CREATING GRAFANA DASHBOARDS WITH LIVE CONNECTION TO THE TIMESCALE AND CLICKHOUSE WSPR DATABASES

GWYN GRIFFITHS, G3ZIL

New in this version:

Updated links after moving to use logs1.wsprdaemon.org as the primary server. Several dashboards have been converted to use the Clickhouse tables, in these instances the Clickhouse code will be given in purple, and there are some new examples.

Incomplete revision, work underway.

Please send any comments or corrections to Gwyn Griffiths:
gwyn at autonomousanalytics.com

Contents

Alphabetical list of Dashboards in the WsprDaemon Community Folder

Clicking the Dashboards icon at left (the four squares) brings up Dashboard options, click Manage, then click the WsprDaemon_community folder and a list in alphabetical order will appear - as below. Click on the Dashboard you'd like to open. Dashboards in blue have no description in the text, but have an extended entry in this list. CH indicates those using Clickhouse

Dashboard in the WsprDaemon_Community folder	**Section**
2D scatterplots of SNR differences vs. distance and vs. azimuth (CH)	6.3
2D scatterplots two receivers same site SNR difference vs. distance and vs. azimuth (CH)	6.4
Decode cycles histogram (CH) from WsprDaemon table that logs decode cycles per spot	
Distance statistics - WsprDaemon table (CH)	3.3
Distance statistics - Wsprnet table (CH)	3.3
Heat maps of noise level, number of spots per hour and mean distance each hour (CH)	7
Merged receiver comparisons from WsprDaemon tables (CH)	5.2
Noise graphs (CH)	3.1 & 3.2
Propagation paths 'Telescope': Two bands with heading and beamwidth	8.2
Propagation paths 'Telescope': Two headings, same band and beamwidth	8.1
Scatterplots with distance, azimuth and time of the SNR at one station not heard by another and vice versa	6.5
SNR comparison between two reporters	4.1
Specific sender spot count and SNR comparisons	5.3
Specific sender SNR and signal level from WsprDaemon tables	5.4
Spot counts over 30 minutes between two grids (CH)	
Two receiver same site SNR time series comparisons	5.1

Part 1 How to build Grafana Dashboards and Panels

0. Installation and initial connection

Grafana is available for Windows, Mac, and Linux (including ARM) operating systems. Download and installation details are at:

https://grafana.com/grafana/download

There is no need to make any changes to the configuration options to run Grafana or to configure and install the dashboards covered in this note.

Connect to your own Grafana installation via a web browser to:

http://localhost:3000

Login with the default userid and password - both 'admin'. Then change your password.

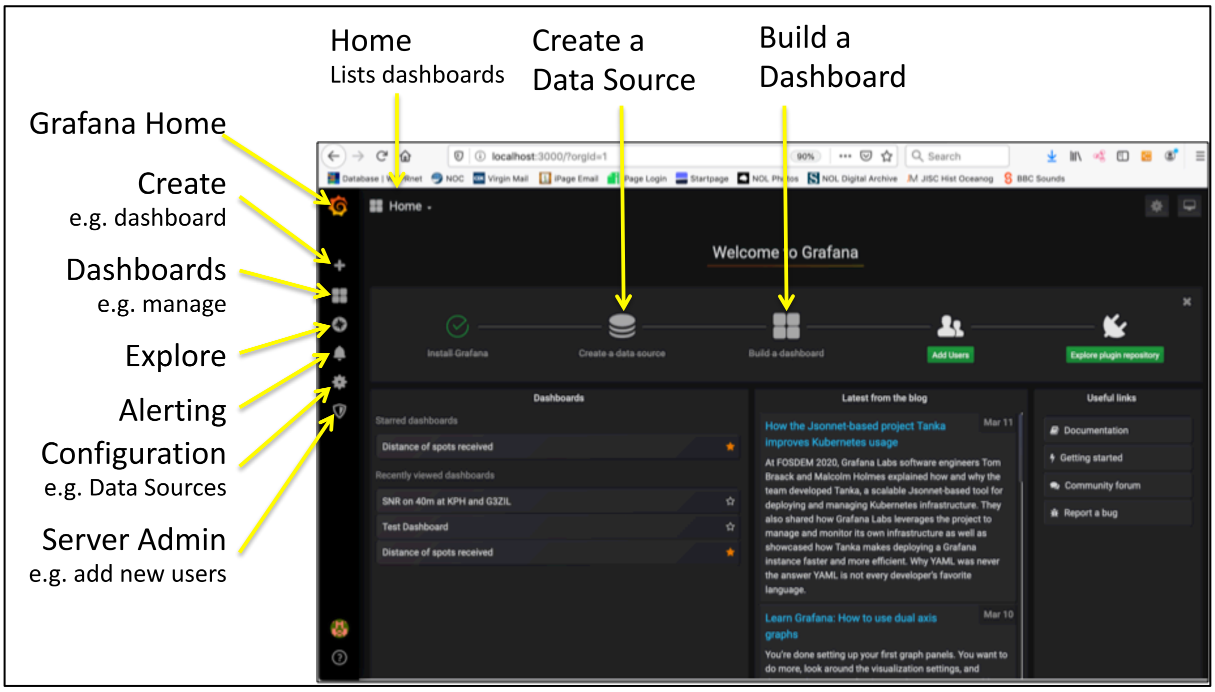
If, however, you would like to try before you install your own version you are welcome to try the Grafana installation at http://logs1.wsprdaemon.org:3000
where you can log in as user Open and password Open
User Open has access to the dashboards in this guide.

WsprDaemon users are welcome to use the Grafana installation at http://logs1.wsprdaemon.org:3000 where you can log in as user wdread and request a password (if not known to you) from gwyn at autonomousanalytics.com. You have permission to export the Dashboard JSON code to a file on your local computer, which you can then import to your local Grafana installation to inspect, edit and alter to suit your own needs and interests.

The following notes are for those with their own installation, but sections 2 onwards are applicable to those with wdread access to at http://logs1.wsprdaemon.org:3000.

On login you should see the Home screen, below, showing you have installed Grafana, although in this example we already have a data source connected and some example Dashboards.

Note: Grafana have recently been raising substantial funding and are likely to make changes to the product in quick succession, perhaps faster than the user interface details in this Guide can be updated. However, the underlying approach should remain the same (I hope!).



How the Jsonnet-based project Tanka improves Kubernetes usage Mar 11

At FOSDEM 2020, Grafana Labs software engineers Tom Braack and Malcolm Holmes explained how and why the team developed Tanka, a scalable Jsonnet-based tool for deploying and managing Kubernetes infrastructure. They also shared how Grafana Labs leverages the project to manage and monitor its own infrastructure as well as showcased how Tanka makes deploying a Grafana instance faster and more efficient. Why YAML was never the answer YAML is not every developer's favorite language.

Learn Grafana: How to use dual axis graphs Mar 10

You're done setting up your first graph panels. You want to do more, look around the visualization settings, and

1. Create a data source

From the screen above (click Grafana Home if you have explored some of the other screens), click on Configuration and then Data Sources. Click on Add data source. Scroll to PostgreSQL and select, bringing up the Settings page.

Give the data source a name, e.g. wsprdaemon_tutorial.

Enter the Host as logs2.wsprdaemon.org:5432

Enter Database as tutorial

Enter User as wdread and password as JTWSPR2008

For SSL Mode select disable

Leave Connection Limits as they are

Scroll down to the section PostgreSQL details (on my browser the scroll bar on the right is both very thin and a shade of grey just off-black making it hard to see).

For Version select 10 in the drop-down list (if 11or 12 appears, select it instead, PostgreSQL is also developing new versions.).

Slide the TimescaleDB switch to the right. Leave Min time interval at 1m.

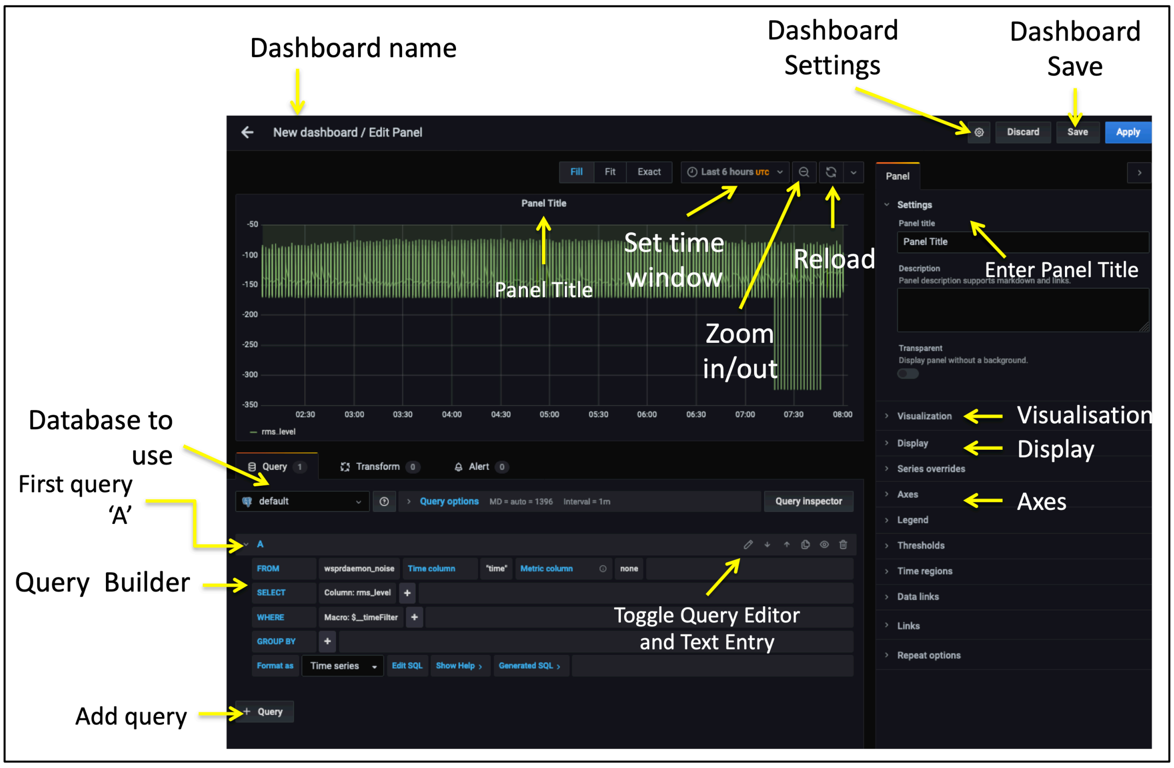
Scroll to bottom of the screen and click the green Save & Test button. All being well you should see a green banner with a tick and Database Connection OK message.

2. Create a dashboard

Hover over the Create icon at top left (big plus), click on Dashboard,

2.1 Create the first Panel.

Click on Add new panel, which will bring up a screen similar to that below. From Version 8 of Grafana you will first click on the icon Add a new panel. (Reminder: frequent changes are being made to the User Interface).



Panel title: Panel Title

Description: Panel description supports markdown and links.

default | Query options MD = auto = 1396 Interval = 1m

A

FROM wsprdaemon_noise | Time column "time" | Metric column none

SELECT Column: rms_level

WHERE Macro: $__timeFilter

GROUP BY

Format as Time series

The upper part of the screen shows a default time series graph - a Panel in Grafana jargon.

The lower part is a form-filling query builder to collect data from, in our case, a postgreSQL database with TimescaleDB extensions. At right (new in version 7) is an extensive list of option headings to name and then customise the graph. The important Save button is top right - too close to Discard for me! There is also Apply, which applies but does not save, your changes.

The query builder is ready for Query 'A', others can be added later. As a first example, let us plot the SNR for ON7KB at GM0UDL on 40m:

First, under Query, pull down the data source name e.g. wsprdaemon_tutorial.

On the FROM line click default and from pull-down options select wsprdaemon_spots_s.

Leave Time column as "time" (or pull down and select if not shown). After Metric column select tx_call from the pull down menu.

On the SELECT line, it may already show Column: "SNR", if it does not, click on whatever text is after Column: and select "SNR" from the pull-down.

If the WHERE line shows a Macro entry, click it until it shows remove, then click remove. Click on the + to the right of WHERE, in the empty box, click and you may see a pull down list, or you may not, type E and it may come up with the option Expression, if not type it. At this point a red error banner may come up, that's ok, we've not set values yet. Click on the first value and select rx_id (may not always 'take', if so try selecting again, if still nothing, type in GM0UDL, the single quotes are needed). Click on the plus again, and in the same manner, click and select in turn Expression: band = '40'. Click on the plus again, and in the same manner, click and select in turn Expression: tx_call = 'ON7KB'.

At this point, data should appear. If there is no data, try zooming out in time using the zoom out icon (magnifying glass with -). If still no data is shown, try other calls known to be active currently on 40 m WSPR.

Save your work so far by clicking on the disk icon to Save Dashboard. Give the Dashboard a suitable name, e.g. SNR at G3ZIL, noting that we'll be adding further Panels to this Dashboard and so the Dashboard name should be appropriate.

2.2 Add a second (and subsequent) time series to this Panel.

After a save, we will have left the Query Builder, so click on the Panel Title then on Edit, which will bring back the Query Builder. We'd best give the panel a title, e.g. SNR of ON7KB and AJ8S, in the box Panel title under Panel Options at top right.

Click on the + Query button at bottom left, which will bring up a query builder for query 'B'. Of course, this second query should be along the same lines at the first, e.g. still SNR at GM0UDL on 40m but, say, for 'AJ8S'. Repeat the steps above to build the query. Click Save.

2.3 Customise the Panel.

At top right right is a pull-down for different types of graphs - for this example we will stick with Time Series the default.

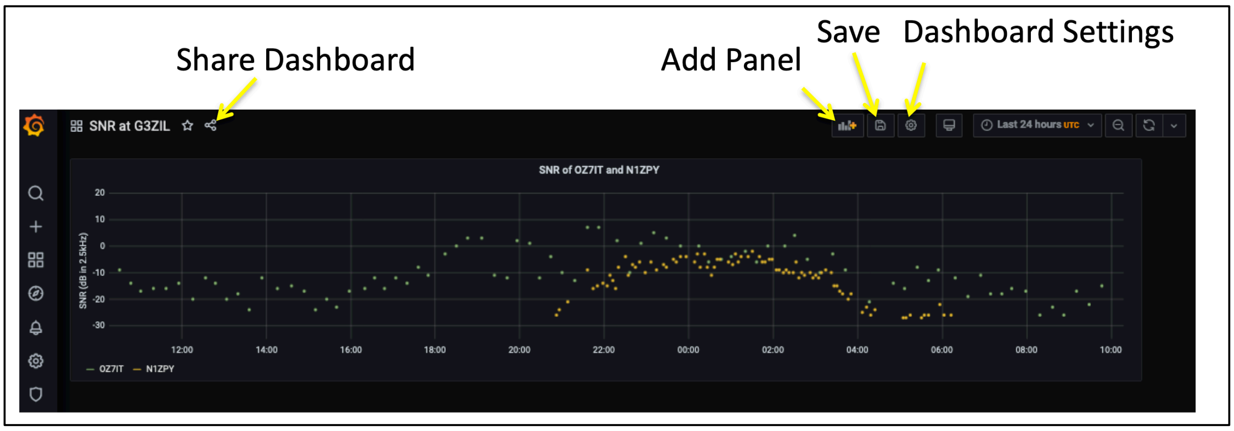
Click against the Panel options, Tooltip and Legend pull-downs to close them, leaving Graph Styles open. Chose your preferred option of lines, bars or points for the graph, (you might want to use points rather than lines for this type of graph), if so, select the points switch. For Point size you might set it to 5. Close the Graph styles pull-down.

On the Axis pull-down, leave placement as Auto. You should also enter a Label for the Y-axis, e.g. SNR (dB in 2.5kHz). There's no need to alter other settings for this Panel. Apply the changes (top right). You can resize the panel by click dragging its bottom right corner. Save the revised Panel.

2.4 Add a second Panel

Click on the Back Arrow at top left. It may leave you at the dasboard you've created or it may present you with a list, in which case select the one where you want to add a panel. This brings up a few other icons as in the screenshot below.

Click on Add Panel (the bargraph icon), then + Add new panel button, and the process is essentially as described for the first Panel. However, we can demonstrate here some additional graphical forms within Grafana, in this case a bar-graph grouped by time.



The theme is still 40 m at G3ZIL for the WHERE clause, but for this panel we will show the number of spots in each 10 minute interval.

Under Query select wsprdaemon_tutorial as the data source.

On the FROM line select wsprdaemon_spots.

On the SELECT line click on the + and you should see as an option Aggregate functions, and to its right, count. If you don't see Aggregate functions, type A and it should appear. Data should be seen in the new Panel at this stage. Column:"SNR" should already be present, and then Alias: "SNR" should automatically appear after count is selected, click on the "SNR" after Alias and over-type the proper descriptor, Spots in 10 minutes, which should then appear as the text alongside the key at the bottom left of the graph.

On the WHERE line, as for the first panel, remove the Macro, add Expressions for rx_id 'G3ZIL' and band '40'.

On the GROUP BY line click $-interval in brackets after time, and select 10m (i.e. 10 minutes).

Click the Display pull-down at right, and you may want to change to Bars rather than Lines..

On the Axes pull-down, for the Left Y axis enter a Label such as Spots in 10 minutes.

Top right, enter a Panel title such as Spots in 10 minutes.

Click Save.

Click on the back arrow at top left, selectthe Panel to resize by click drag on bottom right, adjust time range to suit. If you want to reorder the two panels, click and hold the title bar while you drag. Click the disk icon at top right to save.

2.5 Add a third Panel.

Grafana has a simple form of 'heat map' graph format. We can use it, for example, to show the variation with time of the spread of distance of received spots. Following the Add Panel procedure above for the FROM line, select wsprdaemon_spots. On the SELECT line click the Column variable and select 'km'. On the WHERE line click the Macro: option and select remove. Click the +, select Expression, then select 'rx_id' and then type in 'G3ZIL', click the +, select Expression, then select 'band' and then select or type in '40'. If there is an entry on the GROUP BY line, select the function and select remove.

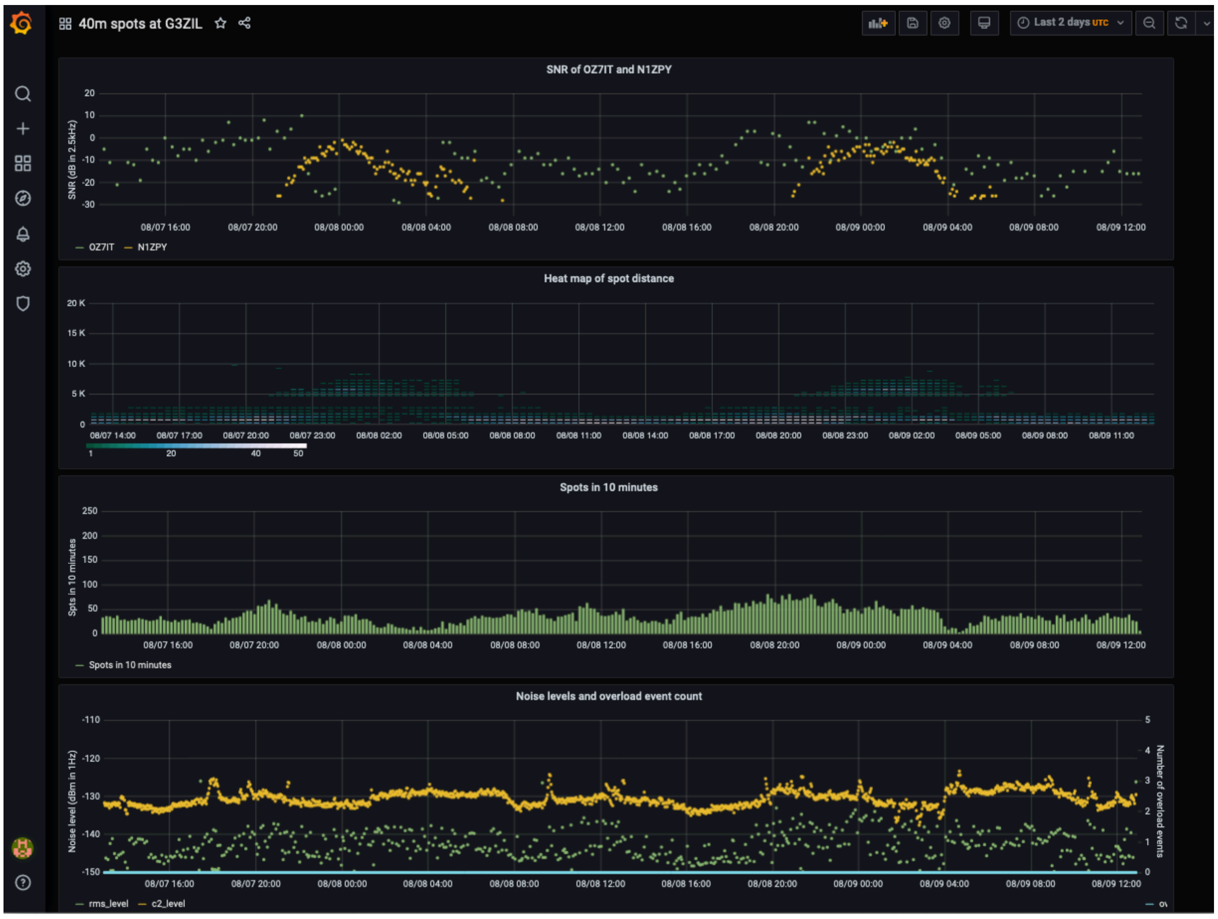
On the Visualisation pull down at right select Heatmap. Close the Visualisation pull-down and select the Axes pull-down. You may want to set a suitable Y-min and Y-max. Under the Buckets column you may want to experiment to set the number of buckets for the x and y axes by number or by Size (e.g. in this case km for y, perhaps 500, and perhaps 20m for x - 20 minutes). Close the Axes pull-down and open the Display pull-down. Under Scheme - try different colour scales to use. And on the Color scale boxes set appropriate min and max, or leave as auto. To right of Legend select Show Legend option for a scale bar. Close Display and open Settings to give the Panel a title. Click Save.

2.6 Add a fourth Panel

From wsprdaemon version 2.8a RMS and c2_FFT noise estimates are sent to the Timescale database on the wsprnet.org server. Noise graphs can therefore be presented alongside spots data. Following the steps as above, but on the FROM line select wsprdaemon_noise as the table. On the SELECT line, Column: rms_level may already be present, click +, select Column and then select c2_level. We can also show the overload flag instances, add Column: ov. On the WHERE line click Macro: and select remove, click +, then Expression, band and '40', click +, then Expression, site, 'G3ZIL'.

Open the Display pull-down at right, under Draw Modes, Points are best, and because of the 2-minute interval data, select Point Radius 1. Close the Display pull-down and open Axes, you will want to set limits on the Left Y axis, and add a Label e.g. Noise Level (dBm in 1Hz). Click on the line next to ov on the graph legend at left just under the plot. Click on the Y-axis tab and move the slider to the right to use the right-hand axis for the overload values. On the Axes pull down on the right, set Decimals to 0 and add a Label such as Number of overload events. Close the Axes pull-down, open Settings and give the Panel a title. Click the back arrow at upper left, resize the panels if need be, rearrange them, click the disk icon top right to save.

The Dashboard we have created should look like the screenshot below. Having gone through this tutorial for three different graph forms and using both left and right Y axes you should now have enough working knowledge to create your own simple Panels and Dashboards.



2.7 Share, export and import the Dashboard and Panel csv data

Share and export the Dashboard: To the right of the Dashboard name at the top left is the Share Dashboard icon (three linked circles). The Export tab brings up a 'Save to file' option; slide the 'Export for sharing externally' option and click 'Save to file'. This will save the JSON code for this Dashboard in a file on your local computer. You need to manually close the Share window.

On your local computer's Grafana installation you can then click the Create icon (plus sign at left) and select Upload JSON file, select your file, and under Options rename if required, select Change uid and edit the uid, e.g. by adding your initials to the end, then select your local name for the data source in the pull-down box, the click Import.

Share and export csv data from a Panel: This route has changed from Grafana 6 and is not via the share / export path described for Dashboard JSON code above. Rather, for the panel for which you wish to obtain csv data, click its edit symbol (down arrow to right of its name). Hover over option Inspect to bring up the option Data. By default, under the Data tab you will see a data list and a Download CSV button that will download to your local computer. Interestingly, the data list and the download can be for a longer time interval than that requested in the Dashboard - no wonder the query time can be long; you can check the time taken in the Stats tab.

2.8 Change Dashboard Settings

Click the Dashboard Settings (cog) icon at the top of the screen.

Under Time Options you can select the Timezone to be determined by the local browser time or UTC. I suggest using UTC.

For Auto-refresh set a value such as 10m.

For Now delay now set a value of 1m to ignore last minute, which may have partial data.

Click Save on the left. You could also click Save As to use this Dashboard as a template for others, e.g. for different bands.

Click on the back arrow to get back to the dashboard.

3. Creating a Panel with pull-down selections

The Dashboard and panels described above have had the variable field selections to plot 'hard-coded' into the Query Builder and hence the plots themselves. Using the Template and Variables features in Grafana we can construct panels with pull-down selections to enable general-purpose plots, along the lines of the original Grafana noise plots produced by Tommy Nourse, KI6NKO. Our example will be a noise measurements panel.

3.1 Pull-down selections within the WHERE clause

On opening Grafana click the + at left and select (to create a) Dashboard. Click on + Add new panel. Click the Dashboard settings cog icon at top right, give the Dashboard a name; in Time options set Timezone to UTC, Auto refresh to 10m and Now delay to 1m.

On the menu list at top left click on Variables, click on Add Variable. For the noise example we need three variables: Site, Receiver and Band, we'll call then site_name, receiver_name and band_m. Under the General heading, enter the Name - site_name. Under Query options, Data source, select wsprdaemon_tutorial, for Refresh, select On Dashboard Load.

The next step is to code the SQL query that will generate the list of site names for the pull-down list. In the Query box enter:

```
select distinct site from wsprdaemon_noise;
```

From the noise table this selects the distinct (unique) site names.

Set Sort as Alphabetical (asc), move the Multi-value slider to the right, to select more than one site to plot. You'll see a preview list of unique site names. Click Add then Save. Click on Duplicate at right against the site_name entry. Click on the copy, rename to receiver_name.

Edit the Query box to read:

```
select distinct receiver from wsprdaemon_noise where site in ($site_name);
```

As well as selecting distinct receiver names, we are specifying only those receiver names applicable to the sites we have already selected[1]. Check the Multi-value slider is to the right. You may well see error messages in a red box, ignore them. Click Update.

Click on the large type Variables at the top. Click Duplicate on the receiver_name line, click on the copy line, as the Variables>Edit page comes up, change the Name to band_m and edit the query box to read:

```
select distinct band from wsprdaemon_noise where site in ($site_name);
```

[1] This helps minimise confusion as there are many receiver names in the list with varying styles.

Set Sort order to Numerical (asc). Click Save dashboard at left. Click the Go Back left arrow, top left. Click the Panel Title and select Edit, which will bring up the Query builder. Select wsprdaemon_tutorial as data source. For Metric pull-down select receiver. On the Select line click the + and select Column, click to select c2_level if a second rms_level has appeared. On the WHERE line click Macro and remove, click the + then select Expression, pull down the first value to be site, and for the second value, select $site_name (i.e. the variable, and not a fixed name). Click + again for Expression, receiver, $receiver_name, + again, for Expression, band, $band_m. Error messages may appear, ignore for now.

In the Format as line, click on Edit SQL, the SQL may well look like the following:

```
SELECT
  "time" AS "time",
  receiver AS metric,
  rms_level,
  c2_level
FROM wsprdaemon_noise
WHERE
  site = '$site_name' AND
  receiver = $receiver_name AND
  band = $band_m
ORDER BY 1,2
```

There is a problem here - the Query Builder hasn't formed the query correctly for postgreSQL, the use of "=" in the WHERE clause is appropriate for a specific site, receiver or band, but not for a list. For a list the correct syntax is WHERE site IN ($site_name); so IN instead of =, and parentheses instead of single quotes.

In addition, we can expand the metric so that the legend has site, receiver name and band. So edit the SQL to read:

```
SELECT
  "time" AS "time",
  (site, receiver, band) AS metric,
  rms_level,
  c2_level
FROM wsprdaemon_noise
WHERE
  site IN ($site_name) AND
  receiver IN ($receiver_name) AND
  band IN ($band_m)
ORDER BY 1,2
```

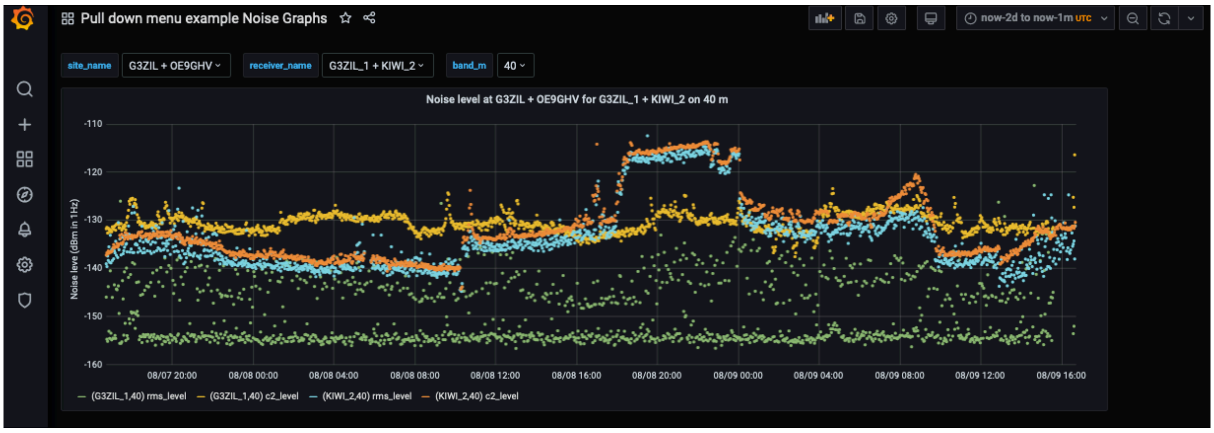
We should now see the three pull downs properly populated with options and some data graphed. Open the Display pull-down at right, select points rather than lines, and set Point Radius to 1. Close Display, open Axes, add Left Y axis label: Noise level (dBm in 1Hz).

Close Axes, open Settings, give the Panel a title, you can refer to the variables, e.g. :

Noise level at $site_name for $receiver_name on $band_m m

Click the disk save icon at top, saving current time range and current variables.

If your expected pull down options do not appear, click on the graph panel, or the refresh icon at far top right and wait a few seconds. Here is the finished panel displaying data from OE9GHV and G3ZIL on 40m for two days.



Related digression: There are times when you want to a numeric column variable as a metric, i.e. label, for a selected timeseries. In the example below from quite a different panel you'll see the use of the function cast as varchar as the way to do this for, in this case, the real variable frequency.

```
SELECT
  "time" AS "time", cast(frequency as varchar) as metric,
  level
FROM noise_fscan
WHERE
  site='$site' and receiver='$receiver' and frequency in ($std_frequencies) and
  $__timeFilter("time")
ORDER BY 1
```

3.2 Pull-down selections within the SELECT clause

We can also create pull-down selection lists for a choice of column names, for example to select c2_level or rms_level or both from the wsprdaemon_noise table. In this case we need the variable for the selection to be accessible in the SELECT clause within the Query Builder. This is rather more complicated and not at all intuitive.

Continuing with the example wsprdaemon_noise table as in 3.1, follow the process to create a new Variable as in that section; let us call the variable noise_type. In Query Options, set Data source as before, Refresh is On Dashboard Load, and in the Query line use the following to list the column names:

SELECT json_object_keys (to_json ((SELECT t FROM public.wsprdaemon_noise t LIMIT 1)));

Explanation: The inner SELECT t FROM public.wsprdaemon_noise t LIMIT 1 returns one line of data from the table, the to_json provides it in a form for json_object_keys to list the keys (column names) of the data.

Alone, this will list us all the column names (see the Preview of values at bottom), but we just want c2_level and rms_level, so in the Regex box enter

/level/

to only list the output from the SELECT that includes 'level'[2]. Select Sort as alphabetical (ascending). In Selection Options slide Multi-value on. Save the Variables and then go back to the Dashboard and pull down edit from its title so we can return to the Query Builder and look at the SQL.

The SELECT clause in the Query Builder needs to be changed to make use of the variable noise_type. In needs to change to the following, **note** the curly braces in ${noise_type:csv}:

```
SELECT
  "time" AS "time",
  (site, receiver, band) AS metric,
   ${noise_type:csv} as c2_level, rms_level
FROM wsprdaemon_noise
WHERE
  site in ($site_name) AND
  receiver IN ($receiver_name) AND
  band IN ($band_m)
ORDER BY 1,2
```

It took me quite some time to find the correct syntax for ${noise_type:csv}. The curly braces and :csv signify "disable quoting" and accepts a comma separated list as its input.

3.3 Queries using SQL, not the Query Builder

There are queries we may want to use that are not supported by the Query Builder options. Grafana allows direct entry of all of the SQL statements needed to form a query. In this example we have three direct entry queries to form a panel to show spot distance statistics in 10 minute intervals.

Use what you've learnt above to create a new Dashboard and a panel, this time with wsprnet as the data source so we can look at all reporters not just WsprDaemon users.

First set up Variables, as we will use this Dashboard to compare two Reporters and two bands we need four variables:

receiver_A	with query	select distinct "Reporter" from spots;
band_A		select distinct wd_band from spots where "Reporter"='$receiver_A';

and repeat for _B

In the Settings pull-down for the first panel the title can be Distance statistics for $receiver_A on band $band_A and we will create a second panel for receiver_B on band_B.

The statistics we will use are the median, lower quartile and upper quartile, where the following SQL is entered in queries A, B and C:

```
SELECT
  $__timeGroupAlias("wd_time",10m),
    percentile_disc(0.5) within group(order by spots.distance) as """median"""
FROM spots
WHERE
  $__timeFilter("wd_time") AND
  "Reporter" = '$receiver_A' AND
  wd_band = '$band_A'
GROUP BY 1
ORDER BY 1
```

[2] See https://www.boost.org/doc/libs/1_38_0/libs/regex/doc/html/boost_regex/syntax/basic_extended.html for a guide to regular expression syntax

Note: it is quite possible that you may see an error: pq: could not find pathkey item to sort, I have yet to find the root cause of this - it is not an error in the SQL, the work-around is to ask for longer data intervals, e.g. if it happens with one day selected, try 2, then 3 etc.

click + Query then insert

```
SELECT
  $__timeGroupAlias("wd_time",10m),
    percentile_disc(0.25) within group(order by spots.distance) as """lower
quartile"""
FROM spots
WHERE
  $__timeFilter("wd_time") AND
  "Reporter" = '$receiver_A' AND
  wd_band = '$band_A'
GROUP BY 1
ORDER BY 1
```

click + Query then insert

```
SELECT
  $__timeGroupAlias("wd_time",10m),
    percentile_disc(0.75) within group(order by spots.distance) as """upper
quartile"""
FROM spots
WHERE
  $__timeFilter("wd_time") AND
  "Reporter" = '$receiver_A' AND
  wd_band = '$band_A'
GROUP BY 1
ORDER BY 1
```

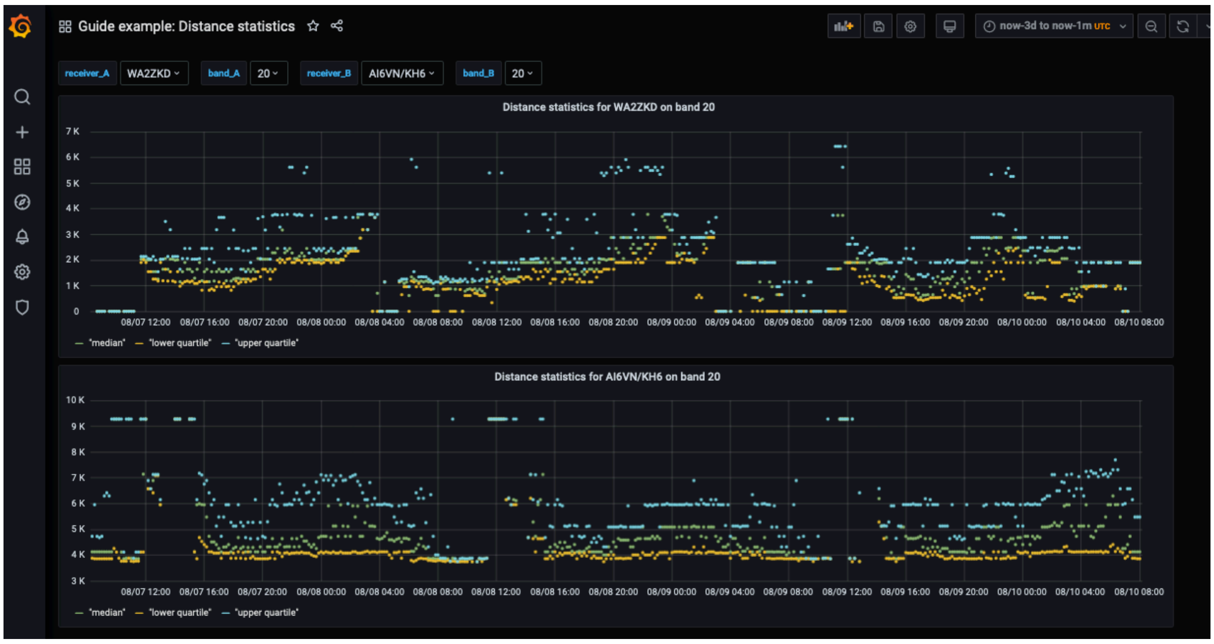
Adjust the plot using the Display pull-down to change to Points with Point Radius 1 rather than lines. Click Save. **Note:** This panel will take a fair time to open as it is getting pull down list options for all wsprnet reporters.

Click the bar graph Add Panel icon top right, and repeat the above steps but refering to _B rather than _A, remembering to set the data source.

Resize the panels to suit, and reorder so -A on top.

The Guide example: Distance Statistics - Wsprnet table dashboard is shown below, where receiver_A has been selected as WA2ZKD (New York State) and receiver_B as AI6VN/KH6 on Maui, and both bands are 20m - an example of where we want to compare the statistics for two different stations on the same band, here over 48 hours. The lower plot shows an example where we compare the statistics for the same station on two bands, here 20 and 40m.

Note that variables - used for the pull-down selections - are defined for a Dashboard, not individual panels. Therefore, the list is above Panel A, while Panel A only uses the first two variables and Panel B the second two.


Guide example: Distance statistics
receiver_A WA2ZKD
band_A 20
receiver_B AI6VN/KH6
band_B 20
now-3d to now-1m UTC
Distance statistics for WA2ZKD on band 20
Distance statistics for AI6VN/KH6 on band 20
"median" "lower quartile" "upper quartile"

4. Time series graphs of derived variables

As TimescaleDB is built upon postgreSQL, a full-featured relational database, we can derive variables from data of two (or more) reporters to plot with Grafana.

4.1 SNR comparison between two reporters

In this example we will form the SNR difference for a sender received by two reporters, if that spot is on the same band and at the same time for both reporters. The Dashboard built in this example can be extended with other panels as described in section 3, e.g. with heat maps of distance and azimuth at the receiver.

Create a new Dashboard and add a new panel. We need three variables - receiver_A, receiver_B and band_m, create them via the Dashboard settings icon as in section 3.1 using wsprnet as the data source and spots as the table, and in the meantime give the Dashboard a name, set timezone as UTC etc.

Select wsprnet as the data source. In the Edit SQL box enter the following:

```
SELECT
  s1."Date" AS "time",
  (s1."dB" - s2."dB" ) as "dB_difference"
from spots s1
inner join spots s2 on s1.wd_time = s2.wd_time
and s1."CallSign" = s2."CallSign"
and s1."Reporter" = '$receiver_A' and s2."Reporter" = '$receiver_B'
and s1.wd_band='$band_m' and s2.wd_band='$band_m'
```

In the SELECT line we form the difference of the SNR for the two selected receivers. s1 and s2 are both aliases for the table spots - declared in the FROM line and the INNER JOIN line. Two aliases for the same table result in a "self-join" query, as if we had two tables. The JOIN happens when the time, CallSign received and band are the same for the two reporters.

Via the Settings pull-down enter a panel title, e.g.

SNR comparison between $receiver_A and $receiver_B on $band_m m

Via the Display pull-down select Points and Point Radius 1, and via the Axes pull-down set appropriate Left Y axis min and max and give the axis a label. An excellent suggestion from Glenn

N6GN was to add a line at zero. This can be done using the Thresholds pull-down, select Add threshold, set T1 as gt (greater than) and enter 0 for the threshold, and slide the Fill button to the left - no fill.

The scatterplot is useful, but we can add a panel with median and quartiles that does help.

Click the back arrow at top left and the bargraph Add panel icon. Select wsprnet as the data source. Into query A Edit SQL box enter:

```
SELECT
  $__timeGroupAlias(s1.wd_time,20m), percentile_cont(0.25) within group(order by
(s1."dB" - s2."dB")) as """lower quartile"""
FROM spots s1
join spots s2 on s1.wd_time = s2.wd_time
and s1."CallSign" = s2."CallSign"
and s1."Reporter" = '$receiver_A' and s2."Reporter" = '$receiver_B'
and s1.wd_band='$band_m' and s2.wd_band='$band_m'
WHERE $__timeFilter(s1.wd_time)
GROUP BY 1
ORDER BY 1
```

Click + Query and enter into the Edit SQL box

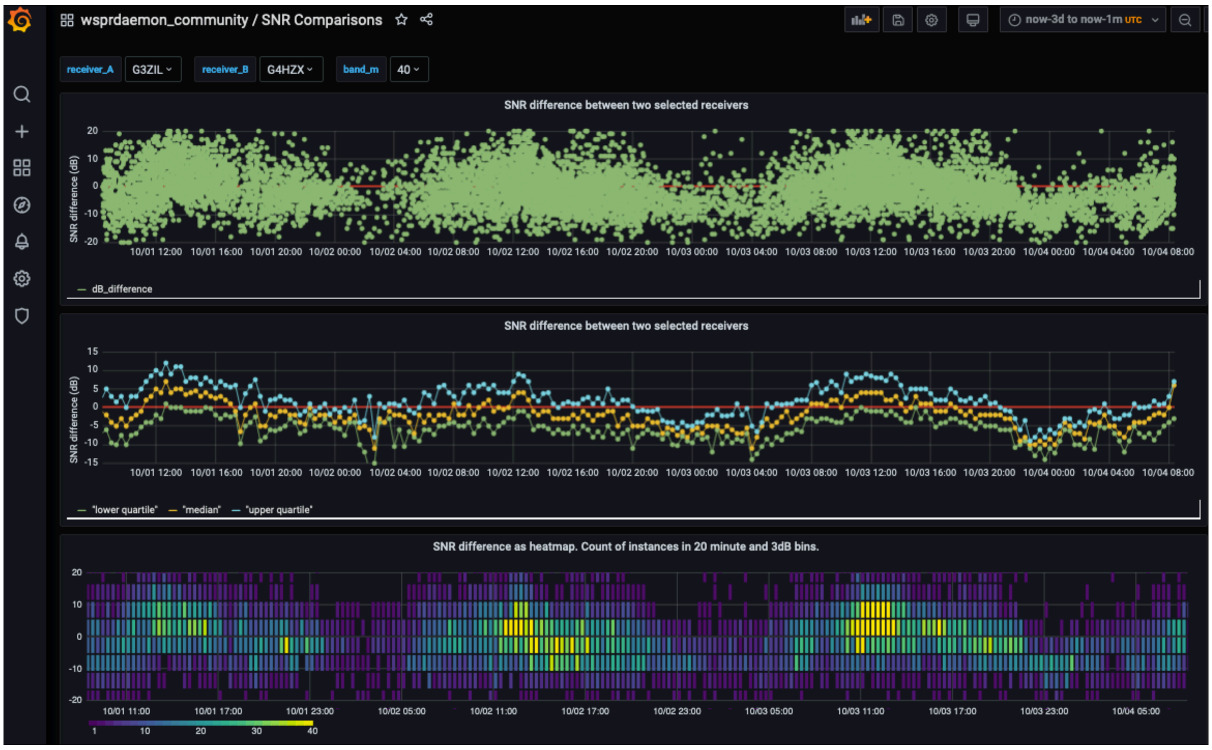
```
SELECT
  $__timeGroupAlias(s1.wd_time,20m), percentile_cont(0.5) within group(order by
(s1."dB" - s2."dB")) as """median"""
FROM spots s1
join spots s2 on s1.wd_time = s2.wd_time
and s1."CallSign" = s2."CallSign"
and s1."Reporter" = '$receiver_A' and s2."Reporter" = '$receiver_B'
and s1.wd_band='$band_m' and s2.wd_band='$band_m'
WHERE $__timeFilter(s1.wd_time)
GROUP BY 1
ORDER BY 1
```

Click + Query and enter into the Edit SQL box

```
SELECT
  $__timeGroupAlias(s1.wd_time,20m), percentile_cont(0.75) within group(order by
(s1."dB" - s2."dB")) as """upper quartile"""
FROM spots s1
join spots s2 on s1.wd_time = s2.wd_time
and s1."CallSign" = s2."CallSign"
and s1."Reporter" = '$receiver_A' and s2."Reporter" = '$receiver_B'
and s1.wd_band='$band_m' and s2.wd_band='$band_m'
WHERE $__timeFilter(s1.wd_time)
GROUP BY 1
ORDER BY 1
```

Use the Settings, Display and Axes pull-downs to customise, although you may want to have lines for this plot, but set area fill to 0, and have Points. Using the Threshold pull-down add a line at zero as above.

The screenshot below shows the resulting SNR Comparisons Dashboard, here for G3ZIL and G4HZX on 40m and with a heatmap panel; despite a great deal of variability there is a clear repeatable daily pattern in the statistics.


wsprdaemon_community / SNR Comparisons
now-3d to now-1m UTC
receiver_A G3ZIL
receiver_B G4HZX
band_m 40
SNR difference between two selected receivers
SNR difference (dB)
dB_difference
SNR difference between two selected receivers
"lower quartile" "median" "upper quartile"
SNR difference as heatmap. Count of instances in 20 minute and 3dB bins.

Part 2 Examples of graphics and interpretation with postgreSQL code.

5.Time series SNR comparisons and deriving Signal Level WsprDaemon tables

5.1 Two receivers same site SNR timeseries comparison

This Dashboard is a variant on the SNR comparison Dashboard between two sites described in section 4.1. This variant is intended for those stations with multiple receivers on one site.

The Dashboard has seven panels, whose postgreSQL is listed below, and the variables: Callsign (that is, the reporting station name), receiver_A and receiver_B (that is, two of the individual receivers in the WsprDaemon merged data stream), band, and a minimum and maximum distance that allows you to look at SNR differences with range in the statistics panel. From the top, all accessing wsprdaemon_tutorial as the data source:

Noise level:

Query A

```
SELECT
  "time" AS "time",
  receiver AS metric,
  c2_level
FROM wsprdaemon_noise
WHERE
  site = '$Callsign' AND receiver='$receiver_A'
  and band = '$band_m'
ORDER BY 1,2
```

Query B

```
SELECT
  "time" AS "time",
  receiver AS metric,
  c2_level
FROM wsprdaemon_noise
WHERE
  site = '$Callsign' AND receiver='$receiver_B' and
  band = '$band_m'
ORDER BY 1,2
```

Noise level difference

```
SELECT
  s1.time AS time,
  (s1."c2_level" - s2."c2_level" ) as "dB_difference"
from wsprdaemon_noise s1
join wsprdaemon_noise s2 on s1.time = s2.time
and s1."site"= '$Callsign' and s2."site" = '$Callsign'
and s1."receiver"='$receiver_A' and s2."receiver"='$receiver_B'
and s1.band='$band_m' and s2.band='$band_m'
```

SNR difference:

```
SELECT
  s1.time AS time,
  (s1."SNR" - s2."SNR" ) as "dB_difference"
from wsprdaemon_spots s1
join wsprdaemon_spots s2 on s1.time = s2.time
and s1."tx_call"= s2."tx_call"
and s1."rx_id"= '$Callsign' and s2."rx_id" = '$Callsign'
and s1."receiver"='$receiver_A' and s2."receiver"='$receiver_B'
and s1.band='$band_m' and s2.band='$band_m'
```

SNR statistics:

Lower quartile:

```
SELECT
  $__timeGroupAlias(s1.time,20m),
    percentile_disc(0.25) within group(order by (s1."SNR" - s2."SNR")) as """lower
```

```
quartile"""
FROM wsprdaemon_spots s1
join wsprdaemon_spots s2 on
s1.time = s2.time
and s1."tx_call" = s2."tx_call"
and s1."rx_id" = '$Callsign' and s2."rx_id" = '$Callsign'
and s1."receiver"='$receiver_A' and s2."receiver"='$receiver_B'
and s1.band='$band_m' and s2.band='$band_m'
WHERE $__timeFilter(s1.time) and s1.km>'$stats_min_distance' and
s1.km<'$stats_max_distance'
GROUP BY 1
ORDER BY 1
```

Median

```
SELECT
  $__timeGroupAlias(s1.time,20m),
    percentile_disc(0.5) within group(order by (s1."SNR" - s2."SNR")) as
"""median"""
FROM wsprdaemon_spots s1
join wsprdaemon_spots s2 on
s1.time = s2.time
and s1."tx_call" = s2."tx_call"
and s1."rx_id" = '$Callsign' and s2."rx_id"= '$Callsign'
and s1."receiver"='$receiver_A' and s2."receiver"='$receiver_B'
and s1.band='$band_m' and s2.band='$band_m'
WHERE $__timeFilter(s1.time) and s1.km>'$stats_min_distance' and
s1.km<'$stats_max_distance'
GROUP BY 1
ORDER BY 1
```

Upper quartile

```
SELECT
  $__timeGroupAlias(s1.time,20m),
    percentile_disc(0.75) within group(order by (s1."SNR" - s2."SNR")) as """upper
quartile"""
FROM wsprdaemon_spots s1
join wsprdaemon_spots s2 on
s1.time = s2.time
and s1."tx_call"= s2."tx_call"
and s1."rx_id" = '$Callsign' and s2."rx_id"= '$Callsign'
and s1."receiver"='$receiver_A' and s2."receiver"='$receiver_B'
and s1.band='$band_m' and s2.band='$band_m'
WHERE $__timeFilter(s1.time) and s1.km>'$stats_min_distance' and
s1.km<'$stats_max_distance'
GROUP BY 1
ORDER BY 1
```

Distance of transmitters from the receiving site:

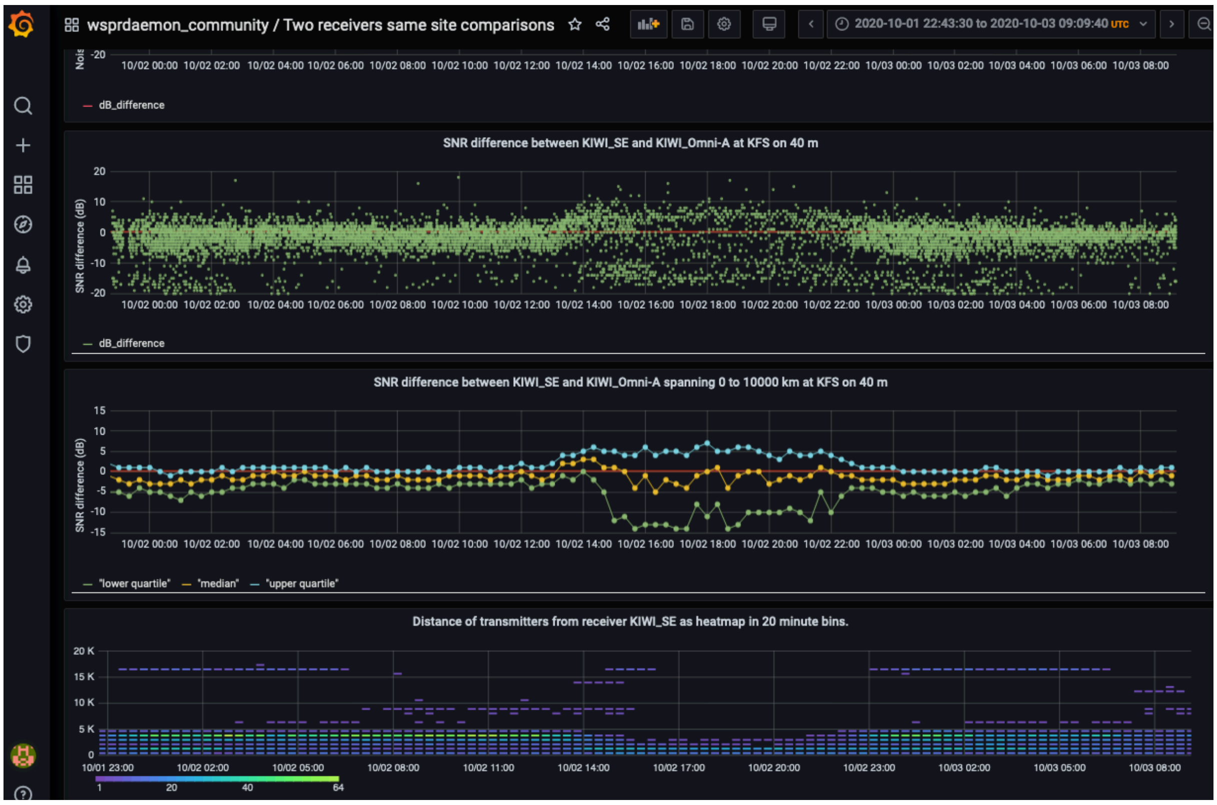
```
SELECT
  time AS time,
  km
FROM wsprdaemon_spots
WHERE
  $__timeFilter(time) AND
  band = '$band_m' AND
  "rx_id" = '$Callsign' and "receiver"='$receiver_A'
ORDER BY 1
```

SNR difference as a heatmap:

```
SELECT
  s1.time AS time,
  (s1."SNR" - s2."SNR" ) as "dB_difference"
from wsprdaemon_spots s1
join wsprdaemon_spots s2 on s1.time = s2.time
and s1."tx_call" = s2."tx_call"
and s1."rx_id"= '$Callsign' and s2."rx_id"= '$Callsign'
and s1."receiver"='$receiver_A' and s2."receiver"='$receiver_B'
and s1.band='$band_m' and s2.band='$band_m'
```

Azimuth of the transmitters as seen at the receiver as a heatmap:

```
SELECT
  time AS time,
  rx_az
FROM wsprdaemon_spots
WHERE
  $__timeFilter(time) AND
  band = '$band_m' AND
  "rx_id"= '$Callsign' and "receiver"='$receiver_A'
ORDER BY 1
```


wsprdaemon_community / Two receivers same site comparisons
2020-10-01 22:43:30 to 2020-10-03 09:09:40 UTC
dB_difference
SNR difference between KIWI_SE and KIWI_Omni-A at KFS on 40 m
SNR difference (dB)
dB_difference
SNR difference between KIWI_SE and KIWI_Omni-A spanning 0 to 10000 km at KFS on 40 m
"lower quartile"
"median"
"upper quartile"
Distance of transmitters from receiver KIWI_SE as heatmap in 20 minute bins.

The example above shows three of the seven panels from this Dashboard. The receiver site is KFS[3] and in this comparison receiver_A is connected to the 'SE Sector' antenna, a TCI 527B Super High Gain Log-Periodic while receiver_B is "Omni_A" a TCI530 Short and Medium Range Log-Periodic. The data is for 40 m. The offset between the two reported noise levels (not shown) is likely down to differences in the antennas and or the RF distribution chain. The SNR differences from the scatter plot show a bimodal distribution, with a main peak just below zero, suggesting the advantage is with the Omni_B, and a secondary peak at about -15 dB, although with far fewer spots - this is worthy of investigation. Another interesting feature is the broadening of the distribution of differences between 1400 and 2200 UTC - daytime - where the lower panel shows fewer spots being received.

5.2 SNR and number of spots comparison between two receivers at the same site

This Dashboard is a variant on that in 4.2 that has panels to show a) the SNR of spots heard by receiver_A but not receiver_B, b) vice versa, those heard by receiver_B but not receiver_A, c) counts

[3] See http://69.27.184.62:8901/about.html for details.

in 20 minute intervals of spots heard by receiver_A and receiver_B and d) counts of spots heard by receiver_A but not receiver_B and receiver_B but not receiver_A.

The postgreSQL for these four panels is listed below:

*SNR of spots heard by receiver_A but not receiver_*B, using left join and null for the tx_call for receiver_B

```
SELECT
  s1."time" AS "time",
  s1."SNR"
FROM wsprdaemon_spots s1
left join wsprdaemon_spots s2
on s1.time = s2.time
  and s1.rx_id = '$Callsign' and s2.rx_id = '$Callsign'
  and s1.receiver='$receiver_A' and s2.receiver='$receiver_B'
  and s1.band='$band_m' and s2.band='$band_m'
  and s1.tx_call = s2.tx_call
where s1.rx_id = '$Callsign' and s1.receiver = '$receiver_A' and s1.band='$band_m'
and s2.tx_call is null
and s1.time between $__timeFrom() and $__timeTo()
```

SNR of spots heard by receiver_B but not receiver_A, using right join and null for the tx_call for receiver_A

```
SELECT
  s2."time" as "time",
  s2."SNR"
FROM wsprdaemon_spots s1
right join wsprdaemon_spots s2
on s1.time = s2.time
  and s1.rx_id = '$Callsign' and s2.rx_id = '$Callsign'
  and s1.receiver='$receiver_A' and s2.receiver='$receiver_B'
  and s1.band='$band_m' and s2.band='$band_m'
  and s1.tx_call = s2.tx_call
where s2.rx_id = '$Callsign' and s2.receiver = '$receiver_B' and s2.band='$band_m'
and s1.tx_call is null
and s2.time between $__timeFrom() and $__timeTo()
```

Counts in 20 minute intervals of spots heard by receiver_A and receiver_B, use of left outer join and time_buckets

for receiver_A

```
SELECT
  time_bucket('20 minutes', s1.time) AS "time",
  count(s1."SNR") as "heard by $receiver_A"
FROM wsprdaemon_spots s1
left outer join wsprdaemon_spots s2
on s1.time = s2.time
  and s1.rx_id = '$Callsign' and s2.rx_id = '$Callsign'
  and s1.receiver='$receiver_A' and s2.receiver='$receiver_B'
  and s1.band='$band_m' and s2.band='$band_m'
  and s1.tx_call = s2.tx_call
where s1.rx_id = '$Callsign' and s1.receiver = '$receiver_A' and s1.band='$band_m'
and s1.time between $__timeFrom() and $__timeTo()
group by time_bucket('20 minutes', s1.time)
order by time_bucket('20 minutes', s1.time)
```

for receiver_B

```
SELECT
  time_bucket('20 minutes', s2.time) AS "time",
  count(s2."SNR") as "heard by $receiver_B"
FROM wsprdaemon_spots s1
right outer join wsprdaemon_spots s2
on s1.time = s2.time
  and s1.rx_id = '$Callsign' and s2.rx_id = '$Callsign'
  and s1.receiver='$receiver_A' and s2.receiver='$receiver_B'
  and s1.band='$band_m' and s2.band='$band_m'
  and s1.tx_call = s2.tx_call
```

```
where s2.rx_id = '$Callsign' and s2.receiver = '$receiver_B' and s2.band='$band_m'
and s2.time between $__timeFrom() and $__timeTo()
group by time_bucket('20 minutes', s2.time)
order by time_bucket('20 minutes', s2.time)
```

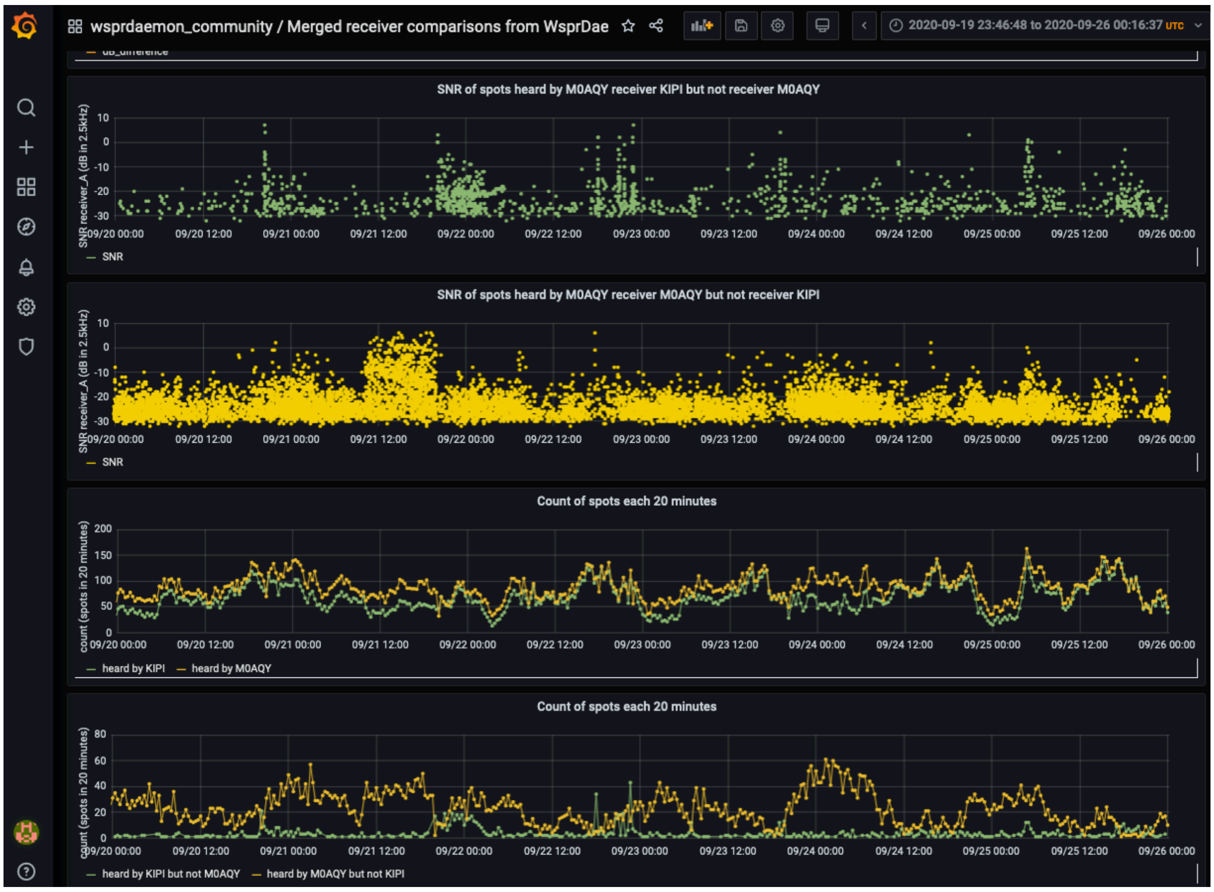
Counts of spots heard by receiver_A but not receiver_B and receiver_B but not receiver_A,

A but not B, using left join, and null for tx_call for receiver_B:

```
SELECT
  time_bucket('20 minutes', s1.time) AS "time",
  count(s1."SNR") as "heard by $receiver_A but not $receiver_B"
FROM wsprdaemon_spots s1
left join wsprdaemon_spots s2
on s1.time = s2.time
  and s1.rx_id = '$Callsign' and s2.rx_id = '$Callsign'
  and s1.receiver='$receiver_A' and s2.receiver='$receiver_B'
  and s1.band='$band_m' and s2.band='$band_m'
  and s1.tx_call = s2.tx_call
where s1.rx_id = '$Callsign' and s1.receiver = '$receiver_A' and s1.band='$band_m'
and s2.tx_call is null
and s1.time between $__timeFrom() and $__timeTo()
group by time_bucket('20 minutes', s1.time)
order by time_bucket('20 minutes', s1.time)
```

B but not A, right join and null for tx_call for receiver_A:

```
SELECT
  time_bucket('20 minutes', s2.time) AS "time",
  count(s2."SNR") as "heard by $receiver_B but not $receiver_A"
FROM wsprdaemon_spots s1
right join wsprdaemon_spots s2
on s1.time = s2.time
  and s1.rx_id = '$Callsign' and s2.rx_id = '$Callsign'
  and s1.receiver='$receiver_A' and s2.receiver='$receiver_B'
  and s1.band='$band_m' and s2.band='$band_m'
  and s1.tx_call = s2.tx_call
where s2.rx_id = '$Callsign' and s2.receiver = '$receiver_B' and s2.band='$band_m'
and s1.tx_call is null
and s2.time between $__timeFrom() and $__timeTo()
group by time_bucket('20 minutes', s2.time)
order by time_bucket('20 minutes', s2.time)
```

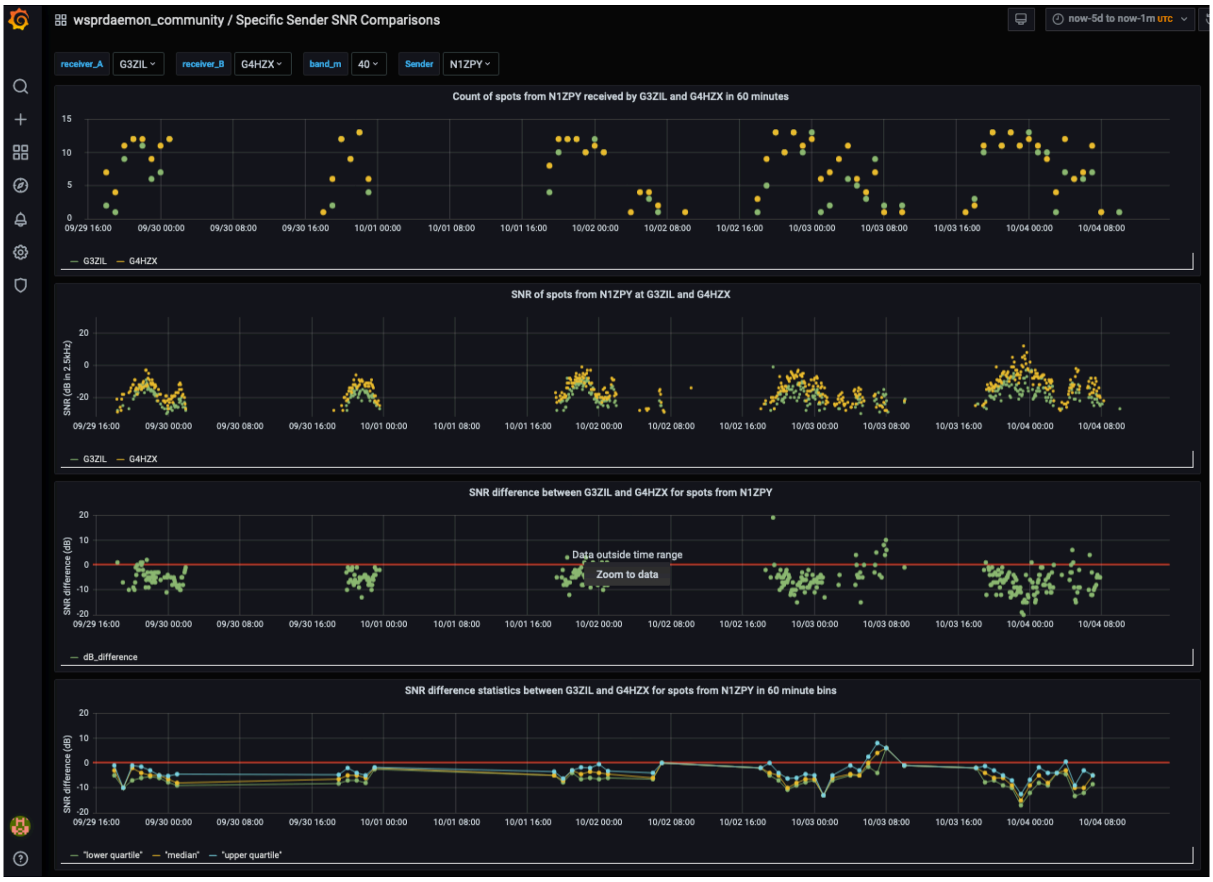

wsprdaemon_community / Merged receiver comparisons from WsprDae
2020-09-19 23:46:48 to 2020-09-26 00:16:37 UTC
SNR of spots heard by M0AQY receiver KIPI but not receiver M0AQY
SNR receiver_A (dB in 2.5kHz)
SNR
SNR of spots heard by M0AQY receiver M0AQY but not receiver KIPI
Count of spots each 20 minutes
count (spots in 20 minutes)
heard by KIPI
heard by M0AQY
heard by KIPI but not M0AQY
heard by M0AQY but not KIPI

Shown above are these four panels out of ten in this Dashboard where the reporting Callsign is M0AQY and the two receivers, KIPI and M0AQY both use LZ1AQ loop antennas but with KIPI oriented N–S and M0AQY oriented E–W. IN this case, the E–W oriented loop, M0AQY, decodes more spots than the N–S oriented loop, but there are periods of some hours, e.g. 1800 UTC 21 September to 0300 UTC 22 September when the N–S loop decodes spots not decoded on the E–W loop. There are some curiosities, for example the period from about 1000 UTC to 1800 UTC on 21 September when the E–W loop M0AQY was decoding spots with SNRs above -10 dB that were not being decoded by the N–S loop, and this period started and stopped quite suddenly. There was no obvious change in the SNR difference of those spots that were decoded by both receivers during this period.

5.3 Specific sender spot count and SNR comparisons

This is a Dashboard that accesses wsprnet data, and so is applicable to all reporters. The idea is that looking at the SNR of a single sender at two reporters is of interest. The Dashboard shows the counts in one hour intervals, the SNR at the two reporters and the SNR difference if the reported spots are coincident in time. The quartile and median differences are also shown. In this transatlantic example the sender is N1ZPY, Maine, and the two UK receivers are G4HZX (vertical antenna, quiet location) and G3ZIL (low dipole, suburban location). These panels pick out nicely the variation in spot count and SNR during each day's opening and the day-to-day variation.

The 'Vee' shape distribution to the SNR difference on most days (here the last three) is intriguing and is worthy of further study[4].



The postgreSQL for the panels is:

Count of spots, receiver_A shown, just change to receiver_B for the other

```
SELECT
  $__timeGroupAlias(wd_time,1h),
  "Reporter" AS metric,
  count("dB") AS "Spot Count"
FROM spots
WHERE
  "Reporter" = '$receiver_A' AND
  "CallSign" = '$Sender' AND
  wd_band = '$band_m'
GROUP BY 1,2
ORDER BY 1,2
```

SNR, receiver_A shown

```
SELECT
  wd_time AS "time",
  "Reporter" AS metric,
  "dB"
FROM spots
WHERE
  "Reporter" = '$receiver_A' AND
```

[4] G3ZIL does have a draft paper investigating the possible role of bias due to spots only above the decoding threshold being reported, and thus randomly stronger signals at the lower SNR reporter being over-represented in any difference calculation.

```
  wd_band = '$band_m' AND
  "CallSign" = '$Sender'
ORDER BY 1,2
```

SNR Difference

```
SELECT
  s1."Date" AS "time",
  (s1."dB" - s2."dB" ) as "dB_difference"
from spots s1
inner join spots s2 on s1.wd_time = s2.wd_time
and s1."CallSign" = s2."CallSign"
and s1."Reporter" = '$receiver_A' and s2."Reporter" = '$receiver_B'
and s1.wd_band='$band_m' and s2.wd_band='$band_m' and s1."CallSign"='$Sender'
```

Statistics postgreSQL is as the code for the panel in section 5.1

5.4 Specific sender SNR and Signal Level from WsprDaemon tables

One of the motivating factors for estimating noise at the same time, and within the same frequency band, as decoding WSPR spots was to enable signal levels to be estimated. Care needs to be taken in interpreting the results of the simple calculations, as the noise level estimate within the decoding program wsprd is not as robust as that within WsprDaemon. For example, the estimate within wsprd is more likely to be affected by the presence of multiple strong signals within the WSPR band. Please be aware that you may find some odd results at times.

It only really makes sense to code a Dashboard and panels for a single sender and a single receiver. As a noise level estimate is needed we are limited to reporters using WsprDaemon, hence the Dashboard uses wsprdaemon_spots and wsprdaemon_noise data. I'll skip the Dashboard creation basics, just noting that the variables we need are receiver, noise receiver, the specific sender, and band.

The postgreSQL for the three panels are, SNR:

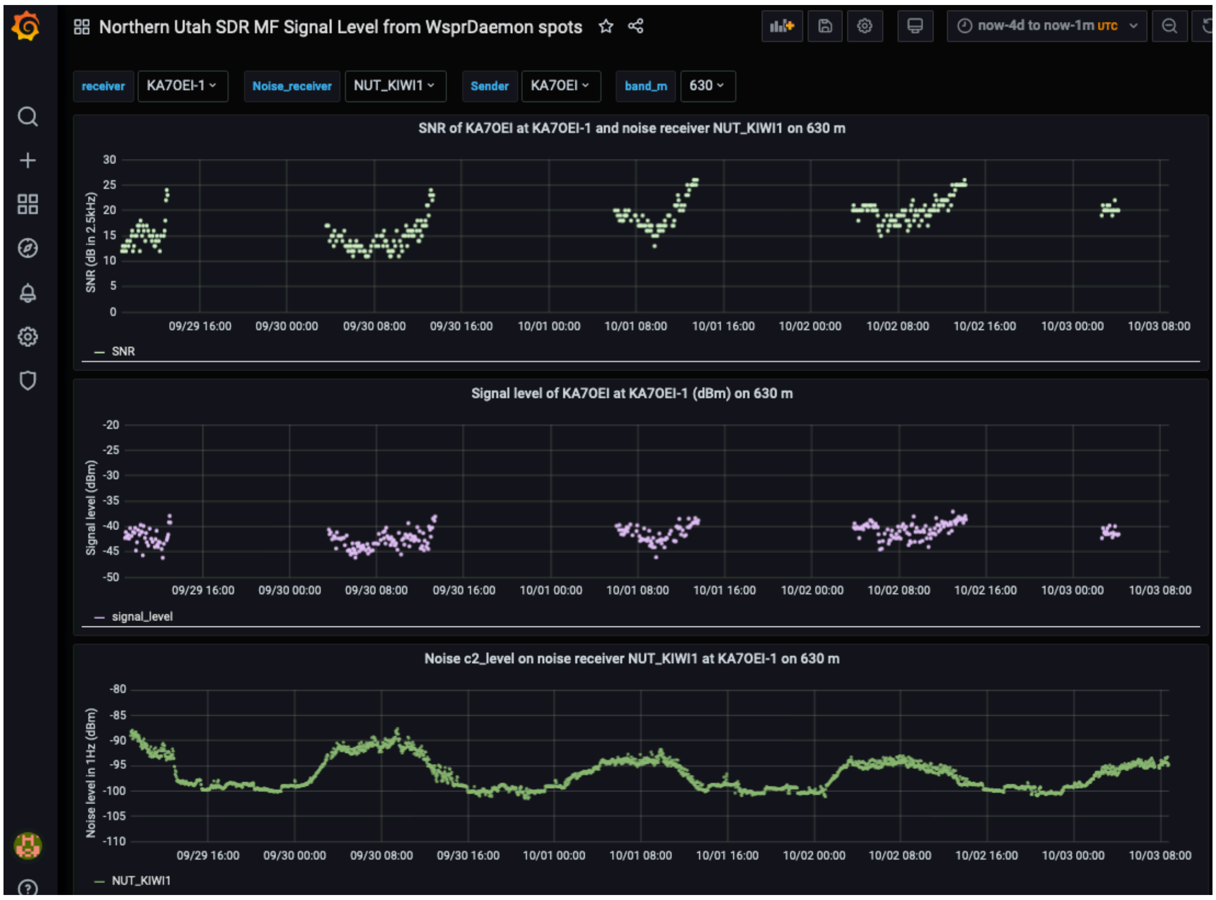
```
SELECT
  "time" AS "time",
  "SNR"
FROM wsprdaemon_spots
WHERE
  rx_id = '$receiver' AND
  band = '$band_m' AND
  tx_call = '$Sender'
ORDER BY 1
```

Signal level, where we perform an inner join between the wsprdaemon_spots table as alias s1 and wsprdaemon_noise as alias s2. In the SELECT clause we simply add the noise level and SNR and add a constant 34 dB to shift the reference bandwidth from 2500 Hz to 1 Hz:

```
SELECT
s1.time as time, (s2.c2_level+s1."SNR"+34) as signal_level
FROM wsprdaemon_spots s1
inner join wsprdaemon_noise s2 on
s1.time = s2.time
and s1."tx_call" = '$Sender'
and s1."rx_id" = '$receiver'
and s2."site"='$receiver'
and s2."receiver"='$Noise_receiver'
and s1."band"='$band_m' and s2.band='$band_m'
WHERE $__timeFilter(s1.time)
GROUP BY 1,2
ORDER BY 1
```

Noise level, where we use the receiver as metric for it to be used as the label:

```
SELECT
  "time" AS "time",
  receiver AS metric,
  c2_level
FROM wsprdaemon_noise
WHERE
  site = '$receiver' AND
  band = '$band_m' AND
  receiver = '$Noise_receiver'
ORDER BY 1,2
```



In this example, on 630 m, the receiver is the Northern Utah SDR site as KA7OEI-1 and the Sender is KA7OEI at a distance of 115 km. It is reasonable to assume that propagation is via ground (surface) wave, and that, as a consequence, the signal level should be essentially constant with time.

The upper panel shows the SNR for four days. One characteristic daily feature is the ramped 8–10 dB increase in SNR toward the end of each transmission period. There is also day-to-day variation in SNR at the same time, e.g. 12–14 dB at 0800 UTC on 30 September while 17–19 dB on 2 October.

The lower panel shows the noise level. One characteristic daily feature is the ramp down of noise after about 1100 UTC. There is also day-to-day variation, e.g. -90 – -92 dBm at 0800 UTC on 30 September while -93 –94 dB on 2 October.

The opposite trends in SNR and noise levels from the upper and lower panels cancel, to a significant extent, to result in a more constant signal level, with time of day and from day-to-day, in the middle panel. The remaining variation is of the order of +/- 3 dB.

6. Non-time series graphs

While it may seem a little odd, given we have specifically chosen a time series database, there are instances where we would like a variable other than time on the x axis, e.g. distance. As installed, Grafana has no such capability, but there is a range of third-party plug-ins that do. I have installed the plotly plug-in[5] on logs2.wsprdaemon.org.

6.1 SNR with distance

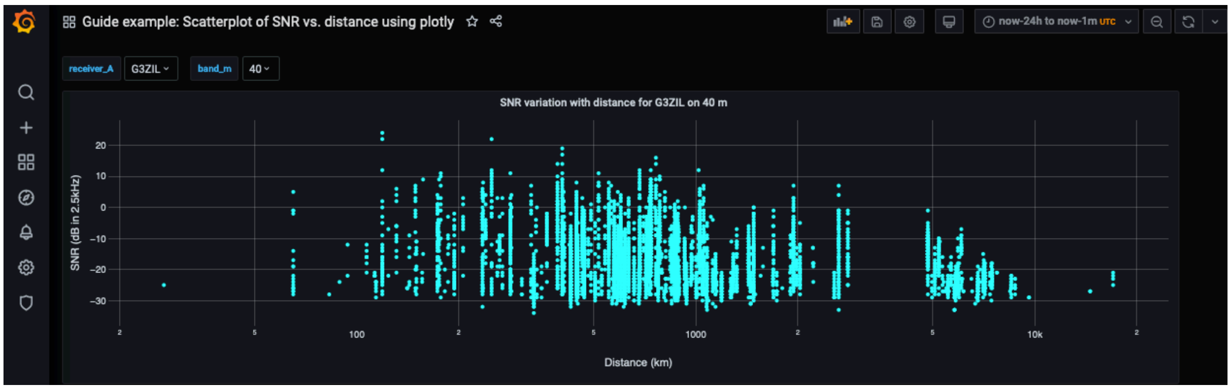
While this may not be the most useful plot, or straightforward to interpret, it serves as a simple example of a non-time series plot and the use of plotly. Create a new Dashboard and + Add panel as before. In the Visualization pull-down scroll down to Plotly, and select. You will see that the list of pull down options on the right is different to what we have seen before.

Select Dashboard settings at top right, as before give the Dashboard a name etc. Create two Variables for the receiver and band, with wsprnet as the data source, as before.

Edit the Panel, select wsprnet as the data source and enter the following into the Edit SQL box:

```
SELECT
  "wd_time" AS "time",distance,
  "dB"
FROM spots
WHERE
  $__timeFilter("wd_time") AND
  "Reporter" = '$receiver_A' AND
  wd_band = '$band_m'
ORDER BY 1
```

Select the Display pull-down on the right, enter an X axis title, select type as Linear, but you might also want to have a look at Log, enter a Y axis title. In the Traces pull-down, enter a Name for the trace, under the Metrics heading select distance for the X axis and dB for the Y axis. Under the Markers heading a size of 5 seems sensible and colour as Solid. In the Solid box you can type a colour by name, e.g. cyan. Under Text you can select Metric as distance and Show as Hov, which will show the distance and SNR when you hover over a spot on the scatterplot. The resulting panel is shown below.



6.2 3D SNR Difference with distance and with azimuth at the receiver with pull-downs

The plotly plug-in for Grafana has few options, but one that may possibly be useful is the 3D scatterplot, although it may also be a bit of a gimmick. This Dashboard starts out as for 5.1 and 5.2 but the SQL code below adds a third variable in the select, so we have azimuth at the receiver, distance and the calculated SNR difference.

In the Display pull-down, under Options select Type as Scatter (3d), and Drag as pan, the latter lets us use a mouse to rotate the final 3D data plot. Under the X axis heading set a title that will match the x

[5] See https://grafana.com/grafana/plugins/natel-plotly-panel for details

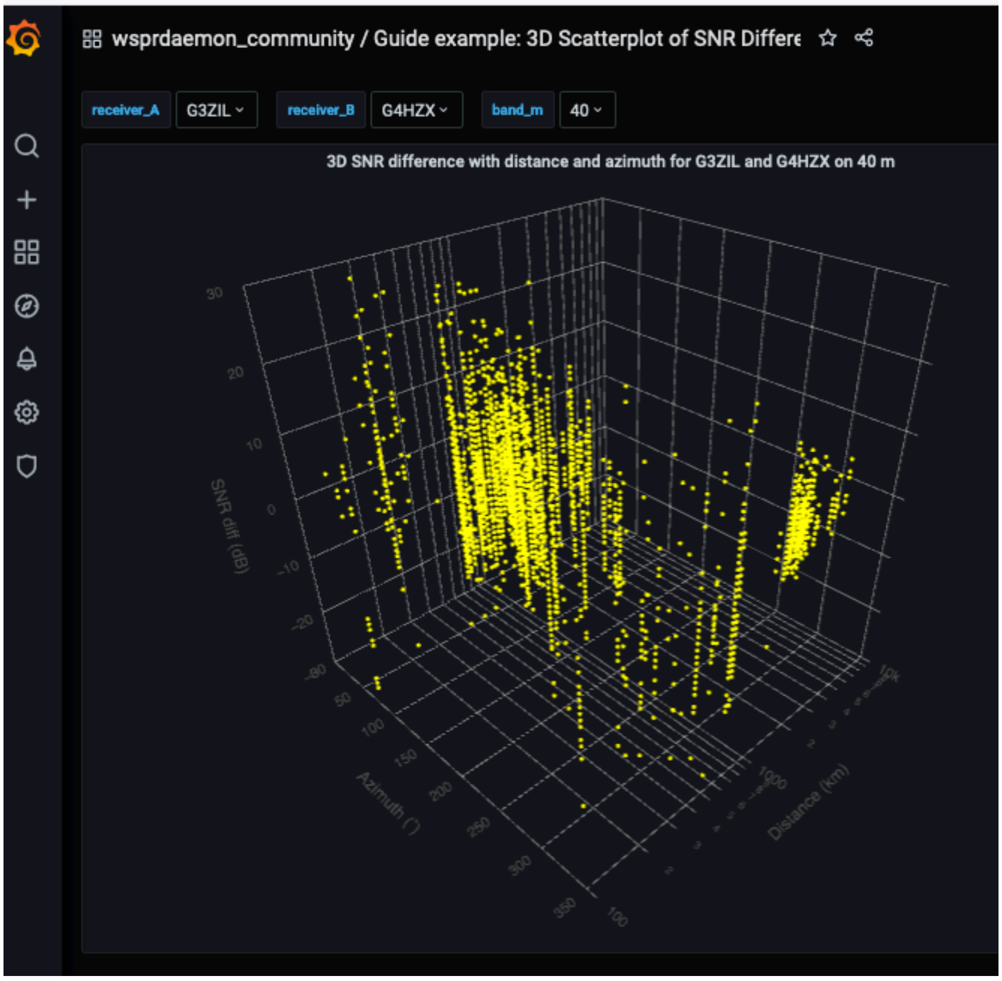
axis we will choose in the Traces pull-down, e.g. Distance (km), Type is perhaps best as Log, Range as Between and Min and Max as powers of 10, so 4 is 10,000 and 2 is 100. Although the labels are Min and Max, a more suitable axis is given if you reverse them, in this case I set 4 for Min and 2 for Max that makes the axis values increase from left to right. Under Y axis, which will be Azimuth, give an appropriate title, Type as Linear, Range as Between and 0 and 360 as Min and Max. Under Z axis, which will be SNR difference, use Type as Linear, Range as Between and set appropriate Min and Max.

In the Traces pull-down, give the Trace a name. Under Metrics, set X Axis to km, Y axis to azi and Z axis to db_difference. Under Markers check Show is to the right, Symbol is Circle, Size is 2, Colour is Solid, and Solid is a colour name of your choice. Under Lines, Show should be off, under Text, select Metric as km and Show to Hov, as we hover over points we can read the distance, azimuth and SNR difference.

This is the SQL code to insert into the Edit SQL box:

```
SELECT
  s1."wd_time" AS "time",
  s1.distance as km,
  s1.wd_rx_az as azi,
  (s1."dB" - s2."dB") as db_difference
FROM spots s1
join spots s2 on s1.wd_time = s2.wd_time
and s1."CallSign" = s2."CallSign"
and s1."Reporter" = '$receiver_A' and s2."Reporter" = '$receiver_B'
and s1.wd_band='$band_m' and s2.wd_band='$band_m'
WHERE
  $__timeFilter(s1."wd_time",10m)
ORDER BY 1
```

Here is an example of the resulting Dashboard panel:



6.3 2D scatterplots of SNR Difference with distance and with azimuth

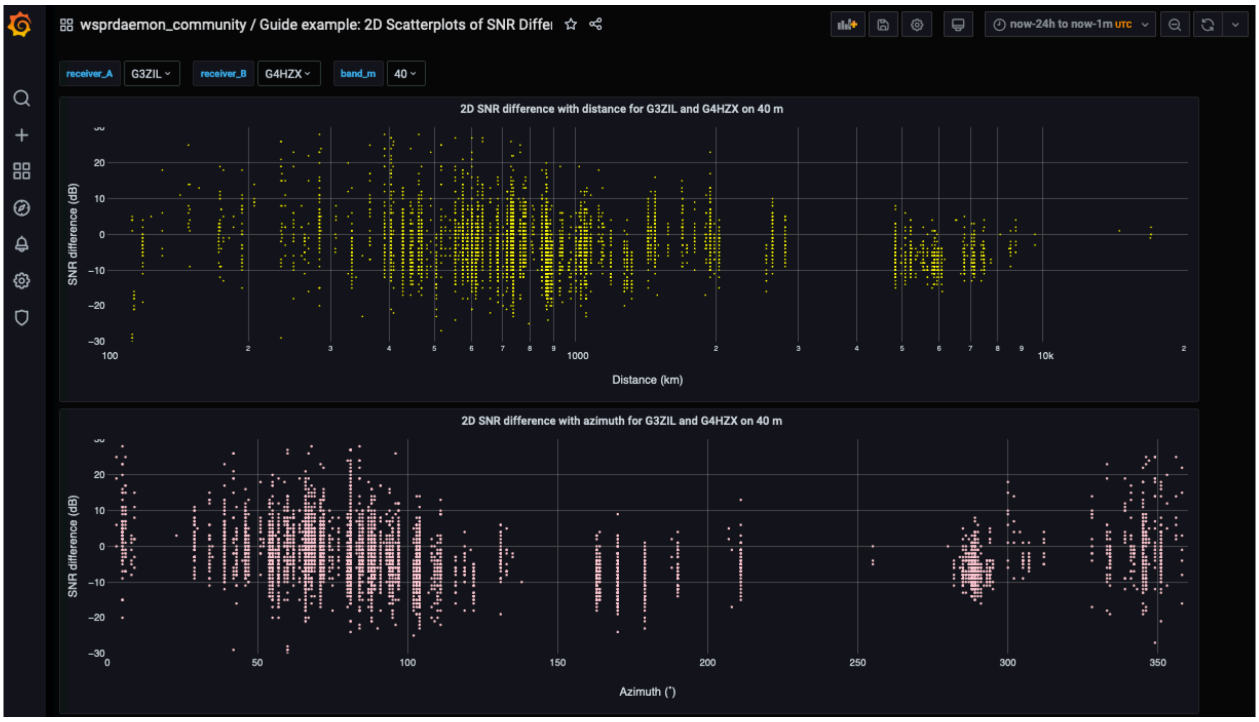
Dear reader, you may well have had enough of hand-holding explanation by now. This Dashboard is a variant on the 3D representation, create the Dashboard and panel as before, use the options on the right for a 2D scatterplot, use the top SQL for the distance panel, duplicate it, and use the SQL beneath for the azimuth panel.

```
SELECT
  s1."wd_time" AS "time",
  s1.distance as km,
  (s1."dB" - s2."dB") as db_difference
FROM spots s1
join spots s2 on s1.wd_time = s2.wd_time
and s1."CallSign" = s2."CallSign"
and s1."Reporter" = '$receiver_A' and s2."Reporter" = '$receiver_B'
and s1.wd_band='$band_m' and s2.wd_band='$band_m'
WHERE
  $__timeFilter(s1."wd_time",10m)
ORDER BY 1
```

and for azimuth

```
SELECT
  s1."wd_time" AS "time",
  s1.wd_rx_az as azi,
  (s1."dB" - s2."dB") as db_difference
FROM spots s1
join spots s2 on s1.wd_time = s2.wd_time
and s1."CallSign" = s2."CallSign"
and s1."Reporter" = '$receiver_A' and s2."Reporter" = '$receiver_B'
and s1.wd_band='$band_m' and s2.wd_band='$band_m'
WHERE
  $__timeFilter(s1."wd_time",10m)
ORDER BY 1
```

These two panels result in the Dashboard below.


wsprdaemon_community / Guide example: 2D Scatterplots of SNR Diffe
now-24h to now-1m UTC
receiver_A G3ZIL
receiver_B G4HZX
band_m 40
2D SNR difference with distance for G3ZIL and G4HZX on 40 m
SNR difference (dB)
Distance (km)
2D SNR difference with azimuth for G3ZIL and G4HZX on 40 m
SNR difference (dB)
Azimuth (°)

6.4 2D scatterplots two receivers same site SNR differences with distance and azimuth

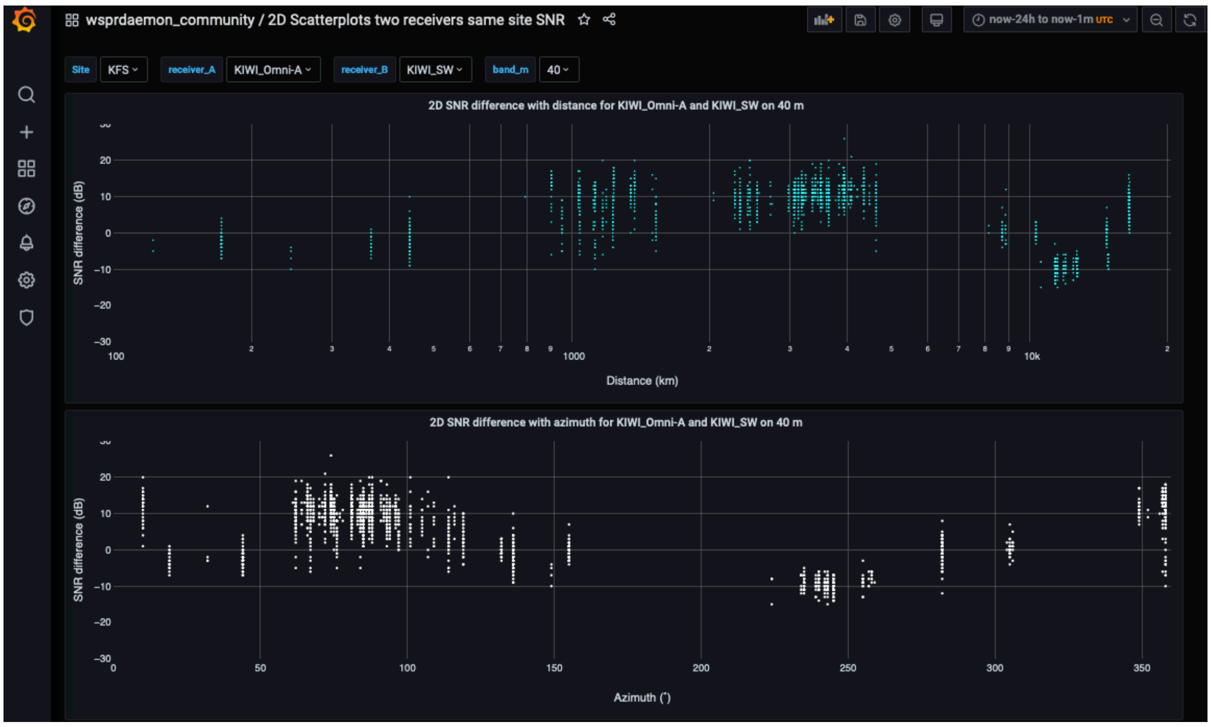
This variant is for WsprDaemon users using merged receivers reporting under one callsign. The pull-downs allow selection of the reporting Site, two receivers at that site and a band. The example below is for KFS[6] where the two receiver channels are connected to different antennas, an Omni-directional TCI530 short and medium range log periodic and a 'SW Sector' antenna, a TCI532 medium range log periodic. The postgreSQL code for the two plotly plug-in panels is:

```
SELECT
  s1."time" AS "time",
  s1.km as km,
  (s1."SNR" - s2."SNR") as db_difference
FROM wsprdaemon_spots s1
join wsprdaemon_spots s2 on s1.time = s2.time
and s1.rx_id='$Site' and s2.rx_id = '$Site'
and s1.receiver= '$receiver_A' and s2.receiver = '$receiver_B'
and s1.band='$band_m' and s2.band='$band_m'
and s1.tx_call = s2.tx_call
WHERE
  $__timeFilter(s1."time",20m)
ORDER BY 1
```

and the same for the azimuth plot except that the select line to plot is:

```
s1.rx_az as azi,
```

[6] See http://69.27.184.62:8901/about.html for details of the antennas


wsprdaemon_community / 2D Scatterplots two receivers same site SNR
now-24h to now-1m UTC
Site KFS
receiver_A KIWI_Omni-A
receiver_B KIWI_SW
band_m 40
2D SNR difference with distance for KIWI_Omni-A and KIWI_SW on 40 m
SNR difference (dB)
Distance (km)
2D SNR difference with azimuth for KIWI_Omni-A and KIWI_SW on 40 m
Azimuth (°)

6.5 Scatterplots with distance, azimuth and time of the SNR at one station for spots not heard on the same time and same band as another station

This is a nice example of what can be done with a relational database. As in section 5.3 I will skip the initial set-up of a Dashboard with a plotly panel, concentrating on the SQL to enter into the Edit SQL box. The data source for all is wsprnet, so we can access all reporters. The bulk of the SQL is the same for the three panels (two using plotly and one using a time series Graph), that for the first panel, distance, being:

```
SELECT
  s1."wd_time" AS "time",
  s1.distance as km,
  s1."dB"
FROM spots s1
left join spots s2
on s1.wd_time = s2.wd_time
  and s1."Reporter" = '$receiver_A' and s2."Reporter" = '$receiver_B'
  and s1.wd_band='$band_m' and s2.wd_band='$band_m'
  and s1."CallSign" = s2."CallSign"
where s1."Reporter" = '$receiver_A' and s1.wd_band='$band_m' and s2."CallSign" is
null
and s1.wd_time between $__timeFrom() and $__timeTo()
```

The main aspects are:

- The SELECT clause accesses wd_time, distance (for the X axis) and SNR here as "dB" (for the Y axis).
- In the FROM clause we alias spots as s1 and use a left join with alias s2 of the same table spots with our list of criteria: wd_time, reporters as those selected in the pull-downs, ensuring band is the same for both, and match the sender CallSigns. A left join can be thought of as the area to the left of an intersection on a Venn diagram.
- It is the WHERE clause that finds the spots we want - where we have a receiver_A and the selected band but no entry for the CallSign for receiver_B for wd_time between

$__timeFrom() and $__timeTo(), which are Grafana global variables for the from and to times in the time picker at top right of the Dashboard.

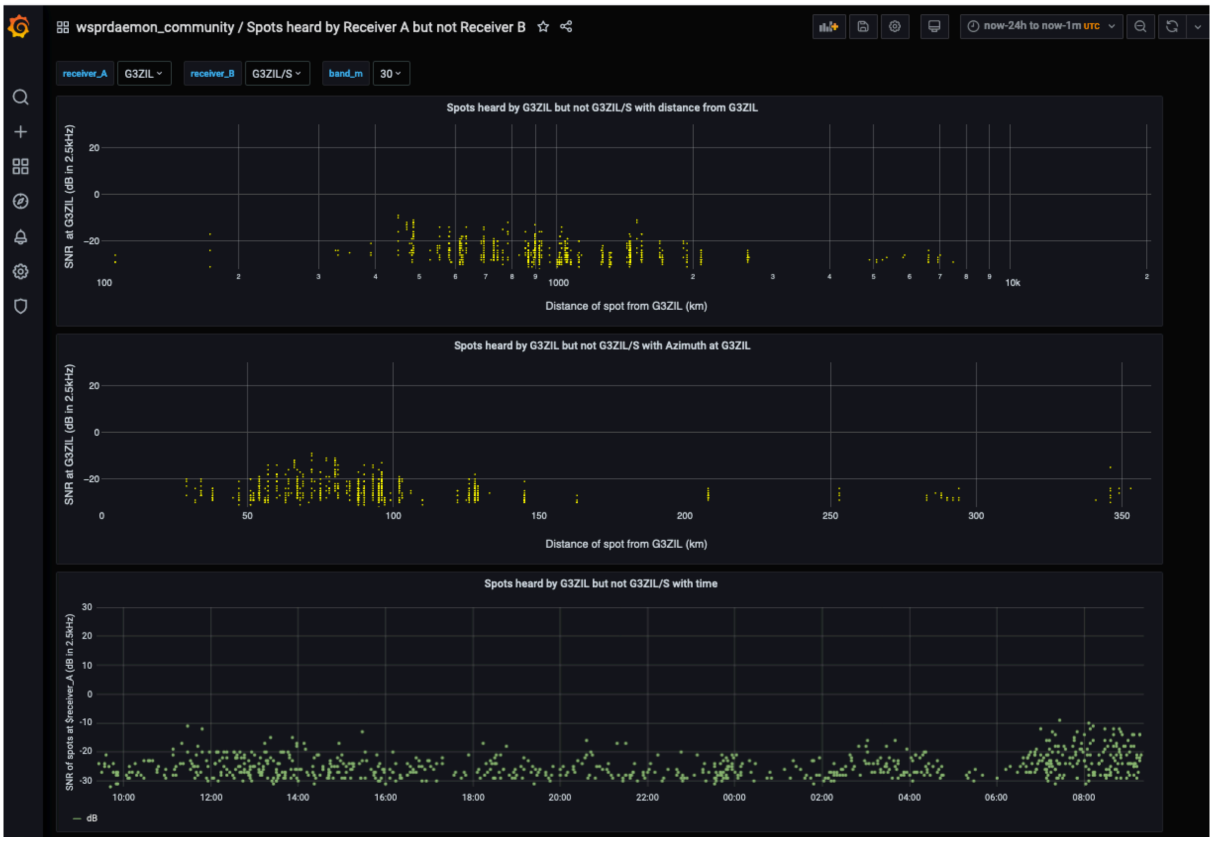
The SELECT clause for the other two panels being, for azimuth:

```
SELECT
  s1."wd_time" AS "time",
  s1.wd_rx_az as azi,
  s1."dB"
```

and as a time series:

```
SELECT
  s1."wd_time" AS "time",
  s1."dB"
```

The resulting Dashboard is shown below for spots heard on 30m by G3ZIL, using an N6GN Active Antenna tilted 22˚ from the horizontal, the down side pointing NE, and G3ZIL/S using a pair of 40m dipoles.


wsprdaemon_community / Spots heard by Receiver A but not Receiver B
now-24h to now-1m UTC
receiver_A
G3ZIL
receiver_B
G3ZIL/S
band_m
30
Spots heard by G3ZIL but not G3ZIL/S with distance from G3ZIL
SNR at G3ZIL (dB in 2.5kHz)
Distance of spot from G3ZIL (km)
Spots heard by G3ZIL but not G3ZIL/S with Azimuth at G3ZIL
Distance of spot from G3ZIL (km)
Spots heard by G3ZIL but not G3ZIL/S with time
SNR of spots at $receiver_A (dB in 2.5kHz)
dB

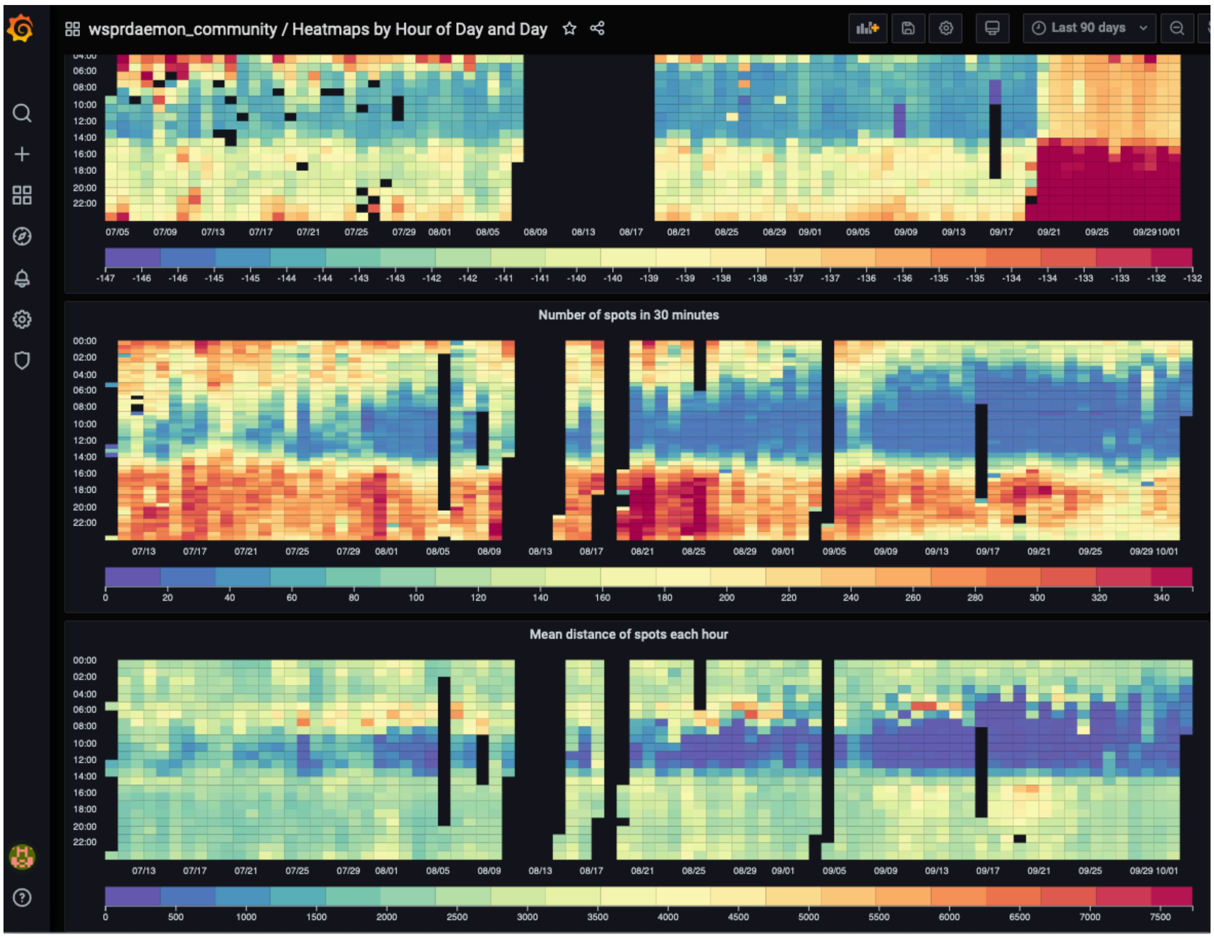
7. Hourly Heat maps

The standard Grafana Heatmap display cannot be used with time of day on the y axis, which, with day on the x axis, is a very useful form of plot for showing summary data on long time series data, e.g. noise, spot counts per hour. Luckily there is a suitable plugin - for hourly heat maps[7].

An example of what's possible is shown below; the data is from the Northern Utah SDR reported as KA7OEO-1 on 20 m and spans from 4 July 2020 to 2 October 2020. Note the times are UTC. The top panel is SNR. There appears to have been a level shift in the SNR on 19 September. It is unlikely to

[7] See https://grafana.com/grafana/plugins/marcusolsson-hourly-heatmap-panel

have been a real degradation in the SNR as the number of spots per hour, middle panel, is much the same either side of the step. The bottom panel is mean distance of the spots each hour.



As before, we'll skip the Dashboard set-up and just provide the SQL, essentially the same except for the variable in each case, the data source being wsprdaemon_tutorial. We have a receiver pull-down variable as several stations operate more than one receiver:

```
SELECT
  "time" AS "time",
  c2_level
FROM wsprdaemon_noise
WHERE
  $__timeFilter("time") and site='$Site' and receiver='$Receiver' and
band='$band_m' AND time between $__timeFrom() and $__timeTo()
ORDER BY 1
```

and for the spot count:

```
SELECT
  $__timeGroupAlias(wd_time,10m),
  count("dB") AS """No. spots"""
FROM spots
WHERE
  "Reporter" = '$Site' AND
  wd_band = '$band_m' AND wd_time between $__timeFrom() and $__timeTo()
GROUP BY 1
ORDER BY 1
```

and for the mean distance

```
SELECT
```

```
  $__timeGroupAlias(wd_time,20m),
  avg(distance) AS "km"
FROM spots
WHERE
  "Reporter" = '$Site' and wd_band='$band_m' AND
  wd_band = '$band_m' AND wd_time between $__timeFrom() and $__timeTo()
GROUP BY 1
ORDER BY 1
```

Note that for these plots I had more consistent results for the time period if I picked up the start and stop times explicitly in the where clause, that is using:

```
AND wd_time between $__timeFrom() and $__timeTo()
```

8. Propagation paths 'Telescope'

The SQL to implement panels in this Dashboard is very similar to several previous panels; it's the concept that is a little different. It is that we count the number of spots in 20 minute time bins and 200 km distance bins from all reporters within a 4-character grid and we set a look-direction for our 'Telescope' and a beamwidth. The aim of aggregating spots over a 4-character grid square is to provide more data; this certainly works in well-populated WSPR areas such as Europe and the US East Coast.

There are two variants: same look direction and beamwidth with a panel for each of two selected bands, and two different look directions, but same beamwidth, on a single band.

8.1 Propagation path 'Telescope': Two bands with heading and beamwidth

This allows you to compare the propagation along a great circle heading, over a selected beamwidth (20˚ or 30˚ is a good starting point) and see the results as a function of range and time of day and to see the day-to-day variations, and the similarities and differences over two bands. The Dashboard accesses WsprDaemon's wsprnet data source.

Obtaining just a four-character "Reporter_Grid" for the pull down variable requires using the Regex line as well as the postgreSQL query line. The query is simply:

```
select distinct "ReporterGrid" from spots;
```

and the regex[8] expression is:

```
/(....).*/
```

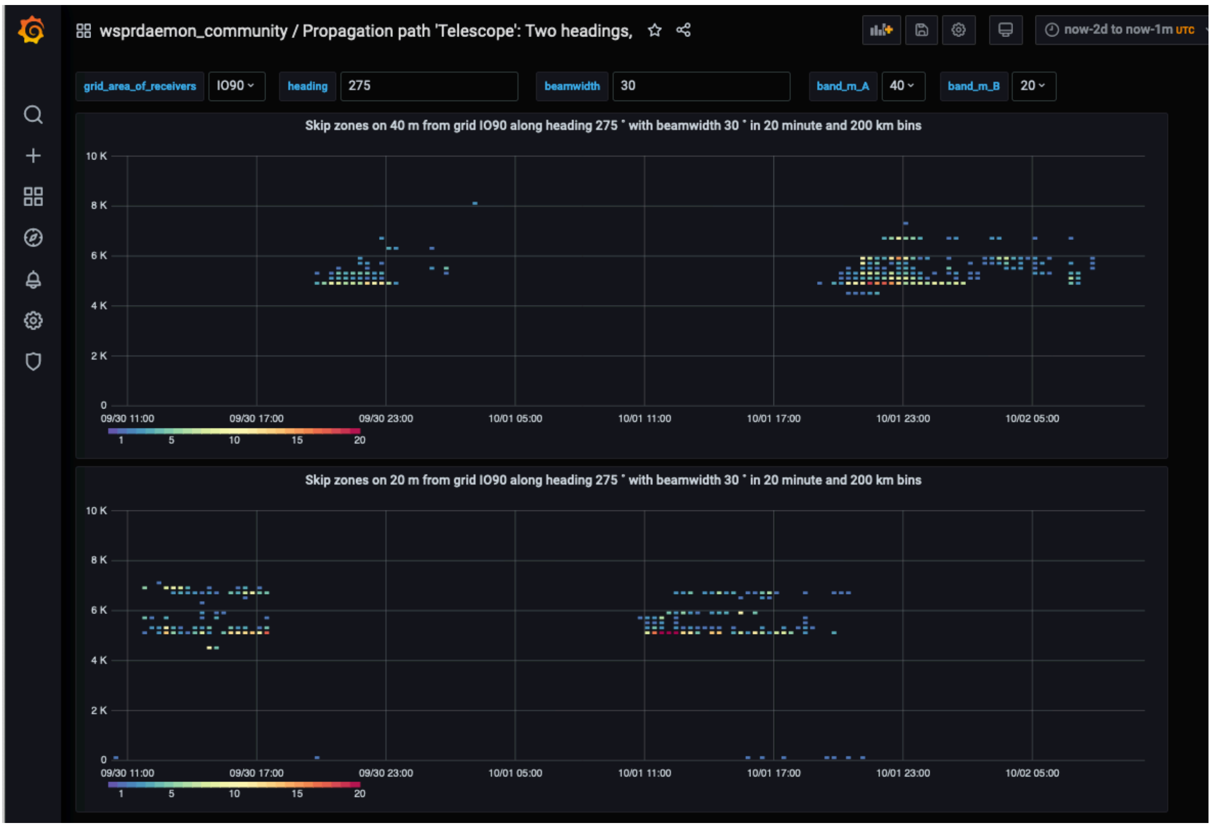
That is, extracting the first four characters (the dots).

The postgreSQL WHERE clause needs to concatenate the variable for the grid area of the receivers with the wildcard character '%' with 'like' rather than '=' as in the code below:

```
SELECT
  wd_time AS "time",
  distance AS """No. of spots"""
FROM spots
WHERE
  wd_band = '$band_m_A' AND
  "ReporterGrid" like concat('$grid_area_of_receivers','%') AND
  wd_rx_az > ($heading - $beamwidth/2) and wd_rx_az < ($heading + $beamwidth/2)
ORDER BY 1
```

Note the use of simple arithmetic to calculate the wd_rx_az range for the spots to select.

[8] There are many online sources with guidance on regex expressions, the one I used was at https://www.grymoire.com/Unix/Regular.html



wsprdaemon_community / Propagation path 'Telescope': Two headings,

now-2d to now-1m UTC

grid_area_of_receivers IO90 heading 275 beamwidth 30 band_m_A 40 band_m_B 20

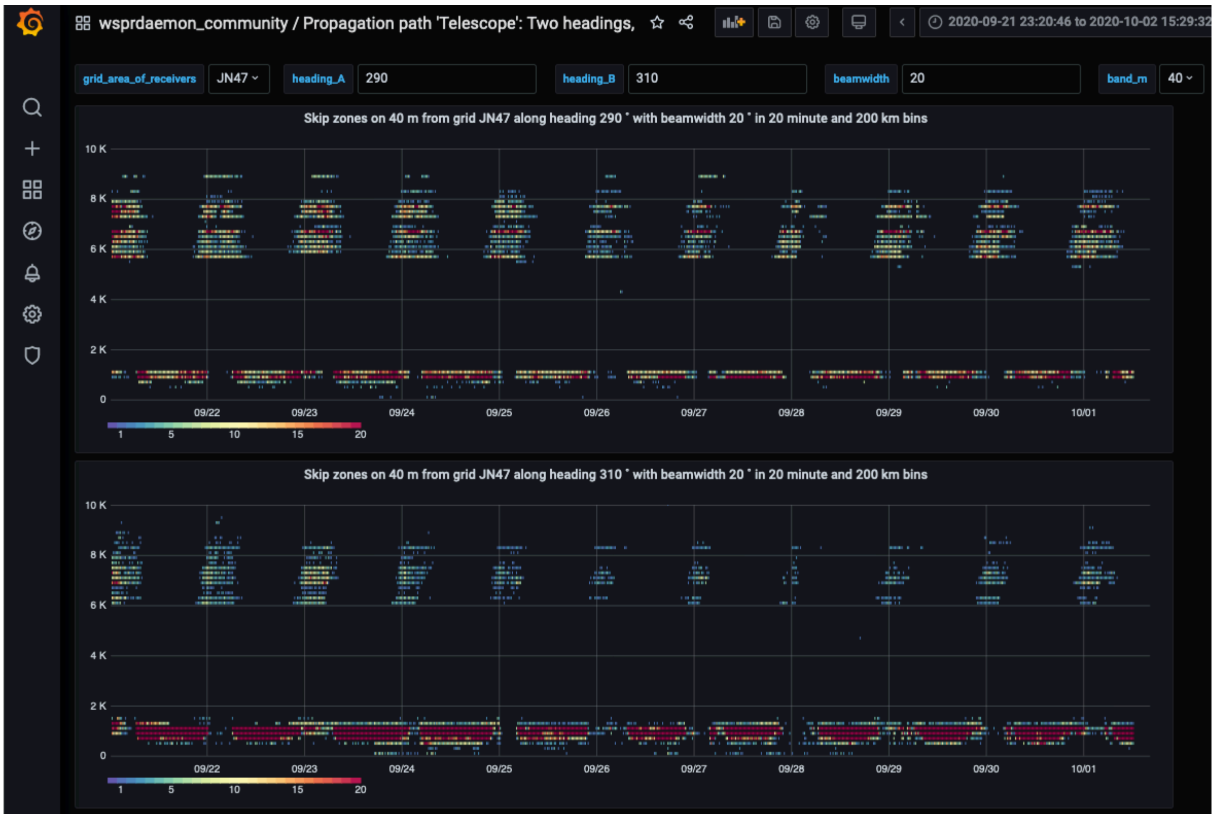
8.2 Propagation path 'Telescope': Two bands with heading and beamwidth

This Dashboard is very similar, just with variables for the two headings instead of two bands, the postgreSQL for the top panel:

```
SELECT
  wd_time AS "time",
  distance AS """No. of spots"""
FROM spots
WHERE
  wd_band = '$band_m' AND
  "ReporterGrid" like concat('$grid_area_of_receivers','%')  AND
  wd_rx_az > ($heading_A- $beamwidth/2) and wd_rx_az < ($heading_A + $beamwidth/2)
ORDER BY 1
```

In the example below the grid square if JN47, which includes OE9GHV, whose station in rural Austria has very low local noise, and the band is 40 m. The upper panel is for a look-direction of 290˚ with a beamwidth of 20˚, to the eastern seaboard of North America, as far west as the Great Lakes, the lower panel is for a look direction of 310˚ which covers the west coast. Both headings include western Europe at ranges of 1500km and less. The 14-day period includes G1 and G2 magnetic storms on 27-28 September. Heading 310˚ is closer to, or over, the southern edge of the Auroral Oval where Auroral flutter, or Frequency Spread, is likely to have exceeded the ~1.5Hz tone spacing of the WSPR signals[9].

[9] See Franke, S., Somerville, W. and J. Taylor, 2020. The FT4 and FT8 communication protocols. *QEX*, July/August 2020 for simulation results for FT4 and FT8 with frequency spread that uses an indicative 10Hz frequency spread for "high-latitude moderate" cannel simulation.


wsprdaemon_community / Propagation path 'Telescope': Two headings,
2020-09-21 23:20:46 to 2020-10-02 15:29:32
grid_area_of_receivers JN47
heading_A 290
heading_B 310
beamwidth 20
band_m 40
Skip zones on 40 m from grid JN47 along heading 290 ° with beamwidth 20 ° in 20 minute and 200 km bins
Skip zones on 40 m from grid JN47 along heading 310 ° with beamwidth 20 ° in 20 minute and 200 km bins
10 K
8 K
6 K
4 K
2 K
0
09/22
09/23
09/24
09/25
09/26
09/27
09/28
09/29
09/30
10/01
1
5
10
15
20

Part 3 Examples of graphics and interpretation with Clickhouse SQL code.

9. Clickhouse wspr.live datbase

Clickhouse[10] is a column-oriented database that can be lighting quick for many types of WSPR time-series data queries. Consequently, it is invariably faster than Timescale/PostgreSQL when used with Grafana. Arne has implemented a Clickhouse database at wspr.live of all WSPR spots since 2008[11]. Do note the names of the variables (columns) on his site, as they differ from those used by wsprnet.org and WsprDaemon.

Simple Clickhouse queries can be submitted at http://wspr.rocks/livequeries/for text response.

Clickhouse SQL[12] dialect is sufficiently different to PostgreSQL to make it useful to share several examples of code for Grafana panels with different visualisations of the same type as described in earlier section for PostgreSQL/Timescale. Some of these have beeb crafted for specific purposes but you can freely adapt given the core SQL listings.

9.1 Heatmap of spot count between grid DM and Japan

This dashboard of two panels was written for Larry, N6NC, as he was interested in what WSPR spot data may show for the path from southern California to Japan.

[10] See https://clickhouse.tech/
[11] See https://wspr.live/ for details.
[12] See https://clickhouse.tech/docs/en/sql-reference/

Create a new Grafana Panel, select Hourly Heatmap (a plugin) as the Visualisation, select Clickhouse as the data source, and the Clickhouse SQL for spots received in grid DM from Japan and Japan to grid DM is:

```
SELECT
    (intDiv(toUInt32(time), 3600) * 3600) * 1000 as one_hour,
    count() as "Spot count in one hour"
FROM rx
WHERE ilike(rx_loc, 'DM%') AND band = $band AND (ilike(tx_sign, 'JA%') or
ilike(tx_sign, 'JG%') or ilike(tx_sign, 'JH%'))
AND time > $from and time < $to
GROUP BY one_hour
ORDER BY one_hour
```

and

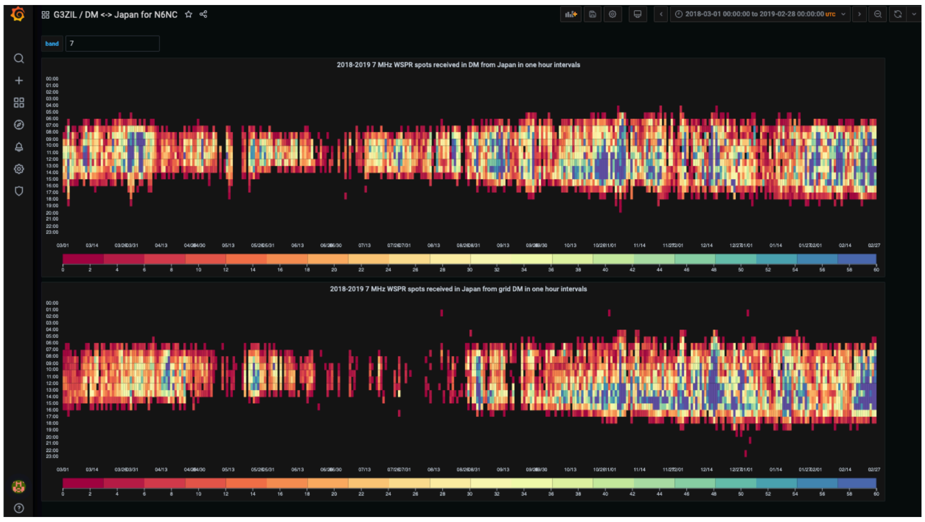
```
SELECT
    (intDiv(toUInt32(time), 3600) * 3600) * 1000 as one_hour,
    count() as "Spot count in one hour"
FROM rx
WHERE ilike(tx_loc, 'DM%') AND band = $band AND (ilike(rx_sign, 'JA%') or
ilike(rx_sign, 'JG%') or ilike(rx_sign, 'JH%'))
AND time > $from and time < $to
GROUP BY one_hour
ORDER BY one_hour
```

Commentary on the SQL:

- For this heatmap we need to count spots in one hour intervals. This is easy in a time series database such as the Timescale extensions to PostgreSQL, but not so obvous in Clickhouse. I thank Arne for the solution - the line

  ```
        (intDiv(toUInt32(time), 3600) * 3600) * 1000 as one_hour
  ```

  converts the time values to an int32 data type, integer divides by 3600, to hours, then multiplies by 3600 to seconds, to get time values that increment in 3600 second intervals and multiplies by 1000 to get 3600000 millisecond intervals.
- The count is made over this span of time values that have identical values, that is, lie within the interval defined above. Not obvious!
- The Clickhouse syntax ilike(tx_loc, 'DM%') is equivalent to PostgresSQL tx_loc like 'DM%'
- $band is a Grafana variable, here it is a text entry for frequency as the first digit of the MHz except 630 m is 0 and 2200 m is -1.
- The selection criteria for Japan are where the prefix is either JA, JG or JH.
- The inbuilt $from and $to variables pick up the wanted time window from the user selection at the head of the top panel. I've found that queries over more than one year may result in odd times appearing in a plot.


G3ZIL / DM <-> Japan for N6NC
2018-03-01 00:00:00 to 2019-02-28 00:00:00 UTC
band 7
2018-2019 7 MHz WSPR spots received in DM from Japan in one hour intervals
2018-2019 7 MHz WSPR spots received in Japan from grid DM in one hour intervals

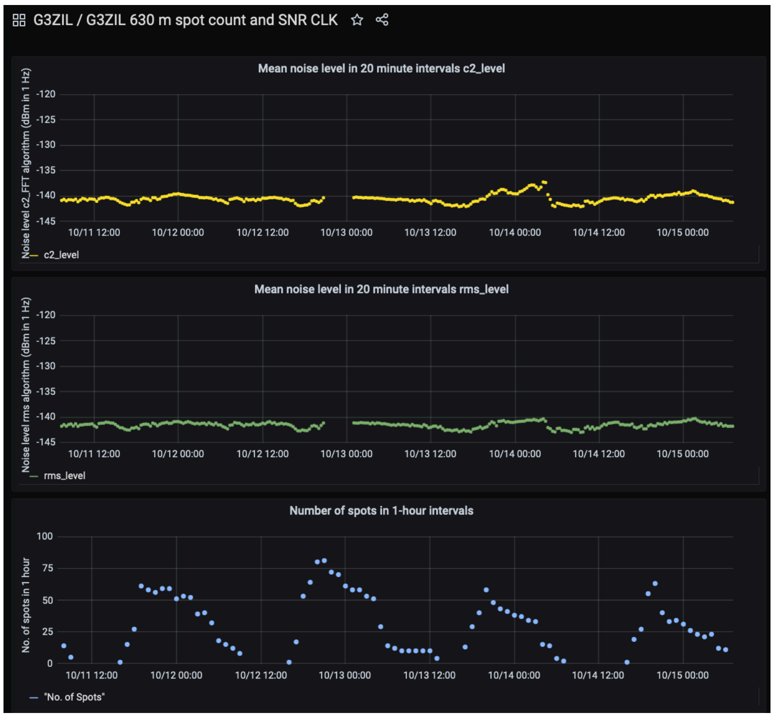
9.2 Mean noise level in 20 minute intervals and number of spots in 1 hour

Datasource Clickhouse, database wspr, table wsprdaemon_noise

```
SELECT
    (intDiv(toUInt32(time), 1200) * 1200) * 1000 as twenty_min,
    avg(c2_level) as c2_level
FROM wsprdaemon_noise
WHERE site='G3ZIL' AND band = '630'
AND time > $from and time < $to
GROUP BY twenty_min
ORDER BY twenty_min
```

Datasource Clickhouse, database wsprnet, table wsprdaemon_spots

```
SELECT
  (intDiv(toUInt32(time), 3600) * 3600) * 1000 as one_hour,
  count("SNR") AS """No. of Spots"""
FROM wsprdaemon_spots
WHERE
  band = '630' AND
  rx_id = 'G3ZIL'
AND time > $from and time < $to
GROUP BY one_hour
ORDER BY one_hour
```


G3ZIL / G3ZIL 630 m spot count and SNR CLK
Mean noise level in 20 minute intervals c2_level
Noise level c2_FFT algorithm (dBm in 1 Hz)
c2_level
Mean noise level in 20 minute intervals rms_level
Noise level rms algorithm (dBm in 1 Hz)
rms_level
Number of spots in 1-hour intervals
No. of spots in 1 hour
"No. of Spots"
10/11 12:00
10/12 00:00
10/12 12:00
10/13 00:00
10/13 12:00
10/14 00:00
10/14 12:00
10/15 00:00

This dashboard was straightforward, no (additional) Clickhouse oddities to catch me out.

9.3 SNR difference vs distance and azimuth at the receiver using plotly for XY graph

Datasource Clickhouse, database wspr, table rx (for all spots from wsprnet.org in this case).

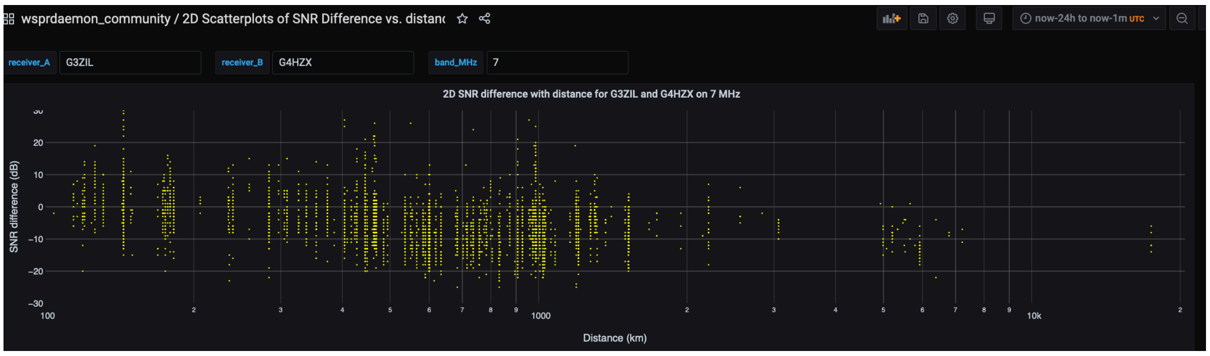
```
SELECT
    (intDiv(toUInt32(s1.time), 600) * 600) * 1000 as ten_min,
    s1.distance as km,
    (s1.snr - s2.snr) as db_difference
FROM (
    SELECT time, tx_sign, distance, snr FROM rx
    WHERE
        time > $from and time < $to AND
        rx_sign = '$receiver_A' AND
        band = '$band_MHz'
) as s1 JOIN (
    SELECT time, tx_sign, distance, snr FROM rx
    WHERE
        time > $from and time < $to AND
        rx_sign = '$receiver_B' AND
        band = '$band_MHz'
) as s2 USING (time, tx_sign)
GROUP BY ten_min, km, db_difference
```

For azimuth at the receiver simply replace references to distance & related with rx_azimuth.

Commentary on the SQL:

- Self join much as in postgreSQL / TS but Clickhouse does not do a good job of optimising the order of SQL. Hence if done as I had for postgreSQL it comes up with an over time limit error.
- Hence the use of SELECT sub-clauses both to restrict the variables being joined to just those we need and, most importantly, over just the time span we need. Else the join would be on all rows in the database, and then down-select! The sub-clause outputs get aliased as s1 and s2.
- Note the use of USING (time, tx_sign) as the columns whose data must match for inclusion in the joined table.
- No pre-populating receiver_A or receiver_B as just too many ... text entry.

- Vital point: at bottom of SQL editor select Format as to be Table else no graph for this XY (rather then time Y) plotly plot.



9.4 SNR difference vs distance and azimuth two receivers same site using plotly

Similar to the code and plot in 9.3 but this is from the wsprdaemon_spots table where there can be more than one receiver reporting under the same callsign. The default case for example is at site KFS with receiver channels connected to the NW and the SW antennas.

First, setting up the template variables is just that little bit different from postgreSQL: there is no need for a ; at the end of the select clause, indeed including it results in an error. This must be a 'feature' of the Clickhouse plugin, so me have, for example:

```
select distinct rx_id from wsprdaemon_spots
select distinct receiver from wsprdaemon_spots where rx_id='$Site'
```

In other respects the code below is essentially the same as for 9.3.

```
SELECT
    (intDiv(toUInt32(s1.time), 600) * 600) * 1000 as ten_min,
    s1.km as km,
    (s1."SNR" - s2."SNR") as db_difference
FROM (
    SELECT time, tx_call, km, "SNR" FROM wsprdaemon_spots
    WHERE
        time > $from and time < $to AND
        rx_id='$Site' and
        receiver = '$receiver_A' AND
        band = '$band_m'
) as s1 JOIN (
    SELECT time, tx_call, km, "SNR" FROM wsprdaemon_spots
    WHERE
        time > $from and time < $to AND
        rx_id='$Site' and
        receiver= '$receiver_B' AND
        band = '$band_m'
) as s2 USING (time, tx_call)
GROUP BY ten_min, km, db_difference
```

9.5 Distance statistics WsprDaemon table

This is the Clickhouse equivalent to the postgreSQL code in section 3.3. Note the method of setting a time interval as previously for Clickhouse, and template variables are set up as in 9.4. The function quantile(x) is used, with 0.5 being median and 0.25 and 0.75 for the lower and upper quartiles, not shown. Distance statistics from the wsprnet table (rx) is very similar.

```
SELECT
    (intDiv(toUInt32(time), 600) * 600) * 1000 as ten_min,
        quantile(0.5) (wsprdaemon_spots.km) as "median"
FROM wsprdaemon_spots
WHERE
  time > $from and time < $to AND
  rx_id = '$receiver_A' AND
  band = '$band_m_A'
```

```
GROUP BY ten_min
ORDER BY ten_min
```

9.6 Heatmaps of noise level, number of spots and mean distance by hour of day and day

This uses the hourly heatmap plugin, and is the Clickhouse equivalent of the dashboard of Section 7. The code below is for noise from wsprdaemon_noise, the number of spots (count) and average distance (avg) follow the same pattern.

```
SELECT
    (intDiv(toUInt32(time), 1200) * 1200) * 1000 as twenty_min,
    avg(c2_level) as c2_level
FROM wsprdaemon_noise
WHERE band = '$band_m' AND site='$Site' and receiver='$Receiver'
AND time > $from and time < $to
GROUP BY twenty_min
ORDER BY twenty_min
```

9.7 Merged receiver comparisons from WsprDaemon tables

This is the Clickhouse equivalent of the postgreSQL / TS Dashboard in section 5.2:

a) SNR difference between two receivers at same site. Note the sub-query approach as used previously.

```
SELECT
    s1.time, (s1."SNR" - s2."SNR") as db_difference
FROM (
    SELECT time, tx_call,"SNR" FROM wsprdaemon_spots
    WHERE
        time > $from and time < $to AND
        rx_id='$Callsign' and
        receiver = '$receiver_A' AND
        band = '$band_m'
) as s1 JOIN (
    SELECT time, tx_call,"SNR" FROM wsprdaemon_spots
    WHERE
        time > $from and time < $to AND
        rx_id='$Callsign' and
        receiver= '$receiver_B' AND
        band = '$band_m'
) as s2 USING (time, tx_call)
```

There is no need for GROUP BY as we are not aggregating, each spot is plotted.

b) SNR of spots heard by receiver_A but not receiver_B at the same site, same band:

```
SELECT
  s1."time" AS "time", s1."SNR"
FROM (
SELECT time, rx_id, receiver, band, tx_call,"SNR" FROM wsprdaemon_spots
    WHERE
        time > $from and time < $to AND
        rx_id='$Callsign' and
        receiver= '$receiver_A' AND
        band = '$band_m'
) as s1 LEFT JOIN (
SELECT time, tx_call,"SNR" FROM wsprdaemon_spots
    WHERE
        time > $from and time < $to AND
        rx_id='$Callsign' and
        receiver= '$receiver_B' AND
        band = '$band_m'
) as s2 USING (time, tx_call)  where empty(s2.tx_call)
```

Here LEFT JOIN is used and the where is a test for an empty s2.tx_call entry in this form, rather than the postgreSQL form s2.tx_call is null as in section 5.2

When we want to see the SNR of spots heard by receiver_B but not receiver A we use a right join and look for s1.tx_call empty, but this seems to need s2.tx_call in the first select to have the data plotted.

```
SELECT
  time AS "time", s2.tx_call, s2."SNR"
FROM (
SELECT time, rx_id, receiver, band, tx_call,"SNR" FROM wsprdaemon_spots
    WHERE
        time > $from and time < $to AND
        rx_id='$Callsign' and
        receiver= '$receiver_A' AND
        band = '$band_m'
) as s1 RIGHT JOIN (
SELECT time, tx_call,"SNR" FROM wsprdaemon_spots
      WHERE
        time > $from and time < $to AND
        rx_id='$Callsign' and
        receiver= '$receiver_B' AND
        band = '$band_m'
) as s2 USING (time, tx_call) where empty(s1.tx_call)
```

c) Simple counts of spots from each receiver in 20 minutes:

```
SELECT
    (intDiv(toUInt32(time), 1200) * 1200) * 1000 as twenty_min,
    count("SNR") as "heard by $receiver_A"
FROM wsprdaemon_spots
    WHERE
        time > $from and time < $to AND
        rx_id='$Callsign' and
        receiver = '$receiver_A' AND
        band = '$band_m'
GROUP BY twenty_min
ORDER BY twenty_min
```

and the same for receiver_B.

d) Count of spots receiver by one receiver but not the other

```
SELECT
    (intDiv(toUInt32(s1.time), 1200) * 1200) * 1000 as twenty_min,
    count(s1."SNR") as "heard by $receiver_A but not $receiver_B"
FROM (
    SELECT time, tx_call,"SNR" FROM wsprdaemon_spots
    WHERE
        time > $from and time < $to AND
        rx_id='$Callsign' and
        receiver = '$receiver_A' AND
        band = '$band_m'
) as s1 LEFT JOIN (
    SELECT time, tx_call,"SNR" FROM wsprdaemon_spots
    WHERE
        time > $from and time < $to AND
        rx_id='$Callsign' and
        receiver= '$receiver_B' AND
        band = '$band_m'
) as s2 USING (time, tx_call) WHERE empty(s2.tx_call)
GROUP BY twenty_min
ORDER BY twenty_min
```

For the other way round, just swap references to receiver_A and receiver_B and keep left join.

e) Heatmap of SNR difference between the two receivers in 20 minute and 3 dB bins

This is as (a) above, the Heatmap plug needs to be set at 20 minutes as well.

```
SELECT
    (intDiv(toUInt32(s1.time), 1200) * 1200) * 1000 as ten_min,
    (s1."SNR" - s2."SNR") as db_difference
FROM (
    SELECT time, tx_call,"SNR" FROM wsprdaemon_spots
    WHERE
        time > $from and time < $to AND
        rx_id='$Callsign' and
        receiver = '$receiver_A' AND
        band = '$band_m'
) as s1 JOIN (
    SELECT time, tx_call,"SNR" FROM wsprdaemon_spots
    WHERE
        time > $from and time < $to AND
        rx_id='$Callsign' and
        receiver= '$receiver_B' AND
        band = '$band_m'
) as s2 USING (time, tx_call)
```

f) Heatmap of distance of transmitters in 20 minute bins

```
SELECT
  time AS time,
  km
FROM wsprdaemon_spots
WHERE
  time > $from and time < $to AND
        rx_id='$Callsign' and
        receiver = '$receiver_A' AND
        band = '$band_m'
```

20 min bins set in the Heatmap Axes selections.

Azimuth at the receiver is the same except for column rx_az rather than km in the above.

g) Noise levels for the two receivers

```
SELECT
  "time" AS "time",
  c2_level as "$receiver_A c2_level"
FROM wsprdaemon_noise
WHERE
  time > $from and time < $to AND
  site = '$Callsign' AND receiver='$receiver_A'
  and band = '$band_m'
```

h) Noise level difference between two receivers

```
SELECT
    time, (s1."c2_level" - s2."c2_level") as db_difference
FROM (
    SELECT time, c2_level FROM wsprdaemon_noise
    WHERE
        time > $from and time < $to AND
        site='$Callsign' and
        receiver = '$receiver_A' AND
        band = '$band_m'
) as s1 JOIN (
    SELECT time, c2_level FROM wsprdaemon_noise
    WHERE
        time > $from and time < $to AND
        site='$Callsign' and
        receiver= '$receiver_B' AND
        band = '$band_m'
) as s2 USING (time)
```