ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOBL04

KARABO DATA LOGGING: InfluxDB BACKEND AND GRAFANA UI

G. Flucke*, V. Bondar, R. Costa, W. Ehsan, S. G. Esenov, R. Fabbri, G. Giovanetti, D. Goeries, S. Hauf, D. G. Hickin, A. Klimovskaia, A. Lein, L. Maia, D. Mamchyk, A. Parenti, G. Previtali, A. Silenzi, D. P. Spruce[1], J. Szuba, M. Teichmann, K. Wrona, C. Youngman
European XFEL GmbH, Holzkoppel 4, 22869 Schenefeld, Germany
[1]*now at* MAX IV, Fotongatan 2, 22484 Lund, Sweden

Abstract

The photon beam lines and instruments at the European XFEL (EuXFEL) are operated using the Karabo control system that has been developed in house since 2011. Monitoring and incident analysis requires quick access to historic values of control data. While Karabo's original custom-built text-file-based data logging system suits well for small systems, a time series data base offers in general a faster data access, as well as advanced data filtering, aggregation and reduction options. EuXFEL has chosen InfluxDB as backend that is operated since summer 2020. Historic data can be displayed as before via the Karabo GUI or now also via the powerful Grafana web interface. The latter is e.g. used heavily in the new Data Operation Center of the EuXFEL. This contribution describes the InfluxDB setup, its transparent integration into Karabo and the experiences gained since it is in operation.

KARABO AND THE EuXFEL

The European X-ray Free Electron Laser (EuXFEL) facility [1] provides hard and soft X-ray beams at three photon beamlines to six instruments. Up to 27,000 photon pulses per second are arranged into 10 Hz trains with an intra-train pulse repetition rate of 4.5 MHz. The Karabo framework [2–4] has been designed and developed in-house since 2011 for control, online data analysis, and data acquisition at the photon beam lines and the scientific instruments.

In Karabo, so-called devices communicate via a central message broker. All devices using the same broker *topic* form a Karabo installation. Whereas broker communication is considered to be "slow" data, big or "fast" data like images are sent via TCP/IP data pipelines that can be flexibly configured, e.g. for calibration, analysis, or preview purposes. Figure 1 gives an overview of a Karabo installation.

A Karabo device exposes a self-description of its control interface, i.e. its *schema*. Karabo's generic graphical user interface (GUI) uses the schema to render the representation of a device. Devices can have one of manifold tasks:

- interface some hardware like a pump or a motor,
- control a detector and read out its data,
- analyse data,
- orchestrate other devices,

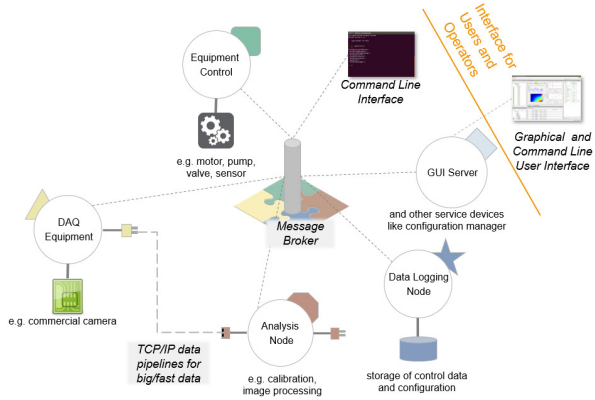

Figure 1: A Karabo installation showing Karabo devices with various tasks. Broker and pipeline communication lines are indicated.

- provide a system service like serving as entry point for the GUI, logging data, managing alarm states, or managing configurations.

To communicate with each other, the Karabo devices expose methods that can be called remotely in the distributed system. Besides being directly called, these *slots* can be subscribed to *signals* of other devices. When such a signal is emitted with arguments, all subscribed slots are called with these arguments. In the process, only a single message is sent to the broker that distributes the message according to the subscriptions, as is shown in Fig. 2. This signal/slot mechanism allows Karabo to be fully event-driven, regular polling of e.g. device properties is not needed. That a single message to the broker is sufficient also for a device with a signal that many other devices have subscribed to, ensures that there is no overhead for such a "popular" device.

FIRST KARABO DATA LOGGING IMPLEMENTATION

Data logging in a Karabo installation is organised via a few dedicated devices. A "data logger" device (or several that share the load) subscribes to the signal for property updates of the other devices. Properties are configuration parameters or read-only values like the reading of a temperature sensor. Via the signal/slot mechanism, the logger is informed about every property update and when this update occurred, i.e. the timestamp of the update, and stores it in the backend of the logger. Similarly, the device schema and its potential

* gero.flucke@xfel.eu

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOBL04

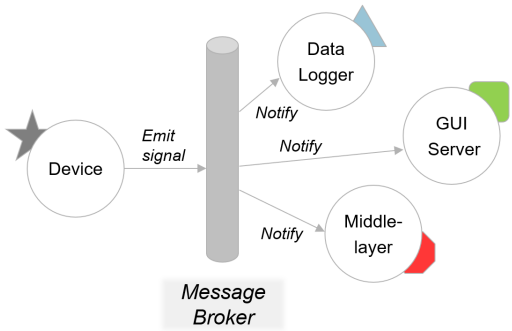

Figure 2: Karabo signal/slot messaging via the broker.

run-time updates are stored. "Data log reader" devices are responsible to read back the stored data. One can query either the changes of a property in a time interval (*trendline* data) or all properties of a device and its schema at a given point in time. A single "Data logger manager" device per Karabo installation orchestrates loggers and readers.

The first Karabo data logging implementation, started in 2014, was based on human readable ascii files. Each file logs properties of a single device with one line per property:

```
20200719T184128.573059Z
  |1595184088.573059|804227972
  |actualPosition|FLOAT|35.91043||VALID
```

Besides the timestamp in two formats and the corresponding id of the EuXFEL train of photon bunches, the name, type and value of the property, and some meta information is logged. A further file stores the device schema, serialised to an XML format. To reasonably speed-up reading all property updates in a time interval, binary index files are created per property of interest. Figure 3 shows the result of such trendline requests in the Karabo GUI. The actual position of a motor and its state are displayed, as part of an incident analysis. Since Karabo is event-driven, no new data points exist when the motor is not moving. The zoom into the end of the curve shows that the motor decelerated and got stuck – in fact, the detector moved by the motor collided with another object since, after a change of the setup, the limit switch that would prevent such collisions was unfortunately not adjusted.

Due to limited screen resolution, a GUI cannot display arbitrary details of a trendline. Therefore, the trendline data request specifies a maximum number of points to return. If the requested time interval contains more data points, a simple down-sampling algorithm is applied: just every 2nd, 3rd, etc. data point is returned.

Drawbacks

Deployment of such a logging and data retrieval mechanism is easy since it depends only on the availability of disc space and therefore suits well for small Karabo installations. On the other hand, operation at EuXFEL reveals some (obvious) drawbacks. Storing data in text files does not scale well concerning disc space. At EuXFEL, data older than three months were automatically moved to an archival

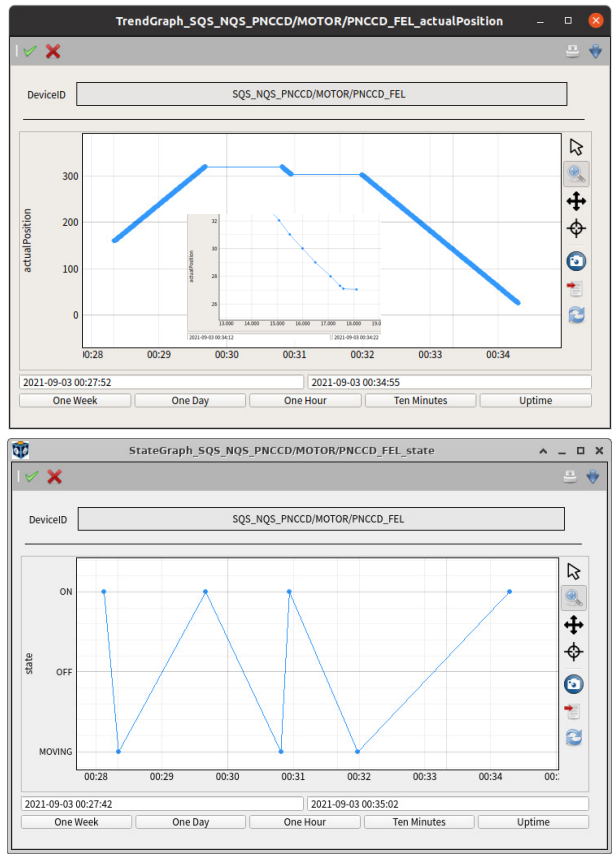

Figure 3: Historic trendline data for the position in mm of a motor that is moved back and forth, overlayed with a zoom into the right end of the curve (top), and the corresponding motor state changes (bottom).

location. Hence, access to such data required extra effort. Furthermore, data read-out could be slow: depending on the number of property updates since a device was last restarted, reading back the configuration of a device at a given point in time could take in the order of minutes. The lack of statistical methods for data downsampling could result in the loss of short timescale features in the data. Two requests of the same data with only slightly differing time intervals could return markedly different data, due to the sparsing algorithm that was implemented.

TRANSITION TO AN INFLUXDB BACKEND

Given the drawbacks of the ascii file based solution, the Karabo development team looked for a better storage backend. Timeseries databases were considered to be particularly well suited, as they are optimised for retrieving data along the time axis. InfluxDB [5], Prometheus [6], and Timescale [7] were considered and finally InfluxDB was chosen. A prototype Karabo device interacting with an InfluxDB proved the feasibility in 2018.

In order to allow for a transparent transition and to keep the text based logging backend available for small Karabo

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOBL04

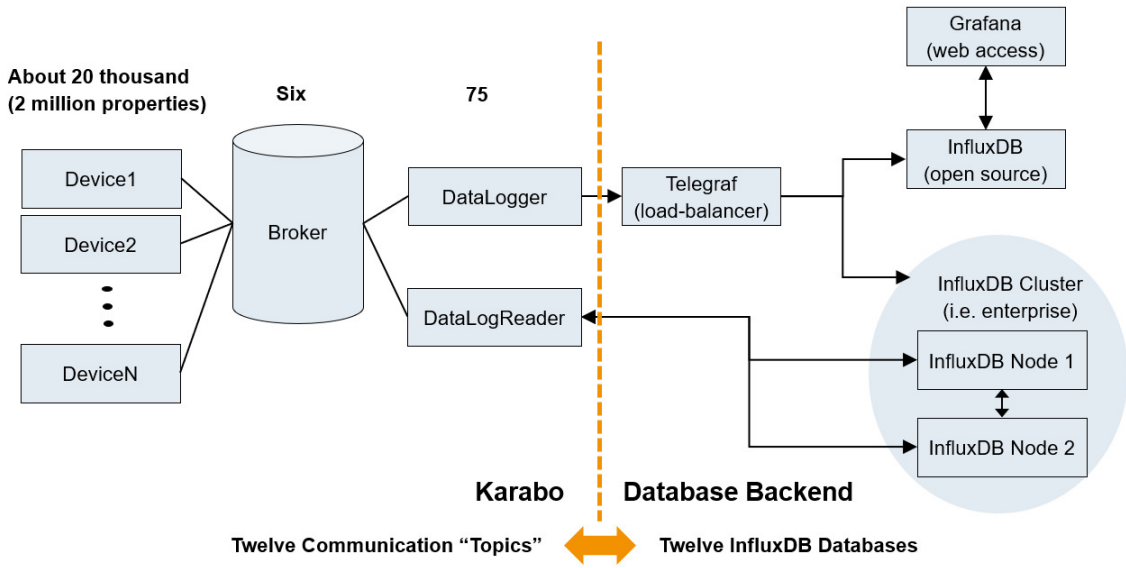

Figure 4: Schematic view of Karabo data logging and its InfluxDB backend.

installations, a new logger and a new log reader device were developed. The logger manager device is now configurable to switch between ascii file and InfluxDB backend.

An InfluxDB installation can host many independent *databases*. All data of a Karabo installation is stored in one such database, identified by the broker topic of the Karabo installation. Each device is mapped to three InfluxDB *measurements*, one for the schema, one for events like device instantiation and shutdown, and the main one for the device properties. InfluxDB *fields* relate to Karabo properties. To ease schema evolution of devices like changing the data type of a property, the field name is suffixed with its type, e.g. `actualPosition-FLOAT`. The mapping between the Karabo and InfluxDB supported types is straight forward for integer or floating point numbers, booleans and strings. Special treatments are needed for the following cases: Non-numeric floating point values (`nan`, `inf`) are stored as strings in fields with an extended field name suffix (e.g. `-FLOAT_INF`), unsigned 64-bit integers are reinterpreted as signed integers, and newline characters in strings have to be mangled. Since InfluxDB does not support arrays, vectors of numbers are stored as comma separated strings. To overcome the ambiguity between an empty vector of strings and a vector with a single empty string, vectors of strings are stored as the base64 encoded JSON representation. Only the Karabo specific data types like raw binary data, table data, and the schema require Karabo for de- and encoding. These types are stored as base64 encoded result of Karabo's binary serialisation.

Many devices update their schema regularly at run time, but typically only a few schema variants exist. Since the serialised binary of a schema (i.e. the self-description of a device) can easily surpass 500 kB, only a schema digest is stored for every update. The schema itself is only stored when the value of such a digest is not yet stored in InfluxDB.

Property updates are transmitted from the data logger device to database backend using the InfluxDB line protocol. By default, data is flushed when at least 200 lines are accumulated.

Figure 4 shows a schematic overview of the Karabo data logging infrastructure with its InfluxDB backend. About 20,000 devices with roughly 2 million properties are distributed among twelve Karabo installations that communicate via six broker instances. In total, 75 data logger devices write to a single load-balancer. The balancer duplicates and forwards the data, once to a standalone InfluxDB instance running the open source edition, and once to a cluster of two instances for data and service redundancy. Running instances in a cluster requires the InfluxDB enterprise edition. In case one of the two storage locations is down, the balancer caches the data in memory. The cluster is the main backend, serving Karabo initiated read requests, whereas the open source instance can be used by other services, without interfering with the control system.

Performance and Operational Experience

InfluxDB data logging at the EuXFEL is in production since summer 2020; data from January 2020 onward has been migrated into the new system. So far, more than 240 billion property updates have been stored, increasing by about 10 billion each month. In total, about 14 TB of disc space is needed per InfluxDB node so far.

The network input to the load balancer is typically about 20 Mb/s, but varies, reflecting that Karabo is event-driven and that the list of Karabo devices that are online at any given point in time can vary. Usually, the data is available for reading with a delay of about 30 s only. This performance was significantly compromised when a Karabo device injected data with timestamps months in the future. Due to the internal data layout in InfluxDB *shards* containing temporal blocks of data, the database frequently internally reorganised

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOBL04

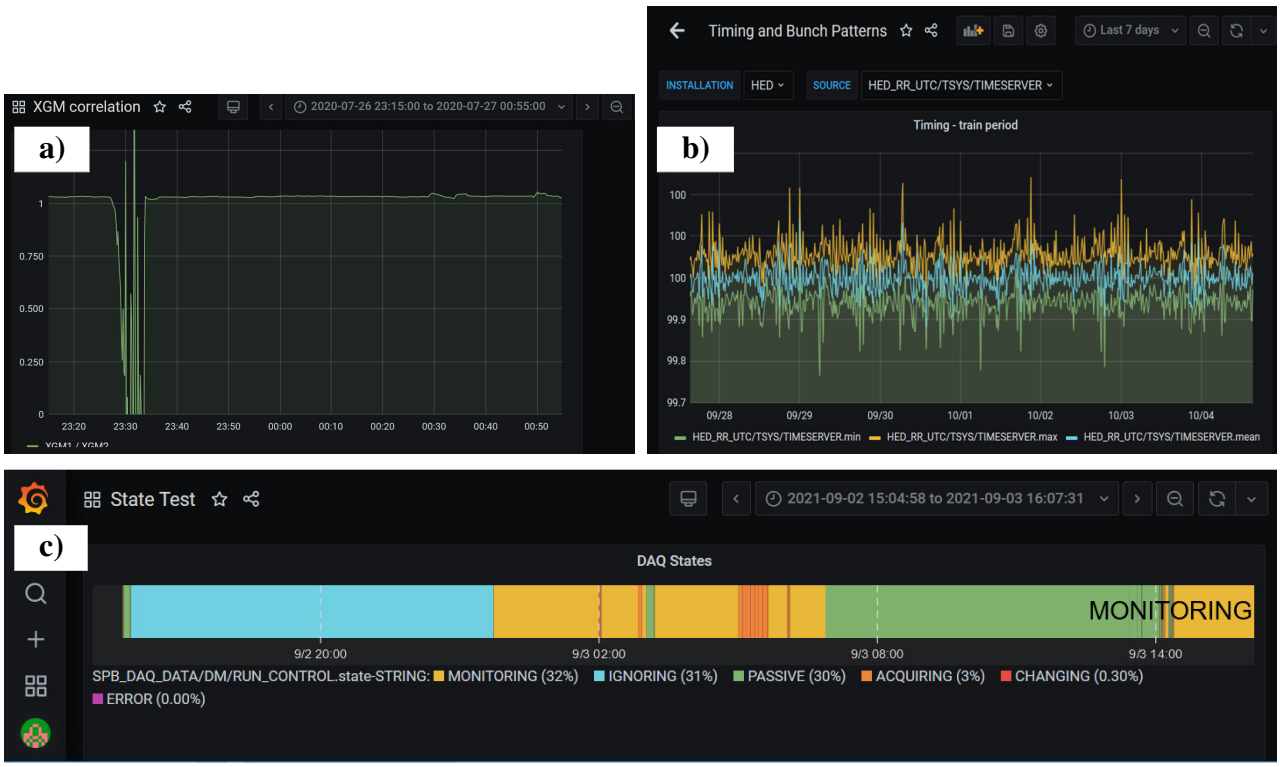

Figure 5: Examples of data displays using Grafana: The ratio of properties from two different devices (a), an overlay of minimum, mean, and maximum of a data trendline that has too many points to show them all (b), and the visualisation of the time development of a string property (c).

data, and the latency increased to over an hour. If there are too many points in a requested time span, down-sampling of trendline data is done in InfluxDB by querying averaged data in equally spaced time intervals. Therefore, short-term features are better preserved and data requested for only slightly different time intervals is more uniform.

All the data stored so far (> 1.5 years) is quickly available. A GUI request to display one week of data of a property that updates every 5 s (and therefore needs averaging) is almost instantaneous.

The planned 3 years of data retention will be affordable. In addition, the InfluxDB internal organisation of the data in shards offers the possibility to keep a reduced set of the data (e.g. averages) for even longer.

GRAFANA USER INTERFACE

Using InfluxDB instead of a custom data storage backend allows external tools to query, retrieve and visualise the data. At the EuXFEL, Grafana [8] has been chosen due to its rich feature set, the fact that both Influx query languages, InfluxQM and Flux, are exposed, and due to a large and active online community that provides many examples.

The resulting low entry threshold enables non-developers to create data views in so-called dashboards, available for others. A nice example of a simple data display that is not easily available within Karabo is the correlation of properties of different devices, see Fig. 5a). When down-sampling is needed, further statistical options besides averaging are available, e.g. minimum and maximum value within the evaluated interval. Overlaying all three as in Fig. 5b) allows allows efficient visualisation of trends alongside outlier preservation. Grafana can even be extended by plugins. The one used in Fig. 5c) displays the time evolution of a string that changes between a few discrete values.

Besides usage in the instrument hutches, Grafana dashboards have become a key element of the Data Operation Center (DOC) at the EuXFEL. The DOC is a co-effort of EuXFEL's Data Department groups: Controls, Electronic and Electrical Engineering, Data Analysis, IT & Data Management, and Detector Operation. The DOC monitors services that the department provides during X-ray operation, and gives pro-active support for the scientific instruments. The main Grafana dashboard of the DOC gives an overview of the status of the most important services and displays alerts if something is outside its expected range, e.g. the frame rate of centrally triggered cameras, see Fig. 6. In this example, the status overviews show no errors whereas the DOC alerts show some long lasting, non-critical problems[1]. Scientific data taking and detector calibration processing are followed closely. For similar systems placed at several instruments, the pull-down menu of a generic dashboard can be used to switch between the different installations, enabling the

[1] Further fine tuning is needed to avoid their appearance on the main page.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOBL04

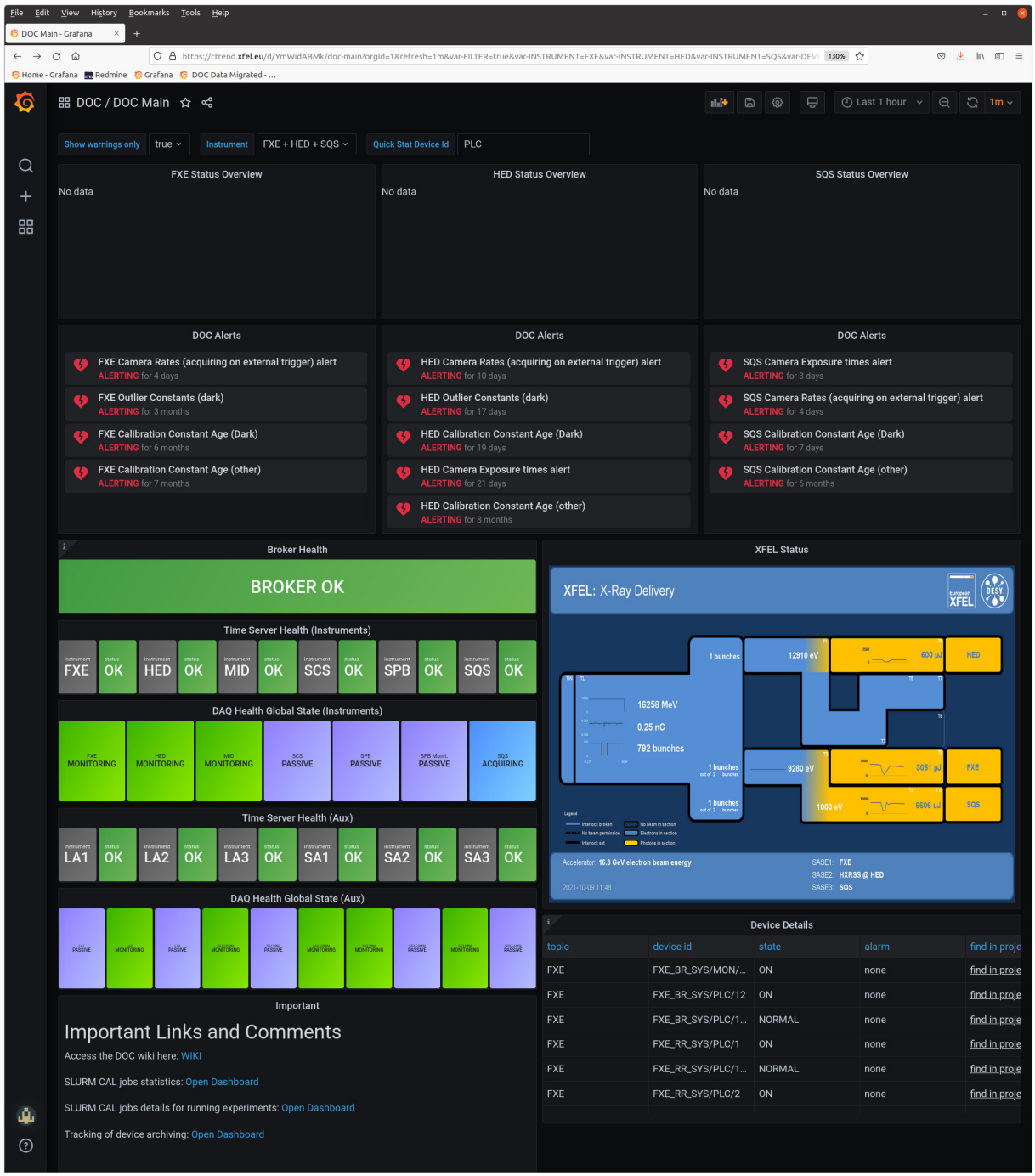

Figure 6: The main Grafana dashboard of the EuXFEL Data Operation Center (DOC), selected to focus on the HED, FXE and SQS instruments.

DOC shift crew to change focus, depending on the running scientific programme, but keeping a unified layout.

CONCLUSIONS

The event-driven, broker-based control system Karabo is used to operate the photon tunnels and scientific instruments at the EuXFEL. Its first ascii file based data logging system was able to fulfil the initial needs, but the longer the facility operated, the clearer it became that the file based backend does not scale well with long term operation. Furthermore, the poor down-sampling capability sometimes confused operators interested in the past state of their system.

A new storage backend using InfluxDB has been developed and is now in operation at the EuXFEL since summer 2020. The goals to overcome slow responses and the lack of statistical methods for data read-back have been met.

Even more, the storage in a community-driven backend allows using a well developed data display and analysis tool,

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOBL04

Grafana, without interference with the control system. Its wealth of features and online examples made Grafana dashboards a key ingredient of the new EuXFEL Data Operation Center that focuses the support of the scientific programme given by the EuXFEL Data Department.

All together, the efforts to interface Karabo data logging with InfluxDB and to setup a reliable database infrastructure have surpassed the expectations to improve the user experience when interacting with historic data in the Karabo control system.

REFERENCES

[1] M. Altarelli, "The European X-ray free-electron laser facility in Hamburg," *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, vol. 269, no. 24, pp. 2845–2849, 2011.

[2] B. C. Heisen *et al.*, "Karabo: An integrated software framework combining control, data management, and scientific computing tasks," in *Proc. 14th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13)*, (San Francisco, CA, USA), JACoW Publishing, Oct. 2013, pp. 1465–1468.

[3] S. Hauf, B. Heisen, *et al.*, "The Karabo distributed control system," *J. Synchrotron Rad.*, vol. 26, pp. 1448–1461, 2019, ISSN: 1600-5775. DOI: 10.1107/S1600577519006696.

[4] G. Flucke *et al.*, "Status of the Karabo control and data processing framework," in *presented at the 17th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'19)*, (New York, NY, USA), JACoW Publishing, Oct. 2019, p. 938.

[5] InfluxData Inc., *InfluxDB*, https://docs.influxdata.com/influxdb.

[6] *Prometheus*, https://prometheus.io.

[7] Timescale, *Timescale*, https://www.timescale.com.

[8] Grafana Labs, *Grafana*, https://grafana.com.