General Rules:

1. Identify what information the app will bring to the users? Is the information useful for troubleshooting, or monitoring?
2. Each app should have a landing dashboard, usually called an **Overview** dashboard that provides a high level view of the health/status of the system/product the app is for. From there, users can get more specific details in other dashboards via drill-downs.
3. In general the layout should be like this:
    a. First show the most imp info. Examples:
        i. Honeycombs to show entities that need immediate attention - e.g. hosts with high CPU
        ii. Single-value panels to show key stats

        iii. Location geo-maps
    b. Then show trend lines that map to the above info - e.g. CPU Util over time
    c. Then show tables that have more details that help someone figure out the root cause - e.g. Events that show instance type was changed
4. For dashboards, limit the number of panels to 9 or fewer. Dashboards with a higher number of panels can load slowly.
5. For logs, identify a set of **core** fields that you will parse out **for all the queries**, even though you **might aggregate on just a few fields in the end**. Some examples of common fields are user name, source/target host, application name. These fields will be very handy when new users drill-down from an aggregated panel to the raw messages (via the Messages tab), they will see more details instead of the raw messages. Furthermore, you can these fields as dashboard filters across your dashboards.
6. **Searches and Dashboard, Panel Titles Naming Convention.** Keywords and Names should be capitalized in the first letter, but not articles like a, an, the. For example: "Events by Hour", "Events Overtime". So basically, use title caps, not sentence caps.
    a. Do not use fields starting with underscores anywhere in the panels (e.g. Don't forget to rename _count to count)
7. Use search templates when you can - they can later be used as filters on dashboards (although for dashboards they don't offer any extra value - their main value is for non-Sumo savvy users when they drill-down into the search UI).

Log Search Queries [VERY IMPORTANT]

1. Use **_sourceCategory** as a metadata tag for your input data source for queries. Please DON'T use **_source, _collector** metadata tags in your queries.

   Example :
   Below query :

```
_source="kdata" and _collector="KCollector" and
_sourceCategory="Kdetections"
| json "category","etype", "eid" as category, etype,
eid
| where etype="principal"
| count by eid
```

   Should be translated to :

```
_sourceCategory="detections"
| json "category","etype", "eid" as category, etype,
eid
| where etype="principal"
| count by eid
```

   More info on _sourceCategory here

2. By default keywords in query scope use AND operator

```
_sourceCategory=Labs/SecDemo/vpcflow OR
_sourceCategory=vpc_logs_single   ACCEPT

(_sourceCategory=Labs/SecDemo/vpcflow OR
_sourceCategory=vpc_logs_single)   ACCEPT
```

   Both of the above queries have different scope.
3. Always put aggregates first before lookups

```
_sourceCategory=Labs/barracuda " TR "
| parse regex "(?<Unit_Name>[^ ]+) TR(?<Log>.*)"
```

```
| split Log delim=' ' extract 4 as Client_Ip, 2 as
Service_Ip, 3 as Service_Port
| timeslice 30m
| count by _timeslice, Client_Ip
| lookup latitude, longitude, country_code,
country_name, region, city, postal_code, area_code,
metro_code from geo://default on ip = Client_Ip
```

4. Do not use heavy operators or joins with large time ranges.
5. **Don't Use Custom Lookups.** Don't use any custom **lookup from a local file.** If you feel the lookup information is static, host it on a static location (e.g AWS S3 bucket) and use Https lookup. We provide static geolookup , port numbers, protocol numbers, and emoji that you can use.

   Refer to the docs to see examples on how to use lookup operator.

6. Always try to improve query performance by adding **keywords** to scope.

   Example, Query # 1 (without keyword):

```
_sourceCategory=="kafka"
| json auto maxdepth 1 nodrop
| if (isEmpty(log), _raw, log) as kafka_log_message
| parse field=kafka_log_message "[*] * *" as
date_time,severity,msg
| where severity in ("ERROR", "FATAL")
| count by date_time, severity, msg
| sort by date_time
| limit 10
```

   Example, Query # 2 (with keywords ERROR or FATAL) :

```
_sourceCategory=="kafka" (ERROR or FATAL)
| json auto maxdepth 1 nodrop
| if (isEmpty(log), _raw, log) as kafka_log_message
| parse field=kafka_log_message "[*] * *" as
date_time,severity,msg
| count by date_time, severity, msg
```

```
| sort by date_time
| limit 10
```

Q#2 will have better performance over Q#1 as it uses keywords in the scope of the query and drops all the events/logs which don't have those keywords before doing further processing.

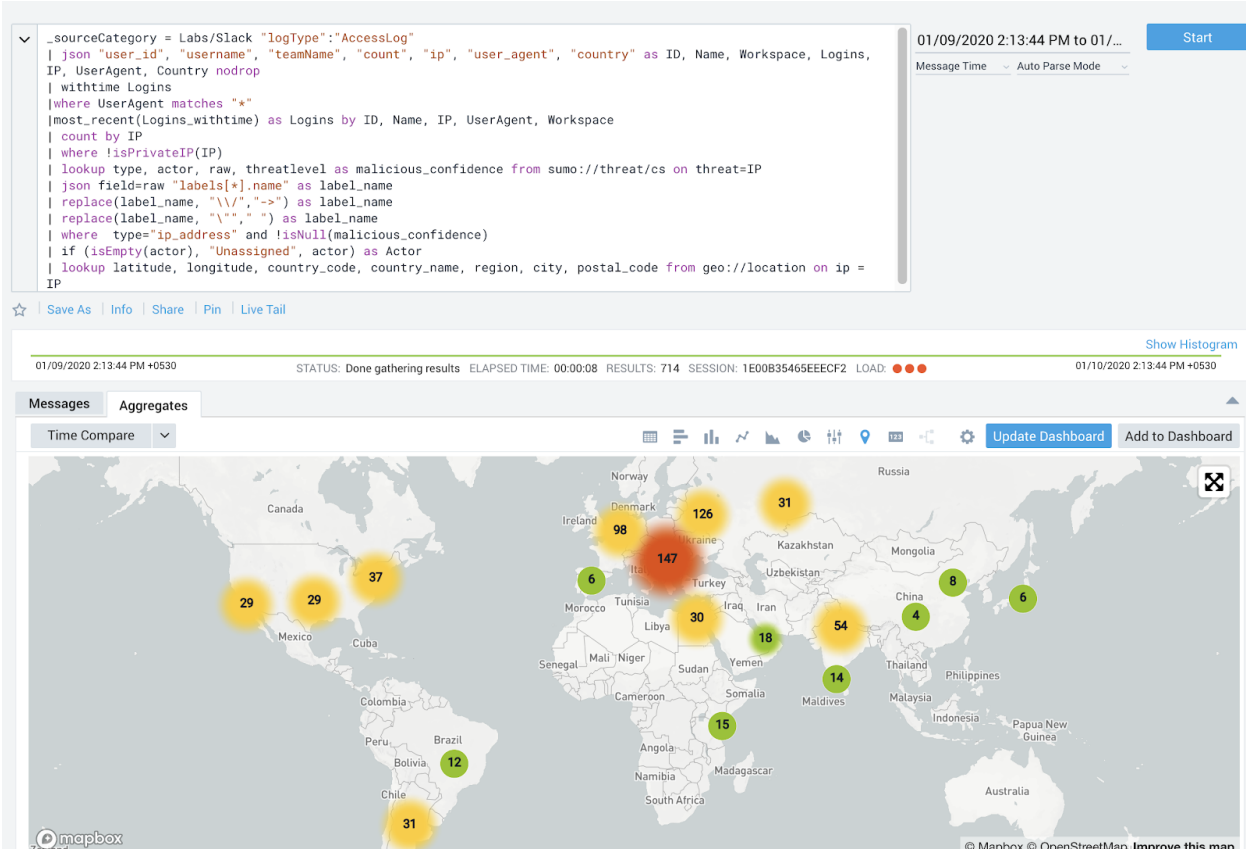
7. Panels where predict with ar model is used. It requires minimum 100 points for proper predictions else it gives warning. In 60 min, 1m timeslice will have approx 60 points. So good way is to have 30s timeslice or increase timerange to 3 hrs.

8. Be careful with parse, json they also filter the logs which do not have the matched fields use nodrop for such type of logs.

9. JSON auto extracts up to 100 fields; additional fields must be explicitly keyed. Do not use json auto if you want only a few fields.

```
|...
| json  "AwsAccountId", "Id", "GeneratorId",
"ProductArn", "CreatedAt", "UpdatedAt", "Resources",
"Severity.Normalized", "SourceUrl", "Title","Types",
"Compliance.Status" as aws_account_id, finding_id,
generator_id, product_arn, created_at, updated_at,
resources, severity_normalized, sourceurl, raw_title,
finding_types, compliance_status nodrop
```

10. While dealing with IP addresses, leverage operators like isPrivate, isPublicIP,isValidIP to optimize the query performance.



11. Provide a unit of size (KB) somewhere (Variable Name or some GUI component) when dealing with fields with units.

```
|....
| parse regex "(?<Unit_Name>[^ ]+) TR(?<Log>.*)"
| split Log delim=' ' extract 13 as Bytes_Sent, 4 as
Client_Ip, 2 as Service_Ip, 3 as Service_Port, 21 as
Response_Type, 17 as Backend_Server
| timeslice 15m
| round((Bytes_Sent / 1024),2) as Bytes_Sent
| sum(Bytes_Sent) as Total_Response_Size_KB by _timeslice
| fillmissing timeslice(15m).
```

You can also use size suffix as mentioned in docs.

Best practices for Dashboard (New)

Dashboard (New) allows you to analyze metric and log data on the same dashboard, in a streamlined user experience. Dashboard (New) is represented by  icon Vs  for the Classic dashboard.

Pro Tip: This is a great video to get started with the New Dashboard workflow.

These are some **best practices** which we follow internally at Sumo Logic while designing Sumo Logic Apps :

1. **Alignment.** Make sure your dashboard panels align nicely on the edges. Also try to fill the Dashboards as much as you can. **9 panels per dashboard** is a great rule of thumb, it avoids users not having to scroll up-down to see panels.
2. **Use Filters** at dashboards. Filter allows you to slice and dice data by important metadata/dimensions. Try to make them match all Panels if possible.
    - Here is a great example of filters at dashboards.
3. **Overview Dashboard.** An Overview Dashboard is required (see Rule 2 above), with the naming convention: **"App Name - Overview".** Yes, a space, a dash, followed by a space.
4. **Use link panels** to other dashboards to provide drill down capability to your app.
5. **Use Single View Panels (SVP)** at overview dashboards as SVPs are great at providing high level overviews. They look great on executive dashboards and can provide at-a-glance information to help decide where to inspect and

troubleshoot further. You can use the link dashboard feature to drill down from SVP to provide more detailed information.

- Here is a great example for an overview dashboard using SVP panels.

6. **Use HoneyComb panels** for use cases such as CPU, Disk, Memory, or to detect any high density information in a cluster. If applicable, use following color codes to show density/intensity of information in HoneyComb:

> Recommended Colors:
> RED: #F36644
> YELLOW: #F6C851
> GREEN: #75BF00
>
> Typical ranges for these colors are :
> 0-70 Green
> 70-80 Yellow
> 80-100 Red
> (It can vary depending upon your use case)

Here is a great example of a HoneyComb panel with color codes.

7. **Use** Map Panels for geo location based use cases.
8. **Use Text panels to organize** your panels into categories or provide more info about the panel to your viewer.
    - Here is an example of a text panel to organize the dashboard into categories.

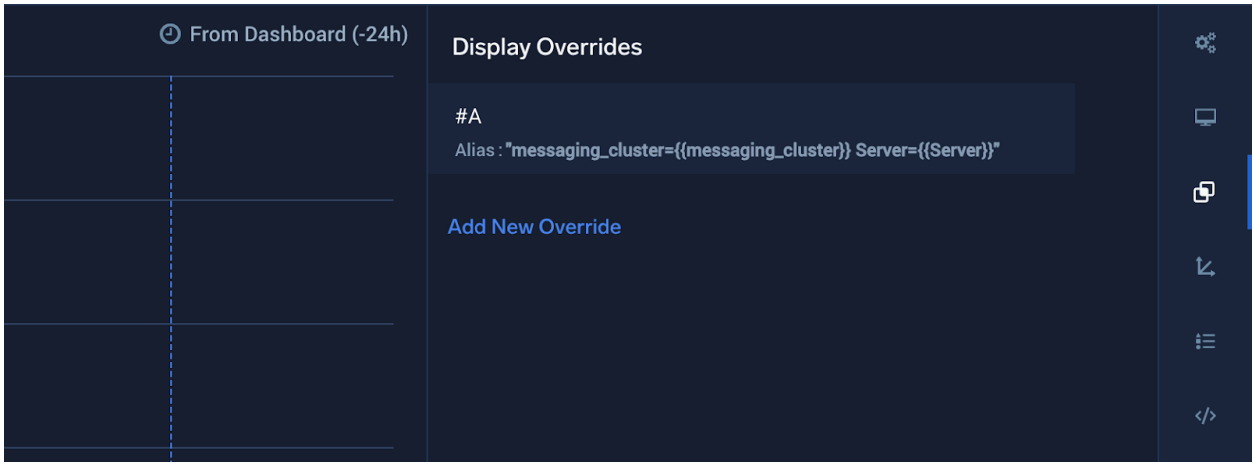
> For Heading panels, use below.
> Recommended Background Color: #DFE5E9
> Recommended Text Color: #222D3B

- Here is an example of a text panel to provide more information to the user.

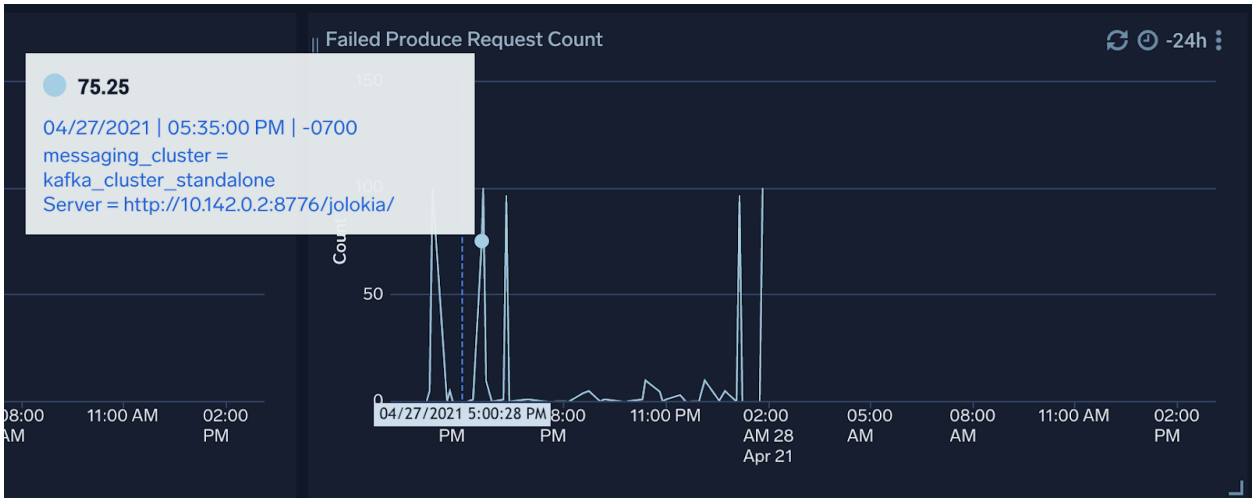
9. **Use Display Overrides** to make panels easy to read.
10. **Use Alias** at Display Overrides to specify output metadata when you hover over a panel. **Metrics** monitors allow you to specify output metadata as a variable wrapped in double curly brackets, `{{ }}`. For example, if my output metrics have `cluster, server` metadata. I can specify it in my alias as **`messaging_cluster={{Cluster}} Server={{Server}}.`**

    Note: specify those meta-data fields as display overrides which help you troubleshoot, and understand the graph in its entirety .

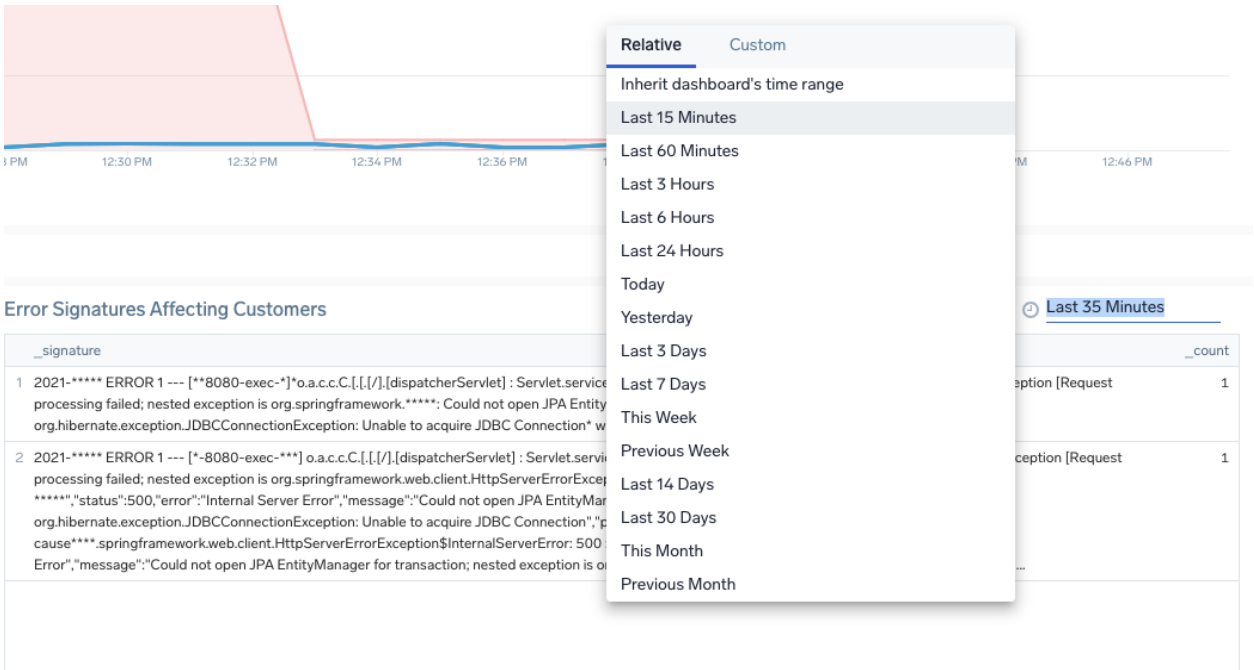
    Display Override settings:



Output metadata upon hover-over



11. If you are using pie/bar charts then you can turn off the legend as it's already a part of the chart. This will free up space in the panel.
12. **Time Ranges.** Each Dashboard should have **at most two time ranges** for the Panels. One for short-term, one for long-term data. Also short-term range should be the default one for dashboards, so the panels using this range should use the “inherit dashboard’s time range” setting (as shown below)



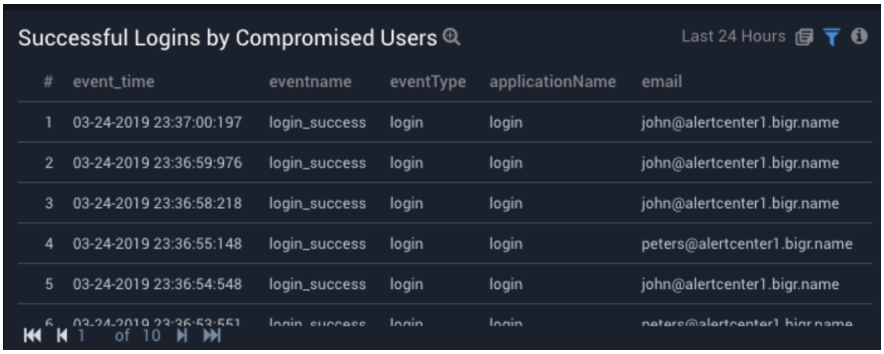
13. **Dark theme** dashboards are usually preferred over white theme

Best Practices for Panels

1. Panel name should be capitalized. Example:
    a. Login Requests by Users
    b. Top 10 Recent Findings
    c. Errors by Users
2. If your query returns lots of results (time series, bars, columns, pie), try to limit by top 10 or bottom 10 (whatever is the best for your use case). Example:
    - Top 10 Users with Failed Logins
    - Top 10 Host with Highest CPU Utilization
    - Top 10 host with Highest Disk Usage

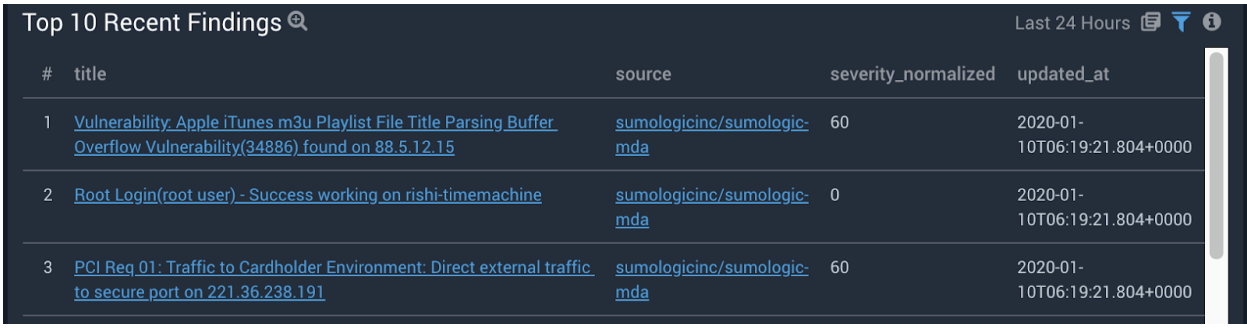
    Here is the great example of this, which shows panel `Top 5 CPU(%)/Memory(MB)/Disk(MB) consuming Pods`

3. For Pie charts, **always sort the list by count.**
4. When showing data in tables
    a. Users may not be interested in looking into Informational and Debug Log Levels. Show the Higher severity information (Emergency/ Critical, Error etc) in separate panels and all other combined in another panel
    b. The order of columns should be such that the timestamp field should come first and then important fields like email etc.



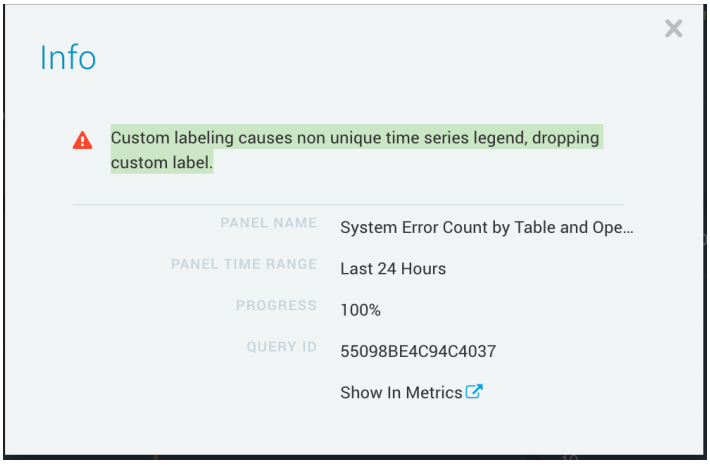
c. Columns names should be [snakecase]

d. When a column is about URLs, try to make the panel interactive by creating dynamic links like the example below



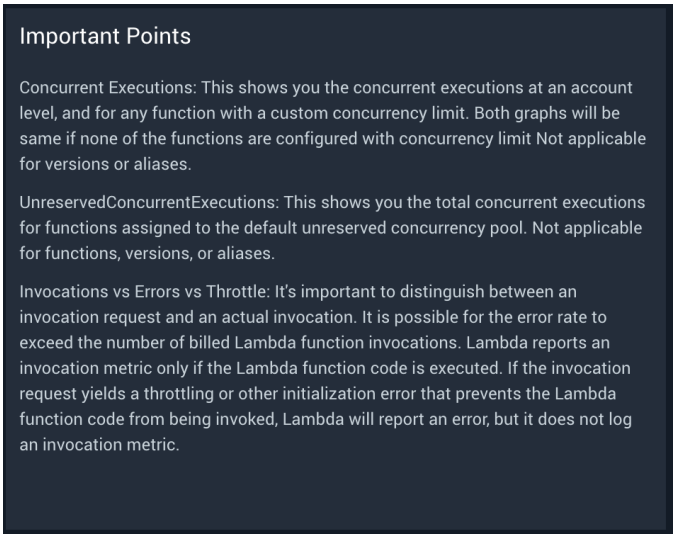
Refer to the [docs] to see examples on how to use tourl operator.

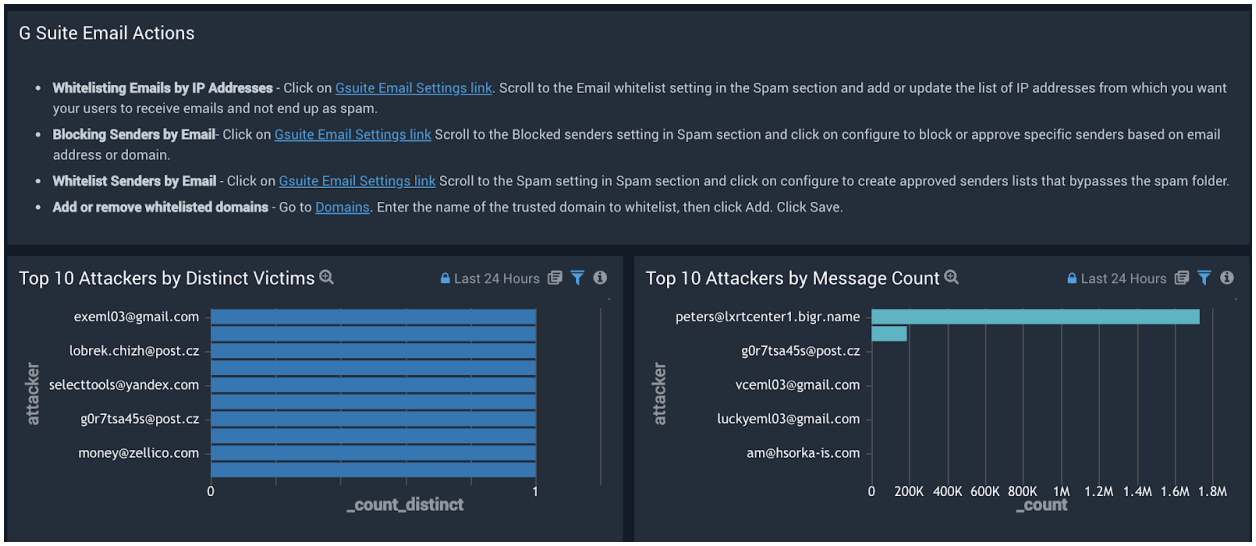
5. General Feedback for all metrics panels - Use custom labels for metrics.Do not use camel case labels, use snakecase instead - e.g. change uniqueContainers to unique_containers. Make sure that custom labelling is not dropped due to “Custom labeling causes non unique time series legend, dropping custom label”.



6. For filters if you use the same field name in logs (say for region), the filter can be shared for both logs and metrics.

7. Use Text Panels extensively to
    a. To describe important fields for better understanding of the information shown in the dashboards.
    b. To give links to action items which utilize information from the panels.





Panel Type and its Use

Undecided on which chart type to use for your use case ? Please go through the following list to understand which chart type may fit best for your use case.

Line Charts

Line charts are used to track changes over short and long periods of time. When smaller changes exist, line graphs are better to use than bar graphs. Line graphs can also be used to compare changes over the same period of time for more than one group.

Area Charts

Area Charts are very similar to line graphs. They can be used to track changes over time for one or more groups. Area graphs are good to use when you are tracking the changes in two or more related groups that make up one whole category (for example public and private groups).

Pie Charts

Pie charts are best to use when you are trying to compare parts of a whole. They do not show changes over time.

Pro Tip : For Pie charts, **always sort the list by count.**

Bar Charts

Bar Charts are used to compare things between different groups or to track changes over time. However, when trying to measure change over time, bar graphs are best when the changes are larger.

Bubble Charts

Bubble charts are used if your data has three data series that each contain a set of values. The sizes of the bubbles are determined by the values in the third data series.

Both horizontal and vertical axes are value axes. In addition to the x values and y values that are plotted in a chart, a bubble chart plots x values, y values, and z (size) values.

Combo Charts

Combo charts let you display different types of data in different ways on the same **chart**. You may display columns, lines, areas, and steps all on the same **combination chart**. **Use** them to visually highlight the differences between different sets of data.

It can be useful when comparing values in different categories, since the combination gives a clear view of which category is higher or lower. An example of this can be seen when using the combination chart to compare the 4XX Errors with the actual number of requests.

Map Charts

Map charts show the Geo-location and number of hits from data on a map.

Honeycomb Charts

Honeycombs are great at helping you find areas of significant activity amongst a set of entities, like hotspots in a cluster. These are ideal if you want to find the instance running high CPU usage amongst all the instances in your cluster.