|  |  |
| --- | --- |
|  | 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](#_Toc66935226)

[Sargassum Forecasting Documentation 2](#_Toc1087008171)

[Introduction/Description 2](#_Toc1690715921)

[Changelog: 3](#_Toc645387348)

[Installation 4](#_Toc2129122922)

[Sargassium Forecasting has the following requirements: 4](#_Toc210636893)

[Usage 4](#_Toc1075801405)

[Worker Details 5](#_Toc84447128)

[Config File 6](#_Toc1766280505)

[Nowcast YAML 6](#_Toc124274403)

[Ecosystem YML 6](#_Toc1456467413)

[Error Codes 6](#_Toc521303056)

[Known Issues 7](#_Toc1406776368)

[Appendices / Glossary 7](#_Toc909607825)

[Appendix A – Example Config File 7](#_Toc1905753978)

## Sargassum 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

### Sargassum 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

Navigate to the BLZ-SURGE main directory and build the python environment using conda (select yes if asked to confirm):

$ 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 but this is all outside the scope of this manual.

## Usage

At this point the framework is installed. It is important that all commands are run from the conda environment created previously so ensure that the environment is loaded as follows:

$ conda activate BLZ-SURGE

It can be started as follows (from the BLZ-SURGE directory):

$ 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 in real time and status returns the current status of PM2 (which processes are running etc).

The system can be stopped with the following (again from the BLZ-SURGE main directory):

$ pm2 stop config/ecosystem.yml

Finally the processes can be removed from PM2 by:

$ pm2 delete config/ecosystem.yml

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

$ pm2 start download\_weather

Logs for individual 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 debugging a single process as the main log feed can get overwhelming with the constant writing from each worker to it.

It is also sensible to install a log rotate module for PM2 that ensures log files don’t get too excessive. This can be installed as follows:

$ pm2 install pm2-log-rotate

Individual workers can also be run using python command (from the BLZ-SURGE main directory), e.g.

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

The -f flag is a force flag that overrides the workers checking to see if it needs to be run by comparing exit codes of the previous worker and itself.

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

$ docker pull thopri/nemo-surge-blz:8814

## Scheduling

While most of the workers just look for the success of the previous worker and start based on that two workers need to be started at a defined interval. These are the download weather and get sargassum workers which get the latest data to feed into the framework. Both of these workers will start automatically with the pm2 start ecosystem.yml command but once complete unlike the other workers they do not get restarted. This functionality is implemented using Cron and the setting up of a cronjob. This is a scheduling process within Linux. To add entries the user needs to create a crontab file.

$ crontab -e

This opens the crontab and the following entry can be added.

30 8 \* \* \* /usr/local/bin/pm2 start download\_weather > /home/thopri/BLZ-SURGE/logs/cron-weather.log ; /usr/local/bin/pm2 start get\_sargassium > /home/thopri/BLZ-SURGE/logs/cron-sar.log

This starts the two workers at 8:30 am, saving the terminal output to a log file. If a different time is desired the user can amend the first two values. The first being the minutes and second is the hours, e.g. 30 8 is 8:30 am.

This is essential to starting the framework operationally as without it the system will only run the model once.

To get email notifications, adding the following to the top of the crontab should result in emails being sent showing the worker starting.

MAILTO=example@email.com

## Worker Details

### Download Worker:

This worker downloads the latest pressure and wind data from the NOAA GFS hourly model, it uses an grib filter to only download the specific region required. The download consists of hourly grib files, the worker downloads the latest midnight run. Each hourly file is checked before downloading to ensure all files are present. If the current day forecast is not up then the worker will download the previous days forecast. If there are errors in retrieving the data then this is logged by the process manager. The downloaded data is saved to the grib folder in the main directory.

While there is code to download other daily runs, the model config only supports the midnight run, so the worker is locked to download this run only.

### Process forcing Worker:

The downloaded data is a set of hourly grib files, (120 of them) whereas NEMO expects an netcdf file. This worker converts the grib files in the grib folder and saves the converted netcdf file to the netcdf folder in the main director. The worker ensures that the format of the netcdf file is compatible with the NEMO model.

### Generate boundary worker:

This is the only worker that is not automatically run by the process manager, it only needs to be run once for each domain to generate the weighting files needed by NEMO to apply forcing data to its boundaries.

It uses the surge container as it also has the NEMO tool-set on board. The command to generate the weighting files is:

$ docker run -rm -v /home/username/BLZ-SURGE thopri/nemo-surge-blz:8814 python worker/generate\_boundary.py config/nowcast.yaml

This will run the weighting tool within the NEMO surge and generate the required weighting netcdf files. This requires the files in namelist folder and also saves the weighting files in the weights folder.

Once this is completed then it will not need to be re run unless the model domain is modified which also requires the container to be rebuilt.

### Run NEMO worker:

This worker prepares NEMO to run, this requires changes to the namelist file, such as updating the forcing file names, the start date of the run. What the starting time step is etc. Once complete it then starts the NEMO surge container and exits.

### Watch NEMO worker:

Once the container is successfully running, this worker watches it and checks its output for errors. It checks for container start, once it sees the container running it will monitor the time.step file for progress and print that progress to screen or log file. Once the container has stopped running, this worker will undertake the following checks to ensure the run was successful.

Tests include:

- does results directory exist?

- does the output.abort.nc file exist? (indicates NEMO aborted)

- does the time.step file exist?

- has the expected number of times steps been run?

- Are there E R R O R flags in the NEMO log file ocean.output?

- Does the solver.stat file exist?

- Are there NaN values in solver.stat?

If all these tests pass then the model is considered to have successfully run, and the worker exits.

### Get Sargassum worker:

The second worker that acts as an origin point and gets data, this worker downloads the latest sargassum product, this is currently a 7 day composite density plot and is saved to the sargassum folder in the main directory.

### Find seed worker:

This worker uses the downloaded sargassum product to generate seed locations based on areas of ‘lots’ of sargassum. The worker saves the locations as CSV files in the sargassum folder.

### Run Parcels worker:

This worker takes the checked results from the NEMO run, which are saved in the OUTPUTS folder within the subfolder netcdf. It also ultilses the locations saved in the sargassium folder. Using the NEMO currents and the seed locations it runs the OceanParcels module and outputs a netcdf with tracks for each particle. This is saved in the OUTPUTS folder, within the subfolder tracks.

### Plot Tracks worker

This worker takes the track output netcdf file from the run parcels worker located in the OUTPUTS/tracks folder and plots the tracks onto a map. As there are so many tracks a random subset of tracks is used. (default number is 500). The worker outputs a PNG image in the OUTPUTS folder located in the subfolder plots.

The extent of the map is determined by the extent of Sargassium locations. This means that the extent is not always the same.

### Clean Up Worker

This worker ensures that files are trimmed to ensure that file space requirements are met, this deletes output files older than a set threshold (default 7 days) and also trims the restart files for NEMO when the are no longer needed and any log files created by workers.

## 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). These files should not need changing aside from updating the paths to reflect the location of the main BLZ-SURGE directory on the system.

## 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