WIS2 in a box

Release 1.0b3

World Meteorological Organization (WMO)

2023-05-11

USER GUIDE

WIS2 in a box (wis2box) is a Free and Open Source (FOSS) Reference Implementation of a WMO WIS2 Node. The project provides a plug and play toolset to ingest, process, and publish weather/climate/water data using standards-based approaches in alignment with the WIS2 principles. wis2box also provides access to all data in the WIS2 network. wis2box is designed to have a low barrier to entry for data providers, providing enabling infrastructure and services for data discovery, access, and visualization.

wis2box enables World Meteorological Organization (WMO) members to publish and download data through the WIS2 network. The main features are:

- WIS2 compliant: easily register your wis2box to WIS2 infrastructure, conformant to WMO data and metadata standards
- WIS2 compliance: enables sharing of data to WIS2 using standards in compliance with WIS2 technical regulations
- event driven or interactive data ingest/process/publishing pipelines
- visualization of stations/data on interactive maps
- discovery metadata management and publishing
- download/access of data from WIS 2 network to your local environment
- standards-based data services and access mechanisms:
- robust and extensible plugin framework. Write your own data processing engines and integrate seamlessly into wis2box!
- Free and Open Source (FOSS)
- containerized: use of Docker, enabling easy deployment to cloud or on-premises infrastructure

Live demonstration instances of wis2box can be found at at https://demo.wis2box.wis.wmo.int.

CHAPTER

ONE

USER GUIDE

The user guide helps you setup your own wis2box instance.

1.1 Introduction

This is a user guide for publishing and downloading data through the WIS2 network using the wis2box software.

wis2box provides a set of services to help you ingest, transform and publish your weather/climate/water data.

wis2box implements the core WIS2 requirements of a WIS2 Node:

- Module to produce WIS2 compliant notifications
- MQTT broker
- HTTP endpoint to enable data download

Additional services included in wis2box include:

- Customizable plugins to transform input data
- API exposing data in GeoJSON using pygeoapi
- Monitoring functions using Prometheus and Grafana
- Data visualization through the wis2box user interface

Next: *Getting started*.

1.2 Getting started

wis2box can be run on any Linux instance (bare metal or cloud hosted VM) with Python, Docker and Docker Compose installed. The recommended OS is Ubuntu 22.04 LTS.

1.2.1 System requirements

System requirements depend on the amount of data ingested. We recommend minimum 2vCPUs, 4GB Memory and 16GB of local storage.

For example, the following Amazon AWS ec2-instance-types have been utilized as part of wis2box demonstrations.

- 0 - 2000 observations per day: "t3a.medium"-instance: 2vCPUs, x86_64 architecture, 4GB Memory, up to 5 Gigabit network, 16GB attached storage (~35 USD per month for on-demand Linux based OS)
- 2000 - 10000 observations per day: "t3a.large"-instance: 2vCPUs, x86_64 architecture, 8GB Memory, up to 5 Gigabit network, 24GB attached storage (~70 USD per month for on-demand Linux based OS)

1.2.2 Software dependencies

The services in wis2box are provided through a stack of Docker containers, which are configured using Docker Compose.

wis2box requires the following prior to installation:

Requirement	Version
Python	3.8 or higher
Docker Engine	20.10.14 or higher
Docker Compose	1.29.2

The following commands can be used to inspect the available versions of Python, Docker and Docker Compose on your system:

```
docker version
docker-compose version
python3 -V
```

Once you have verified these requirements, go to *Installation and configuration* for a step-by-step guide to install and configure your wis2box.

1.3 Installation and configuration

This section summarizes the steps required to install a wis2box instance and setup your own datasets using example configurations.

Ensure you have Docker, Docker Compose and Python installed on your host, as detailed in *Getting started*.

1.3.1 Download

Download the wis2box setup files from the wis2box Releases page. Go to the latest release and download the `wis2box-setup-1.0b3.zip` file from the Assets section.

```
wget https://github.com/wmo-im/wis2box/releases/download/1.0b3/wis2box-setup-1.0b3.zip
unzip wis2box-setup-1.0b3.zip
cd wis2box-1.0b3
```

1.3.2 Environment variables

wis2box uses environment variables from `dev.env` to its containers on startup. An example file is provided in `examples/config/wis2box.extended.env`. Copy this file to your working directory, and update it to suit your needs.

```
cp examples/config/wis2box.env dev.env
```

Note: You must map `WIS2BOX_HOST_DATADIR` to the absolute path of a directory on your host machine. This path will be mapped to `/data/wis2box` inside the wis2box-management container To enable external data sharing you must set `WIS2BOX_URL` to the URL pointing to where your host is exposed on the public network.

Note: Please ensure you set `WIS2BOX_BROKER_PASSWORD` and `WIS2BOX_STORAGE_PASSWORD` to your own unique values.

You will use these passwords to connect to your broker and MinIO storage to help you debug your wis2box services.

Do not share these passwords with external parties.

The next sections assume you use an environment variable for `WIS2BOX_HOST_DATADIR` that is set to same value used in `dev.env`:

```
export WIS2BOX_HOST_DATADIR=/home/example/wis2box-data
```

1.3.3 Data mappings

wis2box configuration requires a data mappings file, which defines the plugins used to process your data. Example mapping files are included in the release archive:

- `synop-bufr-mappings.yml`, input is binary data (BUFR) defined by a `.bufr` extension
- `synop-csv-mappings.yml`, input is comma-separated-values defined by a `.csv` extension
- `synop-synop-mappings.yml`, input is SYNOP defined with a `.txt` extension

For example, if your incoming data contains `.bufr4` files containing synoptic observations, you can copy the following example:

```
cp synop-bufr-mappings.yml ${WIS2BOX_HOST_DATADIR}/data-mappings.yml
```

Note: The file should be called `data-mappings.yml` and should be placed in the directory you defined as `WIS2BOX_HOST_DATADIR`.

Edit `${WIS2BOX_HOST_DATADIR}/data-mappings.yml`:

- Replace `country` with your corresponding ISO 3166 alpha-3 country code in lowercase
- Replace `centre_id` with the string identifying the centre running your wis2node in lowercase, alphanumeric characters

If you need to define multiple datasets, you can add multiple entries in your `data-mappings.yml`. For example:

```
data:
   ita.italy_wmo_demo.data.core.weather.surface-based-observations.synop:
     plugins:
         bufr:
             - plugin: wis2box.data.bufr4.ObservationDataBUFR
               notify: true
               buckets:
                 - ${WIS2BOX_STORAGE_INCOMING}
               file-pattern: '*'
         bufr4:
             - plugin: wis2box.data.bufr2geojson.ObservationDataBUFR2GeoJSON
               buckets:
                 - ${WIS2BOX_STORAGE_PUBLIC}
               file-pattern: '^WIGOS_(\d-\d+-\d+-\w+)_.*\.bufr4$'
   ita.italy_wmo_demo.data.core.weather.surface-based-observations.temp:
     plugins:
         bufr:
             - plugin: wis2box.data.bufr4.ObservationDataBUFR
               notify: true
               buckets:
                 - ${WIS2BOX_STORAGE_INCOMING}
               file-pattern: '*'
         bufr4:
             - plugin: wis2box.data.bufr2geojson.ObservationDataBUFR2GeoJSON
               buckets:
                 - ${WIS2BOX_STORAGE_PUBLIC}
               file-pattern: '^WIGOS_(\d-\d+-\d+-\w+)_.*\.bufr4$'
```

In this case the data mappings configuration has specified 2 datasets (SYNOP, and TEMP).

You can also combine input for the same dataset provided in different formats. For example, if you would like to process input data that is provided both as SYNOP and binary data:

```
data:
   ita.italy_wmo_demo.data.core.weather.surface-based-observations.synop:
     plugins:
         bufr:
             - plugin: wis2box.data.bufr4.ObservationDataBUFR
               notify: true
               buckets:
                 - ${WIS2BOX_STORAGE_INCOMING}
               file-pattern: '*'
         csv:
             - plugin: wis2box.data.csv2bufr.ObservationDataCSV2BUFR
               template: synop_bufr.json
               notify: true
               file-pattern: '*'
         bufr4:
             - plugin: wis2box.data.bufr2geojson.ObservationDataBUFR2GeoJSON
               buckets:
                 - ${WIS2BOX_STORAGE_PUBLIC}
               file-pattern: '^WIGOS_(\d-\d+-\d+-\w+)_.*\.bufr4$'
```

Note: The dataset identifier is used to define the topic hierarchy for your data (see WIS2 topic hierarchy). The top 3 levels of the WIS2 topic hierarchy (`origin/a/wis2`) are automatically included by wis2box when publishing your data.

- dataset: ita.italy_wmo_demo.data.core.weather.surface-based-observations.synop
- topic-hierarchy: origin/a/wis2/ita/italy_wmo_demo/data/core/weather/surface-based-observations/synop

Note: In these examples, files in the `wis2box-incoming` storage bucket are processed to produce `.bufr4` stored in the `wis2box-public` storage bucket, using either the `bufr4.ObservationDataBUFR` or the `wis2box.data.csv2bufr.ObservationDataCSV2BUFR` plugins.

Files in the `wis2box-public` storage bucket are converted to GeoJSON and stored in the wis2box API backend using the `wis2box.data.bufr2geojson.ObservationDataBUFR2GeoJSON` plugin.

You can provide your own plugins as needed; for more information (see *Extending wis2box*).

1.3.4 Station metadata list

wis2box requires information about the stations for which you will be sharing data.

An example of the configuration file for the stations is provided in `station_list.csv`.

You can copy this file to `metadata/station/station_list.csv` in your $WIS2BOX_HOST_DATADIR :

```
mkdir -p ${WIS2BOX_HOST_DATADIR}/metadata/station
cp station_list.csv ${WIS2BOX_HOST_DATADIR}/metadata/station
```

And edit `${WIS2BOX_HOST_DATADIR}/metadata/station/station_list.csv` to include the data for your stations.

Note: The `station_list.csv` requires column names `station_name` and the `wigos_station_identifier` (WSI) with which the station is registered in OSCAR. Optionally, you can provide a `traditional_station_identifier (TSI)` column. The TSI can be left empty if your data contains a WSI. If your data contains a TSI but no WSI, the `station_list.csv` will be used to derive the corresponding WSI for that station.

1.3.5 Discovery metadata

Discovery metadata provides the data description needed for users to discover your data when searching the WIS2 Global Discovery Catalogue.

Updated discovery metadata records are shared globally through the MQTT endpoint defined in your wis2box.

Discovery metadata records can be defined using the YAML syntax shared via `WIS2BOX_HOST_DATADIR`.

An example is provided in `surface-weather-observations.yml`. Each dataset requires its own discovery metadata configuration file.

You can copy the file `surface-weather-observations.yml` to the directory you defined for `WIS2BOX_HOST_DATADIR` and update it to provide the correct discovery metadata for your dataset:

- replace `[country].[centre_id].data.core.weather.surface-based-observations.synop` with the topic as previously used in `$WIS2BOX_HOST_DATADIR/data-mappings.yml`
- text provided in `identification.title` and `identification.abstract` will be displayed in the wis2box user interface
- provide a valid geographic bounding box in `identification.extents.spatial.bbox`

1.3.6 Starting wis2box

Once you have prepared the necessary configuration files as described above you are ready to start the wis2box.

Run the following command to start wis2box:

```
python3 wis2box-ctl.py start
```

This might take a while the first time, as Docker images will be downloaded.

Note: The `wis2box-ctl.py` program is used as a convenience utility around a set of Docker Compose commands. You can customize the ports exposed on your host by editing `docker-compose.override.yml`.

Once the command above is completed, check that all services are running (and healthy).

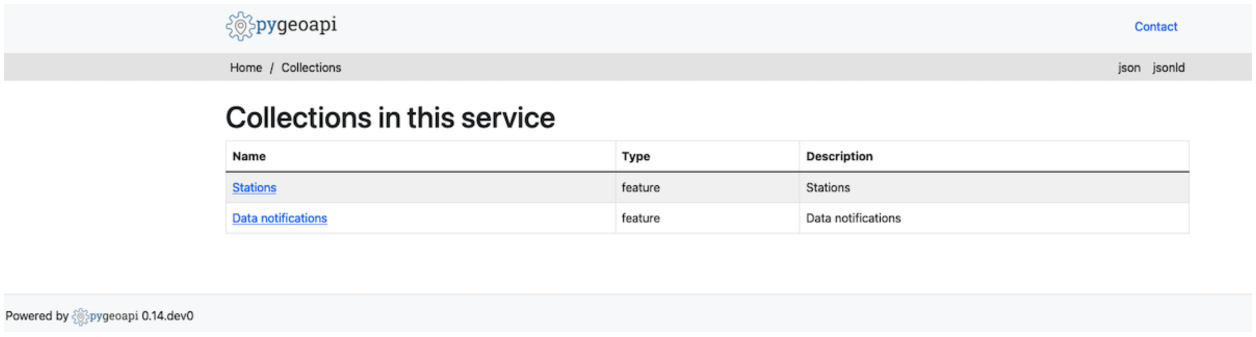
```
python3 wis2box-ctl.py status
```

Which should display the following:

```
        Name                        Command                   State                   ␣
↪   Ports
------------------------------------------------------------------------------------------
↪------------------------------
cadvisor                 /usr/bin/cadvisor -logtostderr   Up (healthy)   8080/tcp
elasticsearch            /bin/tini -- /usr/local/bi ...   Up (healthy)   9200/tcp, 9300/
↪tcp
grafana                  /run.sh                          Up             0.0.0.0:3000->
↪3000/tcp
loki                     /usr/bin/loki -config.file ...   Up             3100/tcp
mosquitto                /docker-entrypoint.sh /usr ...   Up             0.0.0.0:1883->
↪1883/tcp, 0.0.0.0:8884->8884/tcp
mqtt_metrics_collector   python3 -u mqtt_metrics_co ...   Up             8000/tcp, 0.0.0.
↪0:8001->8001/tcp
nginx                    /docker-entrypoint.sh ngin ...   Up             0.0.0.0:80->80/
↪tcp
prometheus               /bin/prometheus --config.f ...   Up             9090/tcp
wis2box                  /entrypoint.sh wis2box pub ...   Up
wis2box-api              /app/docker/es-entrypoint.sh     Up
wis2box-auth             /entrypoint.sh                   Up
wis2box-minio            /usr/bin/docker-entrypoint ...   Up (healthy)   0.0.0.0:9000->
↪9000/tcp, 0.0.0.0:9001->9001/tcp
wis2box-ui               /docker-entrypoint.sh ngin ...   Up             0.0.0.0:9999->
↪80/tcp
```

Refer to the *Troubleshooting* section if this is not the case.

You should now be able to view collections on the wis2box API by visiting `http://localhost/oapi/collections` in a web browser, which should appear as follows:



Collections in this service

Name	Type	Description
Stations	feature	Stations
Data notifications	feature	Data notifications

Powered by pygeoapi 0.14.dev0

The API will show one (initially empty) collection 'Data Notifications'. This collection will be filled when you start ingesting data and publishing WIS2 notifications.

Note: Additional collections will be added during the runtime configuration.

1.3.7 Runtime configuration

The following last design time steps are then required once wis2box is running.

Login to the wis2box-management container

```
python3 wis2box-ctl.py login
```

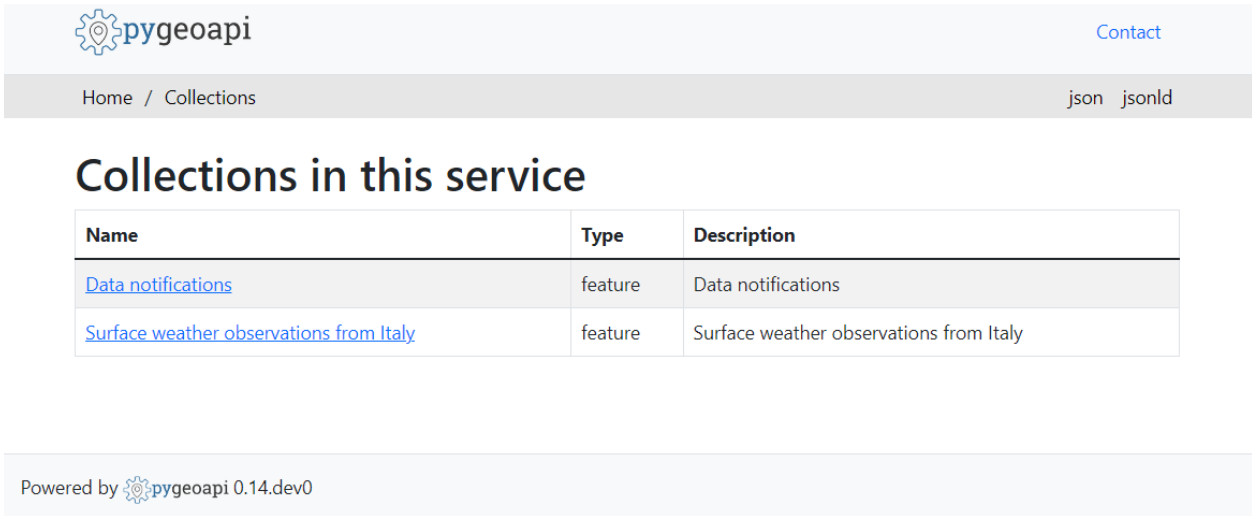
Note: `$WIS2BOX_DATADIR` is the location that `$WIS2BOX_HOST_DATADIR` binds to **inside** the container. This allows wis2box to access the configuration files from **inside** the wis2box-management container. By default, `WIS2BOX_DATADIR` points to `/data/wis2box` **inside** the wis2box-management container.

The first step is add the new dataset as defined by the YAML file for your discovery metadata record defined previously, using the following command:

```
wis2box data add-collection ${WIS2BOX_HOST_DATADIR}/surface-weather-observations.yml
```

Note: If you see an error like `ValueError: No plugins for XXX defined in data mappings`, exit the wis2box-container and edit the `data-mappings.yml` file in the directory defined by `WIS2BOX_HOST_DATADIR`

You can view the collection you just added, by re-visiting `http://localhost/oapi/collections` in a web browser.



Collections in this service

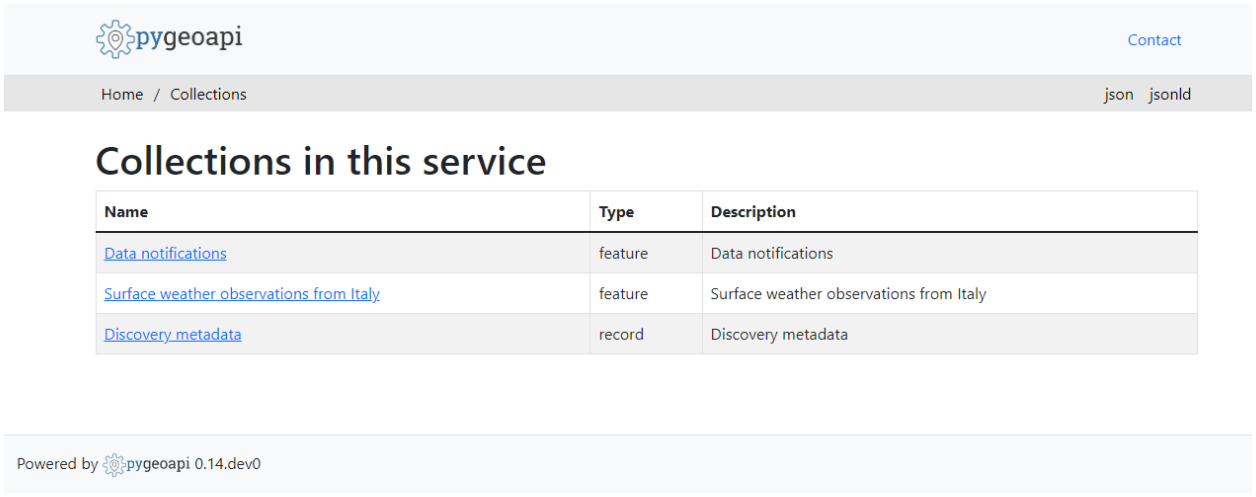
Name	Type	Description
Data notifications	feature	Data notifications
Surface weather observations from Italy	feature	Surface weather observations from Italy

The second step is to publish discovery metadata and cache its content in the wis2box API:

```
wis2box metadata discovery publish ${WIS2BOX_HOST_DATADIR}/surface-weather-observations.
↪yml
```

This command publishes an MQTT message with information about your dataset to the WIS2 Global Discovery Catalogue. Repeat this command whenever you have to provide updated metadata about your dataset.

You can review the discovery metadata just cached through the new link in `http://localhost/oapi/collections`:



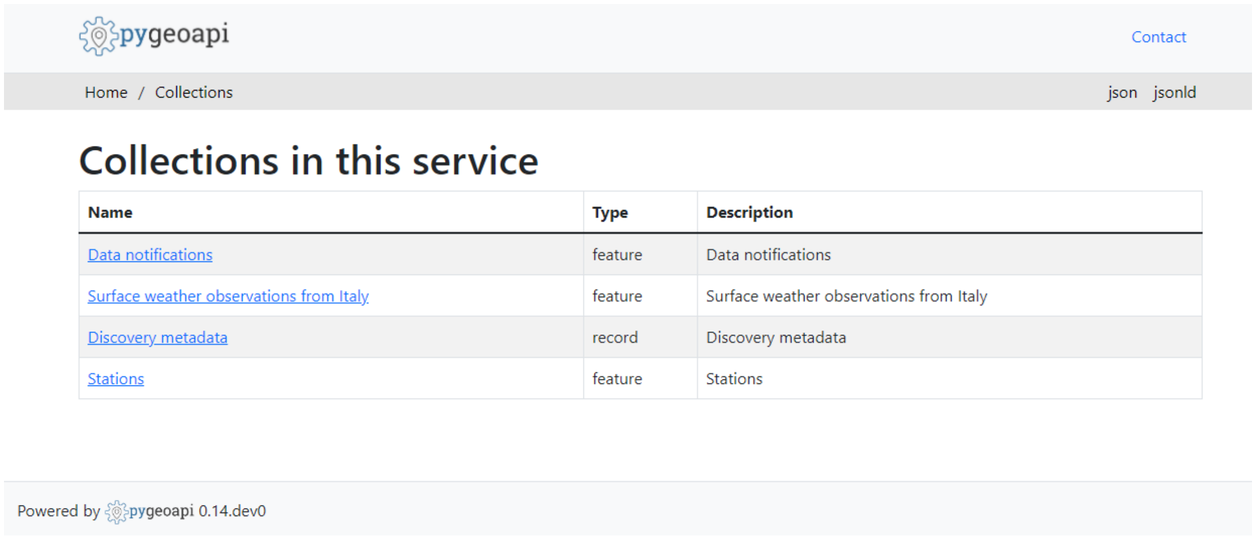
Collections in this service

Name	Type	Description
Data notifications	feature	Data notifications
Surface weather observations from Italy	feature	Surface weather observations from Italy
Discovery metadata	record	Discovery metadata

The final step is to publish your station information to the wis2box API from the station metadata list you prepared:

```
wis2box metadata station publish-collection
```

You can review the stations you just cached through the new link in `http://localhost/oapi/collections`:



Collections in this service

Name	Type	Description
Data notifications	feature	Data notifications
Surface weather observations from Italy	feature	Surface weather observations from Italy
Discovery metadata	record	Discovery metadata
Stations	feature	Stations

Powered by pygeoapi 0.14.dev0

You can now logout of wis2box-management container:

```
exit
```

The next is the *Data ingest setup*.

1.4 Data ingest setup

The runtime component of wis2box is data ingestion. This is an event driven workflow driven by S3 notifications from uploading data to wis2box storage.

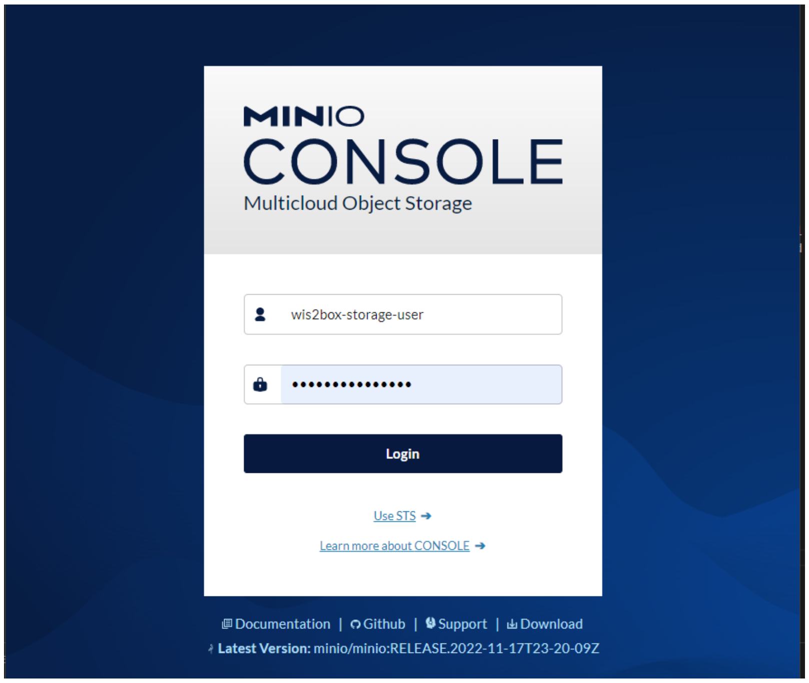
The wis2box storage is provided using a MinIO container that provides S3-compatible object storage.

Any file received in the `wis2box-incoming` storage bucket will trigger an action to process the file. What action to take is determined by the `data-mappings.yml` you've setup in the previous section.

1.4.1 MinIO user interface

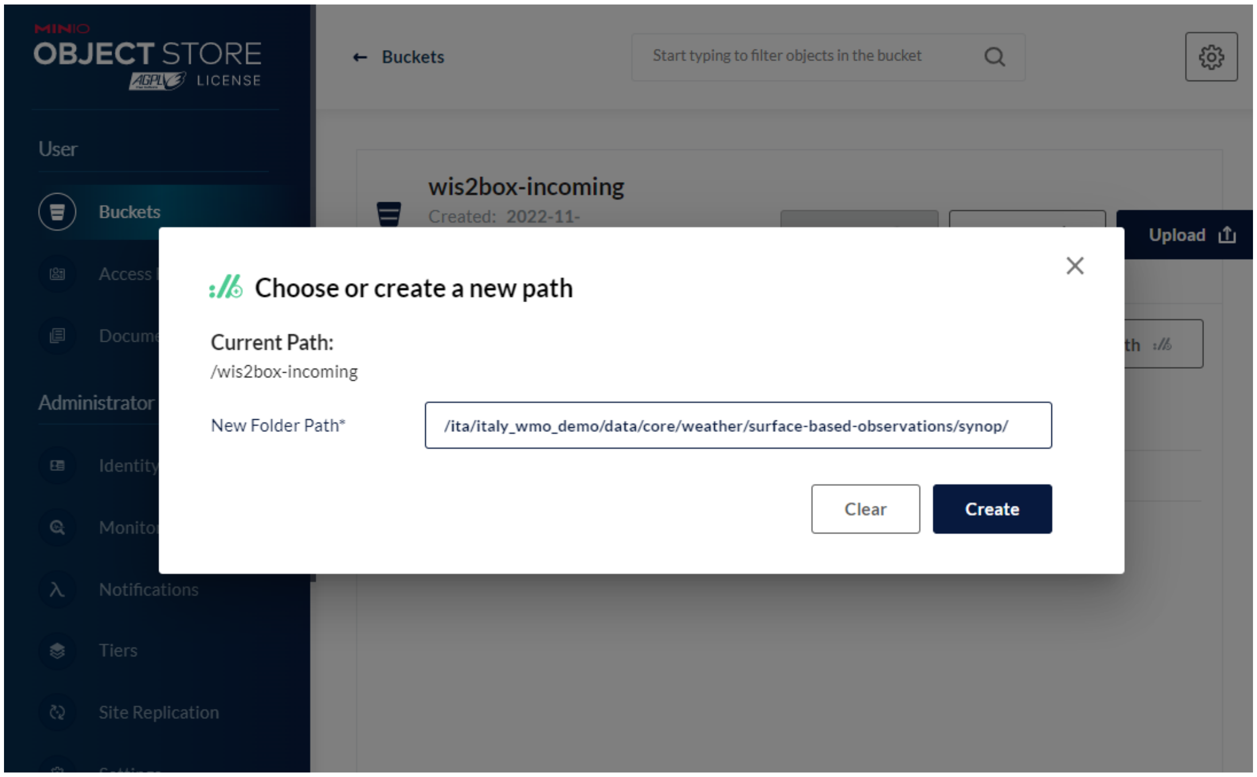
To access the MinIO user interface, visit `http://localhost:9001` in your web browser.

You can login with your `WIS2BOX_STORAGE_USERNAME` and `WIS2BOX_STORAGE_PASSWORD`:


MINIO
CONSOLE
Multicloud Object Storage
wis2box-storage-user
Login
Use STS
Learn more about CONSOLE
Documentation | Github | Support | Download
Latest Version: minio/minio:RELEASE.2022-11-17T23-20-09Z

To test the data ingest, add a sample file for your observations in the `wis2box-incoming` storage bucket.

Select 'browse' on the `wis2box-incoming` bucket and select 'Choose or create a new path' to define a new folder path:



Current Path:
/wis2box-incoming

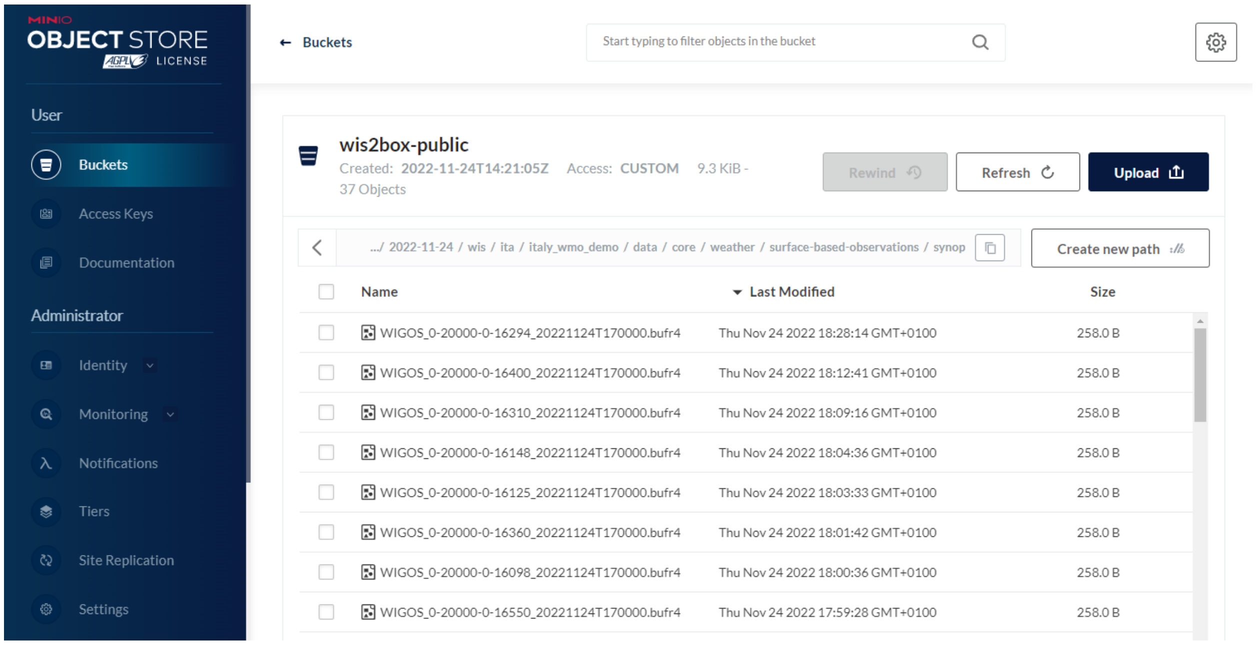
New Folder Path*: /ita/italy_wmo_demo/data/core/weather/surface-based-observations/synop/

Note: The folder in which the file is placed defines the dataset for the data you are sharing. For example, for dataset `foo.bar`, store your file in the path `/foo/bar/`.

The path is also used to define the topic hierarchy for your data (see WIS2 topic hierarchy). The first 3 levels of the WIS2 topic hierarchy `origin/a/wis2` are automatically included by wis2box when publishing data notification messages.

- The error message `Topic Hierarchy validation error: No plugins for minio:9000/wis2box-incoming/... in data mappings` indicates you stored a file in a folder for which no matching dataset was defined in your `data-mappings.yml`.

After uploading a file to `wis2box-incoming` storage bucket, you can browse the content in the `wis2box-public` bucket. If the data ingest was successful, new data will appear as follows:



wis2box-public
Created: 2022-11-24T14:21:05Z Access: CUSTOM 9.3 KiB - 37 Objects

.../ 2022-11-24 / wis / ita / italy_wmo_demo / data / core / weather / surface-based-observations / synop

Name	Last Modified	Size
WIGOS_0-20000-0-16294_20221124T170000.bufr4	Thu Nov 24 2022 18:28:14 GMT+0100	258.0 B
WIGOS_0-20000-0-16400_20221124T170000.bufr4	Thu Nov 24 2022 18:12:41 GMT+0100	258.0 B
WIGOS_0-20000-0-16310_20221124T170000.bufr4	Thu Nov 24 2022 18:09:16 GMT+0100	258.0 B
WIGOS_0-20000-0-16148_20221124T170000.bufr4	Thu Nov 24 2022 18:04:36 GMT+0100	258.0 B
WIGOS_0-20000-0-16125_20221124T170000.bufr4	Thu Nov 24 2022 18:03:33 GMT+0100	258.0 B
WIGOS_0-20000-0-16360_20221124T170000.bufr4	Thu Nov 24 2022 18:01:42 GMT+0100	258.0 B
WIGOS_0-20000-0-16098_20221124T170000.bufr4	Thu Nov 24 2022 18:00:36 GMT+0100	258.0 B
WIGOS_0-20000-0-16550_20221124T170000.bufr4	Thu Nov 24 2022 17:59:28 GMT+0100	258.0 B

If no data appears in the `wis2box-public` storage bucket, you can inspect the logs from the command line:

```
python3 wis2box-ctl.py logs wis2box
```

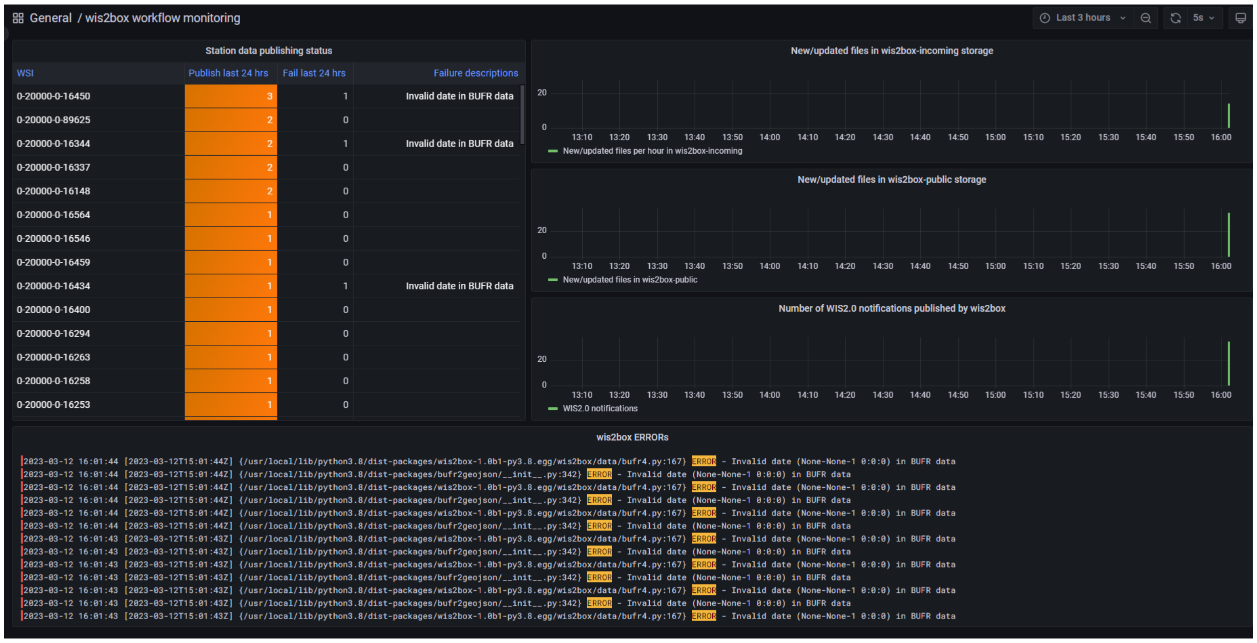
Or by visiting the local Grafana instance running at `http://localhost:3000`

1.4.2 wis2box workflow monitoring

The Grafana homepage shows an overview with the number of files received, new files produced and WIS2 notifications published.

The *Station data publishing status* panel (on the left side) shows an overview of notifications and failures per configured station.

The *wis2box ERRORs* panel (on the bottom) prints all ERROR messages reported by the wis2box-management container.


General / wis2box workflow monitoring
Last 3 hours
5s
New/updated files in wis2box-incoming storage
New/updated files per hour in wis2box-incoming
New/updated files in wis2box-public storage
New/updated files in wis2box-public
Number of WIS2.0 notifications published by wis2box
WIS2.0 notifications

Station data publishing status

WSI	Publish last 24 hrs	Fail last 24 hrs	Failure descriptions
0-20000-0-16450	3	1	Invalid date in BUFR data
0-20000-0-89625	2	0	
0-20000-0-16344	2	1	Invalid date in BUFR data
0-20000-0-16337	2	0	
0-20000-0-16148	2	0	
0-20000-0-16564	1	0	
0-20000-0-16546	1	0	
0-20000-0-16459	1	0	
0-20000-0-16434	1	1	Invalid date in BUFR data
0-20000-0-16400	1	0	
0-20000-0-16294	1	0	
0-20000-0-16263	1	0	
0-20000-0-16258	1	0	
0-20000-0-16253	1	0	

wis2box ERRORs

```
2023-03-12 16:01:44 [2023-03-12T15:01:44Z] {/usr/local/lib/python3.8/dist-packages/wis2box-1.0b1-py3.8.egg/wis2box/data/bufr4.py:167} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:44 [2023-03-12T15:01:44Z] {/usr/local/lib/python3.8/dist-packages/bufr2geojson/__init__.py:342} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:44 [2023-03-12T15:01:44Z] {/usr/local/lib/python3.8/dist-packages/wis2box-1.0b1-py3.8.egg/wis2box/data/bufr4.py:167} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:44 [2023-03-12T15:01:44Z] {/usr/local/lib/python3.8/dist-packages/bufr2geojson/__init__.py:342} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:44 [2023-03-12T15:01:44Z] {/usr/local/lib/python3.8/dist-packages/wis2box-1.0b1-py3.8.egg/wis2box/data/bufr4.py:167} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:44 [2023-03-12T15:01:44Z] {/usr/local/lib/python3.8/dist-packages/bufr2geojson/__init__.py:342} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:43 [2023-03-12T15:01:43Z] {/usr/local/lib/python3.8/dist-packages/wis2box-1.0b1-py3.8.egg/wis2box/data/bufr4.py:167} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:43 [2023-03-12T15:01:43Z] {/usr/local/lib/python3.8/dist-packages/bufr2geojson/__init__.py:342} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:43 [2023-03-12T15:01:43Z] {/usr/local/lib/python3.8/dist-packages/wis2box-1.0b1-py3.8.egg/wis2box/data/bufr4.py:167} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:43 [2023-03-12T15:01:43Z] {/usr/local/lib/python3.8/dist-packages/bufr2geojson/__init__.py:342} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:43 [2023-03-12T15:01:43Z] {/usr/local/lib/python3.8/dist-packages/wis2box-1.0b1-py3.8.egg/wis2box/data/bufr4.py:167} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:43 [2023-03-12T15:01:43Z] {/usr/local/lib/python3.8/dist-packages/bufr2geojson/__init__.py:342} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:43 [2023-03-12T15:01:43Z] {/usr/local/lib/python3.8/dist-packages/wis2box-1.0b1-py3.8.egg/wis2box/data/bufr4.py:167} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
```

Once you have verified that the data ingest is working correctly you can prepare an automated workflow to send your data into wis2box.

1.4.3 Automating data ingestion

See below a Python example to upload data using the MinIO package:

```python
import glob
import sys

from minio import Minio

filepath = '/home/wis2box-user/local-data/mydata.bin'
minio_path = '/ita/italy_wmo_demo/data/core/weather/surface-based-observations/synop/'

endpoint = 'http://localhost:9000'
WIS2BOX_STORAGE_USERNAME = '<your-wis2box-storage-username>'
WIS2BOX_STORAGE_PASSWORD = '<your-wis2box-storage-password>'

client = Minio(
    endpoint=endpoint,
    access_key=WIS2BOX_STORAGE_USERNAME,
    secret_key=WIS2BOX_STORAGE_PASSWORD,
    secure=is_secure=False)

filename = filepath.split('/')[-1]
client.fput_object('wis2box-incoming', minio_path+filename, filepath)
```

1.4.4 wis2box-ftp

You can add an additional service to allow your data to be accessible over FTP.

To use the `docker-compose.wis2box-ftp.yml` template included in wis2box, create a new file called `ftp.env` using any text editor, and add the following content:

```
MYHOSTNAME=hostname.domain.tld

FTP_USER=wis2box
FTP_PASS=wis2box123
FTP_HOST=${MYHOSTNAME}

WIS2BOX_STORAGE_ENDPOINT=http://${MYHOSTNAME}:9000
WIS2BOX_STORAGE_USER=minio
WIS2BOX_STORAGE_PASSWORD=minio123

LOGGING_LEVEL=INFO
```

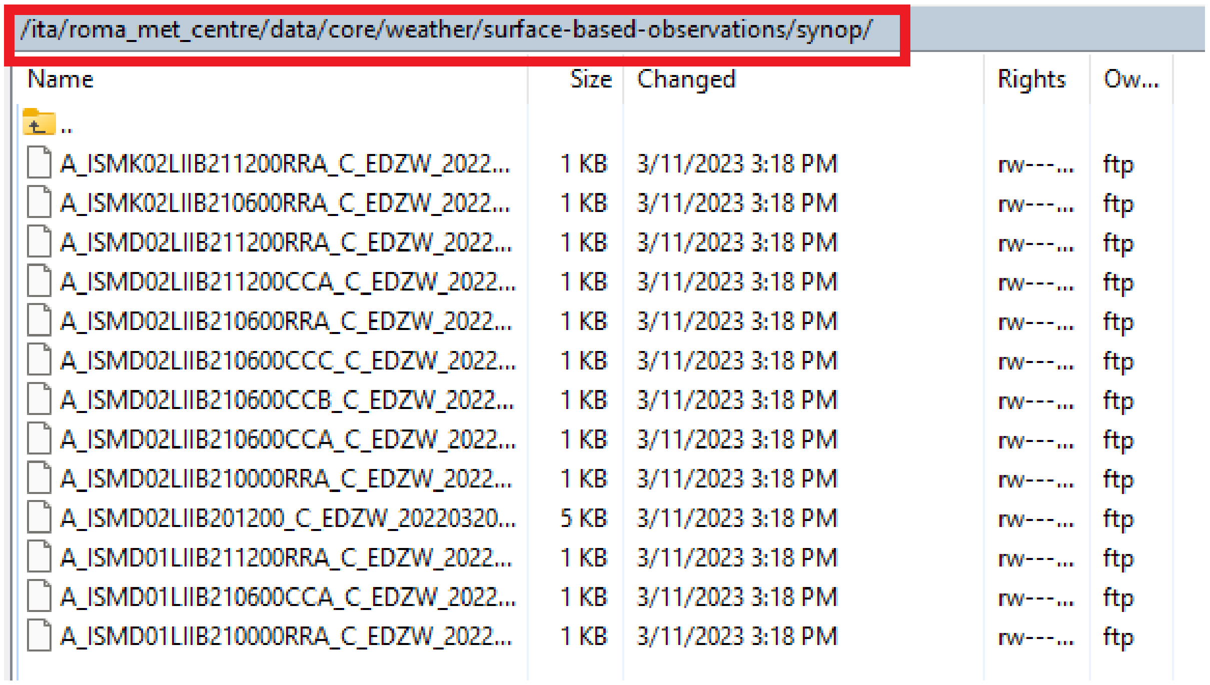
and ensure `MYHOSTNAME` is set to **your** hostname (fully qualified domain name).

Then start the `wis2box-ftp` service with the following command:

```
docker-compose -f docker-compose.wis2box-ftp.yml --env-file ftp.env up -d
```

When using the wis2box-ftp service to ingest data, please note that the topic is determined by the directory structure in which the data arrives.

For example to correctly ingest data on the topic `ita.roma_met_centre.data.core.weather.surface-based-observations.synop` you need to copy the data into the directory `/ita/roma_met_centre/data/core/weather/surface-based-observations/synop` on the FTP server:



/ita/roma_met_centre/data/core/weather/surface-based-observations/synop/

Name	Size	Changed	Rights	Ow...
..				
A_ISMK02LIIB211200RRA_C_EDZW_2022...	1 KB	3/11/2023 3:18 PM	rw---...	ftp
A_ISMK02LIIB210600RRA_C_EDZW_2022...	1 KB	3/11/2023 3:18 PM	rw---...	ftp
A_ISMD02LIIB211200RRA_C_EDZW_2022...	1 KB	3/11/2023 3:18 PM	rw---...	ftp
A_ISMD02LIIB211200CCA_C_EDZW_2022...	1 KB	3/11/2023 3:18 PM	rw---...	ftp
A_ISMD02LIIB210600RRA_C_EDZW_2022...	1 KB	3/11/2023 3:18 PM	rw---...	ftp
A_ISMD02LIIB210600CCC_C_EDZW_2022...	1 KB	3/11/2023 3:18 PM	rw---...	ftp
A_ISMD02LIIB210600CCB_C_EDZW_2022...	1 KB	3/11/2023 3:18 PM	rw---...	ftp
A_ISMD02LIIB210600CCA_C_EDZW_2022...	1 KB	3/11/2023 3:18 PM	rw---...	ftp
A_ISMD02LIIB210000RRA_C_EDZW_2022...	1 KB	3/11/2023 3:18 PM	rw---...	ftp
A_ISMD02LIIB201200_C_EDZW_20220320...	5 KB	3/11/2023 3:18 PM	rw---...	ftp
A_ISMD01LIIB211200RRA_C_EDZW_2022...	1 KB	3/11/2023 3:18 PM	rw---...	ftp
A_ISMD01LIIB210600CCA_C_EDZW_2022...	1 KB	3/11/2023 3:18 PM	rw---...	ftp
A_ISMD01LIIB210000RRA_C_EDZW_2022...	1 KB	3/11/2023 3:18 PM	rw---...	ftp

See the GitHub repository wis2box-ftp for more information on this service.

1.4.5 wis2box-data-subscriber

You can add an additional service on the host running your wis2box instance to allow data to be ingested by publishing an MQTT message to the wis2box broker.

This service subscribes to the topic `data-incoming/#` on the wis2box broker and parses the content of received messages and publishes the result in the `wis2box-incoming` bucket.

To start the `wis2box-data-subscriber`, add the following additional variables to `dev.env`:

```
COUNTRY_ID=zmb  # set country_id used in wis2-topic-hierarchy
CENTRE_ID=zmb_met_centre  # set centre_id for wis2-topic-hierarchy
```

These variables determine the destination path in the `wis2box-incoming` bucket:

`{COUNTRY_ID}/{CENTRE_ID}/data/core/weather/surface-based-observations/synop/`

You then you can activate the optional 'wis2box-data-subscriber' service as follows:

```
docker-compose -f docker-compose.data-subscriber.yml --env-file dev.env up -d
```

See the GitHub wis2box-data-subscriber repository for more information on this service.

1.4.6 Next steps

After you have successfully setup your data ingest process into the wis2box, you are ready to share your data with the global WIS2 network by enabling external access to your public services.

Next: *Public services setup*

1.5 Public services setup

To share your data with the WIS2 network, you need to expose some of your wis2box services to the Global Services:

- The Global Cache needs to be able to access to your HTTP endpoint to download data published by your wis2box instance
- The Global Broker needs to be able to subscribe to your MQTT endpoint to receive WIS2 notifications published by your wis2box instance

1.5.1 SSL

To enable HTTPS and MQTTS on your wis2box you can run wis2box with the option *–ssl*:

```
python3 wis2box-ctl.py --ssl start
```

When running wis2box with SSL, you have to set additional environment variables in your dev.env defining the location of your SSL certificate and private key:

```
WIS2BOX_SSL_CERT=/etc/letsencrypt/live/example.wis2box.io/fullchain.pem
WIS2BOX_SSL_KEY=/etc/letsencrypt/live/example.wis2box.io/privkey.pem
```

Please remember to update the `WIS2BOX_URL` and `` WIS2BOX_API_URL``environment variable after enabling SSL, ensuring your URL starts with ``https://``.

Please note that after changing the `WIS2BOX_URL` and `WIS2BOX_API_URL` environment variables, you will need to restart your wis2box:

```
python3 wis2box-ctl.py stop
python3 wis2box-ctl.py --ssl start
```

After restarting wis2box, repeat the commands for adding your dataset and publishing your metadata, to ensure the URLs are updated accordingly:

```
python3 wis2box-ctl.py login
wis2box data add-collection ${WIS2BOX_HOST_DATADIR}/surface-weather-observations.yml
wis2box metadata discovery publish ${WIS2BOX_HOST_DATADIR}/surface-weather-observations.
↪yml
```

1.5.2 Nginx (HTTP)

wis2box runs a local nginx container allowing access to the following HTTP based services on port 80:

Function	URL
API (wis2box-api)	*WIS2BOX_URL/oapi*
UI (wis2box-ui)	*WIS2BOX_URL/*
Storage (incoming data) (minio:wis2box-incoming)	*WIS2BOX_URL/wis2box-incoming*
Storage (public data) (minio:wis2box-public)	*WIS2BOX_URL/data*

You can edit `nginx/nginx.conf` to control which services are exposed through the nginx-container include in your stack.

You can edit `docker-compose.override.yml` to change the port on which the `web-proxy` service exposes HTTP on the localhost.

Note: The WIS2 notifications published by the wis2box includes the path `<wis2box-url>/data/`. This path has to be publicly accessible by the client receiving the WIS2 notification over MQTT, or the data referenced cannot be downloaded

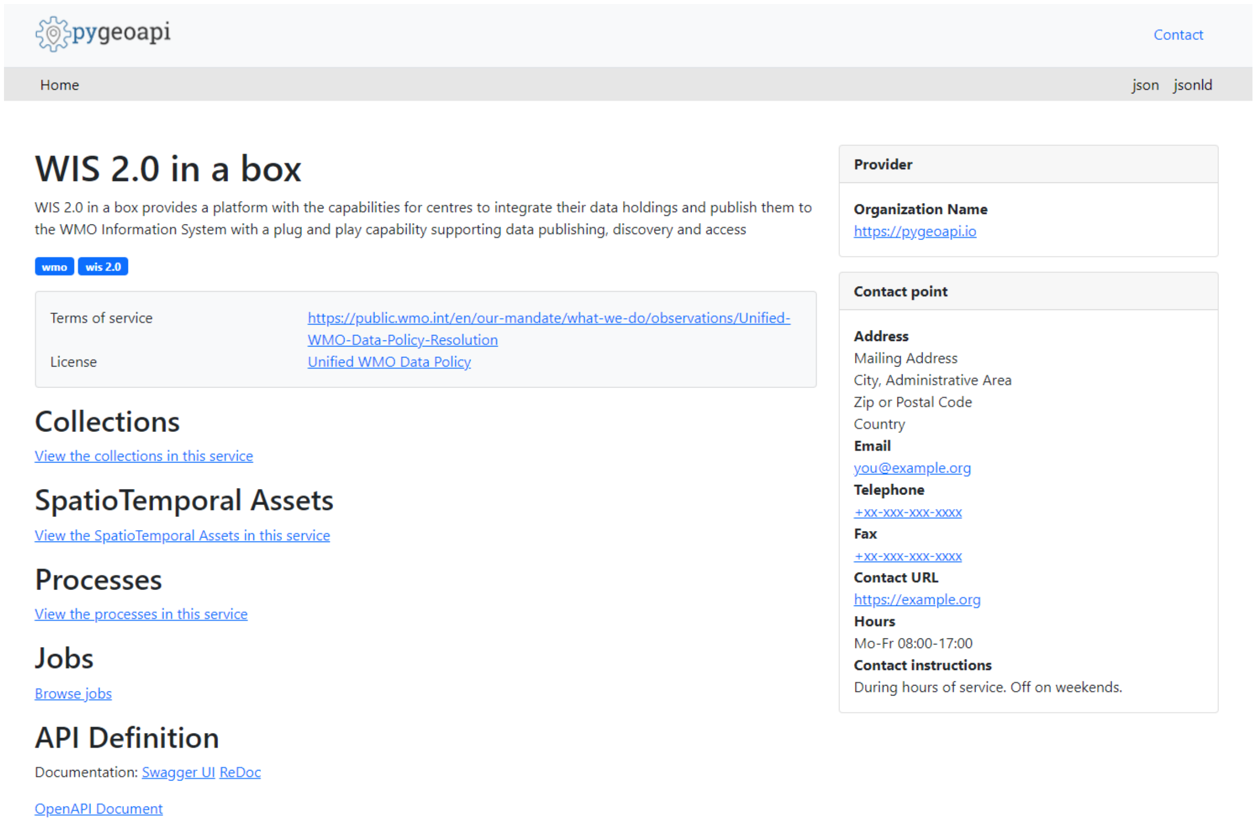
To share your data with the WIS2 network, ensure that `WIS2BOX_URL` as defined in `dev.env` points to the externally accessible URL for your HTTP services.

After updating `WIS2BOX_URL`, please stop and start your wis2box using `wis2box-ctl.py` and republish your data using the command `wis2box metadata discovery publish`.

Note: By default the environment variable `WIS2BOX_URL` resolves to `http://localhost`. This URL will define the `/data` URL used in the canonical link as part of your data in MQTT, as well as the dataset location in your discovery metadata.

wis2box API

The wis2box API uses pygeoapi, which implements the OGC API suite of standards, to provide programmatic access to the data collections hosted in your wis2box.



WIS 2.0 in a box

WIS 2.0 in a box provides a platform with the capabilities for centres to integrate their data holdings and publish them to the WMO Information System with a plug and play capability supporting data publishing, discovery and access

wmo wis 2.0

Terms of service	https://public.wmo.int/en/our-mandate/what-we-do/observations/Unified-WMO-Data-Policy-Resolution
License	Unified WMO Data Policy

Collections

View the collections in this service

SpatioTemporal Assets

View the SpatioTemporal Assets in this service

Processes

View the processes in this service

Jobs

Browse jobs

API Definition

Documentation: Swagger UI ReDoc

OpenAPI Document

Provider

Organization Name
https://pygeoapi.io

Contact point

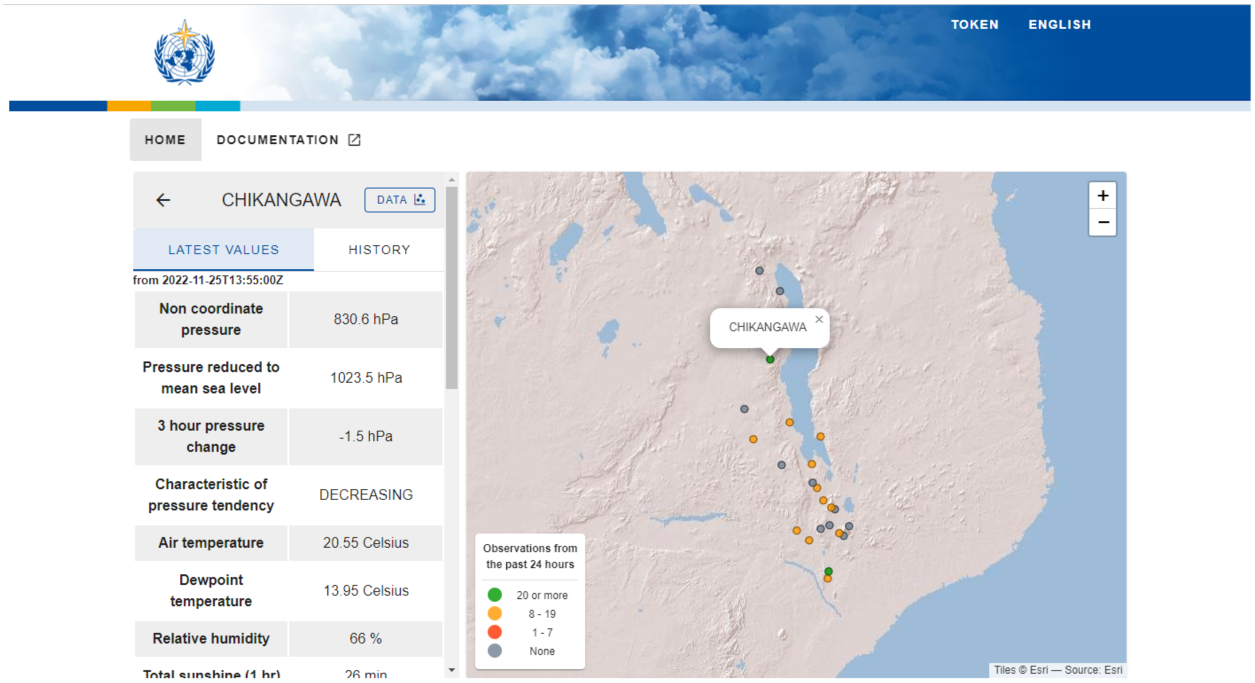
Address
Mailing Address
City, Administrative Area
Zip or Postal Code
Country
Email
you@example.org
Telephone
+xx-xxx-xxx-xxxx
Fax
+xx-xxx-xxx-xxxx
Contact URL
https://example.org
Hours
Mo-Fr 08:00-17:00
Contact instructions
During hours of service. Off on weekends.

Note: Currently, the default API backend in the wis2box stack uses Elasticsearch. A dedicated Docker-volume `es-data` is created on your host when you start your wis2box. As long as this volume is not deleted you can remove/update the containers in the wis2box stack without losing data.

wis2box user interface

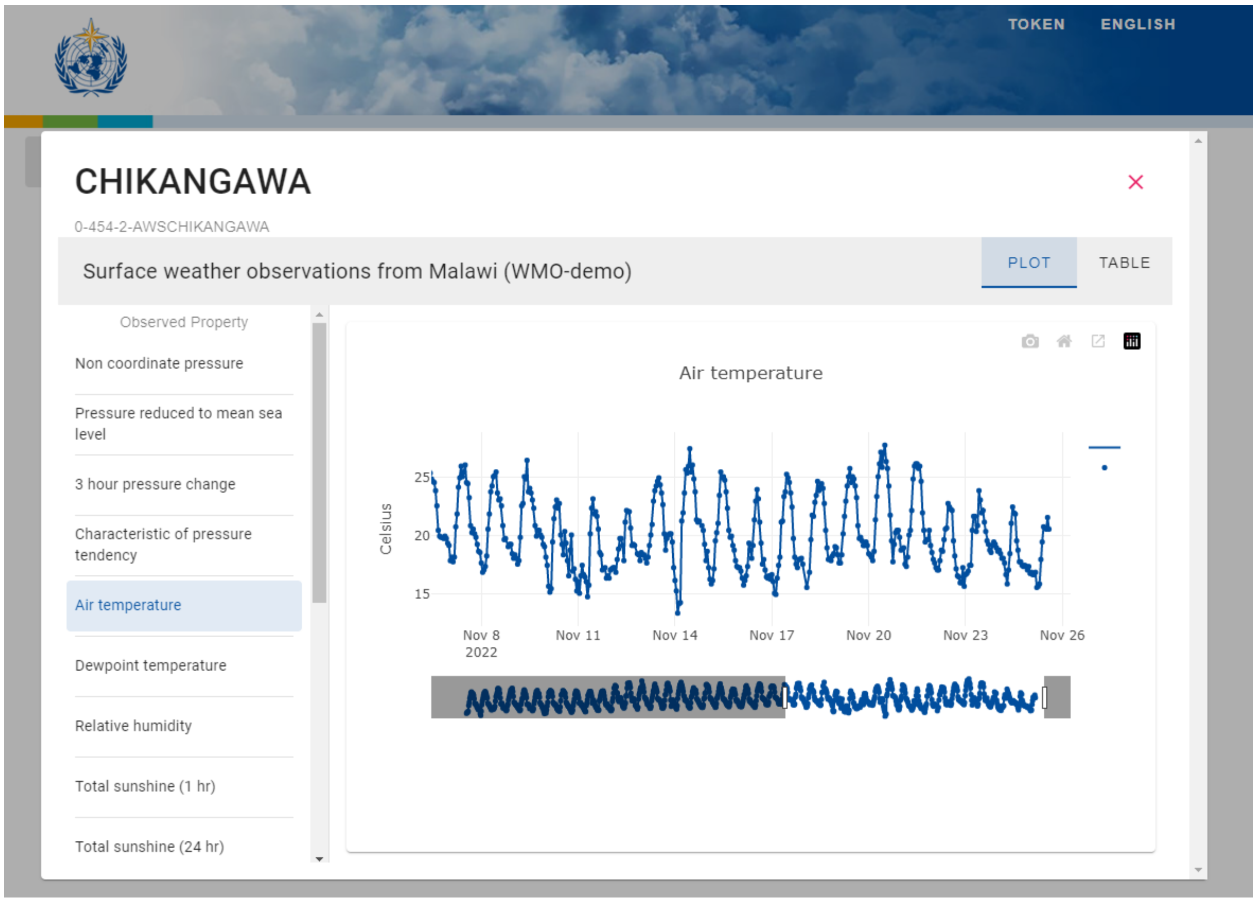
The wis2box user interface uses the wis2box API to visualize the data configured and shared through your wis2box.

The 'map' or 'explore' option of each dataset allows you to visualize Weather Observations per station.



from 2022-11-25T13:55:00Z

Non coordinate pressure	830.6 hPa
Pressure reduced to mean sea level	1023.5 hPa
3 hour pressure change	-1.5 hPa
Characteristic of pressure tendency	DECREASING
Air temperature	20.55 Celsius
Dewpoint temperature	13.95 Celsius
Relative humidity	66 %
Total sunshine (1 hr)	26 min



1.5.3 Mosquitto (MQTT)

By default, wis2box uses its own internal Mosquitto container to publish WIS2 notifications.

To allow the WIS2 Global Broker to subscribe to WIS2 notifications from your wis2box you have 2 options:

- enable access to internal broker running in the MQTT container on your wis2box host
- configure your wis2box to use an external broker

Internal broker

The internal MQTT broker uses the default username/password of `wis2box/wis2box`. Before opening the MQTT port for external access, it is recommended to set a unique password as follows:

```
WIS2BOX_BROKER_USERNAME=wis2box-utopia
WIS2BOX_BROKER_PASSWORD=myuniquepassword
WIS2BOX_BROKER_PUBLIC=mqtt://${WIS2BOX_BROKER_USERNAME}:${WIS2BOX_BROKER_PASSWORD}
↪@mosquitto:1883

# update minio settings after updating broker defaults
MINIO_NOTIFY_MQTT_USERNAME_WIS2BOX=${WIS2BOX_BROKER_USERNAME}
MINIO_NOTIFY_MQTT_PASSWORD_WIS2BOX=${WIS2BOX_BROKER_PASSWORD}
MINIO_NOTIFY_MQTT_BROKER_WIS2BOX=tcp://${WIS2BOX_BROKER_HOST}:${WIS2BOX_BROKER_PORT}
```

The internal MQTT broker is accessible on the host `mosquitto` within the Docker network used by wis2box.

By default port 1883 of the mosquitto container is mapped to port 1883 of the host running wis2box.

By exposing port 1883 on your host, the Global Broker will be able to subscribe directly to the internal MQTT broker on the wis2box.

Note: The `everyone` user is defined by default for public readonly access (`origin/#`) as per WIS2 Node requirements.

External broker

If you do not wish to expose the internal MQTT broker on your wis2box, you can configure your wis2box to publish WIS2 notifications to an external broker by setting the environment variable `WIS2BOX_BROKER_PUBLIC`.

```
# For example to use an external broker at host=example.org
WIS2BOX_BROKER_PUBLIC=mqtts://username:password@example.org:8883
```

Note: The `everyone` user is defined by default for public readonly access (`origin/#`) as per WIS2 Node requirements.

Sharing data with the WIS2 Global Broker

The official procedure for a WIS2 Node to share data with the WIS2 network is currently in development. Contact wis at wmo.int for more information on connectivity with the WIS2 network.

Next: *Downloading data from WIS2*

1.6 Downloading data from WIS2

1.6.1 Overview

This section provides guidance how to download data from WIS2 Global Services.

WIS2 Global Services include a Global Broker that provides users the ability to subscribe to data (via topics) and download to their local environment / workstation / decision support system from the WIS2 Global Cache.

1.6.2 The pywis-pubsub tool

wis2box enables subscribe and data download workflow the WIS2 network, by using the `wis2box-subscribe-download` container, inside of which runs the pywis-pubsub tool

`pywis-pubsub` is a Python package that provides publish, subscription and download capability of data from WIS2 Global Services.

Before starting the `wis2box-subscribe-download` container, the default configuration (provided in `wis2box-subscribe-download/local.yml`) must be updated, by defining the URL of the MQTT broker as well as the desired topic(s) to subscribe to.

In addition, the storage path should be updated to specify where downloaded data should be saved to.

```yaml
# fully qualified URL of broker
broker: mqtts://username:password@host:port

# whether to run checksum verification when downloading data (default true)
verify_data: true

# whether to validate broker messages (default true)
validate_message: true

# list of 1..n topics to subscribe to
topics:
    - 'cache/a/wis2/topic1/#'
    - 'cache/a/wis2/topic2/#'

# storage: filesystem
storage:
    type: fs
    options:
        path: /tmp/foo/bar
```

To start a continuous subscribe and download process, run the `wis2box-subscribe-download` container as follows (`-d` for detached mode, `--build` to ensure changes in `local.yml` are built into the container):

```
docker-compose -f docker.subscribe-download.yml up -d --build
```

To stop the subscribe and download process, run the following command:

```
docker-compose -f docker.subscribe-download.yml down
```

1.6.3 Running pywis-pubsub interactively

pywis-pubsub can also be run interactively from inside the wis2box main container as follows:

```
# login to wis2box main container
python3 wis2box-ctl.py login

# edit a local configuration by using wis2box-subscribe-download/local.yml as a template
vi /data/wis2box/local.yml

# connect, and simply display data notifications
pywis-pubsub subscribe --config local.yml

# connect, and additionally download messages
pywis-pubsub subscribe --config local.yml --download

# connect, and filter messages by bounding box geometry
pywis-pubsub subscribe --config local.yml --bbox=-142,42,-52,84
```

CHAPTER

TWO

REFERENCE GUIDE

The reference documentation is more complete and programmatic in nature. It contains a comprehensive set of information on wis2box for easy reference.

2.1 WIS2

The WMO Information System is a coordinated global infrastructure responsible for telecommunications and data management functions and is owned and operated by WMO Members.

WIS provides an integrated approach suitable for all WMO Programmes to meet the requirements for routine collection and automated dissemination of observed data and products, as well as data discovery, access, and retrieval services for weather, climate, water, and related data produced by centres and Member countries in the framework of any WMO Programme. It is capable of exchanging large data volumes, such as new ground and satellite-based systems, finer resolutions in numerical weather prediction, and hydrological models and their applications. These data and products must be available to National Hydrological and Meteorological Services (NHMS), but also national disaster authorities for more timely alerts where and when needed.

WIS is a vital data communications backbone for integrating the diverse real-time and non-real-time high priority data sets, regardless of location.

Further documentation on WIS2 can be found at the following links:

- WIS Overview

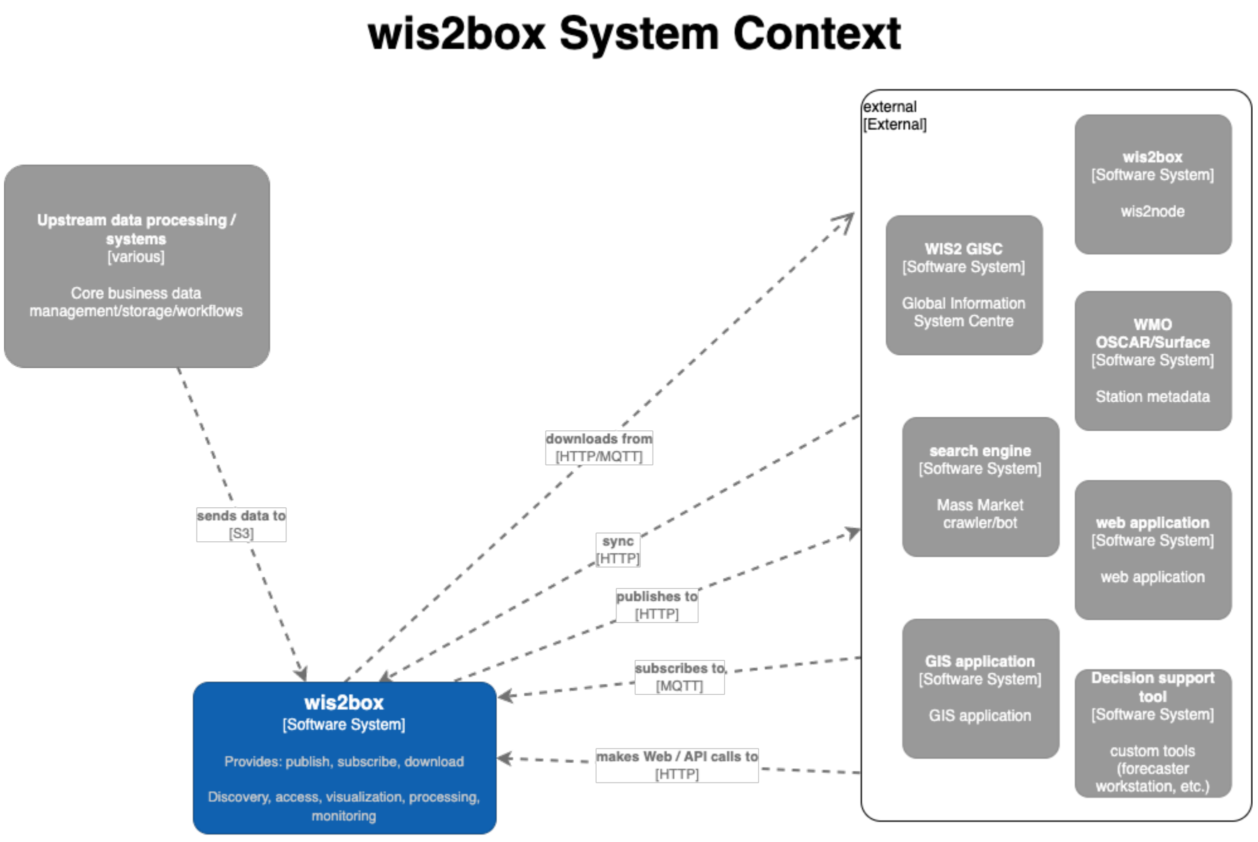
2.2 How wis2box works

wis2box is implemented in the spirit of the Twelve-Factor App methodology.

wis2box is a Docker and Python-based platform with the capabilities for centres to publish their data holdings to the WMO Information System with a plug and play capability supporting data publishing, discovery and access.

2.2.1 High level system context

The following diagram provides a high level overview of the main functions of wis2box:



Core wis2box functionality includes the ability to:

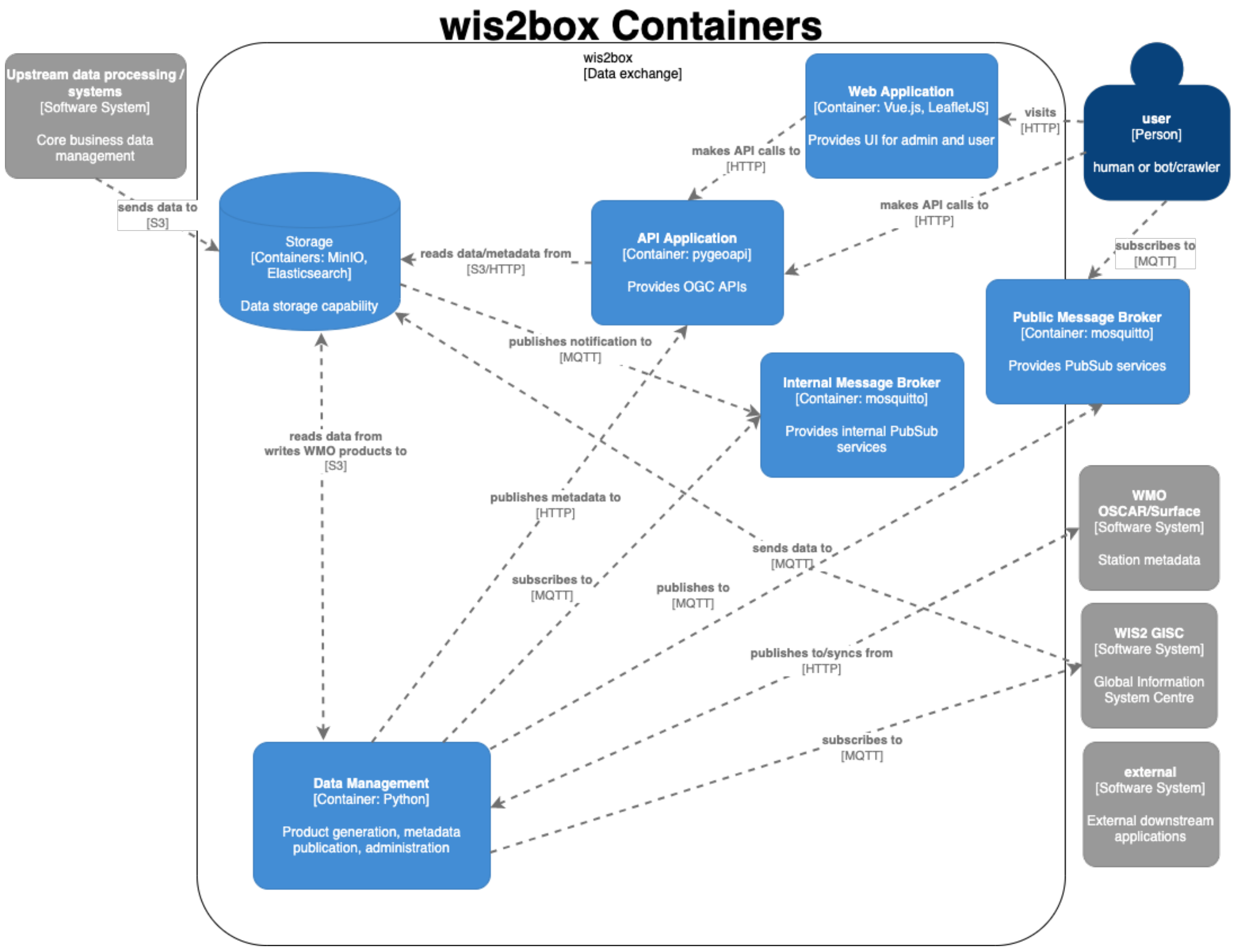
- integrate your existing data processing pipeline
- process and transform your weather/climate/water data into official WMO data formats
- create and publish discovery metadata of your datasets
- provide your data via OGC and Pub/Sub standards mechanisms to your data, enabling easy access for web applications, desktop GIS tools, mobile applications
- connect your wis2box to the WIS2 network
- make your data and services available to market search engines
- subscribe to and download weather/climate/water data from the WIS2 network

2.2.2 Docker Compose

wis2box is built as Docker Compose application, allowing for easy install and container management.

2.2.3 Container workflow

Let's dive a little deeper. The following diagram provides a view of all wis2box containers:



Container functionality can be described as follows:

- **Storage**: core data and metadata persistence, the initial data entry point of wis2box. Data pipelines and workflow are triggered from here
- **Internal Message Broker**: internal message bus
- **Public Message Broker**: public facing broker. Provides data and metadata notifications
- **Data Management**: the epicentre of wis2box. Provides core wis2box administration and data/workflow/publishing utilities
- **API Application**: OGC APIs providing geospatial web services
- **Web Application**: user interface

2.2.4 Technology

wis2box is built on free and open source (FOSS) technology.

Container	Function	Technology	Standards
Storage	data and metadata storage	MinIO Elasticsearch	S3
Internal Message Broker	Pub/Sub	mosquitto	MQTT
Public Message Broker	Pub/Sub	mosquitto	MQTT
Data Management	data processing and publishing	ecCodes csv2bufr bufr2geojson synop2bufr OWSLib pygeometa	WCMP (WMO Core Metadata Profile) WMDR (WIGOS Metadata Record)
API Application	data discovery and access	pygeoapi	OGC API
Web Application	data discovery and visualization	Vue.js Leaflet	OGC API

2.3 Configuration

Once you have installed wis2box, it is time to setup the configuration. wis2box setup is based on a simple configuration that can be adjusted depending the user's needs and deployment environment.

2.3.1 Environment variables

wis2box configuration is driven primarily by a small set of environment variables. The runtime configuration is defined in the Env format in a plain text file named `dev.env` and `default.env`.

Any values set in `dev.env` override the default environment variables in `default.env`. For further / specialized configuration, see the sections below.

`WIS2BOX_HOST_DATADIR`

The minimum required setting in `dev.env` is the `WIS2BOX_HOST_DATADIR` environment variable. Setting this value is **required** to map the wis2box data directory from the host system to the containers.

It is recommended to set this value to an absolute path on your system.

2.3.2 Sections

Note: A reference configuration can always be found in the wis2box GitHub repository. The *Quickstart with test data* uses a variant of `wis2box.env` with mappings to the test data, as an example. For complex installations, it is recommended to start configuring wis2box by copying the example `wis2box.env` file and modifying accordingly.

wis2box environment variables can be categorized via the following core sections:

- **Storage**: MinIO configuration
- **API**: API configuration for provisioning the OGC API capabilities

- **Logging**: logging configuaration for wis2box
- **Pub/Sub**: Pub/Sub options
- **Other**: other miscellaneous options

Note: Configuration directives and reference are described below via annotated examples. Changes in configuration require a restart of wis2box to take effect. See the *Administration* section for information on managing wis2box.

Storage

wis2box currently supports S3 compatible storage (e.g. MinIO, Amazon S3). Additional storage types are planned for future releases.

The following environment variables can be used to configure *WIS2BOX_STORAGE*.

Note: When using wis2box in production and using the default MinIO-container, please specify a unique `WIS2BOX_STORAGE_PASSWORD`

```
WIS2BOX_STORAGE_TYPE=S3
WIS2BOX_STORAGE_SOURCE=http://minio:9000
WIS2BOX_STORAGE_USERNAME=minio  # username for the storage-layer
WIS2BOX_STORAGE_PASSWORD=minio123  # password for the storage-layer
WIS2BOX_STORAGE_INCOMING=wis2box-incoming  # name of the storage-bucket/folder for␣
↪incoming files
WIS2BOX_STORAGE_PUBLIC=wis2box-public  # name of the storage-bucket/folder for public␣
↪files
WIS2BOX_STORAGE_ARCHIVE=wis2box-archive  # name of the storage-bucket/folder for␣
↪archived data
WIS2BOX_STORAGE_DATA_RETENTION_DAYS=7  # number of days to keep files in incoming and␣
↪public
```

MinIO

wis2box uses MinIO as the default S3 storage capability.

When overriding the default storage environment variables, please redefine the `MINIO*` environment variables to match your configuration.

```
MINIO_ROOT_USER=${WIS2BOX_STORAGE_USERNAME}
MINIO_ROOT_PASSWORD=${WIS2BOX_STORAGE_PASSWORD}
MINIO_NOTIFY_MQTT_USERNAME_WIS2BOX=${WIS2BOX_BROKER_USERNAME}
MINIO_NOTIFY_MQTT_PASSWORD_WIS2BOX=${WIS2BOX_BROKER_PASSWORD}
MINIO_NOTIFY_MQTT_BROKER_WIS2BOX=tcp://${WIS2BOX_BROKER_HOST}:${WIS2BOX_BROKER_PORT}
```

API

API configurations drive control of the OGC API setup.

```
WIS2BOX_API_TYPE=pygeoapi  # server tpye
WIS2BOX_API_URL=http://localhost/pygeoapi  # public landing page endpoint
WIS2BOX_API_BACKEND_TYPE=Elasticsearch  # backend provider type
WIS2BOX_API_BACKEND_URL=http://elasticsearch:9200  # internal backend connection URL
WIS2BOX_DOCKER_API_URL=http://wis2box-api:80/oapi  # container name of API container␣
↪(for internal communications/workflow)
```

Logging

The logging directives control logging level/severity and output.

```
WIS2BOX_LOGGING_LOGLEVEL=ERROR  # the logging level (see https://docs.python.org/3/
↪library/logging.html#logging-levels)
WIS2BOX_LOGGING_LOGFILE=stdout  # the full file path to the logfile or ``stdout`` to␣
↪display on console
```

Pub/Sub

Pub/Sub configuration provides connectivity information for the Pub/Sub broker.

```
WIS2BOX_BROKER_HOST=mosquitto  # the hostname of the internal broker
WIS2BOX_BROKER_PORT=1883  # the port of the internal broker
WIS2BOX_BROKER_USERNAME=wis2box  # the username of the internal broker
WIS2BOX_BROKER_PASSWORD=wis2box  # the password of the internal broker
WIS2BOX_BROKER_PUBLIC=mqtt://foo:bar@localhost:1883  # RFC 1738 URL of public broker␣
↪endpoint
WIS2BOX_BROKER_QUEUE_MAX=1000  # maximum number of messages to hold in the queue per␣
↪client
```

Note: `WIS2BOX_BROKER_QUEUE_MAX` should be configured according to the setup of wis2box, relative to the number of expected observations per day. See *Getting started* for more information on system requirements.

Web application

Web application configuration provides the ability to customize web components.

```
WIS2BOX_BASEMAP_URL="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"  # URL of map␣
↪tile server to use
WIS2BOX_BASEMAP_ATTRIBUTION="<a href="https://osm.org/copyright">OpenStreetMap</a>␣
↪contributors"  # attribution of map tile server
```

Other

Additional directives provide various configurationscontrol of configuration options for the deployment of wis2box.

```
WIS2BOX_URL=http://localhost/  # public wis2box url
WIS2BOX_AUTH_STORE=http://wis2box-auth # wis2box auth service location
```

Note: To access internal containers, URL configurations should point to the named containers as specified in `docker-compose.yml`.

A full configuration example can be found below:

```
# please define a data-directory on your host machine
# this will map to /data/wis2box on the wis2box-container
WIS2BOX_HOST_DATADIR=/home/example/wis2box-data

# replace localhost with the IP or public address for your instance
# replace http with https when using SSL certificates
WIS2BOX_URL=http://localhost
WIS2BOX_API_URL=http://localhost/oapi

# logging and data retention
WIS2BOX_LOGGING_LOGLEVEL=INFO
WIS2BOX_DATA_RETENTION_DAYS=30

# update broker default credentials
WIS2BOX_BROKER_USERNAME=wis2box
WIS2BOX_BROKER_PASSWORD=XXXXXXXXX

WIS2BOX_BROKER_PUBLIC=mqtt://${WIS2BOX_BROKER_USERNAME}:${WIS2BOX_BROKER_PASSWORD}
↪@mosquitto:1883

# update storage default credentials
# username should be 3 or more characters
WIS2BOX_STORAGE_USERNAME=wis2box
# password should be 8 or more characters
WIS2BOX_STORAGE_PASSWORD=XXXXXXXXX

# update minio settings after updating storage and broker defaults
MINIO_ROOT_USER=${WIS2BOX_STORAGE_USERNAME}
MINIO_ROOT_PASSWORD=${WIS2BOX_STORAGE_PASSWORD}
MINIO_NOTIFY_MQTT_USERNAME_WIS2BOX=${WIS2BOX_BROKER_USERNAME}
MINIO_NOTIFY_MQTT_PASSWORD_WIS2BOX=${WIS2BOX_BROKER_PASSWORD}
```

```
# data paths and retention
WIS2BOX_DATADIR=/data/wis2box

# API
WIS2BOX_API_TYPE=pygeoapi
WIS2BOX_API_URL=http://localhost/oapi
WIS2BOX_API_BACKEND_TYPE=Elasticsearch
WIS2BOX_API_BACKEND_URL=http://elasticsearch:9200
```

(continues on next page)

(continued from previous page)

```
WIS2BOX_DOCKER_API_URL=http://wis2box-api:80/oapi

# logging
WIS2BOX_LOGGING_LOGLEVEL=ERROR
WIS2BOX_LOGGING_LOGFILE=stdout

# Pub/Sub
WIS2BOX_BROKER_USERNAME=wis2box
WIS2BOX_BROKER_PASSWORD=wis2box
WIS2BOX_BROKER_HOST=mosquitto
WIS2BOX_BROKER_PORT=1883
WIS2BOX_BROKER_QUEUE_MAX=1000

WIS2BOX_BROKER_PUBLIC=mqtt://wis2box:wis2box@mosquitto:1883

# Web application
WIS2BOX_BASEMAP_URL=https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png
WIS2BOX_BASEMAP_ATTRIBUTION=<a href="https://osm.org/copyright">OpenStreetMap</a>␣
↪contributors

# Admin UI
WIS2BOX_UI_ADMIN_BASEURL=/admin

# other
WIS2BOX_URL=http://localhost

# access control
WIS2BOX_AUTH_URL=http://wis2box-auth

# storage
WIS2BOX_STORAGE_TYPE=S3
WIS2BOX_STORAGE_SOURCE=http://minio:9000
# storage username should be 3 or more characters
WIS2BOX_STORAGE_USERNAME=minio
# storage password should be 8 or more characters
WIS2BOX_STORAGE_PASSWORD=minio123
WIS2BOX_STORAGE_INCOMING=wis2box-incoming
WIS2BOX_STORAGE_PUBLIC=wis2box-public
WIS2BOX_STORAGE_ARCHIVE=wis2box-archive
WIS2BOX_STORAGE_DATA_RETENTION_DAYS=7

# you should be okay from here

# MinIO
MINIO_ROOT_USER=${WIS2BOX_STORAGE_USERNAME}
MINIO_ROOT_PASSWORD=${WIS2BOX_STORAGE_PASSWORD}
MINIO_PROMETHEUS_AUTH_TYPE=public
MINIO_NOTIFY_MQTT_ENABLE_WIS2BOX=on
MINIO_NOTIFY_MQTT_USERNAME_WIS2BOX=${WIS2BOX_BROKER_USERNAME}
MINIO_NOTIFY_MQTT_PASSWORD_WIS2BOX=${WIS2BOX_BROKER_PASSWORD}
MINIO_NOTIFY_MQTT_BROKER_WIS2BOX=tcp://${WIS2BOX_BROKER_HOST}:${WIS2BOX_BROKER_PORT}
MINIO_NOTIFY_MQTT_TOPIC_WIS2BOX=wis2box-storage/minio
```

(continues on next page)

(continued from previous page)

```
MINIO_NOTIFY_MQTT_TOPIC_WIS2BOX=wis2box-storage/minio
MINIO_NOTIFY_MQTT_QOS_WIS2BOX=1
```

2.3.3 Docker Compose

The Docker Compose setup is driven from the resulting `dev.env` file created. For advanced cases and/or power users, updates can also be made to `docker-compose.yml` or `docker-compose.override.yml` (for changes to ports).

2.4 Administration

wis2box is designed to be built as a network of virtual machines within a virtual network. Once this is built, users login into the main wis2box machine to setup their workflow and configurations for data processing and publishing.

The `wis2box-ctl.py` utility provides a number of tools for managing the wis2box-management containers.

The following steps provide an example of container management workflow.

```
# build all images
python3 wis2box-ctl.py build

# start system
python3 wis2box-ctl.py start

# stop system
python3 wis2box-ctl.py stop

# view status of all deployed containers
python3 wis2box-ctl.py status
```

Note: Run `python3 wis2box-ctl.py --help` for all usage options.

With wis2box now installed and started, it's time to start up the box and login to the wis2box-management container:

```
python3 wis2box-ctl.py start
python3 wis2box-ctl.py login
```

Now that you are logged into the wis2box-management container, it's now time to manage station metadata, discovery metadata and data processing pipelines.

2.4.1 Public environment variables

The following environment variables are used for public services:

- `WIS2BOX_API_URL`: API application
- `WIS2BOX_BROKER_PUBLIC`: MQTT broker
- `WIS2BOX_URL`: Web application, including access to data download/object storage

2.4.2 Default service ports

A default wis2box installation utilizes the following ports for public services:

Public services

- `80`: Web application, API application, storage
- `1883`: Message broker via MQTT
- `8884`: Message broker via MQTT/WebSockets

Internal services

- `1883`: Message broker
- `9200`: Elasticsearch
- `9000`: MinIO
- `9001`: MinIO admin UI

Changing default ports

The `docker-compose.override.yml` file provides definitions on utilized ports. To change default ports, edit `default.env` before stopping and starting wis2box for changes to take effect.

2.4.3 MQTT Quality of Service (QoS)

The quality of service level of all wis2box powered brokers is always 1 by default.

2.5 Quickstart with test data

The 'quickstart' deploys wis2box with test data and provides a vital reference for wis2box developers to validate their contributions do not break the wis2box core functionality. It is the minimal runtime configuration profile as used in wis2box GitHub CI/CD: GitHub Actions.

Note: wis2box web components are run on port 80 by default. When using wis2box from source, the default port for web components is 8999, to be used for development.

To download wis2box from source:

```
git clone https://github.com/wmo-im/wis2box.git
```

The test enviroment file is provided in `tests/test.env`.

To run with the 'quickstart' configuration, copy this file to `dev.env` in your working directory:

```
cp tests/test.env dev.env
```

Build and update wis2box:

```
python3 wis2box-ctl.py build
python3 wis2box-ctl.py update
```

Start wis2box and login to the wis2box-management container:

```
python3 wis2box-ctl.py start
python3 wis2box-ctl.py login
```

Once logged in, verify the enviroment:

```
wis2box environment show
```

Publish test discovery metadata:

```
wis2box metadata discovery publish $WIS2BOX_DATADIR/metadata/discovery/mwi-surface-
↪weather-observations.yml
wis2box metadata discovery publish $WIS2BOX_DATADIR/metadata/discovery/ita-surface-
↪weather-observations.yml
wis2box metadata discovery publish $WIS2BOX_DATADIR/metadata/discovery/dza-surface-
↪weather-observations.yml
wis2box metadata discovery publish $WIS2BOX_DATADIR/metadata/discovery/rou-synoptic-
↪weather-observations.yml
```

Setup observation collections from discovery metadata:

```
wis2box data add-collection $WIS2BOX_DATADIR/metadata/discovery/mwi-surface-weather-
↪observations.yml
wis2box data add-collection $WIS2BOX_DATADIR/metadata/discovery/ita-surface-weather-
↪observations.yml
wis2box data add-collection $WIS2BOX_DATADIR/metadata/discovery/dza-surface-weather-
↪observations.yml
wis2box data add-collection $WIS2BOX_DATADIR/metadata/discovery/rou-synoptic-weather-
↪observations.yml
```

Ingest data using the data ingest command to push data to the `wis2box-incoming` bucket:

```
wis2box data ingest --topic-hierarchy mwi.mwi_met_centre.data.core.weather.surface-based-
↪observations.synop --path $WIS2BOX_DATADIR/observations/malawi
wis2box data ingest --topic-hierarchy ita.roma_met_centre.data.core.weather.surface-
↪based-observations.synop --path $WIS2BOX_DATADIR/observations/italy
wis2box data ingest --topic-hierarchy dza.alger_met_centre.data.core.weather.surface-
↪based-observations.synop --path $WIS2BOX_DATADIR/observations/algeria
wis2box data ingest --topic-hierarchy rou.rnimh.data.core.weather.surface-based-
↪observations.synop --path $WIS2BOX_DATADIR/observations/romania
```

Publish stations:

```
wis2box metadata station publish-collection
```

Logout of wis2box-management container:

```
exit
```

From here, you can run `python3 wis2box-ctl.py status` to confirm that containers are running properly.

To explore your wis2box installation and services, visit http://localhost in your web browser.

2.6 Running

wis2box workflows can be categorized as design time (interactive) or runtime (automated).

2.6.1 Design time

- environment creation
- topic hierarchy registration
- station metadata caching
- station metadata API publishing
- discovery metadata API publishing

2.6.2 Runtime

- automated data processing and publishing

2.6.3 topics

Concepts

Let's clarify a few concepts as part working with wis2box:

- **topic hierarchy**: structure defined by WMO to categorize and classify data, allowing for easy and efficient search and identification
- **discovery metadata**: description of a dataset to be included in the WIS2 Global Discovery Catalogue
- **catalogue**: a collection of discovery metadata records
- **station metadata**: description of the properties of an observing station, which provides observations and measurements
- **data mappings**: the wis2box mechanism to define and associate a topic hierarchy to a processing pipeline

Topic hierarchy

Note: The WIS2 topic hierarchy is currently in development. wis2box implementation of the topic hierarchies will change, based on ratifications/updates of the topic hierarchies in WMO technical regulations and publications.

wis2box implements the WIS2 topic hierarchies, which are designed to efficiently categorize and classify data.

To publish data and metadata to WIS2, the topic hierarchy requires a valid centre identifier (`centre-id`), which is specified by the member endorsed by the permananent representative (PR) of the country.

Environment

wis2box initializes the environment when starting, before data processing or publishing. To view the environment, run the following command:

```
wis2box environment show
```

For the purposes of documentation, the value `WIS2BOX_DATADIR` represents the base directory for all data managed in wis2box.

The default enviroment variables are below.

```
# data paths and retention
WIS2BOX_DATADIR=/data/wis2box

# API
WIS2BOX_API_TYPE=pygeoapi
WIS2BOX_API_URL=http://localhost/oapi
WIS2BOX_API_BACKEND_TYPE=Elasticsearch
WIS2BOX_API_BACKEND_URL=http://elasticsearch:9200
WIS2BOX_DOCKER_API_URL=http://wis2box-api:80/oapi

# logging
WIS2BOX_LOGGING_LOGLEVEL=ERROR
WIS2BOX_LOGGING_LOGFILE=stdout

# Pub/Sub
WIS2BOX_BROKER_USERNAME=wis2box
WIS2BOX_BROKER_PASSWORD=wis2box
WIS2BOX_BROKER_HOST=mosquitto
WIS2BOX_BROKER_PORT=1883
WIS2BOX_BROKER_QUEUE_MAX=1000

WIS2BOX_BROKER_PUBLIC=mqtt://wis2box:wis2box@mosquitto:1883

# Web application
WIS2BOX_BASEMAP_URL=https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png
WIS2BOX_BASEMAP_ATTRIBUTION=<a href="https://osm.org/copyright">OpenStreetMap</a>␣
↪contributors

# Admin UI
WIS2BOX_UI_ADMIN_BASEURL=/admin

# other
WIS2BOX_URL=http://localhost

# access control
WIS2BOX_AUTH_URL=http://wis2box-auth

# storage
WIS2BOX_STORAGE_TYPE=S3
WIS2BOX_STORAGE_SOURCE=http://minio:9000
# storage username should be 3 or more characters
WIS2BOX_STORAGE_USERNAME=minio
```

(continues on next page)

(continued from previous page)

```
# storage password should be 8 or more characters
WIS2BOX_STORAGE_PASSWORD=minio123
WIS2BOX_STORAGE_INCOMING=wis2box-incoming
WIS2BOX_STORAGE_PUBLIC=wis2box-public
WIS2BOX_STORAGE_ARCHIVE=wis2box-archive
WIS2BOX_STORAGE_DATA_RETENTION_DAYS=7

# you should be okay from here

# MinIO
MINIO_ROOT_USER=${WIS2BOX_STORAGE_USERNAME}
MINIO_ROOT_PASSWORD=${WIS2BOX_STORAGE_PASSWORD}
MINIO_PROMETHEUS_AUTH_TYPE=public
MINIO_NOTIFY_MQTT_ENABLE_WIS2BOX=on
MINIO_NOTIFY_MQTT_USERNAME_WIS2BOX=${WIS2BOX_BROKER_USERNAME}
MINIO_NOTIFY_MQTT_PASSWORD_WIS2BOX=${WIS2BOX_BROKER_PASSWORD}
MINIO_NOTIFY_MQTT_BROKER_WIS2BOX=tcp://${WIS2BOX_BROKER_HOST}:${WIS2BOX_BROKER_PORT}
MINIO_NOTIFY_MQTT_TOPIC_WIS2BOX=wis2box-storage/minio
MINIO_NOTIFY_MQTT_TOPIC_WIS2BOX=wis2box-storage/minio
MINIO_NOTIFY_MQTT_QOS_WIS2BOX=1
```

Data mappings

Once a topic hierarchy is defined, it needs to be included in the wis2box data mappings configuration. wis2box provides a default data mapping (in YAML format):

```
data:
    rou.rnimh.data.core.weather.surface-based-observations.synop:
        plugins:
            txt:
                - plugin: wis2box.data.synop2bufr.ObservationDataSYNOP2BUFR
                  notify: true
                  file-pattern: '^A_SMR.*EDZW_(\d{4})(\d{2}).*.txt$'
            bufr4:
                - plugin: wis2box.data.bufr2geojson.ObservationDataBUFR2GeoJSON
                  file-pattern: '^A_SMR.*EDZW_(\d{4})(\d{2}).*.bufr4$'
    mwi.mwi_met_centre.data.core.weather.surface-based-observations.synop:
        plugins:
            csv:
                - plugin: wis2box.data.csv2bufr.ObservationDataCSV2BUFR
                  template: /data/wis2box/synop_bufr.json
                  notify: true
                  file-pattern: '^WIGOS_(\d-\d+-\d+-\w+)_.*\.csv$'
            bufr4:
                - plugin: wis2box.data.bufr2geojson.ObservationDataBUFR2GeoJSON
                  file-pattern: '^WIGOS_(\d-\d+-\d+-\w+)_.*\.bufr4$'
    ita.roma_met_centre.data.core.weather.surface-based-observations.synop:
        plugins:
            bin:
                - plugin: wis2box.data.bufr4.ObservationDataBUFR
                  notify: true
```

(continues on next page)

(continued from previous page)

```
                file-pattern: '^.*\.bin$'
          bufr4:
              - plugin: wis2box.data.bufr2geojson.ObservationDataBUFR2GeoJSON
                file-pattern: '^WIGOS_(\d-\d+-\d+-\w+)_.*\.bufr4$'
  dza.alger_met_centre.data.core.weather.surface-based-observations.synop:
      plugins:
          bufr4:
              - plugin: wis2box.data.bufr4.ObservationDataBUFR
                notify: true
                buckets:
                  - ${WIS2BOX_STORAGE_INCOMING}
                file-pattern: '^.*\.bufr4$'
              - plugin: wis2box.data.bufr2geojson.ObservationDataBUFR2GeoJSON
                buckets:
                  - ${WIS2BOX_STORAGE_PUBLIC}
                file-pattern: '^WIGOS_(\d-\d+-\d+-\w+)_.*\.bufr4$'
```

The data mappings are indicated by the `data` keyword, with each topic having a separate entry specifying:

- `plugins`: all plugin objects associated with the topic, by file type/extension

Each plugin is based on the file extension to be detected and processed, with the following configuration:

- `plugin`: the codepath of the plugin
- `notify`: whether the plugin should publish a data notification
- `template`: additional argument allowing a mapping template name to be passed to the plugin. Note that if the path is relative, the plugin must be able to locate the template accordingly
- `file-pattern`: additional argument allowing a file pattern to be passed to the plugin
- `buckets`: the name(s) of the storage bucket(s) that data should be saved to (See *Configuration* for more information on buckets)

The default data mapping can be overriden by user-defined data mappings with the following steps:

- create a YAML file similar to the above to include your topic hierarchy
- place the file in the `WIS2BOX_DATADIR` directory
- restart wis2box

See *Extending wis2box* for more information on adding your own data processing pipeline.

Station metadata

wis2box is designed to support data ingest and processing of any kind. For observations, processing workflow typically requires station metadata to be present at runtime.

To manage your stations of interest, create a CSV file named `metadata/station/station_list.csv` in `$WIS2BOX_HOST_DATADIR`, specifying one line per station as follows:

```
station_name,wigos_station_identifier,traditional_station_identifier,facility_type,
↪latitude,longitude,elevation,territory_name,wmo_region
BALAKA,0-454-2-AWSBALAKA,AWSBALAKA,Land (fixed),-14.983333,34.966666,618,Malawi,I
BENI-ABBES,0-20000-0-60602,60602,Land (fixed),30.12846,-2.14953,510,Algeria,I
```

(continues on next page)

(continued from previous page)

```
IN-GUEZZAM,0-20000-0-60690,60690,Land (fixed),19.56388,5.74887,399,Algeria,I
MALOMO,0-454-2-AWSMALOMO,AWSMALOMO,Land (fixed),-13.14202,33.83727,1088,Malawi,I
```

This CSV file is used by wis2box data processing pipelines and is required before starting automated processing.

Note: run the command `wis2box metadata station publish-collection` to publish your stations as a collection to the wis2box API

See also:

API publishing

Summary

At this point, you have cached the required station metadata for your given dataset(s).

Discovery metadata

Discovery metadata describes a given dataset or collection. Data being published through a wis2box requires discovery metadata (describing it) to be created, maintained and published to the wis2box catalogue API.

wis2box supports managing discovery metadata using the WMO Core Metadata Profile (WCMP2) standard.

Note: WCMP2 is currently in development as part of WMO activities.

Creating a discovery metadata record in wis2box is as easy as completing a YAML configuration file. wis2box leverages the pygeometa project's metadata control file (MCF) format. Below is an example MCF file.

```
wis2box:
    retention: P30D
    topic_hierarchy: mwi.mwi_met_centre.data.core.weather.surface-based-observations.
↪synop
    country: mwi
    centre_id: mwi_met_centre

mcf:
    version: 1.0

metadata:
    identifier: urn:x-wmo:md:mwi:mwi_met_centre:surface-weather-observations
    hierarchylevel: dataset

identification:
    title: Surface weather observations from Malawi
    abstract: Surface weather observations from Malawi
    dates:
        creation: 2021-11-29
    keywords:
        default:
```

(continues on next page)

(continued from previous page)

```
            keywords:
                - surface weather
                - temperature
                - observations
        wmo:
            keywords:
                - weatherObservations
            keywords_type: theme
            vocabulary:
                name: WMO Category Code
                url: https://github.com/wmo-im/wcmp-codelists/blob/main/codelists/WMO_
↪CategoryCode.csv
    extents:
        spatial:
            - bbox: [32.6881653175,-16.8012997372,35.7719047381,-9.23059905359]
              crs: 4326
        temporal:
            - begin: 2021-11-29
              end: null
              resolution: P1H
    url: https://example.org/malawi-surface-weather-observations
    wmo_data_policy: core

contact:
    pointOfContact: &contact_poc
        organization: Department of Climate Change and Meteorologial Services (DCCMS)
        url: https://www.metmalawi.gov.mw
        individualname: Firstname Lastname
        positionname: Position Name
        phone: +265-1-822-014
        fax: +265-1-822-215
        address: P.O. Box 1808
        city: Blantyre
        administrativearea: Blantyre District
        postalcode: M3H 5T4
        country: Malawi
        email: you@example.org
        hoursofservice: 0700h - 1500h UTC
        contactinstructions: email

    distributor: *contact_poc
```

Note: There are no conventions to the MCF filename. The filename does not get used/exposed or published. It is up to the user to determine the best filename, keeping in mind your wis2box system may manage and publish numerous datasets (and MCF files) over time.

Summary

At this point, you have created discovery metadata for your given dataset(s).

Data ingest, processing and publishing

At this point, the system is ready for ingest/processing and publishing.

Data ingest, processing and publishing can be run in automated fashion or via the wis2box CLI. Data is ingested, processed, and published as WMO BUFR data, as well as GeoJSON features.

Warning: GeoJSON **data** representations provided in wis2box are in development and are subject to change based on evolving requirements for observation data representations in WIS2 technical regulations.

Interactive ingest, processing and publishing

The *wis2box* CLI provides a data subsystem to process data interactively. CLI data ingest/processing/publishing can be run with explicit or implicit topic hierarchy routing (which needs to be tied to the pipeline via the *Data mappings*).

Explicit topic hierarchy workflow

```
# process a single CSV file
wis2box data ingest --topic-hierarchy foo.bar.baz -p /path/to/file.csv

# process a directory of CSV files
wis2box data ingest --topic-hierarchy foo.bar.baz -p /path/to/dir

# process a directory of CSV files recursively
wis2box data ingest --topic-hierarchy foo.bar.baz -p /path/to/dir -r
```

Implicit topic hierarchy workflow

```
# process incoming data; topic hierarchy is inferred from fuzzy filepath equivalent
# wis2box will detect 'foo/bar/baz' as topic hierarchy 'foo.bar.baz'
wis2box data ingest -p /path/to/foo/bar/baz/data/file.csv
```

Event driven ingest, processing and publishing

Once all metadata and topic hierarchies are setup, event driven workflow will immediately start to listen on files in the `wis2box-incoming` storage bucket as they are placed in the appropriate topic hierarchy directory.

Data pipeline plugins

Driven by topic hierarchies, wis2box is a plugin architecture orchestrating all the required components of a WIS2 Node. wis2box also provides a data pipeline plugin architecture which allows for users to define a plugin based on a topic hierarchy to publish incoming data (see *Data mappings* for more information).

See also:

Extending wis2box

See also:

Data mappings

Default pipeline plugins

wis2box provides a number of data pipeline plugins by default, which users can be used "out of the box". The list below describes each plugin and provides an example data mappings configuration.

`wis2box.data.csv2bufr.ObservationDataCSV2BUFR`

This plugin converts CSV observation data into BUFR using `csv2bufr`. A csv2bufr template can be configured to process the data accordingly. In addition, `file-pattern` can be used to filter on incoming data based on a regular expression. Consult the csv2bufr documentation for more information on configuration and templating.

A typical csv2bufr plugin workflow definition would by defined as follows:

```
csv:
    - plugin: wis2box.data.csv2bufr.ObservationDataCSV2BUFR
      template: /data/wis2box/synop_bufr.json  # locally created csv2bufr mapping␣
↪(located in $WIS2BOX_HOST_DATADIR)
      notify: true  # trigger GeoJSON publishing for API and UI
      file-pattern: '^.*\.csv$'
```

`wis2box.data.bufr4.ObservationDataBUFR2GeoJSON`

This plugin is typically used for wis2box API publication, and converts BUFR observation data into GeoJSON using `bufr2geojson`. A `file-pattern` can be used to filter on incoming data based on a regular expression. Consult the bufr2geojson documentation for more information on configuration and templating.

A typical bufr2geojson plugin workflow definition would be defined as follows:

```
bufr4:
    - plugin: wis2box.data.bufr2geojson.ObservationDataBUFR2GeoJSON
      file-pattern: '^.*\.bufr4$'
```

wis2box.data.geojson.ObservationDataGeoJSON

This plugin is for the purposes of publishing GeoJSON data to the API.

wis2box.data.synop2bufr.SYNOP2BUFR

This plugin converts SYNOP ASCII data into BUFR using `synop2bufr`. A `file-pattern` can be used to filter on incoming data based on a regular expression.

Note that the regular expression **must** contain two groups (for 4-digit year and 2-digit month), which are used as part of synop2bufr processing. Consult the synop2bufr documentation for more information.

A typical synop2bufr plugin workflow definition would be defined as follows:

```
txt:
    - plugin: wis2box.data.synop2bufr.ObservationDataSYNOP2BUFR
      notify: true  # trigger GeoJSON publishing for API and UI
      file-pattern: '^station_123_(\d{4})(\d{2}).*.txt$'  # example: station_123_202305_
↪112342.txt (where 2023 is the year and 05 is the month)
```

This plugin takes an incoming BUFR4 data file and separates it into individual BUFR bulletins if there is more than one in a file. Those bulletins are then further divided into individual subsets for publication on WIS2. As part of the process, files are quality checked for valid WIGOS Station Identifiers and location information. Where these are missing, the information is either infilled using the wis2box station list or the subset discarded if no match is found. Missing temporal information results in the data being discarded.

For processing efficiency, and to allow for concurrent processing, it is recommended that the input data to this plugin is already separated into one BUFR message per file and one subset per message.

A typical BUFR4 plugin workflow definition would be defined as follows:

```
bin:
    - plugin: wis2box.data.bufr4.ObservationDataBUFR
      notify: true  # trigger GeoJSON publishing for API and UI
      file-pattern: '^.*\.bin$'
```

See *Data mappings* for a full example data mapping configuration.

API publishing

When wis2box starts, the API provisioning environment is initialized. At this stage, the following steps are required:

- station metadata has been configured
- discovery metadata has been created
- data pipelines are configured and running

Let's dive into publishing the data and metadata:

wis2box provides an API supporting the OGC API suite of standards using pygeoapi.

Station metadata API publishing

The first step is to publish our station metadata to the API. The command below will generate local station collection GeoJSON for API publication.

```
wis2box metadata station publish-collection
```

Note: This command also runs automatically at startup and thereafter every 10 minutes to keep your stations up to date.

See also:

Station metadata

Discovery metadata API publishing

This step will publish dataset discovery metadata to the API.

```
wis2box metadata discovery publish /path/to/discovery-metadata.yml
```

Dataset collection API publishing

The below command will add the dataset collection to pygeoapi from the discovery metadata MCF created as described in the *Discovery metadata* section.

```
wis2box data add-collection $WIS2BOX_DATADIR/data/config/foo/bar/baz/discovery-metadata.
↪yml
```

To delete the colection from the API backend and configuration:

```
wis2box api delete-collection foo.bar.baz
```

Note: Changes to the API configuration are reflected and updated automatically.

Summary

At this point, you have successfully published the required data and metadata collections to the API.

Data retention

wis2box is configured to set data retention according to your requirements. Data retention is managed via the WIS2BOX_STORAGE_DATA_RETENTION_DAYS environment variable as part of configuring wis2box.

Cleaning

Cleaning applies to storage defined by WIS2BOX_STORAGE_PUBLIC and involves the deletion of files after set amount of time.

Cleaning is performed by default daily at 0Z by the system, and can also be run interactively with:

```
# delete data older than WIS2BOX_STORAGE_DATA_RETENTION_DAYS by default
wis2box data clean


# delete data older than --days (force override)
wis2box data clean --days=30
```

Archiving

Archiving applies to storage defined by WIS2BOX_STORAGE_INCOMING and involves moving files to the storage defined by WIS2BOX_STORAGE_ARCHIVE.

Archive is performed on incoming data by default daily at 1Z by the system, and can also be run interactively with:

```
wis2box data archive
```

Only files with a timestamp older than one hour are considered for archiving.

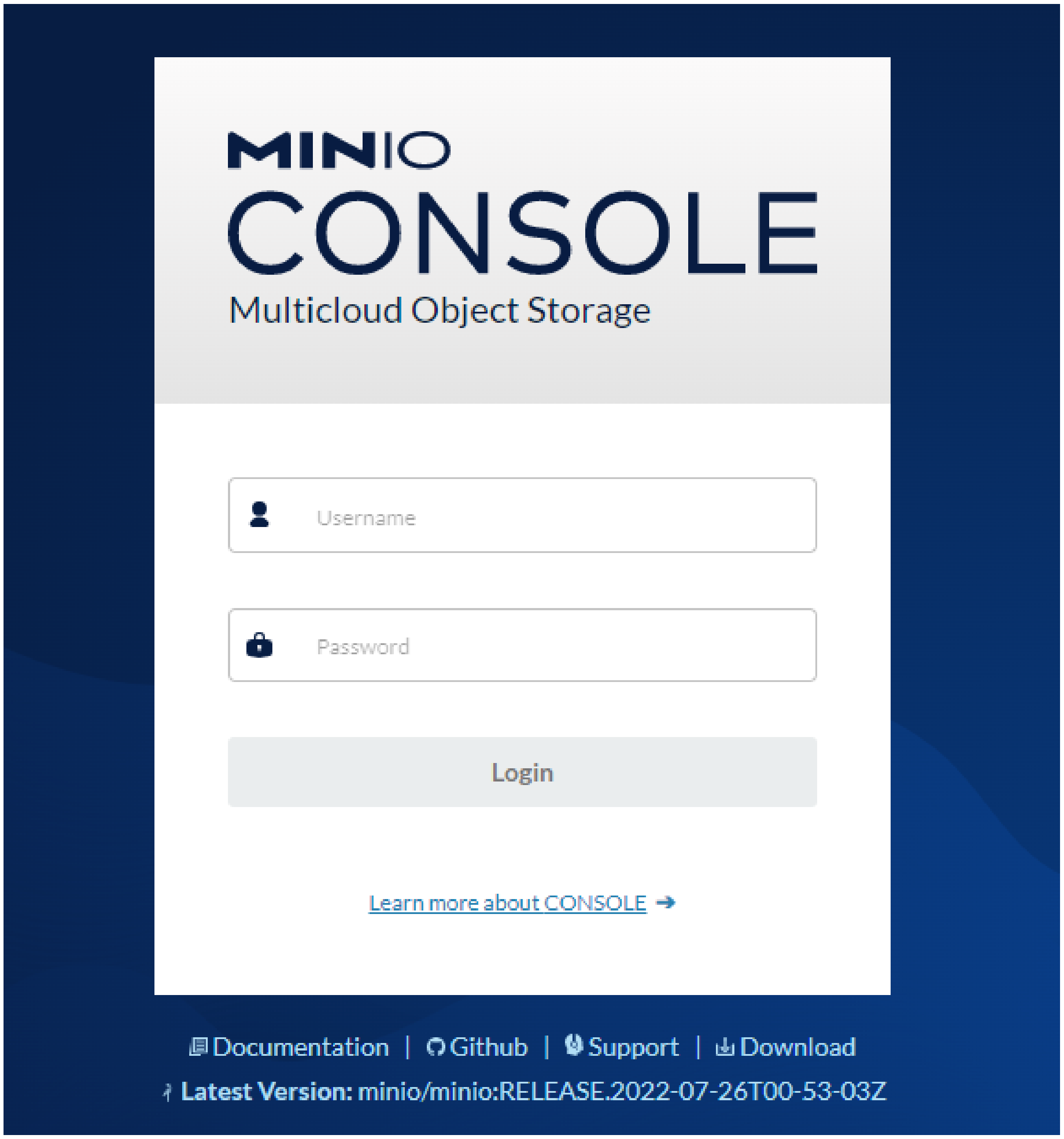
2.7 Storage

2.7.1 Overview

The default wis2box storage capability is powered by MinIO, which provides S3 compatible object storage.

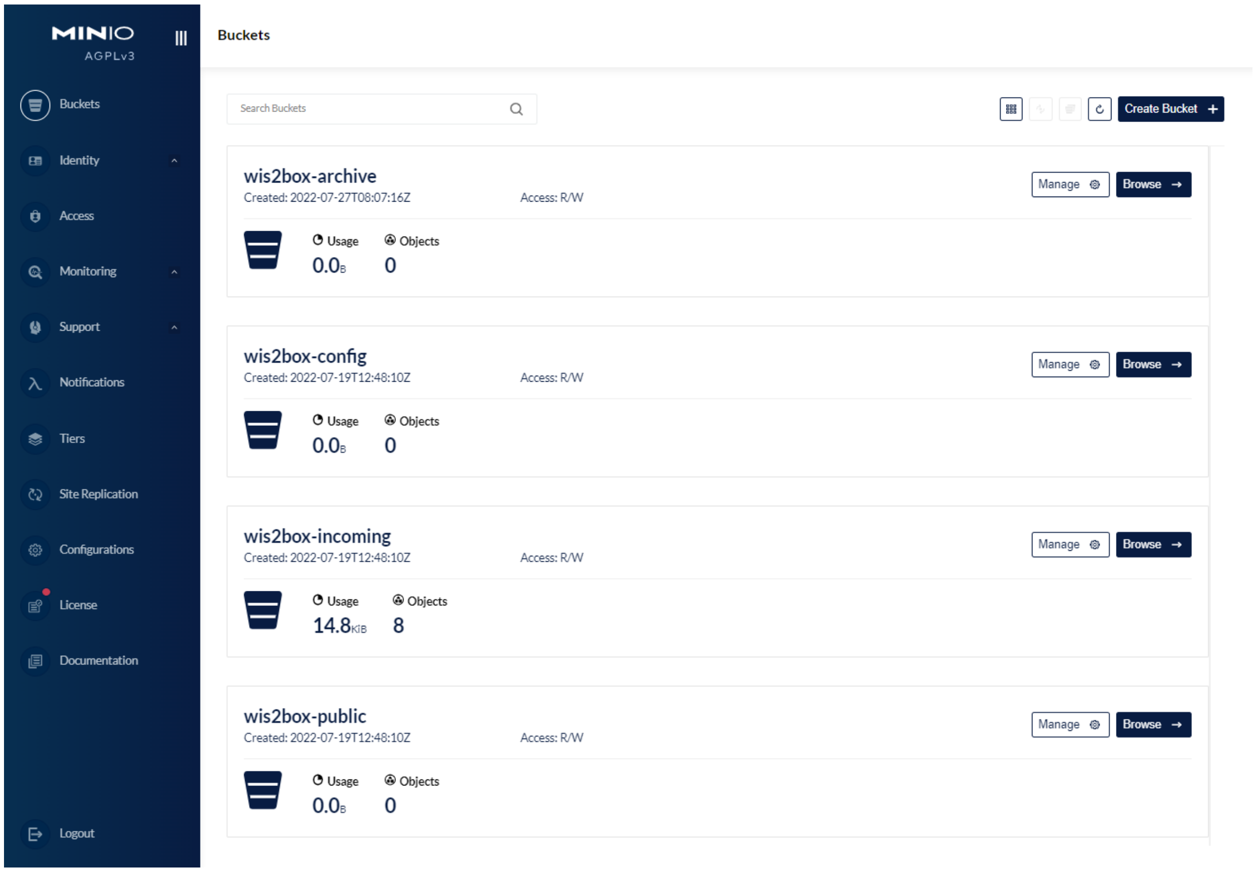
The default wis2box MinIO administration user interface can be accessed locally at `http://localhost:9001`.

The username/password for MinIO is configured through environment variables (see *Configuration*).


MINIO
CONSOLE
Multicloud Object Storage
Username
Password
Login
Learn more about CONSOLE
Documentation | Github | Support | Download

Latest Version: minio/minio:RELEASE.2022-07-26T00-53-03Z

Once logged in, buckets can be managed via the default “Buckets” menu item (click “Manage”). Click “Browse” provides a browsing capability for a storage administrator.



Buckets

wis2box-archive
Created: 2022-07-27T08:07:16Z Access: R/W
Usage 0.0 B Objects 0

wis2box-config
Created: 2022-07-19T12:48:10Z Access: R/W
Usage 0.0 B Objects 0

wis2box-incoming
Created: 2022-07-19T12:48:10Z Access: R/W
Usage 14.8 KiB Objects 8

wis2box-public
Created: 2022-07-19T12:48:10Z Access: R/W
Usage 0.0 B Objects 0

2.7.2 Uploading data to MinIO

Files can uploaded to the MinIO bucket in a number of ways. Any new file received on MinIO will trigger an MQTT notification which is received by wis2box.

Below are basic examples on sending data to the MinIO `wis2box-incoming` bucket. For more information and additional examples, consult the official MinIO documentation.

Using the boto3 Python Client

Install the Python boto3 package via pip:

```
pip3 install boto3
```

The below example copies a local file (`myfile.csv`) to the `wis2box-incoming` bucket with topic `foo.bar.baz`:

```python
import boto3

endpoint_url = '<your-wis2box-url>'
filename = 'myfile.csv'

session = boto3.Session(
    aws_access_key_id='wis2box',
    aws_secret_access_key='Wh00data!'
```

(continues on next page)

(continued from previous page)

```
)

s3client = session.client('s3', endpoint_url=endpoint_url)

with open(filename, 'rb') as fh:
    s3client.upload_fileobj(fh, 'wis2box-incoming', f'foo/bar/baz/{filename}')
```

To allow uploading files into MinIO remotely, the `wis2box-incoming` bucket is proxied via Nginx.

For example, to upload the local file (`WIGOS_0-454-2-AWSNAMITAMBO_2021-11-18T0955.csv with topic`) with topic `mwi.mwi_met_centre.data.core.weather.surface-based-observations.synop` via the Nbinx proxy:

```
import boto3

endpoint_url = 'http://localhost:9000'
filename = 'myfile.csv'

session = boto3.Session(
    aws_access_key_id='wis2box',
    aws_secret_access_key='Wh00data!'
)

s3client = session.client('s3', endpoint_url=endpoint_url)

filename = 'WIGOS_0-454-2-AWSNAMITAMBO_2021-11-18T0955.csv'

with open(filename, 'rb') as f:
    s3client.upload_fileobj(f, 'wis2box-incoming', f'data/core/observations-surface-land/
↪mw/FWCL/landFixed/{filename}')
```

Using the MinIO Python Client

MinIO provides a Python client which can be used as follows:

Install the Python minio module via pip:

```
pip3 install minio
```

The below example copies a local file (`myfile.csv`) to the `wis2box-incoming` bucket to topic `foo.bar.baz`:

```
from minio import Minio

client = Minio(
    'localhost:9000',
    access_key='minio',
    secret_key='minio123',
    secure=False
)

client.fput_object('wis2box-incoming', 'myfile.csv', '/foo/bar/baz/myfile.csv')
```

Using S3cmd

Given MinIO is S3 compatible, data can be uploaded using generic S3 tooling. The below example uses S3cmd to upload data to wis2box MinIO storage:

Edit the following fields in `~/.s3cfg`:

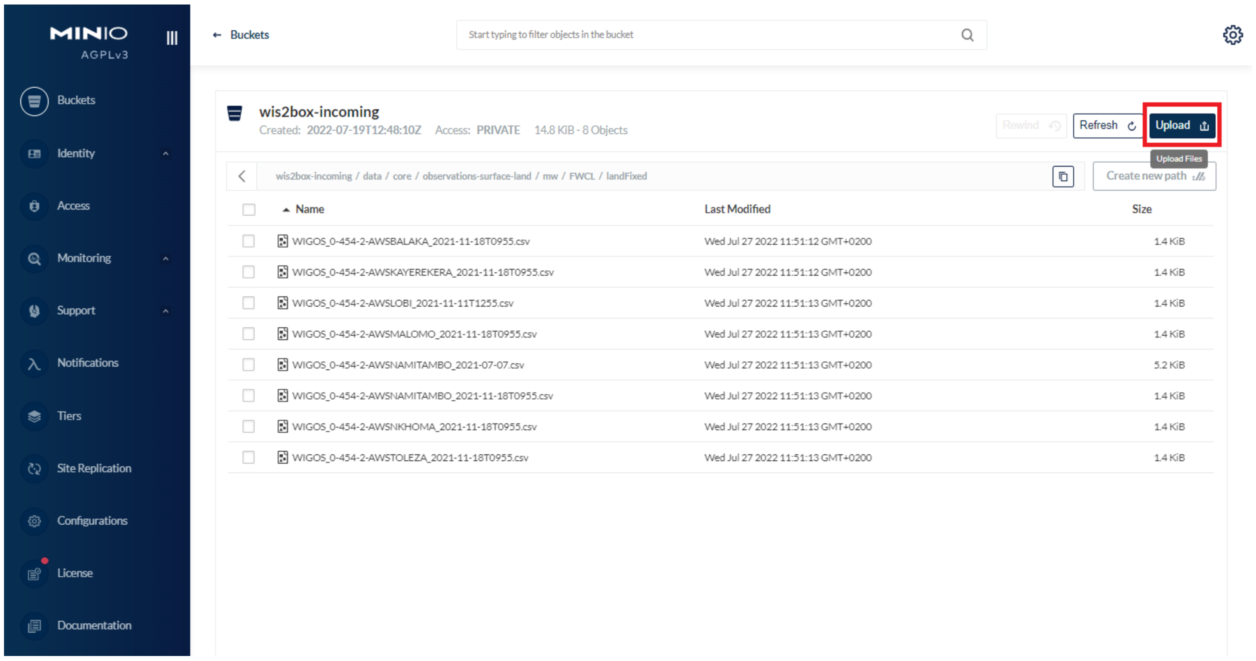
```
cat << EOF > ~/.s3cfg
# Setup endpoint
host_base = localhost:9000
use_https = False

# Setup access keys
access_key = minio
secret_key = minio123
EOF
```

Below is a simple command line example to copy a local file called `myfile.csv` into the `wis2box-incoming` bucket, to topic `foo/bar/baz`:

```
s3cmd myfile.csv s3://wis2box-incoming/foo/bar/baz
```

Using the MinIO UI

Files can also be uploaded interactively via the MinIO adminstration interface. The example below demonstrates this capability when browsing the `wis2box-incoming` bucket:



wis2box-incoming
Created: 2022-07-19T12:48:10Z Access: PRIVATE 14.8 KiB - 8 Objects

wis2box-incoming / data / core / observations-surface-land / mw / FWCL / landFixed

Name	Last Modified	Size
WIGOS_0-454-2-AWSBALAKA_2021-11-18T0955.csv	Wed Jul 27 2022 11:51:12 GMT+0200	1.4 KiB
WIGOS_0-454-2-AWSKAYEREKERA_2021-11-18T0955.csv	Wed Jul 27 2022 11:51:12 GMT+0200	1.4 KiB
WIGOS_0-454-2-AWSLOBI_2021-11-11T1255.csv	Wed Jul 27 2022 11:51:13 GMT+0200	1.4 KiB
WIGOS_0-454-2-AWSMALOMO_2021-11-18T0955.csv	Wed Jul 27 2022 11:51:13 GMT+0200	1.4 KiB
WIGOS_0-454-2-AWSNAMITAMBO_2021-07-07.csv	Wed Jul 27 2022 11:51:13 GMT+0200	5.2 KiB
WIGOS_0-454-2-AWSNAMITAMBO_2021-11-18T0955.csv	Wed Jul 27 2022 11:51:13 GMT+0200	1.4 KiB
WIGOS_0-454-2-AWSNKHOMA_2021-11-18T0955.csv	Wed Jul 27 2022 11:51:13 GMT+0200	1.4 KiB
WIGOS_0-454-2-AWSTOLEZA_2021-11-18T0955.csv	Wed Jul 27 2022 11:51:13 GMT+0200	1.4 KiB

2.8 Monitoring

wis2box has built-in monitoring functions based on Prometheus, Loki and Grafana.

The Grafana endpoint can be visualized at http://localhost/monitoring.

Grafana uses two data sources to display monitoring data:

- Prometheus: actively 'scrapes' data from the configured prometheus-client exporters every X seconds
- Loki: logging endpoint for the Docker containers that compose the wis2box

2.8.1 Prometheus exporters for wis2box

The exporters for wis2box are based on the Prometheus Python Client

- mqtt_metric_collector: collects data on messages published, using an mqtt-session subscribed to the wis2box-broker

wis2box also analyzes prometheus metrics from MinIO.

Note: For more information see the list of supported MinIO metrics

2.8.2 Loki logging

The logs of the following Docker containers are sent to Loki:

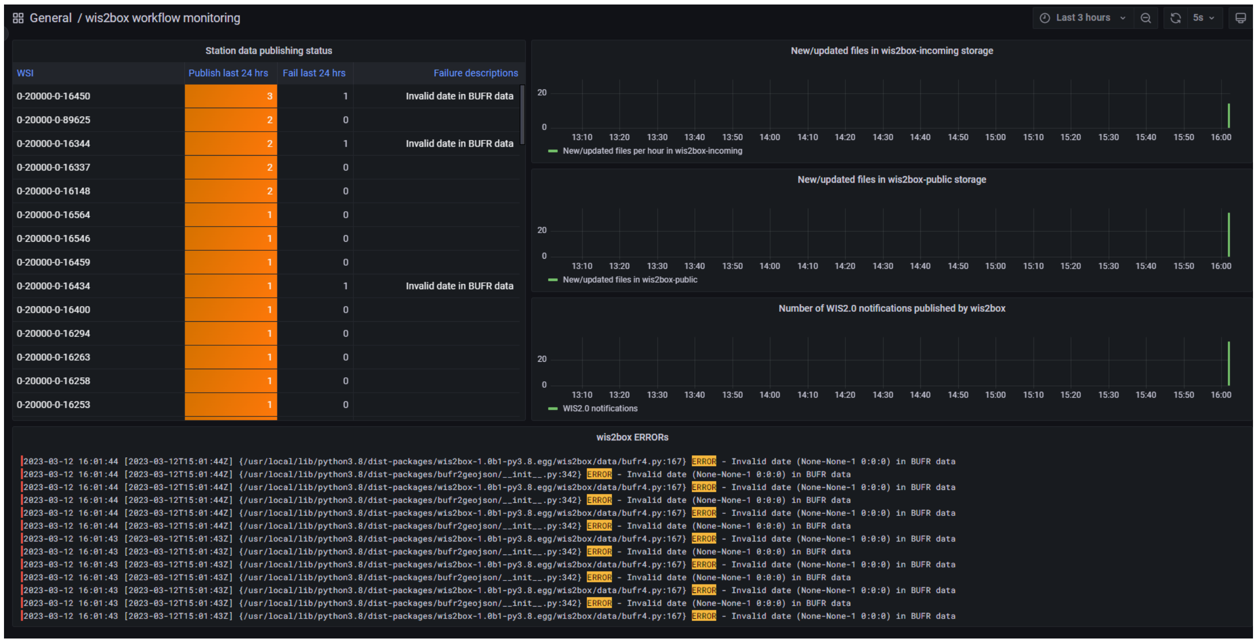
- mosquitto
- mqp-publisher
- wis2box
- wis2box-api
- wis2box-auth
- wis2box-ui

2.8.3 Monitoring topics

Grafana dashboards

wis2box provides a Grafana dashboard in order to visualize and analyze various metrics.

Go to http://localhost:3000 to see the home dashboard of wis2box once the stack is running.


General / wis2box workflow monitoring
Last 3 hours
5s
New/updated files in wis2box-incoming storage
New/updated files per hour in wis2box-incoming
New/updated files in wis2box-public storage
New/updated files in wis2box-public
Number of WIS2.0 notifications published by wis2box
WIS2.0 notifications

Station data publishing status

WSI	Publish last 24 hrs	Fail last 24 hrs	Failure descriptions
0-20000-0-16450	3	1	Invalid date in BUFR data
0-20000-0-89625	2	0	
0-20000-0-16344	2	1	Invalid date in BUFR data
0-20000-0-16337	2	0	
0-20000-0-16148	2	0	
0-20000-0-16564	1	0	
0-20000-0-16546	1	0	
0-20000-0-16459	1	0	
0-20000-0-16434	1	1	Invalid date in BUFR data
0-20000-0-16400	1	0	
0-20000-0-16294	1	0	
0-20000-0-16263	1	0	
0-20000-0-16258	1	0	
0-20000-0-16253	1	0	

wis2box ERRORs

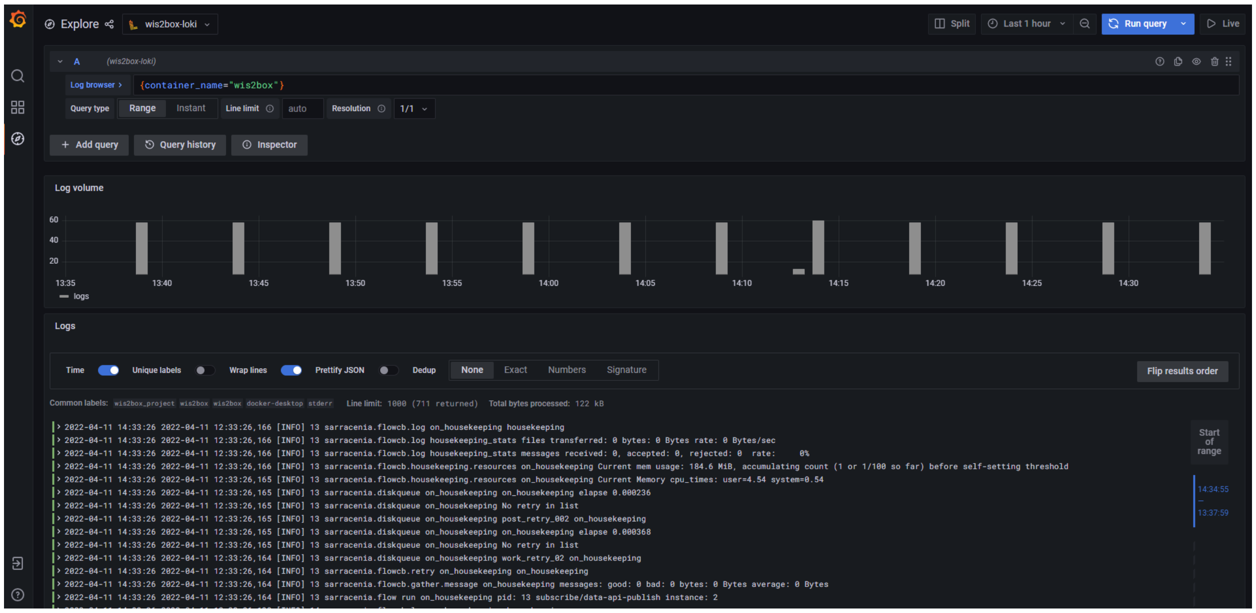
```
2023-03-12 16:01:44 [2023-03-12T15:01:44Z] {/usr/local/lib/python3.8/dist-packages/wis2box-1.0b1-py3.8.egg/wis2box/data/bufr4.py:167} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:44 [2023-03-12T15:01:44Z] {/usr/local/lib/python3.8/dist-packages/bufr2geojson/__init__.py:342} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:44 [2023-03-12T15:01:44Z] {/usr/local/lib/python3.8/dist-packages/wis2box-1.0b1-py3.8.egg/wis2box/data/bufr4.py:167} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:44 [2023-03-12T15:01:44Z] {/usr/local/lib/python3.8/dist-packages/bufr2geojson/__init__.py:342} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:44 [2023-03-12T15:01:44Z] {/usr/local/lib/python3.8/dist-packages/wis2box-1.0b1-py3.8.egg/wis2box/data/bufr4.py:167} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:44 [2023-03-12T15:01:44Z] {/usr/local/lib/python3.8/dist-packages/bufr2geojson/__init__.py:342} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:43 [2023-03-12T15:01:43Z] {/usr/local/lib/python3.8/dist-packages/wis2box-1.0b1-py3.8.egg/wis2box/data/bufr4.py:167} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:43 [2023-03-12T15:01:43Z] {/usr/local/lib/python3.8/dist-packages/bufr2geojson/__init__.py:342} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:43 [2023-03-12T15:01:43Z] {/usr/local/lib/python3.8/dist-packages/wis2box-1.0b1-py3.8.egg/wis2box/data/bufr4.py:167} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:43 [2023-03-12T15:01:43Z] {/usr/local/lib/python3.8/dist-packages/bufr2geojson/__init__.py:342} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:43 [2023-03-12T15:01:43Z] {/usr/local/lib/python3.8/dist-packages/wis2box-1.0b1-py3.8.egg/wis2box/data/bufr4.py:167} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:43 [2023-03-12T15:01:43Z] {/usr/local/lib/python3.8/dist-packages/bufr2geojson/__init__.py:342} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
2023-03-12 16:01:43 [2023-03-12T15:01:43Z] {/usr/local/lib/python3.8/dist-packages/wis2box-1.0b1-py3.8.egg/wis2box/data/bufr4.py:167} ERROR - Invalid date (None-None-1 0:0:0) in BUFR data
```

Note: The dashboard configuration can be found in `grafana/dashboards/home.json`.

Exploring logs

You can explore logs by selecting explore from the side-bar in Grafana.

Select `wis2box-loki` as a data source to browse the logs produced by the Docker containers that compose wis2box:



```
{container_name="wis2box"}
```

Query type: Range · Line limit: auto · Resolution: 1/1

Time: on · Unique labels: off · Wrap lines: on · Prettify JSON: off · Dedup: None

Common labels: wis2box_project wis2box wis2box docker-desktop stderr   Line limit: 1000 (711 returned)   Total bytes processed: 122 kB

```
> 2022-04-11 14:33:26 2022-04-11 12:33:26,166 [INFO] 13 sarracenia.flowcb.log on_housekeeping housekeeping
> 2022-04-11 14:33:26 2022-04-11 12:33:26,166 [INFO] 13 sarracenia.flowcb.log housekeeping_stats files transferred: 0 bytes: 0 Bytes rate: 0 Bytes/sec
> 2022-04-11 14:33:26 2022-04-11 12:33:26,166 [INFO] 13 sarracenia.flowcb.log housekeeping_stats messages received: 0, accepted: 0, rejected: 0  rate:     0%
> 2022-04-11 14:33:26 2022-04-11 12:33:26,166 [INFO] 13 sarracenia.flowcb.housekeeping.resources on_housekeeping Current mem usage: 184.6 MiB, accumulating count (1 or 1/100 so far) before self-setting threshold
> 2022-04-11 14:33:26 2022-04-11 12:33:26,165 [INFO] 13 sarracenia.flowcb.housekeeping.resources on_housekeeping Current Memory cpu_times: user=4.54 system=0.54
> 2022-04-11 14:33:26 2022-04-11 12:33:26,165 [INFO] 13 sarracenia.diskqueue on_housekeeping on_housekeeping elapse 0.000236
> 2022-04-11 14:33:26 2022-04-11 12:33:26,165 [INFO] 13 sarracenia.diskqueue on_housekeeping No retry in list
> 2022-04-11 14:33:26 2022-04-11 12:33:26,165 [INFO] 13 sarracenia.diskqueue on_housekeeping post_retry_002 on_housekeeping
> 2022-04-11 14:33:26 2022-04-11 12:33:26,165 [INFO] 13 sarracenia.diskqueue on_housekeeping on_housekeeping elapse 0.000368
> 2022-04-11 14:33:26 2022-04-11 12:33:26,165 [INFO] 13 sarracenia.diskqueue on_housekeeping No retry in list
> 2022-04-11 14:33:26 2022-04-11 12:33:26,164 [INFO] 13 sarracenia.diskqueue on_housekeeping work_retry_02 on_housekeeping
> 2022-04-11 14:33:26 2022-04-11 12:33:26,164 [INFO] 13 sarracenia.flowcb.retry on_housekeeping on_housekeeping
> 2022-04-11 14:33:26 2022-04-11 12:33:26,164 [INFO] 13 sarracenia.flowcb.gather.message on_housekeeping messages: good: 0 bad: 0 bytes: 0 Bytes average: 0 Bytes
> 2022-04-11 14:33:26 2022-04-11 12:33:26,164 [INFO] 13 sarracenia.flow run on_housekeeping pid: 13 subscribe/data-api-publish instance: 2
```

2.9 Services

wis2box provides a number of data access services and mechanisms in providing data to users, applications and beyond.

2.9.1 Discovery Catalogue

The discovery catalogue is powered by OGC API - Records and is located at http://localhost/oapi/collections/discovery-metadata

The OGC API endpoint is located by default at http://localhost/oapi. The discovery catalogue endpoint is located at http://localhost/oapi/collections/discovery-metadata

Below are some examples of working with the discovery catalogue.

- description of catalogue: http://localhost/oapi/collections/discovery-metadata
- catalogue queryables: http://localhost/oapi/collections/discovery-metadata/queryables
- catalogue queries
  - records (browse): http://localhost/oapi/collections/discovery-metadata/items
  - query by spatial (bounding box): http://localhost/oapi/collections/discovery-metadata/items?bbox=32,-17,36,-8
  - query by temporal extent (since): http://localhost/oapi/collections/discovery-metadata/items?datetime=2021/..
  - query by temporal extent (before): http://localhost/oapi/collections/discovery-metadata/items?datetime=../2022
  - query by freetext: http://localhost/oapi/collections/discovery-metadata/items?q=observations

Note:

- adding `f=json` to URLs will provide the equivalent JSON/GeoJSON representations
- query predicates (`datetime`, `bbox`, `q`, etc.) can be combined

See also:

Data access

2.9.2 Data API

wis2box data is made available via OGC API - Features and is located at http://localhost/oapi standards.

The OGC API endpoint is located by default at http://localhost/oapi

Below are some examples of working with the discovery catalogue.

Note:

- the examples below use the `mwi.mwi_met_centre.data.core.weather.surface-based-observations.synop` collection as described in the *Quickstart with test data*. For other dataset collections, use the same query patterns below, substituting the collection id accordingly

- list of dataset collections: http://localhost/oapi/collections

- collection description: http://localhost/oapi/collections/mwi.mwi_met_centre.data.core.weather.surface-based-observations.synop
- collection queryables: http://localhost/oapi/collections/mwi.mwi_met_centre.data.core.weather.surface-based-observations.synop/queryables
- collection items (browse): http://localhost/oapi/collections/mwi.mwi_met_centre.data.core.weather.surface-based-observations.synop/items
- collection queries
  - set limit/offset (paging): http://localhost/oapi/collections/mwi.mwi_met_centre.data.core.weather.surface-based-observations.synop/items?limit=1&startindex=2
  - query by spatial (bounding box): http://localhost/oapi/collections/mwi.mwi_met_centre.data.core.weather.surface-based-observations.synop/items?bbox=32,-17,36,-8
  - query by temporal extent (since): http://localhost/oapi/collections/mwi.mwi_met_centre.data.core.weather.surface-based-observations.synop/items?datetime=2021/..
  - query by temporal extent (before): http://localhost/oapi/collections/mwi.mwi_met_centre.data.core.weather.surface-based-observations.synop/items?datetime=../2022

Note:

- adding `f=json` to URLs will provide the equivalent JSON/GeoJSON representations
- query predicates (`datetime`, `bbox`, `q`, etc.) can be combined

See also:

Data access

Management API

The Data API also provides a management API to manage resources in alignment with OGC API - Features - Part 4: Create, Replace, Update and Delete, which is available at http://localhost/oapi/admin.

2.9.3 SpatioTemporal Asset Catalog (STAC)

The wis2box SpatioTemporal Asset Catalog (STAC) endpoint can be found at:

http://localhost/stac

…providing the user with a crawlable catalogue of all data on a wis2box.

2.9.4 Web Accessible Folder (WAF)

The wis2box Web Accessible Folder publich bucket endpoint can be found at:

http://localhost/data/

…providing the user with a crawlable online folder of all data on a wis2box.

2.9.5 Broker

The wis2box broker is powered by MQTT and can be found at:

mqtt://everyone:everyone@localhost:1883

mqtt://localhost:1883

. . . providing a Pub/Sub capability for event driven subscription and access.

Note: The `everyone` user is defined by default for public readonly access (`origin/#`) as per WIS2 Node requirements.

2.9.6 Adding services

wis2box's architecture allows for additional services as required by adding Docker containers. Examples of additional services include adding a container for a samba share or FTP server. Key considerations for adding services:

- Storage buckets can be found at http://minio:9000
- Elasticsearch indexes can be found at the container/URL `http://elasticsearch:9200`

2.10 Authentication and access control

wis2box provides built in access control for the WAF and API on a topic hierarchy basis. Configuration is done using the wis2box command line utility. Authentication tokens are only required for topics that have access control configured.

2.10.1 Adding Access Control

All topic hierarchies in wis2box are open by default. A topic becomes closed, with access control applied, the first time a token is generated for a topic hierarchy.

Note: Make sure you are logged into the wis2box-management container when using the wis2box CLI

```
wis2box auth add-token --topic-hierarchy mwi.mwi_met_centre.data.core.weather.surface-
↪based-observations.synop mytoken
```

If no token is provided, a random string will be generated. Be sure to the record token now, there is no way to retrieve it once it is lost.

2.10.2 Authenticating

Token credentials can be validated using the wis2box command line utility.

```
wis2box auth show
wis2box auth has-access --topic-hierarchy mwi.mwi_met_centre.data.core.weather.surface-
↪based-observations.synop mytoken
wis2box auth has-access --topic-hierarchy mwi.mwi_met_centre.data.core.weather.surface-
↪based-observations.synop notmytoken
```

Once a token has been generated, access to any data of that topic in the WAF or API requires token authentication. Tokens are passed as a bearer token in the Authentication header or as an argument appended to the URI. Headers can be easily added to requests using cURL.

```
curl -H "Authorization: Bearer mytoken" "http://localhost/oapi/collections/mwi.mwi_met_
↪centre.data.core.weather.surface-based-observations.synop"
curl -H "Authorization: Bearer notmytoken" "http://localhost/oapi/collections/mwi.mwi_
↪met_centre.data.core.weather.surface-based-observations.synop"
```

2.10.3 Removing Access Control

A topic becomes open and no longer requires authentication when all tokens have been deleted. This can be done by deleting individual tokens, or all tokens for a given topic hierarchy.

```
wis2box auth remove-tokens --topic-hierarchy mwi.mwi_met_centre.data.core.weather.
↪surface-based-observations.synop
wis2box auth show
```

2.10.4 Extending Access Control

wis2box provides access control out of the box with subrequests to wis2box-auth. wis2box-auth could be replaced in nginx for another auth server like Gluu or a Web SSO like LemonLDAP or Keycloak. These services are not yet configurable via the wis2box command line utility.

wis2box is intentionally plug and playable. Beyond custom authentication servers, extending wis2box provides an overview of more modifications that can be made to wis2box.

2.11 Data access

2.11.1 Overview

This section provides examples of interacting with wis2box data services as described in *Services* using a number of common tools and software packages.

2.11.2 API

Using Python, requests and Pandas

Python is a popular programming language which is heavily used in the data science domains. Python provides high level functionality supporting rapid application development with a large ecosystem of packages to work with weather/climate/water data.

Let's use the Python requests package to further interact with the wis2box API, and Pandas to run some simple summary statistics.

[1]:
```
import json

import requests

```

(continues on next page)

(continued from previous page)

```
def pretty_print(input):
    print(json.dumps(input, indent=2))


# define the endpoint of the OGC API
api = 'http://localhost/oapi'
```

Stations

Let's find all the stations in our wis2box:

[2]:
```
url = f'{api}/collections/stations/items?limit=50'

response = requests.get(url).json()

print(f"Number of stations: {response['numberMatched']}")

print('Stations:\n')
for station in response['features']:
    print(station['properties']['name'])
```

```
Number of stations: 26
Stations:

NAMBUMA
BALAKA
BILIRA
CHIDOOLE
CHIKANGAWA
CHIKWEO
CHINGALE
KALAMBO
KASIYA AWS
KASUNGU NATIONAL PARK AWS
KAWALAZI
KAYEREKERA
LENGWE NATIONAL PARK
LOBI AWS
MAKANJIRA
MALOMO
MISUKU
MLARE
MLOMBA
MTOSA BENGA
NAMITAMBO
NANKUMBA
NKHOMA UNIVERSITY
NKHULAMBE
NYACHILENDA
TOLEZA
```

Discovery Metadata

Now, let's find all the dataset that are provided by the above stations. Each dataset is identified by a WIS2 discovery metadata record.

[3]:
```
url = f'{api}/collections/discovery-metadata/items'

response = requests.get(url).json()

print('Datasets:\n')
for dataset in response['features']:
    print(f"id: {dataset['properties']['id']}, title: {dataset['properties']['title']}")
```

```
Datasets:

id: data.core.test-passthrough, title: Surface weather observations (passthrough)
id: mwi.mwi_met_centre.data.core.weather.surface-based-observations.synop, title:␣
↪Surface weather observations (hourly)
```

Let's find all the data access links associated with the Surface weather observations (hourly) dataset:

[4]:
```
dataset_id = 'mwi.mwi_met_centre.data.core.weather.surface-based-observations.synop'

url = f"{api}/collections/discovery-metadata/items/{dataset_id}"

response = requests.get(url).json()

print('Data access links:\n')
for link in response['links']:
    print(f"{link} {link['href']} ({link['type']}) {link['rel']}")
    link['rel']

[link['href'] for link in response['links']]
```

```
Data access links:

{'rel': 'self', 'type': 'application/geo+json', 'title': 'This document as GeoJSON',
↪'href': 'http://localhost/oapi/collections/discovery-metadata/items/mwi.mwi_met_centre.
↪data.core.weather.surface-based-observations.synop?f=json'} http://localhost/oapi/
↪collections/discovery-metadata/items/mwi.mwi_met_centre.data.core.weather.surface-
↪based-observations.synop?f=json (application/geo+json) self
{'rel': 'alternate', 'type': 'application/ld+json', 'title': 'This document as RDF (JSON-
↪LD)', 'href': 'http://localhost/oapi/collections/discovery-metadata/items/mwi.mwi_met_
↪centre.data.core.weather.surface-based-observations.synop?f=jsonld'} http://localhost/
↪oapi/collections/discovery-metadata/items/mwi.mwi_met_centre.data.core.weather.surface-
↪based-observations.synop?f=jsonld (application/ld+json) alternate
{'rel': 'alternate', 'type': 'text/html', 'title': 'This document as HTML', 'href':
↪'http://localhost/oapi/collections/discovery-metadata/items/mwi.mwi_met_centre.data.
↪core.weather.surface-based-observations.synop?f=html'} http://localhost/oapi/
↪collections/discovery-metadata/items/mwi.mwi_met_centre.data.core.weather.surface-
↪based-observations.synop?f=html (text/html) alternate
{'rel': 'collection', 'type': 'application/json', 'title': 'Discovery metadata', 'href':
↪'http://localhost/oapi/collections/discovery-metadata'} http://localhost/oapi/
↪collections/discovery-metadata (application/json) collection
```

```
[4]: ['http://localhost/oapi/collections/discovery-metadata/items/mwi.mwi_met_centre.data.
↪core.weather.surface-based-observations.synop?f=json',
 'http://localhost/oapi/collections/discovery-metadata/items/mwi.mwi_met_centre.data.
↪core.weather.surface-based-observations.synop?f=jsonld',
 'http://localhost/oapi/collections/discovery-metadata/items/mwi.mwi_met_centre.data.
↪core.weather.surface-based-observations.synop?f=html',
 'http://localhost/oapi/collections/discovery-metadata']
```

Let's use the OGC API - Features (OAFeat) link to drill into the observations for Chidoole station

```
[5]: dataset_api_link = 'http://localhost/oapi/collections/mwi.mwi_met_centre.data.core.
↪weather.surface-based-observations.synop'

dataset_api_link
```

```
[5]: 'http://localhost/oapi/collections/mwi.mwi_met_centre.data.core.weather.surface-based-
↪observations.synop'
```

Observations

Let's inspect some of the data in the API's raw GeoJSON format:

```
[6]: url = f'{dataset_api_link}/items'

query_parameters = {
    'wigos_station_identifier': '0-454-2-AWSCHIDOOLE',
    'limit': 10000,
    'name': 'air_temperature'
}

response = requests.get(url, params=query_parameters).json()

pretty_print(response['features'][0])
```

```
{
  "id": "WIGOS_0-454-2-AWSCHINGALE_20220112T135500-25",
  "reportId": "WIGOS_0-454-2-AWSCHINGALE_20220112T135500",
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [
      35.11,
      -15.24,
      623.0
    ]
  },
  "properties": {
    "wigos_station_identifier": "0-454-2-AWSCHINGALE",
    "phenomenonTime": "2022-01-12T13:55:00Z",
    "resultTime": "2022-01-12T13:55:00Z",
    "name": "air_temperature",
    "value": 24.85,
    "units": "Celsius",
```

(continues on next page)

(continued from previous page)

```
    "description": null,
    "metadata": [
      {
        "name": "station_or_site_name",
        "value": null,
        "units": "CCITT IA5",
        "description": "Chingale"
      },
      {
        "name": "station_type",
        "value": 0,
        "units": "CODE TABLE",
        "description": "Automatic"
      },
      {
        "name": "height_of_barometer_above_mean_sea_level",
        "value": 624.0,
        "units": "m",
        "description": null
      },
      {
        "name": "height_of_sensor_above_local_ground_or_deck_of_marine_platform",
        "value": 1.5,
        "units": "m",
        "description": null
      }
    ],
    "index": 25,
    "fxxyyy": "012101",
    "id": "WIGOS_0-454-2-AWSCHINGALE_20220112T135500-25"
  }
}
```

Let's inspect what's measured at Chidoole:

```
[7]: print('Observed property:\n')
feature = response['features'][9]
print(f"{feature['properties']['name']} ({feature['properties']['units']})")
```

```
Observed property:

air_temperature (Celsius)
```

Pandas

Let’s use the GeoJSON to build a more user-friendly table

[8]:
```
import pandas as pd

datestamp = [obs['properties']['resultTime'] for obs in response['features']]
air_temperature = [obs['properties']['value'] for obs in response['features']]

d = {
    'Date/Time': datestamp,
    'Air temperature (°C)': air_temperature
}

df = pd.DataFrame(data=d)
```

[9]:
```
df
```

[9]:
```
                  Date/Time  Air temperature (°C)
0      2022-01-12T13:55:00Z                 24.85
1      2022-01-12T14:55:00Z                 27.25
2      2022-01-12T15:55:00Z                 26.65
3      2022-01-12T16:55:00Z                 25.95
4      2022-01-12T17:55:00Z                 25.45
...                     ...                   ...
5101   2022-06-09T12:55:00Z                 21.35
5102   2022-06-09T13:55:00Z                 22.25
5103   2022-06-09T14:55:00Z                 20.25
5104   2022-06-10T12:55:00Z                 23.75
5105   2022-06-10T14:55:00Z                 21.15

[5106 rows x 2 columns]
```

[10]:
```
print("Time extent\n")
print(f'Begin: {df["Date/Time"].min()}')
print(f'End: {df["Date/Time"].max()}')

print("Summary statistics:\n")
df[['Air temperature (°C)']].describe()
```

```
Time extent

Begin: 2022-01-12T13:55:00Z
End: 2022-06-10T14:55:00Z
Summary statistics:
```

[10]:
```
       Air temperature (°C)
count           5106.000000
mean              23.541559
std                4.053172
min               13.550000
25%               20.950000
50%               23.350000
```

(continues on next page)

(continued from previous page)

```
75%              26.350000
max              37.850000
```

[ ]:

```

```

Using Python and OWSLib

OWSLib is a Python package which provides Pythonic access to OGC APIs and web services. Let's see how easy it is to work with wis2box with standards-based tooling:

[1]:

```python
from owslib.ogcapi.features import Features

import pandas as pd

def pretty_print(input):
    print(json.dumps(input, indent=2))


api = 'http://localhost/oapi'
```

Let's load the wis2box API into OWSLib and inspect some data

[2]:

```python
oafeat = Features(api)

collections = oafeat.collections()
print(f'This OGC API Features endpoint has {len(collections["collections"])} datasets')

for dataset in collections['collections']:
    print(dataset['title'])

malawi_obs = oafeat.collection_items('mwi.mwi_met_centre.data.core.weather.surface-based-
↪observations.synop')
malawi_obs_df = pd.DataFrame(malawi_obs['features'])

# then filter by station
obs = oafeat.collection_items('mwi.mwi_met_centre.data.core.weather.surface-based-
↪observations.synop', wigos_station_identifier='0-454-2-AWSCHIDOOLE', name='air_
↪temperature', limit=10000)

datestamp = [obs['properties']['resultTime'] for obs in obs['features']]
air_temperature = [obs['properties']['value'] for obs in obs['features']]

d = {
    'Date/Time': datestamp,
    'Air temperature (°C)': air_temperature
}

df = pd.DataFrame(data=d)
```

```
This OGC API Features endpoint has 4 datasets
Surface weather observations (passthrough)
```

(continues on next page)

(continued from previous page)

```
Discovery metadata
Stations
Surface weather observations (hourly)
```

[3]:
```
df.dtypes
```

[3]:
```
Date/Time                  object
Air temperature (°C)     float64
dtype: object
```

[4]:
```
df.head(3)
```

[4]:
```
              Date/Time  Air temperature (°C)
0  2022-01-12T13:55:00Z                 24.85
1  2022-01-12T14:55:00Z                 27.25
2  2022-01-12T15:55:00Z                 26.65
```

[5]:
```
print("Time extent\n")
print(f'Begin: {df["Date/Time"].min()}')
print(f'End: {df["Date/Time"].max()}')

print("Summary statistics:\n")
df[['Air temperature (°C)']].describe()
```

```
Time extent

Begin: 2022-01-12T13:55:00Z
End: 2022-06-10T14:55:00Z
Summary statistics:
```

[5]:
```
       Air temperature (°C)
count           5106.000000
mean              23.541559
std                4.053172
min               13.550000
25%               20.950000
50%               23.350000
75%               26.350000
max               37.850000
```

[ ]:
```

```

R

R is a common programming language for data analysis and visualization. R provides easy access to various statiscal analysis libraries. We are going to use the R libraries: sf to load features, dplyr for data manipulation, and

Install Requirements

```
[ ]: install.packages("sf")
     install.packages("dplyr")
```

Import Requirements

```
[1]: library(sf)
     library(dplyr)

     oapi <- "http://oapi/oapi" # jupyter is run through docker
     #oapi = http://localhost/oapi # jupyter is run on host machine
```

```
Linking to GEOS 3.10.2, GDAL 3.4.1, PROJ 8.2.1; sf_use_s2() is TRUE


Attaching package: ‘dplyr’


The following objects are masked from ‘package:stats’:

    filter, lag


The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union
```

Stations

```
[2]: stations <- read_sf(paste0(oapi,"/collections/stations/items?f=json"))
     print(stations)
```

```
Simple feature collection with 7 features and 5 fields
Geometry type: POINT
Dimension:     XYZ
Bounding box:  xmin: 33.67305 ymin: -15.84052 xmax: 35.27428 ymax: -9.92951
z_range:       zmin: 618 zmax: 1288
Geodetic CRS:  WGS 84
# A tibble: 7 × 6
  wigos_station_identifier              name        url   status    id                 ␣
↪geometry
  <chr>                    <chr>       <chr> <chr>  <int>                <POINT [°]>
1 0-454-2-AWSLOBI          LOBI AWS    http... opera... 65618 Z (34.07244 -14.39528 12...
2 0-454-2-AWSKAYEREKERA KAYEREKERA  http... opera... 91840 Z (33.67305 -9.92951 848)
3 0-454-2-AWSMALOMO        MALOMO      http... opera... 91873 Z (33.83727 -13.14202 10...
```

(continues on next page)

(continued from previous page)

```
4 0-454-2-AWSNKHOMA     NKHOMA UNI... http... opera... 91875 Z (34.10468 -14.04422 12...
5 0-454-2-AWSTOLEZA     TOLEZA      http... opera... 91880    Z (34.955 -14.948 764)
6 0-454-2-AWSNAMITAMBO  NAMITAMBO   http... opera... 91885 Z (35.27428 -15.84052 80...
7 0-454-2-AWSBALAKA     BALAKA      http... opera... 91893 Z (34.96667 -14.98333 61...
```

Discovery Metadata

[3]:
```
discovery_metadata <- read_sf(paste0(oapi,"/collections/discovery-metadata/items"))
print(discovery_metadata)
```

```
Simple feature collection with 1 feature and 13 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: 32.68817 ymin: -16.8013 xmax: 35.7719 ymax: -9.230599
Geodetic CRS:  WGS 84
# A tibble: 1 × 14
  identifier externalId title description themes providers language type  extent
  <chr>      <chr>      <chr> <chr>       <chr>  <chr>     <chr>    <chr> <chr>
1 data.core... "[ { \"sc... Surf... Surface we... "[ { ... "[ { \"n... en      data... "
↪{ \"...
# ... with 5 more variables: created <date>, rights <chr>,
#   id <chr>, geometry <POLYGON [°]>
```

Observations

[4]:
```
malawi_obs <- read_sf(paste0(oapi,"/collections/mwi.mwi_met_centre.data.core.weather.
↪surface-based-observations.synop/items"))
print(malawi_obs)
```

```
Simple feature collection with 10 features and 7 fields
Geometry type: POINT
Dimension:     XYZ
Bounding box:  xmin: 35.27 ymin: -15.84 xmax: 35.27 ymax: -15.84
z_range:       zmin: 806 zmax: 806
Geodetic CRS:  WGS 84
# A tibble: 10 × 8
   identifier  phenomenonTime      resultTime          wigos_station_i... metadata
   <chr>       <dttm>              <dttm>              <chr>              <chr>
 1 WIGOS_0-45... 2021-07-07 14:55:00 2022-02-21 14:15:14 0-454-2-AWSNAMI... "[ { \"...
 2 WIGOS_0-45... 2021-07-07 15:55:00 2022-02-21 14:15:14 0-454-2-AWSNAMI... "[ { \"...
 3 WIGOS_0-45... 2021-07-07 16:55:00 2022-02-21 14:15:14 0-454-2-AWSNAMI... "[ { \"...
 4 WIGOS_0-45... 2021-07-07 17:55:00 2022-02-21 14:15:14 0-454-2-AWSNAMI... "[ { \"...
 5 WIGOS_0-45... 2021-07-07 18:55:00 2022-02-21 14:15:14 0-454-2-AWSNAMI... "[ { \"...
 6 WIGOS_0-45... 2021-07-07 19:55:00 2022-02-21 14:15:15 0-454-2-AWSNAMI... "[ { \"...
 7 WIGOS_0-45... 2021-07-07 20:55:00 2022-02-21 14:15:15 0-454-2-AWSNAMI... "[ { \"...
 8 WIGOS_0-45... 2021-07-07 21:55:00 2022-02-21 14:15:15 0-454-2-AWSNAMI... "[ { \"...
 9 WIGOS_0-45... 2021-07-07 22:55:00 2022-02-21 14:15:15 0-454-2-AWSNAMI... "[ { \"...
10 WIGOS_0-45... 2021-07-07 23:55:00 2022-02-21 14:15:15 0-454-2-AWSNAMI... "[ { \"...
# ... with 3 more variables: observations <chr>, id <chr>, geometry <POINT [°]>
```

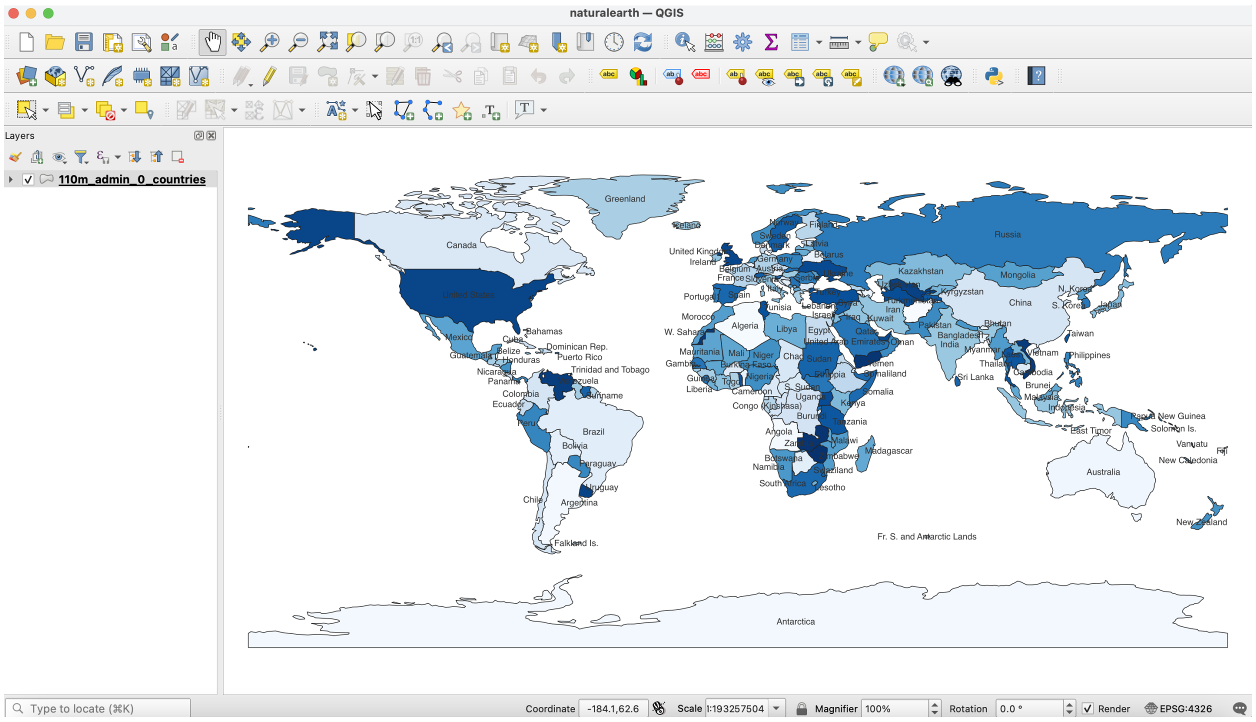
```
[ ]:
```

Using QGIS

Overview

This section provides examples of interacting with wis2box API using QGIS.

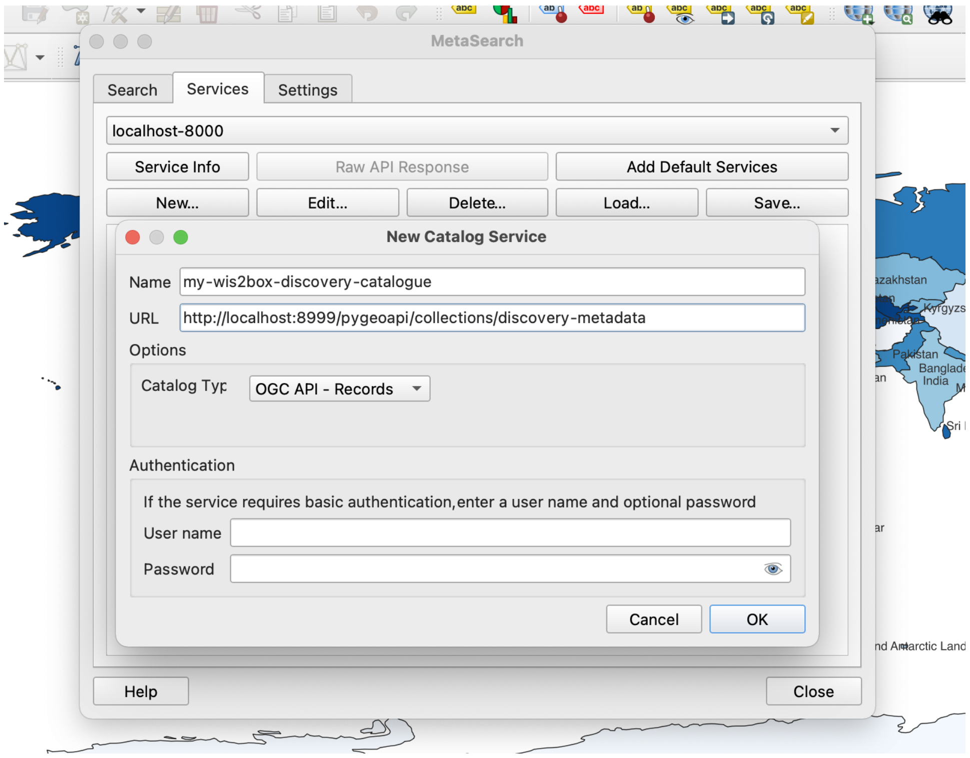
QGIS is a free and open-source cross-platform desktop GIS application that supports viewing, editing, and analysis of geospatial data. QGIS supports numerous format and encoding standards, which enables plug-and-play interoperability with wis2box data and discovery metadata.



Accessing the discovery catalogue

QGIS provides support for the OGC API - Records standard (discovery). To interact with the wis2box discovery catalogue:

- from the QGIS menu, select *Web -> MetaSearch -> MetaSearch*
- click the "Services" tab
- click "New"
- enter a name for the discovery catalogue endpoint
- enter the URL to the discovery catalogue endpoint (i.e. `http://localhost/oapi/collections/discovery-metadata`)
- ensure "Catalogue Type" is set to "OGC API - Records"
- click "OK"



localhost-8000

Name my-wis2box-discovery-catalogue

URL http://localhost:8999/pygeoapi/collections/discovery-metadata

Catalog Typ OGC API - Records

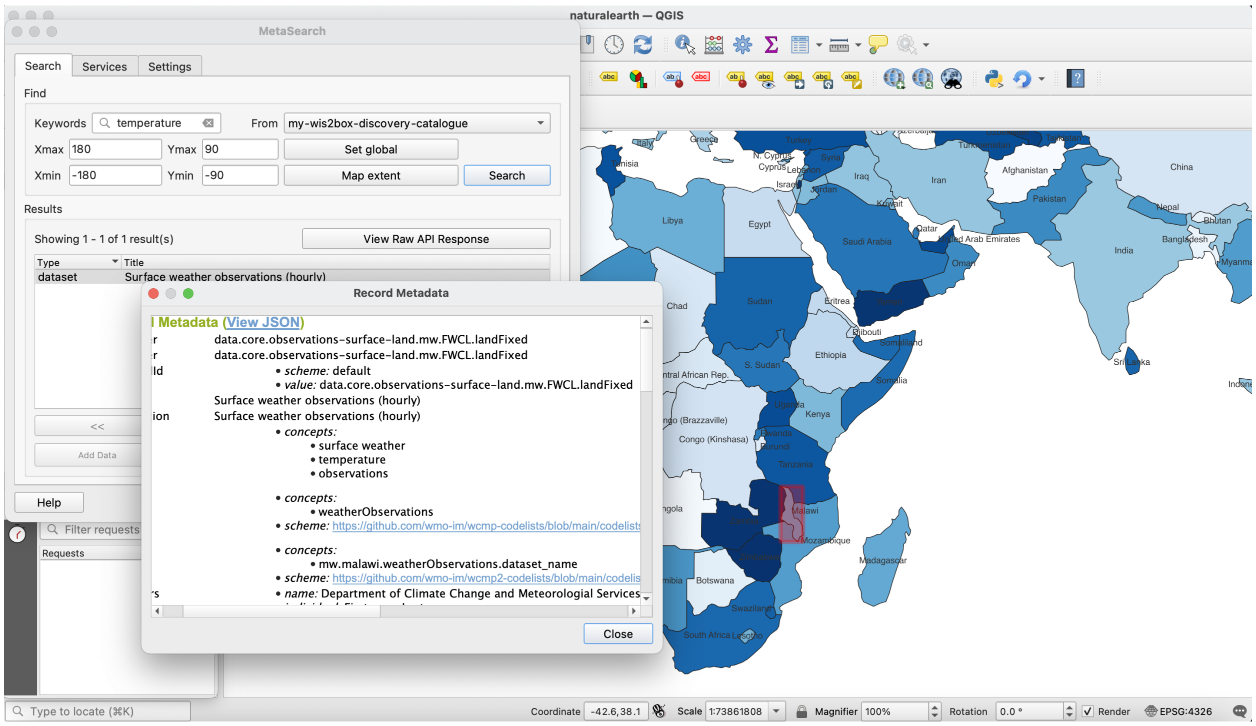
If the service requires basic authentication,enter a user name and optional password

User name

Password

This adds the discovery catalogue to the MetaSearch catalogue registry. Click "Service Info" to display the properties of the discovery catalogue service metadata.

To search the discovery catalogue, click the "Search" tab, which will provide the ability to search for metadata records by bounding box and/or full text search. Click the "Search" button to search the discovery catalogue and visualize search results. Clicking on metadata records in the search result table will show footprints on the map to help provide the location of the search result. Double-clicking a search result will show the entire metadata record.



Keywords temperature From my-wis2box-discovery-catalogue

Xmax 180 Ymax 90

Xmin -180 Ymin -90

Showing 1 - 1 of 1 result(s)

Type	Title
dataset	Surface weather observations (hourly)

l Metadata (View JSON)

r data.core.observations-surface-land.mw.FWCL.landFixed
r data.core.observations-surface-land.mw.FWCL.landFixed
ld
- *scheme:* default
- *value:* data.core.observations-surface-land.mw.FWCL.landFixed

Surface weather observations (hourly)
ion Surface weather observations (hourly)
- *concepts:*
  - surface weather
  - temperature
  - observations
- *concepts:*
  - weatherObservations
- *scheme:* https://github.com/wmo-im/wcmp-codelists/blob/main/codelists
- *concepts:*
  - mw.malawi.weatherObservations.dataset_name
- *scheme:* https://github.com/wmo-im/wcmp2-codelists/blob/main/codelis

rs
- *name:* Department of Climate Change and Meteorologial Services

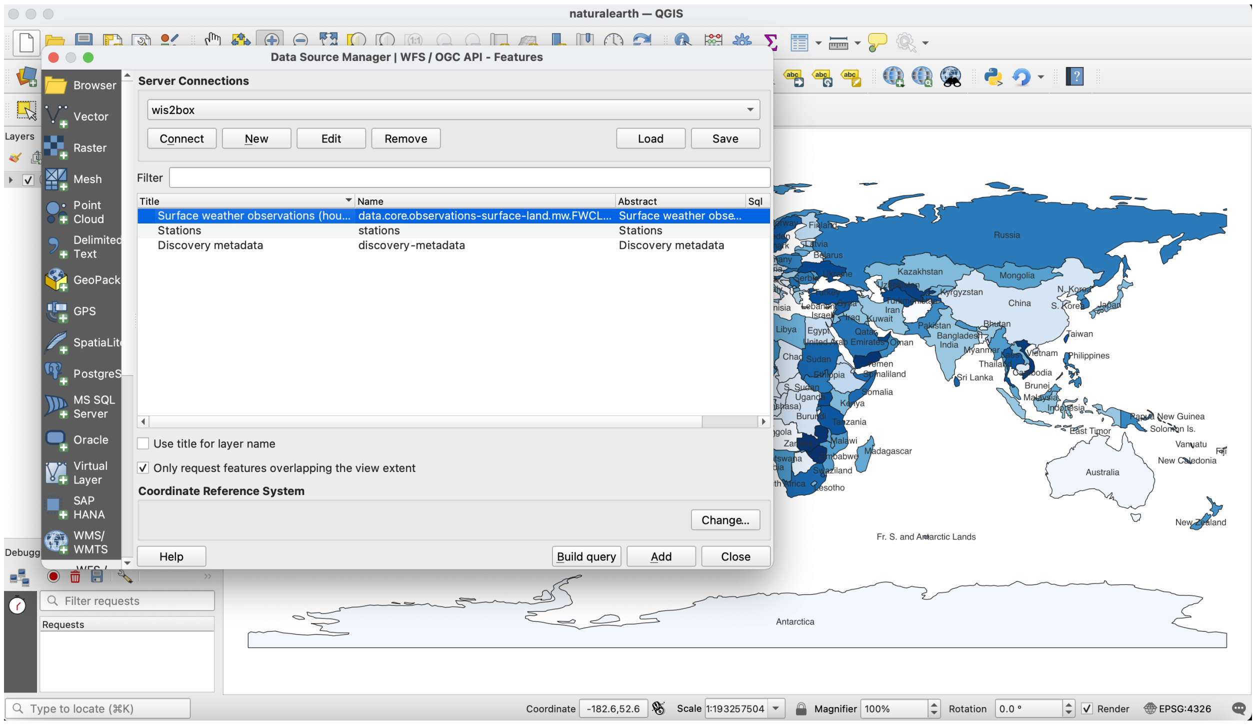
Coordinate -42.6,38.1 Scale 1:73861808 Magnifier 100% Rotation 0.0 ° Render EPSG:4326

Note: For more information on working with catalogues, consult the official QGIS MetaSearch documentation.

Visualizing stations

QGIS provides support for the OGC API - Features standard (access). To interact with the wis2box API:

- from the QGIS menu, select *Layer -> Add Layer -> Add WFS Layer…*
- click "New"
- enter a name for the API endpoint
- enter the URL to the API endpoint (i.e. `http://localhost/oapi`)
- under "WFS Options", set "Version" to "OGC API - Features"
- click "OK"
- click "Connect"



Title	Name	Abstract	Sql
Surface weather observations (hou…	data.core.observations-surface-land.mw.FWCL…	Surface weather obse…	
Stations	stations	Stations	
Discovery metadata	discovery-metadata	Discovery metadata	

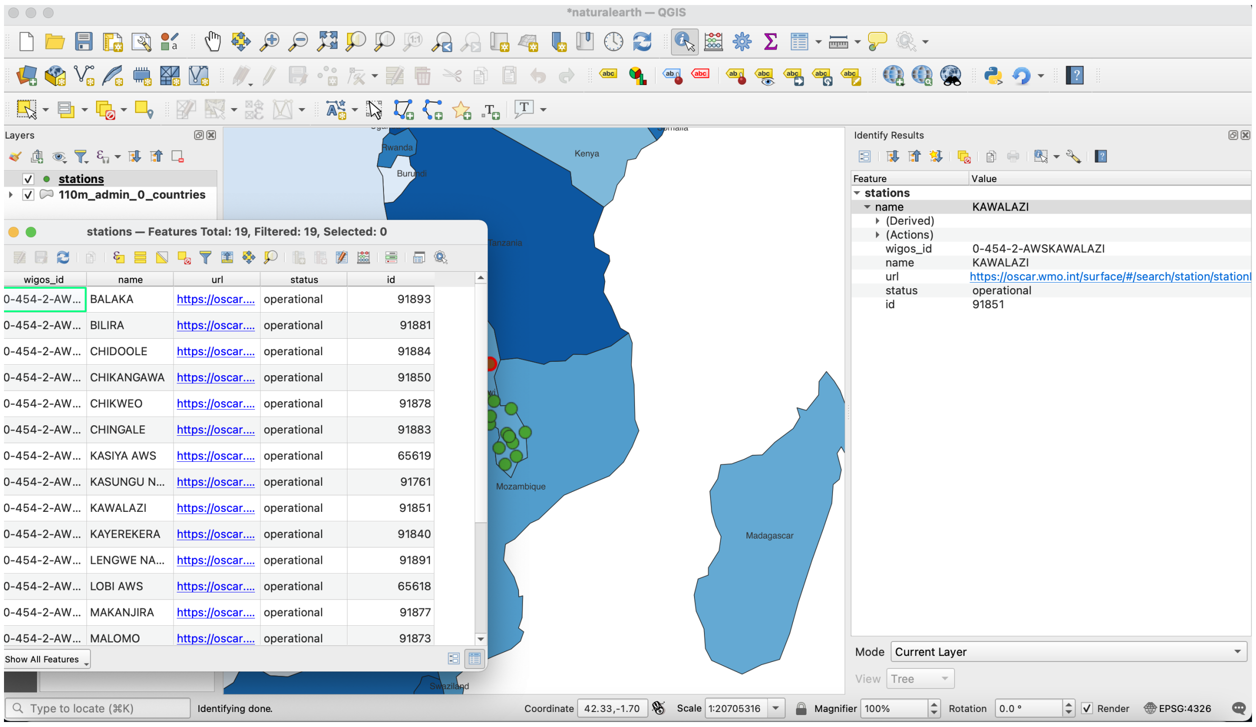
☐ Use title for layer name

☑ Only request features overlapping the view extent

Coordinate Reference System

A list of collections is displayed. Select the "Stations" collection and click "Add". The Stations collection is now added to the map. To further explore:

- click on the "Identify" (i) and click on a station to display station properties
- select *Layer -> Open Attribute Table* to open all stations in a tabular view



stations — Features Total: 19, Filtered: 19, Selected: 0

wigos_id	name	url	status	id
0-454-2-AW...	BALAKA	https://oscar....	operational	91893
0-454-2-AW...	BILIRA	https://oscar....	operational	91881
0-454-2-AW...	CHIDOOLE	https://oscar....	operational	91884
0-454-2-AW...	CHIKANGAWA	https://oscar....	operational	91850
0-454-2-AW...	CHIKWEO	https://oscar....	operational	91878
0-454-2-AW...	CHINGALE	https://oscar....	operational	91883
0-454-2-AW...	KASIYA AWS	https://oscar....	operational	65619
0-454-2-AW...	KASUNGU N...	https://oscar....	operational	91761
0-454-2-AW...	KAWALAZI	https://oscar....	operational	91851
0-454-2-AW...	KAYEREKERA	https://oscar....	operational	91840
0-454-2-AW...	LENGWE NA...	https://oscar....	operational	91891
0-454-2-AW...	LOBI AWS	https://oscar....	operational	65618
0-454-2-AW...	MAKANJIRA	https://oscar....	operational	91877
0-454-2-AW...	MALOMO	https://oscar....	operational	91873

Identify Results

Feature	Value
stations	
name	KAWALAZI
(Derived)	
(Actions)	
wigos_id	0-454-2-AWSKAWALAZI
name	KAWALAZI
url	https://oscar.wmo.int/surface/#/search/station/stationI
status	operational
id	91851

Note that the same QGIS workflow can be executed for any other collection listed from wis2box API.

Note: For more information on working with OGC API - Features, consult the official QGIS WFS documentation.

Summary

The above examples provide a number of ways to utilize the wis2box API from the QGIS desktop GIS application.

2.11.3 Pub/Sub

Using Python and paho-mqtt

This example will use widely available and used Python language and libraries to download some announcements, and then retrieve the corresponding data, using only the paho-mqtt client library, in addition to Python standard libraries.

[1]:
```python
import json
import paho.mqtt.client as mqtt
import random
import urllib
import urllib.request


host='localhost'
user='wis2box'
password='wis2box'

r = random.Random()
```

(continues on next page)

(continued from previous page)

```
clientId='MyQueueName'+ f"{r.randint(1,1000):04d}"
# number of messages to subscribe to.
messageCount = 0
messageCountMaximum = 5

# maximum size of data download to print.
sizeMaximumThreshold = 1023
```

The above imports the required modules. It is also assumed that `localhost` is set up and is publishing messages. Message queueing protocols provide real-time notification about availability of products.

The standard Python package used to subscribe to messages is `paho-mqtt` (`paho.mqtt.client`). The package uses callbacks.

Note that `messageCount` is used to limit the length of the demonstration (otherwise infinite, as it is a continuous flow).

Let's investigate our callbacks.

```
[2]: def sub_connect(client, userdata, flags, rc, properties=None):
         print("on connection to subscribe: ", mqtt.connack_string(rc))
         for s in ["origin/#"]:
             client.subscribe(s, qos=1)
```

The `sub_connect` callback needed is called when the connection is established, which required to subscribe to topics we are interested in (topics are: `origin/#`, where `/` is a topic separator and `#` is a wildcard for any tree of topics.

The `qos=1` refers to Quality of Service, where 1 establishes reception of messages at least once. `qos=1` is recommended.

The next callback is called every time a message is received, and decodes and prints the message.

To keep the output short for the demonstration, we limit the subscriber to a few messages.

```
[3]: def sub_message(client, userdata, msg):
         """
         print messages received.  Exit on count received.
         """

         global messageCount,messageCountMaximum

         m = json.loads(msg.payload.decode('utf-8'))

         print(f"message {messageCount} topic: {msg.topic} received: {m}")
         print(f"message {messageCount} data: {getData(m)}")

         messageCount += 1

         if messageCount > messageCountMaximum:
             client.disconnect()
             client.loop_stop()
```

The message handler above calls the `getData()` (below). The messages themselves are usually announcements of data availability, but when data is small, they can include the data itself (inline) in the `content` field. Usually the message refers to the data using a link. Here is a routine to obtain the data given an announcement message:

```
[4]: def getData(m, sizeMaximum=1000):
         """
```

(continues on next page)

(continued from previous page)

```
    given a message, return the data it refers to
    """

    if 'size' in m and m['size'] > sizeMaximum:
        return f" data too large {m['size']} bytes"
    elif 'content' in m:
        if m['content']['encoding'] == 'base64':
            return b64decode(m['content']['value'])
        else:
            return m['content']['value'].encode('utf-8')
    else:
        url = m['baseUrl'] + '/' + m['relPath']
        with urllib.request.urlopen(url) as response:
            return response.read()
```

The calling code then registers the callbacks, connects to the broker, and starts the event loop:

[ ]:
```
client = mqtt.Client(client_id=clientId, protocol=mqtt.MQTTv5)
client.on_connect = sub_connect
client.on_message = sub_message
client.username_pw_set(user, password)
client.connect(host)

client.loop_forever()
```

```
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
```

(continues on next page)

(continued from previous page)

```
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
on connection to subscribe:  Connection Accepted.
```

```
[ ]:
```

2.11.4 Running the examples

To be able to run these examples, one needs to start up a Jupyter Notebook environment. Below is an example of starting a Jupyter session:

```
git clone https://github.com/wmo-im/wis2box.git
cd docs/source/data-access
jupyter notebook --ip=0.0.0.0 --port=8888
```

When Jupyter starts up it may open a browser window for you. If not you would need to to point a browser at http://localhost:8888 to see the menu of notebooks available in this directory.

2.11.5 Summary

The above examples provide a number of ways to utilize the wis2box suite of services.

2.12 Development

wis2box is developed as a free and open source project on GitHub. The wis2box codebase can be found at https://github.com/wmo-im/wis2box.

2.12.1 GitHub

wis2box can be installed using the git CLI as follows:

```
# clone wis2box GitHub repository
git clone https://github.com/wmo-im/wis2box.git
cd wis2box
```

2.12.2 Testing

wis2box continuous integration (CI) testing is managed by GitHub Actions. All commits and pull requests to wis2box trigger continuous integration (CI) testing on GitHub Actions.

GitHub Actions invokes functional testing as well as integration testing to ensure regressions.

Integration testing

Integration tests are in `tests/integration/integration.py`.

Functional testing

Functional tests are defined as part of GitHub Actions in `.github/workflows/tests-docker.yml`.

2.12.3 Versioning

wis2box follows the Semantic Versioning Specification (SemVer).

2.12.4 Code Conventions

Python code follows PEP8 coding conventions.

2.13 Extending wis2box

At its core, wis2box is a plugin architecture orchestrating all the required components of a node in the WIS2 network. Driven by topic hierarchies, wis2box can be used to process and publish any type of geospatial data beyond the requirements of the WIS2 itself.

In this section we will to explore how wis2box can be extended. wis2box plugin development requires knowledge of how to program in Python as well as Python's packaging and module system.

2.13.1 Building your own data plugin

The heart of a wis2box data plugin is driven from the `wis2box.data.base` abstract base class (ABC) located in `wis2box/data/base.py`. Any wis2box plugin needs to inherit from `wis2box.data.base.BaseAbstractData`. A minimal example can be found below:

```python
from datetime import datetime
from wis2box.data.base import BaseAbstractData

class MyCoolData(BaseAbstractData):
    """Observation data"""
    def __init__(self, defs: dict) -> None:
        super().__init__(defs)

    def transform(self, input_data: Path) -> bool:
        # transform data
        # populate self.output_data with a dict as per:
        self.output_data = {
            'c123': {
                '_meta': {
                    'identifier': 'c123',
                    'relative_filepath': '/path/to/item/',
                    'data_date': datetime_object
                },
                'bufr4': bytes(12356),
                'geojson': geojson_string
            }
        }
        return True
```

The key function that plugin needs to implement is the `transform` function. This function should return a `True` or `False` of the result of the processing, as well as populate the `output_data` property.

The `output_data` property is a `dict` of keys/values. Each key should be the identifier of the item, with the following values `dict`:

The `_meta` element can include the following:

- `identifier`: identifier for report (WIGOS_<WSI>_<ISO8601>)
- `relative_filepath`: path to data, required to publish data with `BaseAbstractData.publish`
- `geometry`: GeoJSON geometry object, required to send geometry with WIS2 notification
- `md5`: md5 checksum of encoded data
- `wigos_station_identifier`: WIGOS identifier
- `data_date`: (as Python datetime objects) based on the observed datetime
- `originating_centre`: Originating centre (see Common code table C11)
- `data_category`: Category of data, see BUFR Table A

- `<format-extension>`: 1..n properties for each format representation, with the key being the filename extension. The value of this property can be a string or bytes, depending on whether the underlying data is ASCII or binary, for example

2.13.2 Packaging

The next step is assembling your plugin using standard Python packaging. All plugin code and configuration files should be made part of the package so that it can operate independently when running in wis2box. For distribution and installation, you have the following options:

- publish to the Python Package Index (PyPI) and install in the wis2node container with `pip3 install wis2box-mypackage`
- `git clone` or download your package, and install via `python3 setup.py install`

See the Python packaging tutorial or Cookiecutter PyPackage for guidance and templates/examples.

Note: It is recommended to name your wis2box packages with the convention `wis2box-MYPLUGIN-NAME`, as well as adding the keywords/topics `wis2box` and `plugin` to help discovery on platforms such as GitHub.

2.13.3 Integration

Once your package is installed on the wis2box-management container, the data mappings need to be updated to connect your plugin to a topic hierarchy. See *Data mappings* for more information.

An example plugin for proof of concept can be found in https://github.com/wmo-cop/wis2box-csv-observations

2.13.4 Example plugins

The following plugins provide useful examples of wis2box plugins implemented by downstream applications.

Plugin(s)	Organization/Project	Description
wis2box-csv-observations	WMO	plugin for CSV surface observation data
wis2box-pyopencdms-plugin	OpenCDMS	plugin for connecting the Open Climate Data Management System to wis2box

CHAPTER THREE

COMMUNITY

The community documentation provides information on where to find support and how to contribute to wis2box.

3.1 Support

Please consult the wis2box Discussions for support with the project.

3.2 Troubleshooting

This page lists several commonly seen issues and how to address them.

3.2.1 './docker-compose.yml' is invalid

When starting wis2box you see the errors:

```
ERROR: The Compose file './docker-compose.yml' is invalid because:
Unsupported config option for volumes: 'auth-data'
Unsupported config option for services: 'wis2box-auth'
```

check the version of docker-compose you are running with:

```
docker-compose --version
```

if not 1.29.2 you can install this using the following docker-compose :

```
# download docker-compose 1.29.2
sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-
↪$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
# set executable
sudo chmod +x /usr/local/bin/docker-compose
# remove current version
sudo rm /usr/bin/docker-compose
# set link to downloaded version
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

3.2.2 OSError: Missing data mappings

The wis2box logging displays the error:

```
OSError: Missing data mappings: [Errno 2] No such file or directory: '/data/wis2box/data-
↪mappings.yml'
```

Check your dev.env and check value that was set for WIS2BOX_HOST_DATADIR

```
WIS2BOX_HOST_DATADIR=/home/wmouser/wis2box-data
```

In this case the value set was '/home/wmouser/wis2box-data'

Check that the file 'data-mappings.yml' is contained in this directory:

```
ls -lh /home/wmouser/wis2box-data/data-mappings.yml
```

After you have ensured the data-mappings.yml is in the directory defined by WIS2BOX_HOST_DATADIR, restart the wis2box:

```
python3 wis2box-ctl.py stop
python3 wis2box-ctl.py start
```

3.2.3 Topic Hierarchy validation error: Unknown file type

Check your `data-mappings.yml` file to adjust the file extension expected by the plugins processing your dataset.

If you are ingesting files with extension .bin:

```
plugins:
    bin:
        - plugin: wis2box.data.bufr4.ObservationDataBUFR
          notify: true
          buckets:
            - ${WIS2BOX_STORAGE_INCOMING}
          file-pattern: '*'
```

If you are ingesting files with extension `.b`:

```
plugins:
    b:
        - plugin: wis2box.data.bufr4.ObservationDataBUFR
          notify: true
          buckets:
            - ${WIS2BOX_STORAGE_INCOMING}
          file-pattern: '*'
```

3.2.4 The Access Key Id you provided does not exist in our records

If you see this error when uploading data to the wis2box-incoming storage, you have provided the wrong username and/or password to access MinIO. Check the values for `WIS2BOX_BROKER_USERNAME` and `WIS2BOX_BROKER_PASSWORD` you have provided in your `dev.env` file. The default username/password for MinIO is `minio/minio123`.

3.2.5 Topic Hierarchy validation error: No plugins for ... in data mappings

A file arrived a folder for which no matching dataset was defined in your `data-mappings.yml`.

For dataset `foo.bar`, store your file in the path `/foo/bar/`.

This requires either updating `data-mappings.yml` or changing the target folder under which the file is received.

3.2.6 ERROR - Failed to publish, wsi: ..., tsi: XXXXX

Data arrived for a station that is not present in the station metadata cache. To add missing stations, update the file `metadata/station/station_list.csv` in the wis2box data directory (see *Installation and configuration*).

3.2.7 Error: no such container: wis2box-management

If the wis2box-management container is not running the 'login' command will fail. The wis2box-management container depends on other services being available before it can successfully started.

Please check all services are Running using the following command:

```
python3 wis2box-ctl.py status
```

Possible issues are:

- port 80 is already in use, the nginx-service will fail to start if there is already a web-server running on your instance
- WIS2BOX_STORAGE_PASSWORD is too short, minio will fail to start if you specify a WIS2BOX_STORAGE_PASSWORD of less than 8 characters

3.2.8 wisbox-UI is empty

If when you access the wis2box-UI you see the interface but no datasets are visible, check the WIS2BOX_URL and WIS2BOX_API_URL are set correctly.

Please note that after changing the WIS2BOX_URL and WIS2BOX_API_URL, you will have to restart your wis2box:

```
python3 wis2box-ctl.py stop
python3 wis2box-ctl.py start
```

And repeat the commands for adding your dataset and publishing your metadata, to ensure the URLs are updated in the records:

```
python3 wis2box-ctl.py login
wis2box data add-collection ${WIS2BOX_HOST_DATADIR}/surface-weather-observations.yml
wis2box metadata discovery publish ${WIS2BOX_HOST_DATADIR}/surface-weather-observations.
↪yml
```

3.3 Contributing

wis2box is developed as a free and open source project on GitHub. Contributions to the project (documentation, bug fixes, enhancements, tests, etc.) are welcome and encouraged. Please consult the wis2box Contribution guidelines for more information.

3.4 License

3.4.1 Software

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

```
      (a) You must give any other recipients of the Work or
          Derivative Works a copy of this License; and

      (b) You must cause any modified files to carry prominent notices
          stating that You changed the files; and

      (c) You must retain, in the Source form of any Derivative Works
          that You distribute, all copyright, patent, trademark, and
          attribution notices from the Source form of the Work,
          excluding those notices that do not pertain to any part of
          the Derivative Works; and

      (d) If the Work includes a "NOTICE" text file as part of its
          distribution, then any Derivative Works that You distribute must
          include a readable copy of the attribution notices contained
          within such NOTICE file, excluding those notices that do not
          pertain to any part of the Derivative Works, in at least one
          of the following places: within a NOTICE text file distributed
          as part of the Derivative Works; within the Source form or
          documentation, if provided along with the Derivative Works; or,
          within a display generated by the Derivative Works, if and
          wherever such third-party notices normally appear. The contents
          of the NOTICE file are for informational purposes only and
          do not modify the License. You may add Your own attribution
          notices within Derivative Works that You distribute, alongside
          or as an addendum to the NOTICE text from the Work, provided
          that such additional attribution notices cannot be construed
          as modifying the License.

      You may add Your own copyright statement to Your modifications and
      may provide additional or different license terms and conditions
      for use, reproduction, or distribution of Your modifications, or
      for any such Derivative Works as a whole, provided Your use,
      reproduction, and distribution of the Work otherwise complies with
      the conditions stated in this License.

   5. Submission of Contributions. Unless You explicitly state otherwise,
      any Contribution intentionally submitted for inclusion in the Work
      by You to the Licensor shall be under the terms and conditions of
      this License, without any additional terms or conditions.
      Notwithstanding the above, nothing herein shall supersede or modify
      the terms of any separate license agreement you may have executed
      with Licensor regarding such Contributions.

   6. Trademarks. This License does not grant permission to use the trade
      names, trademarks, service marks, or product names of the Licensor,
      except as required for reasonable and customary use in describing the
      origin of the Work and reproducing the content of the NOTICE file.

   7. Disclaimer of Warranty. Unless required by applicable law or
      agreed to in writing, Licensor provides the Work (and each
```

(continues on next page)

(continued from previous page)

(continues on next page)

(continued from previous page)

3.4.2 Documentation

The documentation is released under the Creative Commons Attribution 4.0 International (CC BY 4.0) license.