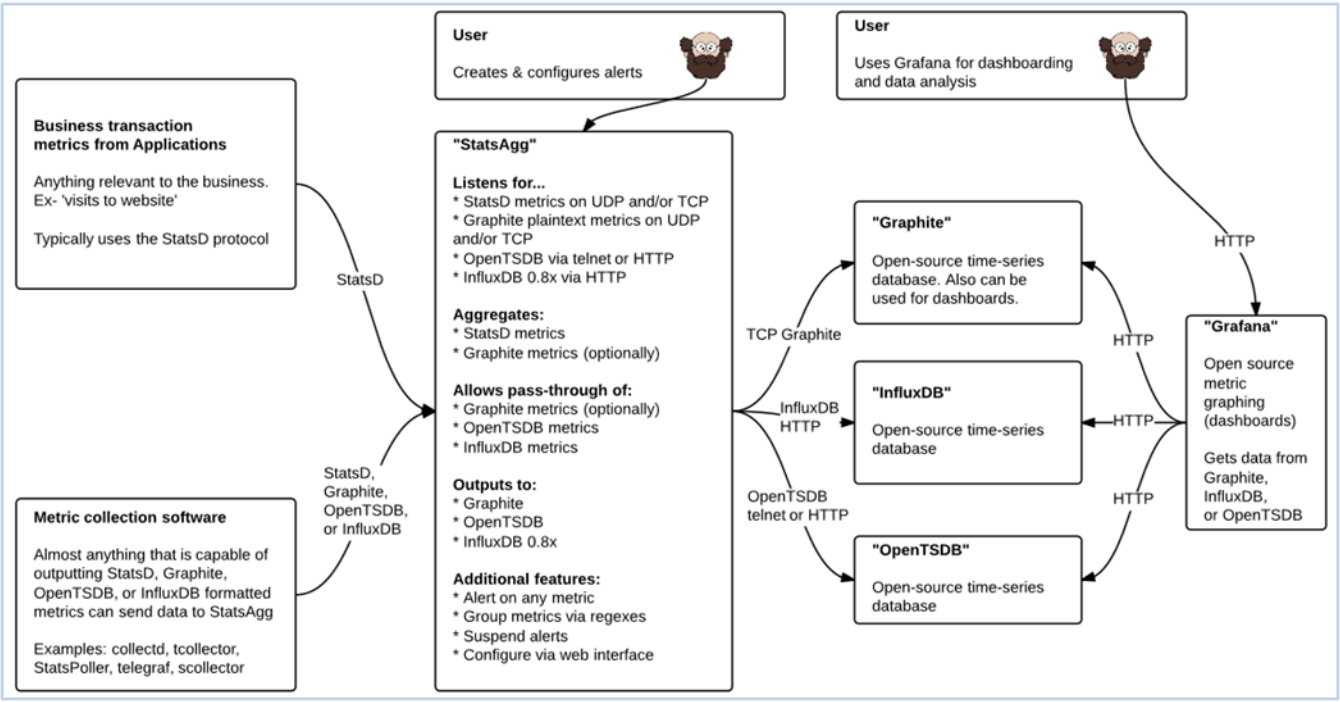
StatsAgg User Manual

Table of Contents

1 Overview

StatsAgg is a metric aggregation and alerting platform. It currently accepts Graphite-formatted metrics, OpenTSDB-formatted metrics, InfluxDB-formatted metrics (0.6x-0.8x format), and StatsD-formatted metrics.

StatsAgg works by receiving Graphite, StatsD, OpenTSDB, and InfluxDB metrics, (optionally) aggregating them, alerting on them, and outputting them to a metric storage platform. In essence, StatsAgg is a middle-man that sits between the metric sender & the metric storage applications. The 'value add' is metric aggregation (Graphite & StatsD), and a common alerting platform for all supported metric types. The diagram (see below diagram) shows an example deployment & use-case pattern for StatsAgg.



1.1 Core Features

- Receives and aggregates StatsD metrics
    a. StatsAgg contains a complete re-implementation of StatsD
    b. All metric-types are fully supported (and in some cases, enhanced)
    c. TCP & UDP support (both can run concurrently)
- Receives (and optionally, aggregates) Graphite metrics
    a. TCP & UDP support (both can run concurrently).
    b. Can aggregates into minimum, average, maximum, median, count, rate, and sum for the values of an individual metric-path (during an 'aggregation window').
    c. Aggregation works similarly to StatsD timer aggregation, but for Graphite metrics.
- Receives OpenTSDB metrics
    a. Telnet & HTTP interfaces are supported
    b. Supports GZIP compression on HTTP interface

- Outputs metrics to metric storage platforms
    a. Graphite (and Graphite compatible services)
    b. OpenTSDB (via telnet & HTTP)
    c. InfluxDB (versions 0.6x, 0.7x, 0.8x -- support for 0.9x will come in a future release)
    d. Outputting to a storage engine is completely optional; you can send metrics into StatsAgg without having them forwarded to a metric storage solution. This also means that if you're having issues with your metric storage solution, StatsAgg will still be available & capable of alerting off the metrics that it receives.
- A robust alerting mechanism
    a. Can alert off of any received/aggregated metric
    b. Regular-expression based mechanism for tying metrics & alerts together
    c. 'threshold' or 'availability' based alerting
    d. A flexible alert & metric suspension mechanism
    e. Alerts notifications can be sent via email or viewed in the StatsAgg website
- A web-based user-interface for managing alerts & metrics

1.2 Why use StatsAgg?

StatsAgg was originally written to fill some gaps that in some other popular open-source monitoring tools. Specifically...

- Graphite, StatsD, OpenTSDB, and InfluxDB do not have native alerting mechanisms
    a. Most alerting solutions for StatsD, Graphite, OpenTSDB, and/or InfluxDB metrics are provided by (expensive) SaaS venders.
    b. The alerting mechanism in StatsAgg compares favorably to many pay-based solutions.
- StatsAgg can act as a sort of 'metric transcoder' between various technologies.
    a. It allows any combination of input metrics StatsD, Graphite, InfluxDB (0.8x), and OpenTSDB metrics to be output to OpenTSDB, Graphite, InfluxDB (0.8x), etc.
    b. Support for more input & output formats will increase as StatsAgg evolves.
- StatsAgg provides an alternative way of managing servers/services/etc compared to tools like Nagios, Zabbix, etc
    a. StatsAgg allows you to break away from viewing everything from the perspective of servers/hosts. You could structure your metrics to group everything by host, but you aren't required to.
    b. Generally speaking, tools like Nagios, Zabbix, etc lack the ability to alert off of free-form metrics. Since StatsAgg uses regular-expressions to tie metrics to alerts, you can just as easily alert off of a 'free-form metric hierarchy' as you can a 'highly structured metric hierarchy'.
    c. Tools like Nagios, Zabbix, etc aren't built around having fine data point granularity (more frequent than once per minute). StatsAgg was written to be able to receive, aggregate, alert on, and output metrics that are sent at any interval.
- StatsD limitations
    a. StatsD doesn't allow the TCP server & the UDP server to be active at the same time. StatsAgg does.
    b. StatsD doesn't persist Gauge metrics, so a restart will result of StatsD 'forgetting' what the previous Gauge metric values were. StatsAgg can be used to persist Gauge values, so a StatsAgg restart won't result in Gauges resetting themselves.
- StatsAgg provides a web-based UI for managing alerts & metrics.
- Performance
    a. StatsAgg is Java-based, and has been thoroughly tuned for performance.
    b. A server with 2 cpu cores & 4 gigabytes of RAM should have no trouble processing thousands of metrics per second.

1.3 What *isn't* StatsAgg?

StatsAgg aims only to fill a gap in open-source monitoring tool stack. The biggest void that it is filling is answering the question of "how can we alert off of all these metrics that we're collecting".

StatsAgg is not, and likely never will be, a solution for:

- Metrics dashboarding. There are many great tools on the market that accomplish this. For example, Grafana.
- Metric storage. OpenTSDB, Graphite, InfluxDB, etc all are specifically made for metric storage, whereas StatsAgg is mainly meant to function as a metric 'pass-through'.

2 Installation

Note: The installation/deployment process for StatsAgg is somewhat complex (compared to StatsD). A simpler/streamlined installer for StatsAgg is planned for a future release.

2.1 System requirements

- Any OS capable of running Java 17 (Linux, macOS, Windows, etc). Linux is preferred.
- Java 17 (or newer)
- 1 CPU core for smaller StatsAgg servers. 4 CPU cores (or more) for heavy-use StatsAgg servers.
- 3GB RAM for smaller StatsAgg servers. 16GB of RAM (or more) for larger StatsAgg servers.
- 1.2GB hard drive space. The main StatsAgg deployment will take less than 100MB, but StatsAgg is configured to retain up to 1.1GB of log file data.

2.2 Installation

StatsAgg's installation steps have changed over the years as the tech stack has changed. As such, the previous steps for installing StatsAgg into Tomcat no longer apply. Future releases of StatsAgg will be installer based, but until the installers are created, manual installation is required.

Manual installation steps (Linux):

1. Install Java JRE
2. Create a folder for StatsAgg to be installed into. Recommended: /opt/statsagg
3. Unzip the statsagg release zip file into this folder
4. Change ownership of the install folder + files to be owned by whatever user you want to run StatsAgg
    a. ex: `sudo chown -R statsagg:statsagg /opt/statsagg` (assumes StatsAgg runs as the 'statsagg' user)
5. Copy the `(statsagg-install-location)/system/statsagg.service` file to `/etc/system.d/system`
    a. Make sure it is owned by 'root' (ex ` sudo chown root:root /etc/system.d/system/statsagg.service`)
    b. Change any details about `statsagg.service` that you need to (what user it runs as, JVM args, etc)
    c. Run 'sudo systemctl daemon-reload'
6. Configure StatsAgg (see below documentation)
7. Start StatsAgg with 'sudo systemctl start statsagg'
    a. Stop StatsAgg with 'sudo systemctl stop statsagg'

2.3 StatsAgg configuration

For StatsAgg to function properly, a minimum number of configuration settings must be set. This documentation will point out the most critical fields that need to be set. All settings are documented in the configuration files.

2.3.1 StatsAgg application configuration

StatAgg's application configuration file usually requires minimal configuration. That said, StatsAgg still expects to see a user-maintained application configuration file (application.properties).

1. Edit "application.properties" @ (statsagg-install-location)\conf
   Example location (Windows): C:\ StatsAgg\conf\application.properties
   - You can override the default configuration file location by adding a JVM startup arg: -*DsaAppConfLocation="YourLocation"*
     Example (Windows): -DsaAppConfLocation="C:\StatsAgg_Conf\application.properties"
2. Put any values in this file that you want to deviate from the default.
3. If you wish to change any of these values after starting StatsAgg, you will have to edit the application.properties file and restart StatsAgg.

The full list settings, default values, and examples usages can be found in StatsAgg\conf\example_application.properties

Some common settings to put in the application.properties file include:

- Output to Graphite
  - graphite_output_module_1 = true,graphite-server.some-domain.com,2003,2,1000,true,true
- Output to OpenTSDB
  - opentsdb_telnet_output_module_1 = true,opentsdb-server.some-domain.com,4242,2,true
- Output to InfluxDB
  - influxdb_v1_output_module_1 = true,http://influxdb-server.some-domain.com:8086/,2,10
- Enabling email alerts
  - alert_send_email_enabled = true
  - alert_statsagg_location = (url of your StatsAgg deployment. Ex: http://some-domain.com/StatsAgg)
  - Fill in the alert_smtp_* settings with the details of your SMTP server.
- Prefix StatsAgg output. This groups all StatsAgg output together in Graphite via a top-level directory.
  - global_metric_name_prefix_enabled = true
  - global_metric_name_prefix_value = statsagg

2.3.2 StatsAgg database configuration

StatsAgg uses a relational database for storing StatsD Gauge values, metric group info, alert info, etc. The database is NOT used for storage of metric data (Graphite/OpenTSDB/InfluxDB/etc is used for this). StatsAgg supports two different database technologies: Apache Derby Embedded & MySQL.

StatsAgg expects to see a user-created database configuration file (database.properties). If one is not specified, then StatsAgg will use an ephemeral (in-memory) database. While functional, the ephemeral database will lose all its data as soon as StatsAgg is shutdown or restarted. For users that don't care about StatsAgg's alerting capabilities & just want to use StatsAgg as a StatsD metric aggregator, this mode should work just fine. Please note that when StatsAgg fails to connect to a database that you have assigned to it, it may instead attempt to use ephemeral database.

2.3.2.1 Apache Derby Embedded

This is the default database type, and the simplest to configure. The advantages of using Apache Derby are…

- No separate server needed.
- The engine runs inside the StatsAgg JVM, so there is 0 query latency.
- Minimal configuration required.
- Upgrades to the database engine are embedded into the StatsAgg deployment.
- No worries about losing database connectivity.

The disadvantages of using this database type are…

- Due to engine limitations, this database will perform worse than MySQL when processing large numbers of Gauge metrics.
- You can't connect to the database from an external client while StatsAgg is running. This is a limitation inherit in Apache Derby Embedded.

- Less community support than MySQL.
- Backup mechanisms are less robust.
- The database engine uses up JVM heap space, leaving less room for the rest StatsAgg's data.

If you choose to use Apache Derby, then the steps to configure StatsAgg to use it are…

1. Edit "database.properties" @ (statagg-install-location)\conf
   Example location (Windows): C:\StatsAgg\database.properties
   - You can override the default configuration file location by adding a JVM startup arg: -*DsaDbConfLocation="YourLocation"*
     Example (Windows): -DsaDbConfLocation="C:\StatsAgg_Conf\database.properties"
2. Create a folder somewhere on your StatsAgg server for the database data. StatsAgg must have read/write permissions to this folder. In the database.properties file, assign the location of this folder to the variable 'db_localpath'.
   - Failure to perform this step may lead to Apache Derby attempting to create the database at a location of its choosing (ex – at the root of your filesystem).
3. Review the "example-database.properties" file for specific settings/configurations for the database configuration.

2.3.2.2 MySQL

MySQL can be used instead of Apache Derby. MySQL is recommended for more customers that require a more enterprise-friendly database. The advantages of MySQL are…

- Faster than Apache Derby at processing amounts of StatsD Gauge metrics.
- Database processing can be performed on a separate server from the StatsAgg server. This has the overall effect freeing up resources on the StatsAgg server.
- You can connect to the database from an external client while StatsAgg is running.
- Backup mechanisms are robust.

The disadvantages of using this database type are…

- Runs as a separate process from StatsAgg. If the database goes down (or becomes unavailable to StatsAgg), StatsAgg will not function properly.
  - Without a restart, StatsAgg doesn't guarantee a complete recovery from a database connectivity issue. It is recommended that StatsAgg be restarted if there is a MySQL connectivity loss.
- Complicated configuration (compared to Apache Derby).

If you choose to use MySQL, then the steps to configure StatsAgg to use it are…

4. Install/configure MySQL 5.7+
   - MySQL's configuration can get really complex. The MySQL configuration settings listed below are required to ensure that StatsAgg functions properly. Note that there are MANY other settings that can be set on MySQL to achieve faster performance, greater durability, etc, but those are out-of-scope for this documentation.

     ```
     #In my.ini on Windows. On Linux, set in my.cnf.
     explicit_defaults_for_timestamp = 1
     ```

5. Connect to MySQL & run "create database statsagg;"
6. Create appropriate usernames, passwords, and privileges on the MySQL server. Make sure that the user that the StatsAgg application connects as has full grants/privileges on the 'statsagg' database that was created in step 1.
7. Edit "database.properties" @ (statagg-install-location)\conf
   Example location (Windows): C:\StatsAgg\database.properties
   - You can override the default configuration file location by adding a JVM startup arg: -*DsaDbConfLocation="YourLocation"*
     Example (Windows): -DsaDbConfLocation="C:\StatsAgg_Conf\database.properties"

8. Review the "example-database.properties" file for specific settings/configurations for the database configuration.

Notes

- StatsAgg expects/requires a low-latency/high-bandwidth connection to the MySQL database. If such a connection isn't available, then it is recommended that Apache Derby Embedded be used instead.

2.3.2.3 Apache Derby vs MySQL

The author of StatsAgg generally recommends Derby over MySQL. Derby is simpler to setup, has adequate performance for almost all use-cases, is incapable of being unavailable (at least independently of the application), and is more portable. MySQL is generally only recommended for users that want to standardize around a MySQL tech-stack (which may make backups easier, etc), or are trying to achieve higher performance (depending on the use-case, MySQL can be faster).

2.4 Updating/upgrading

StatsAgg can be upgraded relatively easily, but it still requires some hand-holding. As the product matures, the upgrade process may get easier.

Upgrade by following these steps:

1. Stop StatsAgg (this can sometimes take over 30 seconds)
2. If you are using the 'DsaAppConfLocation' and 'DsaDbConfLocation' JVM options to point to configuration files outside of the StatsAgg deployment, then you can skip this step. If you use application.properties & database.properties files are inside of the application deployment (@ (statsagg-install-location)\conf), then you must save these files somewhere for later access.
3. Update the StatsAgg's main jar file & lib files (delete old lib files & replace with the ones from the new release).
4. Update the application.properties and/or database.properties files as needed.
    a. Review 'example_application.properties' & 'example_database.properties' to see if any changes to your configuration settings are needed.
5. Review the release notes to make sure that there isn't anything new/important that you have to know about for the new release.
6. Start StatsAgg

2.5 Configuring for High Availability

Due to the nature of how StatsAgg aggregates, stores, and alerts on metrics, StatsAgg is not capable of being truly highly available. That is to say, the architecture only supports one StatsAgg server handling incoming metric traffic at any given time.

In the event of a server failure, downtime can be minimized if StatAgg is deployed in a way similar to this:

- Deploy StatsAgg to two separate servers. Configure them identically, but only one start StatsAgg on one of the servers. The server that is running StatsAgg will be considered the 'active' node, and the server that is not running StatsAgg will be considered the 'standby' node.
    - Use DNS (or so other traffic direction mechanism) to direct all traffic to the 'active' StatsAgg node.
    - In the event of a failure on the 'primary' node, start StatsAgg on the 'standby' node & switch the DNS (or so other traffic direction mechanism) to point to the 'standby' node.
- In a normal active/standby deployment, StatsAgg should use MySQL for its database.
- StatsAgg's MySQL database can be configured to be highly available via several mechanisms.

- MySQL can run in a primary/replica configuration, where the slave can be promoted to the replica in the event of hardware failure. StatsAgg should always point to whichever node is the primary node.
  - If this mechanism is used, then make sure to use 'row' based replication. Also, be aware that in the event of a failover from the primary to the replica, there could be a small amount of data loss.

3 Metric inputs/outputs

3.1 Terminology

StatsAgg supports multiple metric formats, and this documentation tries to use the naming convention that is appropriate for each metric-type. This makes the documentation easier for people that know their specific metric format, but it also means that the same thing can go by several different names. Listed below are some of the naming conventions used.

'The identifier of a metric'
StatsD calls this a 'bucket'
Graphite calls this a 'metric-path'
OpenTSDB calls this a 'metric', through OpenTSDB also supports tags on metrics.
Many metric storage engines call this a 'metric-series'
StatsAgg refers to this as a 'metric-key'

'Metric-key + metric-value'
This combination is often referred to as a 'data point'.
This combination is sometimes referred to as a 'metric'

3.2 Graphite metrics

3.2.1 Overview

Graphite is a popular tool for visualizing & storing time-series data. Among open-source tools that provide this functionality, Graphite was a pioneer. Many other tools are compatible with the Graphite format, so providing native support for Graphite metrics was a core design goal in StatsAgg.

By default, StatsAgg can listen for Graphite 'plaintext' metrics on ports 2003 & 22003 (TCP & UDP, on both ports). Port 2003 is used for Graphite 'passthrough' metrics, and port 22003 is used for Graphite 'aggregated' metrics. For more information on the distinction between 'passthrough' and 'aggregated', please read about 'passthrough' and 'aggregated' graphite metrics.

StatsAgg does not currently support receiving Graphite metrics in the 'pickle protocol'.

3.2.2 Metric format

The Graphite format is: metricPath metricValue unixEpochTime\n
Example: some.descriptive.name.for.a.metric 123.643 1408334418\n

Notes:

- Every individual metric MUST end with a newline character.
- The maximum size of a single metric (metric + timestamp + value + whitespace) must be less than 32767 characters long.
- StatsAgg can handle metric values of almost any size & scale, with a few limitations.

    - Graphite may not be able to handle extremely large and/or small numbers.
    - 'Aggregated' Graphite metrics will round to 7 decimal places.
- Sending multiple metrics in a single TCP or UDP push is allowed, but every individual metric must end in a newline character.
    - Example: some.descriptive.name.for.a.metric1 123 1408334418\nsome.descriptive.name.for.a.metric2 456 1408334418\n
    - Sending more than a few metrics per UDP push is not recommended (due to limitations in the UDP protocol).
- In the actual Graphite program, several characters are not allowed in metric-paths. Some examples include (but are not limited to): '%', '\', '/', '[', ']', '{', '}', ' '.
    - It is up to the program sending the Graphite metrics to use an 'allowed' character set. StatsAgg will not reformat/correct metrics that use characters that don't play well with Graphite.
- In the actual Graphite program, the '.' character is used to create folders between the metric-path descriptors.
    - This is only relevant if you intend on outputting your metrics to Graphite for visualization.
    - Example: folder1.folder2.value1 123 1408334418\n folder1.folder2.value2 456 1408334418\n
    - folder1
        - folder2
            * value1=123
            * value2=456
- StatsAgg doesn't know/care that the '.' character is used for creating folder hierarchies, nor does it know/care about other 'special' characters. These are distinctions that matter to the actual Graphite program, but not to StatsAgg.

3.2.3 'Passthrough' metrics

By default, port 2003 is used for Graphite 'passthrough' metrics. The inputs to 'passthrough' Graphite metrics are normal/plain Graphite metrics. The 'passthrough' aspect refers to how StatsAgg handles the metrics. 'Passthrough' means that StatsAgg performs no aggregation on the metrics. All metrics that are received as inputs are also capable of being outputted. In other words, every metric that is sent to StatsAgg is simply 'passing through'.

Why use 'passthrough' Graphite metrics?

- You don't benefit from aggregating metrics if you send Graphite metrics less frequently than the StatsAgg 'flush interval'.
- Aggregating Graphite metric uses far more CPU resources on StatsAgg servers than 'passthrough' Graphite metrics does.
- Graphite 'passthrough' metrics are the most lightweight metric type that StatsAgg is capable of processing.
- StatsAgg's built-in alerting mechanism can alert on Graphite 'passthrough' metrics.

Notes:

- In the actual Graphite program, metrics that have the same metric-path & timestamp overwrite each other. StatsAgg doesn't know/care about this, and if metrics are sent with the same metric-path & timestamp, StatsAgg will pass all the metrics through.

3.2.4 'Aggregated' metrics

By default, port 22003 is used for Graphite 'aggregated' metrics. The inputs to 'aggregated' Graphite metrics are normal/plain Graphite metrics. The 'aggregated' aspect refers to how StatsAgg handles the metrics. 'Aggregated' means that StatsAgg will aggregate the Graphite metrics that are sent to it.

Aggregation works as follows:

1) StatsAgg receives Graphite metrics.
2) Every 'flush interval' (10 seconds, by default), StatsAgg will aggregate the Graphite metrics.
    a. The values from identical metric-paths are aggregated together.

    b. StatsAgg will output the following aggregated metrics: Count of metrics received for a given metric-path, rate (per second) of metrics received for a given metric-path, average value, median value, minimum value, maximum value, sum of values
    c. All values are rounded to 7 decimal places.
    d. The timestamps are averaged together.
3) Only the aggregated values are capable of being alerted on & outputted. The original metric values are discarded after they are aggregated.

Example:

folder1.folder2.value1 4 1408334415\n
folder1.folder2.value1 2 1408334416\n
folder1.folder2.value1 8 1408334417\n
folder1.folder2.value1 10 1408334418\n
folder1.folder2.value1 6 1408334419\n

is aggregated into (assuming a 'flush interval' of 10 seconds)…

folder1.folder2.value1.Count 5 1408334417\n
folder1.folder2.value1.Rate-Sec 0.5 1408334417\n
folder1.folder2.value1.Avg 6 1408334417\n
folder1.folder2.value1.Median 6 1408334417\n
folder1.folder2.value1.Sum 30 1408334417\n
folder1.folder2.value1.Min 2 1408334417\n
folder1.folder2.value1.Max 10 1408334417\n

On an aggregation, all the metric values associated with a metric-path are aggregated together & the following metrics are outputted:

- Count (the count of 'how many metrics were in a metric-path's list of metrics')
- Rate-Sec (the per-second rate of 'how many metrics were in a metric path's list of metrics')
    - Rate-Sec = Count/flush_interval
- Avg (the average value of a metric-path's list of metrics)
- Median (the median value of a metric-path's list of metrics)
- Sum (the sum of a metric-path's list of metrics)
- Mix (the minimum value of a metric-path's list of metrics)
- Max (the largest value of a metric-path's list of metrics)

Why use 'aggregated' Graphite metrics?

- In the actual Graphite program, metrics that have the same metric-path & timestamp overwrite each other. Aggregating metrics in StatsAgg preserves the range of values for the metric-path that were sent during the 'flush interval'.
    - Warning: if the current 'averaged metric timestamp' for a metric-path is the same as a previously outputted 'averaged metric timestamp', then the new aggregated metrics will overwrite the previous aggregated metrics in the actual Graphite application.
- StatsAgg's built-in alerting mechanism can alert on the aggregated metrics.
- Performance – if you're sending lots of metrics with the same metric-path, StatsAgg may perform better you aggregate these metrics (as opposed to using the StatsAgg Graphite 'pass-through' interface).
- Example use-case: you send a metric-path, such as CPU %, very frequently (ex, once per second). You want to know the range of values that were sent, but you don't want to store every individual data point.

3.3 StatsD metrics

3.3.1 Overview

StatsD is a popular protocol/program for sending/aggregating data. While StatsD has a time-series component to it, the protocol does not send timestamps. StatsD is oriented toward sending business metrics & leaving it up to the metric receiver (the official StatsD app, StatsAgg, etc) to make sense of them. By default, StatsAgg will listen for StatsD metrics on ports 8125 (TCP & UDP, on both ports).

StatsD works as follows:

1) StatsAgg receives StatsD metrics.
2) Every 'flush interval' (10 seconds, by default), StatsAgg will aggregate the StatsD metrics.
    a. The values from identical metric-paths are aggregated together (the aggregation logic will depend on their metric type).
    b. All aggregations that involve division will round to 7 decimal places.
    c. The timestamps are averaged together.
3) The aggregated metrics are made available to StatsAgg's alerting & outputting routines.

A good-faith effort was made to accurately re-implement the StatsD protocol in StatsAgg. As of Spring 2015, StatsAgg's StatsD protocol implementation mirrors the behavior of every variation of every StatsD metric-type. However, there are implementation differences that users should be aware of. Please read the metric format section for more details.

Note – if you prefer to use the actual StatsD application along with StatsAgg, then that configuration should work fine. Just have StatsD output Graphite metrics to StatsAgg.

3.3.2 Metric format

The metric format closely mirrors the format of the official StatsD application.
The official StatsD documentation can be found here:
https://github.com/etsy/statsd/blob/master/docs/metric_types.md

General format: bucket:metricValue|metricType\n

Notes:

- StatsD calls the name of a metric a "bucket" (instead of 'metric-path', 'metric key', etc)
- The maximum size of a single metric (bucket + value + type) must be less than 32767 characters long.
- Best practice is to end metrics in a '\n' character.
    - This is required for sending metrics via TCP.
    - This is not required for sending individual UDP metrics, but newlines are required for sending multiple metrics in a single UDP post.
- There are minor implementation differences between how the official StatsD application works compared to StatsAgg. Please read notes for each metric type for details.

Below is a list of StatsD formats that StatsAgg supports.

3.3.2.1 Counters

Format: bucket:metricValue|c
Example:

```
myBucket:15.5|c
myBucket:3|c
myBucket:-9|c
```

… outputs (assuming a 'flush interval' of 10 seconds):

myBucket.count = 9.5
myBucket.rate = .95

This is a simple counter. The input values are added together at each 'flush interval', and then output. Two outputs are generated: a count (which is the sum of the aggregated values), and a per-second rate (count/'flush interval').

The counts & sums from different 'flush interval' windows are NOT aggregated together. Only metrics that arrive during the current 'flush interval' are aggregated together. If you want StatsAgg to aggregate metrics over multiple flush intervals, then you must use a 'gauge' metric.

count & rate reset to 0 after each aggregation.

Note:

- StatsAgg does support the StatsD 'legacy namespacing' convention, but it is not enabled by default. Please consult the application configuration file for more details.
- If no new metrics are received by StatsAgg for a bucket during a flush interval, StatsAgg can be configured to output either '0' or nothing.
    - The default behavior of StatsAgg is to send '0'.
    - After a restart, StatsAgg forgets bucket names for counter metrics. As a result, if StatsAgg is configured to output '0' for a counter-metric that has had a new value, StatsAgg won't begin outputting '0' again until this bucket receives at least one new data point.
    - After a restart, StatsAgg forgets bucket names for StatsD counter metrics. As a result, StatsAgg won't be capable of outputting '0's for a bucket until it receives at least one new data point for that bucket (assuming that StatsAgg is configured to output '0' for a bucket that hasn't recently had a new value).

3.3.2.2 Sampling-based counters

Format: bucket:metricValue|c|@samplePercent
Example:

myBucket:15.5|c|@0.1
myBucket:3|c|@0.5

… outputs (assuming a 'flush interval' of 10 seconds):
myBucket.sum = 161
myBucket.rate_ps = 16.1

Sampling based counters work the same way as regular counters, but the metric value is divided by the sampling percentage. In the example, the sampling percentage for the value of '15.5' is 10%. The sampling percentage for the value of '3' is 50%. The math works as follows: (15.5 / 0.1) + (3 / 0.5) = 161.

Sum & rate reset to 0 after each aggregation.

Note:

- Note: regular counter & sampling-based counters can safely be used together on the same bucket.
- StatsAgg does support the StatsD 'legacy namespacing' convention, but it is not enabled by default. Please consult the application configuration file for more details.
- If no new metrics are received by StatsAgg for a bucket during a flush interval, StatsAgg can be configured to output either '0' or nothing.
    - The default behavior of StatsAgg is to send '0'.

- After a restart, StatsAgg forgets bucket names for StatsD counter metrics. As a result, StatsAgg won't be capable of outputting '0's for a bucket until it receives at least one new data point for that bucket (assuming that StatsAgg is configured to output '0' for a bucket that hasn't recently had a new value).

3.3.2.3 Gauges

Format: bucket:metricValue|g
Example:

myBucket:+4|g
myBucket:15.5|g
myBucket:+0.5|g
myBucket:-3|g

… outputs "myBucket = 13". This is because …
0 (starting value) - 4 = -4
Set to 15.5 ('set' values always overwrite the previous value)
15.5 + 0.5 = 16
16 - 3 = 13

Gauges are arbitrary values. They can be static ('set') values or additive values. 'Additive' values are prefixed by +/- signs. 'Set' values do not include an operator ("+" or "-"). Anytime a 'set' gauge is encountered, it immediately overwrites whatever the previous value of the gauge was. 'Additive' values add or subtract from the previous gauge value.

Gauges can be very timing sensitive because a mixed series of 'set' & 'additive' gauges can result in major differences to the gauge value (depending on the order that the metrics are received). StatsAgg tags each incoming gauge metric at the exact moment it is received, so Gauges are processed in completely in sequence.

Unlike all other StatsD metric types, the value of a gauge metric persists after every aggregation. That means that you can add or subtract values to a gauge across an infinite period of time & never have the metric reset to 0 (unless you purposefully set it to 0).

Notes:

- A nuance of the Gauge metric type is that you can't 'set' a Gauge to a negative value without first setting the Gauge to 0. This is because the '-' sign is always interpreted as the operator for an 'additive' Gauge.
- If no new metrics are received by StatsAgg for a bucket during a flush interval, StatsAgg can be configured to output either "the gauge's value from the previous flush-interval's aggregation" or nothing.
  - If StatsAgg is configured to 'send previous values when no new value is received' and to 'persist statsd gauges', then gauges values are persisted in StatsAgg's database. That means that when StatsAgg is restarted, the last known gauge value is automatically loaded into StatsAgg. Also, additive values are executed against whatever the previous value of the gauge was. In this configuration, gauge values will never reset to 0.
  - If StatsAgg configured to 'send nothing when no new value is received', then gauge values are reset to 0 after each StatsD aggregation.
  - The default behavior of StatsAgg is to send the gauge's previous value & to persist gauges into the StatsAgg database.

3.3.2.4 Timers

Format: bucket:metricValue|ms
Example:

myBucket: 4|ms
myBucket:12|ms
myBucket:2|ms

… outputs (assuming a 'flush interval' of 10 seconds. For percentile values, assume the 70th percentile):
myBucket.count = 3
myBucket.count_70 = 2
myBucket.count_ps = 0.3
myBucket.lower = 2
myBucket.mean = 6
myBucket.mean_70 = 3
myBucket.median = 6
myBucket.std = 4.32
myBucket.sum = 18
myBucket.sum_70 = 6
myBucket.sum_squares = 164
myBucket.sum_squares_70 = 20
myBucket.upper = 12
myBucket.upper_70 = 4

Like counter, timers also support sampling. Metric sampling alters the timer 'count' and 'count_ps' metrics.
Example: myBucket:12|ms|@0.1
In the above example, the sample-rate is '0.1'. This results in 'count' and 'count_ps' being multiplied by 10.

Timer metrics are the most complex/heavy metric type that StatsD supports. The input is expected to be in the form of a time (in milliseconds), but it can be any arbitrary list of values. On an aggregation, all the metric values associated with a bucket are aggregated together & the following metrics are outputted:

- count (the count of 'how many metrics were in a bucket's list of metrics')
  - If a sample rate was used, then the formula is: count = count/sample_rate
- count_ps (the per-second rate of 'how many metrics were in a bucket's list of metrics')
  - count_ps = count/flush_interval
  - If a sample rate was used, then the formula is: count_ps = count/flush_interval/sample_rate
- count_nth (the count of 'how many metrics were in a within the bucket's nth percentile of metrics')
  - The 'nth' part of 'count_nth' is the numeric value of the nth percentile that is calculated. Example: 'count_40'
  - In the series "120, 334, 450, 496, 553, 675, 844, 994", the 40th percentile count is '3'.
- count_topnth (the count of 'how many metrics were in a within the bucket's top nth percentile of metrics')
  - The 'nth' part of 'top_topnth' is the numeric value of the top nth percentile that is calculated. Example: 'count_top40'
  - When a negative nth percentile is specified by the user, this makes StatsD present the 'top' nth percentile.
  - In the series "120, 334, 450, 496, 553, 675, 844, 994", the top 35th percentile (user specified '-35') count is '3'.
- lower (the smallest value of a bucket's list of metrics)
- lower_topnth (the smallest value, within the top nth percentile, of a bucket's metrics)
  - The 'nth' part of 'lower_nth' is the numeric value of the nth percentile that is calculated. Example: 'lower_top40'
  - When a negative nth percentile is specified by the user, this makes StatsD present the 'top' nth percentile.
  - In the series "120, 334, 450, 496, 553, 675, 844, 994", the top 35th percentile (user specified '-35') lower is '675'.
- mean (the average of a bucket's list of metrics)
- mean_nth (the average of a bucket's nth percentile of metrics)
  - The 'nth' part of 'mean_nth' is the numeric value of the nth percentile that is calculated. Example: 'mean_40'
  - In the series "120, 334, 450, 496, 553, 675, 844, 994", the 40th percentile mean is '301.3333333'.

- mean_topnth (the average of a bucket's top nth percentile of metrics)
    - The 'nth' part of 'mean_topnth' is the numeric value of the top nth percentile that is calculated. Example: 'mean_top40'
    - When a negative nth percentile is specified by the user, this makes StatsD present the 'top' nth percentile.
    - In the series "120, 334, 450, 496, 553, 675, 844, 994", the top 35th percentile (user specified '-35') mean is '837.6666667'.
- median (the median value of a bucket's list of metrics)
- std (the 'population standard deviation' of a bucket's list of metrics).
- sum (the sum of a bucket's list of metrics)
- sum_nth (the sum of a bucket's nth percentile of metrics)
    - The 'nth' part of 'sum_nth' is the numeric value of the nth percentile that is calculated. Example: 'sum_40'
    - In the series "120, 334, 450, 496, 553, 675, 844, 994", the 40th percentile sum is '904'.
- sum_topnth (the sum of a bucket's top nth percentile of metrics)
    - The 'nth' part of 'sum_topnth' is the numeric value of the top nth percentile that is calculated. Example: 'sum_top40'
    - When a negative percentile is specified by the user, this makes StatsD present the 'top' nth percentile.
    - In the series "120, 334, 450, 496, 553, 675, 844, 994", the top 35th percentile (user specified '-35') sum is '2513'.
- sum_squares (the 'sum of squares' of a bucket's list of metrics)
    - sum_squares = metricValueA^2 + metricValueB^2 + …
- sum_squares_nth (the 'sum of squares' of a bucket's nth percentile of metrics)
    - The 'nth' part of 'sum_squares_nth' is the numeric value of the nth percentile that is calculated. Example: 'sum_squares_40'
    - In the series "120, 334, 450, 496, 553, 675, 844, 994", the 40th percentile 'sum of squares' is '328456'.
- sum_squares_topnth (the 'sum of squares' of a bucket's top nth percentile of metrics)
    - The 'nth' part of 'sum_squares_topnth' is the numeric value of the top nth percentile that is calculated. Example: 'sum_squares_top40'
    - When a negative percentile is specified by the user, this makes StatsD present the 'top' nth percentile.
    - In the series "120, 334, 450, 496, 553, 675, 844, 994", the top 35th percentile (user specified '-35') 'sum of squares' is '2155997'.
- upper (the largest value of a bucket's list of metrics)
- upper_nth (the large value, within the nth percentile, of a bucket's metrics)
    - The 'nth' part of 'upper_nth' is the numeric value of the nth percentile that is calculated. Example: 'upper_40'
    - In the series "120, 334, 450, 496, 553, 675, 844, 994", the 40th percentile upper is '450'.

The values reset to 0 after each aggregation.

Timers also (optionally) support histograms. Histograms work as follows:

- On an aggregation cycle, metric data-points that share the same bucket can have their metric values categorized in a histogram. Histograms allow one to see the frequency distribution of timer metric values by counting how many metric values fall within a predefined set of ranges.
    - The ranges are referred to as 'bins', and the lower/upper ends of the bin range are the 'bin boundaries'.
    - The lower end of the first 'bin' is assumed to be 0.
- A counter for the bin is incremented by 1 if…
    - A metric's numeric value is greater than or equal to the lower-boundary of histogram 'bin value'
    - A metric's number value is less than the upper-boundary of the histogram 'bin value'
- One can specify 'inf' in the 'bin values' list as a catch-all for values that are not covered by any other bin.
    - 'inf' must be the last bin specified in the 'bin values' list. 'inf' stands for infinity.

- Multiple histogram configuration sets can be listed. They are evaluated in the order that they are specified. The histogram configuration that will be used is the first configuration where the 'metric' field has a value that is part of metric's bucket.
    - Example histogram configuration: [{metric:'taco',bins:[1,10,20]},{metric:'lalalala',bins:[5,'inf']}]
      For a bucket of "cpu.server.lalalala", the bins:[5,'inf'] would be used.
      That means that the histogram boundaries are: 0-5, 5-infinity
    - If one does not specify a 'metric' value in the histogram configuration, then all metrics are matched.
      Example: [{metric:'',bins:[1,10,20]}]

Example:

Histogram configuration: [{metric:'myBucket',bins:[0.1,1,1.21,'inf']}]

Metrics:
myBucket:0|ms
myBucket:0.1|ms
myBucket:1|ms
myBucket:1.1|ms
myBucket:1.2|ms
myBucket:10|ms
myBucket:10.1|ms
myBucket:15|ms

Output:
myBucket.histogram.bin_0_1 = 1
myBucket.histogram.bin_1 = 1
myBucket.histogram.bin_1_21 = 3
myBucket.histogram.bin_1_inf = 3

Notes:

- On histograms…
    - StatsAgg sanitizes histogram configurations more thoroughly than StatsD does. For a given histogram configuration, StatsAgg de-duplicates bin values, removes negative bin values, removes leading/trailing zeros on bin values, and sorts bin values.
    - Since timer metric values cannot be negative, one cannot have a negative 'bin value'.
    - Since a metric value must be greater than the 'bin value' for the bin counter to be incremented, and the lowest allowed metric value is '0', histogram bins must be greater than '0'.
    - For compatibly with Graphite, StatsD replaces all occurrences of '.' with '_' in the outputted metrics. StatsAgg also does this.
- The outputted metric-keys for the nth percentile keys will substitute the '_' for the '.' character in the event that the user specifies an nth percentile that has a fractional component.
    - Example: if the user-specified nth percentile is "70.5", then an example output metric-key would be "myBucket.upper_70_5"
- When a negative percentile is specified by the user, this makes StatsD present the 'top' nth percentile. This also has the side effect of changing the users negative percentage into a positive percentage, and will result in metric-keys between outputted with 'top' in place of '-'.
    - Example: if the user-specified top nth percentile is '-45, then an example output metric-key would be "myBucket.sum_top45".
- StatsAgg allows for multiple 'nth percentile' values to be computed & outputted. This can be configured in the StatsAgg application configuration file.
- If no new metrics are received by StatsAgg for a bucket during a flush interval, StatsAgg can be configured to output either '0' or nothing.
    - The default behavior of StatsAgg is to send '0'.

- After a restart, StatsAgg forgets bucket names for StatsD timer metrics. As a result, StatsAgg won't be capable of outputting '0's for a bucket until it receives at least one new data point for that bucket (assuming that StatsAgg is configured to output '0' for a bucket that hasn't recently had a new value).

3.3.2.5 Sets

Format: bucket:metricValue|s
Example:

```
myBucket:15.5|s
myBucket:15.5|s
myBucket:88|s
myBucket:2.6|s
myBucket:88|s

… outputs: "myBucket = 3"
```

Sets count the number of unique metric values that a bucket has associated with it at aggregation time. If the value of '1000' is sent 30 times during the flush interval, it will only count as +1 for the set.

The value resets to 0 after each aggregation.

Note:

- If no new metrics are received by StatsAgg for a bucket during a flush interval, StatsAgg can be configured to output either '0' or nothing.
  - The default behavior of StatsAgg is to send '0'.
  - After a restart, StatsAgg forgets bucket names for StatsD set metrics. As a result, StatsAgg won't be capable of outputting '0's for a bucket until it receives at least one new data point for that bucket (assuming that StatsAgg is configured to output '0' for a bucket that hasn't recently had a new value).

3.3.3 StatsD legacy namespacing

StatsD has a metric output naming convention called 'legacy namespacing'. 'Legacy namespacing' has the following effects:

- All metric output Sprefixes that are specific to StatsD counters are ignored
  - Counter 'rates' are put in stats.{myBucket}
  - Counter 'counts' are put in stats_counts.{myBucket}
- All other metric types are put in stats.{metricTypePrefix}.{myBucket}

StatsAgg supports this convention, but it is disabled by default.

3.3.4 Other differences between StatsD & StatsAgg

While the various metric formats have some differences compared to the official StatsD application (see the individual metric format sections for more details), there are a few other differences in StatsAgg (compared to StatsD) that are worth mentioning. They are:

- StatsAgg is not capable of being a StatsD 'repeater'
  - If you wish to accomplish this functionality, you could use the actual StatsD application as a repeater & have the final destination be StatsAgg. This will only work for StatsD UDP metrics.
- StatsAgg cannot disable counters ("flush_counts") like StatsD can.
- StatsAgg does not have a mechanism for user-provided 'backends'.
  - Because StatsAgg is a compiled/bundled program, 'backend' functionality must be built into the core program.

- Graphite is the only currently supported backend (for now). However, since several services support the receiving Graphite metrics (ex- influxdb), you are not necessarily limited to only Graphite.

3.3.5 Sample StatsAgg configuration file compared to a StatsD configuration file

StatsD

```
{
        server: "./servers/tcp",
        graphitePort: 2003,
        graphiteHost: "graphite.some-domain.com",
        port: 8125,
        percentThreshold: [40],
        histogram: [ { metric: '', bins: [ 0, 0.1, 1, 1.21, 'inf' ] } ],
        graphite: {globalPrefix : "stats", legacyNamespace : false}
}
```

StatsAgg

```
graphite_output_module_1 = true,graphite.some-domain.com,2003,3
statsd_tcp_listener_enabled = true
statsd_tcp_listener_port = 8125
statsd_metric_name_prefix_enabled = true
statsd_metric_name_prefix_value = stats
statsd_counter_metric_name_prefix_enabled = true
statsd_counter_metric_name_prefix_value = counters
statsd_timer_metric_name_prefix_enabled = true
statsd_timer_metric_name_prefix_value = timers
statsd_gauge_metric_name_prefix_enabled = true
statsd_gauge_metric_name_prefix_value = gauges
statsd_set_metric_name_prefix_enabled = true
statsd_set_metric_name_prefix_value = sets
statsd_use_legacy_name_spacing = false
statsd_nth_percentiles = 40
statsd_histograms = [ { metric: '', bins: [ 0, 0.1, 1, 1.21, 'inf' ] } ]
```

3.4 OpenTSDB

OpenTSDB (open time-series database) is primarily a metrics storage database. It is highly scalable & has advanced query & aggregation mechanisms. StatsAgg is tested for compatibility with OpenTSDB 2.0+.

Notes:

- Since OpenTSDB is very capable of aggregating metrics on its own, StatsAgg only acts as a ‘pass-through’ for OpenTSDB metrics. If you’re not planning on using StatsAgg for alerting on OpenTSDB metrics, then you should just send the metrics directly to OpenTSDB.
- By default, StatsAgg will listen for OpenTSDB telnet metrics on ports TCP port 4242.
- By default, StatsAgg will listen for OpenTSDB HTTP (JSON format) metrics on TCP port 4243.
- The HTTP endpoint for OpenTSDB metrics in StatsAgg is:
    - (domain:port)/api/put
    - Example: myServer.com:4243/api/put
- OpenTSDB’s format is extremely similar to Graphite’s format. The only meaningful difference is that every metric must be ‘tagged’.
- Other than "put" requests, the **only** other OpenTSDB API call that StatsAgg responds to is the "version" API request.

- This was done solely because the official OpenTSDB metric collector, tcollector, uses "version" as a health-check for OpenTSDB.
- This is only implemented for the telnet interface

3.4.1 Metric Format

Telnet interface
The OpenTSDB format is: metric epochTime metricValue tagKey1=tagValue1 tagKey2=tagValue2 \n
Example: some.descriptive.name.for.a.metric 123.643 1408334418 type=taco food=Mexican\n

When using telnet, you must use the telnet "put" command to send the metric.
Example: put some.descriptive.name.for.a.metric 1408334418 123.643 type=taco food=Mexican

More details can be found here:
http://opentsdb.net/docs/build/html/user_guide/writing.html
http://opentsdb.net/docs/build/html/api_telnet/put.html

HTTP interface
To send a single metric, send an HTTP POST with a body that has content formatted like this:

```
{
  "metric": "some.descriptive.name.for.a.metric",
  "timestamp": 1408334418,
  "value": 18.888,
  "tags": {
    "food": "Mexican",
    "type": "taco"
  }
}
```

To send multiple metrics in a HTTP POST, format the HTTP body as a Json Array, like this:

```
[
{
  "metric": "some.descriptive.name.for.a.metric.1",
  "timestamp": 1408334418,
  "value": 16.888,
  "tags": {
    "food": "CatFood",
    "type": "meowmix"
  }
},
{
  "metric": "some.descriptive.name.for.a.metric.2",
  "timestamp": 1408334418,
  "value": 19.888,
  "tags": {
    "food": "American",
    "type": "hotdog"
  }
}
]
```

More details can be found here: http://opentsdb.net/docs/build/html/api_http/put.html
Details on sending compressed payloads can be found here: http://opentsdb.net/docs/build/html/api_http/index.html

Notes:

- The implementation of the telnet input interface is more efficient than the implementation of the HTTP/JSON input interface. If performance is a concern, then use the telnet interface.
- Compared with the official OpenTSDB program, StatsAgg can be configured to be more secure than OpenTSDB.
    - OpenTSDB runs its user-interface, http metric listener, telnet metric listener, and http APIs off the same port. Since everything runs off the same port, it is somewhat difficult to control access via traditional control mechanisms (ex – firewall).
    - StatsAgg runs the telnet listener, http listener, and user-interface on separate ports. This allows one to open up the ports that are used for receiving incoming metrics to a broader audience, while restricting the ability to read those metrics to a narrower audience.
    - StatsAgg can also act as a sort of 'front door' for OpenTSDB. That is to say, StatsAgg could receive metrics on a public network and then forward them to OpenTSDB (which could be on a private network).
- The OpenTSDB HTTP interface has full support for the 'summary' parameter, and partial support for the 'details' parameter. See the OpenTSDB documentation for more information.
    - The 'details' parameter will return accurate counts for 'failed' & 'success', but the 'errors' field is not currently being populated.
- StatsAgg also supports compression on the OpenTSB HTTP interface. As long as the "Content-Encoding" field is set to "gzip" in the HTTP request header, users can send a gzip-compressed HTTP request body (the OpenTSDB metric JSON) to StatsAgg.
- When using the telnet interface, every individual metric MUST end with a newline character.
- The timestamp field can either be 10 characters long with second precision, or 13 characters long with millisecond precision.
- OpenTSDB tags cannot have duplicate tag-keys on the same metric.
    - Valid: some.descriptive.name.for.a.metric 123.643 1408334418 **type**=taco **food**=Mexican\n
    - Invalid: some.descriptive.name.for.a.metric 123.643 1408334418 **type**=taco **type**=Mexican\n
- The maximum size of a single metric (metric + timestamp + value + tags) must be less than 32767 characters long. This is larger than OpenTSDB supports.
- To maintain compatibility with outputting to Graphite, it may make sense to follow Graphite's metric-path conventions.
- OpenTSDB supports a limited character set for metric names, tag keys, and tag values. The allowed characters include alphanumeric characters, '.', '_', '-', and '/'. StatsAgg does not care about these character-set limitations, but you may receive errors when writing metrics with forbidden characters to OpenTSDB.
- OpenTSDB supports scientific notation for metric values.
- There can be multiple key=value tag pairs. StatsAgg does not limit the number of tags, but when outputting to OpenTSDB, you must have least 1 tag, and at most 8 tags.

3.4.2 How StatsAgg represents OpenTSDB metrics in the WebUI

To allow StatsAgg to match & alert off of any combination of OpenTSDB's metric/tags, StatsAgg generates what it calls a 'metric key'.

The 'metric key' format is:
metric : alphabetically_sorted_tags

So if the OpenTSDB metric is "cpu.core0", and the tags are "percentUsed=95 host=server1", then the StatsAgg 'metric key' would be:
cpu.core0 : host=server1 percentUsed=95

This allows StatsAgg users to write regular expressions for 'metric groups' that can accommodate any combination of metric & tag values.

Note – The concept of a 'metric key' stays internal to StatsAgg. This field is not outputted anywhere (other than the StatsAgg WebUI).

3.5 InfluxDB

NOTE: The version of InfluxDB that StatsAgg is compatible with is depreciated. Support for modern versions of InfluxDB will be included in a future StatsAgg release.

InfluxDB is primarily a metrics storage database. It is highly scalable & has advanced query capabilities. StatsAgg is tested for compatibility with InfluxDB 0.6x - 0.8x.

Notes:

- StatsAgg only acts as a 'pass-through' for InfluxDB metrics. If you're not planning on using StatsAgg for alerting on InfluxDB metrics, then you should just send the metrics directly to InfluxDB.
- By default, StatsAgg will listen for InfluxDB HTTP metrics on ports TCP port 8086.
- The HTTP endpoint for InfluxDB metrics in StatsAgg is:
    - (domain:port)/db/(your_influxdb_database)/series
    - Example: myServer.com:8086/db/statsagg/series?u=saUser&p=saPW
- More details can be found here: https://influxdb.com/docs/v0.8/api/reading_and_writing_data.html

3.5.1 Metric Format

A HTTP POST with a body that has content formatted like this:

URL: myServer.com:8086/db/statsagg/series?u=saUser&p=saPW&time_precision=ms

HTTP Body:

```
[
 {
  "name" : "hd_used",
  "columns" : ["metric_1", "metric_2", "host", "mount", "is_full", "time", "sequence_number"],
  "points" : [
   [23.2, 33.44, "serverA", "/mnt", false, 1400425947368, 2]
  ]
 }
]
```

More details can be found here: https://influxdb.com/docs/v0.8/api/reading_and_writing_data.html

Notes:

- The "time" column is optional. If specified, StatsAgg will use the time that is specified for the metric. If time is not specified, then StatsAgg will use the current time.
    - If time is specified, but the "time_precision" url query parameter is not, then StatsAgg will treat "time" as being in milliseconds.
    - "time" fields cannot be used for alerting. The field serves to specify the timestamp for the other fields in a 'point'.
    - The "time" column can be in seconds (10 digits), milliseconds (13 digits), or microseconds (16 digits). Timestamps must follow unix-time (epoch time) conventions.

        - "time_precision" should be submitted when using a timestamp precision that isn't millisecond. "time_precision" is a url query parameter, and must be set to "s" (for seconds), "ms" (for milliseconds), or "u" (for microseconds).
        - Example: myServer.com:8086/db/statsagg/series?u=saUser&p=saPW&time_precision=u
- The "sequence_number" column is ignored by StatsAgg (other than preserving it for outputting).
- If the "Authorization" field is present in the HTTP header, then StatsAgg will preserve that information for outputting to an InfluxDB server.
- Boolean 'point' fields are treated as numeric (so StatsAgg can alert on them). 'true' will be turned into a 1, and 'false will be turned into a 0.
- Any 'point' that has non-numeric field, then that field is treated by StatsAgg as a 'tag'. It will show up in the metric-key of a metric, but it will not be a separate metric in StatsAgg.
    - Exception – if a 'point' contains exclusively non-numeric fields, then it is treated as a unique metric. The value of such a metric is always '1'.
- If a 'point' has more than one numeric field, then each numeric field is treated as being a separate metric in StatsAgg. These separate metrics will share the same tags, but will be listed in the StatsAgg interface as separate metrics.
    - If these metrics are outputted to an InfluxDB database, they are outputted together (as they were originally received).
- When metric prefixes are used (set via the StatsAgg application configuration file), the metric prefix is appended onto the metric name.
    - Example: An InfluxDB prefix of "myPrefix" would result in a metric name like "myPrefix.metric_1".

3.5.2 How StatsAgg represents InfluxDB metrics in the WebUI

To allow StatsAgg to match & alert off of any combination of InfluxDB's metric/tags, StatsAgg generates what it calls a 'metric key'. StatsAgg will convert each numeric field in a 'point' into a unique metric-key.

The 'metric key' format is:
DatabaseName : MetricName : MetricColumn : sorted-tag(s)

Using the example from the previous section, InfluxDB metric-keys look like this:
statsagg : hd_used : metric_1 : "host"="serverA" "mount"="/mnt"
statsagg : hd_used : metric_2 : "host"="serverA" "mount"="/mnt"
statsagg : hd_used : is_full : "host"="serverA" "mount"="/mnt"

This allows StatsAgg users to write regular expressions for 'metric groups' that can accommodate any combination of database/name/column/tag(s).

Note – The concept of a 'metric key' stays internal to StatsAgg. This field is not outputted anywhere (other than the StatsAgg WebUI).

3.6 Outputs Modules

3.6.1 Graphite

StatsAgg can (optionally) output metrics to Graphite. StatsAgg will transform any metric type that isn't natively in the Graphite format (ex- StatsD, OpenTSDB, InfluxDB) into Graphite metrics (keeping in mind the limitations of Graphite's format).

Notes:

- The only metrics that are outputted are 'aggregated' metrics.
    - In the case of Graphite-Passthrough, OpenTSDB, and InfluxDB metrics, this simply means that all metrics are sent to Graphite (though their metric-key may have been slightly altered by StatsAgg).

    - In the case of 'aggregated' metrics (StatsD, Graphite-Aggregated), only the final results of the aggregation are sent to Graphite.
- After X connection attempt retries (by default, X=2), StatsAgg will give up on trying to connect to Graphite. All metrics that were queued to be sent to Graphite are permanently discarded.
- After X retries (by default, X=2), StatsAgg will give up on trying to send a 'batch' metrics to Graphite. Any metrics that haven't been outputted after the configured number of retries are permanently discarded.
    - A 'batch' means the aggregation window's worth of StatsD metrics, Graphite-Passthrough metrics, Graphite-Aggregation metrics, OpenTSDB, or InfluxDB metrics. StatsAgg handles these different metric types on separate threads, so for a given aggregation window, the 'batches' for StatsD metrics, Graphite-Passthrough metrics, Graphite-Aggregation metrics, OpenTSDB, and InfluxDB metrics are independent of each other.
- If the connection between StatsAgg & Graphite is slow, this could cause stability problems in StatsAgg. The potential risk occurs when StatsAgg is receiving/processing/outputting metrics at a faster rate than they can be sent to Graphite. Slowness on metric transmission to Graphite could result in metrics building up in StatsAgg's memory, which could ultimately cause StatsAgg to experience degraded performance (and possibly crash).
    - Oftentimes the network isn't the problem, and Graphite's ability to receive/consume/process metrics is. If you have ample bandwidth between StatsAgg & Graphite, but Graphite doesn't seem to be keeping up, look into optimizing your Graphite deployment.
- When outputting OpenTSDB or InfluxDB metrics to Graphite, tags are not outputted/preserved in any way.
- When outputting to Graphite, StatsAgg can (optionally) remove forbidden characters out of your metrics. The StatsAgg development team recommends using this option as a safety mechanism to ensure that your metrics reach Graphite & are capable of being queried.
- When outputting InfluxDB metrics, StatsAgg will default to outputting metric-paths in this format:
(influxdb_databaseName).(influxdb_metricName).(influxdb_columnName)
Outputting the InfluxDB database name can be disabled via a an application configuration option.

3.6.2 OpenTSDB

StatsAgg can (optionally) output metrics to OpenTSDB. StatsAgg will transform any metric type that isn't natively in the OpenTSDB format (ex- StatsD, Graphite, InfluxDB) into OpenTSDB metrics (keeping in mind the limitations of OpenTSDB's format).

Notes:

- StatsAgg can output in via the OpenTSDB telnet interface or the OpenTSDB HTTP interface.
    - The OpenTSDB HTTP interface is supported by OpenTSDB 2.0+.
    - Generally speaking, the OpenTSDB telnet interface will perform better & is less prone to errors.
- StatsAgg will not make API calls to OpenTSDB in to populate UIDs for metrics. This means that OpenTSDB will have to be 'primed' by telling OpenTSDB all the metrics that will be sent to it ahead of time. Alternatively, you can set the OpenTSDB variable 'tsd.core.auto_create_metrics' to 'True' (which is not the default).
    - The StatsAgg development team recommends setting the OpenTSDB variable 'tsd.core.auto_create_metrics' to 'True'.
    - See the OpenTSDB documentation for more information.
- By default, OpenTSDB's HTTP interface cannot handle chunked HTTP requests. StatsAgg's OpenTSDB HTTP output module does not have a configuration option for HTTP chunking, so HTTP requests sent from StatsAgg to OpenTSDB could be chunked.
    - The StatsAgg development team recommends setting the OpenTSDB variable 'tsd.http.request.enable_chunked' to 'True'.
    - The StatsAgg development team recommends setting the OpenTSDB variable 'tsd.http.request.max_chunk' to '32768' (or larger).
- The only metrics that are outputted are 'aggregated' metrics.
    - In the case of Graphite-Passthrough, OpenTSD, and InfluxDB metrics, this simply means that all metrics are sent to OpenTSDB (though their metric-keys may have been prefixed by StatsAgg).

    - In the case of 'aggregated' metrics (StatsD, Graphite-Aggregated), only the final results of the aggregation are sent to OpenTSDB.
- After X connection attempt retries (by default, X=2), StatsAgg will give up on trying to connect to OpenTSDB. All metrics that were queued to be sent to OpenTSDB are permanently discarded.
- Each metric in a 'batch' is sent to OpenTSDB independently. Metrics that fail to send will be resent up to X times (by default, X=2). If a metric fails to send after that, it is permanently discarded.
    - A 'batch' means the aggregation window's worth of StatsD metrics, Graphite-Passthrough metrics, Graphite-Aggregation metrics, OpenTSDB metrics, or InfluxDB metrics. StatsAgg handles these different metric types on separate threads, so for a given aggregation window, the 'batches' for StatsD metrics, Graphite-Passthrough metrics, Graphite-Aggregation metrics, OpenTSDB, and InfluxDB metrics are independent of each other.
- If the connection between StatsAgg & OpenTSDB is slow, this could cause stability problems in StatsAgg. The potential risk occurs when StatsAgg is receiving/processing/outputting metrics at a faster rate than they can be sent to OpenTSDB. Slowness on metric transmission to OpenTSDB could result in metrics building up in StatsAgg's memory, which could ultimately cause StatsAgg to experience degraded performance (and possibly crash).
    - Oftentimes the network isn't the problem, and OpenTSDB's ability to receive/consume/process metrics is. If you have ample bandwidth between StatsAgg & OpenTSDB, but OpenTSDB doesn't seem to be keeping up, look into optimizing your OpenTSDB deployment.
- The OpenTSDB spec mandates that all metrics have at least one key=value tag. Graphite metrics & StatsD metrics do not have a concept of 'tag', so StatsAgg adds a tag for them.
    - For Graphite metrics, StatsAgg adds a tag of 'Format=Graphite'
    - For StatsD metrics, StatsAgg adds a tag of 'Format=StatsD'
    - For InfluxDB metrics, when no tags exist for a metric, StatsAgg adds a tag of 'Format=InfluxDB'. If InfluxDB tag fields are present, then they are outputted to OpenTSDB.
- StatsAgg sorts OpenTSDB tags alphabetically as it processes/stores them. As a result, when StatsAgg outputs a metric to OpenTSDB, the tags will be ordered alphabetically. In other words, the tag order may change. OpenTSDB should not care about this.
- When outputting to OpenTSDB, StatsAgg can (optionally) remove forbidden characters out of your metrics. The StatsAgg development team recommends using this option as a safety mechanism to ensure that your metrics reach OpenTSDB.
- When outputting InfluxDB metrics, StatsAgg will default to outputting metric name in this format: (influxdb_databaseName).(influxdb_metricName).(influxdb_columnName)
  Outputting the InfluxDB database name can be disabled via a an application configuration option.

3.6.3 InfluxDB 0.6x – 0.8x

StatsAgg can (optionally) output metrics to InfluxDB (versions 0.6x – 0.8x). StatsAgg will transform any metric type that isn't natively in the InfluxDB 0.6x-0.8x format (ex- StatsD, Graphite, OpenTSDB) into InfluxDB metrics (keeping in mind the limitations of InfluxDB's format).

Notes:

- Metrics that were sent in the InfluxDB format will retain 100% of their format as they are passed through to InfluxDB. Details like time_precision, database, username, password, etc will be preserved.
- Metrics that were sent in non-InfluxDB formats (Graphite, OpenTSDB, StatsD, etc) are converted into InfluxDB format as closely as possible.
    - Timestamps from non-InfluxDB formats will always be sent to InfluxDB in with a time-unit of milliseconds.
    - The StatsAgg application configuration file specifies fields for 'default InfluxDB database, username, and password'. Non-InfluxDB formats will output to the specified 'default' InfluxDB database, with the specified 'default' username & password. Native InfluxDB metrics will use whatever database/username/password combination that the metrics were originally sent with.

- Users must take care to create appropriate 'databases' inside of InfluxDB before sending metrics to InfluxDB. StatsAgg will not create InfluxDB databases.
- The only metrics that are outputted are 'aggregated' metrics.
    - In the case of Graphite-Passthrough, OpenTSD, and InfluxDB metrics, this simply means that all metrics are sent to InfluxDB (though their metric-keys may have been prefixed by StatsAgg).
    - In the case of 'aggregated' metrics (StatsD, Graphite-Aggregated), only the final results of the aggregation are sent to InfluxDB.
- After X connection attempt retries (by default, X=2), StatsAgg will give up on trying to connect to InfluxDB. All metrics that were queued to be sent to InfluxDB are permanently discarded.
- Each metric in a 'batch' is sent to InfluxDB independently. Metrics that fail to send will be resent up to X times (by default, X=2). If a metric fails to send after that, it is permanently discarded.
    - A 'batch' means the aggregation window's worth of StatsD metrics, Graphite-Passthrough metrics, Graphite-Aggregation metrics, OpenTSDB metrics, or InfluxDB metrics. StatsAgg handles these different metric types on separate threads, so for a given aggregation window, the 'batches' for StatsD metrics, Graphite-Passthrough metrics, Graphite-Aggregation metrics, OpenTSDB, and InfluxDB metrics are independent of each other.
- If the connection between StatsAgg & InfluxDB is slow, this could cause stability problems in StatsAgg. The potential risk occurs when StatsAgg is receiving/processing/outputting metrics at a faster rate than they can be sent to InfluxDB. Slowness on metric transmission to InfluxDB could result in metrics building up in StatsAgg's memory, which could ultimately cause StatsAgg to experience degraded performance (and possibly crash).
    - Oftentimes the network isn't the problem, and InfluxDB's ability to receive/consume/process metrics is. If you have ample bandwidth between StatsAgg & InfluxDB, but InfluxDB doesn't seem to be keeping up, look into optimizing your InfluxDB deployment.

4 Alerting

One of the key reasons StatsAgg was created was to provide a robust alerting mechanism for the various metric formats.

4.1 Alerting overview

StatsAgg offers two kinds of alerts – 'threshold' and 'availability'. Alerts can be configured in the StatsAgg via the website interface.

'Availability alerts' are triggered when a 'metric-key' (a unique metric that is associated with the metric group) stops reporting data points. Put another way, if StatsAgg stops receiving metric values for a particular metric-key, then an alert can be triggered. Receipt of a new data point for a 'metric-key' is the 'positive alert' criteria for the availability alerts.

'Threshold alerts' are triggered when the value of an individual metric-key is above/below/equal-to a specified 'threshold' value. The values of a metric-key that fall within the 'window duration' are used for alerting consideration.

Alerts are evaluated on an interval that is specified in the StatsAgg configuration file. By default, the interval is every 5 seconds. When an alert is becomes active, it displays its status through the website interface, and can (optionally) send an email notification. Additional details about the alerts can be found here.

4.2 How does alerting work?

The alert routine is independent from all of the metric aggregation routines (StatsD, Graphite-Passthrough, Graphite-Aggregated, OpenTSDB, InfluxDB). It runs periodically (the period can be changed in the application configuration file). When it executes, it will typically go through the following steps:

1. Metric association. This is the process of mapping metric-keys (Graphite metric-paths, StatsD buckets, OpenTSDB metrics/tags, etc) to StatsAgg 'Metric Groups'.

    a. This process can be CPU intensive if StatsAgg receives a large burst of metric-keys that it hasn't seen before.
    b. Only new/fresh metrics go through metric association. Metrics that haven't been seen in a long time, or only exist in the metric storage engine (Graphite, OpenTSDB, etc) will not be associated with 'metric groups' until a new data point arrives.
2. Determine what alerts are "suspended". This maps StatsAgg 'Suspensions' to StatsAgg 'Alerts'. If an alert is suspended, StatsAgg will not perform some (or all) of the remaining routines on a particular alert.
3. Determine which StatsAgg 'alerts' are in a triggered state. Caution & Danger alert statuses are evaluated at this stage. Alerts that are in a triggered state are marked as such.
    a. Alerts are triggered ONLY when the criteria for triggering them is met. In the absence of any data that indicates that an alert should be in a triggered state, the alert will be in a not-triggered state.
    b. In other words, alerts trigger on the premise of 'innocent until proven guilty'.
        i. For threshold alerts, absent of evidence of guilt (datapoints that match the alert criteria) counts as proof of innocence.
        ii. For availability alerts, absence of ANY evidence (datapoints) can prove guilt (assuming the criterion is met).
4. Send notifications via email. When appropriate (based on an alert's configuration), emails are sent out to the recipients that are specified in the alert's 'Notification Group'.
    a. Caution & Danger notifications are treated as being completely independent from each other.
    b. If an alert is 'acknowledged', it will not be 'resent' (even if 'resend after…' is set).
5. Output alert status metrics. StatsAgg can be configured to output the status of all alerts to Graphite.
   Note: when an alert's name has characters that don't translate well into Graphite's naming convention, StatsAgg may try to reformat the alert-name. Also, the 'metric-path' in Graphite will have a unique identifier at the end of it. This was done to guarantee that no two alerts are capable of outputting to the same metric-path in Graphite (which could otherwise happen as a result of StatsAgg's metric-path renaming).
   Here is the alert-status key:
    a. 0 = alert not triggered
    b. 1 = caution triggered
    c. 2 = danger triggered
    d. 3 = caution & danger triggered
6. Update global variables. After everything else has finished, the results of the alert routine are made available to the rest of the application. 'Metric group associations', 'alert triggered statuses', etc are now capable of being viewed in the WebUI.
    a. A larger 'alert routine execution frequency' will result in increased WebUI lag for 'Alerts', 'Metric Groups', etc.

4.3 What happens to triggered alerts when StatsAgg is restarted?

When StatsAgg is restarted, StatsAgg will handle alerts as follows:

- Threshold alerts that were in a triggered state before the restart will go into a status of 'pending'. This gives them time to evaluate their status before erroneously firing 'positive alert' emails. A threshold alert will get out of 'pending' status when one of the following criteria is met:
    - StatsAgg has received enough new data points to determine that the alert is still in a 'triggered' status. No email notifications will be sent in this case.
        - This is only evaluated for 'Threshold' alerts.

    - StatsAgg has been running for at least 'window duration' seconds, and the alert has not re-entered a 'triggered' status. The alert will no longer be triggered & a 'positive alert' notification will be send (if configured).
    - StatsAgg's configuration file has a 'max pending' time. If this timeout is reached & the alert is not in a 'triggered' condition, then the alert will no longer be triggered & a 'positive alert' notification will be send (if configured).
- Availability alerts pick-up exactly where they left off. There is no 'pending' status for an availability alert.
- All metric-keys must be re-associated with all 'metric groups'. This is done as new metric-keys arrive into StatsAgg. This can be a slow/heavy process if a lot of unique metrics-keys are sent to StatsAgg, so it may take a little while (seconds to minutes) for all metric-keys to become associated with all 'metric groups'.
- StatsAgg does not currently store/retain the datapoints that were in memory prior to the restart, nor does it retrieve them from any external datasource (ie - Graphite). After a restart, the only datapoints that are considered for alerting are whatever new datapoints arrive.

4.4 Notes

- The alert routine runs in a separate thread from the other 'aggregation' routine threads. In other words, alert evaluation is NOT performed as part of the StatsD, Graphite, and OpenTSDB metric aggregation routines.
- **StatsAgg does not query metric-storage engines for data points.** Put another way, StatsAgg can only alert on what flows through it.
- **StatsAgg aggressively removes unneeded data points from memory.** If a metric data point is not needed by an alert, then it is removed from memory within a few seconds. Furthermore, StatsAgg will not try to load data points into memory from metric storage engines – even if an alert and/or 'metric group' is created that says that it needs the last X seconds of data for that metric. The alert and/or 'metric group' will only react to data points that arrive *after* the alert and/or 'metric group' were created.
- **For StatsD & Graphite-Aggregated metrics, alert evaluation is only performed on *aggregated* metrics.**
    - Example: If you send 4000 StatsD gauge metrics during a single 'flush interval' (which is 10secs, by default), then StatsAgg will only use the final aggregated value of those 4000 gauges for alert evaluation.
    - Since Graphite-Passthrough & OpenTSDB metrics are not aggregated, every single individual data point is used for alert evaluation.

5 Administrative Website User Interface

5.1 Normal 'path' through the website

Once data is being fed into StatsAgg, you may want to alert on it. This section describes the general path through the website to create an alert.

1. Create a 'notification group'. 'Notification groups' are simply a list of email addresses that alerts get sent to. If you do not want/need StatsAgg to send email alerts, then you can ignore 'notification groups' entirely.
2. Create a 'metric group'. This 'metric group' should have regular expressions that match against whatever metric-keys you're interested in alerting on. If you which, you can also 'tag' your 'metric group'.
3. Create an alert. When you create an alert, it ties into a single 'metric group'. This allows you to alert on whatever values are associated with the 'metric group'. A single 'metric group' can be used by many alerts, but an alert can only be linked with a single 'metric group'.
4. Create a suspension. Suspensions allow you to temporarily disable alerting. If you don't need to suspend alerts, then you can ignore suspensions entirely.

The remains subsections go into greater detail about how to create & configure 'notification groups', 'metric groups',

alerts, and suspensions.

5.2 Common UI components

Tables

- All tables in the StatsAgg UI are generated server-side. On each page load, entire table is sent to the client, and Javascript is used for pagination, filtering, etc.
- If a table appears to be mangled, try clearing all StatsAgg related cookies in your browser.
- StatsAgg will remember table customizations for 30days via a browser cookie.
- For pages that contain tables, a 'search' can be applied to the table via url parameters. The url parameter is 'TableSearch'.
    - For example, if you wanted to display the 'Alerts' table, and you wanted to have the table displayed with a search phrase already in the 'search' bar, then you'd enter something like: http://127.0.0.1 /StatsAgg/Alerts?TableSearch=my+search+criteria

Sorting tables by column

On a table, any column, or set of columns, can be used for sorting. Sorting works by:

- For the first column that you want to sort by, click on the first column header that you want to sort by
- For the second, third, etc columns that you want to sort by, hold down shift & click on the second, third, etc column header(s) that you want to sort by

5.3 Metric Groups

'Metric Groups' are a mechanism to tie together individual metrics. Using regular expressions, you can group together any set of metrics. You can then create an alert that alerts off of the metrics that are associated with the metric group. A single 'metric group' can be associated with any number of alerts.

5.3.1 Metric Groups table

This table lists all current created 'notification groups'. Much of the functionality on this page speaks for itself, but some functionality requires some explanation.

Metric Associations

- Altering a 'metric group' will erase all 'metric group associations' with the 'metric group'. As StatsAgg receives new metrics, the 'metric group associations' for a 'metric group' will repopulate.
- Individual metrics are associated with 'metric groups' when the 'alert routine' is executed. That means that one should expect a delay between when the metric is initially received by StatsAgg & when it shows up as being associated with a metric group.
- Any metric that has had a new data point within the last 24hrs will show up as being 'associated' with a 'metric group'. Metrics that haven't received any new data points for 24hrs will be 'forgotten' by StatsAgg.

Clone

Clicking this button clones the 'metric group'. A new 'metric group' will be spawned with the same configuration. The only difference is:

- The new 'metric group' will be named "previousMetricGroupName_#" (where # is some number that doesn't collide with another 'metric group' name).

Remove

A 'metric group' can't be removed when it is associated with an alert. To make a 'metric group' be capable of being

removed, you must delete or alter all alerts that are associated with it.

5.3.2 Create Metric Group

Name (required): A unique name for this metric group. Other than uniqueness, the only limitation is that it must be under 500 characters long.

Description (optional): A description of the metric group. Avoid descriptions longer than 1000000 characters.

Regular Expressions (required): These are regular expressions that are used to tie individual metrics to the 'metric group'. If a metric matches any one of the provided regular expressions, then it is considered to be 'associated' with the 'metric group'. If using multiple regular expressions, put each regular expression on a separate line in the textbox. For performance reasons, try to use as few regexes as possible, and try to write them as efficiently as possible.

Blacklist Regular Expressions (optional): These are regular expressions that nullify metrics that were matched by the previous 'regular expressions' field. Any metric is matched by a 'blacklist regular expression' will not be associated with the 'metric group'. If using multiple 'blacklist regular expressions', put each blacklist regular expression on a separate line in the textbox. The purpose of this field is to give users an easy way to say "I want to match these metrics… except I don't want to match *these* metrics".

The regular expression format is Java Regex, and the 'matches' method is used.
For more information, visit…
https://docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html#matches--
https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html

Tags (optional): Metric groups can be 'tagged'. Tags allows for convenient filtering in the various tables on the StatsAgg web user-interface. Alerts can also be suspended by metric group tags.

Notes:

- Hitting 'Preview' for the 'regular expressions' field will show regular expression matches using all the metrics that StatsAgg is currently aware of. 'Blacklist regular expressions' are not taken into consideration when hitting this button.
- Hitting 'Preview' for the 'blacklist regular expressions' field will show the regular expression matches from the 'regular expressions' field, but filtering out any metrics that match a 'blacklist regular expression'. The results from this 'preview' button are reflective of what the final matches will be when the 'metric group' is created or altered.

5.4 Notification Groups

5.4.1 Notification Groups table

This table lists all current created 'notification groups'. Much of the functionality on this page speaks for itself, but some functionality requires some explanation.

Clone
Clicking this button clones the 'notification group'. A new 'notification group' will be spawned with the same configuration. The only difference is:

- The new 'notification group' will be named "previousNotificationGroupName_#" (where # is some number that doesn't collide with another 'notification group' name).

Test

Clicking on ‘test’ will send an email to all the members of the ‘notification group’. This functionality works regardless of whether the alert-routine is configured to send emails. The only requirement for this functionality to work is that the SMTP settings in the StatsAgg configuration file need to be valid. Emails will only be sent to email addresses that are properly formatted.

Remove
A ‘notification group’ can’t be removed when it is associated with an alert. To make a ‘notification group’ be capable of being removed, you must delete or alter all alerts that are associated with it.

5.4.2 Create Notification Group

Name (required): A unique name for this ‘notification group’. Other than uniqueness, the only limitation is that it must be under 500 characters long.

Email addresses (optional): A common separated list of email addresses.

5.5 Alerts

5.5.1 Alerts table

This table lists all current created alerts. Much of the functionality on this page speaks for itself, but some functionality requires some explanation.

Colored rows
Red: Danger alert is currently active & not suspended
Caution: Caution alert is currently active & not suspended
Blue: Either a caution or danger alert is currently suspended. Also, blue can indicate that StatsAgg was recently restarted & the alerting mechanism hasn’t determined whether a caution or danger alert that was active prior to the restart is still triggered after the restart.

Clone
Clicking this button clones the alert. A new alert will be spawned with the same configuration. The only differences are:

- The new alert will be named "previousAlertName_#" (where # is some number that doesn’t collide with another alert name).
- The state of the alert will not be cloned. That means that triggered/not-triggered, acknowledged/unacknowledged, last alert send time, etc are not cloned.

Remove
An alert can’t be removed when it is associated with a suspension. To make an alert be capable of being removed, you must delete or alter all suspensions that are associated with it.

Acknowledge/Unacknowledge
Alerts can be ‘acknowledged’ when they are in a ‘triggered’ state. This is done so that StatsAgg users can know that someone (hopefully the recipient of the notification group’s email notification) has seen that the alert is in a triggered state & is looking into the cause. When alerts are ‘acknowledged’, this disables resending of email notifications (even if resending is enabled).

5.5.2 Create Alert

Core Alert Criteria

Name (required): A unique name for this alert. Other than uniqueness, the only limitation is that it must be under 500 characters long.

Description (optional): A description of the alert. Avoid descriptions longer than 1000000 characters.

Metric group name (required): The 'metric group' that this alert is associated with. Alerts *must* be paired with 'metric groups', but multiple alerts can be associated with the same 'metric group'. Alerts are only capable of alerting on metric-keys that are associated with the alert's 'metric group'.

Alert type (required): Availability, Threshold
Availability: 'Availability alerts' are triggered when a 'metric-key' (a unique metric that is associated with the metric group) stops reporting data points. Put another way, if StatsAgg stops receiving metric values for a particular metric-key (no new data points for the last 'window duration' seconds), then an alert will be triggered. Receipt of a new data point for a 'metric-key' is the 'positive-alert' criteria for the availability alerts. Please note that the metrics generated by the 'resend metrics' options do not count as new data points (from the perspective of the availability alert logic).

Threshold: 'Threshold alerts' are triggered when the value of an individual metric-key is above/below/equal-to a specified 'threshold' value. The values of a metric-key that fall within the 'window duration' are used for alerting consideration.

Is alert enabled? (required): If you want the alert to be enabled right away after creation or alteration, then check this checkbox. When an alert is disabled, caution & danger alerts will not fire (they are not even evaluated).

Is caution alerting enabled? (required): When caution alerting is disabled, caution alerts will not trigger (they are not even evaluated).

Is danger alerting enabled? (required): When danger alerting is disabled, danger alerts will not trigger (they are not even evaluated).

Alert on positive? (required): If this is checked, alerts that change states from 'triggered' to 'not triggered' will send an email notification to the notification group's recipients.

Resend alert? (required): If this is checked, alerts that are in a 'triggered' state will send a new notification to the notification group's recipients after the specified amount of time. The new notification will reflect whatever metrics are currently resulting in the alert to be triggered (not just whatever was in the original email notification). Note— if an alert has been acknowledged, then notifications will not be resent (regardless of whether 'resend alert' is enabled).

Resend alert every (optional): If 'resend alert?' is enabled, then alerts in a 'triggered' state will send a new notification to the notification group's recipients after the amount of time specified by this option.

<u>Caution/Danger Criteria</u>

Notification group name (optional): The 'notification group' that is associated with the alert. Alerts that are triggered will be sent to the members of this 'notification group'.

Positive notification group name (optional): The 'positive notification group' that is associated with the alert. Positive alerts will be sent to the members of this 'notification group'.

Window duration (availability and threshold alerts): The 'window duration' is the amount of time (between 'now' and 'x time units ago') that metrics values are allowed to be considered for alerts. That is to say, if you specify that the 'window duration' is 300 seconds, then the rest of the alert criteria variables will only operate on metric values that have timestamps between 'now' and '300 seconds ago'. You should set the value of 'window duration' to be *at least* one time-unit larger than the metric transmission interval of the metrics associated with the alert (it is often best to oversize 'window duration' by 25% or so to give it some slack).

Example:

- If you send a new datapoint for a metric (ex - 'myServer.cpu') to StatsAgg once every 30 seconds, and you want to alert off this metric with a 'threshold alert', then you would set the 'window duration' to *at least* 31 seconds (but 45 seconds or 1 minute would usually be better). If you set the 'window duration' to something smaller than 30 seconds (ex- 29 seconds), then the alert could flap between 'triggered' and 'not-triggered' because the alert criteria would evaluating all the datapoints between 'now' and '29 seconds ago', and a datapoint from 30 seconds ago could fall outside of that window.

Notes:

- This field must be set to a value greater than 0.
- To preserve StatsAgg resources, it is highly recommended that you try to keep the window duration as small as possible.

Stop tracking after (availability alerts only): For availability alerts, StatsAgg requires that you eventually 'give up' on tracking a metric that hasn't had any new data points. This prevents StatsAgg from having to track metrics forever, and prevents availability alerts from potentially being in a 'triggered' state forever. 'Stop tracking after' is the amount of time that you will wait for a new data point to show up for a metric before the availability alert 'gives up' on looking for a new data point to show up. When an availability alert 'gives up', it is treated as a positive alert, and the reason given will be that the alert reached its 'stop tracking time limit'.

Minimum sample count (threshold alerts only): For a threshold-alert to be triggered, one of its associated metric-key must have a minimum number of recent data points within the 'alert window duration'. For example, if the 'alert window duration' is 300 seconds, and the 'minimum sample count' is 2, then an alert can only be triggered if 2 or more metric values (of an individual metric-key) have been received by StatsAgg during the last 300 seconds.

This field must be set to a value greater than 0. The most commonly set value is 1.

Operator (threshold alerts only): The values of a metric-key are considered for threshold-based alerting when they are above/below/equal-to a certain threshold. This value controls the above/below/equal-to aspect of the alert.

Combination (threshold alerts only): For a threshold-alert, we must decide *how* the values will be used for alert consideration. This variable asks: for the metric values of a single metric-key that fall within the 'alert window duration', what condition will cause the alert to be triggered? Is the *average* of the metric values above or below the threshold? Are *all* metrics values above or below the threshold? Is *any* metric value above or below the threshold? Are 'at least' or 'at most' X metric values above or below the threshold?

For example, say a metric-key has several values within the 'window duration', and those values are '1, 3, 5, 7'. Let's also say that we have a threshold of '6' & an operator of 'less than'. What happens – did we trigger the alert? Do we consider the average of those numbers? Do all of them have to be less than 6?

The 'combination' field makes the user choose *how* the metric values are evaluated. One can choose from 'any value', 'all values', the 'average of the values', 'at most X values', or 'at least X values'.

In the above example, if we choose 'average', then we'd be saying "*The average value* of the metric-key must be less than of 6". Since the average value is 4, the alert would be triggered.

Combination count (threshold alerts only): When one chooses a 'combination' of 'at most X values' or 'at least X values', a count is required. The 'combination count' refers to the number of independent metric values for a single metric-key that fall within the 'alert window duration'.

In the previous example of '1, 3, 5, 7' < 6, if we have a 'combination' of "at least" and a 'combination count' of 2, then we're effectively saying: "At least 2 metric-key values must be less than 6". Since both 1, 3, and 5 are all less than 6, the alert would be triggered.

This value must be greater than or equal to 0.

Threshold (threshold alerts only): The 'threshold' is the value that is compared against when deciding whether an alert is active or not.

For example, if an alert states that "At least 2 metric-key values must be less than the value of 6", then the 'threshold' in that alert is 6.

Notes:

- For an alert, one can set the 'caution criteria', the 'danger criteria', both, or neither. StatsAgg does not require that the caution or danger criteria be valid for an alert to be created. However, if you want the alert to *do* something, then you should set some valid alert criteria.
- A good way to test alert criteria for validity is to hit the 'preview' button. This will give you a preview of what the email alert would look like. The wording of the email alerts was very carefully chosen to give readers a 'plain english' version of the alert criteria.
- When an alert is triggered by one or more metrics, a single notification is sent to the receipts listed in the alert's 'notification group'. StatsAgg does not currently have the capability of alerting on each metric-key that tripped the alert separately.
- Each metric-key that is associated with the alert's 'metric group' is individually considered for triggering the alert. The data points from different metric-keys are never aggregated together. For example, say that we have 'combination' set to "average":

  MetricSeries1 values: 1 2 3 4 5 -- avg = 3
  MetricSeries2 values: 1 1 1 1 1 -- avg = 1

  If the alert's criterion says to alert when the average value is greater than 2, then 'MetricSeries1' will trigger the alert, but 'MetricSeries2' won't. In the notification email, only 'MetricSeries1' will be mentioned.

5.6 Suspensions

5.6.1 Suspensions table

This table lists all current created suspensions. Much of the functionality on this page speaks for itself, but some functionality requires some explanation.

Colored rows
Blue: The suspension is currently active.

5.6.2 Create Suspension

Options

Name (required): A unique name for this suspension. Other than uniqueness, the only limitation is that it must be under 500 characters long.

Description (optional): A description of the suspension. Avoid descriptions longer than 1000000 characters.

Is enabled? (required): If you want the suspension to be enabled after creation or alteration, then check this checkbox. When a suspension is disabled, the suspension won't be able to suspend any caution or danger alerts.

Suspend notification only? (required): When checked, suspended alerts still evaluate their alert criteria (and display the caution/danger triggered status on the StatsAgg WebUI); they just don't send out emails alerts. When unchecked, suspended alerts will not be evaluated at all. The most common approach is to have this field checked. Note – this option is not available for a 'metric suspension'.

Suspend by…

Alert name (optional): If you want to suspend a single alert, then suspend it by alert name.

Tags (optional): If you want to suspend a group of alerts that have one or more tags in common, then suspend by tags. For an alert to be suspended, it must be tagged with every tag specified in the suspension. Only put one tag per line in the 'tags' textbox.

Example: if the suspension suspends by *tags*, with tags: "tag1", "tag2", "tag3"…

- Alert_1 (tags: "tag1", "tag2", "tag3") – meets suspension criteria
- Alert_2 (tags: "tag1", "tag2") – does not meet suspension criteria
- Alert_3 (tags: "tag1", "tag2", "tag3", "tag4") – meets suspension criteria

Everything (optional): If you want to suspend ALL alerts, except for a select few, then select this option. For an alert to be spared from suspension, it must be tagged with at least one of the tags specified in the suspension. Only put one tag per line in the 'everything' textbox.

Example: if the suspension suspends by *everything*, with tags: "tag1", "tag2", "tag3"…

- Alert_1 (tags: "tag1", "tag2", "tag3") – meets suspension criteria
- Alert_2 (tags: "tag3") – meets suspension criteria
- Alert_3 (tags: "tag4") – does not meets suspension criteria

Metrics (optional): Unlike all other 'suspend by' fields, this 'metrics' does not suspend alerts. Instead, choosing 'metrics' will all one to suspend all metrics (that match the regexes specified) from being considered by ANY alert. This sort of suspension can be useful if you want to suspend a set of metrics that span multiple alerts, but aren't adequately covered by alert tags. Note that this only prevents metrics from alerting. This does not prevent metrics from entering StatsAgg, being associated with 'metric groups', and being outputted to the various datastores that StatsAgg supports.

Example:

- 'Suspend by': 'metrics' & enter a regex of *.*123.**
- Wait until the suspension is active…
- Alerts would not be allowed to consider metrics with *123* in the metric-key for alerting.

Suspension Type…

Recurring (daily): This suspension type means that the suspension recurs every day. The suspension will be active starting at whatever is specified in 'Start Time', and will expire after 'duration' minutes.

One Time: This suspension type means that the suspension is only ever used once. Once the suspension is complete, it will self-delete. The suspension will be active starting at whatever is specified in 'Start Date/Time', and will expire after 'duration' minutes.

Start Date: Recurring 'suspensions' are only active on or after the date specified in this field.
The format is: MM/dd/yyyy
Example: 03/23/2014

Recurs on ('recurring' suspensions only): Recurring suspensions are active only on the days specified by 'recurs on'. This can be used for cases such as "we have maintenance only on Wednesday nights, so we only want to suspend alerts on Wednesday night". In a case like this, you would put a check next to 'W' (and no other boxes).

Start Time: What time does the 'suspension' become active?
The format is: h:mm a.
Example: 9:23 AM

Duration: Specifies the amount of time that the suspension remains active. This must be a positive value. For recurring a suspension, duration must be less than or equal to 1440 minutes (one day).

Notes

- If a suspension becomes active on a particular day, and the specified duration takes it into the next day, then the suspension will continue to remain active (until 'duration' minutes have passed). This happens regardless of whether the 'recurs on' field allows suspensions on the 'next day' or not.
- The 'Preview Suspension Associations' button next to 'Suspend By…' can be used to determine if your suspension rule is suspending the alert(s) that you want to suspend.

5.7 Regex Tester

If you want to see what metrics StatsAgg is currently aware of (seen in the last 24 hours), then you can use free-form regular expressions to view the list of known metrics.

Note: regular expressions are matched against *aggregated* metrics. For example, if they are StatsD metrics, then you should expect to metrics with keys like *originalMetricName.sum*. Also, if metric prefixes and/or suffixes were used, then they will also be matched against.

The regular expression format is Java Regex, and the 'matches' method is used.

5.8 Metric Alert Associations

If you want to see what alerts that are associated with the current metrics that StatsAgg is aware of (seen in the last 24 hours), then you can use free-form regular expressions to match against the metrics, and any alerts that are associated with matched metrics will be displayed.

For example,

- Search for:
  .*cpu
- This may match several metrics:
  myServer1.cpu
  myServer2.cpu
  myServer3.cpu
  …
- If there is an alert / metric-group combination that is looking for 'myServer1.cpu', and that alert name is 'MyServer1 CPU % Alert', then you will see 'MyServer1 CPU % Alert' listed in the results of this page
- If there is an alert / metric-group combination that is looking for '.*cpu', and that alert name is 'All Servers CPU % Alert', then you will see 'All Servers CPU % Alert' listed in the results of this page

Note: regular expressions are matched against *aggregated* metrics. For example, if they are StatsD metrics, then you should expect to metrics with keys like *originalMetricName.sum*. Also, if metric prefixes and/or suffixes were used, then they will also be matched against.

5.9 Forget Metric(s)

'Forget Metric(s)' is a way of telling StatsAgg to completely forgot about a metric-key & all its values. Everything that there is to know about the metric key is wiped out.

Examples of when you would want to use this:

- You have StatsD metrics that are constantly sending their last seen value (like gauges), or are sending 0s (like timers, counters, etc). If you don't care about these metric-keys anymore, then 'forgetting' them will have StatsAgg stop sending values for them.
- You have an alert that is triggered based off of criteria that you know is no longer true. A quick way of clearing the alert-status could be to 'forget' the metric-key(s) that are triggering the alert.

On the 'forget metric(s)' page, you can either forget a single metric-key, or forget several (via a regex). If you enter values for both fields, then both fields will be evaluated.

Note: regular expressions are matched against *aggregated* metrics. For example, if they are StatsD metrics, then you should expect to metrics with keys like *originalMetricName.sum*. Also, if metric prefixes and/or suffixes were used, then they will also be matched against.

The regular expression format is Java Regex, and the 'matches' method is used.

5.10 Output Blacklist

'Output Blacklist' is a mechanism to prevent metrics from being outputted by StatsAgg.

For this to work, one must create a metric group. Once the 'metric group' is created, the 'metric group' can be associated with the 'output blacklist'. Once associated, any metric that is associated with the 'metric group' will be blacklisted from being outputted to all output modules (Graphite, OpenTSDB, InfluxDB, etc).

Example usage: bad/stupid tool is sending junk metrics. To avoid storing them permanently in in Graphite/OpenTSDB/etc, we use the 'output blacklist' in StatsAgg.

5.11 Metric Benchmark

The page is used to benchmark the regular expressions used by Metric Groups & Metric Suspensions.

StatsAgg's performance, especially after a reboot, can be affected by the 'heaviness' of the regular expressions found in metric groups & (some) suspensions. It's not always obvious how heavy a regular expression is, and finding which ones are the heaviest is even harder when StatsAgg is loaded with thousands of metric groups.

This page works by picking a random sampling of N metrics (where you input the value of N) that are known to StatsAgg. All metric group & suspension regular expressions are run against the subset of metrics & timed. After running all regexes against the subset of metrics, the heaviest regexs are returned (in descending order).

The accuracy of the benchmark is roughly proportional to the number of metrics that the regexes are tested against. As a starting point, a metric subset of 50000 metrics can usually provide a good baseline (without running for too long).

5.12 Alerts Report

The 'Alerts Report' page is a rollup of several other pages into a single view (Alerts, Metric Groups, Notification Groups). The spirit of it is to give those that want as many details about what goes into an alert as possible, which can be useful when reviewing alerts, auditing, etc. There is no information on this page that cannot be found elsewhere in the

application.

5.13 Health Check

StatsAgg exposes a health-check page @ /HealthCheck

Under normal circumstances, the health-check page should return some information about StatsAgg, and a status code of HTTP 200.

StatsAgg will return HTTP 500 on the health-check page when one of the following criteria is met:

- Failure on connecting to the database.
- Application initialization has failed. Many conditions can lead to this (failure to initialize the logging library, failure to start the StatsD and/or Graphite server listeners, etc).

6 The Json API

6.1 /api/alert-create

Url: /api/alert-create

Purpose: Create & modify alerts.

Type: HTTP POST

Json body field descriptions:

- id – The 'alert id'. Specify if you want to alter an existing alert (as opposed to creating a new alert). Type: Number
- name – The 'name' field @ here. Type: String
- description – The 'description' field @ here. Type: String
- metric_group_name – The 'metric group name' field @ here. Type: String
- alert_type – The 'alert type' field @ here. Type: String
- enabled – The 'is alert enabled?' field @ here. Can be 'true' or 'false'. Type: Boolean
- caution_enabled – The 'is caution alerting enabled?' field @ here. Can be 'true' or 'false'. Type: Boolean
- danger_enabled – The 'is danger alerting enabled?' field @ here. Can be 'true' or 'false'. Type: Boolean
- alert_on_positive – The 'alert on positive' field @ here. Can be 'true' or 'false'. Type: Boolean
- allow_resend_alert – The 'resend alert?' field @ here. Can be 'true' or 'false'. Type: Boolean
- resend_alert_every – The numeric field of the 'resend alert every' field @ here. Ex- for "resend an alert every 4 minutes", the value of this field would be 4. Type: Number
- resend_alert_every_time_unit – The time unit of the 'resend alert every' field @ here. Valid values: 'Seconds', 'Minutes', 'Hours', or 'Days'. Type: String
- caution_notification_group_name – The 'caution notification group name' field @ here. Type: String
- caution_positive_notification_group_name – The 'caution positive notification group name' field @ here. Type: String
- caution_minimum_sample_count – The 'caution minimum sample count' field @ here. Type: Number
- caution_operator – The 'caution operator' field @ here. Valid values: '<', '>', '<=', '>=', '='. Type: String
- caution_combination – The 'caution combination' field @ here. Valid values: 'Any', 'All', 'Average', 'At least', 'At most'. Type: String
- caution_combination_count – The 'caution combination count' field @ here. Type: Number
- caution_threshold – The 'caution threshold' field @ here. Type: Number
- caution_window_duration – The numeric field of the 'caution window duration' field @ here. Ex- for "a window duration of 4 minutes", the value of this field would be 4. Type: Number

- caution_window_duration_time_unit – The time unit of the 'caution window duration' field @ here. Valid values: 'Seconds', 'Minutes', 'Hours', or 'Days'. Type: String
- caution_stop_tracking_after – The numeric field of the 'caution stop tracking after' field @ here. Ex- for "stop tracking a metric after 4 minutes", the value of this field would be 4. Type: Number
- caution_stop_tracking_after_time_unit – The time unit of the 'caution stop tracking after' field @ here. Valid values: 'Seconds', 'Minutes', 'Hours', or 'Days'. Type: String
- danger_notification_group_name – The 'danger notification group name' field @ here. Type: String
- danger_positive_notification_group_name – The 'danger positive notification group name' field @ here. Type: String
- danger_minimum_sample_count – The 'danger minimum sample count' field @ here. Type: Number
- danger_operator – The 'danger operator' field @ here. Valid values: '<', '>', '<=', '>=', '='. Type: String
- danger_combination – The 'danger combination' field @ here. Valid values: 'Any', 'All', 'Average', 'At least', 'At most'. Type: String
- danger_combination_count – The 'danger combination count' field @ here. Type: Number
- danger_threshold – The 'danger threshold' field @ here. Type: Number
- danger_window_duration – The numeric field of the 'danger window duration' field @ here. Ex- for "a window duration of 4 minutes", the value of this field would be 4. Type: Number
- danger_window_duration_time_unit – The time unit of the 'danger window duration' field @ here. Valid values: 'Seconds', 'Minutes', 'Hours', or 'Days'. Type: String
- danger_stop_tracking_after – The numeric field of the 'danger stop tracking after' field @ here. Ex- for "stop tracking a metric after 4 minutes", the value of this field would be 4. Type: Number
- danger_stop_tracking_after_time_unit – The time unit of the 'danger stop tracking after' field @ here. Valid values: 'Seconds', 'Minutes', 'Hours', or 'Days'. Type: String

Notes:

- Modifying an alert will replace everything about the alert with the new fields for the alert. This applies to **ALL** fields; not just those specified. For example, with this API, you can't update the 'caution_threshold' field without posting all the other fields.
- If 'id' is specified, then that tells this API that an alert of the specified id will be modified with the specified values.
- 'metric_group_id' can be used in place of 'metric_group_name'. If both are specified, then 'metric_group_name' will be used.
- 'caution_notification_group_id' can be used in place of 'caution_notification_group_name'. If both are specified, then 'caution_notification_group_name' will be used.
- 'caution_positive_notification_group_id' can be used in place of 'caution_positive_notification_group_name'. If both are specified, then 'caution_positive_notification_group_name' will be used.
- 'danger_notification_group_id' can be used in place of 'danger_notification_group_name'. If both are specified, then 'danger_notification_group_name' will be used.
- 'danger_positive_notification_group_id' can be used in place of 'danger_positive_notification_group_name'. If both are specified, then 'danger_positive_notification_group_name' will be used.

Example request POST body:


```
{
  "id": 944,
  "name": "Firewall Disk Space",
  "description": "Send an alerts when firewall disk space is low. Send a caution alert when disk space is greater than 50%. Send a danger alert when disk space is greater than or equal to 90%.",
  "metric_group_name": "Firewall Disk Space",
  "alert_type": "Threshold",
  "enabled": true,
  "caution_enabled": true,
  "danger_enabled": true,
```

```
  "alert_on_positive": true,
  "allow_resend_alert": true,
  "resend_alert_every": 2,
  "resend_alert_every_time_unit": "Hours",
  "caution_notification_group_name": "Firewall Team Oncall",
  "caution_positive_notification_group_name": "Firewall Team",
  "caution_minimum_sample_count": 2,
  "caution_operator": ">",
  "caution_combination": "Any",
  "caution_threshold": 50,
  "caution_window_duration": 4,
  "caution_window_duration_time_unit": "Minutes",
  "danger_notification_group_name": "Firewall Team Oncall",
  "danger_positive_notification_group_name": "Firewall Team",
  "danger_minimum_sample_count": 4,
  "danger_operator": ">=",
  "danger_combination": "At Least",
  "danger_combination_count": "3",
  "danger_threshold": 90,
  "danger_window_duration": 10,
  "danger_window_duration_time_unit": "Minutes"
}
```


6.2 /api/alert-details

Url: /api/alert-details

Purpose: Retrieves details about an individual alert. The fields will match the fields seen here & here.

Type: HTTP POST or HTTP GET

Json body field descriptions:

- id – The 'alert id'. Specifies which alert you want to get details about (by id). Type: Number
- name – The 'name' field @ here. Specifies which alert you want to get details about (by name). Type: String

Notes:

- If 'id' and 'name' are both specified, the value of 'id' will be used.
- If making an HTTP GET request, specify the id or name as url-encoded query string parameters.
    - Ex: /api/alert-details?name=MyAlert
- Some fields may appear in the HTTP response that are not directly a part of an alert. For example, the 'metric_group_tags' values are properties of the 'metric group' that is associated with the alert, but they are provided in the body of this response to make the output more useful.

Example request POST body:


```
{
  "id": 944
}
```


Example response body:


```
{
 "id": 944,
 "name": "Firewall Disk Space",
```

```
  "description": "Send an alerts when firewall disk space is low. Send a caution alert when disk space is greater than 50%. Send a danger alert when disk space is greater than or equal to 90%.",
  "metric_group_id": 1161,
  "enabled": true,
  "caution_enabled": true,
  "danger_enabled": true,
  "alert_on_positive": true,
  "allow_resend_alert": true,
  "caution_notification_group_id": 976,
  "caution_positive_notification_group_id": 987,
  "caution_minimum_sample_count": 2,
  "caution_alert_active": false,
  "danger_notification_group_id": 976,
  "danger_positive_notification_group_id": 987,
  "danger_minimum_sample_count": 4,
  "danger_alert_active": false,
  "metric_group_name": "Firewall Disk Space",
    "metric_group_tags": [
      "demo",
      "disk space",
      "firewall"
    ],
  "caution_notification_group_name": "Firewall Team Oncall",
  "caution_positive_notification_group_name": "Firewall Team",
  "danger_notification_group_name": "Firewall Team Oncall",
  "danger_positive_notification_group_name": "Firewall Team",
  "alert_type": "Threshold",
  "resend_alert_every": 2,
  "resend_alert_every_time_unit": "Hours",
  "caution_operator": ">",
  "caution_combination": "Any",
  "caution_threshold": 50,
  "caution_window_duration": 4,
  "caution_window_duration_time_unit": "Minutes",
  "danger_operator": ">=",
  "danger_combination": "Any",
  "danger_threshold": 90,
  "danger_window_duration": 10,
  "danger_window_duration_time_unit": "Minutes"
}
```


6.3 /api/alert-enable

Url: /api/alert-enable

Purpose: Enable or disable an alert.

Type: HTTP POST

Json body field descriptions:

- id – The 'alert id'. Specifies which alert you want to enable or disable (by id). Type: Number
- name – The 'name' field @ here. Specifies which alert you want to enable or disable (by name). Type: String

- enabled – The ‘is alert enabled?’ field @ here. Can be ‘true’ or ‘false’. Type: Boolean

Notes:
- If ‘id’ and ‘name’ are both specified, the value of ‘id’ will be used.

Example request POST body:
```
{
  "id": 944,
  "enabled": true
}
```

6.4 /api/alert-remove

Url: /api/alert-remove

Purpose: Remove (delete) an alert.

Type: HTTP POST

Json body field descriptions:
- id – The ‘alert id’. Specifies which alert you want to remove (by id). Type: Number
- name – The ‘name’ field @ here. Specifies which alert you want to remove (by name). Type: String

Notes:
- If ‘id’ and ‘name’ are both specified, the value of ‘id’ will be used.

Example request POST body:
```
{
  "id": 944
}
```

6.5 /api/alerts-list

Url: /api/alerts-list

Purpose: Retrieves details about all of StatsAgg’s alerts. The details provided match those of /api/alert-details, but for all alerts. The fields will match the fields seen here & here.

Type: HTTP GET

Notes:
- Some fields may appear in the HTTP response that are not directly a part of an alert. For example, the ‘metric_group_tags’ values are properties of the ‘metric group’ that is associated with the alert, but they are provided in the body of this response to make the output more useful.

Example response body:
```
[
{
  "id": 944,
  "name": "Firewall Disk Space",
```

```
    "description": "Send an alerts when firewall disk space is low. Send a caution alert when disk space is greater than 50%. Send a danger alert when disk space is greater than or equal to 90%.",
    "metric_group_id": 1161,
    "enabled": true,
    "caution_enabled": true,
    "danger_enabled": true,
    "alert_on_positive": true,
    "allow_resend_alert": true,
    "caution_notification_group_id": 976,
    "caution_positive_notification_group_id": 987,
    "caution_minimum_sample_count": 2,
    "caution_alert_active": false,
    "danger_notification_group_id": 976,
    "danger_positive_notification_group_id": 987,
    "danger_minimum_sample_count": 4,
    "danger_alert_active": false,
    "metric_group_name": "Firewall Disk Space",
    "metric_group_tags": [
      "demo",
      "disk space",
      "firewall"
    ],
    "caution_notification_group_name": "Firewall Team Oncall",
    "caution_positive_notification_group_name": "Firewall Team",
    "danger_notification_group_name": "Firewall Team Oncall",
    "danger_positive_notification_group_name": "Firewall Team",
    "alert_type": "Threshold",
    "resend_alert_every": 2,
    "resend_alert_every_time_unit": "Hours",
    "caution_operator": ">",
    "caution_combination": "Any",
    "caution_threshold": 50,
    "caution_window_duration": 4,
    "caution_window_duration_time_unit": "Minutes",
    "danger_operator": ">=",
    "danger_combination": "Any",
    "danger_threshold": 90,
    "danger_window_duration": 10,
    "danger_window_duration_time_unit": "Minutes"
  },
  {
    "id": 945,
    "name": "Web Disk Space",
    "description": "Send an alerts when web server disk space is low. Send a caution alert when disk space is greater than 50%. Send a danger alert when disk space is greater than or equal to 90%.",
    "metric_group_id": 1162,
    "enabled": true,
    "caution_enabled": true,
    "danger_enabled": true,
    "alert_on_positive": true,
    "allow_resend_alert": false,
    "caution_notification_group_id": 977,
    "caution_positive_notification_group_id": 977,
```

```
  "caution_minimum_sample_count": 2,
  "caution_alert_active": true,
  "caution_alert_last_sent_timestamp": "Sep 6, 2016 9:15:00 PM",
  "caution_alert_acknowledged_status": false,
  "caution_first_active_at": "Sep 6, 2016 9:15:00 PM",
  "danger_notification_group_id": 977,
  "danger_positive_notification_group_id": 977,
  "danger_minimum_sample_count": 4,
  "danger_alert_active": true,
  "danger_alert_last_sent_timestamp": "Sep 6, 2016 9:15:10 PM",
  "danger_alert_acknowledged_status": false,
  "danger_first_active_at": "Sep 6, 2016 9:15:10 PM",
  "metric_group_name": "Web Disk Space",
  "metric_group_tags": [
   "demo",
   "disk space",
   "web"
  ],
  "caution_notification_group_name": "Web Team",
  "caution_positive_notification_group_name": "Web Team",
  "danger_notification_group_name": "Web Team",
  "danger_positive_notification_group_name": "Web Team",
  "alert_type": "Threshold",
  "caution_operator": ">",
  "caution_combination": "Any",
  "caution_threshold": 50,
  "caution_window_duration": 240,
  "caution_window_duration_time_unit": "Seconds",
  "danger_operator": ">=",
  "danger_combination": "Any",
  "danger_threshold": 90,
  "danger_window_duration": 360,
  "danger_window_duration_time_unit": "Seconds"
}
]
```


6.6 /api/suspension-create

Url: /api/suspension-create

Purpose: Create & modify suspensions.

Type: HTTP POST

Json body field descriptions:

- id – The 'suspension id'. Specify if you want to alter an existing suspension (as opposed to creating a new suspension). Type: Number
- name – The 'name' field @ here. Type: String
- description – The 'description' field @ here. Type: String
- enabled – The 'is enabled?' field @ here. Can be 'true' or 'false'. Type: Boolean
- suspend_notification_only – The 'suspend notification only?' field @ here. Can be 'true' or 'false'. Type: Boolean

- suspend_by – The ‘suspend by…’ field @ here. Valid values: ‘AlertName’, ‘Tags’, ‘Everything’, ‘Metrics’. Type: String
- alert_name – The ‘alert name’ field @ here. Type: String
- metric_group_tags_inclusive – The ‘tags’ field @ here. Type: Json Array of Strings
- metric_group_tags_exclusive – The ‘everything’ field @ here. Type: Json Array of Strings
- metric_suspension_regexes – The ‘metrics’ field @ here. Type: Json Array of Strings
- one_time – Covers the ‘recurring’ & ‘one time’ fields @ here. If this field is false, then this is a ‘recurring’ suspension. If this field is true, then this is a ‘one time’ suspension. Type: boolean
- start_date – The ‘start date’ field @ here. Type: String
- start_time – The ‘start time’ field @ here. Type: String
- recur_sunday – The Sunday field of ‘recurs on’ @ here. Can be ‘true’ or ‘false’. Type: Boolean
- recur_monday – The Monday field of ‘recurs on’ @ here. Can be ‘true’ or ‘false’. Type: Boolean
- recur_tuesday – The Tuesday field of ‘recurs on’ @ here. Can be ‘true’ or ‘false’. Type: Boolean
- recur_wednesday – The Wednesday field of ‘recurs on’ @ here. Can be ‘true’ or ‘false’. Type: Boolean
- recur_thursday – The Thursday field of ‘recurs on’ @ here. Can be ‘true’ or ‘false’. Type: Boolean
- recur_friday – The Friday field of ‘recurs on’ @ here. Can be ‘true’ or ‘false’. Type: Boolean
- recur_saturday – The Saturday field of ‘recurs on’ @ here. Can be ‘true’ or ‘false’. Type: Boolean
- duration – The numeric field of the ‘duration’ field @ here. Ex- for "duration of 4 minutes", the value of this field would be 4. Type: Number
- duration_time_unit – The time unit of the ‘duration’ field @ here. Valid values: ‘Seconds’, ‘Minutes’, ‘Hours’, or ‘Days’. Type: String

Notes:

- Modifying a suspension will replace everything about the suspension with the new fields for the suspension. This applies to **ALL** fields; not just those specified. For example, with this API, you can’t update the ‘description’ field without posting all the other fields.
- If ‘id’ is specified, then that tells this API that a suspension of the specified id will be modified with the specified values.

Example request POST body:


```
{
  "name": "Disk Space -- Demo",
  "description": "This alert suspension will suspend alerting every Saturday & Sunday night at 9pm. The suspension will last for 2 hours. Only alerts that are associated with the 'demo' tag will be suspended.",
  "enabled": true,
  "suspend_notification_only": true,
  "suspend_by": "Tags",
  "metric_group_tags_inclusive": [
    "demo"
  ],
  "one_time": false,
  "start_date": "01/01/2015",
  "start_time": "9:00 PM",
  "recur_sunday": true,
  "recur_monday": false,
  "recur_tuesday": false,
  "recur_wednesday": false,
  "recur_thursday": false,
  "recur_friday": false,
  "recur_saturday": true,
  "duration": 2,
  "duration_time_unit": "Hours"
```

}

6.7 /api/suspension-details

Url: /api/suspension-details

Purpose: Retrieves details about an individual suspension. The fields will match the fields seen here & here.

Type: HTTP POST or HTTP GET

Json body field descriptions:

- id – The 'suspension id'. Specifies which suspension you want to get details about (by id). Type: Number
- name – The 'name' field @ here. Specifies which suspension you want to get details about (by name). Type: String

Notes:

- If 'id' and 'name' are both specified, the value of 'id' will be used.
- If making an HTTP GET request, specify the id or name as url-encoded query string parameters.
    - Ex: /api/suspension-details?name=MySuspension

Example request POST body:

```
{
   "id": 353
}
```

Example response body:

```
{
  "id": 353,
  "name": "Disk Space",
  "description": "This alert suspension will suspend alerting every Saturday & Sunday night at 9pm. The suspension will last for 2 hours. Only alerts that are associated with the 'demo' tag will be suspended.",
  "enabled": true,
  "suspend_notification_only": true,
  "metric_group_tags_inclusive": [
    "demo"
  ],
  "one_time": false,
  "recur_sunday": true,
  "recur_monday": false,
  "recur_tuesday": false,
  "recur_wednesday": false,
  "recur_thursday": false,
  "recur_friday": false,
  "recur_saturday": true,
  "suspend_by": "Tags",
  "duration": 2,
  "duration_time_unit": "Hours",
  "start_date": "01/01/2015",
  "start_time": "9:00 PM"
}
```

6.8 /api/suspension-enable

Url: /api/suspension-enable

Purpose: Enable or disable a suspension.

Type: HTTP POST

Json body field descriptions:

- id – The 'suspension id'. Specifies which suspension you want to enable or disable (by id). Type: Number
- name – The 'name' field @ here. Specifies which suspension you want to enable or disable (by name). Type: String
- enabled – The 'is suspension enabled?' field @ here. Can be 'true' or 'false'. Type: Boolean

Notes:

- If 'id' and 'name' are both specified, the value of 'id' will be used.

Example request POST body:

```
{
  "id": 353,
  "enabled": true
}
```

6.9 /api/suspension-remove

Url: /api/suspension-remove

Purpose: Remove (delete) a suspension.

Type: HTTP POST

Json body field descriptions:

- id – The 'suspension id'. Specifies which suspension you want to remove (by id). Type: Number
- name – The 'name' field @ here. Specifies which suspension you want to remove (by name). Type: String

Notes:

- If 'id' and 'name' are both specified, the value of 'id' will be used.

Example request POST body:

```
{
  "id": 353
}
```

6.10 /api/suspensions-list

Url: /api/suspensions-list

Purpose: Retrieves details about all of StatsAgg's suspensions. The details provided match those of /api/suspension-details, but for all suspensions. The fields will match the fields seen here & here.

Type: HTTP GET

Example response body:


```
[
  {
    "id": 353,
    "name": "Disk Space",
    "description": "This alert suspension will suspend alerting every Saturday & Sunday night at 9pm. The suspension will last for 2 hours. Only alerts that are associated with the 'demo' tag will be suspended.",
    "enabled": true,
    "suspend_notification_only": true,
    "metric_group_tags_inclusive": [
      "demo"
    ],
    "one_time": false,
    "recur_sunday": true,
    "recur_monday": false,
    "recur_tuesday": false,
    "recur_wednesday": false,
    "recur_thursday": false,
    "recur_friday": false,
    "recur_saturday": true,
    "suspend_by": "Tags",
    "duration": 2,
    "duration_time_unit": "Hours",
    "start_date": "01/01/2015",
    "start_time": "9:00 PM"
  },
  {
    "id": 67,
    "name": "cpu_nonprd",
    "description": "Suspends non-production alerts for CPU utilization",
    "enabled": false,
    "suspend_notification_only": false,
    "metric_group_tags_exclusive": [
      "production"
    ],
    "one_time": false,
    "suspend_notification_only": true,
    "recur_sunday": true,
    "recur_monday": true,
    "recur_tuesday": true,
    "recur_wednesday": true,
    "recur_thursday": true,
    "recur_friday": true,
    "recur_saturday": true,
    "suspend_by": "Everything",
    "duration": 8,
    "duration_time_unit": "Hours",
    "start_date": "07/20/2014",
    "start_time": "9:00 PM"
  }
]
```

6.11 /api/metric-group-create

Url: /api/metric-group-create

Purpose: Create & modify metric groups.

Type: HTTP POST

Json body field descriptions:

- id – The 'metric group id'. Specify if you want to alter an existing metric group (as opposed to creating a new metric group). Type: Number
- name – The 'name' field @ here. Type: String
- description – The 'description' field @ here. Type: String
- match_regexes – The 'regular expressions' field @ here. Type: Json Array of Strings
- blacklist_regexes – The 'blacklist regular expressions' field @ here. Type: Json Array of Strings
- tags – The 'tags' field @ here. Type: Json Array of Strings

Notes:

- Modifying a metric group will replace everything about the metric group with the new fields for the metric group. This applies to **ALL** fields; not just those specified. For example, with this API, you can't update the 'description' field without posting all the other fields.
- If 'id' is specified, then that tells this API that a metric group of the specified id will be modified with the specified values.

Example request POST body:

```
{
  "name": "Application Disk Space",
  "description": "Ties together all disk-space metrics that are associated with application1 and application2.",
  "match_regexes": [
    ".*application1.*diskSpace.*",
    ".*application2.*diskSpace.*"
  ],
  "blacklist_regexes": [
    ".*application1.*diskSpace.*temp_disk",
    ".*application2.*diskSpace.*temp_disk "
  ],
  "tags": [
    "application",
    "demo",
    "disk space"
  ]
}
```

6.12 /api/metric-group-details

Url: /api/metric-group-details

Purpose: Retrieves details about an individual metric group. The fields will match the fields seen here.

Type: HTTP POST or HTTP GET

Json body field descriptions:

- id – The 'metric group id'. Specifies which metric group you want to get details about (by id). Type: Number
- name – The 'name' field @ here. Specifies which metric group you want to get details about (by name). Type: String
- include_metric_associations – Specifies if you want to include a list of current metric associations. Can be 'true' or 'false'. Type: Boolean

Notes:

- If 'id' and 'name' are both specified, the value of 'id' will be used.
- If making an HTTP GET request, specify the id or name as url-encoded query string parameters.
    - Ex: /api/metric-group-details?name=MyMetricGroup

Example request POST body:

```
{
   "id": 1130
}
```

Example response body:

```
{
  "id": 1130,
  "name": "Application Disk Space",
  "description": "Ties together all disk-space metrics that are associated with application1 and application2.",
  "match_regexes": [
    ".*application1.*diskSpace.*",
    ".*application2.*diskSpace.*"
  ],
  "blacklist_regexes": [],
  "associated_metrics": [
    "someImportantPrefix.application2.diskspace.root-drive",
    "someImportantPrefix.application2.diskspace.swap-drive"
  ],
  "tags": [
    "application",
    "demo",
    "disk space"
  ]
}
```

6.13 /api/metric-group-remove

Url: /api/metric-group-remove

Purpose: Remove (delete) a metric group.

Type: HTTP POST

Json body field descriptions:

- id – The 'metric group id'. Specifies which metric group you want to remove (by id). Type: Number
- name – The 'name' field @ here. Specifies which metric group you want to remove (by name). Type: String

Notes:

- If ‘id’ and ‘name’ are both specified, the value of ‘id’ will be used.

Example request POST body:

```
{
   "id": 353
}
```

6.14 /api/metric-groups-list

Url: /api/metric-groups-list

Purpose: Retrieves details about all of StatsAgg’s metric groups. The details provided match those of /api/metric-group-details, but for all metric groups. The fields will match the fields seen here.

Type: HTTP GET

Example response body:

```
[
 {
   "id": 1129,
   "name": "Database Disk Space",
   "description": "",
   "match_regexes": [
    ".*database.*disk.*",
   ],
   "blacklist_regexes": [],
   "tags": [
    "database",
    "demo",
    "disk space"
   ]
  },
  {
   "id": 1130,
   "name": "Application Disk Space",
   "description": "Ties together all disk-space metrics that are associated with application1 and application2.",
   "match_regexes": [
    ".*application1.*diskSpace.*",
    ".*application2.*diskSpace.*"
   ],
   "blacklist_regexes": [
    ".*application1.*diskSpace.*temp_disk",
    ".*application2.*diskSpace.*temp_disk "
   ],
   "tags": [
    "application",
    "demo",
    "disk space"
   ]
  }
]
```

6.15 /api/notification-group-create

Url: /api/notification-group-create

Purpose: Create & modify notification groups.

Type: HTTP POST

Json body field descriptions:

- id – The 'notification group id'. Specify if you want to alter an existing notification group (as opposed to creating a new notification group). Type: Number
- name – The 'name' field @ here. Type: String
- email_addresses – The 'email addresses' field @ here. Type: Json Array of Strings

Notes:

- Modifying a notification group will replace everything about the notification group with the new fields for the notification group. This applies to **ALL** fields; not just those specified.
- If 'id' is specified, then that tells this API that a notification group of the specified id will be modified with the specified values.

Example request POST body:

```
{
  "name": "Application Team Demo",
  "email_addresses": [
    "apps@noreply.com",
    "demo@noreply.com"
  ]
}
```

6.16 /api/notification-group-details

Url: /api/notification-group-details

Purpose: Retrieves details about an individual notification group. The fields will match the fields seen here.

Type: HTTP POST or HTTP GET

Json body field descriptions:

- id – The 'notification group id'. Specifies which notification group you want to get details about (by id). Type: Number
- name – The 'name' field @ here. Specifies which notification group you want to get details about (by name). Type: String

Notes:

- If 'id' and 'name' are both specified, the value of 'id' will be used.
- If making an HTTP GET request, specify the id or name as url-encoded query string parameters.
    - Ex: /api/notification-group-details?name=MyNotificationGroup

Example request POST body:

```
{
  "id": 961
```

}

Example response body:


```
{
  "id": 961,
  "name": "Application Team",
  "email_addresses": [
    "apps@noreply.com",
    "demo@noreply.com"
  ]
}
```


6.17 /api/notification-group-remove

Url: /api/notification-group-remove

Purpose: Remove (delete) a notification group.

Type: HTTP POST

Json body field descriptions:

- id – The 'notification group id'. Specifies which notification group you want to remove (by id). Type: Number
- name – The 'name' field @ here. Specifies which notification group you want to remove (by name). Type: String

Notes:

- If 'id' and 'name' are both specified, the value of 'id' will be used.

Example request POST body:


```
{
   "id": 961
}
```


6.18 /api/notification-groups-list

Url: /api/notification-groups-list

Purpose: Retrieves details about all of StatsAgg's notification groups. The details provided match those of /api/notification-group-details, but for all notification groups. The fields will match the fields seen here.

Type: HTTP GET

Example response body:


```
[
  {
    "id": 961,
    "name": "Application Team",
    "email_addresses": [
      "apps@noreply.com",
      "demo@noreply.com"
    ]
  },
  {
```

```
  "id": 962,
  "name": "Database Team",
  "email_addresses": [
   "dba@noreply.com",
   "demo@noreply.com"
  ]
 }
]
```

6.19 /api/pagerduty-service-create

Url: /api/pagerduty-service-create

Purpose: Create & modify PagerDuty services.

Type: HTTP POST

Json body field descriptions:

- id – The 'pagerduty service id'. Specify if you want to alter an existing pagerduty service (as opposed to creating a new pagerduty service). Type: Number
- name – The 'name' field @ here. Type: String
- description – The 'description' field @ here. Type: String
- routing_key – The 'routing key' field @ here. Type: String

Notes:

- Modifying a PagerDuty service will replace everything about the PagerDuty service with the new fields for the PagerDuty service. This applies to **ALL** fields; not just those specified.
- If 'id' is specified, then that tells this API that a PagerDuty service of the specified id will be modified with the specified values.

Example request POST body:

```
 {
  "name": "Application Team PagerDuty Service",
  "description": "This PagerDuty service sends incidents to the Application Team.",
  "routing_key": "akljehykljdklj45yjklefglkjsdf"
 }
```

6.20 /api/pagerduty-group-details

Url: /api/pagerduty-service-details

Purpose: Retrieves details about an individual PagerDuty service. The fields will match the fields seen here.

Type: HTTP POST or HTTP GET

Json body field descriptions:

- id – The 'pagerduty service id'. Specifies which PagerDuty service you want to get details about (by id). Type: Number
- name – The 'name' field @ here. Specifies which PagerDuty service you want to get details about (by name). Type: String

Notes:

- If ‘id’ and ‘name’ are both specified, the value of ‘id’ will be used.
- If making an HTTP GET request, specify the id or name as url-encoded query string parameters.
    - Ex: /api/pagerduty-service-details?name=MyPagerDutyServiceName

Example request POST body:

```
{
  "id": 966
}
```

Example response body:

```
{
  "id": 966,
  "name": "Application Team PagerDuty Service",
  "description": "This PagerDuty service sends incidents to the Application Team.",
  "routing_key": "akljehykljdklj45yjklefglkjsdf”
}
```

6.21 /api/pagerduty-service-remove

Url: /api/pagerduty-service-remove

Purpose: Remove (delete) a PagerDuty service.

Type: HTTP POST

Json body field descriptions:

- id – The ‘pagerduty service id’. Specifies which PagerDuty service you want to remove (by id). Type: Number
- name – The ‘name’ field @ here. Specifies which PagerDuty service you want to remove (by name). Type: String

Notes:

- If ‘id’ and ‘name’ are both specified, the value of ‘id’ will be used.

Example request POST body:

```
{
  "id": 966
}
```

6.22 /api/pagerduty-services-list

Url: /api/pagerduty-services-list

Purpose: Retrieves details about all of StatsAgg’s registered PagerDuty services. The details provided match those of /api/pagerduty-service-details, but for all PagerDuty services. The fields will match the fields seen here.

Type: HTTP GET

Example response body:

```
[
  {
    "id": 966,
```

```
    "name": "Application Team PagerDuty Service",
    "description": "This PagerDuty service sends incidents to the Application Team.",
    "routing_key": "akljehykljdklj45yjklefglkjsdf"
  },
  {
    "id": 967,
    "name": "Database Team PagerDuty Service",
    "description": "This PagerDuty service sends incidents to the Database Team.",
    "routing_key": "bsfjkerkjlewrbjkldsfsghjeuief"
  }
]
```

7 HTTP POST Tips & recommended usage patterns.

7.1 Memory optimization

StatsAgg achieves high-performance by keeping nearly all of its data in memory (Java heap). As a result, the more notifications that are sent to it, the more memory is needed. 'Notification groups' & alerts also contribute to memory usage. Here are some tips for making sure you don't run out of memory:

- Try to keep the number of notifications that get associated with a 'notification group' to a minimum.
- For alerts, keep 'window duration' to as narrow of a value as possible. StatsAgg only keeps the absolute minimum number of data points in memory that it needs to, but if the 'window duration' for an alert is set to a wide time-frame (ex—1 day), then StatsAgg will have to keep all of the data points associated with that alert's 'notification group' in memory for the entire 'window duration'.
- A 4GB heap should be enough for most moderate-use use-cases. When consuming notifications at higher rates, an 8GB heap (or more) may be needed.
    - Proper memory sizing will ultimately on your specific usage pattern, so it may take some trial/error.
- StatsD timer notifications use a lot of memory compared to other notification types. This is because StatsD timer notifications, upon aggregation, are ballooned up into many separate notifications.
    - Incidentally, StatsD timer notifications also use the most CPU of any notification type.
- StatsAgg has absolutely no limitations on how many notifications it will try to consume/keep in memory/aggregate/output/etc. If you create a 'notification group' that captures every single notification, and then create an alert (using that 'notification group') with a window-duration of 1000 years, StatsAgg will let you do this. Given enough data points, this will likely result in StatsAgg running out of memory & crashing.
- Send notifications less frequently. StatsAgg is very capable of handling millions of notification-series that send data infrequently. It will have more when receiving notifications at very high rates – even if the number of notification-series is very low.
- Because StatsAgg is written in Java & it keeps much of its data in-memory, heap dumps can be very useful in figuring out what specifically is causing all your memory usage. Is it an alert with an overly-wide 'window duration'? Is it that an application is feeding notifications into StatsAgg at a rate of 1,000,000,000 notifications per second? Heap dumps are great at helping to figure this stuff out.
    - Eclipse MAT is the first-choice for heap dump analysis.
    - VisualVM is also good at analyzing heap dumps.

7.2 TCP vs UDP

Under most circumstances, TCP is recommended over UDP for use with StatsAgg. UDP is an inherently packet-loss prone protocol, and under normal conditions, it is often possible to lose a non-trivial percentage of your packets. TCP has more guarantees on connectivity & delivery of data, so it is the preferred protocol of the author of StatsAgg. TCP & UDP notification consumption are very lightweight in StatsAgg, so there is no real performance consideration for using one or the other.

For TCP, the recommended approach is to open a connection, send the notifications, and close the connection. Leaving a connection open/hanging will tie up StatsAgg & operating system resources with no meaningful benefit.

7.3 JRE/JDK Version

- Java 17 is the minimum Java spec that StatsAgg supports. An OpenJDK based runtime is recommended.

7.4 Recommended JVM options

- Recommended garbage collector: G1GC
  - -XX:+UseG1GC
  - -XX:MaxGCPauseMillis=5000
- Maximum Heap size: this heavily depends on your usage of StatsAgg.
  - A good starting place is a 2.5GB heap via -Xmx2560m
  - StatsAgg has been tested with maximum heap sizes as low as 512MB & as high as over 100GB.
  - If StatsAgg is the only application running on your server, then consider raising the maximum heap size as large as 75% of the physical memory of your server.
- Any JVM option that wasn't explicitly mentioned somewhere in this manual is not recommended (unless you're a Java JVM tuning expert). Tune the JVM settings at your own risk.

8 Frequently asked questions

1) Why bother completely re-implementing the StatsD protocol?
   a. When StatsAgg was first started, StatsD did not support TCP-based notification transmission, so StatsAgg sought to correct that. Also, limitations like 'forgets Gauge values after a restart' were a deal-breaker for many use-cases, and that would be difficult to fix in the main StatsD application.
   b. If you don't want to use StatsAgg's implementation of StatsD, then you can send notifications to an instance of the official StatsD application & then have StatsD output its notifications to StatsAgg (which would receive them as Graphite notifications).
2) How does StatsAgg's performance compare to StatsD?
   a. In apples-to-apples performance comparisons of StatsD vs StatsAgg, StatsAgg tends to be faster by a margin of 25% to 50%. These results may vary by hardware, StatsAgg JVM options, application configurations, etc.
   b. The above StatsD performance benchmark was conducted on a multi-core server. StatsD is largely a single-threaded application, and it may perform better than StatsAgg on a single-core server.
3) OpenTSDB listens for notifications on a single port for the telnet & HTTP formats. StatsAgg listens on two different ports. Why?
   a. OpenTSDB implemented their TCP listener in a non-traditional way. That is to say, it is uncommon to have a single port listen for two completely different protocols. Because of the difficulty of implementation, StatsAgg currently listens for notifications on two different ports. One is a user-defined port for OpenTSDB HTTP notification input (4243 by default), and the other is a user-defined port for OpenTSDB telnet notification inputs (4242 by default).
4) "Do I *need* to output to Graphite/OpenTSDB/Influx/etc"?
   a. No. StatsAgg doesn't mandate that notifications be output anywhere.
5) "Can I run a multi-server deployment of StatsAgg?"
   a. No. StatsAgg only works in a single-server deployment model. There are no plans to support a multi-server deployment model in the future.
   b. Why not!?
      i. StatsD Gauge notifications. Gauge notifications can be directly set, incremented, or decremented – and the order matters. If the order is mixed up, even slightly, then the value of the Gauge notification could end up being completely different/wrong. There is no efficient way

to guarantee correct ordering with multiple servers for notifications types like this (at least not that this author could think of…).

ii. Alerting. StatsAgg's alerting mechanism expects to be able to quickly assess what notifications are in an alerted state. If multiple servers had a partial dataset of the notification-values for a notification-key, then alert criteria couldn't be properly evaluated.

iii. You can always just run multiple StatsAgg servers that are completely independent from each other.

6) "Performance?"
    a. StatsAgg's author has a day-job of 'Performance Engineer', so a lot of effort has gone into making StatsAgg perform well under heavy-load conditions. To give a general sense of overall performance, StatsAgg will use…
        - around 40% CPU & ~1GB JVM Heap on a 4-core Intel i5 processor VM (running Windows 2008 R2)…
        - … while consuming 100,000 Graphite-Aggregated notifications per second & a diverse set of 100,000 StatsD notifications per second. That's 200,000 notifications per second total…
        - … with 50 notification groups & 50 alerts set (window durations under 5mins).

    If one were to send mostly Graphite-Passthrough notifications, then StatsAgg could potentially scale to millions of notifications per second on moderate hardware.
7) StatsAgg uses a lot of CPU after a restart! What gives?!
    a. As a result, after a StatsAgg restart, it is common to have a period of high CPU utilization while notifications-keys are re-associated with 'notification groups'. The time it takes will be proportional to (#unique-notification-keys * #notification-groups).
    b. The 'heaviness' of the regular-expressions that are associated with the 'notification groups' also plays a large role in contributing to the startup time. Try to reduce the complexity of your 'notification group' regular-expressions whenever possible.
8) I altered a 'notification group' and/or alert and my alert status changed from triggered->not triggered. Why?
    a. Altering a 'notification group' is treated as a complete refresh of everything unique about that 'notification group'. That means that all 'notification group' associations are reset, and all data points that were being kept in memory because of their association with the 'notification group' were allowed to be cleaned up.
    b. Altering an alert mandates a reset of the status of the alert. If the alert is still in a triggered state after the alteration, then it will re-alert (complete with a new notification).
9) Is any data lost as it passes through StatsAgg? Is numeric precision/accuracy maintained?
    a. StatsAgg internally stores all notification-values with full numeric precision intact. Furthermore, StatsAgg does not use floating-point math operators, so you'll never have a situation where the 'average' value of a set of notifications is equal to '74.999999' instead of '75'.
        - The only place where precision may be lost is in the post-aggregation output of the StatsD & Graphite-Aggregator routines. Whole numbers always maintain full precision, but the fractional part (the scale of the number) is limited to 7 decimals. In such a situation, the last decimal is rounded up.
10) Will StatsAgg ever support receiving notifications in format ABC?
    a. Support for various notification formats will improve over time. The only criteria is that the notification format must be open-source & capable of working within the established StatsAgg framework.
    b. Feel free to suggest/request support for a particular notification format.
11) Will StatsAgg ever output to service XYZ?
    a. Support for writing to other backends will likely be built into StatsAgg over time. Possible candidates include: InfluxDB, MySQL, Mongo, PostgreSQL, Apache Derby. Backends for commercial services are unlikely to be built into StatsAgg. This is because the author of StatsAgg doesn't feel that it is

appropriate to selectively implement solutions for particular closed-sourced (for-profit) companies & technologies.

9 Troubleshooting

1. Many errors/issues with StatsAgg will be captured in StatsAgg log file(s). The main log file is located at (statsagg-root-folder)/logs/statsagg.log.
2. StatsAgg doesn't seem to shutdown, or litters the logs when shutting down.
    a. StatsAgg attempts to shutdown gracefully, which gives each of the threads several seconds to shut down. If these threads don't shut down after a certain period of time, they are forcibly terminated. This may result in errors in the log files, but the errors should be harmless overall. StatsAgg was built to be resilient to hard crashes.
3. The StatsAgg WebUI homepage says "StatsAgg is using its default application configuration settings. This occurs when there was an error loading the 'application.properties' configuration file."
    a. As the error suggests, this usually means that the application.properties file was not created. This discusses how create/configure the application.properties file. If there was some other type of error (invalid settings, etc), then odds are good that you will find a more detail description of the error in the StatsAgg log file.
4. The StatsAgg WebUI homepage says "StatsAgg is using an ephemeral (in-memory) database. This occurs when there was an error loading the 'database.properties' configuration file."
    a. As the error suggests, this usually means that the database.properties file was not created and/or improperly configured. This discusses how create/configure the database.properties file. If there was some other type of error (invalid settings, etc), then odds are good that you will find a more detail description of the error in the StatsAgg log file.
5. The StatsAgg WebUI homepage says "StatsAgg did not successfully connect to the database. Please view the StatsAgg log files for more details."
    a. This most frequently occurs StatsAgg can't connect to a MySQL database. This could be because MySQL has crashed, connectivity was lost (network is down, firewall, etc), MySQL is overload/unresponsive, etc. You will have to investigate what is wrong with the StatsAgg ←→ database interaction to solve this problem. Oftentimes the StatsAgg log file is helpful in these circumstances.
    b. Once the issue is discovered/fixed, it is recommended that StatsAgg be restarted.
6. Alerts/emails/WebUI are out of sync with each other. What gives?
    a. While this behavior is rare, such behavior can happen when the StatsAgg database becomes unavailable for a period of time. This is exclusively a MySQL concern. The underlying reason for this behavior is that the StatsAgg application caches a lot of data to keep performance high. Cases like this are being identified/fixed, but there may be edge-cases that could lead to synchronization issues.
    b. When such behavior is noticed, restart StatsAgg.

10 Known bugs/limitations/etc

- A few StatsD features that are in the main StatsD program are missing in StatsAgg. Missing features include…
    - configuring StatsAgg to be a 'repeater' (you can use the official StatsD program to do this and forward to StatsAgg).
    - configuring StatsAgg to be in a 'proxy cluster' configuration (you can use the official StatsD program to do this, and forward to StatsAgg)
    - outputting frequently sent notification-keys to log files

- StatsAgg is not (currently) meant for use as an incident management tool. It doesn't support alert history, event management, etc.
- StatsAgg only supports running in a single-server configuration. See the FAQs section for more information.
- StatsAgg does not currently support the Graphite 'Pickle' format (may be included in a future build).
- OpenTSDB listens for notifications on a single port for the telnet & HTTP formats. StatsAgg listens for OpenTSDB notifications on two different ports. See the FAQs section for more information.
- The OpenTSDB HTTP interface has full support for the 'summary' parameter, and partial support for the 'details' parameter. See the OpenTSDB documentation for more information.
    - The 'details' parameter will return accurate counts for 'failed' & 'success', but the 'errors' field is not currently being populated.

11 About StatsAgg

Development for StatsAgg started in August 2013 by Pearson Assessments (a division of Pearson Education). It started as an internal application that was conceived as an enhanced version of StatsD, and continued to evolve after the initial design goals were achieved.

StatsAgg is developed in Java. Java was chosen because it is cross-platform, well supported, mature, and achieves relatively high performance.

StatsAgg's design was based on the following principles:

- Keep framework usage to an absolute minimum. Only modular libraries were used (libraries that can easily be swapped out).
    - No hibernate, struts, spring, play!, etc
- CPU & Memory usage must linearly scale by input notification count, notification group count, alerts count, etc. Exponential usage of CPU or Memory as a factor of any StatsAgg input was specifically avoided.
- Assume people want LOTS of unique notifications & want frequent data points for each notification.
- An admin-interface that is intuitive, but powerful.

12 License