ENRGDAQ

An Introduction and Current Applications

Summary

ENRGDAQ is a **modular data acquisition platform** designed for efficient and streamlined data management. The platform operates with distinct components called **DAQJobs (Data Acquisition Jobs)**, each responsible for a specific task to ensure clarity and longevity in the system's design.

Objectives of ENRGDAQ

1. Acquire data.
2. Store data.

This streamlined approach ensures clean code maintenance and a smooth onboarding process for new team members.

Problem Solving with ENRGDAQ

- Modular architecture.
- Separation of concerns via message queue management.
- Remote communication through ZeroMQ's PUB/SUB paradigm.

---

1. Architecture Overview

Message Queues

Each DAQJob is equipped with:

- `message_in` and `message_out` queues.

The **Supervisor** component manages these queues, routing messages between DAQJobs.

- DAQJobs filter messages they accept using `handle_message` methods.
- With these constraints, store jobs that selectively store data according to each message store configuration are orchestrated, ensuring separation of concerns.

ZeroMQ Integration

- Implements PUB/SUB paradigm using `DAQJobRemote`.
- Enables real-time, high-performance communication across different platforms and languages.

---

2. Example Configuration

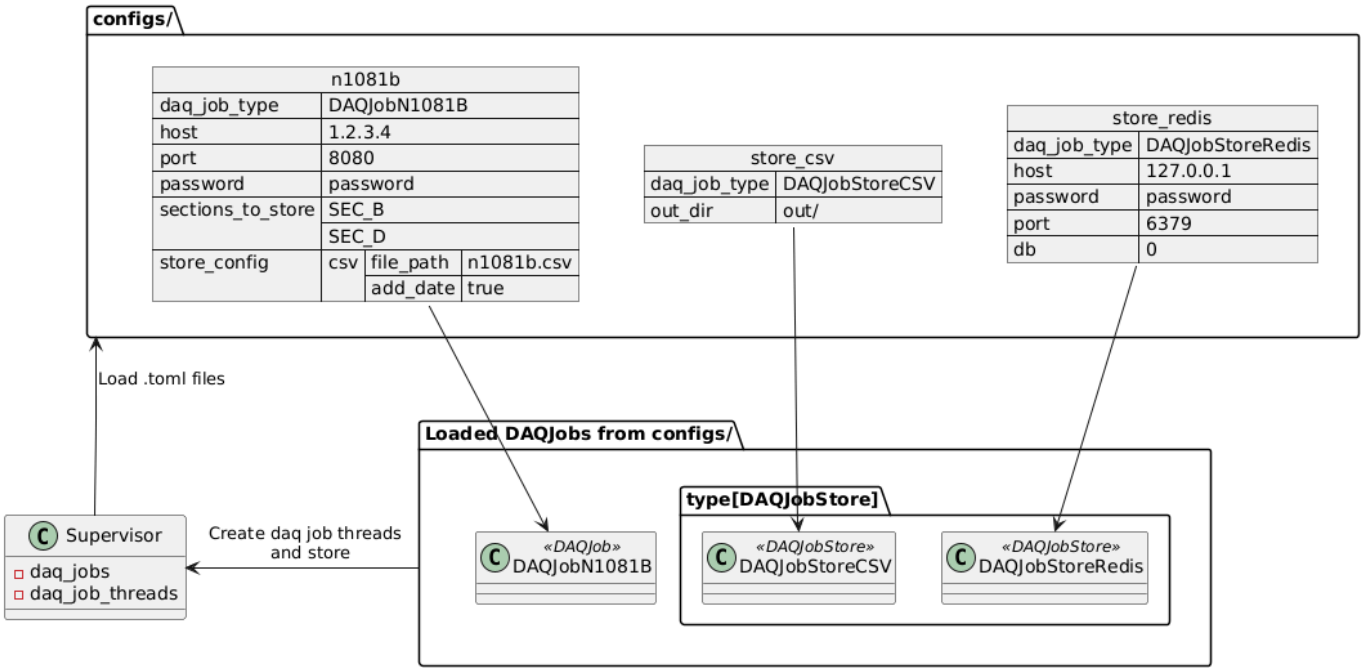
To understand the architecture of ENRGDAQ, let us demonstrate it using an example configuration.

Supervisor

The central class, **Supervisor**, loads configuration files for DAQJobs and creates instances accordingly.

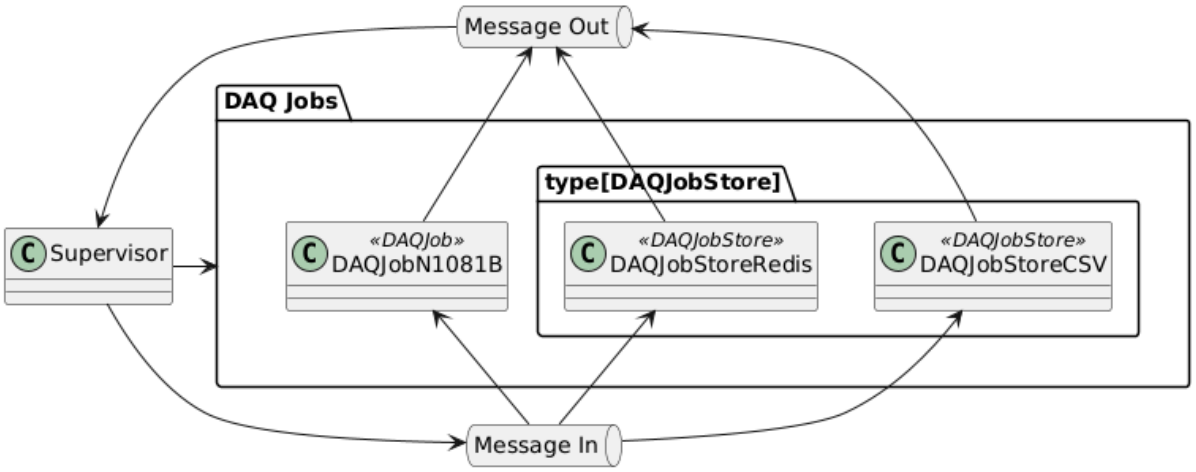
Configuration Files

- **n1081b.toml**:
  Creates `DAQJobN1081B` to connect to a configured host:port and store sections using CSV format.
- **store_csv.toml** and **store_redis.toml**:
  Load storage jobs, with only `store_csv.toml` utilized in this example.



Software Configuration Workflow

1. `Supervisor`: Handles communication between DAQJobs, sends statistics messages, and more.
2. `DAQJobN1081B`: Reads hit data using N1081B's WebSocket API, sending messages for storage.
3. `DAQJobStoreRedis` and `DAQJobStoreCSV`: Stores tabular or raw data based on configuration.

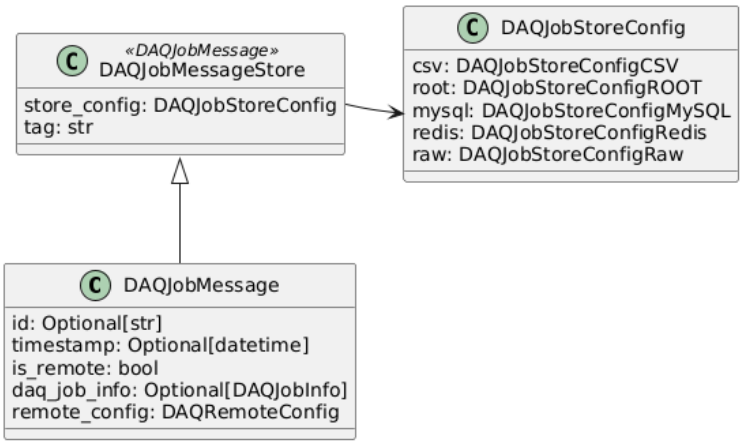


---

3. Messaging Workflow

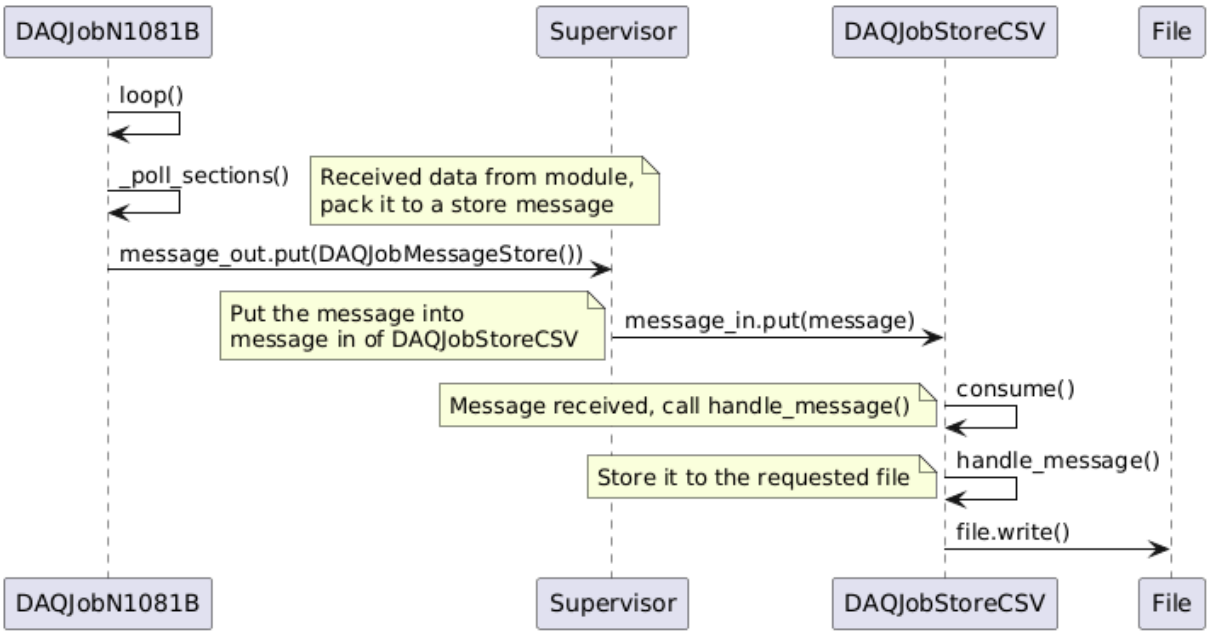
The `DAQJobMessageStore` class facilitates consistent data storage across various formats like CSV, ROOT, Redis, MySQL, etc. Some of its benefits are:

- Gives DAQJobs control as to what to store, where to store, and how to store, via `store_config` parameter.
- Configuration-driven storage logic ensures simplicity and flexibility.
- Restricts I/O operations to avoid code complexity and maintain focus.



Sequence of Events for Message Storage

The following sequence diagram shows the process flow of `DAQJobN1081B` module storing hit information to CSV format.



Conclusion

- Simplifies interface management.
- Allows easy addition of new storage types without modifying existing logic.
- Supports efficient remote connections and ZeroMQ integration.

---

4. Remoting

Why Use Remoting?

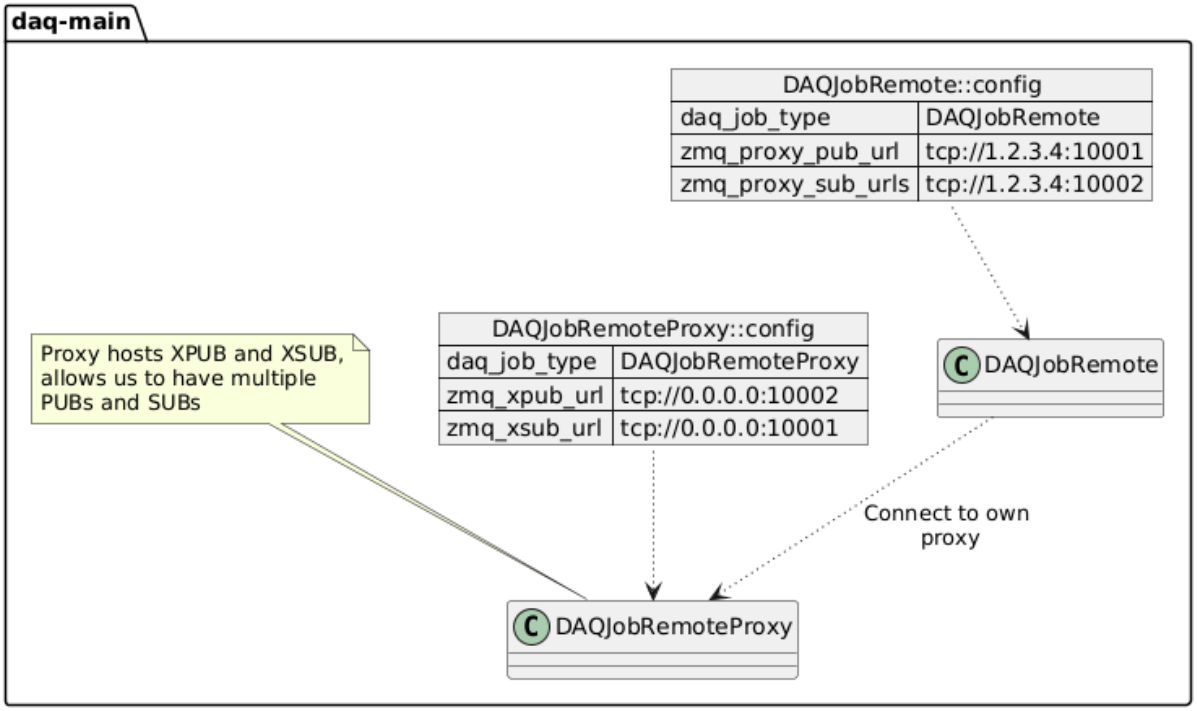
- Instruments may be connected to different computers.
- Data storage may require parallelization.
- Real-time data analysis and data quality monitoring for ML/AI systems.

ZeroMQ Integration

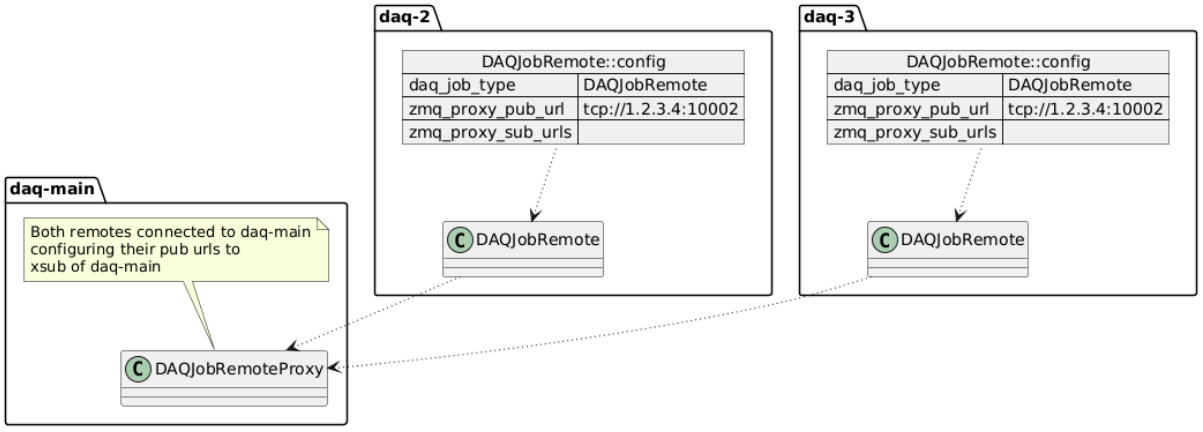
- ZeroMQ PUB/SUB architecture connects message queues seamlessly across instances.
- Supports multiple languages using MsgPack for message serialization, enabling us to connect multiple platforms effortlessly.

Example Setup

Main DAQ instance communicates with two remote instances using ZeroMQ's PUB/SUB paradigm.

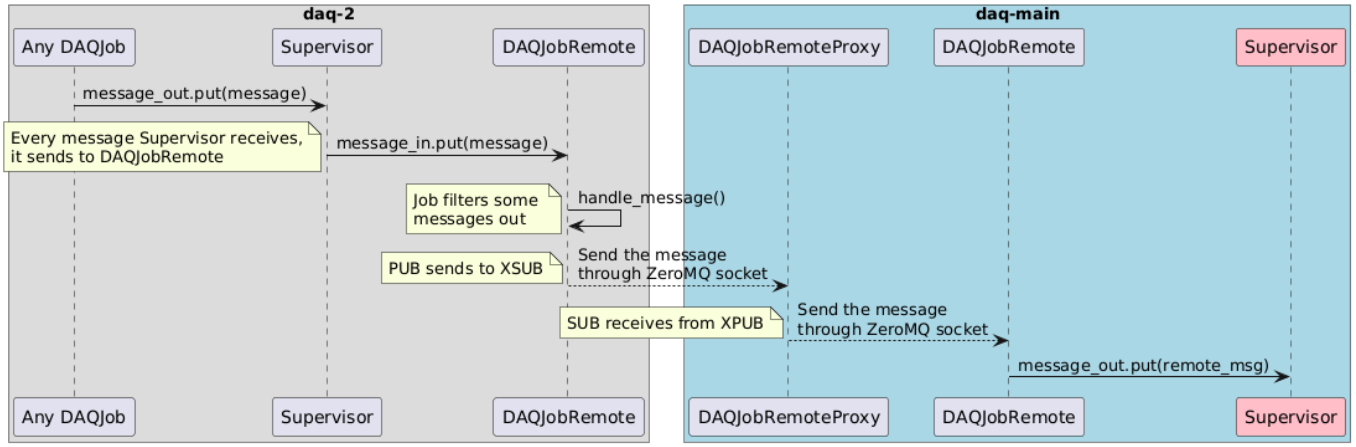


Configuration for Main DAQ instance



Configuration for other two DAQ instances

With configurations like above, communication would happen as in the diagram below:



Conclusion

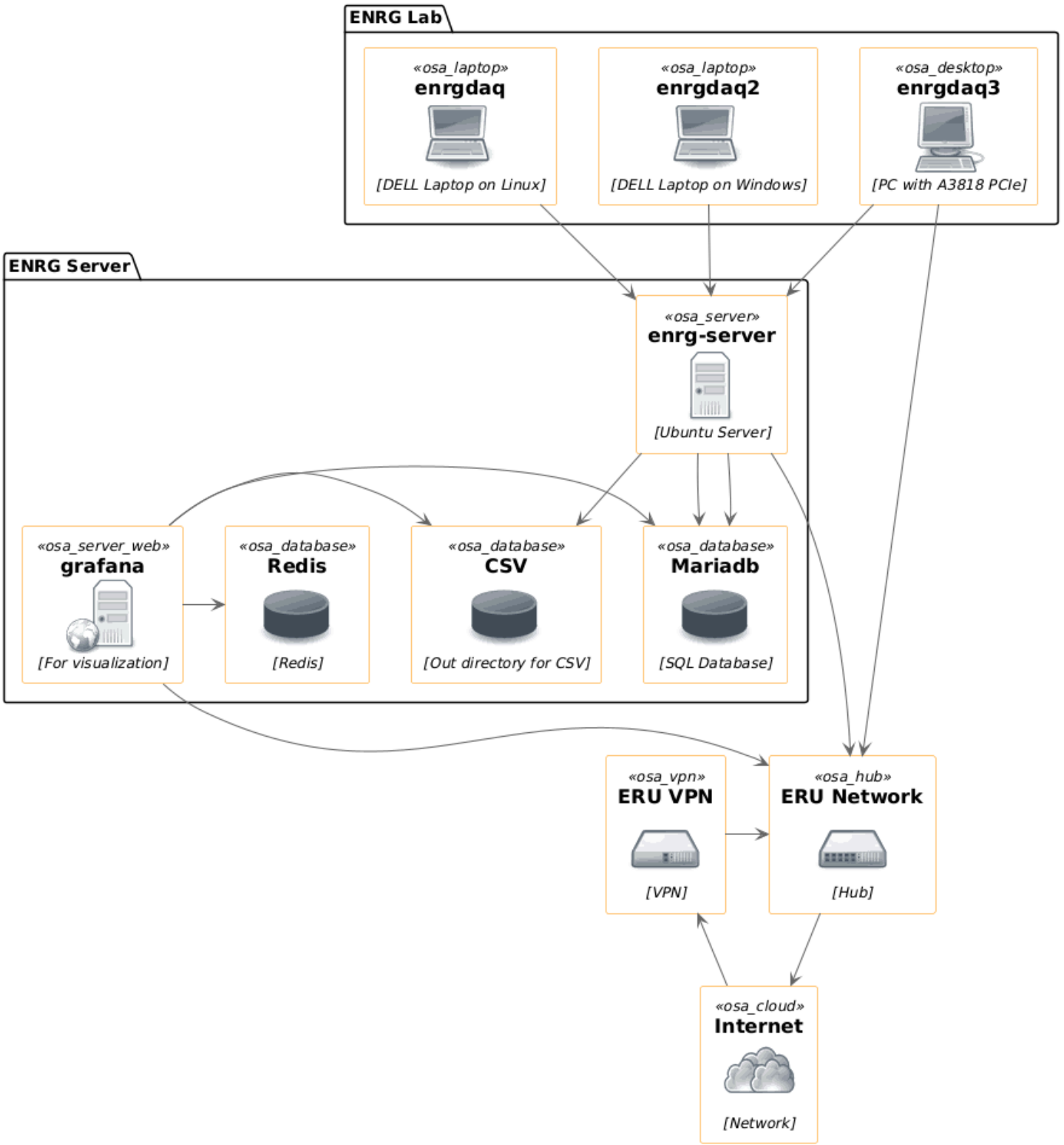
ENRGDAQ's architecture benefits from using a single way to store messages. This makes development easier, the system more scalable, and keeps it stable:

- By using one interface for all message storage, ENRGDAQ avoids the complexity of dealing with different APIs. This makes the system more user-friendly and maintainable.
- The architecture can easily add new storage types (like CSV, ROOT, Redis, or MySQL) without major changes. This ensures adaptability to new technologies.
- By limiting direct I/O access within DAQJobs, the system keeps workflows simple. This prevents overlapping tasks and reduces the chance of errors.
- The unified messaging approach simplifies data transfer between remote systems. The ZeroMQ PUB/SUB paradigm aligns seamlessly with the message queue-based architecture, ensuring reliable and efficient communication.

---

5. Current Usage of ENRGDAQ

Overview



Currently, we operate 3 different computers and a server for our data acquisition, storage and visualization needs.

- `enrgdaq2` is used for data acquisition from N1081B module.
- `enrgdaq3` will be used for the connection to CAEN VME and NIM, via the Optical Link connected through CAEN A3818 PCie card, which we will soon start to gather data from.
- `enrg-server` stores all of the data gathered from `enrgdaq`, `enrgdaq2`, and `enrgdaq3`. It hosts Redis and MariaDB for storage, and has an out directory for CSV files that we back-up regularly (every day). We also have Grafana that visualises data from these sources.

Sensory Data Acquisiton

We gather hit counts from N1081B module via `DAQJobN1081B` and store them to both Redis and CSV. Redis is configured to have a sliding window of 7 days, as we do not use it as a permanent storage, but a fast in-memory storage for our visualization needs in Grafana. Long-term storage needs are met by using CSV, and in the future we plan to add more formats like HBF5 and ROOT.

Realtime Camera

Via `DAQJobCamera` we store cameras on our network in real time in raw form, using `DAQJobStoreRaw`. We poll the camera out in Grafana to see the state of the CAEN instruments and take action immediately if needed.

Realtime Alerts

Via `DAQJobHealthcheck` and `DAQJobAlertSlack`, we can alert a Slack channel realtime in case any DAQJobs go down or any other specific state has been reach that researches need to be aware of, e.g. for a module not responding for the past 2 minutes. This ensures that researchers are immediately informed of potential issues and can take prompt action to mitigate any disruptions.

`DAQJobHealthcheck` is highly configurable, allowing thresholds and conditions for alerts to be tailored to specific operational needs. Additionally, the alerts include detailed diagnostic information, such as the supervisor ID, originated supervisor ID, timestamp, and failure type to streamline troubleshooting. By integrating these tools, we enhance the reliability and responsiveness of our DAQ systems, minimizing downtime and maximizing research productivity.

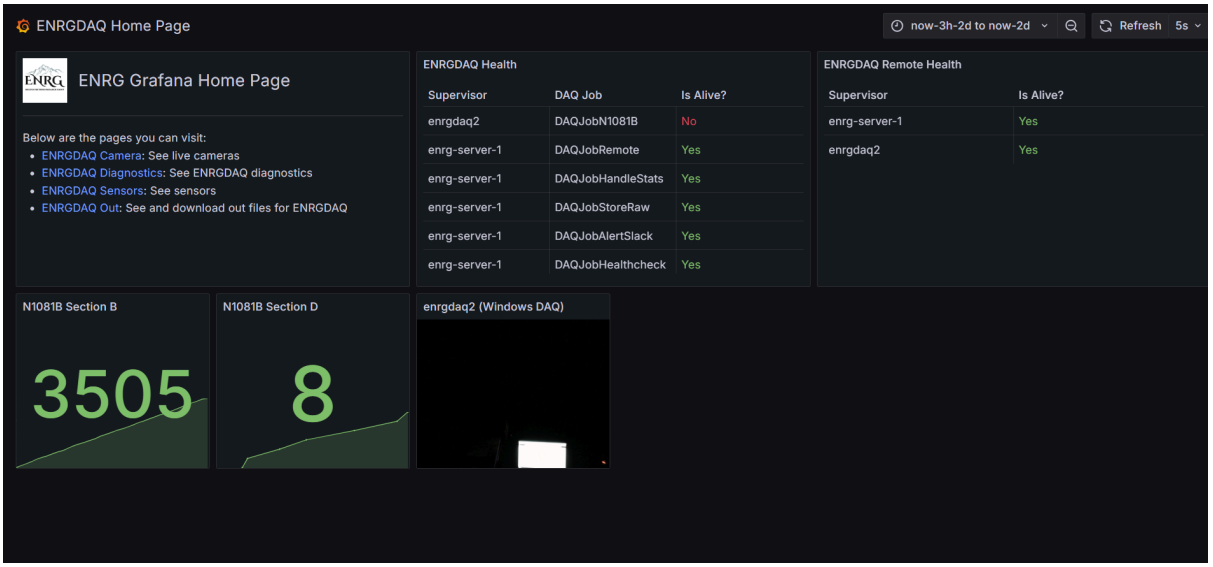
These alerts also are saved via `DAQJobHandleAlerts` to CSV store, which gets displayed in the Diagnostics Dashboard of our Grafana instance.

CSV Out

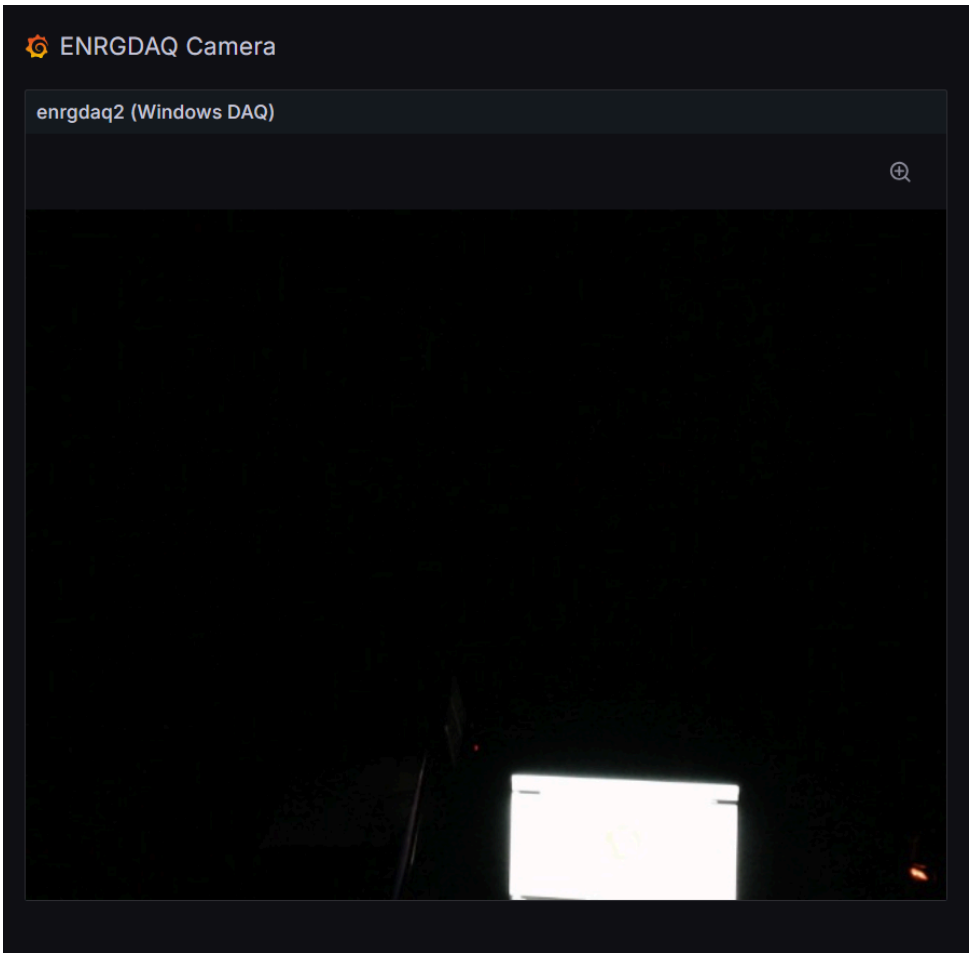
A file index exists for our researches to gather the out data for their needs. This index is updated real time and is categorized by year-month-day. This service is handled by nginx.

Grafana

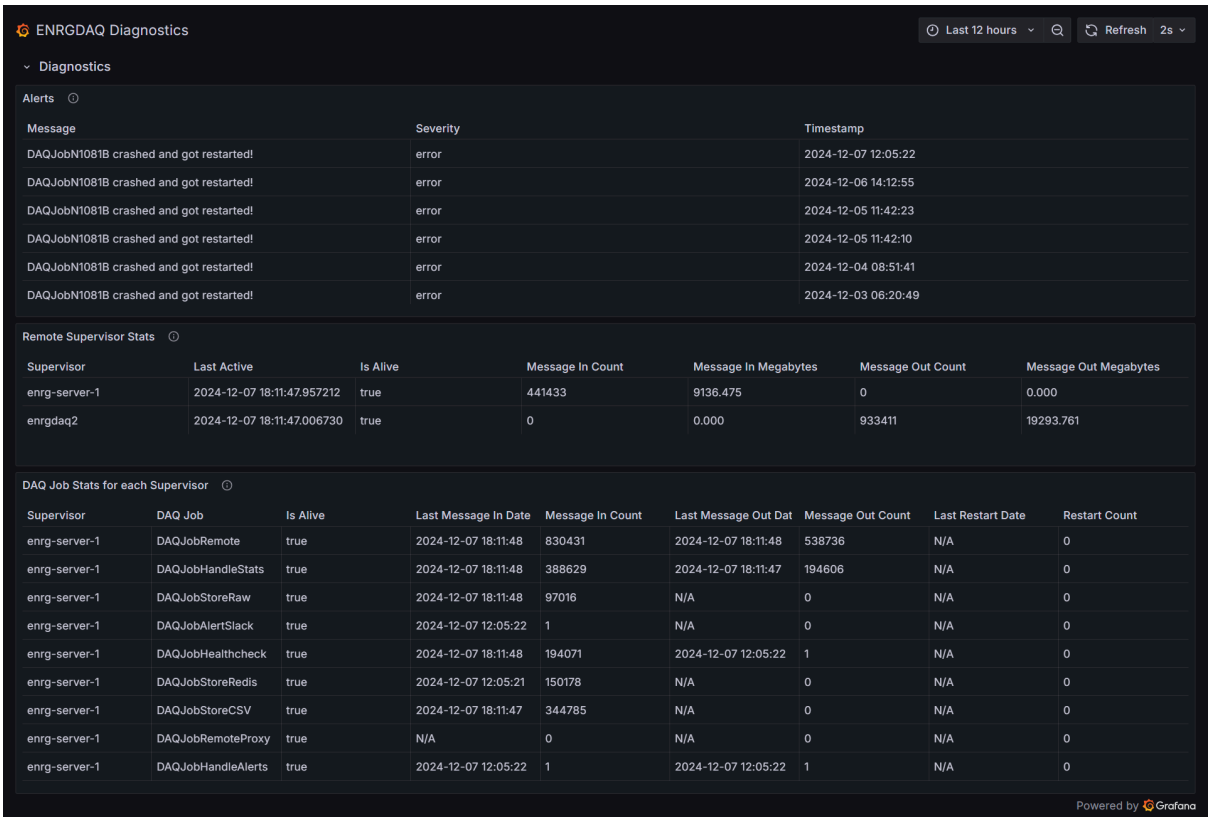
We host a Grafana instance for our researches to see and diagnose the state of ENRGDAQ. Below are some pictures that shows our dashboards:



Home Page Dashboard



Camera Dashboard



Diagnostics Dashboard

Sensors Dashboard

---

6. Development

We use GitHub to develop ENRGDAQ and have a public GitHub Repository for everyone to contribute. CI for the repository has been set-up, so every commit to the main branch is

thoroughly tested, linted and the project documentation gets updated. We also use Github Issues to track down bugs, feature requests, and enhancement ideas.