La Trobe University

HiveKeepers

Standalone Data Visualisation Platform

PREPARED FOR

HiveKeepers

PREPARED BY

Andrew McDonald

Supervised BY

Prof. Wei Xiang

Executive Summary

This project was designed and developed in partnership between La Trobe University and HiveKeepers as part of the La Trobe University industry-based learning internship.

Project Directory Structure and Files

```
project
├── container1
│   ├── docker-entrypoint.sh
│   ├── Dockerfile
│   ├── fail2ban                        <- fail2ban config dir
│   │   ├── action.d
│   │   │   └── docker-iptables-multiport.conf
│   │   ├── fail2ban.local
│   │   ├── filter.d
│   │   │   └── nginx-http-auth.conf
│   │   ├── jail.d
│   │   │   └── nginx.conf
│   │   └── jail.local
│   ├── fixed_envsubst-on-templates.sh
│   ├── healthcheck.sh
│   ├── monit                           <- monit watchdog config dir
│   │   ├── fail2ban.conf
│   │   ├── monitrc
│   │   └── nginx.conf
│   ├── nginx                           <- nginx proxy config dir
│   │   ├── default.old
│   │   ├── html
│   │   │   ├── background.jpg
│   │   │   └── index.html
│   │   ├── nginx.conf
│   │   └── templates
│   │       └── default.conf.template
│   └── password_script.sh
├── container2
│   ├── dash_app                        <- Python visual application config dir
│   │   ├── gunicorn_config.py
│   │   ├── hivekeepers_app.py
│   │   ├── hivekeepers_config.py
│   │   ├── hivekeepers_helpers.py
│   │   ├── requirements.txt
│   │   ├── start_app.sh
│   │   ├── startup_update_db.py
│   │   └── update_db.py
│   ├── docker-entrypoint.sh
│   ├── Dockerfile
│   ├── healthcheck.sh
│   └── monit                           <- monit watchdog config dir
│       ├── gunicorn3.conf
│       └── monitrc
├── docker-compose.yml
├── htpasswd                            <- default user:password file
├── README.md
└── scripts
    ├── password_script.sh              <- encrypted password script
    └── user_credentials.txt            <- example user:password file for script
```

1. Project Overview

Build and present a containerised website for presenting apiary data from remote MySQL server, with user authentication and IP banning services.

2. System Design

The system comprises of two distinct Docker containers, running on their own private container network. Each container is given a static IP address for reliable inter-container communication and referencing.

Container1 handles all incoming network requests to the container network, and proxies any permissible requests destined for container2 to its respective static IP address.

To handle incoming requests, container1 runs the NGINX service on port 80. To control access to the container network, NGINX has basic-auth turned on and references a user:password file to determine relevant access privileges.

To handle requests that fail NGINX basic-auth 5 times, container1 also runs the service Fail2ban. Fail2ban monitors the NGINX error log and records the IP address of failed access attempts to its log for future reference. Once an IP address has reached 5 failed attempts within a given time span (10 mins) the IP address is banned from future access for 10 minutes – the number of attempts. the time frame for attempts, and the ban time can all be configured within Fail2bans configuration file before container1 build time if desired.

Monit is used as the watchdog handler for monitoring the NGINX and Fail2ban services and restarts if either are found to be down/unresponsive.

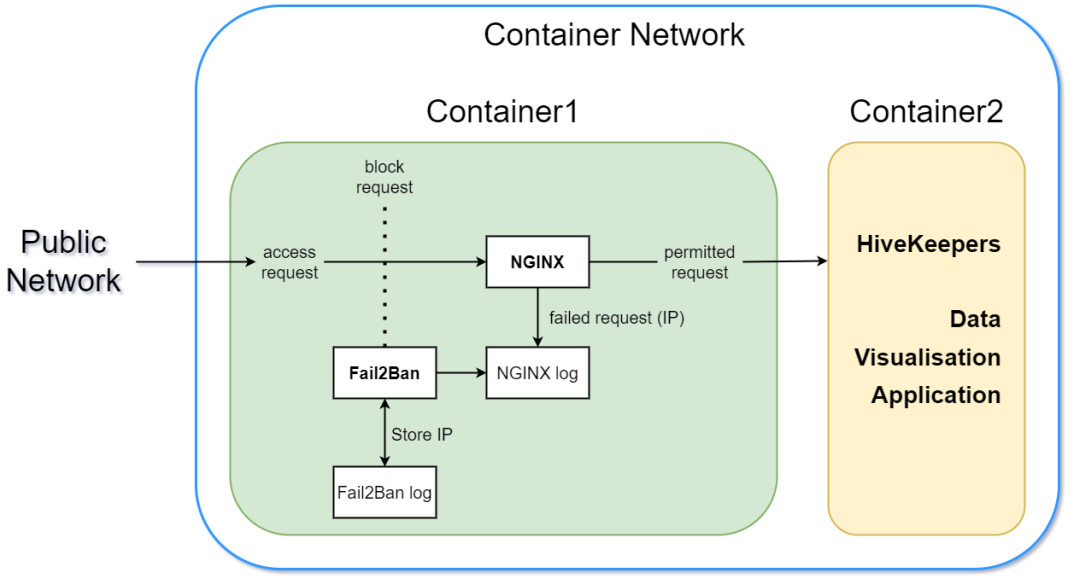


Fig1: Container1 process diagram

Container2 runs the HiveKeepers data visualisation web application, which displays 2d and 3d charts from timeseries data collected from apiaries. The application is written in Python and relies heavily on the Plotly Dash visualisation library. Web Server Gateway Interface (WSGI) Gunicorn is used to handle all web requests to and from the application. And data for visualising is pulled from the HiveKeepers remote MySQL database and stored locally in an SQLite database.

Container2 has no exposed ports and is only accessible from outside the container network via the container1 reverse proxy.

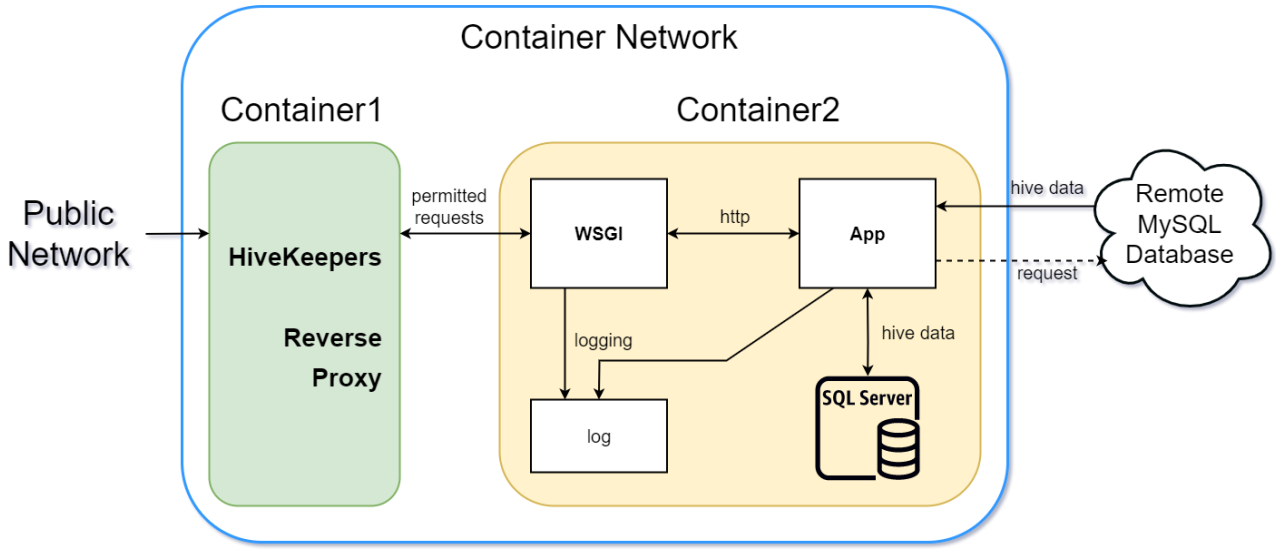


Fig2: Container2 process diagram

Both containers have in-built system logging, which is stored on a shared container volume, but can also be stored remotely via volume binding.

Both containers are free to send requests outside of their private container network.

3. Data Visualisation Application (HiveKeepers Dash App)

The data visualisation application is written using the Dash Plotly framework and consists of 4 plotted charts from data acquired from the HiveKeepers MySQL aviary database.

The app is built using the main hivekeeper_app.py for the central logic, and the building and displaying of charts. There are two separate database update scripts, one for initial start-up (startup_update_db.py) and one for updating incremental updates once a local database is in place.

There is also a helper script (hivekeepers_helpers.py) which houses the main functions for data handling (getting data from local SQLite db), data cleaning and data building for charts.

There is also a config file (hivekeepers_config.py) for storing relevant STATIC variables and the MySQL remote database credentials.

These files are all stored in project folder 'container1/dash_app/' and on the running container2 in folder '/home/hivekeeper/dash_app/'

Application Logic

Once the user has accessed the HiveKeepers Dash App, they are presented with an 'update database' button, a 'date selector' and an 'aviary selector' drop-down menu.

Depending on the environment 'start_type' chosen when starting the containers, the local database may or may not already be populated. If the container has been started using 'warm_start', then the user simply chooses an aviary from the drop down and the date selector will automatically choose the most recent single days' worth of data to represent in the charts. If the container was started with 'cold_start', then the user must update the database before the drop-down aviary selector will be populated.

Each of the 2d charts also have their own respective date range sliders, so that the user can move around and select specific subsets of days data for viewing.

The 3d and 4d charts have a further two drop down menus for affecting how they are represented. The first selects subsets of FFT-bin data; there are 64 FFT-bins in total, which have been divided into 4 groups of 16 and the drop down lets you choose between each group or to view the entirety at once.

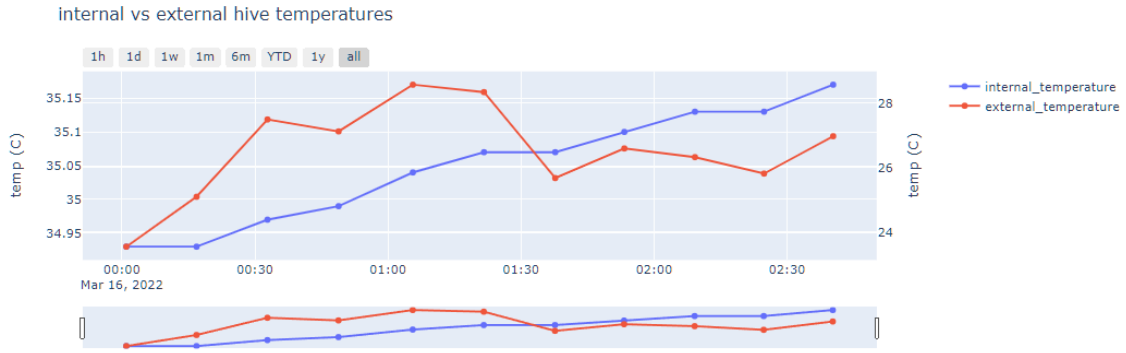
There is also a further drop-down menu that lets you choose alternative colour-scales to represent the c-axis, the fourth dimensional aspect of the charts.

All charts are fully interactive, letting you move between ranges, rotate them, and even download a copy of the graph that is currently set by the user.

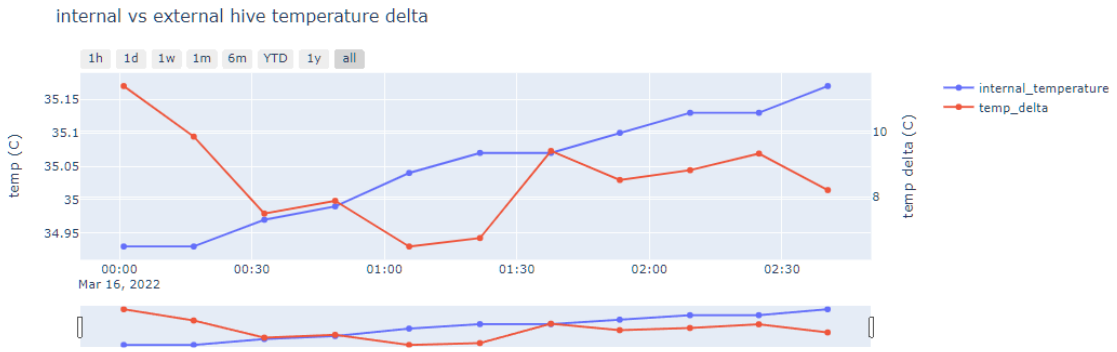
Application Layout

Webpage Header and Selectors

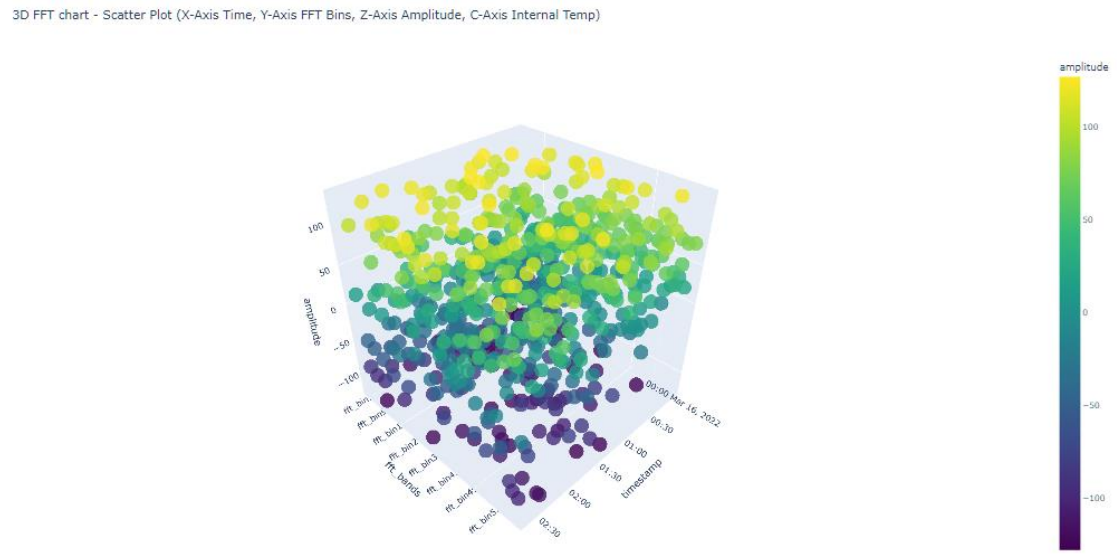
2D Chart (X-Axis Time, Y-Axis Internal Temperature, Y2-Axis External Temperature)



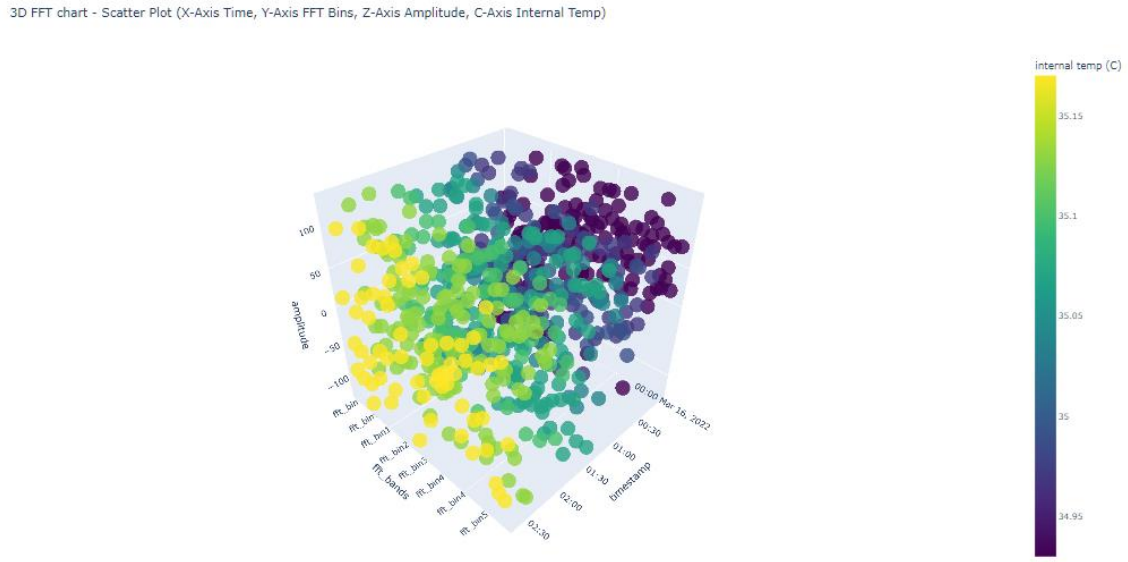
2D Chart (X-Axis Time, Y-Axis Internal Temperature, Y2-Axis “Internal – (difference) External Temperature”



3D FFT chart (X-Axis Time, Y-Axis FFT Bins, Z-Axis Amplitude)



3D FFT chart (X-Axis Time, Y-Axis FFT Bins, Z-Axis Amplitude, Colour Scale Hive Internal Temperature)



4. System Info

names:
container1: reverse-proxy
container2: dash-app

services:
container1: nginx, fail2ban, monit
container2: Dash (Python), Gunicorn, monit

container network:
subnet: 172.75.0.0/16
container1 IP: 172.75.0.2
container2 IP: 172.75.0.3

in container main working directory:
container2: /home/hivekeeper/

container persistent storage:
name: container_data:
stores: logs (both containers), database (container1)
container bind directory: /home/hivekeeper/persistent/

container1 persistent storage:
logs: /home/hivekeeper/persistent/logs/container1

container2 persistent storage:
logs: /home/hivekeeper/persistent/logs/container2
database: /home/hivekeeper/persistent/db/

Default Visualisation App Access (through proxy; can be changed/removed):
Username: hivekeepers
Password: hivekeepers

5. Software Versions

container1:

service	source	version
Docker base Image	Docker Hub (NGINX official)	nginx:1.20.2
NGINX	Baked into base image	1.20.2
Fail2ban	Debian repository	0.11.2
Monit	Debian repository	5.27.2

container2

service	source	version
Docker base Image	Docker Hub (Debian official)	debian:stable-slim
Python	Debian repository	3.9.2
SQLite3	Debian repository	3.34.1
Gunicorn3	Debian repository	20.1.0

6. Environment Variables

container1:

APP_PORT	INT	Port to proxy to on container2, must match in both containers
PROXY_LOG_LEVEL	STRING	options: simple (no nginx access logging), detailed (with nginx access logging)
NGINX_ERROR_LOG_LEVEL	STRING	options: info, notice, warn, error, crit, alert, emerg (case sensitive)

container2

MYSQL_USER	STRING	username for remote MySQL DB
MYSQL_PASS	STRING	password or remote MySQL DB
MYSQL_HOST	STRING	URL for remote MySQL DB
MYSQL_DB	STRING	database name of remote MySQL DB
APP_WORKERS	INT	Gunicorn workers - defaults to number of cores
APP_THREADS	INT	Gunicorn threads - defaults to number of cores – 1
APP_PORT	INT	listening port for Gunicorn WSGI, must match in both containers
APP_LOG_LEVEL	STRING	options: debug, info, warning, error, critical

7. Watchdogs

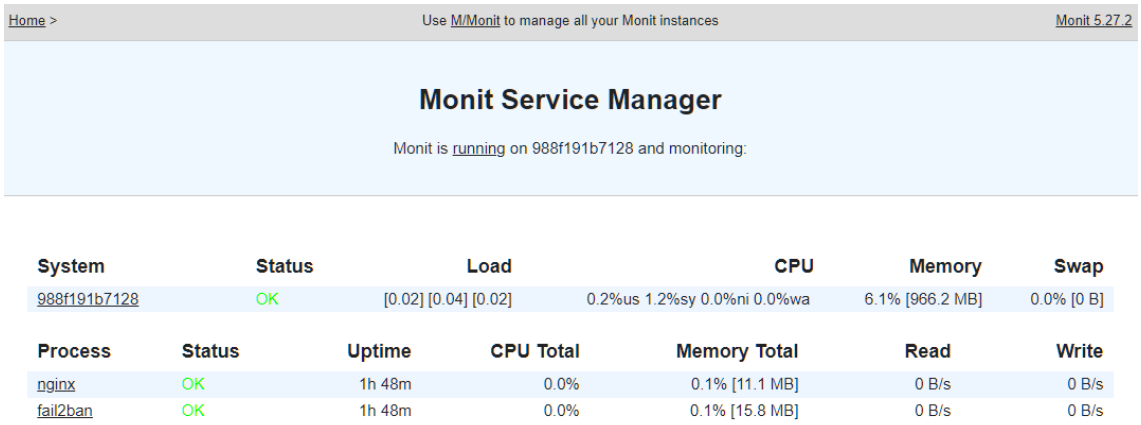
Container1:
Software: Monit
Monitoring: NGINX, Fail2ban
Web-monitor portal: Yes

Monit is set up to monitor NGINX and Fail2ban services every 2 mins and reports the status of each and handles service restart duties if they are found to be inactive. Nginx is monitored via PID file and /healthcheck on port 80 Nginx that returns http code 200 on success; Fail2ban is monitored via PID file and socket.

Monit also provides a web port to both monitor and control system services. This portal is located on port 2812 of container1, and access is provided via NGINX reverse proxy features.

To access Monit's web portal, visit http://localhost/monit/ credentials:
username: admin
password: hivekeeper

Monit Service Manager web portal:



Monit config files are stored in project directory: container1/monit/

Container2:
Software: Monit
Monitoring: Gunicorn3
Web-monitor portal: No

Monit is set up to monitor the Guniicorn3 service every 2 minutes and reports the status and handles service restart duties if they are found to be inactive. Gunicorn is monitored via PID file and /ping on port 80 - /ping located in hivekeepers_app.py, which returns string: 'status: ok'

Monit config files are stored in project directory: container2/monit/
Gunicorn restart script is stored in project directory: container2/dash_app/start_app.sh

Command line access to watchdogs:
It is also possible to access and control watchdog states and status via the command line using: ‘docker exec CONTAINER-NAME COMMAND ARGS’

```
Available monit commands:

monit start all                 # Start all services
monit start <name>              # Only start the named service
monit stop all                  # Stop all services
monit stop <name>               # Stop the named service
monit restart all               # Stop and start all services
monit restart <name>            # Only restart the named service
monit monitor all               # Enable monitoring of all services
```

```
monit monitor <name>            # Only enable monitoring of the named service
monit unmonitor all             # Disable monitoring of all services
monit unmonitor <name>          # Only disable monitoring of the named service
monit reload                    # Reinitialize monit
monit status [name]             # Print full status information for service(s)
monit summary [name]            # Print short status information for service(s)
monit report [up|down|..]       # Report state of services. See manual for options
monit quit                      # Kill the monit daemon process
monit validate                  # Check all services and start if not running
monit procmatch <pattern>       # Test process matching pattern
```

8. Logging

System logs are stored in the directory /home/hivekeeper/logs within each container.

Log files:

app.log	Python/SQL	The main visualisation app log
container1-entrypoint.log	Docker	Start-up script log for container1
container2-entrypoint.log	Docker	Start-up script log for container2
fail2ban.log	Fail2ban	records failed attempts and b IP bans
gunicorn.log	Gunicorn (WSGI)	web server for container2 run log
gunicorn-access.log	Gunicorn (WSGI)	web server for container2 access log
gunicorn-error.log	Gunicorn (WSGI)	web server for container2 error log
nginx-access.log	NGINX	web server for container1 access log
nginx-error.log	NGINX	web server for container1 error log
monit.log	Monit	watchdog for container1 services

Log format:

YYYY-MM-DD HH:MM:SS [SOURCE] [LEVEL] file.ext: message

Example:

2022-03-15 11:11:41,522 [PYTHON] [INFO] hivekeepers_app.py: init app server...

Options:

Logging level and detail/verbosity can be set for both containers within the docker-compose.yaml file.

Container1:

The container log level can be set system-wide for nginx and fail2ban services. Available options are info, notice, warn, error, crit, alert, or emerg

There is also the choice to log all NGINX access requests or to turn this feature off. Available options are simple (no nginx access logging), or detailed (with nginx access logging).

Container2:
The container log level can be set system-wide for the Plotly Dash application and Gunicorn service. Available options are debug, info, warning, error, or critical.

There is also the choice to log SQL queries or to turn this feature off. Available options are yes (all SQL queries will be logged) and no (SQL queries will not be logged).

9. Technical Obstacles

Throughout this project there were many obstacles, both technical and logical, that required thoughtful and resourceful navigating.

Docker:
There was a steep learning curve in learning the Docker framework from the ground up, to build and deploy reliable images, with highly configurable container deployment options.

Issues faced:

- container networking
- container storage persistence
- system permissions
- learning how services interact and operate within a container environment vs a normal bare-metal installation

Python-Dash:
While I'm experienced with python, this was the first time I had encountered the Dash framework. Sometimes the hardest thing is working with something you know well in a novel environment. In this situation your instincts can lead you astray as expectation and practice can diverge in subtle ways that are hard to predict *a priori*.

Issues faced:

- Dash behaves different inside vs outside of the container
- Steep learning curve understanding syntax – documentation could be improved
- counter intuitive logical behaviour - cycles through logic more than once at startup, which caused unexpected file and variable duplications - this is normal behaviour.

Data:
The HiveKeepers database structure was a bit confusing and required a lot of forethought and follow up analysis to understand and process correctly.

Issues faced:

- data needed to be manipulated and reformatted for 3d charts
- delay in accessing remote database
- real data and junk data both being present in dataset
- understanding and using the python SQL framework

10. Additional Notes

User Authentication
NGINX basic authentication user access is managed through a txt file container user:password combinations for referencing requests against. This file needs to be passed/shared with container1. And while it is possible to simply pass such a file in plain text, it is much safer to first encrypt the passwords listed before doing so...

The simplest way to make such an encrypted user:password file is to use the apache2-utils library on Linux – the following is for Debian systems but can be modified for others.

To get the library installed, run: 'apt-get update && apt-get install apache2-utils -y'
Then run as root, run: 'htpasswd -c .htpasswd username'
This will start the application, create the file, then ask you to input the password.

To add more users, simply rerun the command, but use the -b flag in place of the -c flag.

Or, if you have many users you wish to add, I have included a small BASH script (password_script.sh) that can automate the process by reading a plain text user:password structured file and converting it to an encrypted password version using apache2-utils.

This script is located within the /scripts directory within the main project directory.

to use, create a text file with a user:password text pair per user, per line. Then call the script with the text file as the first argument. e.g.: ' ./password_script user_credentials.txt'

e.g., user_credentials.txt file layout for creating 3 users:
user1:password1
user2:password2

user3:password3

11. Quick Start

1. Clone the HiveKeepers project repo to your local host
2. Install Docker
3. Create the user:password file using your preferred method – see below for options.
4. Edit the docker-compose.yml file to your preferred setting, adjust the options to fit your desired log level and location (hosted or in container only).and make sure the container1 htpasswd volume is pointing to your created user:password file
    - If wanting to store logs on host, make sure the log volume is pointing to the desired local directory path
5. Start build and start process with:
    - To run live and see outputs: 'docker-compose up --build --remove-orphans'
    - To run daemonised: 'docker-compose up -d --build --remove-orphans'
6. Browse to http://host_IP/app to access the data visualisation app.
    - Enter username and password
7. Browse to http://host_IP/monit to access container1 watchdog status web portal
    - Enter username and password
8. Enjoy your data visualisations!