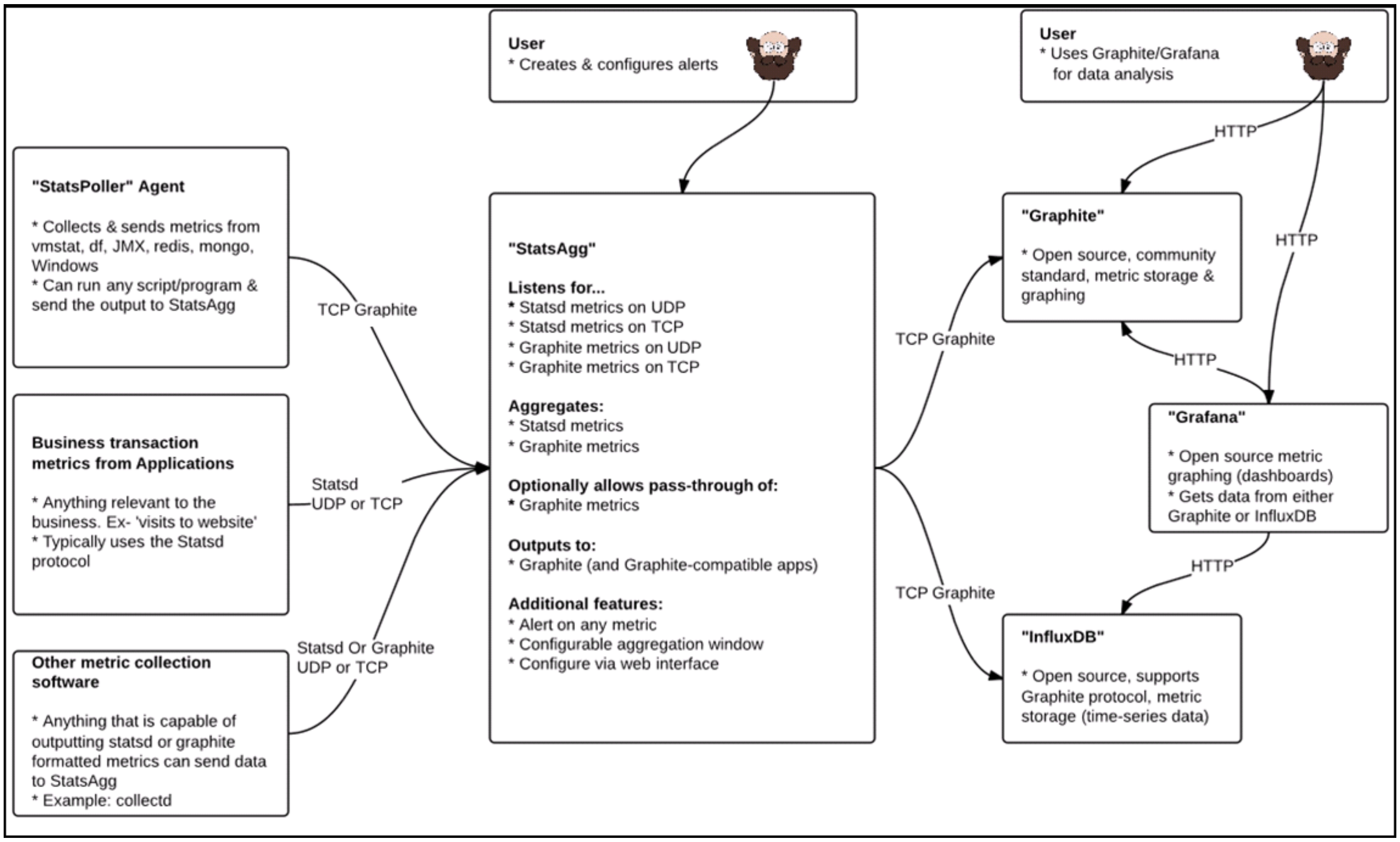
StatsAgg User Manual

Contents

1) Overview

StatsAgg is a metric aggregation and alerting platform. It currently accepts Graphite-formatted metrics and StatsD formatted metrics.

StatsAgg works by receiving Graphite & StatsD metrics, aggregating them, alerting on them, and outputting the aggregated metrics to a Graphite-compatible metric storage platform. The diagram (linked to below) shows a typical deployment & use-case pattern for StatsAgg.


User
* Creates & configures alerts
User
* Uses Graphite/Grafana for data analysis
"StatsPoller" Agent
* Collects & sends metrics from vmstat, df, JMX, redis, mongo, Windows
* Can run any script/program & send the output to StatsAgg
TCP Graphite
Business transaction metrics from Applications
* Anything relevant to the business. Ex- 'visits to website'
* Typically uses the Statsd protocol
Statsd UDP or TCP
Other metric collection software
* Anything that is capable of outputting statsd or graphite formatted metrics can send data to StatsAgg
* Example: collectd
Statsd Or Graphite UDP or TCP
"StatsAgg"
Listens for...
* Statsd metrics on UDP
* Statsd metrics on TCP
* Graphite metrics on UDP
* Graphite metrics on TCP
Aggregates:
* Statsd metrics
* Graphite metrics
Optionally allows pass-through of:
* Graphite metrics
Outputs to:
* Graphite (and Graphite-compatible apps)
Additional features:
* Alert on any metric
* Configurable aggregation window
* Configure via web interface
TCP Graphite
TCP Graphite
HTTP
HTTP
"Graphite"
* Open source, community standard, metric storage & graphing
HTTP
"Grafana"
* Open source metric graphing (dashboards)
* Gets data from either Graphite or InfluxDB
HTTP
"InfluxDB"
* Open source, supports Graphite protocol, metric storage (time-series data)

1.1) Core Features

- A complete re-implementation of StatsD
  a. TCP & UDP support (both can run concurrently).
  b. Support for all metric-types.
- Receives (and optionally, aggregates) Graphite metrics
  a. TCP & UDP support (both can run concurrently).
  b. Aggregates into minimum, average, maximum for the values of an individual metric-path (during an 'aggregation window').
- Outputs aggregated metrics to Graphite (and Graphite compatible services)
- A robust alerting mechanism
  a. Can alert off of any received/aggregated metric.
  b. Regular-expression based mechanism for tying together metrics & alerts.
  c. 'threshold' based alerting, or 'availability' based alerting.
  d. A flexible alert-suspension mechanism.

    e. Alerts notifications can be sent via email, or viewed in the StatsAgg website.
- A web-based UI 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 and StatsD do not have a native alerting mechanism
    a. Most alerting solutions for StatsD and/or Graphite metrics are provided by (expensive) SaaS venders.
    b. The alerting mechanism in StatsAgg compares favourably to many pay-based solutions.
- 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 can structure your metrics in such a way so as 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 datapoint 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 persists Gauge values, so a 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.

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.

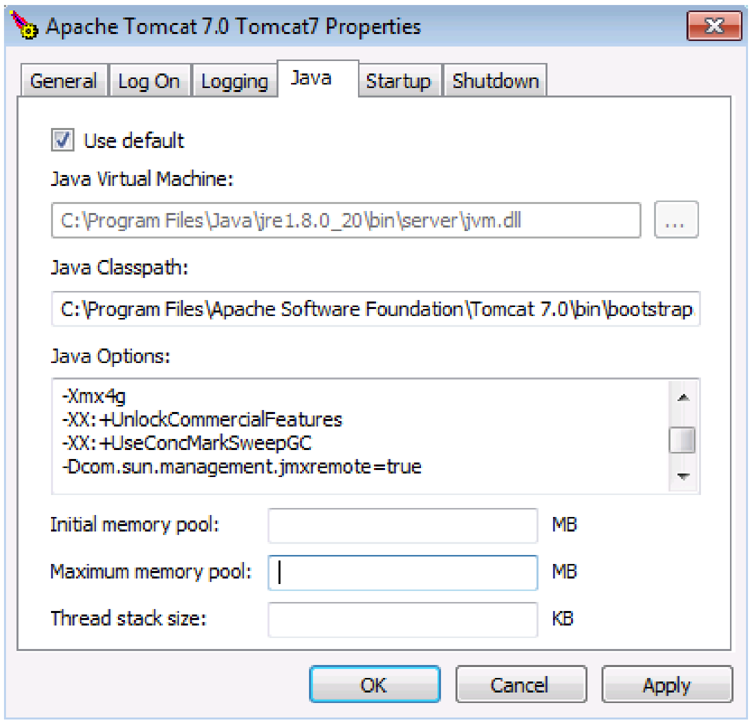
2.1) System requirements

- Oracle Java 1.7+ JRE (64bit recommended)
- Tomcat 7.0.x
    - Other versions of Tomcat may work, but are untested. Likewise, other application servers may work (Weblogic, JBoss, etc), but are untested.
- 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 (along with Tomcat) will take less than 100MB, but StatsAgg is configured to retain up to 1.1GB of log file data.
- Any OS capable of running Oracle Java 1.7 & Tomcat 7.
    - Windows XP & higher (though using a recent release of Windows is highly recommended)
    - Almost all recent versions of Linux

2.2) Windows installation

1. Install an Oracle 1.7+ Java JDK
   - Section 5.3 discusses this in more detail.
2. Install the latest version of Apache Tomcat 7 using the MSI installer. The following non-default configuration settings are recommended:
   - Check: Install the APR based native .dll (recommended, not required)
   - Check: Service Startup (recommended, not required)
   - Set: HTTP/1.1 connector port = 80 (recommended, not required)
3. (Optional, Recommended) Add these 'Java options' to Tomcat through the Tomcat configuration application.
   - More on this in section 6.4.
   - Xmx (max heap size) is discussed in more detail in section 6.4. *-Xmx2560m* is merely a suggestion.
   - Remote JMX options can be added to allow tools like VisualVM to connect. If you don't intend on using tools like VisualVM, then don't set the jmxremote options.

```
-Xmx2560m
-XX:+UnlockCommercialFeatures
-XX:+UseConcMarkSweepGC
-Dcom.sun.management.jmxremote=true
-Dcom.sun.management.jmxremote.port=20000
-Dcom.sun.management.jmxremote.ssl=false
-Dcom.sun.management.jmxremote.authenticate=false
```



Use default [✓]

Java Virtual Machine: C:\Program Files\Java\jre1.8.0_20\bin\server\jvm.dll

Java Classpath: C:\Program Files\Apache Software Foundation\Tomcat 7.0\bin\bootstrap

Java Options:

```
-Xmx4g
-XX:+UnlockCommercialFeatures
-XX:+UseConcMarkSweepGC
-Dcom.sun.management.jmxremote=true
```

Initial memory pool: MB

Maximum memory pool: MB

Thread stack size: KB

4. Configure Tomcat
   - See section 2.4.
5. Deploy StatsAgg to Tomcat
   - Stop the "Apache Tomcat 7.0 Tomcat7" service
   - Create a folder in (Tomcat_Home)/webapps called "StatsAgg"
   - Unzip the StatsAgg file into (Tomcat_Home)/webapps/StatsAgg
6. Configure the application.properties file & the database.properties file (see section 2.5)

    - Be especially mindful to properly configure the database.properties file. If this this isn't configured properly, StatsAgg won't function properly.
7. Start the "Apache Tomcat 7.0 Tomcat7" service.

2.3) Linux installation

Because installation instructions vary widely based on distro, only high-level instructions will be provided for Linux deployments.

1. Install an Oracle 1.7+ Java JDK
2. Install Tomcat 7
    a. Configure Tomcat to explicitly use the version of Java that was installed in step 1, or change the OS path to use JDK installed in step 1 as the default JDK.
3. Configure Tomcat
    a. (Optional, Recommended) Set Java options.
        i. Create a file -- (Tomcat_Home)/bin/setenv.sh
            - Make sure the file can be read by the user that Tomcat will run as
        ii. Add this line into the file:
        export JAVA_OPTS="-Xmx2560m -XX:+UnlockCommercialFeatures -XX:+UseConcMarkSweepGC -Dcom.sun.management.jmxremote=true -Dcom.sun.management.jmxremote.port=20000 -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false"
            - More on this in section 6.4.
            - Xmx (max heap size) is discussed in more detail in section 6.4. *-Xmx2560m* is merely a suggestion.
            - Remote JMX options can be added to allow tools like VisualVM to connect. If you don't intend on using tools like VisualVM, then don't set the jmxremote options.
            - If you have Tomcat configured to read JAVA_OPTS from some other location, then set the JAVA_OPTS there.
    b. Make the changes to server.xml discussed in section 2.4.
4. Deploy StatsAgg to Tomcat 7.
    a. Stop Tomcat. On many systems, this can be done with "sudo service tomcat7 stop".
    b. Create a folder in (Tomcat_Home)/webapps called "StatsAgg"
    c. Unzip the StatsAgg war file into (Tomcat_Home)/webapps/StatsAgg
5. Configure the application.properties file & the database.properties file (see section 2.5)
    a. Be especially mindful to properly configure the database.properties file. If this file isn't configured properly, StatsAgg may not function properly.
6. Start Tomcat. On many systems, this can be done with "sudo service tomcat7 start".

2.4) Tomcat configuration

In the (Tomcat_Home)/conf/server.xml file, change the HTTP connection to look like this:

```
...

<!-- port should be set to whatever port want to interact with the StatsAgg WebUI on -->
 <Connector port="80" protocol="HTTP/1.1"
            connectionTimeout="20000"
            compression="on"
```

```
        useSendfile="false"
        compressableMimeType="text/html,text/xml,application/javascript,text/css,application/x-font-woff"
        compressionMinSize="2048"
        redirectPort="8443" />

...
```

2.5) 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.

2.5.1) 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" @ (tomcat-location)\webapps\StatsAgg\WEB-INF\config
   Example location (Windows): C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps\StatsAgg\WEB-INF\config\application.properties
   - You can override the default configuration file location by adding a Java argument to Tomcat: -*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.

The full list settings, default values, and examples usages can be found in StatsAgg\WEB-INF\config\example_application.properties

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

- Output to Graphite
  - graphite_output_module_1 = true,some-graphite-server.some-domain.com,2003,3
- 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

2.5.2) 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/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.

2.5.2.1) Apache Derby Embedded

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

- No separate sever 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" @ (tomcat-location)\webapps\StatsAgg\WEB-INF\config
   Example location (Windows): C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps\StatsAgg\WEB-INF\config\database.properties
   - You can override the default configuration file location by adding a Java argument to Tomcat: -*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. The settings listed below are the recommended set of database.properties settings for Apache Derby Embedded.

```
#database.properties settings
cp_max_connections = 50
cp_acquire_retry_attempts = 3
cp_acquire_retry_delay = 250
cp_connection_timeout = 5000
cp_enable_statistics = false
cp_default_auto_commit = false
connection_validity_check_timeout = 5

# Set db_localpath to the location created in step #2. Example: C:\\StatsAgg\\Data
# Set the username/password to whatever values you deem appropriate.
db_type = derby_embedded
db_localpath =
db_name = statsagg_db
db_username =
db_password =
db_attributes = create=true;upgrade=true;

derby.system.home = ${db_localpath}
derby.infolog.append = true
derby.stream.error.file = ${db_localpath}${file.separator}${db_name}${file.separator}log${file.separator}db_log.txt
derby.storage.pageSize = 16384
derby.storage.pageCacheSize = 15000
```

A full list settings, default values, and examples usages can be found in StatsAgg\WEB-INF\config\example_database.properties

2.5.2.2) 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…

1. Install/configure MySQL 5.6+
    - Percona Server 5.6+ & MariaDB 10+ are supported.
    - MySQL's configuration can get really complex. The MySQL configuration settings listed below are recommended 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.
    autocommit = 0
    innodb_file_format = Barracuda
    innodb_file_format_max = Barracuda
    innodb_file_per_table = ON
    innodb_strict_mode = ON
    max_allowed_packet = 64M
    sql_mode = STRICT_ALL_TABLES
    ```

2. Connect to MySQL & run "create database statsagg;".
3. 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.
4. Edit "database.properties" @ (tomcat-location)\webapps\StatsAgg\WEB-INF\config
   Example location (Windows): C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps\StatsAgg\WEB-INF\config\database.properties
    - You can override the default configuration file location by adding a Java argument to Tomcat: -*DsaDbConfLocation="YourLocation"*
      Example (Windows): -DsaDbConfLocation="C:\StatsAgg_Conf\database.properties"
5. The settings listed below are the recommended set of database.properties settings for MySQL.

```
#database.properties settings
cp_max_connections = 50
cp_acquire_retry_attempts = 3
cp_acquire_retry_delay = 250
cp_connection_timeout = 1000
cp_enable_statistics = false
cp_default_auto_commit = false
connection_validity_check_timeout = 1

# Set the hostname/port to point to the server that MySQL runs on. If running MySQL on the same server as StatsAgg, use
```

```
db_hostname=127.0.0.1
# Set the username/password to whatever values you deem appropriate.
db_type = mysql
db_hostname =
db_port = 3306
db_name = statsagg
db_username =
db_password =
db_attributes = autoReconnect=true&failOverReadOnly=false&maxReconnects=1
```

Notes

- A full list settings, default values, and examples usages can be found in StatsAgg\WEB-INF\config\example_database.properties
- StatsAgg was only tested against MySQL 5.6. However, it should work on MySQL 5.5+ (the author has no intentions of testing on any platform lower than MySQL 5.6).
- 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.

2.5.2.3) 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, and 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).

2.5.3) StatsAgg WebUI security configuration (optional)

The initial release plans for StatsAgg did not include any form of security for the WebUI. However, since Tomcat provides a mechanism for relatively easy/simple authentication, StatsAgg can be configured to use authentication.

This can be accomplished via the following steps:

1. Open "(tomcat-location)\webapps\StatsAgg\WEB-INF\web.xml" in a text editor
    a. Uncomment the <security-constraint> & <login-config> sections
2. Open "(tomcat-location)\conf\tomcat-users.xml" in a text editor
    a. Add the following roles & users into the <tomcat-users> section. Substitute your own passwords.
    ```
    <role rolename="statsagg-admin"/>
    <user username="admin" password="somePW_1" roles="statsagg-admin"/>
    <role rolename="statsagg-read-only"/>
    <user username="guest" password="somePW_2" roles="statsagg-read-only"/>
    ```
3. Restart Tomcat

Notes

- Without securing the wire traffic, Tomcat's authentication mechanism is not particularly secure. Ideally, HTTP traffic should be SSL encrypted. It is up to the user to configure this.
- StatsAgg currently does not provide a 'logout' mechanism. Usually closing & reopening your browser will accomplish this.
- Authentication is NOT enabled by default

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'
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 'datapoint'.
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 will listen for Graphite metrics on ports 2003 & 2004 (TCP & UDP, on both ports). Port 2003 is used for Graphite 'passthrough' metrics, and port 2004 is used for Graphite 'aggregated' metrics. For more information on the distinction between 'passthrough' and 'aggregated', please read sections 3.2.3 & 3.2.4.

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.
- 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.description.name.for.a.metric1 123 1408334418\nsome.description.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.
- After a restart, StatsAgg forgets metric-paths for Graphite metrics. As a result, StatsAgg won't be capable of outputting the previous value for a metric-path until it receives at least one new datapoint for that metric-path (assuming that StatsAgg is configured to output the previous value for a metric-path that hasn't recently had a new value).

3.2.4) 'Aggregated' metrics

By default, port 2004 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.
It 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: Average Value, Minimum Value, Maximum Value
    c. The average value will round 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…

folder1.folder2.value1.Min 2 1408334417\n
folder1.folder2.value1.Max 10 1408334417\n
folder1.folder2.value1.Avg 6 1408334417\n
```

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.
- 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 datapoint.

Notes:

- After a restart, StatsAgg forgets metric-paths for Graphite metrics. As a result, StatsAgg won't be capable of outputting the previous value for a metric-path until it receives at least one new datapoint for that metric-path (assuming that StatsAgg is configured to output the previous value for a metric-path that hasn't recently had a new value).

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. However, there are implementation differences that users should be aware of. Please read the metric format section for more details.

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|statsdMetricType\n

Notes:

- StatsD calls the name of a metric a "bucket" (instead of 'metric-path', 'metric key', etc)
- 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 datapoint.
  - 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 datapoint 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 datapoint 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 values or additive values. Additive values are prefixed by +/- signs. 'Set' values do not have a prefix. Anytime a 'set' value is encountered, it immediately sets the gauge to equal the value that it is being set to.

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:

- Unlike the official StatsD application, you **can** set a gauge to a negative number without first setting it to zero.

- 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’, 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.

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

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’)
- count_ps (the per-second rate of ‘how many metrics were in a bucket’s list of metrics’)
  - count_ps = count/flush_interval
- 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.

Notes:

- At this time, StatsAgg does not support timer histograms. This will likely be added in a future release of StatsAgg.
- 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.
- There is undocumented support for sampling based timers in the official StatsD application. At this time, StatsAgg does not support this functionality, and will not support this functionality until StatsD's documentation formally declares support for this syntax.
- 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 datapoint 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 datapoint 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,
	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
```

3.4) Outputs

StatsAgg can optionally output metrics to Graphite. Graphite, and Graphite compatible programs (ex- InfluxDB), are the only programs that StatsAgg currently supports outputting to.

Notes:

- The only metrics that are outputted are 'aggregated' metrics.
    - In the case of Graphite-Passthrough metrics, this simply means that the metrics are sent to Graphite (though their metric-key may have been slightly altered by StatsAgg).
- After X retries (by default, X=3), 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 discarded permanently.
    - A 'batch' means the aggregation window's worth of StatsD metrics, Graphite-Passthrough metrics, or Graphite-Aggregation metrics. StatsAgg handles these different metric types on separate threads, so for a given aggregation window, the 'batchs' for StatsD metrics, Graphite-Passthrough metrics, and Graphite-Aggregation 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.

4) Alerting

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

4.1) How does alerting work?

The alert routine is independent from all of the metric aggregation routines (StatsD, Graphite-Passthrough, Graphite-Aggregated). 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, 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.
2. Determine what alerts are "suspended". This maps StatsAgg 'Alert 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 alerts are both evaluated at this stage. Alerts that are in a triggered state are marked as such.
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.
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. Metric cleanup. StatsAgg evicts metric key/value pairs out of memory after they are no longer of any use.

    a. Warning – if you set the StatsAgg ‘alert routine execution frequency’ to a higher-than-recommended value, then the metric cleanup routine will execute less frequently. This means that you increase the potential of filling up StatsAgg’s heap memory space.
7. 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.

4.2) 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 datapoints 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’.

4.3) 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 metric aggregation routines.
- **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 metrics are not aggregated, every single individual datapoint 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’.
4. Create an ‘alert suspension’. ‘Alert suspensions’ allow you to temporarily disable alerting. If you don’t need to suspend alerts, then you can ignore ‘alert suspensions’ entirely.

The remains subsections go into greater detail about how to create & configure notification groups, metric groups, alerts, and alert 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.

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.

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 datapoint within the last 24hrs will show up as being ‘associated’ with a ‘metric group’. Metrics that haven’t received any new datapoints for 24hrs will be ‘forgotten’ by StatsAgg.

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 (optional): 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. The regular

expression format is Java Regex. For more information, visit:
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.

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.

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.

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.

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 datapoints. 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 datapoint for a ‘metric-key’ is the ‘positive-alert’ criteria for the availability alerts.

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 email notification to the notification group’s recipients after the specified amount of time. The new email notification will reflect whatever metrics are currently resulting in the alert to be triggered (not just whatever was in the original email notification)

<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.

Window duration (availability and threshold alerts): The ‘window duration’ is an amount of time (between ‘now’ and ‘x seconds 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’.

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 datapoints. 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 (in seconds) that you will wait for a new datapoint to show up for a metric before the availability alert ‘gives up’ on looking for a new datapoint 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 ‘window duration’. For example, if the ‘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.

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' specifies how many values are required.

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. You ca
- 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 datapoints 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) Alert Suspensions

5.6.1) Alerts Suspensions table

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

Colored rows

Blue: The 'alert suspension' is currently active.

5.6.2) Create Alert Suspension

Options

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

Is enabled? (required): If you want the alert suspension to be enabled after creation or alteration, then check this checkbox. When an 'alert suspension' is disabled, the alert 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.

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 'alert suspension'. Only put one tag per line in the 'tags' textbox.

Example: if the 'alert 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 'alert suspension'. Only put one tag per line in the 'everything' textbox.

Example: if the alert 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

Suspension Type…

Recurring (daily): This suspension type means that the 'alert suspension' recurs every day.

One Time: This suspension type means that the 'alert suspension' is only ever used once. Once the 'alert suspension' is complete, it will self-delete.

Start Date: Recurring 'alert 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: 'Alert 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 'alert suspension' become active?
The format is: h:mm a.
Example: 9:23 AM

Duration (mins): Specifies the amount of time that the 'alert suspension' remains active (in minutes). This must be a positive value, and must be less than 1441.

Notes

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

The regular expression format is Java Regex. For more information, visit:
https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html

5.8) 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.

5.9) 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) Tips & recommended usage patterns.

6.1) Memory optimization

StatsAgg achieves high-performance by keeping nearly all of its data in memory. As a result, the more metrics that are sent to it, the more memory is needed. 'Metric 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 metrics that get associated with a metric group to a minimum. StatsAgg has to keep track of how each metric maps to each 'metric group'.
- 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 metric group in memory for the entire 'window duration'.
- A 4GB heap should be enough for most moderate-use use-cases. When consuming metrics 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 metrics use a lot of memory compared to other metric types. This is because StatsD timer metrics, upon aggregation, are ballooned up into many separate metrics.
  - Incidentally, StatsD timer metrics also use the most CPU of any metric type.
- StatsAgg has absolutely no limitations on how many metrics it will try to consume/keep in memory/aggregate/output/etc. If you create a metric group that captures every single metric, and then create an alert (using that 'metric group') with a window-duration of 1000 years, StatsAgg will let you do this. Given enough datapoints, this will likely result in StatsAgg running out of memory & crashing.
- 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 metrics into StatsAgg at a rate of 1,000,000,000 metrics 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.

6.2) 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 metric 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 metrics, and close the connection. Leaving a connection open/hanging will tie up StatsAgg & operating system resources with no meaningful benefit.

6.3) JRE/JDK Version

- Java 1.7 is the minimum Java spec that StatsAgg supports.
- Oracle Java 1.8, x64, is the recommended runtime engine for StatsAgg.
- StatsAgg was primarily developed & tested against Oracle Java on Windows.
- OpenJDK should work with StatsAgg, but it is untested, so functionality & performance are unknown.

- Using the full JDK package is recommended, as the JDK package comes with diagnostic tools that can be used to troubleshoot StatsAgg.
    - Ex – VisualVM

6.4) Recommended VM options

- Recommended garbage collector: Concurrent Mark & Sweep.
    - -XX:+UseConcMarkSweepGC
- Heap size: this heavily depends on your usage of StatsAgg.
    - A good starting place is a 2.5GB heap.
    - StatsAgg has been tested with max heap sizes as low as 1GB & as high as 8GB.

7) Frequently asked questions

1) Why bother completely re-implementing the StatsD protocol?
    a. When StatsAgg was first started, StatsD did not support TCP-based metric 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 metrics to an instance of the official StatsD application & then have StatsD output its metrics to StatsAgg (which would receive them as Graphite metrics).
2) "Do I *need* to output to Graphite/Influx/etc"?
    a. No. StatsAgg doesn't mandate that metrics be output anywhere.
3) "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 metrics. Gauge metrics 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 metric could end up being completely different/wrong. There is no efficient way to guarantee correct ordering with multiple servers for metrics 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 metrics are in an alerted state. If multiple servers had a partial dataset of the metric-values for a metric-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.
4) "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 metrics per second & a diverse set of 100,000 StatsD metrics per second. That's 200,000 metrics per second total, and does not include any Graphite-Passthrough metrics…
        - … with 50 metric groups & 50 alerts set (window durations under 5mins).

        If one were to send most Graphite-Passthrough metrics, then StatsAgg could scale to millions of metrics per second on moderate hardware.
5) 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 metrics-keys are re-associated with 'metric groups'. The time it takes will be proportional to (#unique-metric-keys * #metric-groups).

6) 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: OpenTSDB, 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.

8) Troubleshooting

1. Many errors/issues with StatsAgg will be captured in StatsAgg log file(s). The main log file is located at (tomcat-location)/logs/statsagg.log, but you may also find it useful to inspect other log files (like catalina.out).
2. 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. Section 2.5.1 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.
3. 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. Section 2.5.2.x 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.
4. 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. The StatsAgg log file is located at: (tomcat-location)/logs/statsagg.log.
    b. Once the issue is discovered/fixed, it is recommended that StatsAgg be restarted.
5. 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 almost 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.

9) Known bugs/limitations/etc

- A few StatsD features that are in the main StatsD program are missing in StatsAgg. Missing features include…
    - histograms on timers (will be included in a future build)
    - 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 metric-keys to log files
- StatsAgg is not (currently) meant for use as an incident management tool. It doesn't support alert history, alert acknowledgement, 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 (will be included in a future build)

10) About StatsAgg

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 metric count, metric 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 metrics & want frequent datapoints for each metric.
- An admin-interface that is intuitive, but powerful.

11) Credits

Lead developer/designer/architect/documenter: Jeffrey Schmidt
WebUI consultants: Pasi Jouhikainen, Adam Parsons, Deepak Patnam
Architecture consultations: Pasi Jouhikainen, Adam Parsons, Michael Anderson

StatsD: etsy @ https://github.com/etsy/statsd/
Graphite: Orbitz @ http://graphite.wikidot.com/
Pearson Assessments, a division of Pearson Education: http://www.pearsonassessments.com/