



WOD – Weather On Demand System Description and User Manual

Karolina Stanislawska
Logi Ragnarsson
Ólafur Rögnvaldsson

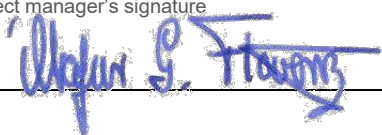
Prepared for UNECA

Short report
ÍSOR-16022

Project no.: 15-0130
04.07.2016

ICELAND GEOSURVEY

Reykjavík: Orkugarður, Grensásvegur 9, 108 Rvk. – Tel: 528 1500 – Fax: 528 1699
Akureyri: Rangárvellir, Hlíðarfjallsvegur, 603 Ak. – Tel: 528 1500 – Fax: 528 1599
isor@isor.is – www.isor.is

Project manager's signature 	Reviewed by ÓGF
--------------------------------------------------------------------------------------------------------------------	--------------------

Introduction

The ISOR-Belgingur weather forecasting tools provide access to up-to-date accurate wind and weather predictions, formatted to suit operational requirements of local conditions. Key advantages of the range of solutions are superior forecasting in complex terrain and the “On Demand” capability to produce forecasts for new regions on-the-fly.

The ISOR-Belgingur partnership provides accurate weather forecasting solutions for the global wind energy and emergency rescue sectors. Among them is the Global Disaster Alerts and Coordination System (GDACS¹) consortium. GDACS is a cooperation framework between the United Nations, the European Commission and disaster managers worldwide to improve alerts, information exchange and coordination in the first phase after major sudden-onset disasters.

Our core expertise is in downscaling and optimizing weather forecasts according to local topography, land use and weather patterns, making our approach particularly suitable for islands and complex terrain. We have developed proprietary technology to improve the efficiency and scalability of forecast computation resulting in greater flexibility and responsiveness for a range of applications. Our team of experts has many years of experience collaborating with leading international meteorological organizations in meteorological research and development.

System description

The backbone of the ISOR-Belgingur forecasting system (called WOD – Weather On Demand) is the WRF-Chem² atmospheric model, with a number of in-house customizations. Initial and boundary data are taken from the Global Forecasting System (GFS³) operated by the National Oceanic and Atmospheric Administration (NOAA⁴) in USA. Operational forecasts use cycling of a number of parameters, mainly deep soil and surface fields. This is done to minimize spin-up effects and to ensure proper book-keeping of hydrological fields such as snow accumulation and runoff, as well as the constituents of various chemical parameters.

The WOD system can be used to create conventional short- to medium-range weather forecasts for any location on the globe. The WOD system can also be used for air quality purposes (e.g. dispersion forecasts from volcanic eruptions) and as a tool to provide input to other modelling systems, such as hydrological models. A wide variety of post-processing options are also available, making WOD an ideal tool for creating highly customized output that can be tailored to the specific needs of individual end-users.

¹ <http://www.gdacs.org>

² <http://ruc.noaa.gov/wrf/WG11>

³ <http://www.emc.ncep.noaa.gov/index.php?branch=GFS>

⁴ <http://www.noaa.gov>

The Weather On Demand Architecture

The design philosophy of the WOD system embraces the open source philosophy. This makes it possible to update, scale out and deploy the system without having to worry about licensing fees. The system code is written in Python and uses many scientific libraries from the NumPy package. It is run on clusters of Linux machines, uses PostgreSQL and nginx webservices. The system code is event driven; processes wait until they get an event from an external source before they do any kind of processing at all. That is, we don't have to wait until we are certain of a particular process can take place because we know exactly when the conditions for said process to take place are fulfilled. This also means that throughput can be added as needed to handle increased workload. The main components of the system are shown in Figure 1.

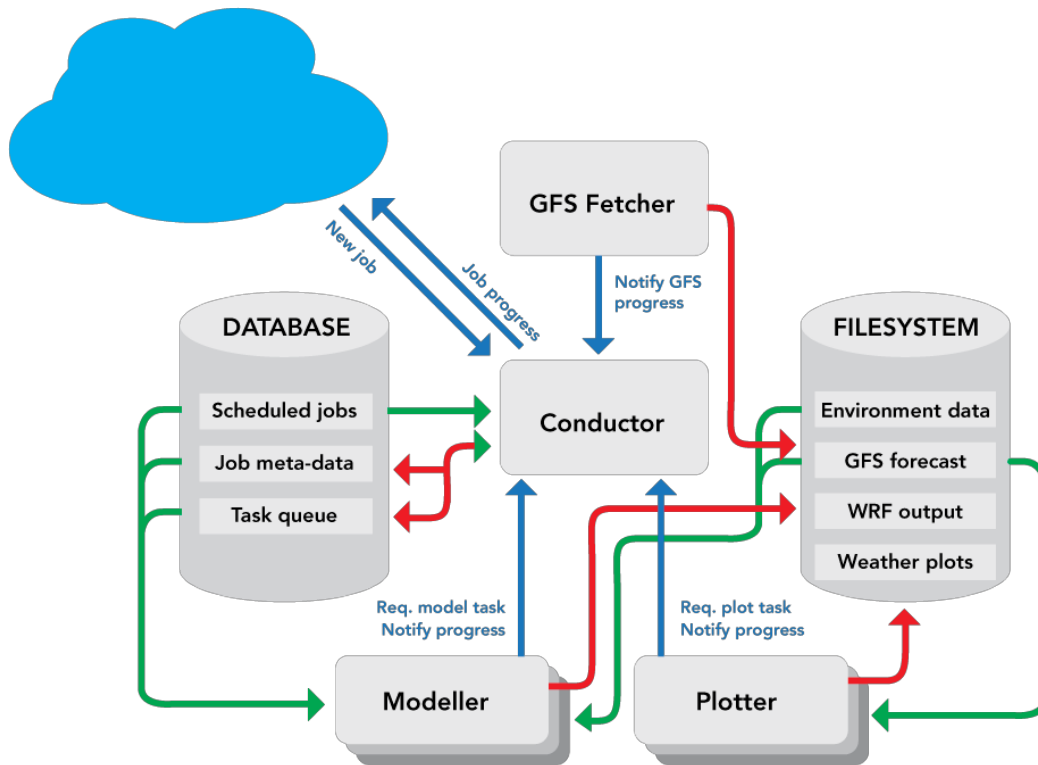


Figure 1. Schematic of the main components of the WOD system. Red lines indicate data being written whilst green lines show data being read. Blue lines represent messages being sent and blue boxes show individual tasks within the WOD system. The cylinders are databases, to the left is the meta-database and to the right is the shared file-system database. The black boxes represent components of the WOD system that perform specific tasks, such as running the weather model (i.e. Modeller) or create weather charts (i.e. Plotter). See text for more detailed description of each component.

Components

The WOD system is built around a database, large file systems, the WRF-Chem weather model and its utilities, as well as several services developed by Belgingur. The building blocks are:

An **API**⁵ that communicates with external systems. Most notably it:

- Delivers job meta-data from the database.
- Delivers weather plots from the file-system.
- Delivers model output files from the file-system.
- Updates job meta-data in the database on behalf of clients.
- Notifies the Conductor if a client requests a job to run.

The API can be used to create Jobs and Schedules as well as trigger Schedules. It has a discoverable web interface and reads Job and/or Schedule meta-data, given the reference value for the object and it can also read weather images.

A **Conductor** that has an XML-RPC API for internal use only, which is called by the external WOD API and other services in the system. The Conductor tasks are:

- Keep a priority queue of Tasks in the database and hands out tasks to workers who request them.
- Create Tasks when notified that a Job has been submitted.
- Create Tasks in response to progress on the Job.
- Listen for new weather data becoming available and trigger Schedule instances to create Job instances as needed.
- Listens on a [separate port](#) for HTTP requests and responds with a plethora of diagnostics metrics for monitoring. This also causes the Conductor to write its status to the logs.

A **Modeller** that polls the Conductor for Tasks for running the weather model and notifies it on progress and completion. In general there may be one or two modellers per host where the cores are distributed between them.

A **Plotter** that polls the Conductor for Tasks for generating weather plots and notifies it on completion. Each plotter uses a single core and the number of plotters run is equal to the number of cores assigned to plotting.

The **GFS Fetcher** is usually launched by the `cron` daemon. It downloads weather data from NOAA as it becomes available, converts it into a format suitable for the WRF-Chem weather model and notifies the Conductor.

Configuration

The configuration for all components is loaded from an ini-file at `~/wod_conf/wod.conf` unless the location is overridden with the `WOD_CONF` environment variable. This configuration is used to override the default values

⁵ <https://wod.belgingur.is/wod/api/1.0>

found in `config/defaults/wod.conf` in the WOD project directory. This file also describes each available option.

Config values can contain `${variable_name}` tokens which will be replaced by the value of the named variable.

The Weather On Demand Virtual Environment

The WOD system expects to be run in a virtual environment in the anaconda (or miniconda) distribution ⁶. To set it up, execute the script found in `~/git/scripts/requirements.sh` and follow the instructions. Please note that you have to have super-user privileges to install some of the external packages.

The virtual environment design

The virtual machines are based on KVM/QEmu. The DL380 machine is the hosting machine. It maps `em1` (real network) to `br0` (the bridging interface for the KVM so *some* of the virtual machines may have an internet presence) and `p1p1` (10Gbit backbone) to `br1` (bridging interface for the KVM so the virtual machines may exist on the same network as the computing nodes).

The virtual machines are created by role

- web
 - The front-end web server for the system. This machine receives all requests through the virtual front-end interface and serves the default web interface.
 - Has two interfaces – a public one and a private one
- trigger
 - The trigger machine fires upon events of post-processing
 - Does only exist on the private network
- wod
 - The wod machine is the main processing node – it contains the logic for WOD.
 - Has two interfaces (public and private) as the wod api is open to the outside
- service
 - Contains only service modules – DNS and PostgreSQL (metadata store for WOD). Other micro-services should be added here.
 - Does have two interfaces to the world (the public interface is not required, but it is requested).
- dbstore
 - A dedicated machine for running PostgreSQL for post-processing queries and storage
 - Only exists on the private network

⁶ <http://conda.pydata.org/docs/>

Each virtual machine exists on one or more LVM filesystems on the physical host machine. Care must be taken that the ownership of the filesystems is on libvirt-qemu:kvm only.

A NTP server is running on the hosting machine, providing the VMs and computing nodes with the correct time.

Moving the virtual environment from one network to another:

On the machines that have the public interfaces (wod, web, and service), edit `/etc/network/interfaces` and change the public IP address on the machine to what IP address it will have on the new network (addresses with the pattern 10.1.0.XXX are the candidates). **USUALLY** the addresses given to the virtual host are RFC1918 addresses, and the real addresses mapped to these in a *firewall*. Note that we also need to make certain changes on the "service" virtual machine. In `/var/named/named.conf.options` is a section named "forwarders". It should point to the internal DNS of the company/institution (so hosts may be resolved on the internal network).

Hostnames need to be changed in `/etc/nginx/*` - search for "belgingur.is" - the hostnames should identify which is which - hosts needing this change are "web" and "wod".

The NTP server config on the hosting computer must be changed. Find the line saying "server time0.rhnet.is iburst" and change it to "server xxx.yyy.zz iburst" where xxx.yyy.zz is the local time server for that region and network. *If* no servers are known, please refer to the free and open NTP project ([NTP Pool for Africa](#)).

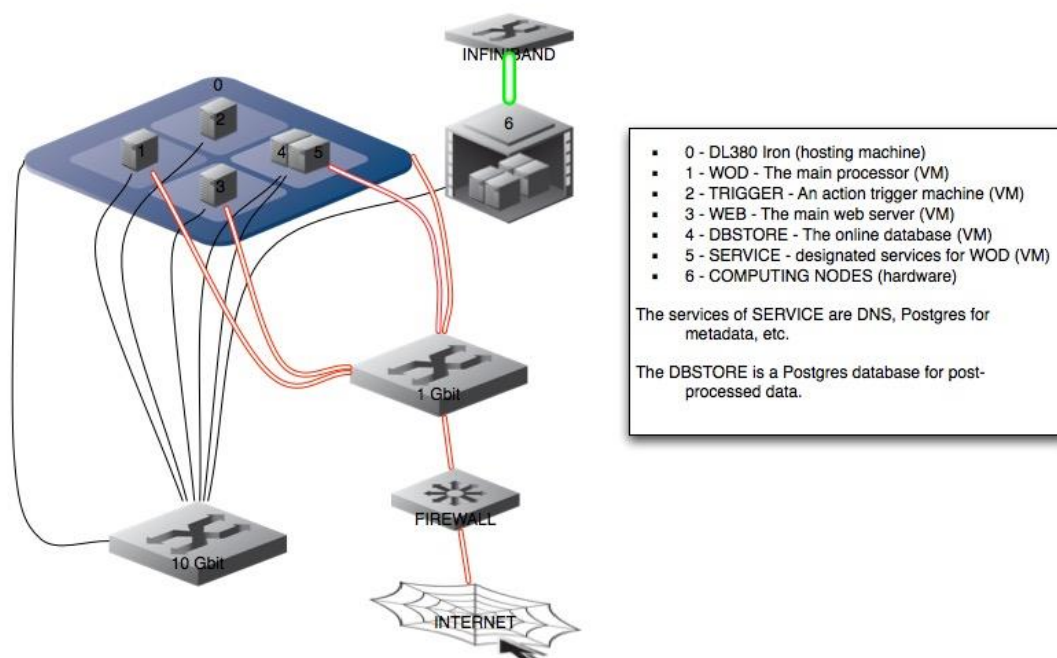


Figure 2. Schematics of the basic WOD hardware configuration. See text for further description.

The Weather On Demand Executables

For an up-to-date list of executable in the WOD system, enter the program folder `~/git` and execute `find . executable -type f -name '*.py'`. The full usage of each is given by passing the `-help` argument to the command. Here, we give a short description of the purpose of each command to avoid duplication:

Services

`~/git/api/api_server.py` Runs the WOD API.
`~/git/conductor/conductor.py` Runs the Conductor.
`~/git/modeller/model_worker.py` Runs a Modeller, the `-cores` argument limits the number of cores assigned to each forecast run.
`~/git/plotter/plot_worker.py` Runs a Plotter.

Processing Steps

`~/git/gfs/fetcher.py` Fetches the GFS input data, by default the most recent analysis and forecast is downloaded. Using the `-analysis` argument allows the user to download a GFS analysis and forecast that is not the most recent one.
`~/git/modeller/steps/cycle_state.py` Cycles state from the WRF-Chem model output to the input for a subsequent simulation.
`~/git/modeller/process_job.py` Runs all, or some, of the pre-processing and modelling steps of a job. Does no plotting and needs the WOD database with meta-data.

`~/git/plotter/generate_plots.py` Generates all, or some, plots for a job or wrfout-file. Can both be run with, or without, input from the WOD meta-data database.

The Weather On Demand Database

The production WOD database is named `wod` and is hosted on the virtual machine named `service`. To log on to the database do `psql -h service.DOMAIN.?? -U wod` where `DOMAIN.??` is the domain name and ending of the virtual machine `service`. A similar connection can also be created in e.g. pgAdmin⁷.

A list of useful WOD SQL commands is given in the User Manual part of this document.

The main database tables are depicted in Figure 3, a more detailed meaning of each column is given in the text.

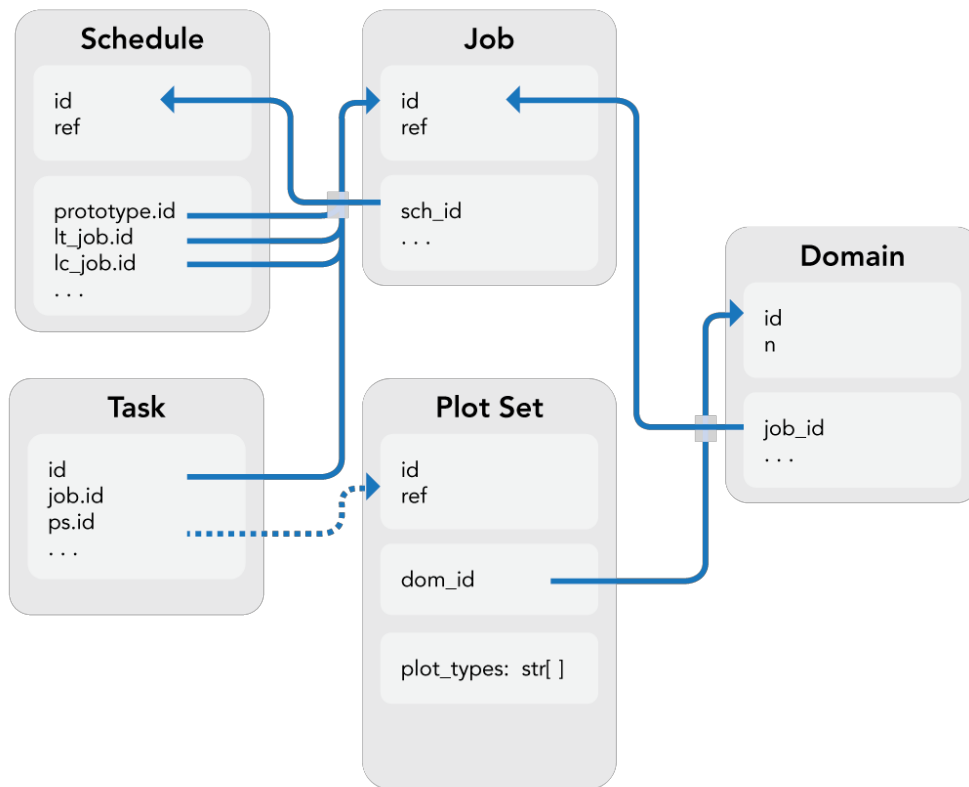


Figure 3. Schematic of the WOD meta-data database. In general, the top-level ID-numbers are cascaded down the hierarchy. The unique ID-number of the Schedule becomes the `sch_id` (or `schedule_id`) of the corresponding Job. This Job in turn has an ID-number that becomes the `job_id` number of its corresponding Domain. Again, the Domain has a unique ID-number that becomes the `dom_id` of its corresponding Plot_set. Note that the ID's are purely for internal database use, i.e. to link together various database transactions. See text for further description of individual columns of the database.

⁷ <http://www.pgadmin.org/>

Schedule

Each record corresponds to a scheduled modelling job, whether it is currently active or not. It also corresponds to the class `Schedule` in `indata_model/schedule.py` where more details can be found.

Identifiers

id	ID used only internally in the database
ref	Externally visible reference value for this job. It is generally chosen by a human
created	Timestamp when the schedule was created

Configuration

prototype_id	When the schedule is triggered, create a new job by copying the one with this ID
period_offset	New job should generate useful data for the period starting this long after the analysis time
period_length	New job should generate useful data for this long a period. The array can have a period for each domain of the job
recurrence	A rule describing for which analyses to trigger
expires	Timestamp when the schedule should stop triggering

Recurrence rules are given as segments separated by `+`, each of which is given as `WEEKDAY:HOURS` where the weekday is either a two-letter name or `'*'` and hours is either a comma separated list of numbers or `'*'`.

Examples: `*:*` triggers for all analyses every day. `*:12+MO:00` triggers on the noon analysis every day as well as the midnight analysis on Mondays.

Priority

pri_base	Pass this base priority to new jobs
pri_slope	Pass this priority slope to new jobs

Progress

lc_job_id	ID of last Job created from this Schedule which has completed
lt_job_id	If one or more Jobs are in progress from this Schedule, the ID of the on that is furthest progressed. Otherwise equals lc_job_id
lt_analysis	The last analysis for which the Schedule was triggered or rejected based on its recurrence rule

job

Each record corresponds to a modelling job, whether it is currently executing, failed, or completed. It also corresponds to the class `Job` in `~/git/data_model/job.py` where more details can be found.

Identifiers

id	ID used only internally in the database
ref	Externally visible reference value for this job. It is generated as a timestamp, a dash, and random characters.
title	Human-readable title of the job
user	The use-name of the person who requested this job
client_id	Reference to the Client application which requested this job
schedule_id	A reference to the Schedule which the Job was created from or NULL if this is a stand-alone job

Job Settings

type	The name of the job type. It corresponds to a folder in <code>data_model/job_types</code>
public	boolean

Data Used

analysis	Timestamp indicating the global analysis used as input to this job
----------	--------------------------------------------------------------------

Priority

pri_base	The base priority of this job. Often inherited from Schedule
pri_slope	Increase in priority per hour that Job is unprocessed
pri_boost	Additional priority for this job

Progress

submitted	Timestamp when the job was submitted
last_update	Timestamp when the job meta-data was last updated
completed	Timestamp when the job was completed or NULL if ongoing or failed
scrubbed	Whether the job has been scrubbed
scrub_time	Timestamp when the job is/was scheduled to be scrubbed
phase	The current phase of the job: MODEL_PENDING=3 MODEL=4 PLOT=5 DONE=6 or negative if the job failed in the corresponding phase. Phases 1 and 2 are reserved for adding pre-processing phases.
st_done	The number of completed processing steps for this job
st_total	Total processing steps for this job

domain

Each record corresponds to a modelling domain for a job. It also corresponds to the class `Domain` in `~/git/data_model/domain.py` where more details can be found.

Identifiers

id	ID used only internally in the database
job_id	A reference to the Job that this Domain belongs to
n	The index of the Domain within the Job. This corresponds to the WRF domain index and starts at 1

Dimensions

cen_lat	Latitude of the centre of the domain
cen_lon	Longitude of the centre of the domain
res	Resolution of domain grid in metres per cell
sz_sn	South-North size of domain in cells
sz_we	West-East size of domain in cells
pad	Number of cells on each side to discard as padding

Time

start	Beginning of the modelling period
stop	End of the modelling period
st_len	Model-step length in minutes
st_spinup	Numer of steps to discard as spinup
st_done	Number of steps which have been modelled so far

plot_set

Each record corresponds to a set of plots created for a particular modelling domain in a job. It also corresponds to the class `PlotSet` in `~/git/data_model/plot_set.py` where more details can be found.

Identifiers

id	ID used only internally in the database
ref	Externally visible reference value for this job. It is usually “full” for the PlotSet that corresponds to the entire Domain or chosen by a human.
dom_id	A reference to the Domain that this PlotSet belongs to

Settings

plot_types	Array of the names of the types of plots to generate. They must correspond to plots registered in PlotRegistry found in <code>plotter/registry.py</code>
plot_groups	Array of the names of plot groups to generate plots from. They must correspond to group names associated with plots registered in PlotRegistry or be empty
z_svg	Zoom-levels to generate SVG plots for
z_png	Zoom-levels to generate PNG plots for
cen_lat	Latitude of centre of plot area
cen_lon	Longitude of centre of plot area
height	Height of plot area (south-north) in metres
width	Width of plot area (west-east) in metres

Progress

st_done	Number of steps which have been plotted in the PlotSet
st_cons	Number of consecutive steps which have been plotted in the PlotSet
st_ratio	Number of steps in Domain which correspond to 1 step in the PlotSet
first_done	Timestamp when first plot was completed or <code>NULL</code> if no plots are done
last_done	Timestamp when last plot was completed or <code>NULL</code> if not all plots are done

task

Each record corresponds to an atomic task to be performed by a worker process, such as running the weather model or generating some plots. It also corresponds to the class `Task` in `~/git/data_model/task.py` where more details can be found.

Identifiers

id	ID used only internally in the database
type	What needs to be done to fulfill the Task. 4=MODEL, 5=PLOT matching the Job phases.
step	For task types where multiple steps are required (PLOT) this is the step being handled (plotted)
dom_id	A reference to the Job that the Task is for
plot_set_id	A reference to the PlotSet that the Task is for, if needed (PLOT)

Progress

submitted	Timestamp when the Task was submitted to the queue
worker	The name of the worker the Task has been assigned to or NULL
started	When the Task was started by the worker
ended	When the Task was completed or processing otherwise ended by the worker
status	Status of Task. FAILED=-2, CANCELLED=-1, PENDING=0, RUNNING=1, DONE=2

Triggers

The WOD system's interaction with external systems is described in Figure 4.

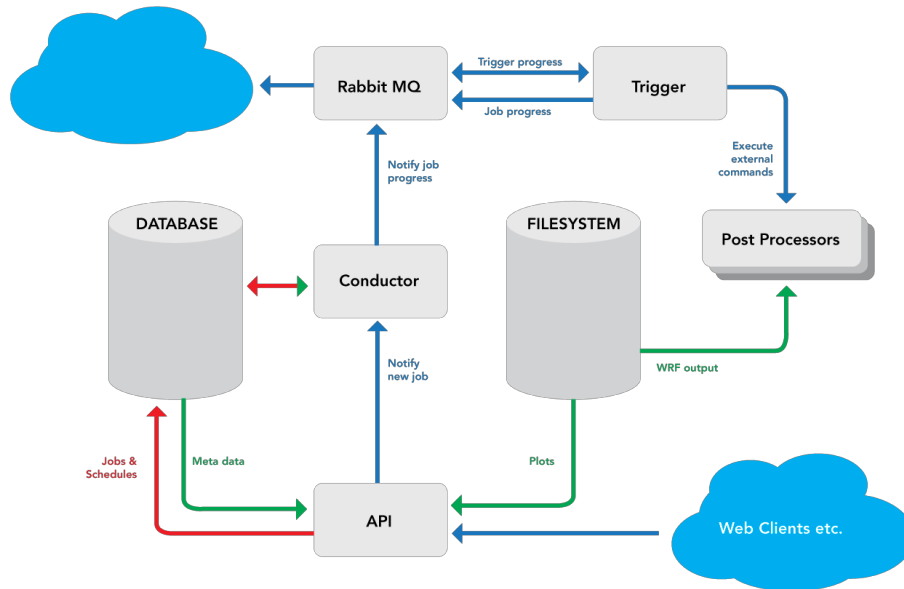
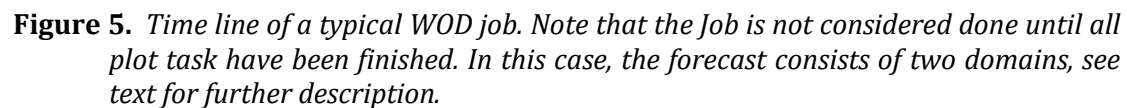


Figure 4: The WOD system's API handles communications with external web clients. These can both be used to visualize weather charts created by the Plotter or to launch new types of forecasts. In that case, the API informs the Conductor about the new job, which in turn notifies a Plotter to run the forecast. The Conductor also delivers information on job process to the RabbitMQ. The RabbitMQ informs the Trigger on job progress. The Trigger can then be used to execute specific post-processing tasks, e.g. extracting a set of point forecasts from chosen forecasts. The Trigger in turn sends messages to the RabbitMQ on its progress, this can in turn create a new set of triggers sent back to the Trigger (e.g. converting the format of the point forecasts from csv files to xml files).

The Triggers are connected by a RabbitMQ message bus and can be run remotely. The Triggers are unable to interfere with core system processing of the WOD system. If the Trigger goes down the messages queue up. The Trigger run custom post-processing jobs once a specific Job is done. These can e.g. be specialized plots, point forecasts or customized output used as input for other decision support software, e.g. in the renewable industry, be it hydro or wind. The Trigger can also be used to trigger another job once it has completed a specific task. Importantly, the Trigger can give external (i.e. as not part of the WOD system) post-processing software access to model output.

Figure 5 shows the life cycle of a typical Job within the WOD system.



1. GFS Fetcher has fetched 48 hours of data and sends a message to the Conductor
2. Conductor sees that a Schedule record needs 48 hours of global weather data and clones the prototype Job with updated date fields
3. Conductor creates Model Task for the Job and puts in the queue
4. Model Task waits in queue until it reaches the front of the queue and ...
5. an idle Modeller requests a Model Task from the Conductor and is assigned the one for the Job in question
6. Modeller works on the Model Task, writes a frame of model output and sends a message notifying the Conductor
7. Conductor creates a Plot Task for each PlotSet for which there is new data and places them in the queue
8. Each Plot Task waits in the queue until it reaches the front of the queue and ...
9. an idle Plotter requests a Plot Task from the Conductor and is assigned the one for our Job
10. The above steps are repeated until the Plot Task is completed
11. The Job is marked as completed
12. The Schedule is update with the new last-completed Job

User Manual

HowTo create a forecast type

By default the WOD system comes with a number of pre-defined forecast types, of which the most commonly used are labeled `medium.1`, `medium.3`, and `medium.9`. Here, `medium` refers to the physical dimensions of the forecast domain, and the enumerator to the horizontal resolution of the grid; 1, 3, and 9 km, respectively. All forecast types are listed under the `~/git/data_model/job_types` directory. Each forecast type directory contains the following template files:

- `geogrid.tpl`
- `iofields.txt`
- `meta.yml`
- `metgrid.tpl`
- `modeller.tpl`

The `geogrid.tpl` file contains the `&share` and `&geogrid` parts from the `namelist.wps` configure file from the WRF pre-processing system. Similarly, the `metgrid.tpl` file contains the `&metgrid` part from the `namelist.wps` configure file. The `modeller.tpl` file is a template file used to create the `namelist.input` file, needed to run the `real.exe` and `wrf.exe` parts of the WRF modelling system. The content of the `iofields.txt` file is passed to the `iofields_filename` parameter in the `namelist.input` file. More information regarding run-time IO options of the WRF model can be found online⁸. Lastly, the `meta.yml` file gives the user the option to choose the number of domains, domain resolution and size, output frequency per domain (in minutes) and spinup time per domain. In addition to this there are options for padding⁹ and choosing whether this is a regular or volcanic configuration.

To create a new forecast type it is simplest to copy an existing `job_type` directory and modify it to fit ones specific needs.

Once a new forecast type has been created it is necessary to restart the WOD service on the WOD machine:

```
sudo /etc/init/wod_restart
```

by a user that has true sudo-privileges, e.g. user `sysop`. If it is only necessary to restart the WOD Conductor, simply do `sudo restart wod_conductor`, e.g. as user `wod`.

⁸

http://www2.mmm.ucar.edu/wrf/users/docs/user_guide_V3/users_guide_chap5.htm#Phys

⁹ By “padding” we mean that the user can choose how many grid-points will be disregarded from the plotting process. By default the ten outermost grid-points are not used in the plots.

Also, it is necessary to restart the WOD service on all modeller machines. One needs to log on to all of them and execute `sudo /etc/init/wod_restart`.

HowTo submit a forecast

To create a new forecast in the WOD system it is simplest to log onto the `https://wod.DOMAIN.??/wod/api/1.0/jobs` web page. Once logged on, the user needs to and click on the horizontal **Submit** bar. Leave the **ref** box empty as the system will automatically fill it with something sensible (to the system). However, put something short and sensible in the **title** box and choose your location (Lat/Lon) for the domain center. By default the forecast runs from the latest analysis so there is generally no need to fill in the **start** box. The format is YYYY-MM-DDThh:mm:ss. For testing purposes stick to short duration (i.e. box **length**), six hours is usually sufficient. Finally, choose which forecast type you want to run. For a full list of available types, see the list under the `~/git/data_model/job_types` directory on WOD.

HowTo create a routine forecast

Choose **Schedule** and fill in the necessary forms (**period_length**=duration of the forecast; **period_offset**=when do we start creating useful data after the analysis time; **ref**=a meaningful name of the schedule). Once you have filled in the form you **MUST** click the **Schedule** button. Then you may click the **Update** button to set the **recurrence** rule. Default value is four times per day.

HowTo cancel a routine forecast

In order to cancel a routine forecast it is necessary to log onto the WOD database `psql -h service.DOMAIN.?? -U wod` where `DOMAIN.??` is the domain name and ending of the virtual machine `service`. Once logged in, do the following where the `prototype_id` matches with the forecast you want to cancel:

```
wod=> begin;
BEGIN
wod=> update schedule set expires=lt_analysis where
prototype_id=711;
UPDATE 1
wod=> select * from schedule where expires>'2015-07-01
00:00:00';
 id |          ref          | prototype_id | lc_job_id |
lt_analysis      | lt_job_id | period_offset |
period_length    | pri_base | pri_slope |
recurrence       |          created          | expires
-----+-----+-----+-----+-----+-----+-----
-----+-----+-----+-----+-----+-----+-----
-----+-----+-----+-----+-----+-----+-----
--+-+-----+-----+-----+-----+-----+-----
13 | foroyar-3          |          4724 |      13627 | 2015-
07-09 06:00:00 |      13627 | 03:00:00      | {"2 days","2
```

```

days"} | 100 | 10 |
| 2014-06-13 14:47:46 | 3000-01-01 00:00:00
10 | island-3 | 4183 | 13623 | 2015-
07-09 06:00:00 | 13631 | 03:00:00 | {"7 days", "3
days 09:00:00"} | 220 | 10 |
| 2014-05-02 14:05:20 | 3000-01-01 00:00:00
7 | fjallabak | 711 | 13604 | 2015-
07-09 06:00:00 | 13604 | 03:00:00 | {60:00:00}
| 70 | 10 | *:12+SA:00+SU:00 | 2013-08-07
11:55:39 | 2015-07-09 06:00:00

wod=> commit;

```

HowTo delete and/or change plotsets in WOD

\x -> to list items one line at a time

```
SELECT * from schedule where expires > now();
```

```
select * from job where id=11670; (here 11670 is the prototype_id)
```

```
select * from domain where job_id=11670;
```

```
select * from plot_set where dom_id in (21550, 21551);
```

begin; -> **this is a failsafe mode :)**

```
delete from task where plot_set_id=29598 ;
```

```
delete from plot_set where id=29598;
```

commit; -> **if all has gone OK, we can now commit the changes**

```
select * from plot_set where dom_id in (21550, 21551); -> see the changes
```

begin; update plot_set set z_png=z_svg where id=29599; -> **setting the PNG plotset zoom level to what is done for the SVG figures**

update plot_set set z_svg=array[]::integer[] where id=29599; -> **we now stop producing SVG figures**

```
select * from plot_set where dom_id in (21551); -> again, see the changes
```

```
commit;
```

HowTo change period_lenght in WOD

```
select * from schedule where expires > now();
```

and you get a full list of active jobs, choose the prototype_id that fits the ref "job" you want to change

wod=> begin; -> **this is a failsafe mode :)**

BEGIN

wod=> update schedule set period_length='{2 days, 2 days}' where
prototype_id=4724;

UPDATE 1

wod=> select * from schedule where id=13; -> **see the changes**

id	ref	prototype_id	lc_job_id	lt_analysis	lt_job_id	period_offset	period_length	pri_base	pri_slope	recurrence	created	expires
13	foroyar-3	4724	13622	2015-07-09 00:00:00	13622	03:00:00	{ "2 days", "2 days" }	100	10		2014-06-13 14:47:46	3000-01-01 00:00:00

(1 row)

wod=> commit; -> **if all has gone OK, we can now commit the changes**

COMMIT

wod=> select * from schedule where id=13;

id	ref	prototype_id	lc_job_id	lt_analysis	lt_job_id	period_offset	period_length	pri_base	pri_slope	recurrence	created	expires
13	foroyar-3	4724	13622	2015-07-09 00:00:00	13622	03:00:00	{ "2 days", "2 days" }	100	10		2014-06-13 14:47:46	3000-01-01 00:00:00

(1 row)

wod=>

HowTo change plot_types in WOD

select * from domain where job_id=12061;

select * from plot_set where dom_id=22466;

plot_types |
{volcanic,volcanic_agl_0300m,volcanic_agl_0700m,volcanic_agl_1000m,volcanic_agl_1500
m}

plot_groups | {}

```
wod=> begin;
```

```
BEGIN
```

```
wod=> update plot_set set
```

```
plot_types='{volcanic,volcanic_agl_0300m,volcanic_agl_0700m,volcanic_agl_1000m,v  
olcanic_agl_1500m}' where dom_id=22466;
```

```
wod=> commit ;
```

HowTo change job type name in WOD

On ~/git/data_model/job_types do:

```
git mv island.9 volcanoIS-0095x0090.9
```

then log on to the database

```
psql -h SERVER -U wod
```

and do

```
wod=> select count(*) from job where type='island.9' ;
```

```
count
```

```
-----
```

```
736
```

```
(1 row)
```

```
wod=> begin;
```

```
BEGIN
```

```
wod=> update job set type='volcanoIS-0095x0090.9' where type='island.9' ;
```

```
UPDATE 736
```

```
wod=> commit;
```

```
COMMIT
```

```
wod=>
```

HowTo change center Lat/Lon in WOD

```
or@volundur:~$ psql -h SERVER -U wod
```

```
Password for user wod: ????
```

```
psql (9.3.5)
```

```
SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)
```

```
Type "help" for help.
```

```
wod=> SELECT * from schedule where expires > now();
```

```
wod=> \x
```

Expanded display is on.

```
wod=> SELECT * from job where ref='20150504-154114-c9fb769f3559' ;
```

```
-[ RECORD 1 ]-----
```

```
ref      | 20150504-154114-c9fb769f3559
```

```
type     | westafrica.3
```

```
title    | West Africa
```

```
scrubbed  | f
```

```
scrub_time | 2015-05-07 15:41:14
```

```
submitted | 2015-05-04 15:41:13
```

```
last_update | 2015-05-04 15:47:09
```

```
completed |
```

```
analysis  | 2015-05-04 12:00:00
```

```
id        | 11850
```

```
schedule_id |
```

```
phase     | 4
```

```
st_total  | 44
```

```
client_id | 5
```

```
public    | t
```

```
user      |
```

```
st_done   | 5
```

```
pri_base  | 100
```

```
pri_boost | 0
```

```
pri_slope | 10
```

```
wod=> SELECT * from domain where job_id=11850 ;
```

```
-[ RECORD 1 ]-----
```

```
id      | 21975
```

```

job_id  | 11850
n       | 1
cen_lat | 11
cen_lon | -12
res     | 9000
sz_sn   | 220
sz_we   | 400
pad     | 10
start   | 2015-05-04 09:00:00
stop    | 2015-05-05 00:00:00
st_len  | 60
st_spinup | 3
st_done | 1

```

-[RECORD 2]-----

id | 21976

```

job_id  | 11850
n       | 2
cen_lat | 11
cen_lon | -12
res     | 3000
sz_sn   | 157
sz_we   | 157
pad     | 10
start   | 2015-05-04 12:00:00
stop    | 2015-05-05 00:00:00
st_len  | 30
st_spinup | 6
st_done | -4

```

wod=> select * from plot_set where dom_id=21976;

-[RECORD 1]-----

```

id      | 30154
dom_id  | 21976
ref     | full
z_svg   | {9,7}
z_png   | {7}
plot_types |
{cloud_cover_low,cloud_cover_total,composite,precipitation_accumulated,snow,tem
perature,wind,wind_chill}
plot_groups | {standard}
cen_lat  | 11
cen_lon  | -12
st_done  | 0
st_ratio | 1
height  | 411000
width    | 411000
st_cons  | 0
first_done |
last_done |

```

wod=> begin ;

BEGIN

**wod=> update plot_set set cen_lat=15.95293, cen_lon=-23.91628 where
dom_id=21976;**

UPDATE 1

wod=> select * from plot_set where dom_id=21976;

-[RECORD 1]-----

```

id      | 30154
dom_id  | 21976
ref     | full
z_svg   | {9,7}
z_png   | {7}

```

```

plot_types |
{cloud_cover_low,cloud_cover_total,composite,precipitation_accumulated,snow,tem
perature,wind,wind_chill}
plot_groups | {standard}
cen_lat    | 15.95293
cen_lon    | -23.91628
st_done    | 0
st_ratio   | 1
height     | 411000
width      | 411000
st_cons    | 0
first_done |
last_done  |

```

wod=> select * from plot_set where id=30154;

```

-[ RECORD 1 ]-----
-----
id          | 30154
dom_id      | 21976
ref         | full
z_svg       | {9,7}
z_png       | {7}
plot_types  |
{cloud_cover_low,cloud_cover_total,composite,precipitation_accumulated,snow,tem
perature,wind,wind_chill}
plot_groups | {standard}
cen_lat     | 15.95293
cen_lon     | -23.91628
st_done     | 0
st_ratio    | 1
height      | 411000
width       | 411000
st_cons     | 0

```

first_done |

last_done |

wod=> update plot_set set z_png=z_svg where id=30154;

UPDATE 1

wod=> update plot_set set z_svg=array[]::integer[] where id=30154;

UPDATE 1

wod=> select * from plot_set where id=30154;

-[RECORD 1]-----

id | 30154

dom_id | 21976

ref | full

z_svg | {}

z_png | {9,7}

plot_types |

{cloud_cover_low,cloud_cover_total,composite,precipitation_accumulated,snow,temperature,wind,wind_chill}

plot_groups | {standard}

cen_lat | 15.95293

cen_lon | -23.91628

st_done | 0

st_ratio | 1

height | 411000

width | 411000

st_cons | 0

first_done |

last_done |

wod=> commit;

COMMIT

wod=> select * from plot_set where id=30154;

```

-[ RECORD 1 ]-----
-----
id          | 30154
dom_id      | 21976
ref         | full
z_svg       | {}
z_png       | {9,7}
plot_types  |
{cloud_cover_low,cloud_cover_total,composite,precipitation_accumulated,snow,tem
perature,wind,wind_chill}
plot_groups | {standard}
cen_lat     | 15.95293
cen_lon     | -23.91628
st_done     | 1
st_ratio    | 1
height      | 411000
width       | 411000
st_cons     | 1
first_done  | 2015-05-04 16:07:49
last_done   |

```

wod=> select * from plot_set where dom_id=21976;

```

-[ RECORD 1 ]-----
-----
id          | 30154
dom_id      | 21976
ref         | full
z_svg       | {}
z_png       | {9,7}
plot_types  |
{cloud_cover_low,cloud_cover_total,composite,precipitation_accumulated,snow,tem
perature,wind,wind_chill}
plot_groups | {standard}

```

```
cen_lat   | 15.95293
cen_lon   | -23.91628
st_done   | 1
st_ratio  | 1
height    | 411000
width     | 411000
st_cons   | 1
first_done | 2015-05-04 16:07:49
last_done |
```

```
wod=> begin;
```

```
BEGIN
```

```
wod=> update domain set cen_lat=15.95293, cen_lon=-23.91628 where id=21976;
```

```
UPDATE 1
```

```
wod=> commit;
```

```
COMMIT
```

```
wod=>
```

Conductor HowTo

When a new **forecast type** has been created it is necessary to restart the WOD service on the **wod** virtual machine by doing **sudo /etc/init/wod_restart** by a user that has sudo-privileges on the machine. If it is only necessary to restart the WOD Conductor, simply do **sudo restart wod_conductor**, e.g. as user **wod**.

Also, it is necessary to restart the WOD service on all modeller machines. One needs to log on to all of them and execute **sudo service wod_modeller-0 restart**.

WOD virtual machine

When the WOD virtual machine has been started, one needs to manually start the wod service by doing **sudo /etc/init/wod_restart start** by a user that has sudo-privileges on the machine.

HowTo configure the GFSFetcher

Before submitting a forecast for the first time one has to make sure that there are sufficient input data in the **/wodroot/gfs/gfs.YYYYMMDDhh** directory. One also has to make sure that the system is aware of the latest available analysis data. Importantly, this date must not be set to far into the past, otherwise the GFSFetcher routine goes haywire. This is done by editing a configuration file **/home/wod/wod_conf/wod.conf**. Here one needs to set the values of two parameters:

- `gfs_first_analysis = 2016012218`
- `analysis_bootstrap = 2016012218`

These values should be set to the current date, i.e. the values when the GFSFetcher is first turned on.

Furthermore, one has to make sure that the file `/wodroot/gfs/gfs.analys/progress` is empty.

In order to download a sub-region of the global GFS data, one needs to edit the subregion part of the forecast URL, found in the `wod.conf` file. In the following example we are downloading an area spanning 45°W to 75°E and from 45°S to 45°N. The source data is the 0.25° GFS global forecast:

```
forecast_url = http://nomads.ncep.noaa.gov/cgi-bin/filter_gfs_0p25.pl?
dir=%2Fgfs.{analysis_date:%Y%m%d%H}&
file=gfs.t{analysis_date:%H}z.pgrb2.0p25.f{offset:03g}&
subregion=&leftlon=-45&rightlon=75&toplat=45&bottomlat=-45
```

HowTo modify the web-interface

`/home/neo/neo.git/src/wod/model` -> edit the `WodModel.js` file to put correct links to the wod API service.

`/home/neo/neo.git/src/server` -> edit `SendClient.js` and `config.js` files (for changes see e.g. the difference between `SendClient.js` and `SendClient.js_belgingur`).

Once this is done, or after any changes to e.g. `config.js` you need to restart the neo service on the web-machine:

```
sudo service neo restart
```

HowTo change the recurrence rules

update schedule set recurrence='*:00,12' where `prototype_id=XXX`;

HowTo clean the cache on the nginx web server

On the wod virtual machine do `sudo /etc/nginx/nginx_clear.sh` and then empty the cache on your local browser (if so needed). The nginx server's main task is to cache the individual plots and provide SSL wrapping for the https communications.

Some handy sql commands

```
select s.ref, s.lt_analysis, s.recurrence, s.pri_base,
```

j.submitted-j.analysis as submit_after, (time between analysis time and job submission)

t.started-j.submitted as wait, (how much time did pass since the job was submitted until a modeller machine could start running the forecast)

t.ended-t.started as model_time (i.e. how long did it take to run the forecast)

from schedule s join job j on j.id=s.lc_job_id join task t on t.job_id=j.id and t.type=4

where expires > now() order by model_time desc;

gives us the following information

ref	lt_analysis	recurrence	pri_base	submit_after	wait	model_time
island-3	2015-05-05 06:00:00		220	04:45:50	00:22:32	02:45:20
fjallabak	2015-05-05 06:00:00	*:12+SA:00+SU:00	70	04:04:14	01:06:55	02:04:51
eneco-01	2015-05-05 06:00:00		200	04:25:41	00:00:10	01:49:49
bergen	2015-05-05 12:00:00		90	03:48:38	00:00:07	01:35:21
reykjavik	2015-05-05 06:00:00		100	03:53:29	00:00:04	01:25:19
kathmandu-3	2015-05-05 06:00:00		70	04:01:01	00:00:00	01:07:16
foroyar-3	2015-05-05 06:00:00		100	04:26:34	00:52:22	00:48:03
islandshaf-9	2015-05-05 12:00:00	MO:00+WE:00+FR:00	100	08:52:53	00:00:02	00:07:55

(8 rows)

HowTo change/add clients to the wod database

Sometime it may be necessary to add and/or delete a client from the wod database on the wod virtual machine. This can be accomplished by taking the following steps. First of all log onto the database

```
sysop@wod:~$ psql -U wod -h service.gnb.wod wod
```

To list current clients do

```
wod=> select * from client;
```

To delete a client/user do

```
wod=> begin;
```

```
BEGIN
```

```
wod=> delete from client where ref='name_of_client_you_want_to_remove';
```

```
DELETE=1;
```

```
wod=> select * from client;
```

did everything work out OK, if so, commit the change (otherwise type rollback)

```
wod=> commit;
```

To change the password of a client/user, do

```
wod=> begin;
```

```
BEGIN
```

```
wod=> update client set key=md5('wodsalt'||ref||'NEW-PASSWORD') where  
ref='name_of_client_you_want_to_update' ;
```

```
UPDATE 1
```

```
wod=> select * from client; (to see if the password has been changed the value of  
"key" should be different from before)
```

```
commit ; (if you are satisfied with the changes, otherwise type rollback).
```

Some plotting tips and tricks

It is possible to use the plot part of the WOD environment in a command line mode. The following example works for wod_dev@sindri.belgingur.is:

```
cd git
```

```
. activate wod
```

```
plotter/generate_plots.py --help
```

```
python plotter/generate_plots.py --wrfout /riv/hawwa/2011-2012/wrfout_d02_2011-  
12-07_12:00:00 --steps-to 1 --plot-types wind,terrain --zooms=8 --geo  
/riv/hawwa/geo_emRAV/geo_em.d02.nc --margin=5 --replot
```

To get a list of available plots, do:

```
(wod)wod_dev@sindri:~/git$ plotter/generate_plots.py --list-plots
```

Available plot groups and the plot types belonging to each:

aviation.1:

wind_agl_0100, wind_agl_0150, wind_agl_0250, wind_agl_0500, wind_agl_1000,
wind_asl_02500, wind_asl_05000

aviation.3:

wind_agl_0150, wind_agl_0500, wind_agl_1000, wind_asl_02500, wind_asl_05000,
wind_asl_10000, wind_asl_18000

aviation.9:

wind_asl_05000, wind_asl_10000, wind_asl_18000, wind_asl_24000, wind_asl_30000,
wind_asl_34000, wind_asl_39000

standard:

cloud_cover_low, cloud_cover_total, composite, precipitation_accumulated, snow,
temperature, wind, wind_chill

volcanic:

volcanic, volcanic_agl_0300m, volcanic_agl_0700m, volcanic_agl_1000m,
volcanic_agl_1500m, volcanic_agl_3000m, volcanic_asl_10000m

wind.aviation:

wind_asl_0500m, wind_asl_1000m, wind_asl_1500m, wind_asl_2000m,
wind_asl_2500m, wind_asl_3000m

wind_farm:

composite, wind, wind_lv

Plots which belong to no group:

cloud_base, cloud_cover_middle, precipitation, terrain, visibility

(wod)wod_dev@sindri:~/git\$

In case of plots not being produced

If the system sends out an error e-mail stating that there are too few plot workers it is a good idea to check the `/var/log/wod/wod.log` file. In case of "Failed to send status" errors, one should check if the conductor is running by giving the command `sudo status wod_conductor`. If the conductor is up'n running one might have to restart the plotter process thusly `sudo`

`/etc/init/wod_restart_plotters`. Individual plot-processes can also be killed, but the ID of individual processes is given in the `wod.log` file. Another way to do this is by restarting individual workers: `sudo restart wod_worker-?`, the problem is that it is very difficult to know which worker is causing the problem. If one wants to restart all worker individually one can do something like this: `for p in 0 1 2 3 4 5 6 7 8 9 a b c d ; do sudo service wod_plotter-${p} restart ; done.`

Post-processing: generating point forecasts

You can generate a point forecast from a single wrfout file, defining which variables and stations/points of interest you want your point forecasts for. The script will produce one point forecast file per station/poi, including timeseries for the defined variables and some metadata.

If you have collected observations and used linear regression procedure to generate weights for point forecast, you can use a file with weights for the same forecast schedule as your wrfout file, containing weights for stations and components (variables) that you use. If a station/component is not found in a weights file, or if the weights file is not provided, a bilinear interpolation from the grid to the station's location would be performed. For the metadata about stations (location etc.) the script can use database table or, as the simplest case, a yaml file.

The minimum set up for a new point forecast generation procedure for a list of stations/points of interest includes:

*** preparing a yaml file with metadata about your locations**

the minimal set of keys for the metadata is: lon, lat, id, name, ref.

Ref is the reference that will be used in point forecast config file to indicate locations we want point forecasts for. Usually, refs are created by concatenating the initials of the provider, country and (a part of) station name, e.g. for Icelandic station "Reykjavik" of Vedurstofa Islands it is "vi.is.rvk"

*** preparing a config (yaml) file describing variables and stations you want to use for your forecast.**

The description for the config file entries can be found in `PointForecast/pf_template.yml`. The fields in a config file that you might want to use are: `station_metadata`, `poi_metadata`, `stations`, `poi`, `output_template`, `store_dir`, `spinup`. Use 'station' when your locations are those you will have observations from, i.e. they are real observation stations. Otherwise, if you have a set of locations you just want your forecasts for, use `poi`.

Your yaml files can have arbitrary names, but make sure you refer to them properly (as poi/station metadata and as --config when running the script).

*** running the script for selected wrf output files or generate a trigger rule for automatic generation of point forecasts after each wrfout producing job is done.**

Running PointForecast/point_forecast.py -help will print the available parameters for running the point forecast.

The components you can create a point forecast for can be: temp, wind_speed, wind_dir, prec_rate, snow_ratio, pressure, mslp, humidity, rel_hum, total_clouds, or any non-staggered, three-dimensional (lat, lon, time) variable that is present in a wrfout file.

Step-by-step instructions of configuring point forecast generation and setting it up within a trigger rule.

0. Preparing your environment

If you haven't used anything from the WeatherHills system before, you have to make sure it is properly set up. All you need is a virtual environment called weather_hills, that should be available in a path similar to ~/anaconda/envs/weather_hills.

A. Configuring point forecast generation

1. Create your file with metadata about places you want your forecasts for. Make "ref" values contain a common prefix indicating your country/organization. Save it as locations.yml. The format of the file should be according to the example:

```
- {id: 1, lat: 40.2, lon: 17.4, name: Capital City, ref: my.capital}
- {id: 2, lat: 40.23, lon: 17.48, name: Seaside, ref: my.seaside}
- {id: 3, lat: 40.27, lon: 17.52, name: Highlands, ref: my.highlands}
- {id: 4, lat: 49.85, lon: 17.6, name: Lake District Town, ref: my.lakes}
```

2. Create your config file (call it pf_config.yml), with the following fields:

```
'store_dir' = 'path/to/your/data' # where you want your point forecasts to be      stored
'poi' = 'vi.is.%' # means all stations/poi that have "ref" starting from          vi.is. Change to your
prefix, indicating the stations you have in locations.yml
'poi_metadata' = 'locations.yml' # path to the file describing station            metadata you
created
```

3. You can now activate your virtual environment and call point forecast script by this commands

```
. activate weather_hills
```

```
python [path_to_WeatherHills]/PointForecast/point_forecast.py --config pf_config.yml --
components wind_dir wind_speed temp --analysis [your analysis date in format like 2015-07-
28T00:00:00 ] --wrfout [path to your wrfout]
```

B. Setting up routine point forecast generation within a trigger rule

1. Ensuring the directory structure

Create a directory that you will use to run the point forecast script. Create symbolic link called 'input' to the directory where the results of your jobs are stored. Create a directory called 'output' where you will collect your point forecasts. Put your pf_config.yml and poi.yml files in this directory. It is worth to add/edit the following entries to your config file content:

```
output_template: '{analysis_date:%Y-%m-%dT%H:%M:%S}/pf-{ref}.csv' # this way it
will create a new directory for each forecast. Make sure to set 'store_dir' to 'output' (the
directory you just created)

spinup: 3 # if you want to omit the first few timestamps from your forecast
```

2. Adding a trigger rule for point forecast generation: analogously to other rules in your trigger setup file. It is handy and keeps your trigger config clean to create a shell script running the command and call this script within the trigger rule.

```
point-forecast: # you can change this name to something more meaningful to
                you
  exchange: 'jobs.oper' # change this if your main exchange emitting
                    message with model-done key has a different name
  key: 'job.oper.island-3.*.model-done' # change to an analogous key in
                    your system
  dir: '${TJ_DIR}/point-forecast_vi.is' # this is the directory where
                    your script will be ran, change it appropriately
  cmd: './pf.sh {analysis} input/{dir}/model/{wrfout_d02}'
  timeout: 10
```

3. Put the following script in the directory you created in 1, ensuring you replace the paths, config file name and components with what suits you.

The sample script:

```
#!/bin/bash
analysis=$1
wrfout=$2
HOME=/home/oper
WH_PY=$HOME/miniconda/envs/weather_hills/bin/python
WH_DIR=$HOME/code/WeatherHills.git
${WH_PY} ${WH_DIR}/PointForecast/point_forecast.py \
  --config pf.yml \
  --components temp wind_speed wind_dir prec_rate total_clouds mslp\
  --analysis ${analysis} \
  --wrfout ${wrfout}
```