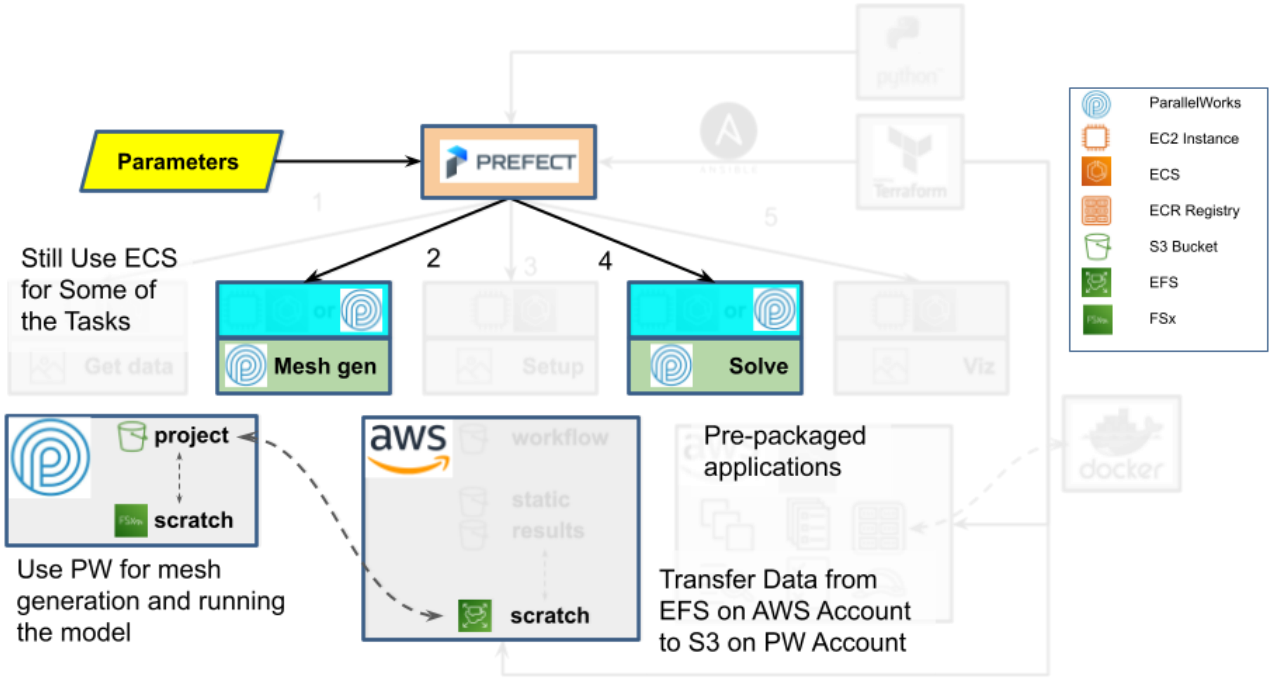

Figure 10 Workflow execution on PW in hybrid mode

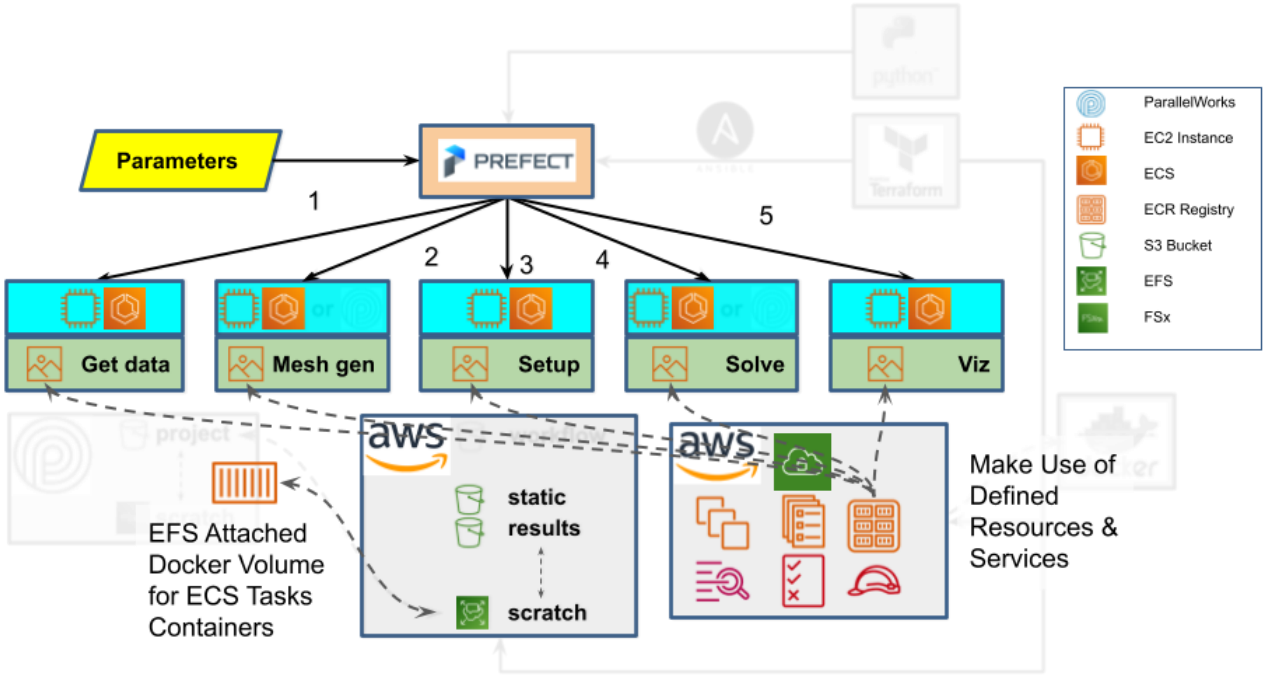

Figure 11 ECS-only workflow execution

There are some more details about specific aspects of the workflow which are described in the following subsections.

Building Docker Image

In the current implementation of the system Docker images are built and pushed to the registry manually. Six different images are used, five images for modeling tasks and one image for Prefect agent. When images are built and tagged, the Terraform configuration must be updated as well to match the new image tags.

Syncing between Flows

As described in the workflow management section, during certain tasks, data shared between concurrent runs might be modified, leading to data corruption. For example, this can happen during static data initialization or run cleanup. In essence, the issue is multiple processes competing to modify the same resource; this is known as race condition in programming. In specific, two concurrent runs might try to copy S3 bucket data to EFS, causing unnecessary redundancy. Alternatively, it is possible that while one run is wiping out all the data from the workflow EFS, another tries to copy S3 data into it. One could think of other scenarios for this.

To avoid these kinds of race conditions, a simple FLock mutex mechanism is utilized. Newer versions of Linux FLock support network file systems (NFS), such as EFS. During the critical operations a lock file is created and locked on the EFS to prevent other critical operations from being executed. Since not all critical operations have conflict with one another, multiple lock files can be introduced and each pair of conflicts can be handled using one of these files. In the examples above, in the case of clean up vs static data conflict, if the clean-up task is executed first, it cleans up the static data and then the other run will copy the static data back again! Although this makes the process less efficient, it makes it robust; i.e. we don’t run into a case where one run removes all static files while the other run assumes the static files exist.

Results Caching

When running multiple simulations of the same storm event with different parameters, such as different mesh sizes, some of the data downloaded or generated during the modeling process can be reused. For example, atmospheric forcing from ERA5 (used for hindcast runs) can be stored and reused for the same storm runs later. This is called data caching. Right now, it is only implemented for atmospheric forcing, however, this is a useful concept to expand to other data files, such as mesh or hydrology or tidal forcing.

The cached data is stored along with the rest of static data files on the S3 bucket. This ensures that during each run newly created cache files are copied to EFS storage for use. Note that this requires some level of data transfer synchronization as discussed earlier. During the model setup, if cache files are found on the EFS, they are copied to the model input directory. However, if no cache file exists for a specific storm, the files are generated from scratch and copied to a shared cache location on the EFS. At the end of the simulation tasks and before the clean-up, the cached data is copied to the static data S3 bucket.

Caching requires some way of identifying if the cached data of interest for the simulation exists. Currently for the atmospheric forcing the unique identifier is a hash of the domain bounding box along with atmospheric data (sflux) start and end date. For caching other types of results, different criteria might be required for creating a unique identifier.

Static Data

In this document there are a lot of references to “static data” and its S3 bucket. There are some datasets and input files that are required by all of the simulations and don’t change between simulations or over time (or rarely change), but downloading them can be time consuming and inefficient for various reasons. This data is thus stored on an S3 bucket and copied over to EFS for every run. This can actually be viewed as some form of caching! The following is what falls under the category of static data:

- **Topography and bathymetry data in the form of DEMs:** All the DEM tiles used in the project are publicly available, however the size of the data and the download speed from their providers make them good candidates for S3 storage.
- **Forcing (NWM and TPXO) data sets:** The model-setup task uses specific datasets for generating the model inputs such as forcing or boundary conditions. These datasets are stored on the S3 bucket.
- **Precomputed shapefiles:** As discussed in the mesh generation section, two precomputed shapes are used for dynamic mesh domain calculations. The process of recalculating these shapes are very time consuming and the shapes won’t change frequently over time. These shapefiles are stored on the S3 bucket.
- **Existing domain mesh:** For mesh generation by merging existing mesh, these mesh files need to be stored on the S3 bucket.
- **Cached results:** The data cached for reuse by other runs are also transferred to the static S3 bucket at the end of a given run.
- **Miscellaneous:** Other files such as static configuration files or other similar data are also stored on the static S3 bucket.

Note that obtaining, generation or transfer to the S3 bucket of static data (except for “cached results” category) is manual. The workflow administrator needs to download the

DEMs and forcing datasets and transfer them to the S3 bucket. Precomputed shapefiles need to be calculated in advance and copied to the bucket as well. In later iterations of the system some automation might be considered for this process.

Code Organization

The system described in this document is implemented using Python scripts, Ansible and Terraform configuration files, and Dockerfiles. These files are organized by technology in a repository on GitHub at https://github.com/SorooshMani-NOAA/odssm-infra. The system is still in the development phase, but the latest working iteration of the workflow code is available in the `main` branch.

In the root of the repository a Conda `environment.yml` file defines the environment needed to set up the workflow; this includes packages needed to generate Prefect artifacts and registering it with the Prefect 1.x server. This environment needs to be kept in sync with the environment defined for Prefect agents in the relevant Dockerfile. The following sections describe the contents of the root directories in the repository.

Terraform Configuration

Terraform HCL script files reside in the `/terraform` directory of the repository. Currently all the resource definitions reside in a single `main.tf` file. In future iterations this file will be split based on the resource categories, e.g. `compute.tf`, `storage.tf`, `networking.tf`, etc. The Terraform output is also defined in this directory (`output.tf`). ECS cluster configuration files for EC2 Launch Templates reside in this directory as well (i.e. user data files at `/terraform/ud/userdata-*.txt`).

Ansible Configuration

Ansible configuration files including playbooks, inventory, variables, collections, and roles are stored under `/ansible` directory:

- `/ansible/playbooks/provision-prefect-agent.yml` for defining how to set up the manager EC2 and start the Prefect Agent
- `/ansible/collections` for plugins used in the playbook
- `/ansible/requirement.yml` for defining ansible plugins
- `/ansible/ansible.cfg` for defining path to other ansible configuration files

The rest of the files in Ansible directory are no longer relevant to the current implementation and need to be cleaned up. Note that the playbook relies on the definition of some variables. These variables are defined by the local file generated after Terraform configuration is applied.

Prefect Script

The Python script that defines the Prefect workflow can be found under the `/prefect` directory. The workflow is implemented as a Python module named `workflow`.

In /prefect/workflow directory, conf.py defines the workflow constants (such as paths, names and values) and reads dynamically generated variable files from Terraform. These values are used throughout the Python scripts. Workflow root directory also includes a pw_client module which is required for interaction with ParallelWorks API.

The rest of the Prefect code is organized into flows and tasks. The tasks submodule defines all the basic Prefect Tasks, such as executing a specific shell command, copying files from S3 to EFS or starting an EC2 instance. The flows submodule uses the Tasks defined in the tasks submodule and creates Prefect Flows. It also establishes dependencies between the Flows. Both the flows and tasks submodules are divided into files that include code for the same type of actions. For example, Tasks are split into data management (data.py), infrastructure management (infra.py) or job management (jobs.py) and so on. Similar split exists for Flow definition files.

The main.py script at the root of the workflow module combines all the Flows and generates the end-to-end Flow. Finally, it registers the Flows with the Prefect server.

Dockerfiles

Dockerfiles and their dependencies are stored inside the /docker directory in the repository. For each image, a separate directory is created which contains the Dockerfile as well as the dependencies. There's also a main directory which contains Docker-compose configuration that make it easier to build the images.

All the Docker images (except for the solver) are based on Conda-Alpine. The main reasons behind using Conda-Alpine are to avoid building the GIS libraries (installed by Conda) and also keep the images secure (Alpine has minimal attack surface). Since Alpine has a custom C library, the solver cannot be compiled on it.

ParallelWorks Configurations

ParallelWorks doesn't support remote provisioning and definition of resources. However, through its web portal, it supports textual declaration of the resources and initialization scripts. To keep track of how the virtual clusters are set up on ParallelWorks, the contents of these textual configurations and scripts are stored in the /rdhpcs directory. Apart from that, a copy of all the scripts used for running jobs on ParallelWorks clusters is also kept. This includes Slurm job definition or Python client scripts. The process of copying these scripts and configurations to ParallelWorks is manual.

Setup and Usage

This section describes the steps for setting up and using the system. It is assumed that no infrastructure has been set up in advance. Currently the workflow system doesn't support multiple administrators, one person needs to set it up and then others with access to the Prefect project can run simulations. Another nuisance in the current iteration is that the infrastructure names are hardcoded. That means two separate instances of the system **cannot** be provisioned on the same AWS account due to name conflicts.

Setting up the Accounts

This section assumes that the administrator or the developer has access to an AWS account, a ParallelWorks account and a Prefect account. This is a high-level overview of what to do; for the details about setting up tokens or how to get permission for specific resources, please refer to the provider's documentation.

AWS Account

As a standard practice, AWS users' access is secured by using multi-factor authentication (MFA). First, add an MFA device for your (Identity and Access Management) IAM User. After that, create an access key for AWS API. Make sure you note what your access key is, as it will be needed later.

If your IAM User doesn't have administrative access for AWS resources you need to work with your account administrator to get permissions required for this project. A super set of these permissions are documented, but are not included as a part of this report.

Prefect Cloud Account

This project relies on Prefect Cloud v1.x for workflow management. In order to use Prefect Cloud, a Prefect account is needed. In case a dedicated Prefect Server is deployed, there's no need for an account. Prefect Cloud's paid subscription provides the ability for multiple accounts to have access to a shared project; this is useful for setting up the system and allowing project contributors to also use the system.

Login to Prefect Cloud and create an API key for the account. Note what the API key is as this will be needed later as well.

ParallelWorks Platform Access

For hybrid workflow runs ParallelWorks access is needed. After logging in to ParallelWorks note your API access key assigned to your username.

Setting up the Local Environment

The next step is setting up the local machine's environment. This should only be done on a trusted machine to avoid compromising accounts and their credentials. This step consists of setting up secrets to allow API access to services such as AWS or Prefect, and installing packages that are required for the API access.

Packages

Terraform, Ansible as well as AWS CLI and boto3 packages are required to be installed for the system. All the package requirements are included in the Conda environment file within the root of the repository. Create the Conda environment from the YAML file, or manually install required packages with the version mentioned in the Conda file.

```
conda env create -f environment.yml
```

Installing Docker CLI is also required. Please refer to Docker documentation for setting up Docker on your platform of choice. Docker-Compose script needs to be installed as well to facilitate the process of building images.

Environment for AWS

To use AWS API on an account secured by MFA, obtaining a temporary session token is needed. To do so, a profile with the "permanent" access key ID and secret must be created and used along with the MFA code to retrieve the token from AWS token service.

To create a profile with the permanent keys and MFA serial (assigned to MFA device by AWS) Use:

```
aws configure --profile <my_profile_name>
```

To set up the permanent key's ID and secret as well as profile's default region and CLI output format for the profile:

- **AWS Access Key ID:** Obtained when creating access key in AWS console
- **AWS Secret Access Key:** Obtained when creating access key in AWS console
- **Default region name:** Region in which AWS resources are created by default of this profile

- **Default output format:** Output format for AWS-CLI commands, e.g. json, yaml, table

After this step, MFA device identification must be added to the profile by:

```
aws configure --profile <my_profile_name> set mfa_serial arm:aws:iam:<account_id>:mfa/<iam_user>
```

Then you can obtain the temporary credentials:

```
aws --profile <my_profile_name> sts get-session-token --token <the-6-digit-pin>
```

Using the output of this command you can now create another profile that has temporary access to AWS resources:

```
aws configure --profile <temporary_profile_name>
aws configure --profile <temporary_profile_name> set aws_session_token <retrieved_token>
```

During the development, this project used AWS US-East-1 region. All the Terraform configurations use this region.

Prefect and Ansible need to have access to the AWS resources as well. The temporary access credentials need to be set as environment variables to provide this access. Assuming a bash shell is used:

```
export AWS_ACCESS_KEY_ID=id_value_from_get_session_token
export AWS_SECRET_ACCESS_KEY=secret_value_from_get_session_token
export AWS_SESSION_TOKEN=token_value_from_get_session_token
```

To test if the environment works, get a list of S3 buckets on the account:

```
aws --profile temporary_profile_name s3 ls
```

Other Secrets

In order to use Prefect Cloud v1, a Prefect API key must be provided in the environment. This is done by either setting it as an environment variable:

```
export PREFECT_AGENT_TOKEN=<PREFECT_API_KEY_FROM_BEFORE>
```

and also, by authenticating the CLI:

```
prefect auth login -k <PREFECT_API_KEY_FROM_BEFORE>
```

Interacting with ParallelWorks from a remote machine requires two sets of credentials. In order to access ParallelWorks API, an API key must be provided and for accessing

the data stored on the ParallelWorks project S3 bucket an AWS permanent key ID and secret are needed. These keys and tokens can be obtained from ParallelWorks web portal. To set the local environment for Prefect and Ansible to interact with PW:

```
export RDHPCS_S3_ACCESS_KEY_ID=from_parallelworks
export RDHPCS_S3_SECRET_ACCESS_KEY=from_parallelworks
export PW_API_KEY=from_parallelworks
```

Another service used in this workflow that requires providing credentials is ERA5 for atmospheric forcing. This service is provided by Copernicus project. An account is needed to have API access to this service. Each account is assigned a key that need to be set in the environment along with the server URL:

```
export CDSAPI_URL=https://cds.climate.copernicus.eu/api/v2
export CDSAPI_KEY= <CDSAPI_UID>:<CDSAPI_TOKEN>
```

SSH Keypair

When provisioning EC2 instances, in order to provide SSH access for initialization as well as debugging, an SSH keypair needs to be specified. This SSH keypair can be created locally. Terraform script must be updated to refer to the public key of this keypair before provisioning any resources.

Docker Images

Currently Docker images are manually built and pushed to the ECR registry. As mentioned in the code organization section a Docker-Compose file is provided for convenience as well.

Build

In order to build the docker images, in a command line environment navigate to the /docker/main directory under the repository and execute docker-compose build. This will build all the images with tags given in the compose file.

Note that in the current iterations, post processing scripts are pulled from a private repository on GitHub. When building this image, access to this repo through an SSH key is needed. The private key address must be passed to the Compose command as an argument.

Push to ECR

In order to push the images to ECR, they should be properly tagged. The tags used on the images should match the ones used to define ECS Tasks in the Terraform file. To tag the images use the following tag format:

```
docker tag <local_tag> <AWS_ACCOUNT_ID>.dkr.ecr.<REGION>.amazonaws.com/<repo_name>:<version>
```

In addition to the requirement that tags match the ECR Task image names, the repository section of the tags need to match the ECR repository names defined in the Terraform file. To push images to ECR, Docker client needs to login to ECR using the following command:

```
aws --profile <temporary_profile_name> ecr get-login-password --region <REGION> | docker login --
        username AWS --password-stdin <AWS_ACCOUNT_ID>.dkr.ecr.<REGION>.amazonaws.com
```

Then the following command pushes the image:

```
docker push <AWS_ACCOUNT_ID>.dkr.ecr.<REGION>.amazonaws.com/<repo_name>:<version>
```

Note that the images can only be pushed after the ECR repositories are established by Terraform.

Setting up ParallelWorks Clusters

For hybrid workflow, ParallelWorks clusters need to be defined. The first step is to create the cluster from the web portal (refer to ParallelWorks documentation). Version 2 of AWS Slurm clusters in ParallelWorks was used for the system development. For each cluster (mesh cluster and solver cluster) in its configuration page copy the content of the relevant `_cluster.json` and `_lustre.json` files (in `/rdhpcs/clusters/`) to the appropriate fields. Then copy the content of `.sh` files into the user script field.

The details of packaging different tools are not discussed in this document. After the clusters are configured, the Prefect agent as well as mesh generation environments are created using Conda and packaged using Conda-Pack. Then these packaged environments are stored in the persistent `/contrib` directory. These packages are copied and activated during the initialization of the clusters, by the user script. The solver binaries are compiled separately and then are stored similar to the Conda environments.

Usage

Provision Backend

With the proper environment (described in the previous section), navigate to /terraform directory in the repository root and apply the configuration:

```
terraform init
terraform apply
```

The prompt asks you to verify changes. After inspecting the plan for changes in resources and verifying it's correct, confirm it to be executed. The required AWS cloud resources get provisioned and two local files are also written to the disk. These two files are automatically picked up and used in subsequent steps.

Upload Static Data

As mentioned before, the data on the S3 bucket for static files need to be uploaded manually. The data includes the following:

- dem/gebco/<gebco tiff files>
- dem/ncei19/<ncei 1/9th of arc tiff files>
- dem/ tileindex_NCEI_ninth_Topobathy_2014.zip
- nwm/NWM_v2.0_channel_hydrofabric/
- shape/base_geom/base_geom.{shp,*}
- shape/high_geom/high_geom.{shp,*}
- tpxo/grid_tpxo9.nc
- tpxo/h_tpxo9.v1.nc
- tpxo/u_tpxo9.v1.nc

Prefect Agent Setup

Applying Terraform configuration creates an EC2 instance which needs to be configured to serve as a Prefect agent. Navigate to /ansible directory under the root and execute:

```
ansible-playbook -i inventory/inventory ./playbooks/provision-prefect-agent.yml
```

Three Prefect agents should now show up in the Prefect dashboard on Prefect Cloud (or local Server).

Prefect Workflow Registration and Run

The next step is to register Prefect Flows. Using the same environment navigate to /prefect under the repository's root and execute the following commands:

```
python workflow/main.py register
```

This will register the workflow with the project in your Prefect v1.x server (or Prefect Cloud). Once the workflow is registered the system is ready to start Flow-runs. To start a simulation, simply execute

```
prefect run -n end-to-end --param name=<STORM> --param year=<YEAR>
```

You can pass other workflow parameters from the command line as well (e.g. `--param mesh_hmin_high=150`). You can watch the success/failure of the workflow in the Prefect web interface or by getting logs in the command line environment by executing:

```
prefect get flow-runs
prefect get logs --name <flow-run-of-interest>
```

Results

Simulation Validation

This section presents some preliminary results for hurricane simulation using this system with default parameter values. The simulation is a hindcast for Hurricane Harvey (2017). The input parameters used for this simulation are:

- Maximum element size = 20km
- Impact area's cut-off depth = 200m
- Low resolution region's minimum element size =1500m
- Low resolution region's expansion rate = 0.02
- High resolution region's minimum element size = 300m
- High resolution region's expansion rate = 0.01

Note that the meshing logic internally applies stepwise contour refinements as follow:

- 10km at 4km depth
- 6km at 1km depth
- Low-res min size at 10m depth
- High-res min size at 0m depth

The result of mesh generation using these parameters can be seen in Figure 12:

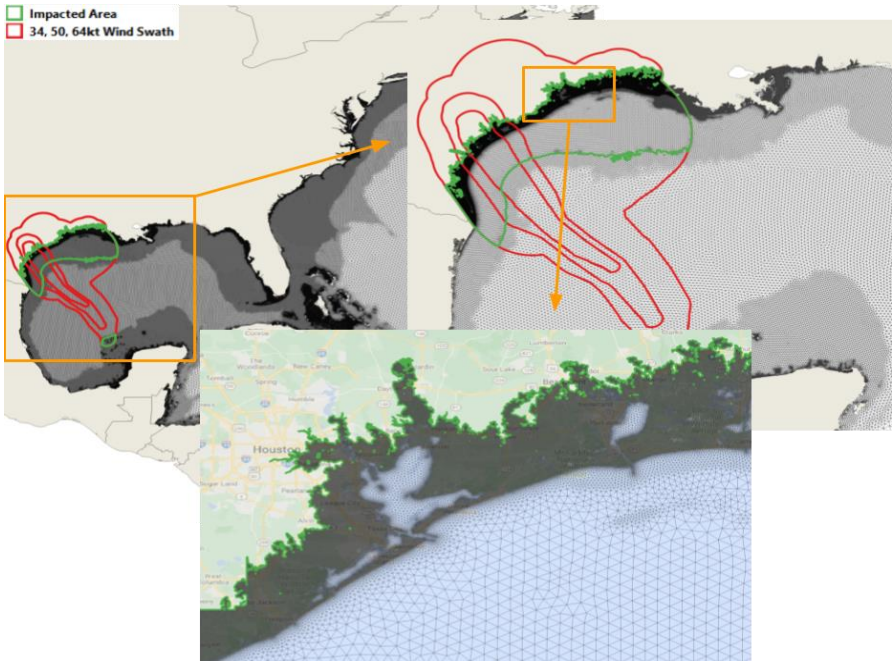

Figure 12 The result of mesh generation for Hurricane Harvey (2017) using the workflow system with the default parameters.

Using this mesh the model is set up with NWM hydrology forcing, ERA5 atmospheric forcing and TPXO tidal forcing. The simulation runs (cold start) from August 11, 2017 until September 6th 2017. The result of this simulation at three of the stations close to a populated landfall area are shown below (figures Figure 13-Figure 15). Note that the bias is removed from the results to check if the hurricane signal is caught by the simulation.

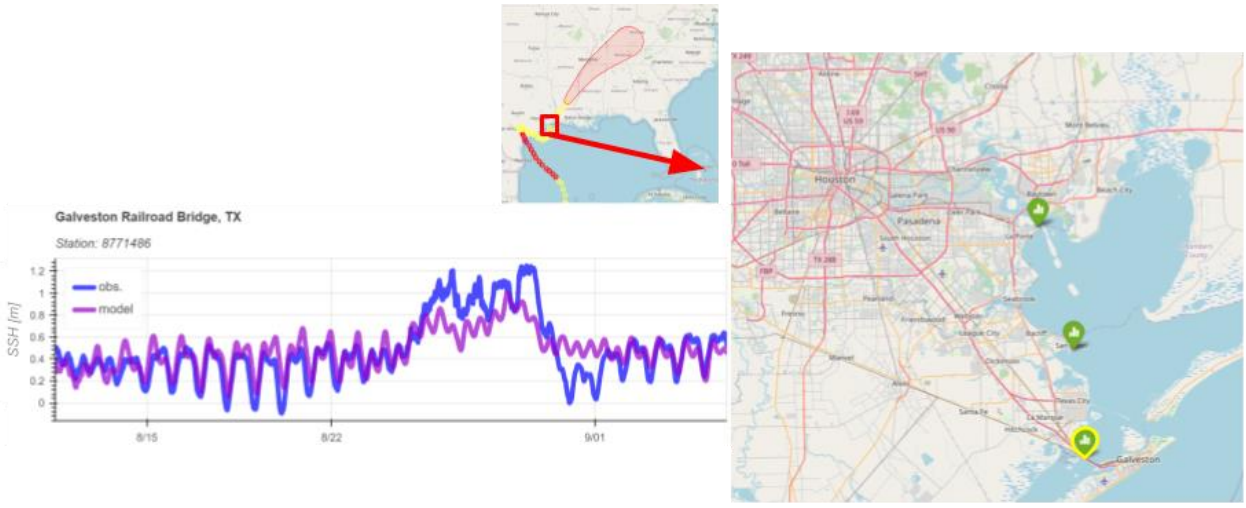

Figure 13 Comparison of measurement and simulation time series at Galveston Railroad Bridge station in Texas for Hurricane Harvey (2017). The bias is removed to only emphasize on the signal of the storm

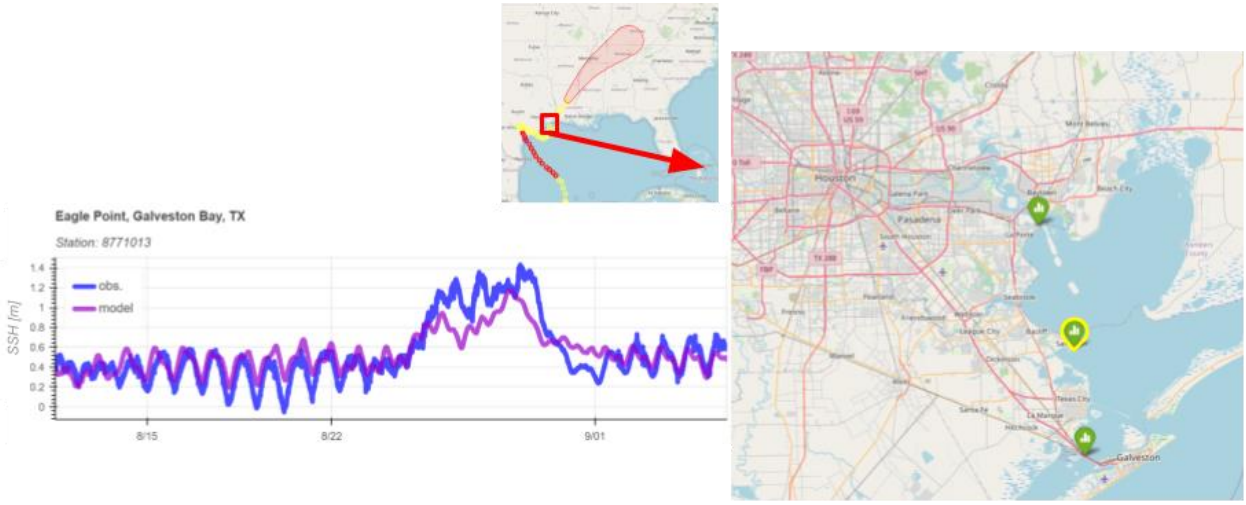

Figure 14 Comparison of measurement and simulation time series at Galveston Bay station in Texas for Hurricane Harvey (2017). The bias is removed to only emphasize on the signal of the storm

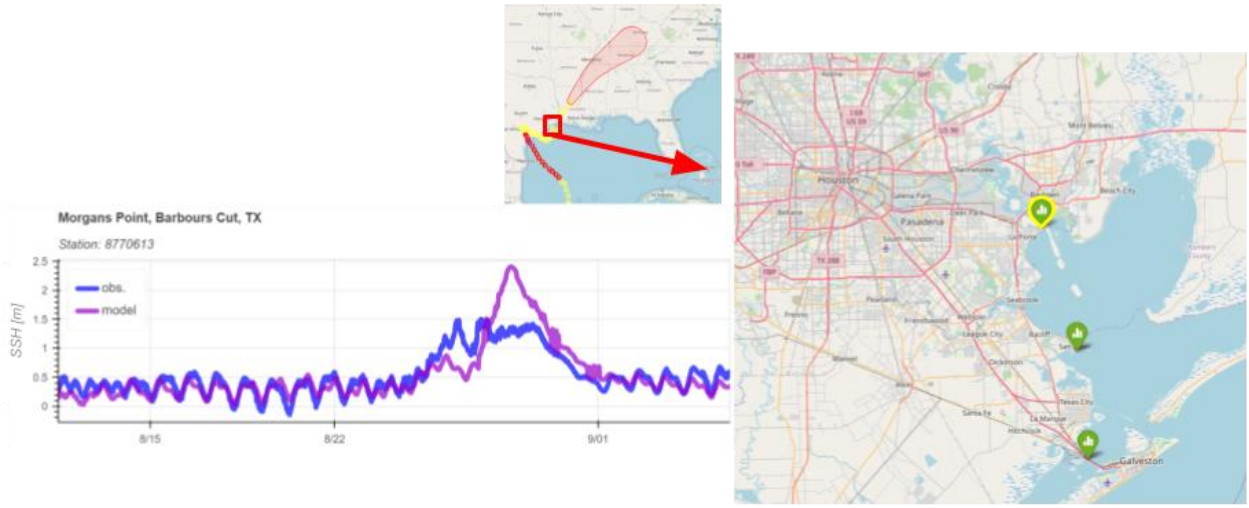

Figure 15 Comparison of measurement and simulation time series at Morgan's Point station in Texas for Hurricane Harvey (2017). The bias is removed to only emphasize on the signal of the storm

Atmospheric Forcing Comparison

This comparison study is done as a part of the NOPP project, but without the use of the on-demand system. The main objective of it was validating correct set up of the atmospheric forcing used within the workflow system.

Case study: Tropical Storm Claudette (2021)

Tropical Storm Claudette was the third named storm of the 2021 Atlantic hurricane season, which produced gusty winds, flash flooding, and tornadoes across the southeastern United States in June 2021. Originating from a trough of low pressure over the Bay of Campeche on June 12, it acquired tropical storm strength at 00:00 UTC on June 19 just before landfall in southeast Louisiana. It weakened to a depression as it turned east-northeastward, followed by a re-intensification to tropical storm over North Carolina early on June 21. Although it is a low-end tropical storm, Claudette brought a tremendous amount of moisture from the Gulf of Mexico. The heaviest rainfall focused to the east of New Orleans, amounts up to eight inches across Lake Pontchartrain. The wind gusts were up to 43 mph in New Orleans and as high as 59 mph in Gulfport, Mississippi. We did a post-hurricane assessment on Tropical Storm Claudette with STOFS-3D grid. Two atmospheric datasets (GFS and HRRR) were used in this hindcast simulation. HRRR is a high-resolution atmospheric model with 3km horizontal resolution, which is used as the primary sflux forcing, considered to be more resolved or accurate than the secondary sflux GFS. The two were blended in the SCHISM model. Figure 16 shows 24-hr time-averaged precipitation rate from these two datasets. With high-resolution, HRRR captured the isolated maximum rainfall rate above 10 kg m^{-1} s^{-1} across the southeast of New Orleans. Both datasets show similar pattern on sea

surface pressure and wind vector, except that pressure depression center from HRRR is closer to the land than that from GFS. Comparison of wind magnitude (Figure 18) between observation and simulations at ten NOAA gauge stations (Figure 17) shows that, overall, the wind magnitude of HRRR is more accurate than that of GFS in representing wind gusts, for example, at stations New Canal Stations (8761927) and Shell Beach (8761305). Station Grand Isle (8761724) locates behind the barrier island, which may be the reason of large errors in wind magnitude. Figure 19 and Figure 20 shows the comparison of total water elevation and sub-tides, respectively. The storm surge on June 19 is well captured. The averaged correlation coefficient is 0.92. The averaged MAE (mean absolute error) for total water elevation is 17cm, while the averaged MAE for subtidal comparison is 10cm.

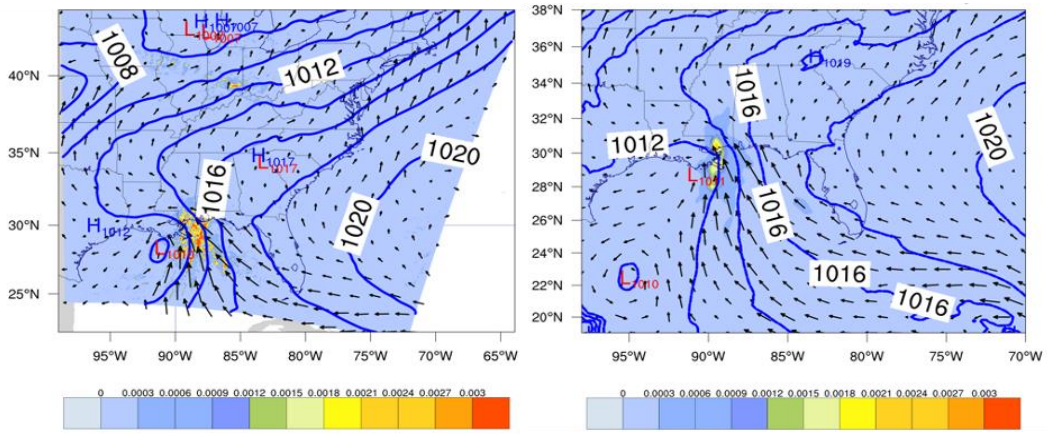

Figure 16 Time averaged (2021-06-18 12:00:00 to 2021-06-19 12:00:00) precipitation rate (kg m-2 s-1, or inch/24hr) from two datasets HRRR (left, primary) and GFS (right, secondary), contour lines are sea surface pressure, wind field is shown as vectors.

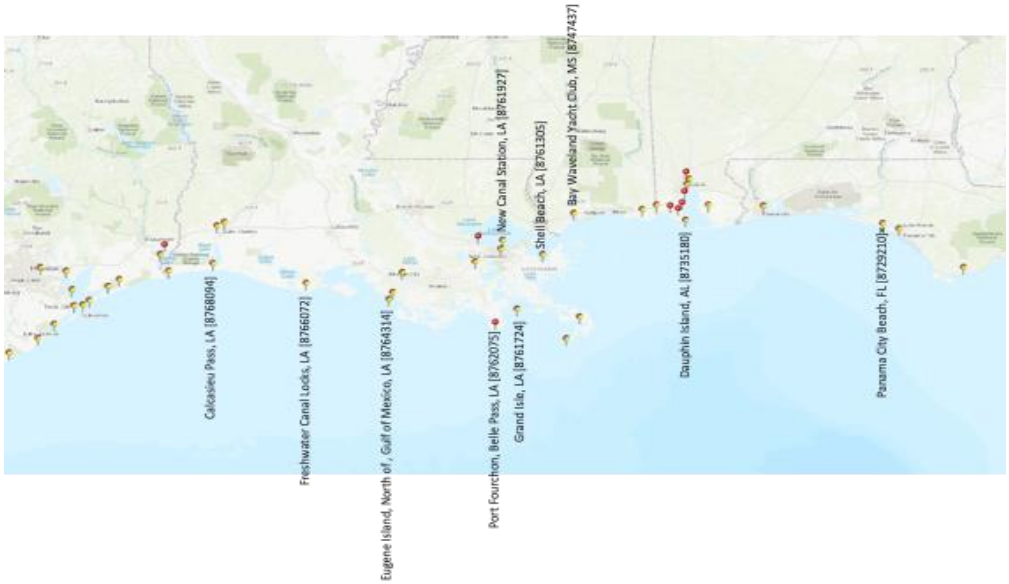

Figure 17 Location of ten NOAA gauge stations.

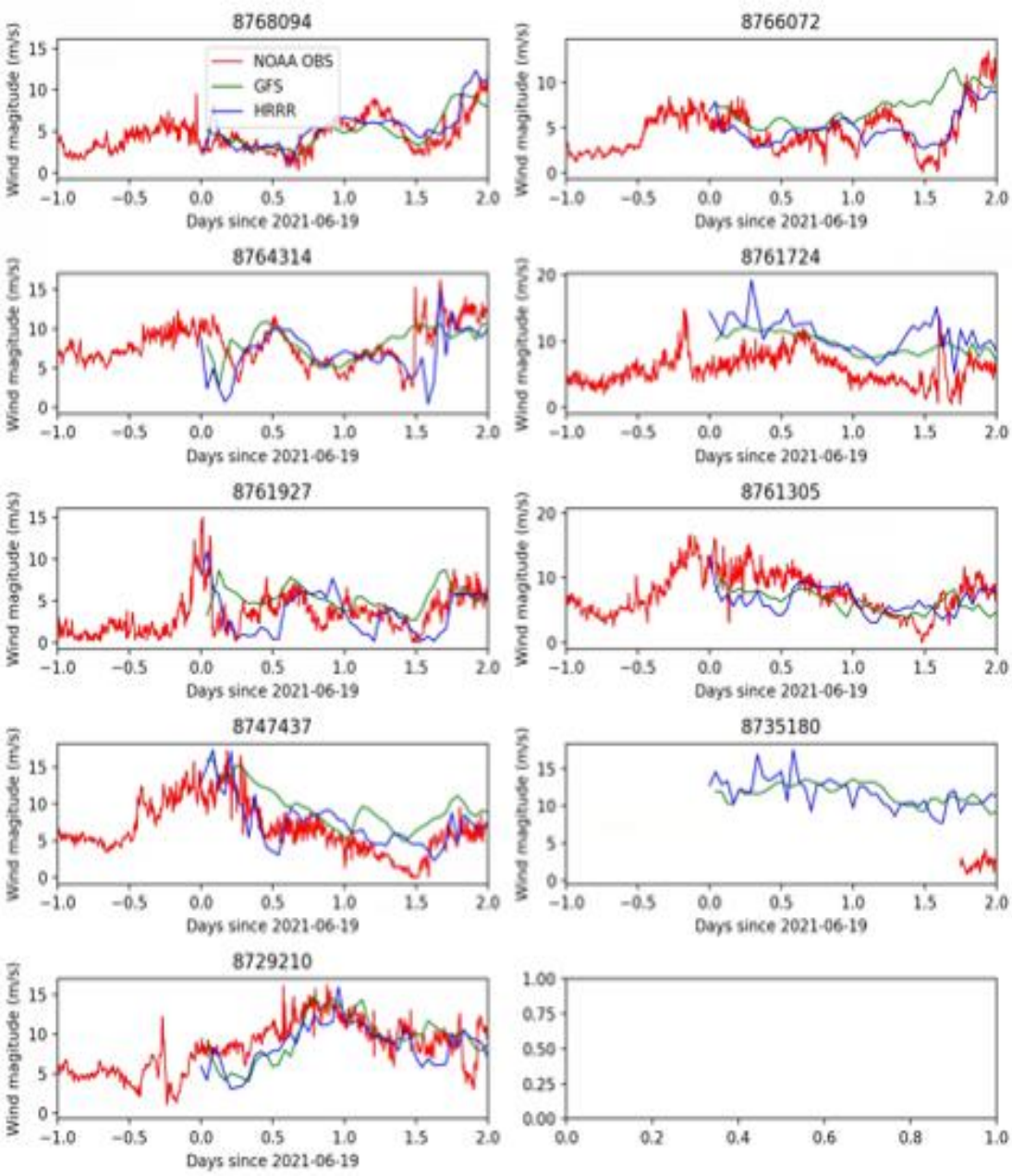

Figure 18 Comparison of wind magnitude upon landfill (2021-06-18) among observation (red lines), HRRR (blue lines), and GFS (green lines).

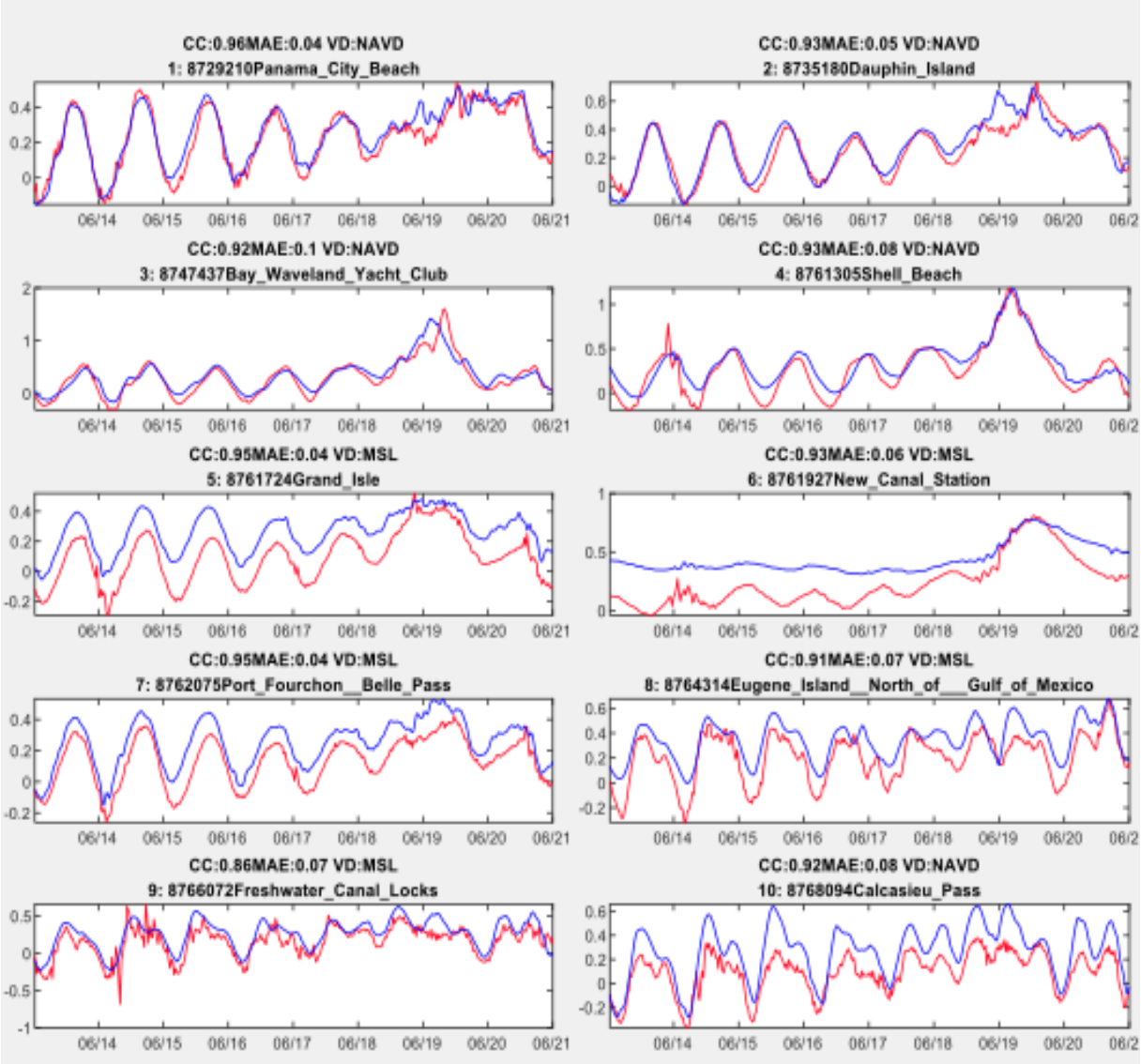

Figure 19 Time series of observed (red lines) and modeled (blue lines) total water elevation at ten NOAA gauge stations.

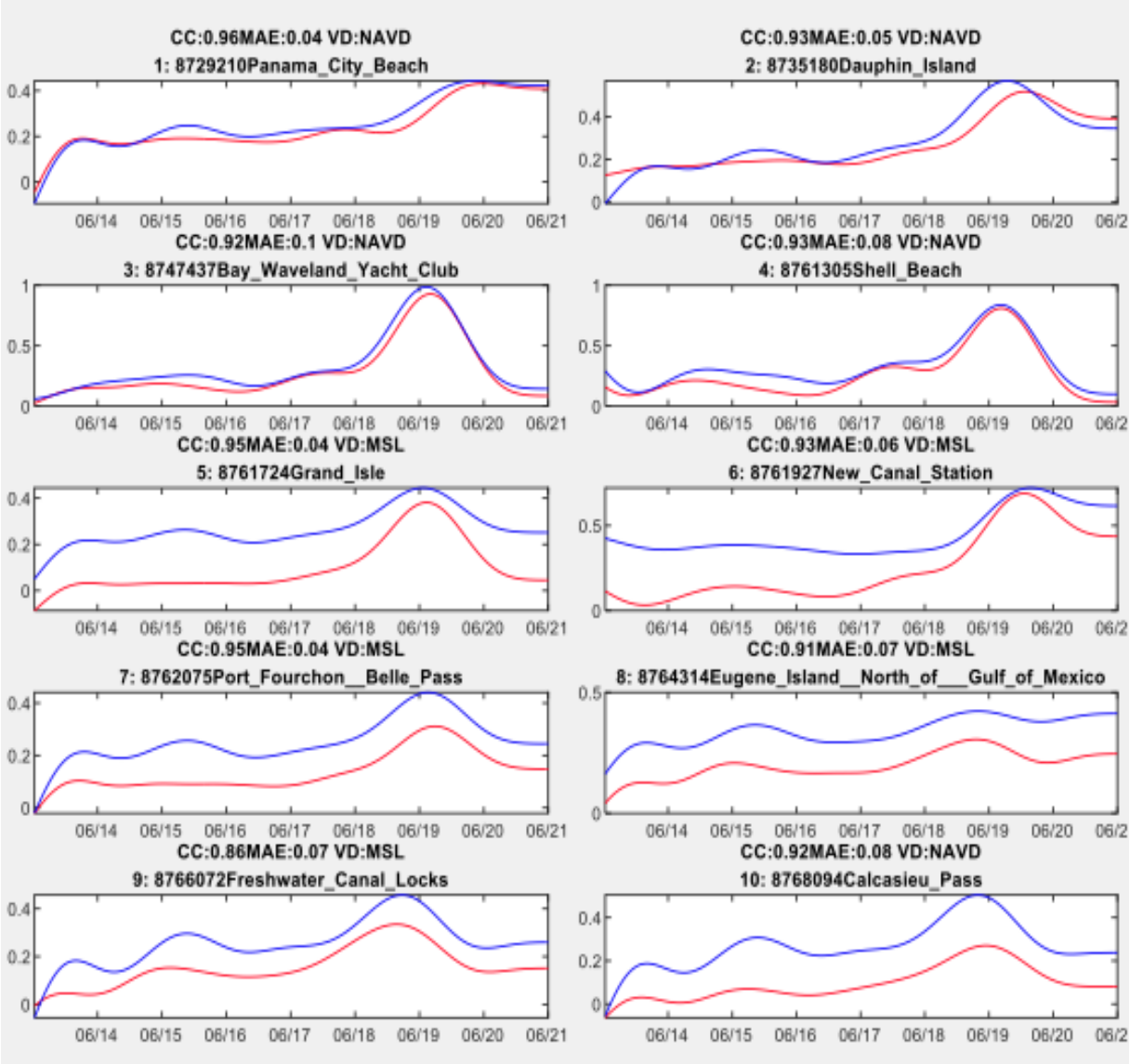

Figure 20 Time series of observed (red lines) and modeled (blue lines) sub-tides at ten NOAA gauge stations.

Conclusion & Recommendation

This project addresses the need for implementation of an automated end-to-end system for simulating hurricane events using a coastal ocean model. The implementation takes the form of an on-demand on-cloud system that is scalable. This system demonstrates the flexibility achieved by using a proper workflow management system such as Prefect. The project uses a variety of available open-source tools and data-sets to process elevation data, generate computational mesh, set up the model, simulate the storm and visualize the results. While the system can deliver reasonable results, there are many aspects of it that can be improved; for example, each component of the system can be independently optimized to get a better performance or quality of the results. Additionally, while the system is fully automated for running the simulation, the system setup process is manual and can benefit from automation. This includes proper versioning of the workflow definition to compare the results between iterations and automated testing.

References

Burnett, Z. R. (2021). Python interfaces for observational data surrounding named storm events. From https://github.com/oceanmodeling/StormEvents

Calzada, J., Cui, L., & Wang, Z. (2020). A Python interface for SCHISM model runs. From https://github.com/schism-dev/pyschism

Egbert, G. D., & Erofeeva., S. Y. (2002). Efficient inverse modeling of barotropic ocean tides. *Journal of Atmospheric and Oceanic Technology, 19*, 183-204.

Huang, W., Ye, F., Zhang, Y. J., Park, K., Du, J., Moghimi, S., . . . Liu, Z. (2021). Compounding factors for extreme flooding around Galveston Bay during Hurricane Harvey. *Ocean Modelling, 158*. doi:10.1016/j.ocemod.2020.101735

Mani, S., Calzada, J. R., Moghimi, S., Zhang, Y. J., Myers, E., & Pe'eri, S. (2021). *OCSMesh: a data-driven automated unstructured mesh generation software for coastal ocean modeling.* Tech. rep., Coast Survey Development Laboratory (U.S.). doi:10.25923/csba-m072

Moghimi, S., Myers, E., Pe'eri, S., Zhang, Y. J., & Ye, F. (2021). Forecasting compound floods in complex coastal regions. *EOS, 102*. doi:10.1029/2021EO210604

Pringle, W. J. (2020). Combining Mesh by Subsetting for Storm Surge Modeling. From https://github.com/WPringle/Storm-Surge-Mesh-Subsetting

Pringle, W. J., Burnett, Z. R., Sargsyan, K., Moghimi, S., & Myers, E. (2022). Efficient Probabilistic Prediction and Uncertainty Quantification of Hurricane Surge and Inundation. *Artificial Intelligence for the Earth Systems*, (in review).

Velissariou, P. (2021). Parametric Hurricane Modeling System (PaHM). From https://github.com/noaa-ocs-modeling/PaHM

Ye, F., Huang, W., Zhang, Y. J., Moghimi, S., Myers, E., Shachak, P., & Yu, H.-C. (2021). A cross-scale study for compound flooding processes during Hurricane Florence. *Natural Hazards and Earth System Sciences, 21*, 1703-1719. doi:10.5194/nhess-21-1703-2021

Ye, F., Zhang, Y. J., Yu, H., Sun, W., Moghimi, S., Myers, E., Liu, Z. (2020). Simulating storm surge and compound flooding events with a creek-to-ocean model: Importance of baroclinic effects. *Ocean Modelling, 145*. doi:10.1016/j.ocemod.2019.101526

Zhang, Y., Ye, F., Stanev, E. V., & Grashorn, S. (2016). Seamless cross-scale modeling with SCHISM. *Ocean Modelling, 102*, 64-81.

Zhang, Y., Ye, F., Yu, H., Sun, W., Moghimi, S., Myers, E., Liu, Z. (2020). Simulating compound flooding events in a hurricane. *Ocean Dynamics, 70*, 621-640. doi:10.1007/s10236-020-01351-x