|  |  |
| --- | --- |
|  | Sargassium Forecast  Sargassium Forecast using NEMO surge and openparcels. version 0.1.0 |
|  |  |
| 2021 Edition | Running a NEMO surge model and particle tracking module in an operational framework |
|  |  |



# Table of Contents

[Table of Contents 1](#_Toc1)

[Sagassium Forecasting Documentation 2](#_Toc2)

[Introduction/Description 2](#_Toc3)

[Changelog: 3](#_Toc4)

[Installation 4](#_Toc5)

[Sargassium Forecasting has the following requirements: 4](#_Toc6)

[Usage 4](#_Toc7)

[Worker Details 5](#_Toc8)

[Config File 5](#_Toc9)

[Nowcast YAML 6](#_Toc10)

[Ecosystem YML 6](#_Toc11)

[Error Codes 6](#_Toc12)

[Known Issues 6](#_Toc13)

[Appendices / Glossary 6](#_Toc14)

[Appendix A – Example Config File 7](#_Toc15)

# Sagassium Forecasting Documentation

## Introduction/Description

This manual details the framework that is used to run an Sargassium Forecasting product, briefly this comprises of:

* a containerised NEMO surge model
* python script workers to undertake specific tasks
* java-script process manager

These three parts work together to undertake the following workflow:

* download the latest atmospheric forcing at defined intervals (default 24 hours)
* process forcing into NEMO compatible netcdf
* run NEMO surge model using docker/podman container
* watch NEMO model and QA output
* download the latest sargassium locations at defined intervals (default 24 hours)
* generate seed location list
* run open parcels module that takes seed locations and NEMO current output and returns particle tracks within a netcdf file.
* plots a defined number of tracks on to a map and saves as png
* clean up logs, NEMO run directory and old outputs.

This system will run at a set interval creating new data and outputs to be interpreted by end users. The source workers, (download weather and get sargassium) determine when the process starts as both need to successfully download data to be able to run the model. The other workers are secondary and make regular checks (default 10 mins) to see if there is new data or if the previous worker has run successfully.

## Changelog:

0.1.0: First release

## Installation

### Sargassium Forecasting has the following requirements:

* containerisation framework: docker or podman
* python package manager: conda
* node package manager: npm

The framework has been designed and tested on Linux, both Fedora and Ubuntu have been used. Once the requirements are in place the framework can be installed as follows:

Clone the repository and switch to correct branch

$ git clone https://github.com/ocgabs/BLZ\_SURGE  
$ git checkout surge-container

Build the python environment using conda:

$ conda env create -f environment.yaml  
$ conda activate BLZ-SURGE

Install the process manager PM2:

$ npm install pm2:latest -g

A useful monitoring program to install is ctop, while it only workers for docker containers, once installed it allows the monitoring of the container. If using podman this does not work but running “podman events in the terminal” will provide a feed showing container events e.g. start stop etc. There is also the possibility of using cockpit to provide a website based interface which PM2 also supports but this is not currently supported.

## Usage

At this point the framework is installed. It can be started as follows:

$ pm2 start config/ecosystem.yml

To monitor the system PM2 has some terminal tools:

$ pm2 monit  
$ pm2 logs  
$ pm2 status

Running each of these commands in a separate terminal will give information on the system, monit is an interactive dashboard, logs shows the log files as they are written too and status returns the current status of PM2.

The system can be stopped with the following:

$ pm2 stop config/ecosystem.yml

Finally the processes can be removed from PM2 by:

$ pm2 delete config/ecosystem.yml

Indivdual workers can be stopped, started and deleted in the same way:

$ pm2 start download\_weather

Logs for indivdual workers can be inspected as follows:

$ pm2 logs download\_weather

This will show both standard output and also errors (stdout and stderr). It is useful for debuggin a single process as the main log feed can get overwhelming with the constant writing from each worker to it.

Indivdual workers can also be run using python command, e.g.

$ python download\_weather.py config/nowcast.yaml config/ecosystem.yml -f

The -f flag is a force flag that overrides the check for new data and exit code check of previous worker (where apropriate).

When needed docker will pull and run the container but it can be pulled beforehand with this command:

$ docker pull thopri/nemo-surge:8814

## Worker Details

Download Worker:

Process forcing Worker:

Generate boundary worker:

Run NEMO worker:

Watch NEMO worker:

Get Sargassium worker:

Find seed worker:

Run Parcels worker:

## 

## Config File

Sargassium Forecasting has two config files, one to configure the workers (nowcast.yaml) and one to configure the process manager PM2 (ecosystem.yml). Once setup for a system, this is largely just changing file paths to reflect system location and then these configuration files should not need to be edited.

## Nowcast YAML

At the moment each worker is configured individually with all the paths and parameters it needs to run successfully. However this ends up with a lot of duplication as many parameters are shared, e,g. directory paths model info etc. So future development will change the layout so that parameters are not repeated. At the moment if a parameter is changed e.g. model name the end user will have to ensure the multiple instances of this parameter are changed.

## Ecosystem YML

This configuration file is used by PM2 to define what processes to start and what parameters they require, e.g. restart delay. While processes can be started on the command line individually, it is much easier and more repeatable to use a config file.

## Error Codes

The system will return error messages and codes to show if there is an issue. The error messages are usually descriptive enough to give an indication of the problem and the system has some error handling code to try and catch and deal with common errors.

|  |  |  |
| --- | --- | --- |
| Error Code: | Function | Reason for Code: |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Known Issues

# Appendices / Glossary

### Appendix A – Example Config File