

Creating Grafana Dashboards with live connection to the Timescale and Clickhouse WSPR databases

Gwyn Griffiths, G3ZIL

New in this version:

Section 9 on examples using the wsp.live Clickhouse database, all WSPR spots since 2008.

*Please send any comments or corrections to Gwyn Griffiths:
[gwyn at autonomousanalytics.com](mailto:gwyn@autonomousanalytics.com)*

Contents

Alphabetical list of Dashboards in the WsprDaemon Community Folder.....	3
Part 1 How to build Grafana Dashboards and Panels.....	4
0. Installation and initial connection	4
1. Create a data source.....	5
2. Create a dashboard	6
2.1 Create the first Panel.....	6
2.2 Add a second (and subsequent) time series to this Panel.....	7
2.3 Customise the Panel.....	7
2.4 Add a second Panel.....	7
2.5 Add a third Panel.....	8
2.6 Add a fourth Panel	8
2.7 Share, export and import the Dashboard and Panel csv data	9
2.8 Change Dashboard Settings	10
3. Creating a Panel with pull-down selections	10
3.1 Pull-down selections within the WHERE clause.....	10
3.2 Pull-down selections within the SELECT clause.....	12
3.3 Queries using SQL, not the Query Builder	13
4. Time series graphs of derived variables	15
4.1 SNR comparison between two reporters.....	15
Part 2 Examples of graphics and interpretation with postgresQL code.....	18
5. Time series SNR comparisons and deriving Signal Level WsprDaemon tables.....	18
5.1 Two receivers same site SNR timeseries comparison.....	18
5.2 SNR and number of spots comparison between two receivers at the same site	20
5.3 Specific sender spot count and SNR comparisons.....	23
5.4 Specific sender SNR and Signal Level from WsprDaemon tables.....	24
6. Non-time series graphs	26
6.1 SNR with distance.....	26
6.2 3D SNR Difference with distance and with azimuth at the receiver with pull-downs ..	27
6.3 2D scatterplots of SNR Difference with distance and with azimuth	28
6.4 2D scatterplots two receivers same site SNR differences with distance and azimuth ...	29
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.....	30
7. Hourly Heat maps.....	31
8. Propagation paths 'Telescope'.....	33
8.1 Propagation path 'Telescope': Two bands with heading and beamwidth.....	33
8.2 Propagation path 'Telescope': Two bands with heading and beamwidth.....	34
Part 3 Examples of graphics and interpretation with Clickhouse SQL code.	35
9. Clickhouse wspr.live database	35
9.1 Heatmap of spot count between grid DM and Japan	35

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.

Dashboard in the WsprDaemon_Community folder	Section
2D scatterplots of SNR differences vs. distance and vs. azimuth	6.3
2D scatterplots two receivers same site SNR difference vs. distance and vs. azimuth	6.4
Distance statistics - WsprDaemon table	3.3
Distance statistics - Wsprnet table	3.3
Heat maps of noise level, number of spots per hour and mean distance each hour	7
Merged receiver comparisons from WsprDaemon tables	5.2
Noise graphs	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
Two receiver same site SNR timeseries 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://logs.wsprdaemon.org:3000>

where you can log in as user **Open** and password **Open**

User Open has access to three dashboards to illustrate Grafana's capability with WsprDaemon and WSPR data:

- > Open Access Noise Graphs from WsprDaemon reporters
- > Open Access SNR Comparisons between two wsprnet reporters
- > Open Access Distance Statistics for two WsprDaemon reporters or two bands

The Open Access Noise Graph dashboard lets you choose the site_name, receiver_name, band_m and noise_type from the pull-down menus. You can also change the time interval from a pull-down list at top right. The site_names are of course only those stations using WsprDaemon to report their noise levels.

The Open Access SNR Comparison dashboard lets you select any two stations reporting WSPR spots and a band (the list is limited to the bands reported by receiver_A). The panels give you a wealth of information on the SNR differences between the sites for the same stations heard at the same time. As this access the our copy of the entire wsprnet table the initial dashboard load is slower than for the WsprDaemon table with its fewer reporters.

The Open Access Distance Statistics shows the lower and upper quartiles, the mean and the median distances of spots received in 10-minute intervals by WsprDaemon users. Two reporters and two bands can be selected, enabling a comparison for the same reporter on two bands, or different reporters on the same band.

WsprDaemon users are welcome to use the Grafana installation at

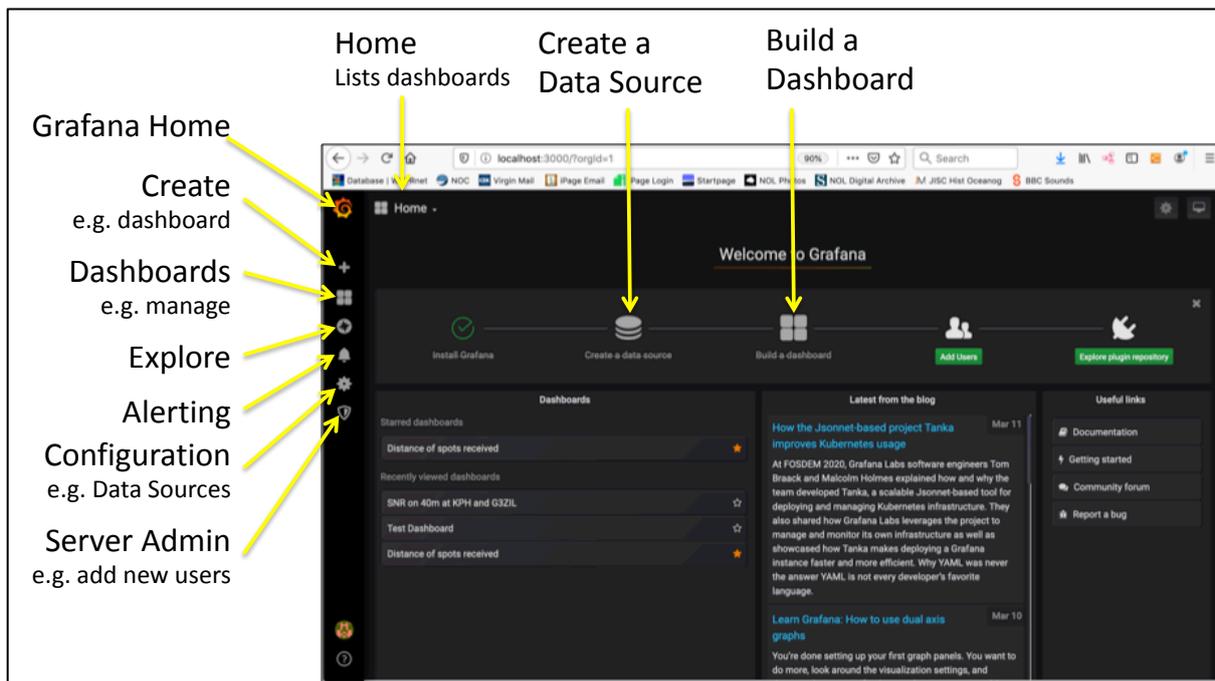
<http://logs2.wsprdaemon.org:3000>

where you can log in as user **wdread** and request a password (if not known to you) from [gwyn at autonomoussanalytics.com/](mailto:gwyn@autonomoussanalytics.com) This gives you read access to a wider range of dashboards for you to see what is possible. 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://logs2.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 raised \$50m 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!).



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 **11** or **12** appears, select it instead, PostgreSQL is also developing new versions.).

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

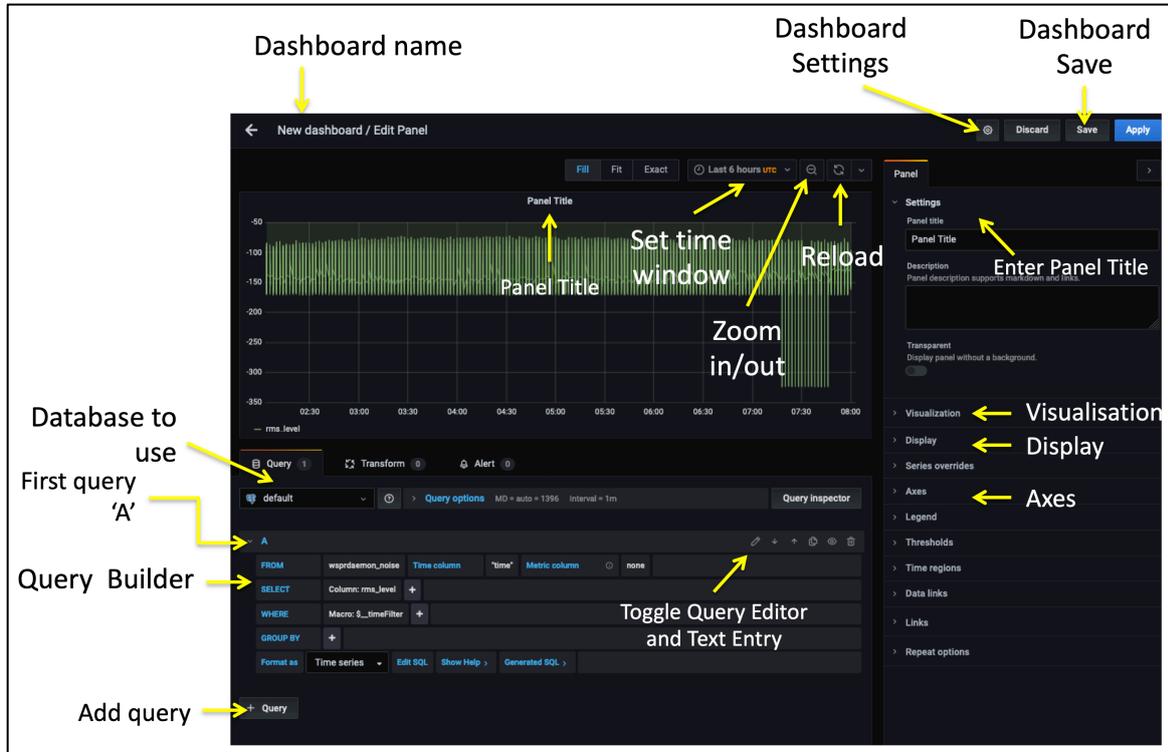
Scroll to bottom of the screen and click the green **Save & Test** button. All being well you should see a green banner with a tick and **Database Connection OK** message.

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.



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 postgresSQL 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!

The query builder is ready for Query 'A', others can be added later. As a first example, let us plot the SNR for OZ7IT at G3ZIL 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**.

Leave **Time column** as "time". 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 'G3ZIL', 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 = 'OZ7IT'**.

At this point, data should appear. If there is no data, try zooming out in time using the **zoom out** icon (magnifying glass with -).

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 OZ7IT and N1ZPY**, in the box **Panel title** under **Settings** 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 G3ZIL on 40m but, say, for **N8VIM**. Repeat the steps above to build the query. Click **Save**.

2.3 Customise the Panel.

Click on the **Visualisation** heading on the right, you'll see a number of different types of graphs - for this example we will stick with **'Graph'** the default. Close the **Visualisation** pull-down.

Click the **Display** pull-down, as you might want to use **points** rather than **lines** for this type of graph, if so, slide the **lines** switch to the left and the **points** switch to the right. For **Point Radius** you might set it to **1**. Close the **Display** pull-down.

Click the **Axes** pull-down, under the **Left Y** column you might want to set fixed **Y-min**, and perhaps **Y-max**. 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.

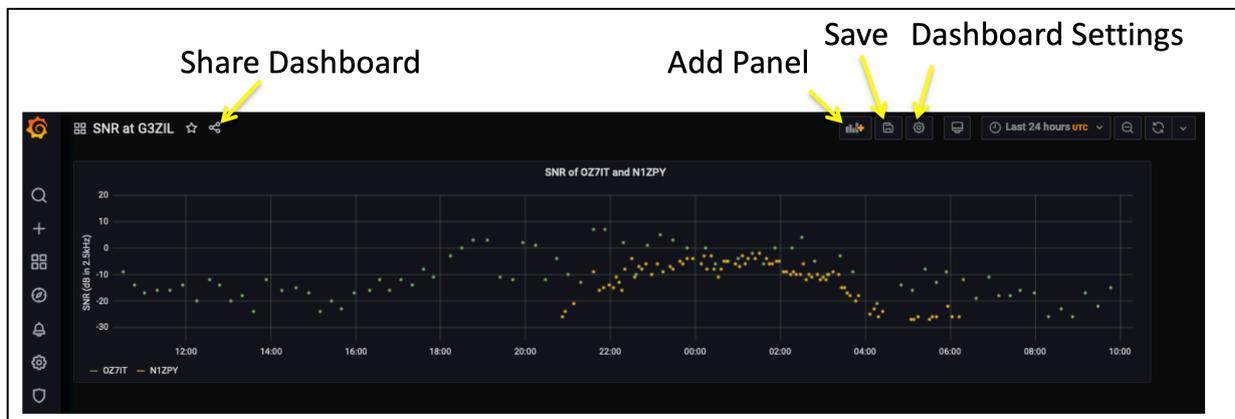
You can resize the panel by click dragging its bottom right corner.

Save the revised Panel.

2.4 Add a second Panel

Click on the **Back Arrow** at top left. It may leave you at the dashboard 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, select the **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, then 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 them **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 selected¹. 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);
```

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.

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

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**. It 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 referring 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.



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.



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

```



The example above shows three of the seven panels from this Dashboard. The receiver site is KFS³ 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 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 postgresSQL 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

³ See <http://69.27.184.62:8901/about.html> for details.

```

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)

```



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 study⁴.



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.

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" = '$receiver_A' AND
  wd_band = '$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-in⁵ 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.

⁵ See <https://grafana.com/grafana/plugins/natel-plotly-panel> for details



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.

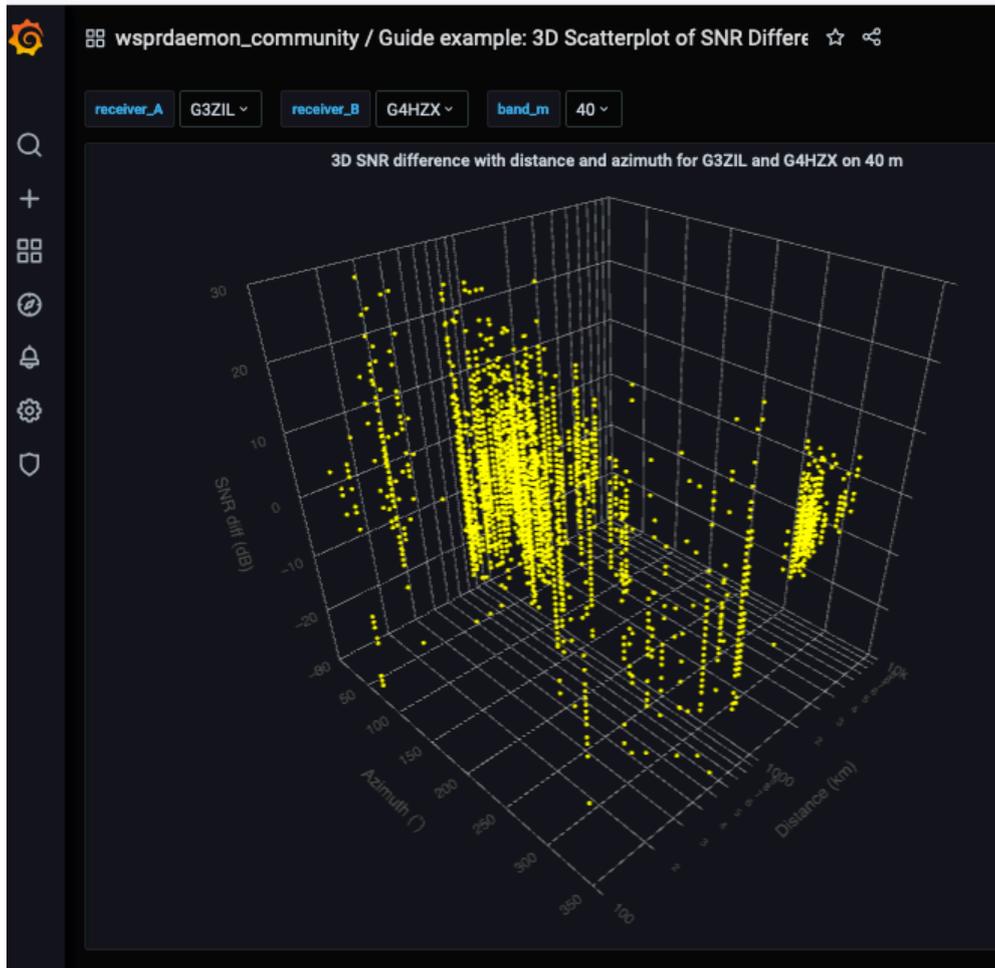
In the **Display** pull-down, under **Options** select **Type** as **Scatter (3d)**, and **Drag** as **pan**, the latter lets us use a mouse to rotate the final 3D data plot. Under the **X axis** heading set a **title** that will match the x axis we will choose in the **Traces** pull-down, e.g. **Distance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

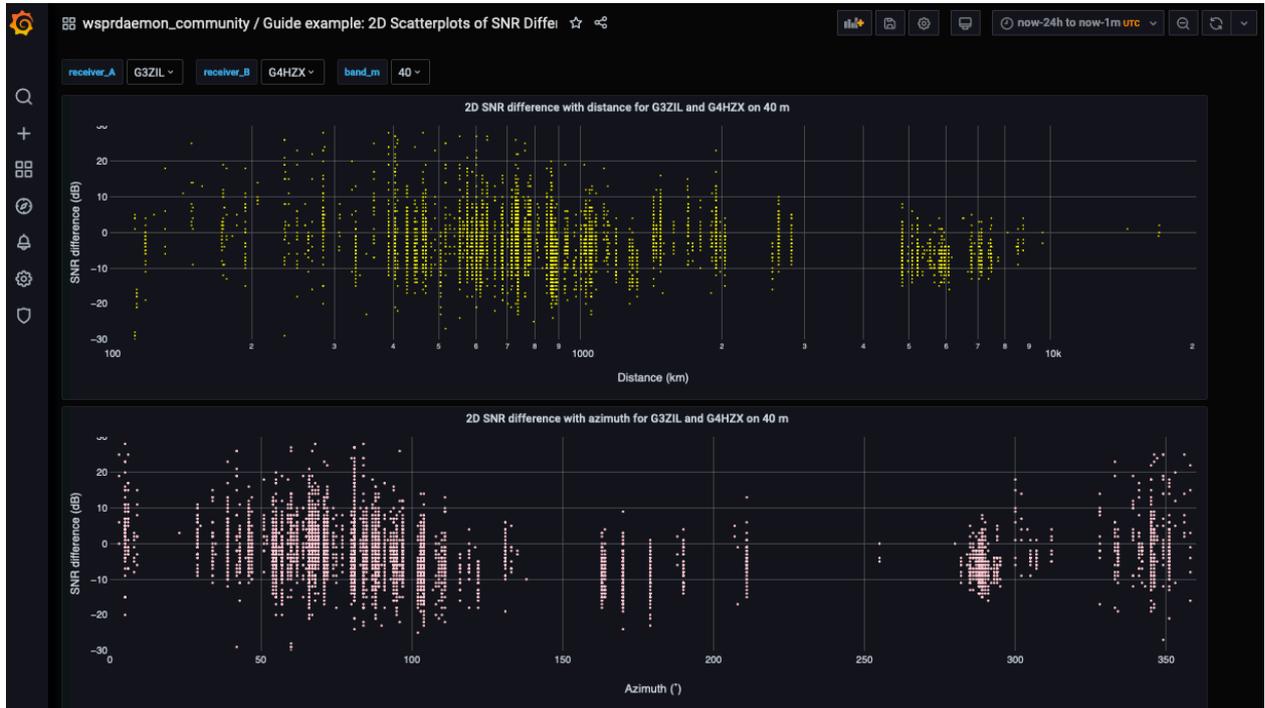
```
SELECT
  s1."wd_time" AS "time",
  s1.distance AS km,
  (s1."dB" - s2."dB") as db_difference
FROM spots s1
join spots s2 on s1.wd_time = s2.wd_time
and s1."CallSign" = s2."CallSign"
and s1."Reporter" = '$receiver_A' and s2."Reporter" = '$receiver_B'
and s1.wd_band='$band_m' and s2.wd_band='$band_m'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



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

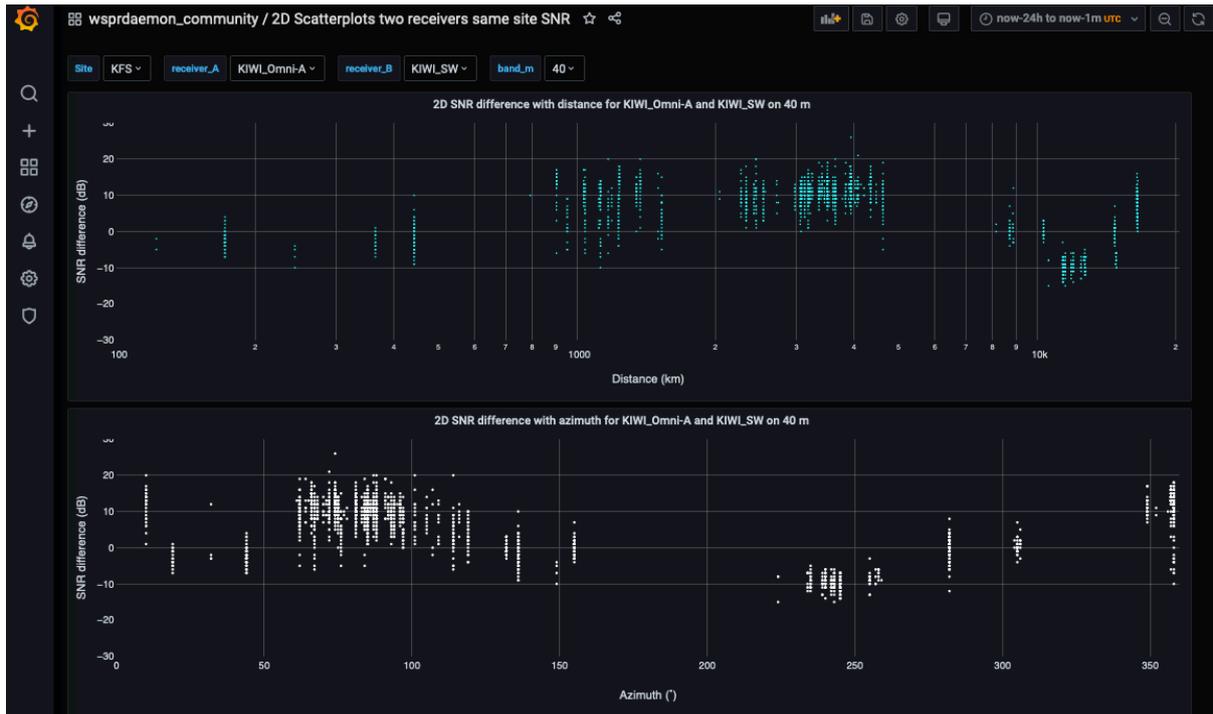
This variant is for WsprDaemon users using merged receivers reporting under one callsign. The pull-downs allow selection of the reporting Site, two receivers at that site and a band. The example below is for KFS⁶ where the two receiver channels are connected to different antennas, an Omni-directional TCI530 short and medium range log periodic and a 'SW Sector' antenna, a TCI532 medium range log periodic. The postgresSQL 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



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.



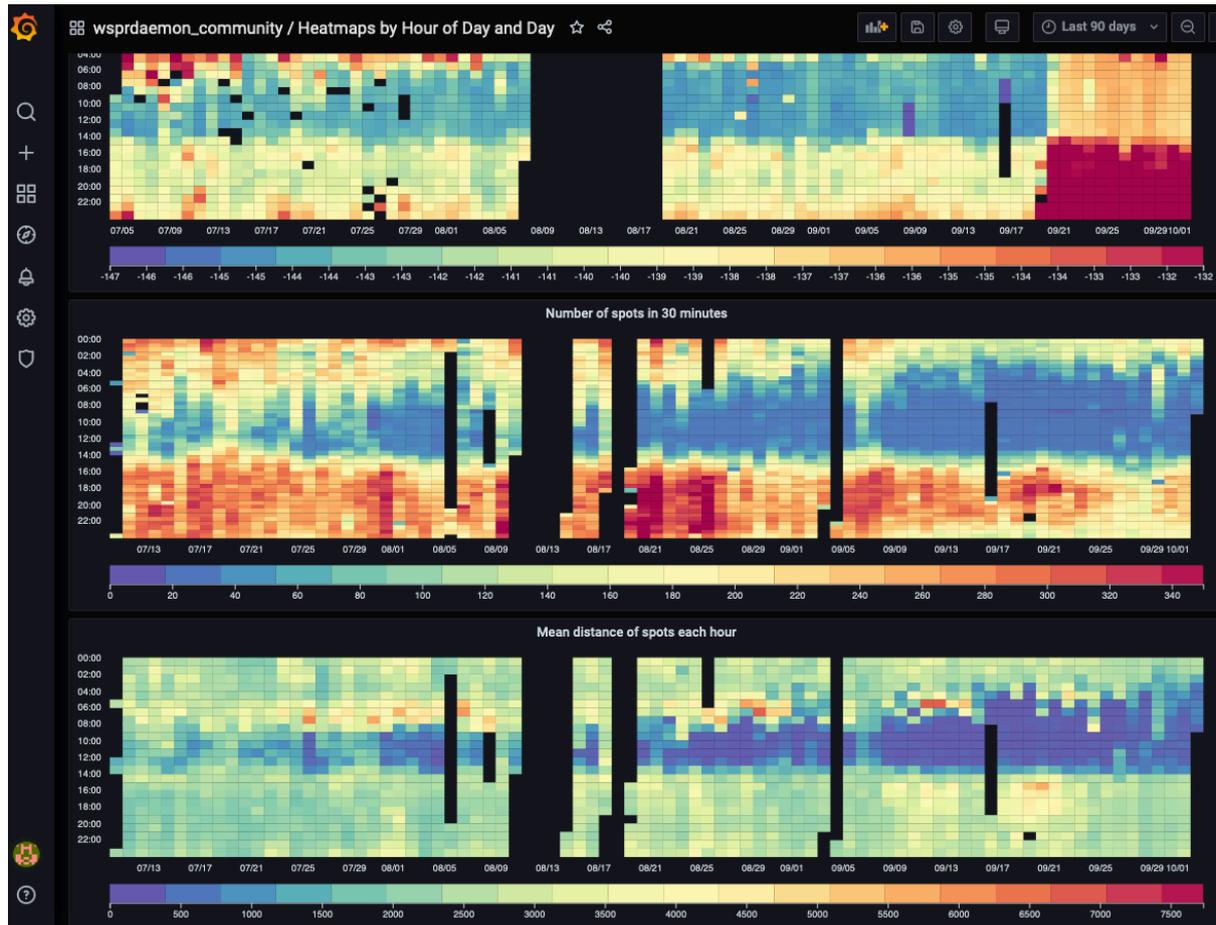
7. Hourly Heat maps

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 maps⁷.

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 have been a real degradation in the SNR as the number of

⁷ See <https://grafana.com/grafana/plugins/marcusolsson-hourly-heatmap-panel>

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 `wspdrdaemon_tutorial`. We have a receiver pull-down variable as several stations operate more than one receiver:

```
SELECT
  "time" AS "time",
  c2_level
FROM wspdrdaemon_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 postgresSQL query line. The query is simply:

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

and the regex⁸ expression is:

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

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

The postgresSQL 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>



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 is 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 signals⁹.

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" channel simulation.



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

9. Clickhouse wspr.live database

Clickhouse¹⁰ is a column-oriented database that can be lightning 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 2008¹¹. 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 SQL¹² 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 been 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.

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:

¹⁰ See <https://clickhouse.tech/>

¹¹ See <https://wspr.live/> for details.

¹² See <https://clickhouse.tech/docs/en/sql-reference/>

```

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 obvious 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 PostgreSQL `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.

