
Grafana@ctrend.xfel.eu Documentation

Release 1.0

CTRL, ITDM

Mar 07, 2021

Contents

1	Accessing ctrend.xfel.eu	3
2	Teams and Authorization	5
2.1	Team Administration	6
2.2	Support Group Teams	7
3	Dashboards	9
3.1	Organizing Content via Dashboards	9
3.2	Dashboard Layout	9
3.3	Dashboard Variables	9
4	Panels	21
4.1	XFEL Datasource Specifics	21
4.2	Data Type Support	22
4.3	Creating Panels and Visualizations	22
4.4	Examples	23
5	Exporting Data	35
5.1	Reducing and Selecting Data For Export	39
6	Indices and tables	43

Contents:

CHAPTER 1

Accessing ctrend.xfel.eu

European XFEL staff can access ctrend.xfel.eu using their regular credentials.

Note: General access to ctrend.xfel.eu does not automatically authorize you to view or possibly edit the content. Which content you can access depends on the *teams* you are a member of.

Teams and Authorization

Permissions to view and edit content on the Grafana installation at ctrend.xfel.eu are controlled through a combination of so-called *teams* and *folders*:

Folders folders group dashboards and panels into areas of similar contents. Folders exist e.g. for each instrument, or for general topics such as detectors, photon diagnostics, or vacuum. Multiple XFEL groups may be responsible for curating the content of a folder.

Teams a team is a group of users which should have the same minimum access level to a *Folder*. Hence, teams are assigned to *Folders* with e.g. view-only, or editor access. A given user can be member of multiple teams, and multiple teams may be authorized to access a folder.

For the ctrend.xfel.eu installation teams generally map to the folder structure: there are at least two teams configured to access each folder:

View-only teams users in these teams can view the content, i.e. the dashboards and contents of a folder. They cannot edit existing dashboards, nor can they create new ones. Modification of dates and times to be accessed is possible, but cannot be persisted. Data can be exported, e.g. to CSV.

Editor teams users in these teams can view the content, i.e. the dashboards and contents of a folder. In addition, they can create new dashboards and panels within the folder the team is assigned to, change date and times a dashboard or panel shows, and also persist this change as the default view. Users can modify existing dashboards and panels, and edit e.g. graphing option. Data can be exported, e.g. to CSV.

Note: Teams do not strictly relate to XFEL groups: a SPB scientist wanting to view a panel in the MID folder would be added to the MID view-only team to do so.

Note: Privileges on folders are assigned to Teams by CTRL, and team editors cannot change them. However, the association of Teams to Folders is not expected to change frequently. Users should be added to a team already assigned to a given folder, rather than assigning a new team, and thus many users to that folder.

Table 1: Folders at ctrend.xfel.eu and team access to them. View-only teams always have the suffix *-view-only*, even if not indicated in the table. SUPPORT contains all teams from the DATA and instrumentation departments. ALL contains all teams.

Folder	View-only teams	Editor teams
FXE	FXE-view-only, SUPPORT	FXE, CTRL
SPB	SPB-view-only, SUPPORT	SPB, CTRL
MID	MID-view-only, SUPPORT	MID, CTRL
HED	HED-view-only, SUPPORT	HED, CTRL
SCS	SCS-view-only, SUPPORT	SCS, CTRL
SQS	SQS-view-only, SUPPORT	SQS, CTRL
SASE1	ALL	VAC, XPD, BKR, CTRL, EEE, XRO
SASE2	ALL	VAC, XPD, BKR, CTRL, EEE, XRO
SASE3	ALL	VAC, XPD, BKR, CTRL, EEE, XRO
Beckhoff	SUPPORT	EEE, CTRL
Calibration	SUPPORT	DET, DA, CAL, CTRL
Cameras	SUPPORT	DET, DA, CAL, CTRL
DAQ	SUPPORT	ITDM, CTRL
Data Analysis	SUPPORT	DA, CTRL
Detectors	SUPPORT	DET, CTRL
Lasers	SUPPORT	LAS, EEE, CTRL
Photon Diagnostics	SUPPORT	XPD, CTRL
Timing	SUPPORT	EEE, CTRL
Vacuum	SUPPORT	VAC, CTRL
X-ray operations	ALL?	XO, CTRL

2.1 Team Administration

For each team at least one privileged user exists, who can administer the team through the ctrend.xfel.eu interface. Specifically, this user can add new members to the team. Usually, the privileged users will be the same for both the view-only and editor teams of a given folder.

Note: Staff wishing to view the content of a folder should contact these privileged users to be granted access.

Please see the table below, for who administers which folder.

Table 2: Folders at ctrend.xfel.eu and corresponding privileged users.

Folder	View-only team	Editor team
FXE		
SPB		
MID		
HED		
SCS		
SQS		
SASE1		
SASE2		
SASE3		
Beckhoff	S. Hauf	
Calibration		
Cameras	S. Hauf	
DAQ		
Data Analysis		
Detectors		
Lasers		
Photon Diagnostics		
Timing	S. Hauf	
Vacuum		
Xray-Operations		

If you are a privileged user see team_admin for how to add users to a team.

2.2 Support Group Teams

Staff from the DATA and instrumentation department groups will be placed into teams named after their groups. These teams are administrated by privileged users from the group and should only contain members from the group. These teams will be authorized to minimally view, and in case of need, edit all folders, such as to provide efficient support.

In Grafana, Dashboards are screens which aggregate multiple **:ref:'panels'_** into logical views. Detailed information on dashboards can be found in the Grafana documentation: <https://grafana.com/docs/grafana/latest/features/dashboard/dashboards/>. Here we will only give a short introduction on examples how Dashboards are used in the trend.xfel.eu context.

Note: Depending on your access rights for a given folder you may not be able to execute or view some of the actions and features described in this section.

3.1 Organizing Content via Dashboards

Dashboards are foreseen to be the usual way of grouping related information (panels) within each Team's folder. As such it is advised that they are given meaningful names, e.g. relating to a certain components, or component types: PP Lasers, Cameras, XGM,

3.2 Dashboard Layout

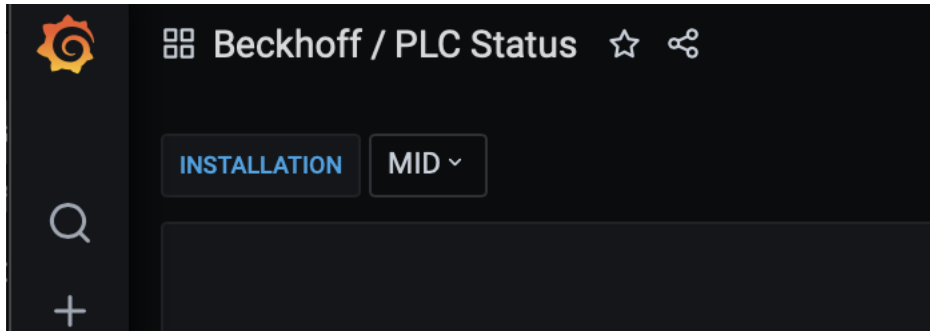
Dashboards organize content on a row x column grid. While you can add rows such that scrolling is necessary, keep in mind that you might want to display a Dashboard on overview screens in the hutches and control rooms. Here you likely want everything in view - without scrolling. Check for instance if multiple data sources can be aggregated in the same panel, or if organizing content over multiple dashboards may make sense.

3.3 Dashboard Variables

At the European XFEL many components are implemented in standardized ways. Hence you will find very similar devices in Karabo for different instruments, or in different component groups. Dashboard variables allow you to

parameterize panels, by variables which can be entered on a dashboards. This allows you to use a single panel to display data from similar data sources.

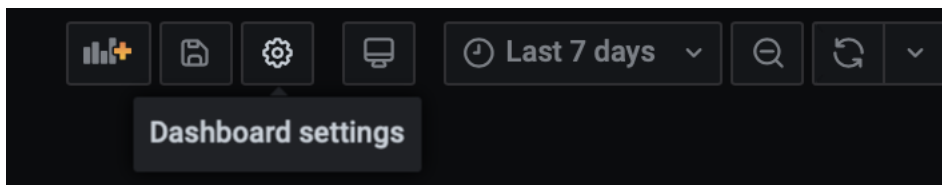
An example of such usage is the parameterization of the general overview dashboards and panels on the Karabo topics, as shown below.



Here the variable refers to different data sources which are then used in the panel to query information for.

Variables can also be used to parameterize e.g. device names.

To create a variable for a dashboard, click on the gears symbol to access the dashboard settings.



Then click on “Variables” and “New”. You should now see a screen similar to the following:

The important fields to edit in this dialog are:

Name give a name to the variable. This is also the parameter name under which you will later refer to the variable. E.g. a variable “INSTALLATION” will be accessible via the parameter `$INSTALLATION` in panels and queries. For this reason the name should not contain spaces or special characters. In fact, it is advised to use all capital letters with underscores as separators where needed. This will allow you to quickly identify variables e.g. in queries.

Type determines the options which can be selected for a variable. Common useful types are *Query*, *Datasource*, *Textbox* and *Custom*. See the example below for how to use each of these. Depending on what you select here the dialog may change in the following.

3.3.1 Example: Jungfrau Frame Rates

In the following we will use each of the dashboard variable options introduced above to display the last recorded input and output framerates of Jungfrau receiver device.

We start by creating a panel in an otherwise empty dashboard which simply refers to a single Jungfrau receiver device, `HED_IA1_JF500K1/DET/JNGFR01` in `HED` directly:



To accommodate for the four Jungfrau detectors currently available at HED, we could either

- add all other detectors into the same panel, or
- create another three panels, for each detector.

We could then do the same for the other instruments with Jungfrau detectors, such as MID, FXE and SPB. Every time a detector is renamed or added, we would then adapt our panels.

Variables allow us to parameterise the dashboard and panels however and reduce this error-prone copy-paste-adapt work.

In an first attempt we create a **Textbox** which allows the user to enter the device id of the Jungfrau detector they are interested in. We already give a meaningful default value as an example. For this we create a variable *DETECTOR*:

Variables › Edit

General

Name	DETECTOR	Type	Text box
Label	optional display name	Hide	

Text options

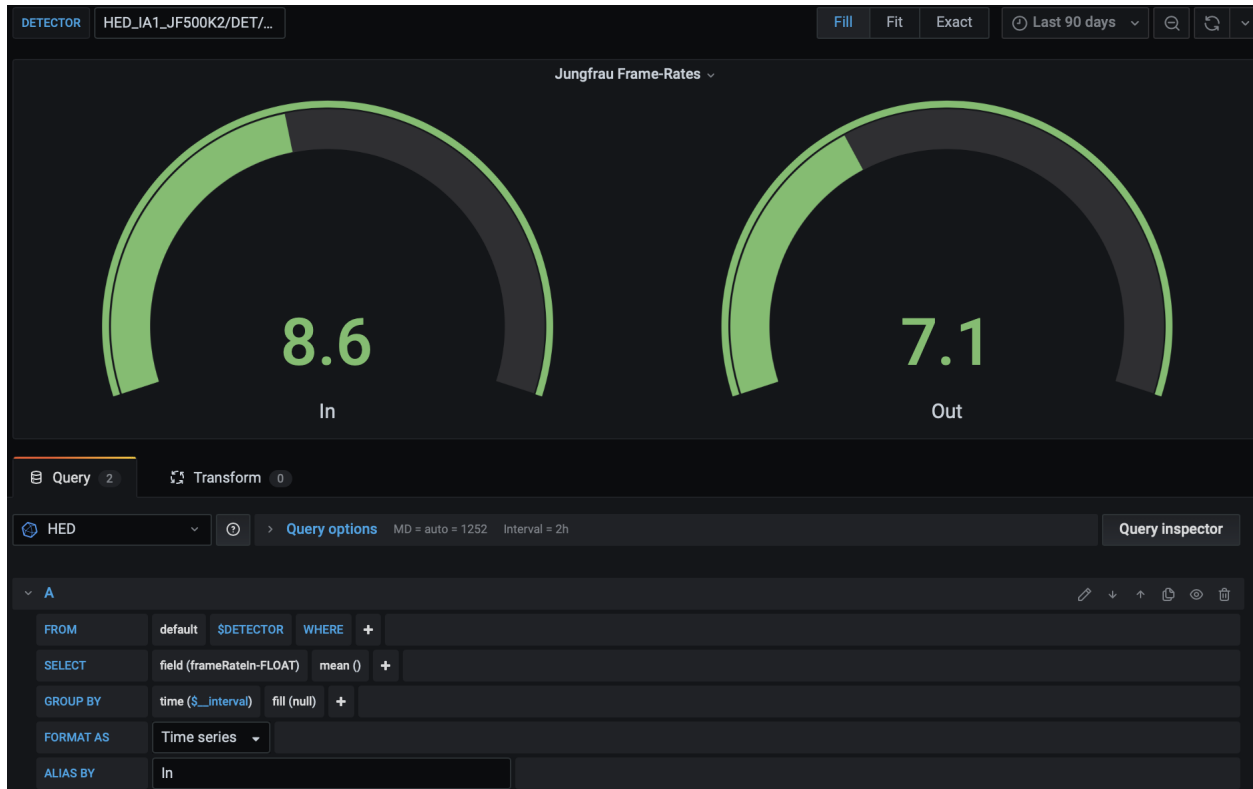
Default value	HED_IA1_JF500K1/DET/JNGFR01
---------------	-----------------------------

Preview of values

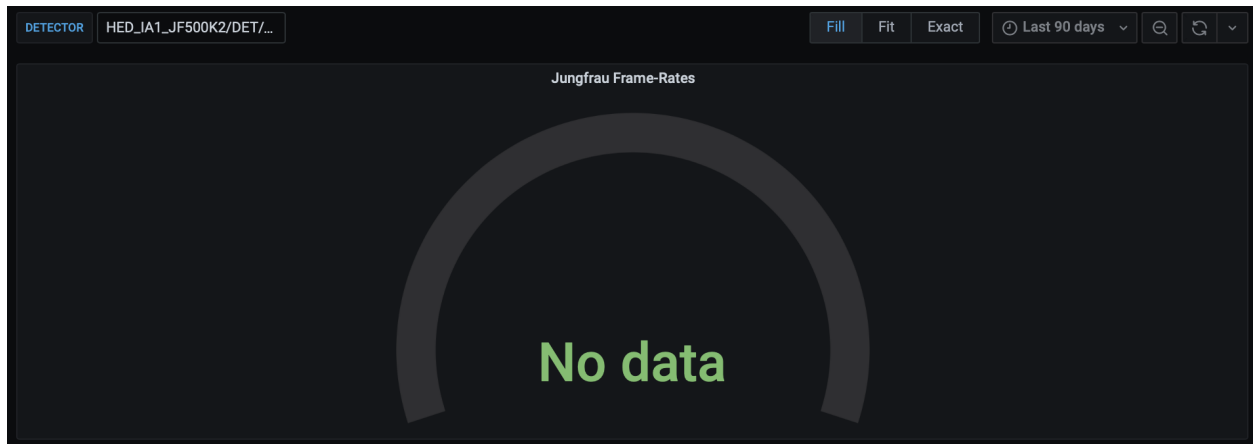
HED_IA1_JF500K1/DET/JNGFR01

[Update](#)

In our panel, we can now refer to this variable via `$DETECTOR`. We thus adapt our queries to use the the variable as measurement name, rather than fixing it in the query itself:



We can now type a different device id in the text box Grafana created to enter the variable. However, this immediately highlights a down-side of the textbox approach. The text is free-form, and thus, errors can easily occur. If in case of our Jungfrau detector we neglect to adjust both indices in the name, e.g by typing `HED_IA1_JF500K2/DET/JNGFR01`, while it should be `HED_IA1_JF500K2/DET/JNGFR02`, no data will be found in the database:



It would thus be much better to restrict the input the user can give for the `$DETECTOR` variable to meaningful values only.

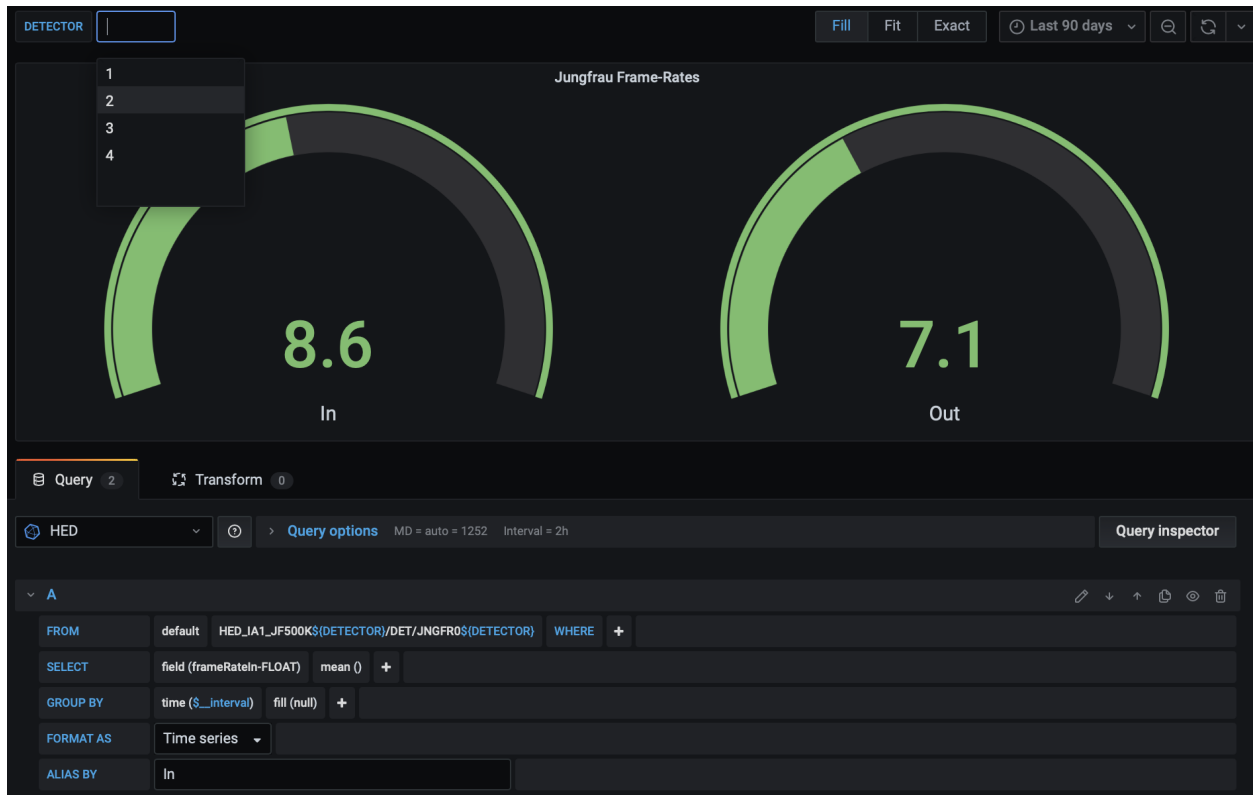
We do this by using the **Custom** variable type, which allows us to define a fixed set of options.

Since the only part of the same that changes is the numerical index, we shorthand the options to the numbers 1-4:

The image shows a configuration dialog for a dashboard variable named 'DETECTOR'. It is divided into three sections: 'General', 'Custom Options', and 'Selection Options'.
- **General:** 'Name' is 'DETECTOR', 'Type' is 'Custom', 'Label' is 'optional display name', and 'Hide' is unchecked.
- **Custom Options:** 'Values separated by comma' is checked, and the value '1,2,3,4' is entered.
- **Selection Options:** 'Multi-value' and 'Include All option' are both unchecked.
- **Preview of values:** Shows four buttons labeled '1', '2', '3', and '4'.

Note how we get a preview of the selectable options as the bottom of the dialog.

Since the variable `$DETECTOR` now only contains the numerical index and not the full measurement name anymore we have to slightly adapt our query:



The measurement field in the query now inserts the index at the correct locations: `HED_IA1_JF500K${DETECTOR}/DET/JNGFR0${DETECTOR}`. Note the `${VAR}` syntax to refer to variables within static text.

While this approach is already much better in that it allows to select only values for the `$DETECTOR` variable which resolve to data in the database, it still has a few shortcomings: we will need to update our variable definition whenever a Jungfrau detector at HED is removed or added; furthermore, we cannot simply copy the panel to other instruments. Their naming structure will not match the HED names. The panel will also break if a Jungfrau detector is ever introduced at HED that does not strictly follow the `HED_IA1_JF500K${DETECTOR}/DET/JNGFR0${DETECTOR}` template we used, e.g. if it is located in `IA2` instead of `IA1`.

Much better would be if we could create a variable which takes its options from the database itself, rather than us statically defining these. This we can do using the **Query** variable type:

Variables › Edit

General

Name: DETECTOR Type: Query

Label: optional display name Hide:

Query Options

Data source: HED Refresh: Never

Query: SHOW MEASUREMENTS WITH MEASUREMENT = /^.*\/DET\/JNGFR[0-9]+\$/

Regex: /.*(.*)-*/

Sort: Disabled

Selection Options

Multi-value:

Include All option:

Value groups/tags (Experimental feature)

Enabled:

Preview of values

HED_JA1_JF500K1\/DET\/JNGFR01 HED_JA1_JF500K2\/DET\/JNGFR02 HED_JA1_JF500K3\/DET\/JNGFR03 HED_JA1_JF500K4\/DET\/JNGFR04

Here we define a query on the *HED* database that lists us all *MEASUREMENTS*, i.e. device ids that match a specific regular expression: `SHOW MEASUREMENTS WITH MEASUREMENT = /^.*\/DET\/JNGFR[0-9]+$/`.

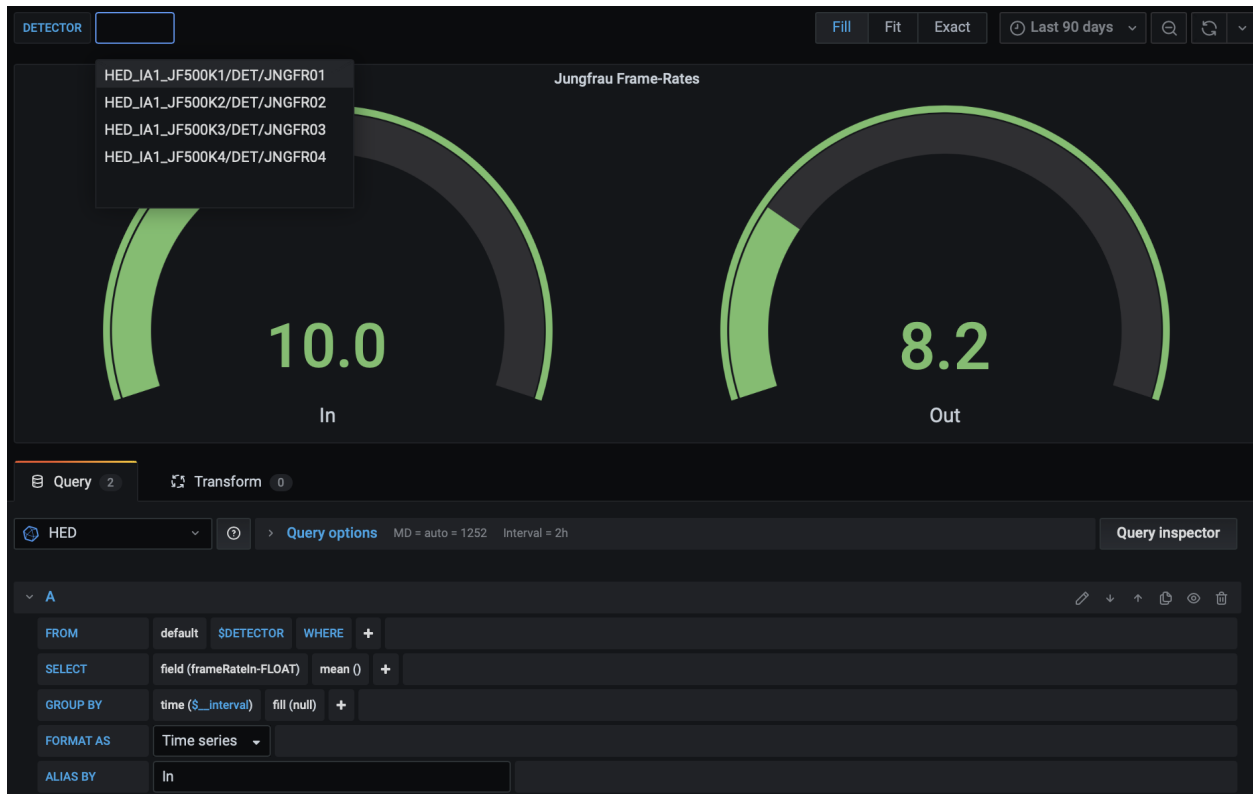
Let's first look at the general *InfluxQL* syntax for listing measurements in a database. `SHOW MEASUREMENTS` will list all measurements. However, we know that the device ids and thus measurement names of Jungfrau detectors follow a certain pattern: they will be of class *DET* and thus contain `/DET/` in their name, alongside the *JNGFR* identifier. We can thus qualify the measurements we are interested in using the *WITH* statement coupled with a regular expression that defines our naming constraints: `SHOW MEASUREMENTS WITH MEASUREMENT = /^.*\/DET\/JNGFR[0-9]+$/`.

Let's have a closer look at the regular expression. In Influx (and Grafana) regular expressions are enclosed in slashes (/). The expression then defines the start of a string to be part of the name (^). While not strictly needed here, this is good practice to indicate, alongside the end of the string (\$). We then allow any number of characters to match up to the `/DET/` part, for which we need to escape the slashes (/) as they are to be interpreted literally in the expression. We then fix that `/DET/` needs to be followed immediately by *JNGFR*, which in turn is followed by digits (`[0-9]`) of which there needs to be at least one (+). Afterwards we do not allow additional characters - the device id needs to end after a digit (\$) .

Note: More details on the `SHOW MEASUREMENTS` can be found here: https://docs.influxdata.com/influxdb/v1.8/query_language/explore-schema/#show-measurements. A good overview of regular expression syntax is available on the Python documentation: <https://docs.python.org/3/library/re.html>.

With the above changes the `$DETECTOR` variable will now contain the full measurement name again and will have options selectable which match the regular expression `^.*\/DET\/JNGFR[0-9]+$`. If we add or remove a detector at HED these options will automatically update and be correct as long as the detector name minimally contains `/DET/JNGFR` followed by digits only at the end.

We adapt the query accordingly, as we can now use the `$DETECTOR` variable directly again:



We've now solved the problem of automatically updating the variable if Jungfrau detectors are added or removed in the HED topic. However, we still cannot easily copy our dashboard into a different instrument. Both in the query defining the options for the `$DETECTOR` variable and the queries populating our panel with data the HED topic is hard-coded.

To make our dashboards and panels reusable in other topics we now define a `**Datasource*` variable alongside the `$DETECTOR` variable. We name it `INSTALLATION`:

General

Name	INSTALLATION	Type	Datasource
Label	optional display name	Hide	

Data source options

Type	InfluxDB
Instance name filter	/^[A-Z]{3,3}\$/

Selection Options

Multi-value	<input type="checkbox"/>
Include All option	<input type="checkbox"/>

Preview of values

FXE HED MID SCS SPB SQS

Add

Here we have selected the *Datasource* type, and then defined the source type as *InfluxDB*. We've further used a regular expression to limit the options to the instrument topics. For this we assume the an instrument topic name will consist of exactly three capitalized alphabetical characters: `/^[A-Z]{3,3}$/`, indicated by the `{3,3}` restriction, which is *{minimum length, maximum length}*. There should be no characters before or after these three letters, hence the `^` and `$` tokens at the beginning and the end of the regular expression.

Conveniently we can use the `$INSTALLATION` variable directly in the definition of the query determining the options of the `$DETECTOR` variable:

General

Name: DETECTOR Type: Query

Label: optional display name Hide:

Query Options

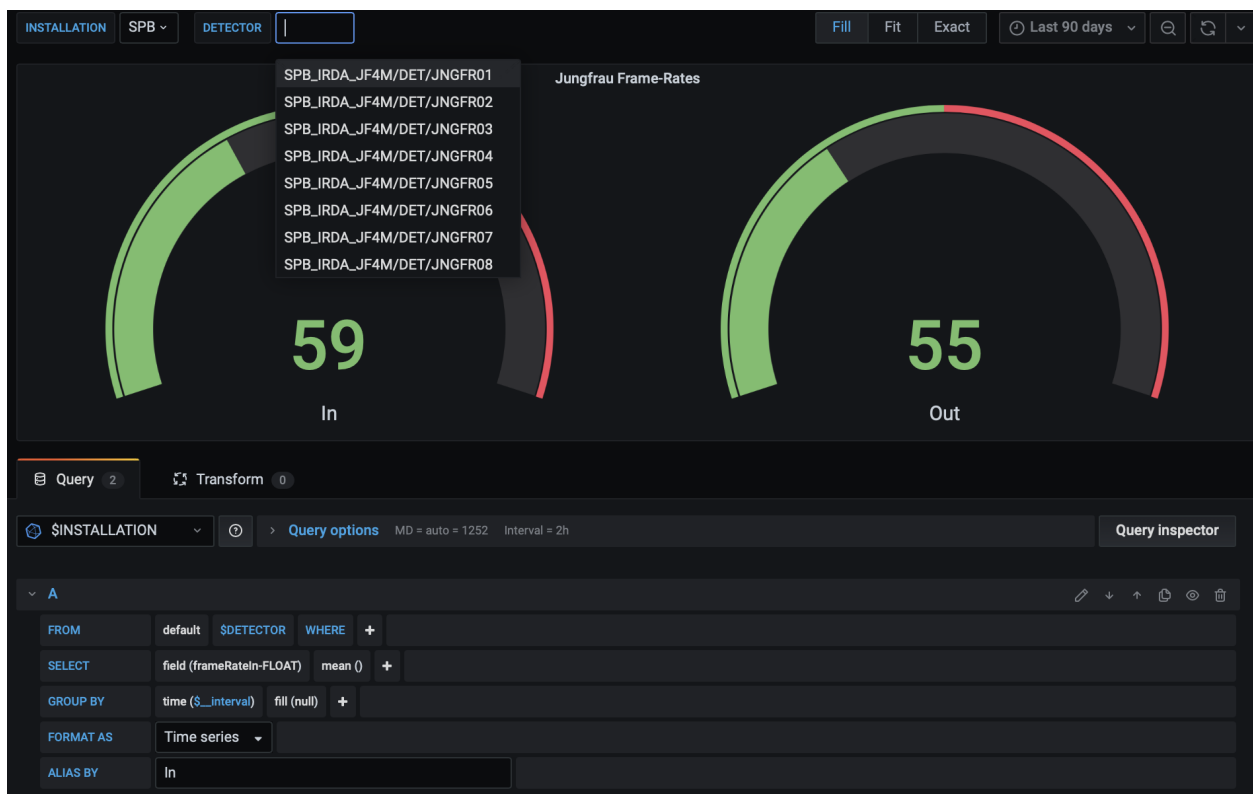
Data source: \$INSTALLATION Refresh: Never

Query: SHOW MEASUREMENTS WITH MEASUREMENT = /^.*\VDET\JNGFR[0-9]+\\$/

Regex: /.*-(.*)-*/

Sort: Disabled

All that is left is to also make use of the variable in our panel queries. We can now select any of the instruments as *INSTALLATION* and any of their Jungfrau detectors as *DETECTOR* to be shown in our panel:



Panels

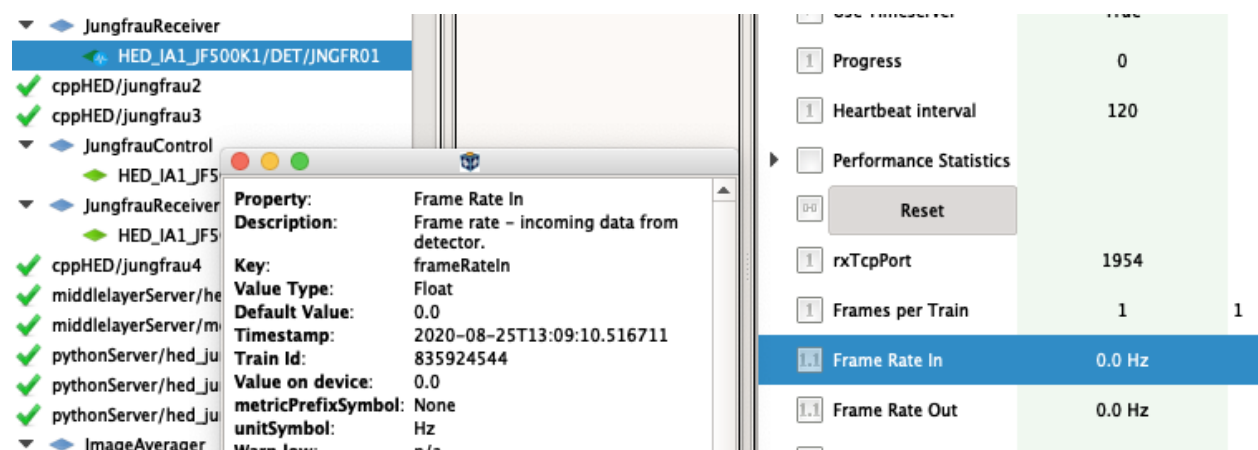
In Grafana, panels are plots or tables which are displayed on dashboards.

Note: Depending on your access rights for a given folder you may not be able to execute or view some of the actions and features described in this section.

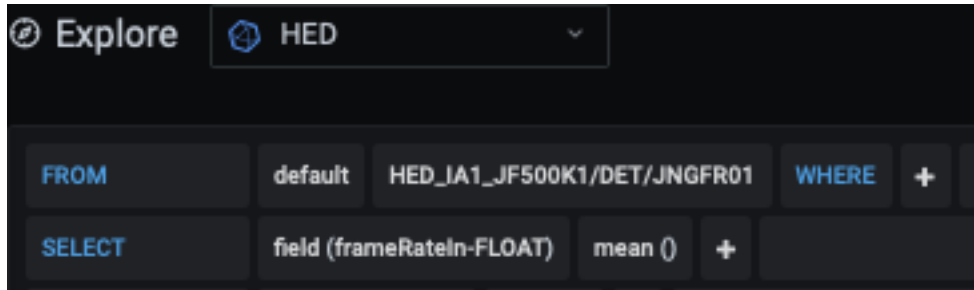
4.1 XFEL Datasource Specifics

Panels are used to display so-called *fields* which belong to *measurements*. For Karabo logging, each device relates to a *measurement* and each parameter of the device is a *field*. For the latter, the field name consists of the parameter name with the type appended to it. This is so that Karabo types can properly be restored when reading back from the database, even in case of evolving schema on the Karabo side.

A parameter *frameRateIn* on a device *HED_IA1_JF500K1/DET/JNGFR01* as shown in the following screenshot



will thus be represented in a measurement *HED_IA1_JF500K1/DET/JNGFR01* under the field *frameRateIn-FLOAT* in InfluxDB and thus Grafana:



Since device name uniqueness is guaranteed for each Karabo topic, and measurement name uniqueness must be assured in InfluxDB, each Karabo topic stores its logging data in a dedicated Influx database named after the topic. In the example above this is *HED*.

4.2 Data Type Support

All Slow Control Karabo data types are supported by the InfluxDB loggers.

However, Karabo has a few datatypes which are not represented well by InfluxDB native types, or have limitations in being displayed in Grafana. The below list indicates current support and limitations:

Floating Point the *float* and *double* data types are handled by the native floating point type of InfluxDB and will display as expected in Grafana.

Integer types are represented by the native integer type of InfluxDB. They will display as expected in Grafana. There is a minor exception in that Influx does not support *uint64* ('*unsigned long long*') data types. We thus cast this type to the supported *int64* type. This will lead extremely large numbers, which use the 63rd bit, being displayed negative in Grafana. This problem is foreseen to occur very rarely, if at all.

Boolean types are converted to 0 (false) and 1 (true) and display as such in Grafana

Strings are stored as strings and display as such in Grafana. Note that for strings the aggregate function set is greatly reduced in InfluxDB, e.g. you cannot aggregate strings to a *mean* or *max* value. However, counting the occurrence of (distinct) strings is possible. Strings best display in the *table* display widget of Grafana.

Vectors are serialized into a base64-encoded string form. Their content is thus opaque to InfluxDB and Grafana. However, counting the number of distinct occurrences can be useful in some scenarios, such as for the bunch pattern table.

4.3 Creating Panels and Visualizations

Creating a panel and the available visualizations are well described in the Grafana Documentation: <https://grafana.com/docs/grafana/latest/panels/add-a-panel/>, and <https://grafana.com/docs/grafana/latest/panels/visualizations/>. Make sure to check the menu on the left of these pages for chapters with detailed information.

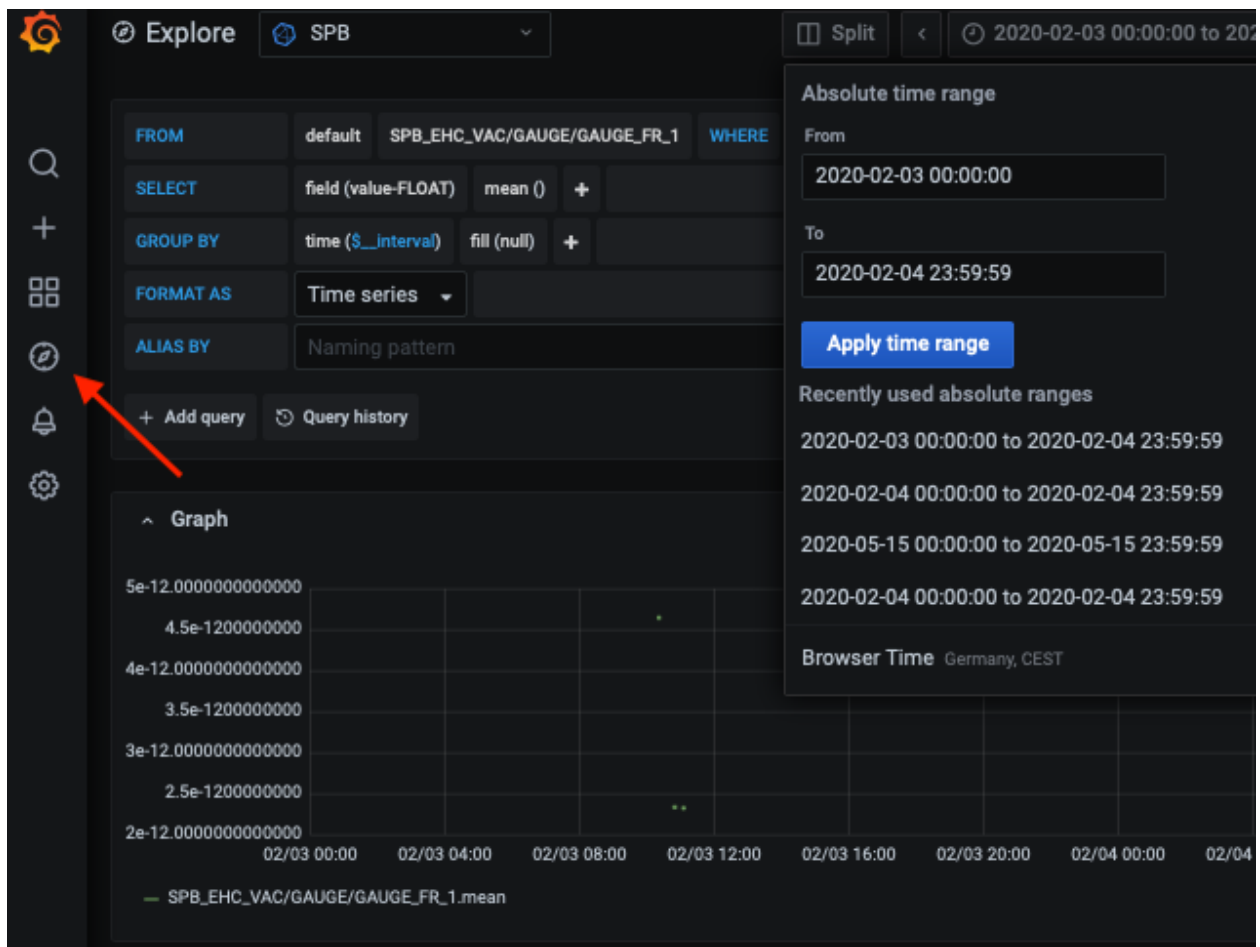
4.4 Examples

4.4.1 Quick Inspection of a Vacuum Incident

On the morning of February 4th 2020 the `SPB_EHC_VAC/VALVE/VALVE_ROUGH_1` which is part of the AGIPD vacuum system was found in an `UNKNOWN` state. To investigate we want to plot related pressure values from the archive.

Since initially we want to check what data might be useful we choose not to create a panel or even dashboard right away. Rather, we use the *Explore* feature of Grafana, which allows for ad-hoc investigations.

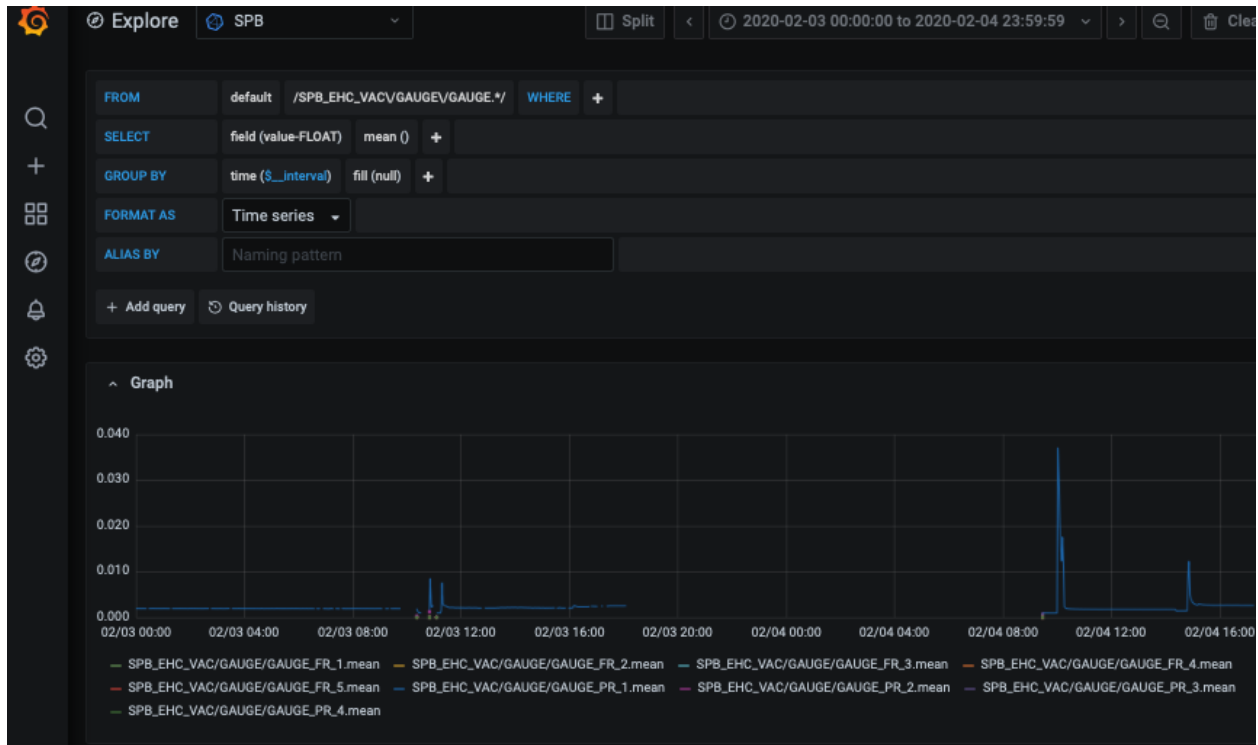
We thus click on the “Compass” icon in the menu, which opens an ad-hoc panel.



We choose SPB as *data source* as this corresponds to our Karabo topic. Since we are interested in the *EHC* component we look for a gauge there and select it as *measurement*. The pressure itself will be in the *value-FLOAT* field. Finally we adjust the time range to the period of interested, as should in the screenshot.

We now see the development of the pressure of `SPB_EHC_VAC/GAUGE/GAUGE_FR_1`. If we are interested in the other gauges tracking pressures in the *EHC* component, we could add more queries using the *Add Query* button. However, ****using a Regular Expression*** we can avoid this repetitive work.

Measurements can be given as regular expressions, so instead of `SPB_EHC_VAC/GAUGE/GAUGE_FR_1` we can write `/SPB_EHC_VAC/GAUGE/GAUGE.*/` which will plot the *value-FLOAT* field from all devices which start with `SPB_EHC_VAC/GAUGE/GAUGE`. Note how in InfluxDB a regular expression is enclosed in `/`. Also not that we need to escape the slashes in the device name, i.e. use `/`. Our plot will now look like



We now see that there were spikes in pressure and also that for a longer period in the night the vacuum system was in a state where no logging information was received. This might have to do with the observed *UNNOWN* state.

Hence, to recover when the detector powered down, we need additional information. We decided to check the logs of the power supplies which should trip in case of a vacuum incident.

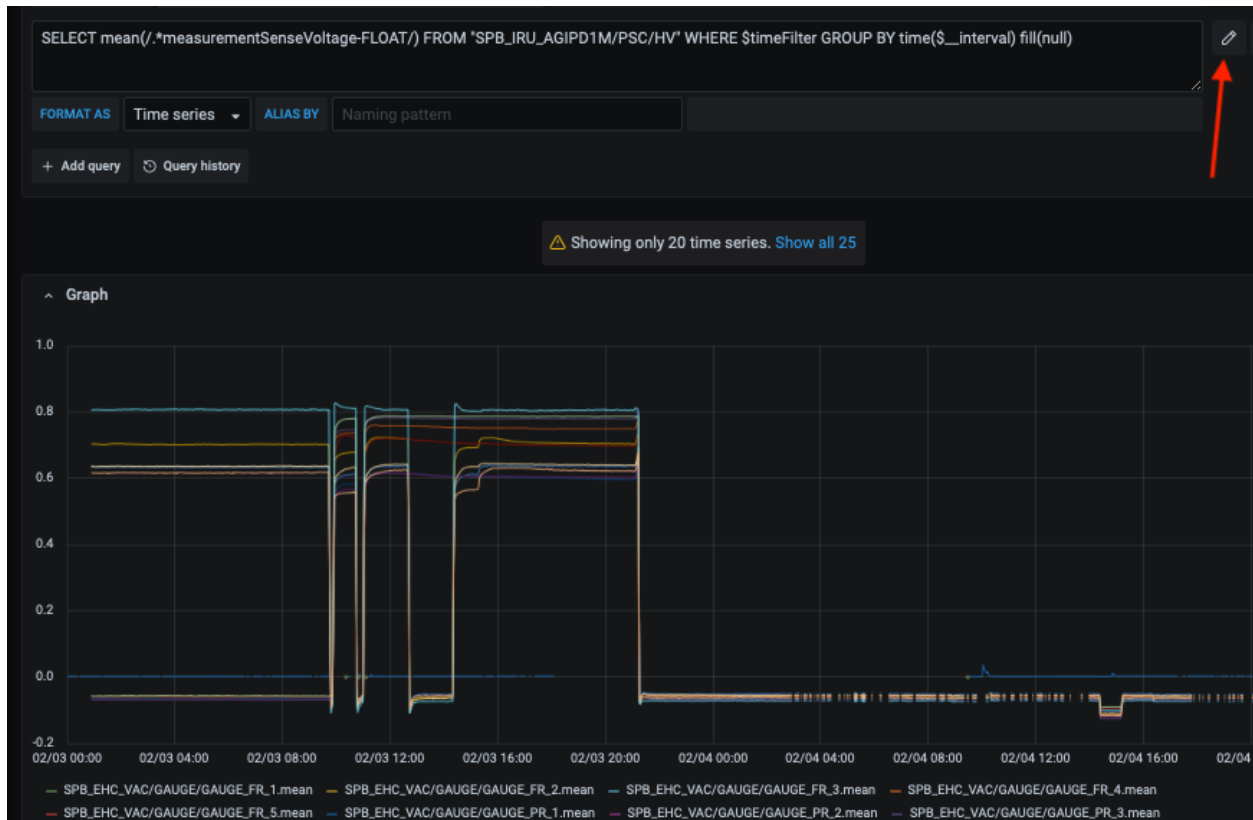
To do so we click *Add Query* and add the MPOD device controlling the high voltage of AGIPD: *SPB_IRU_AGIPD1M/PSC/HV*. The interesting values are the sense voltages, so we select this as field from channel 0: *channels.U0.measurementSenseVoltage-FLOAT*. This already shows us some correlation between the pressure behaviour and the voltage. Especially, we can see that the detector tripped or was powered down around 21.00 on Feb. 3rd.



To verify we decide to plot voltage trends for all HV channels. Again, we could add more queries manually, but decide to use a regular expression instead. For this we switch into textual query mode by clicking on the *pen* icon. This allows us to edit the query we've previously defined through the graphical editor:

```
SELECT mean(/.*measurementSenseVoltage-FLOAT/) FROM "SPB_IRU_AGIPD1M/PSC/HV" WHERE
↔$timeFilter GROUP BY time($__interval) fill(null)
```

Here the important part is the regex in the `mean()` function, which follows the same pattern as we used for the pressure value. The regex is enclosed in `/`, and via `.*` we allow any field to be evaluated which ends with `measurementSenseVoltage-FLOAT`.

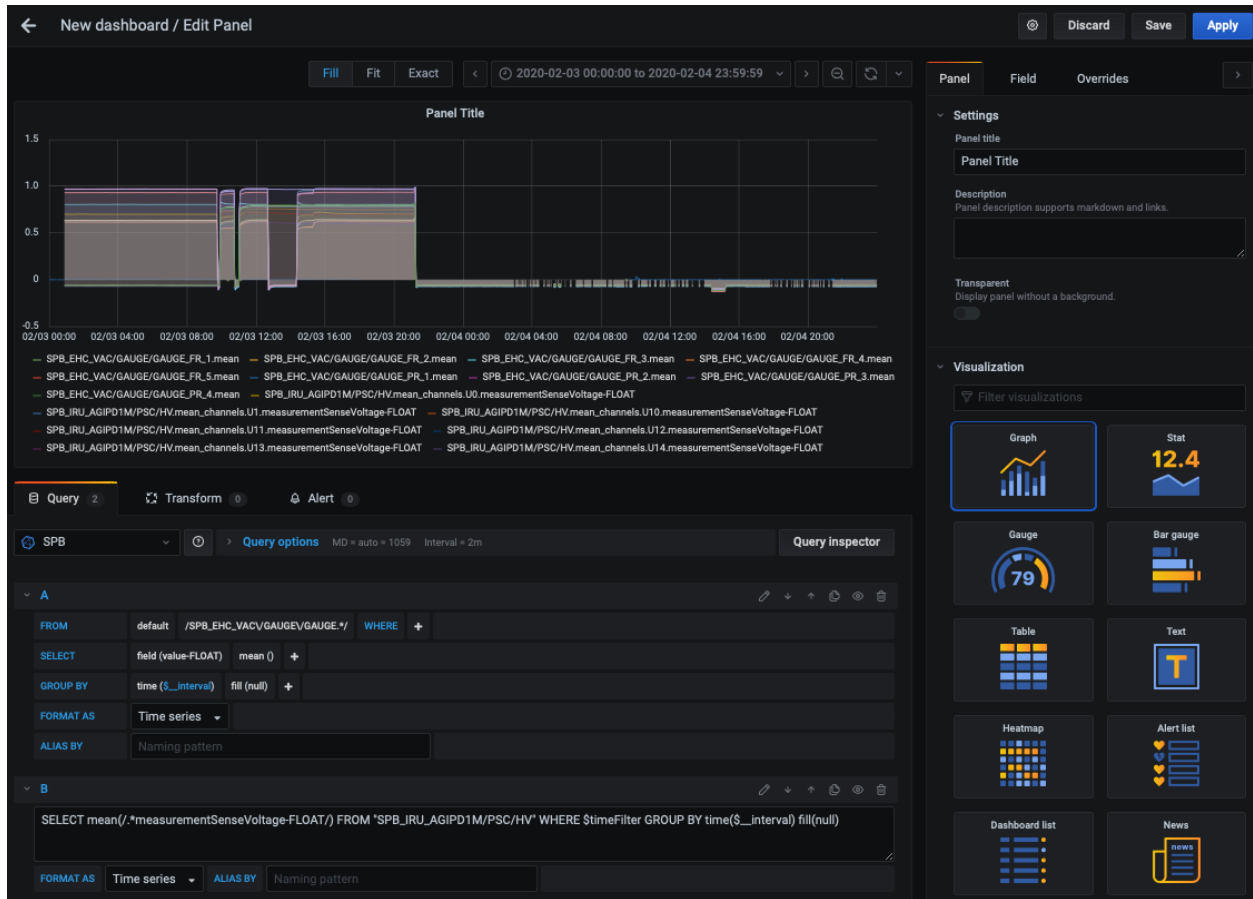


We can now see that all channels exhibited similar behaviour at the same time. However, we also observe that the value ranges of pressure and voltage are too different to display well on a single y-axis. Rather, two axes would be beneficial. We cannot do this in the *Explore* view, but since we consider the panel useful for future events we decide to create a proper dashboard with panel.

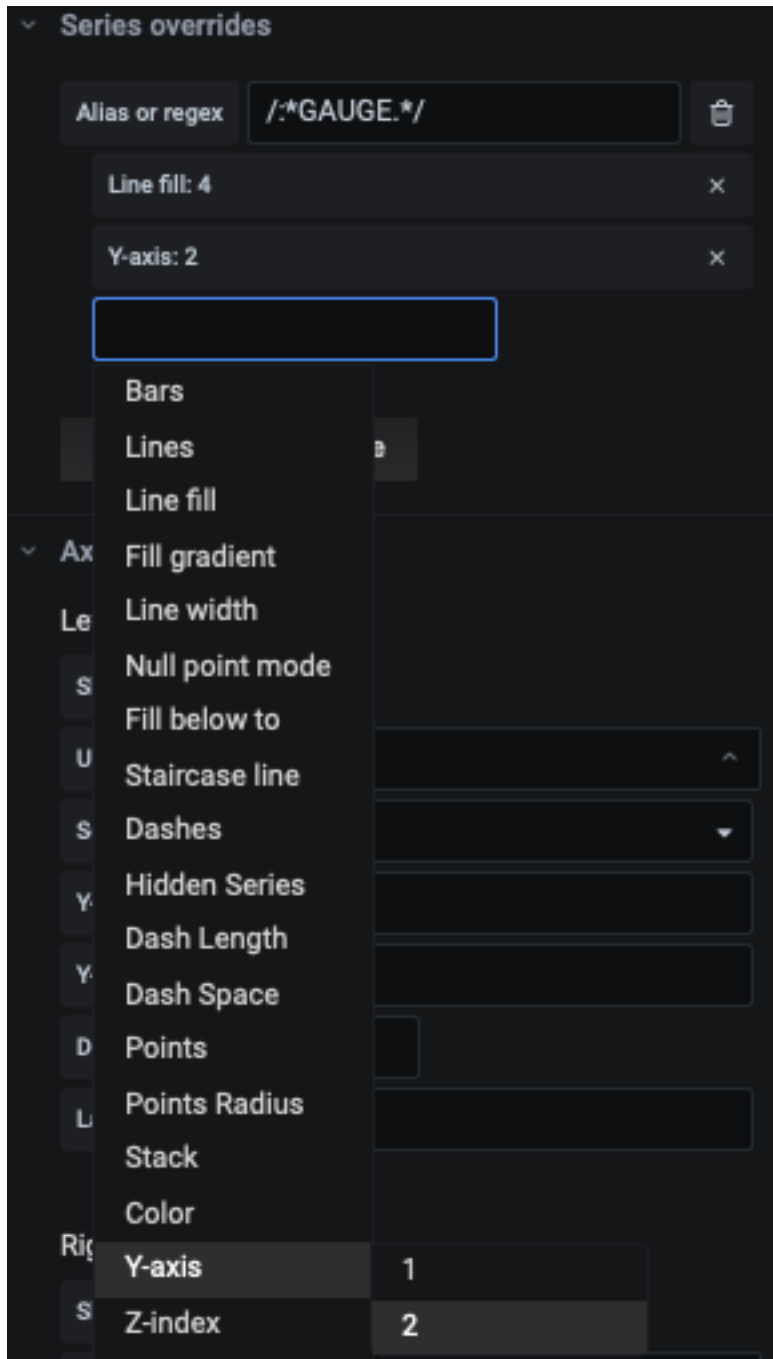
4.4.2 Using Two Y-axes

In this example we convert the preliminary investigation of the previous example, *Quick Inspection of a Vacuum Incident*, into a panel on a dashboard and adjust it to have two y-axes. This will allow us to display in the same plot values with divergent y-values.

We start by creating a new dashboard in the *Detector* folder. We then add a *New Panel* on the following screen, and enter the two queries from the previous example. Setting the time period to Feb 4th again, we should get something like the following:

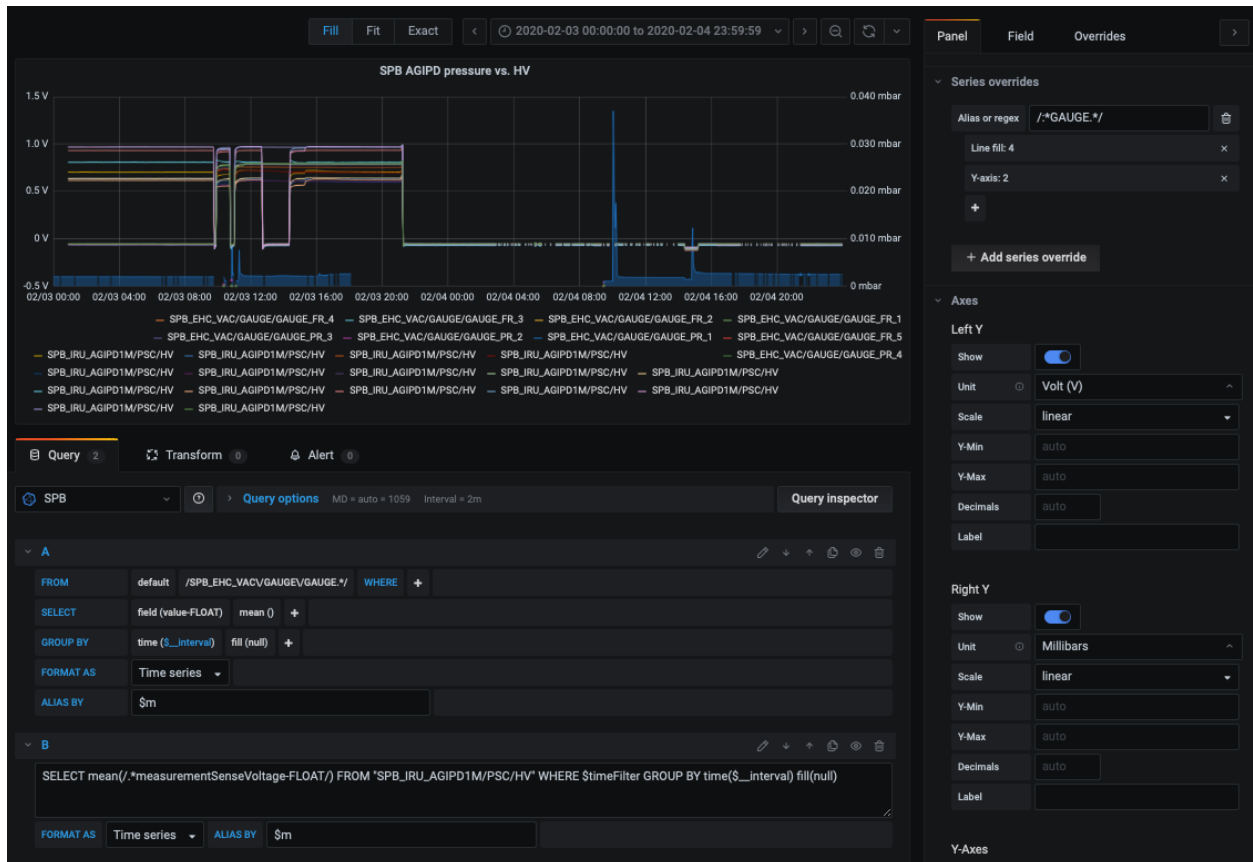


Our primary goal is to add a second y-axis to have a better display of pressure and voltages in the same plot. For this we scroll down the right *Panel* to *Series overrides* (<https://grafana.com/docs/grafana/latest/panels/visualizations/graph-panel/#series-overrides>). Here we can assign a separate y-axis to our pressure data. For this we identify the data which is to have its axis overwritten via a regular expression. We know the pressure device/measurement names will contain *GAUGE*, while the voltage measurements will not. Hence, `/:*GAUGE.*` is a suitable expression. We can then click on the + icon to add an override for the y-axis:



To further improve display, we make a few more changes: we set the line fill in the main options to 0 but override it for pressure with 4. Additionally, we add units to the Axes: *Energy*->*Volt(V)* for *Left Y* and *Pressure* -> *Millibars* for *Right Y*. Finally, we add *\$m* to the *ALIAS BY* field of our queries. This will shorten the legend entries to the measurement name only (<https://grafana.com/docs/grafana/latest/features/datasources/influxdb/#alias-patterns>).

Our plot should now look similar to the following:



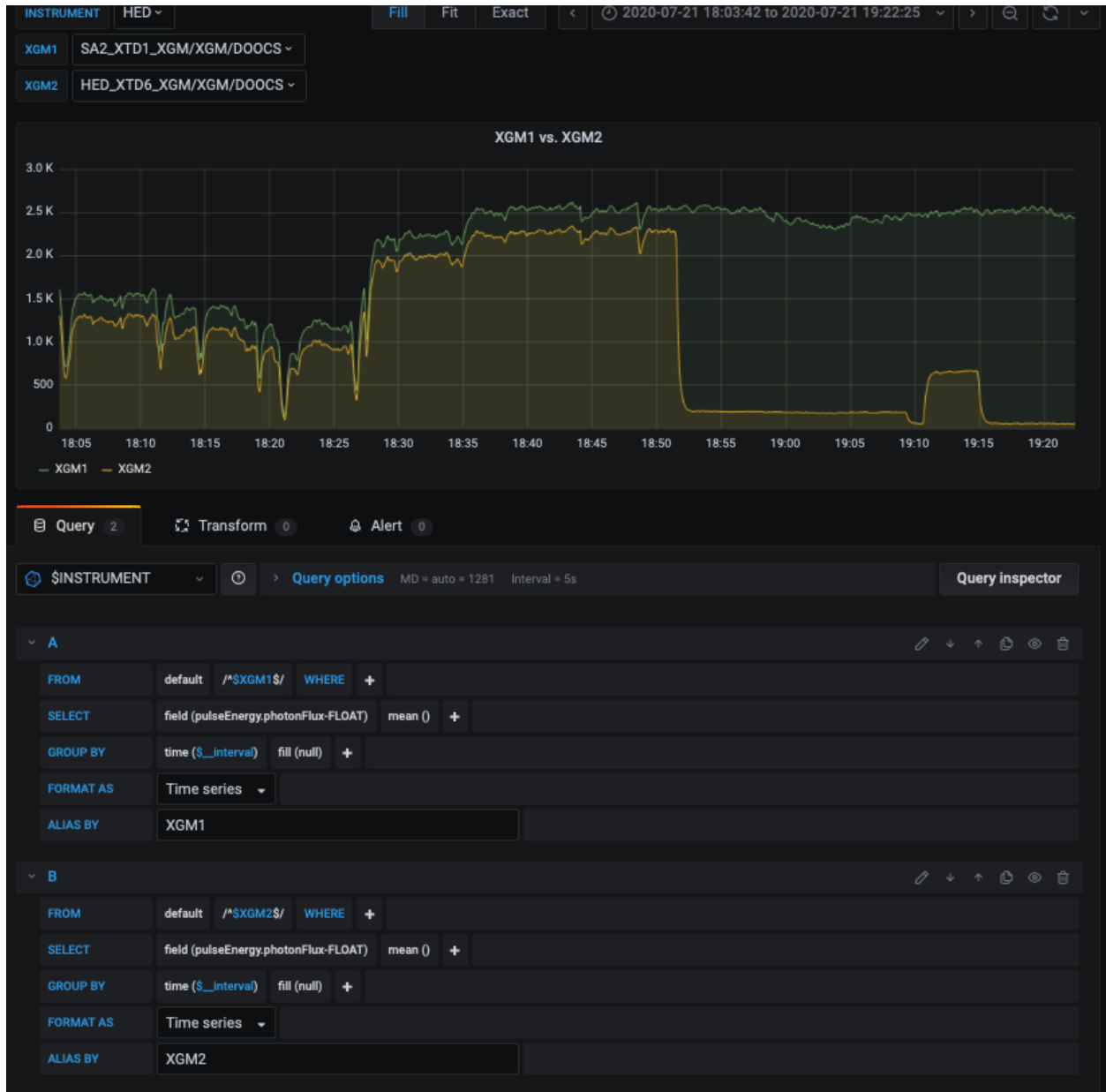
We could now parameterise the panel via Dashboard variable to e.g. make it useful for the MID AGIPD as well. See [:ref:'db_variables'](#) for an example on this.

4.4.3 Correlate two XGMS

In this example we correlate the intensities recorded by two XGMs in the same SASE. The data shown here is from an EPIX irradiation beamtime in August 2020. For this it was of interest how the *SA1_XTD1_XGM* and the *HED_XTD6_XGM* correlate in the photon energy they register per train. The panel we will be discussing in the following is available in *General->XGM correlation* and parameterised for all instruments (not all have two XGMs though). It looks like this for the time period we will evaluate in the example:



Let's recreate it step by step. We start by adding the relevant queries (see examples above). Note that in the screenshot these are parameterised via an *\$INSTRUMENT* variable referring to the database, and regexes on *\$XGM1* and *\$XGM2* variables referring to the measurements of the to-be-correlated XGMs.



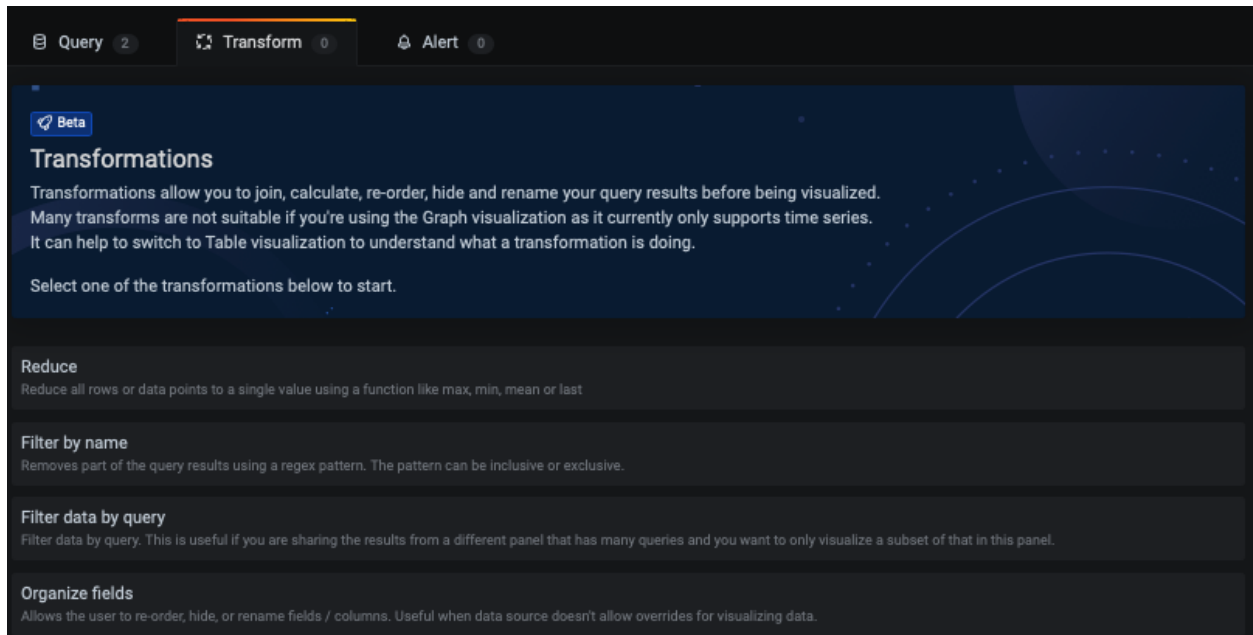
The regex is necessary as the `$XGM1/2` variable is defined as a query variable which shows all measurements matching an expression `/. *DOOCS.*XGM/`.

Variable	Definition
INSTRUMENT	influxdb
XGM1	SHOW MEASUREMENTS ON \$INSTRUMENT WITH MEASUREMENT = /. *XGM.*DOOCS\$/
XGM2	SHOW MEASUREMENTS ON \$INSTRUMENT WITH MEASUREMENT = /. *XGM.*DOOCS\$/

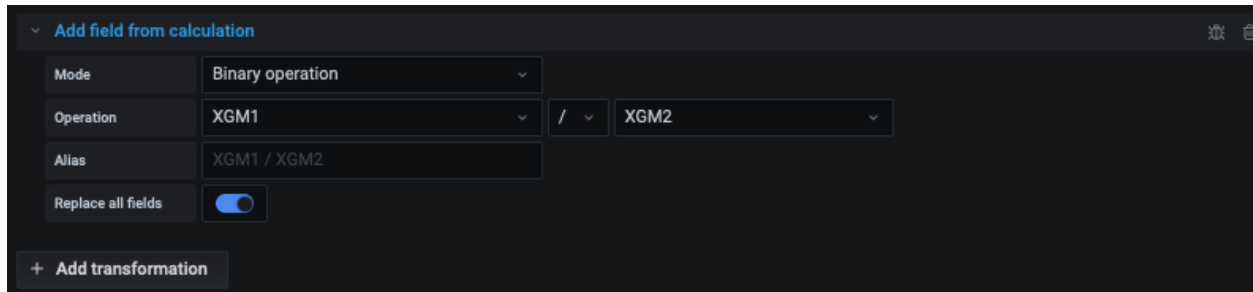
This would also match the `_events` and `_schema` internal measurements, which we reject by defining that there cannot

be any characters before or after the variable's content: `^$XGM1$`, where `^` is the start of the field entry and `$` its end.

Note that currently we simply have two series in our plot. However, we would like to have their correlation, i.e. in terms of XGM1's recorded energy with respect to XGM2's. We can do this using Grafana's **Transform** feature:



There are many different transforms to select from, which are documented here: <https://grafana.com/docs/grafana/latest/panels/transformations/>. Interesting for us is *Add field from calculation* which is described as *use the row values to calculate a new field*. We thus select this and the select *Binary operation* as *Mode*. We can now enter the operation as `XGM1 / XGM2`. Importantly, we select *Replace all fields* to hide the original data series and display only the calculated value:



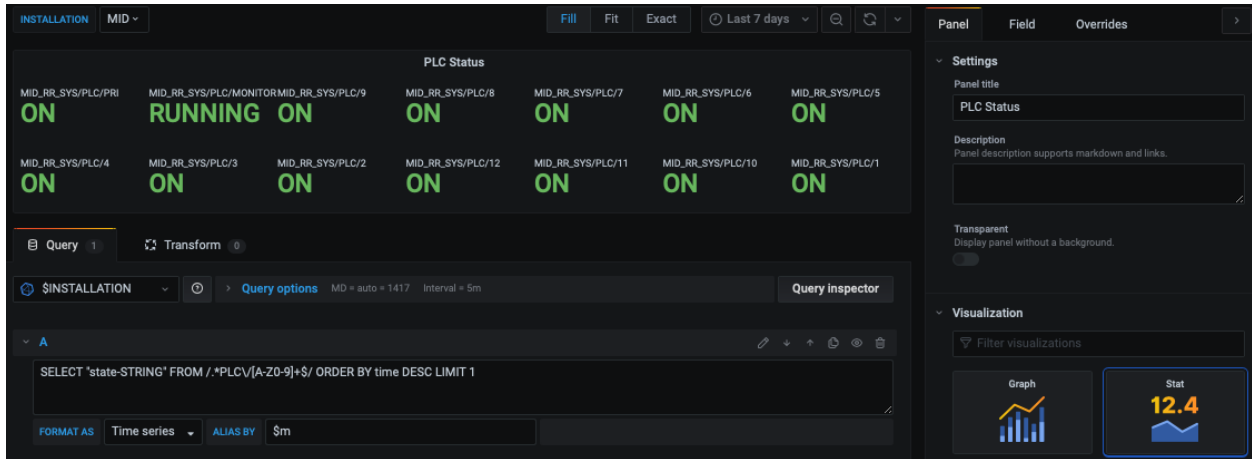
4.4.4 Displaying String Fields - Karabo Device States

For string fields many of the aggregate functions are not defined. Consider e.g. what the mean value of a string should be. However, there are a few useful visualizations Grafana can provide for string fields, and here most importantly Karabo device states.

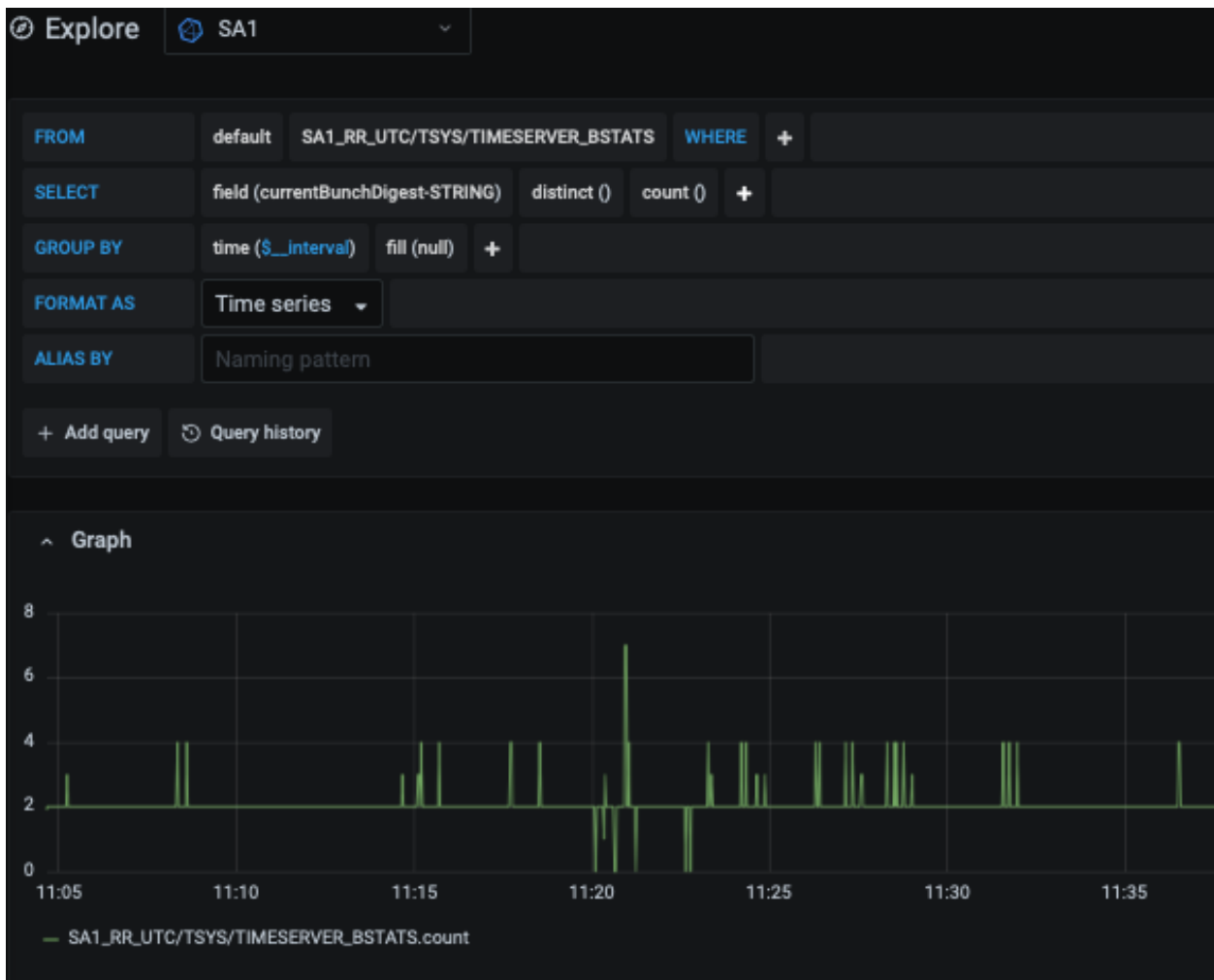
Displaying the current State of a device can be done via a query which orders by time and only retrieves the latest result, e.g.

```
SELECT "state-STRING" FROM /. *PLC\[A-Z0-9]+\$/ ORDER BY time DESC LIMIT 1
```

will retrieve the current state of all PLCs in a database/Karabo topic. A good visualization option for this is the *Stat* graph:



To **evaluate the number of state updates** a device has in a given time period, i.e. to check if a digital output is “flickering”, we can evaluate the number (count) of discrete values. This is also useful to check how often a bunch pattern changes, where each pattern is stored in a string-encoded hash:

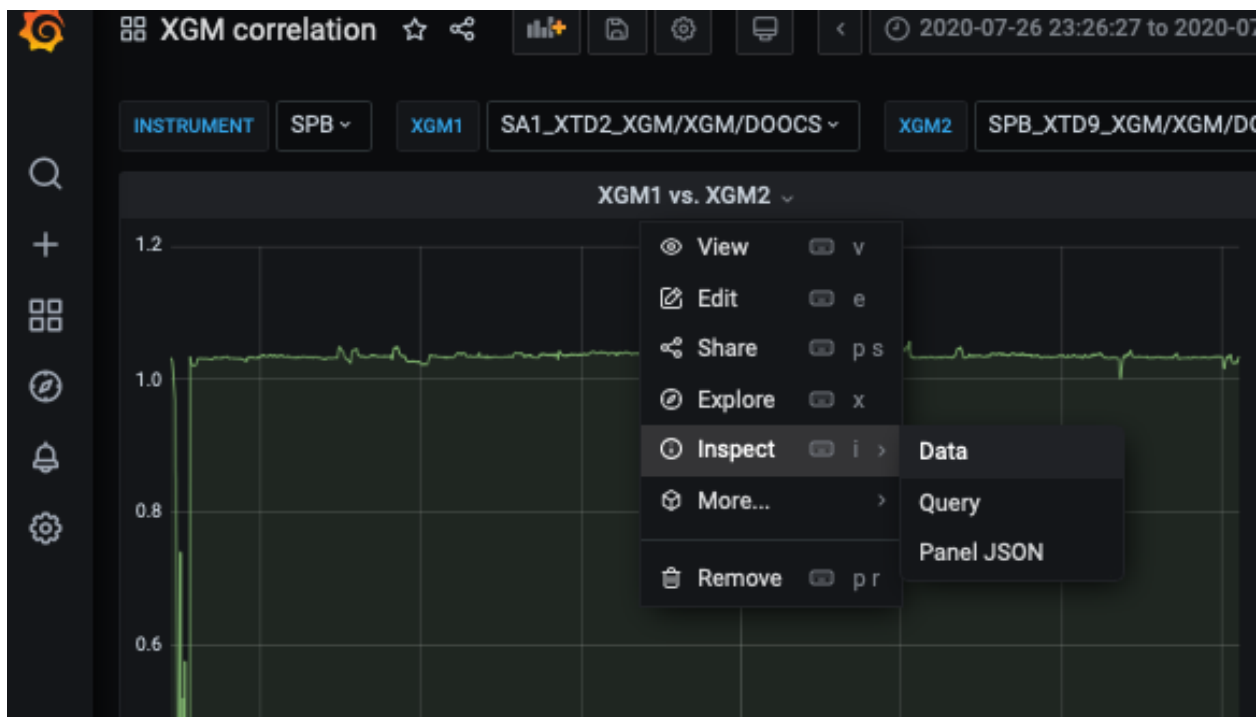


CHAPTER 5

Exporting Data

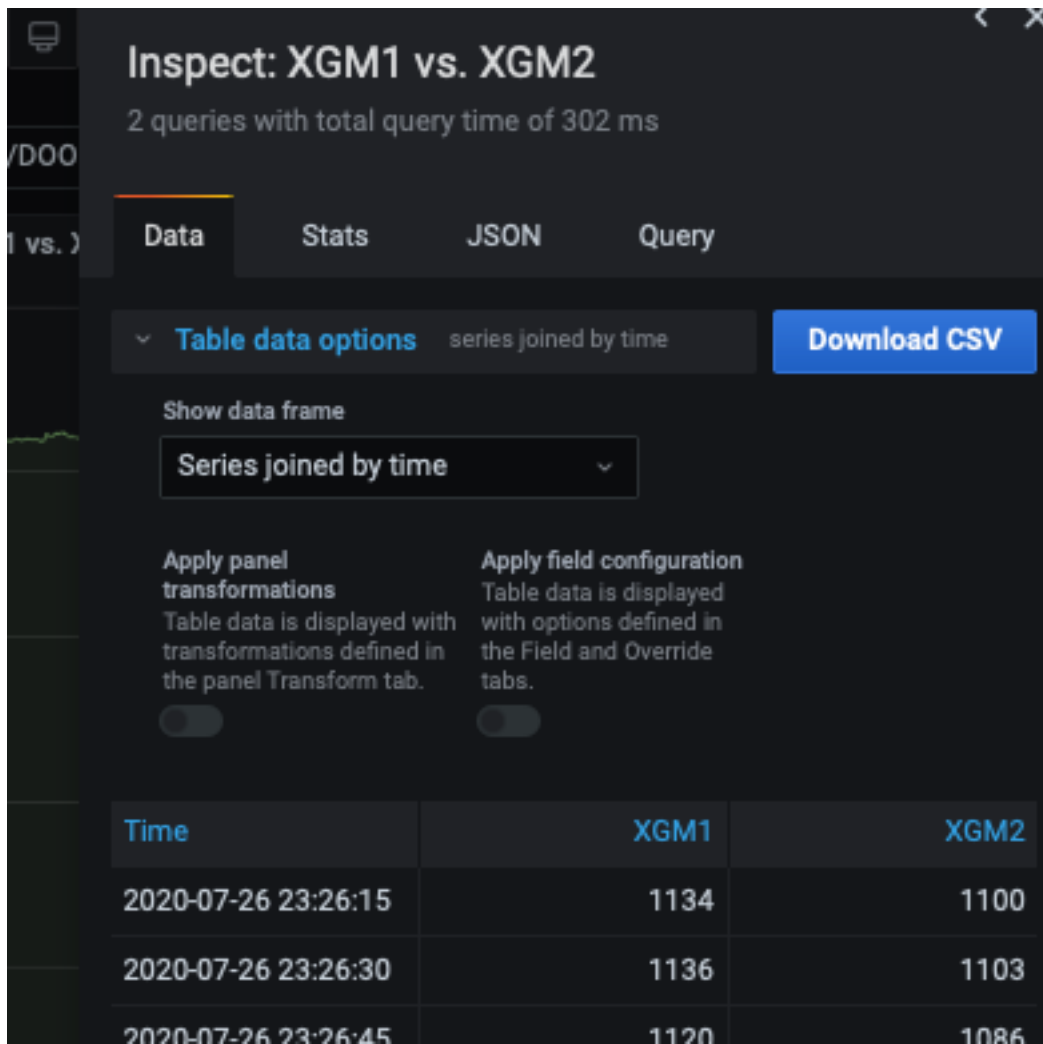
You can use Grafana to export data you are viewing into a comma separated value (CSV) file, which can be read in with most data analysis tools and environments.

In the panel containing your data click on the dropdown of the panel title:



This will open a context menu. You'll find the export option by clicking *Inspect->Data*.

A panel on the right side should now open giving you export options:



Inspect: XGM1 vs. XGM2
2 queries with total query time of 302 ms

Data Stats JSON Query

Table data options series joined by time Download CSV

Show data frame
Series joined by time

Apply panel transformations
Table data is displayed with transformations defined in the panel Transform tab.

Apply field configuration
Table data is displayed with options defined in the Field and Override tabs.

Time	XGM1	XGM2
2020-07-26 23:26:15	1134	1100
2020-07-26 23:26:30	1136	1103
2020-07-26 23:26:45	1120	1086

In the default setting all data series contained in your panel will be output to their own column in the CSV file. The left-most column gives a timestamp in *ISO* format.

You can also select only individual series to be output:

Inspect: XGM1 vs. XGM2
2 queries with total query time of 302 ms

Data Stats JSON Query

Table data options XGM1 [Download CSV](#)

Show data frame

- XGM1 (0) ^
- Series joined by time
- XGM1 (0) ✓ figuration
isplayed
efined in
verride
- XGM2 (1)

Time	XGM1
2020-07-26 23:26:15	1134
2020-07-26 23:26:30	1136
2020-07-26 23:26:45	1120
2020-07-26 23:27:00	1199

or toggle the *Apply panel transformations* switch to output data any *transformations* produce. In the XGM correlation example shown here, this switch needs to be toggled to export the data plotted in the panel, as it is based on a transformation:

The screenshot shows the 'Inspect: XGM1 vs. XGM2' panel in Grafana. It displays two queries with a total query time of 302 ms. The 'Data' tab is selected, showing 'Table data options' with a 'Download CSV' button. The 'Show data frame' dropdown is set to 'Series joined by time'. Below this, there are two toggle switches: 'Apply panel transformations' (checked) and 'Apply field configuration' (unchecked). The table below shows the data results:

Time	XGM1 / XGM2
2020-07-26 23:26:15	1.0
2020-07-26 23:26:30	1.0
2020-07-26 23:26:45	1.0

The CSV files generally then look like the following:

Such files can e.g. easily be imported in *Jupyter* via *numpy* or *pandas*:

```
from dateutil.parser import parse
import numpy as np
import pandas as pd

# import in numpy, converting ISO date time to timestamp
d = np.loadtxt("XGM1 vs. XGM2-data-as-seriestocolumns-2020-09-11 10_27_05.csv",
              skiprows=1, delimiter=",", converters={0: lambda x: parse(x).
              ↳timestamp()})

# import into a Pandas dataframe
df = pd.read_csv("XGM1 vs. XGM2-data-as-seriestocolumns-2020-09-11 10_27_05.csv",
                sep=',', header=0)
```

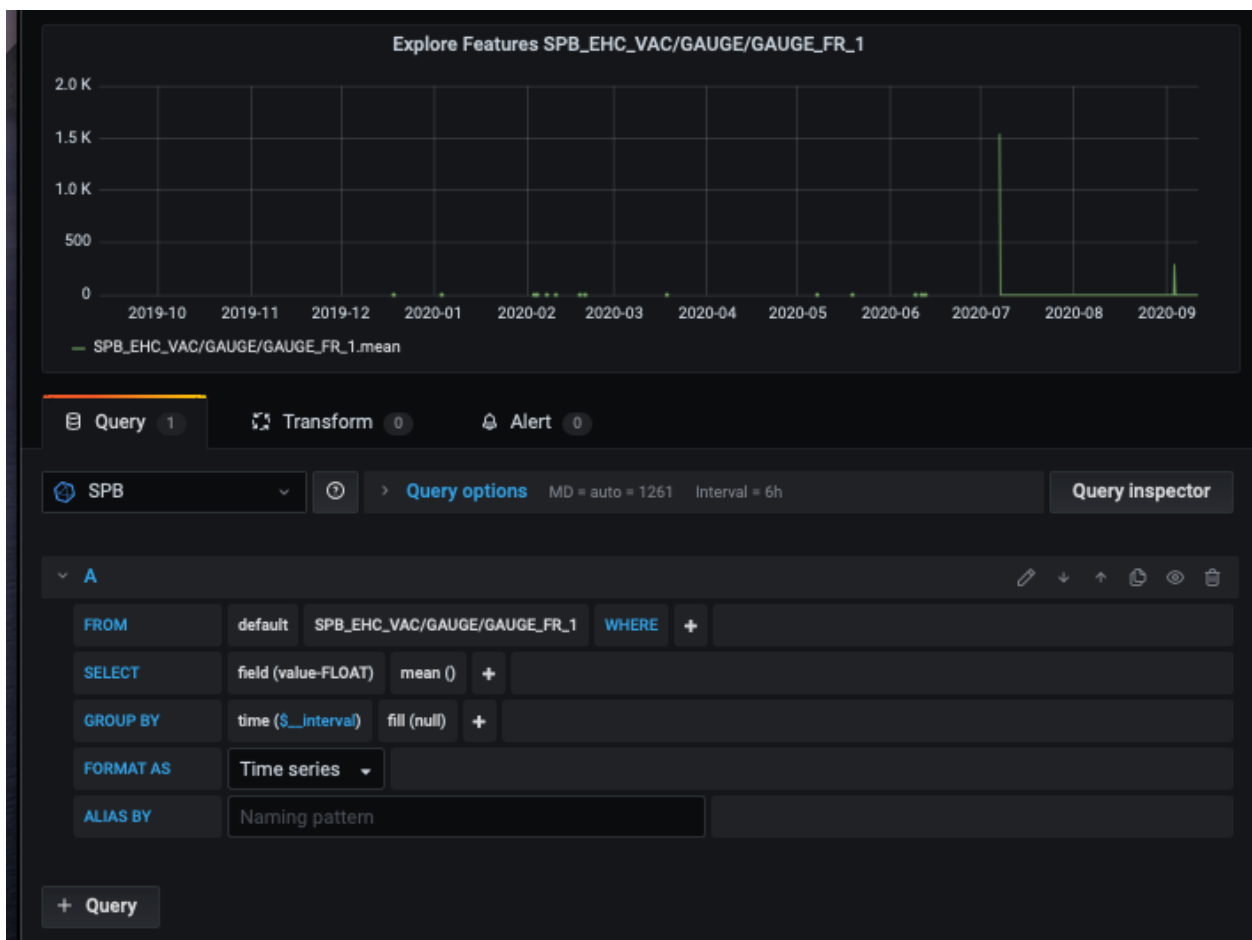
Note that in the pandas example time information is imported as string values.

Warning: Grafana has a timeout of 30 seconds on queries. Thus you might need to split large data series in multiple time periods if you want to export fine-grained long-term trends. However, consider using the InfluxDB inbuilt aggregate functions to calculate e.g. *min*, *max*, *mean* and *standard deviation* while binning data on larger time intervals.

5.1 Reducing and Selecting Data For Export

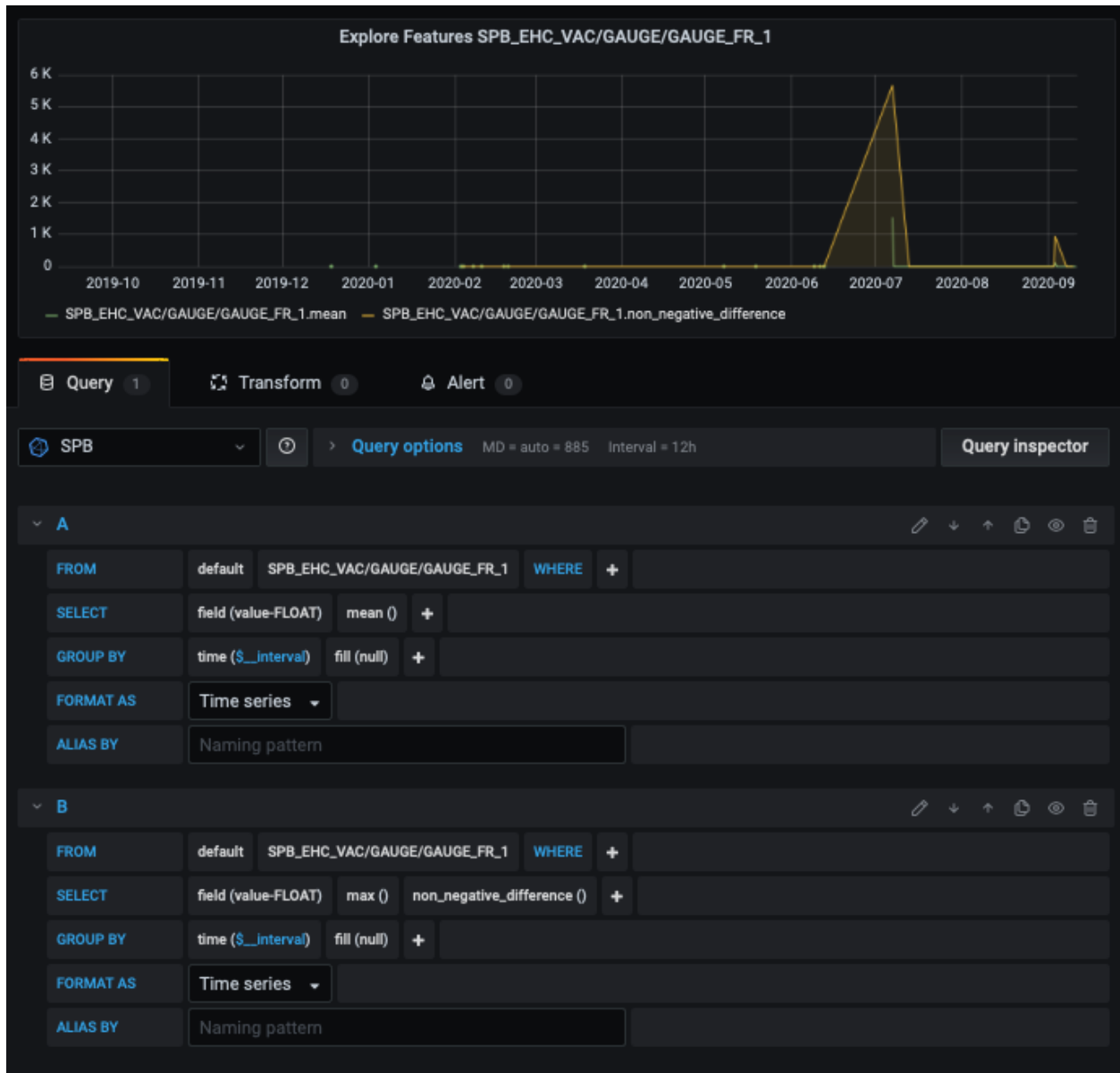
Before exporting large time periods of data it is often worthwhile to use Grafana to narrow down times of interest for a given data series. For this statistical aggregation functions can be used, which are provided by InfluxDB itself. A list can be found here: https://docs.influxdata.com/influxdb/v1.8/query_language/functions/

Consider for instance a gauge in SPB. We want to narrow down such time intervals, which show sudden changes in pressure. At first we look at the mean value of the gauge over the last 6 months:



We observe that there are areas without data and a few spikes. However, the data covers a wide range of values, so we are not sure if we actually see all “points of interest”.

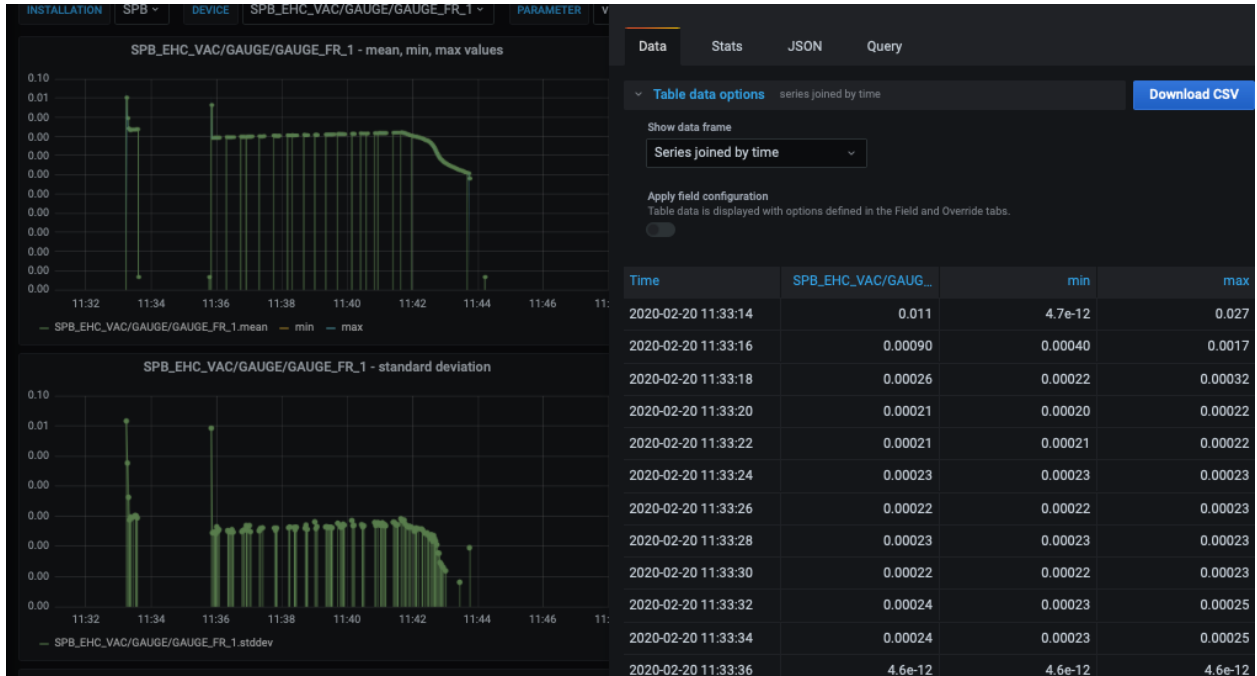
We can use aggregations to highlight sudden changes in value. In the example below we use the maximum non-negative difference of subsequent values. The peaks are now much more visible:



In the *General->Series Stats* dashboard quite a few of these statistical functions are aggregated into a single dashboard. It is parameterised to easily select a device and expected parameter from the dashboard interface. Furthermore, the y-axis scale is logarithms further highlighting changes:



For our gauge we e.g. see that for Feb 20 there is an interesting behaviour of the pressure. We can now zoom into this day and then export this data much more fine grained:



CHAPTER 6

Indices and tables

- `genindex`
- `modindex`
- `search`