nzodn

Release 0.0.1

Jesse Wood, Susi Woelz, Andrea Mari

Apr 05, 2022

CONTENTS:

CHAPTER

ONE

ABOUT

Ingesting data into the New Zealand Ocean Data Network (aka. NZODN).

The data comes from various proprietary instruments in the field. So there is little standard or consistency across the format that the data may come. This makes complete automation of the process difficult. However, new data that need to be ingested may have similar formats to existing data sets. We can glean inspiration for how to approach new datasets, from the existing datasets on the NZODN.

This project occurs on an ongoing basis, as a State-Owned Enterprise (SOE) and Research Company, Niwa has an obligation to publish a certain quota of their research to the public. The process for doing so needs to be well documented and maintained, such that it can easily be replicated by others in the future.

CHAPTER

TWO

DATASETS

We provide a description of each dataset that we ingested into the NZODN database.

2.1 Benthic I - Chattham Sound

The benthic zone is the ecological region at the lowest level of a body of water such as an ocean, lake, or stream, including the sediment surface and some sub-surface layers. (source)

Biodiversity is the variety and variability of life on Earth. Biodiversity is typically a measure of variation at the genetic, species, and ecosystem level. Terrestrial biodiversity is usually greater near the equator, which is the result of the warm climate and high primary productivity. (source)

Notes:

- benthic biodiversity relates to species and their habitat
- statistical modeling on the likelihood of a taxon being in its habitat
- zip file
- display bounding box only on the map
- bounding box crosses the 180 degree line

Links:

- GeoNetwork
- NZODN Portal

2.2 Benthic II - Chatham Sound and Campbell Plateau

See Benthic I for more information.

Details:

- See Benthic I.

Links:

- Geonetwork TODO
- NZODN Portal

2.3 Chemical Analysis

The New Zealand offshore seabed hosts diverse resources including phosphate rich rocks. Phosphate rock deposits on the Chatham Rise have been the focus of previous investigations into their composition and mining potential; however, the diversity of the geochemistry of phosphate deposits, including their wider distribution beyond the Chatham Rise, their trace metal budget, and potential for ecotoxicity, remain poorly characterised. This study addresses some of these gaps by presenting a geochemical investigation, including trace metals, for a range of phosphate nodules from across the Chatham Rise, Bollons Seamount and offshore southeastern South Island. Elutriate and reconnaissance bioaccumulation experiments provide insights into the potential for ecotoxic trace metal release and effects on biota should sediment disturbance through mining activities occur. (source)

Details:

- NZ Phophorite Geochemistry readings.
- The data is an XSLX file that has been transposed into a CSV file.
- Each row contains a station, coordinates and its respective measurements.

Links:

- CSV file
- Geonetwork

2.4 CTD

A CTD device's primary function is to detect how the conductivity and temperature of the water column changes relative to depth. Conductivity is a measure of how well a solution conducts electricity and it is directly related to salinity. By measuring the conductivity of seawater, the salinity can be derived from the temperature and pressure of the same water. The depth is then derived from the pressure measurement by calculating the density of water from the temperature and the salinity. (source)

Notes:

- Conductivity, Temperature and Depth
- a proxy for the salinity of water
- this is generated on a on-going basis
- uses cron jobs to ingest the data each day

Links:

- TODO

2.5 Oceanographic

Notes:

- TODO

Links:

- Geonetwork
- NZODN Portal TODO

CHAPTER

THREE

TECHNOLOGY

Here we provide an overview with code examples for the technologies used in the NZODN database. These examples cover the use cases encountered when ingesting data into the database. This section does not provide an exhaustive documentation of NZODN internals, only what the ingestor need worry about. Working examples, and links to further documentation, are provided for each technology to give a user intuition for how to use to ingest data.

3.1 Cron

The software utility cron was developed at AT&T and Bell Labs. It is often referred to as a cron job. This utility provides a time-based scheduler on Linux systems. It can be used to run a script at specific times, repeatedly. For example, we could check for updates on the for the *apt* package _once_ a week [1].

Cron is most suitable for scheduling repetitive tasks. In our case, we ingest data from the moorings once a day. We need to check the central databases, which contains mooring data and update the existing database on NZODN to include the new record.

3.1.1 crontab

The frequency at which a cronjob is run depends on the *crontab* (cron table) file. This is a configuration file that specifies scripts to be run on a schedule. All the *crontab* files are stored in a directory where the cron *daemon* is kept. Users can have their own crontab files, and there is usually a system-wide contrab file located in the */etc* or a subdirectory of */etc*. This file can only be edited by system administrators [1].

```
# ┌──────────── minute (0 - 59)
# │ ┌──────────── hour (0 - 23)
# │ │ ┌──────────── day of the month (1 - 31)
# │ │ │ ┌──────────── month (1 - 12)
# │ │ │ │ ┌──────────── day of the week (0 - 6) (Sunday to Saturday;
# │ │ │ │ │                                       7 is also Sunday on some systems)
# │ │ │ │ │
# │ │ │ │ │
# * * * * * <command to execute>
```

Each line of a crontab starts with five fields. These fields are used to represent the time schedule for the command to be run.

Online tools exist to easily calculate this seemingly cryptic language, much like regex, sometimes recall vs. recognition is important, we must recognise this is a niche skill and outsource knowledge when convenient.

3.1.2 Examples

The example below clears the Apache error log at one minute past midnight (00:01) every day.

```
$ 1 0 * * * printf "" > /var/log/apache/error_log
```

This example runs a shell script called export_dump.sh at 23:45 (11:45 pm) every Saturday.

```
$ 45 23 * * 6 /home/oracle/scripts/export_dump.sh
```

Note: It is also possible to specify */n* t run for every _n_-th interval of time.

The output below would write "hello world" to standard output every 5th minute of every first, second, and third hour (i.e., 01:00, 01:05, 01:10, up until 03:55).

```
$ */5 1,2,3 * * * echo hello world
```

3.1.3 Marcos

Some cron implementations support non-standard macro definitions. For example, *@reboot* configures a job to run once the daemon is started. Cron is rarely restarted, so this macro corresponds to when the machine is rebooted. In some Debian distributions this behaviour is enforced, where restarting cron will not run the *@reboot* jobs.

Some other examples include

|Entry|Description|Equivalent to| |---|—|---| |`@yearly (or @annually)`|Run once a year at midnight of 1 January|`0 0 1 1 *`| |`@monthly`|Run once a month at midnight of the first day of the month|`0 0 1 * *`| |`@weekly`|Run once a week at midnight on Sunday morning|`0 0 * * 0`| |`@daily (or @midnight)`|Run once a day at midnight|`0 0 * * *`| |`@hourly`|Run once an hour at the beginning of the hour|`0 * * * *`| |`@reboot`|Run at startup| `na`|

3.1.4 Permission

There are two files that affect the permissions for executing cron jobs.

- */etc/cron.allow* - if file exists, it contains a list of users allowed to use cron jobs
- */etc/cron.deny* - if *cron.allow* does not exist, this becomes a blacklist, all other users are permitted

If neither of these files exists, either only the superuser can use cron jobs, or every user can.

Tutorial

We can edit CRON jobs for our user using this command:

```
$ crontab -e
```

Take the cronjob below, which appends the string "Hello World" to a log file in our */tmp* directory, once a every minute.

We then wait a couple of minutes for these messages to build up. Then check the log file.

```
$ cat /tmp/test.log
```

After 4 minutes have passed, the output should be something similar to this:

```
Hello World
Hello World
Hello World
Hello World
```

This verifies that our cronjob is working for our user.

Finally, to stop the cronjob from running forever, we execute the command below. This clears all of the cronjobs for the current user.

```
$ crontab -r
```

3.1.5 Data Flow

This is the flow of the data for research projects such as CTD and Mooring data. The information is recorded on a regular interval by sensors, for example, a Mooring. This is stored on a database on a local machine. That is then transferred to Niwa's central database, which aggregates all the field data into a single place. We then construct a cronjob, which ensures the NZODN Postgres database is updated each day, with the most recent data.

3.1.6 NZODN Example

This is the crontab from the NZODN server. This is on the crontab of the root user of Wellimos.

```
#Schedule a daily import for mooring data. Get don't expect new data
#most days, but when there is data we want it imported as soon as possible
2 10 * * * robot /home/robot/aodn/import/mooring/bin/loadAllMooringData.sh -c /home/
↪robot/aodn/import/mooring/config/prod.ini >> /tmp/nzodn-mooring-import.log
```

It runs everyday, at 2 minutes past 10. The 2 minute offset is to ensure that not all cron jobs are run at the same time on a server. That would put unnecessary strain on the existing server, and require additional compute for a very short amount of time, and no need for that power the rest of the time.

The crontab executes a bash script with the *robot* user. This is a service account used by NZODN for automated processes. It has permission to read and write to directories, such as the */data/niwa/publish* that exposes files to NZODN FTP server.

This script takes in a configuration file *prod.ini* which specifies that parameters to the bash script *loadAllMooring-Data.sh*. Using a configuration file makes avoids hard-coding, and makes this script re-usable in the future for other purposes. A non-technical user can easily adjust the configuration, without needing to have an understanding of the implementation details of the program. The appendix gives an example a configuration file.

The output of the bash script is appended to a log file */tmp/nzodn-mooring-import.log*. Such that any runtime errors, executions, or unintended side affects can be reviewed in future. We use the *tmp* directory, such that we don't keep erroneous log files that are no longer relevant for extended amounts of time. These would be a waste of disk space. We use the append command >>, since we do not want to overwrite the existing file entirely each time an ingestion is performed. In the event of a catastrophic error, we would have four days to review the log files.

3.1.7 Configuration File example

This is an example of the kind of configuration file *prod.ini* which is passed as an argument to our bash script that loads the mooring data. Sensitive details are obfuscated for obvious security reasons.

3.1.8 Log Files

By running *cat /tmp/nzodn-mooring-import.log* we can see the contents of the ingestion log files:

```
[2021/02/08 10:02:01] INFO    : Fetching data
[2021/02/09 10:02:01] INFO    : Fetching data
[2021/02/10 10:02:02] INFO    : Fetching data
[2021/02/11 10:02:01] INFO    : Fetching data
```

Here we see that the ingestion process has been completed for the past consecutive four days, with no further previous records. This is because the */tmp* directory is cleared every four days (I assume).

3.2 FTP

FTP is the standard network protocol that is used to transfer files from a server to a client. It is not considered a secure form of communication, since although it can be configured to use a username and password for each client, this is vulnerable to brute force dictionary and social engineering attacks.

3.3 GeoNetwork

GeoNetwork is a catalogue application for managing spatially referenced resources. It stores the metadata for each data set we see on NZODN. This includes a variety of information, such as copyrights, attributions, abstracts, etc… A metadata record on the GeoNetwork points to a GeoServer WMS.

3.4 GeoServer

GeoServer is an open-source tool developed using Java. It provides a user interface to create web map services that connect to databases, with interoperability with PostgreSQL. It publishes data from any major spatial data source using open standards published by the Open Geospatial Consortium. A map service can be accessed using many libraries, such as Leaflet.

3.5 Postgres

PostgreSQL is a free and open-source relational database management software. We use the Postgis extension to add support for spatial data to Postgres. This extension follows the SQL specification from the Open Geospatial Consortium. We use this table to store the geo-spatial data shown on the map and to store the file path and URL for the download data.

3.6 NZODN

NZODN, is a fork of the open-source AODN. This utilizes open-source tools with best practise OGC standards to create a web portal to freely access research data. It allows the research community to search for useful resources to further their own work. Including the ability to download subsets (i.e., temporal, geographical extents) of a larger dataset.

3.7 SFTP

SFTP, which stands for SSH File Transfer Protocol or Secure File Transfer Protocol. This is a secure version of FTP that uses **`SSH <https://en.wikipedia.org/wiki/SSH_(Secure_Shell)`_** to transfer files remotely. The IT department prefers, for the sake of security, to use the SFTP protocol instead, due to security vulnerabilities.

In order to use SSH, the users need to have a have generated an SSH key. The public key of the client needs to be on the list of known hosts for the server. If not, the client will not be able to authenticate their SSH connection with the server. In order to design for reuse, it is wise to create a service account with its own SSH keys, to perform the automated data ingestion, rather than exposing a the public key of an individual user.

The OpenSSH software package for Linux systems, allows us to perform all the necessary steps required to establish an SSH connection. Here is the tutorial for that process.

3.7.1 Generate SSH Keys

First we must generate an SSH key. This is stored in the hidden folder *.ssh* in the home directory *home/<username>* of your user.

```
ssh-keygen
```

Then we must copy our public key to the server. The copying may ask for a password.

```
$ ssh-copy-id -i ~/.ssh/mykey user@host
```

Now we can test our key has been created by trying to SSH to the server. This will automatically look in our *.ssh* folder if it exists, for our public key.

```
$ ssh user@host
```

We only ever send our public key to the server, we keep our private key a secret, and do not share it with anyone. Treat this like a password.

3.7.2 SFTP Linux Command

sftp is a command that can be used directly in the Linux terminal. [Tutorial

To connect to a SFTP server simply execute this:

```
$ sftp remote_username@server_ip_or_hostname
```

If the ssh keys have been generated correctly and copied to the server the user should not be prompted for a password.

Once connected to the server, it functions very similarly to the *ftp* command. Typing *help* will produce a list of all the possible commands and their description.

To download a file, say *example.txt* from the server to the client, execute the following command:

```
sftp> get example.txt
```

Then to close the SFTP connection the user must say good bye.

```
sftp> bye
```

3.7.3 WinSCP

TODO:

- Expand w screenshots.
- Use Paegent for ssh-rsa key.
- My SSH key works for NZODN.
- Use WinSCP to connect to the server.
- Visualization of directory structure.

3.8 SSH

SSH relies on a two-way handshake between the server and the client. The handshake is established using public and private key pairs. The server and the client share their public keys, and keep their private key a secret. The private key of the client, and the public key of the server, can be combined to form a send a connection request to the server. The server then uses its private key, and the clients public key, to authenticate the request. And so, a secure shell connection can be formed.

CHAPTER

FOUR

SUPPORT

The easiest way to get help with the project is to email the author directly jesse.wood@niwa.co.nz. The other good way is to open an issue on Github.

Documentation: https://psychic-invention.readthedocs.io/en/latest/

Github: https://github.com/woodRock/psychic-invention

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search