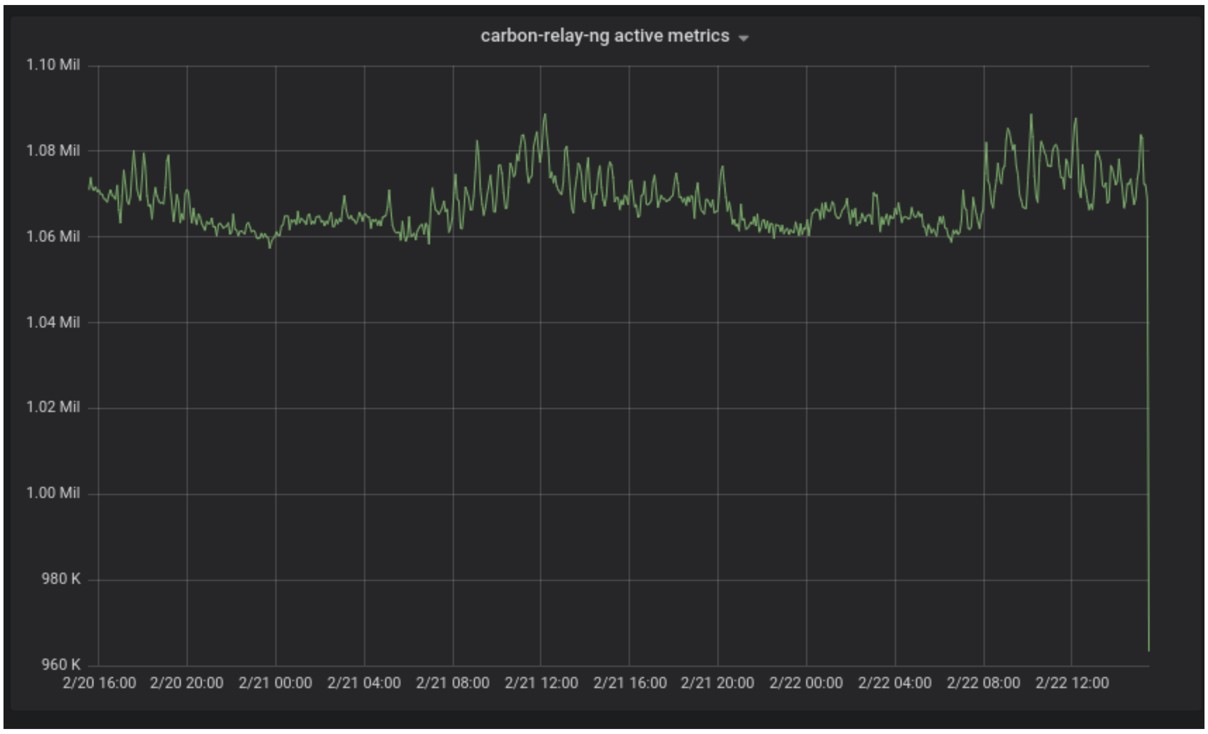
Scaling Graphite to Millions of Metrics

Author: John Meichle

Claps: 302

Date: Feb 23, 2019

Klaviyo Engineering has had a long and incredibly successful history with Graphite instrumentation â€” we love the technology so much weâ€™ve even built video walls showcasing our Grafana dashboards. We rely on it for deep visibility across our workloads for common use cases such as system throughput, latency, and overall performance. Although we started with a fairly vanilla Graphite and Statsd server, Klaviyoâ€™s growth over the years forced an evolution of our backend infrastructure providing the Graphite stack. This post summarizes the challenges weâ€™ve had as weâ€™ve grown our usage of time series instrumentation and how the Graphite ecosystem at Klaviyo has scaled. Currently our stack reliably handles over a million active metric keys at any given time across 17 million total metric keys



Active metric keys at our graphite stack edge

Klaviyoâ€™s usage of statsd instrumentation started in 2015 according to our written lore (Slack channel history). This growth and the needs of a fast growing startup to continually make trade-offs between many high-value projects has meant that we have often neglected internal tooling such as Graphite until it becomes a problem impacting our developers productivity or our platform reliability.

Background for a StatsD + Graphite stack

StatsD, originally written by Etsy, is a metrics aggregator daemon as well as a basic text based line protocol using UDP. It is a push based architecture where you instrument your code with counters and timers and UDP metric packets are emitted when your code runs. As an aggregator, the statsd daemons goal is to accept data points within a time interval, aggregate the results, and flush them to a backend such as Graphite. The original version is written in Node.js although there are alternate server versions in numerous other languages.

Graphite is a larger project containing 3 major areas. These areas are the following:

- Graphite-Web, a Django application responsible for the presentation of already stored metric data. Graphite exposes direct time series values, transform views based on a rich library of functions, and rendering graphs as images.
- Carbon, a suite of daemons and a line protocol, used to enable the routing and persisting of metrics. The line protocol typically listens on TCP or UDP port 2003 and encodes metric values in either plain text or python pickles for batching. The carbon project provides a few other programs:
- Whisper, is a fixed size file based time series database. It has a defined storage schema for retention (for example retaining 15s resolution 1 week, 1 min resolution for the next 30 days, and 5 minutes for the next 60 days). Whisper has a defined storage aggregation configuration that defines how values roll up between retention windows (Such as to apply an average or sum for data points falling being rolled up into a single datapoint at a larger time resolution).

The carbon suite contains multiple programs which serve unique purposes

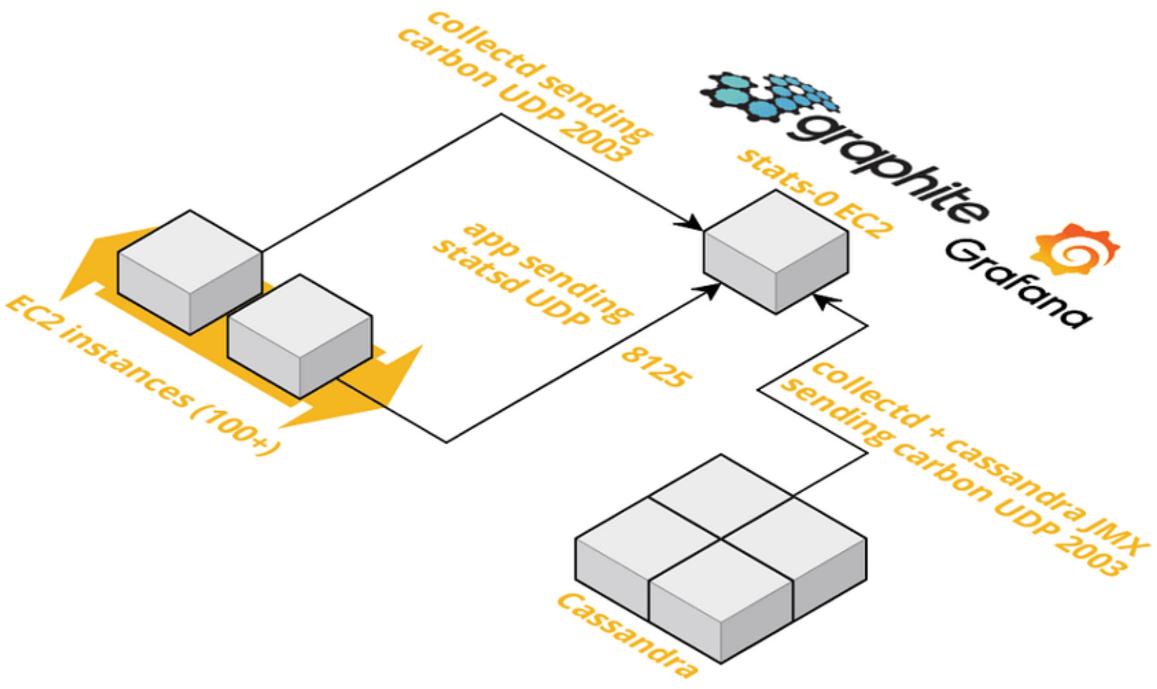
- carbon-relay.py is a router for the carbon protocol. Its behavior is based on rules written as regular expressions and is useful for providing a common endpoint for multiple backend carbon servers, duplicating metrics to multiple hosts for persistence and availability, or blacklisting metrics.
- carbon-cache.py is a write through cache layer for metrics data persisted into underlying whisper database files. It is a naive persister and does not do any aggregation, so multiple writes for the same time range overwrite each other. The cache component is implemented so reading metric values is fast and helps avoid disk IO.
- carbon-aggregator.py is an aggregator for multiple carbon metrics into a single one, as well as providing a buffer to reduce disk IO workload from carbon-cache.py. This is mostly useful for combining metrics together into a single measurement. For example: memory usage for multiple instances in a cluster could be aggregated for a common view.

Additionally, two other tools round out the stack:

- Collectd, a pluggable metrics collection daemon written in C. It typically runs on all hosts and pushes metrics to a configured backend. Collectd contains a graphite writer plugin which is used to emit metrics into the graphite stack via directly emitting metrics into the carbon daemon. Klaviyo relies on collectd for base system metrics as well as monitoring services via in house plugins we have developed in python for services like Proxysql, Postfix, nscd, and Apache Zookeeper.
- Grafana, is the leading frontend for metric analytics and monitoring. It is modular and supports many different data source backends, including but not limited to Graphite, MySQL, Prometheus and others. Klaviyo heavily uses Grafana for building and sharing over 350 dashboards across our teams.

Initial Implementation of a Graphite Stack

In the years before I joined Klaviyo in 2017, the company ran Graphite + StatsD + Grafana via a single EC2 instance. Since this was before my time, I searched Slack history for clues to when we started using Graphite. It appears that the single statsd instance existed essentially forever at Klaviyo with the first few messages, on March 31st 2015, in our generic #dev-team Slack channel from our CEO citing how to access Grafana when out of the office.



original single instance graphite stack

The original StatsD instance, of an unknown size at the time, was launched the same as most of our infrastructure at the time: an EC2 box that was launched via Boto and configured with Fabric.

This worked well enough for us to gain visibility and focus on building the product. However, as time moved forward we ran into a few problems.

Black Friday Cyber Monday 2016 Preparation

As an ecommerce company, the largest days of the year at Klaviyo are Black Friday and Cyber Monday (internally referred to as BFCM). Our platform typically handles a monthâ€™s worth of workload over the course of that weekend which stresses every component in our system. Our engineering team focuses heavily on scaling for this weekend in the months leading into Cyber Weekend. In preparation for the 2016 holiday season in October of 2016 we implemented a few improvements for our metrics visibility.

We had performed an instance upsize in early October from an m3.xlarge to a c3.2xlarge to provide more resource capacity and replaced the StatsD server from Etsyâ€™s StatsD (written in Node.js) to Githubâ€™s Brubeck (written in C). This server change allowed us to move away from a single CPU core bottleneck for the StatsD daemon. This was a more involved change since we had to fork brubeck and apply a open PR to support having statsd metrics namespaced.

Otherwise, the engineering team focused mainly on application visibility and increased the sampling rate of many metrics. A quote from chat at the time was

> Crashing statsd / graphite in the past has always been about too many metrics, not publishing to them too often

This also lead to us auditing which JMX metrics from Apache Cassandra were being emitted, which was was about 11,000 per instance. This reduced the number of metrics greatly by adding some regex whitelisting to metrics.yaml for Cassandra for what we cared about.

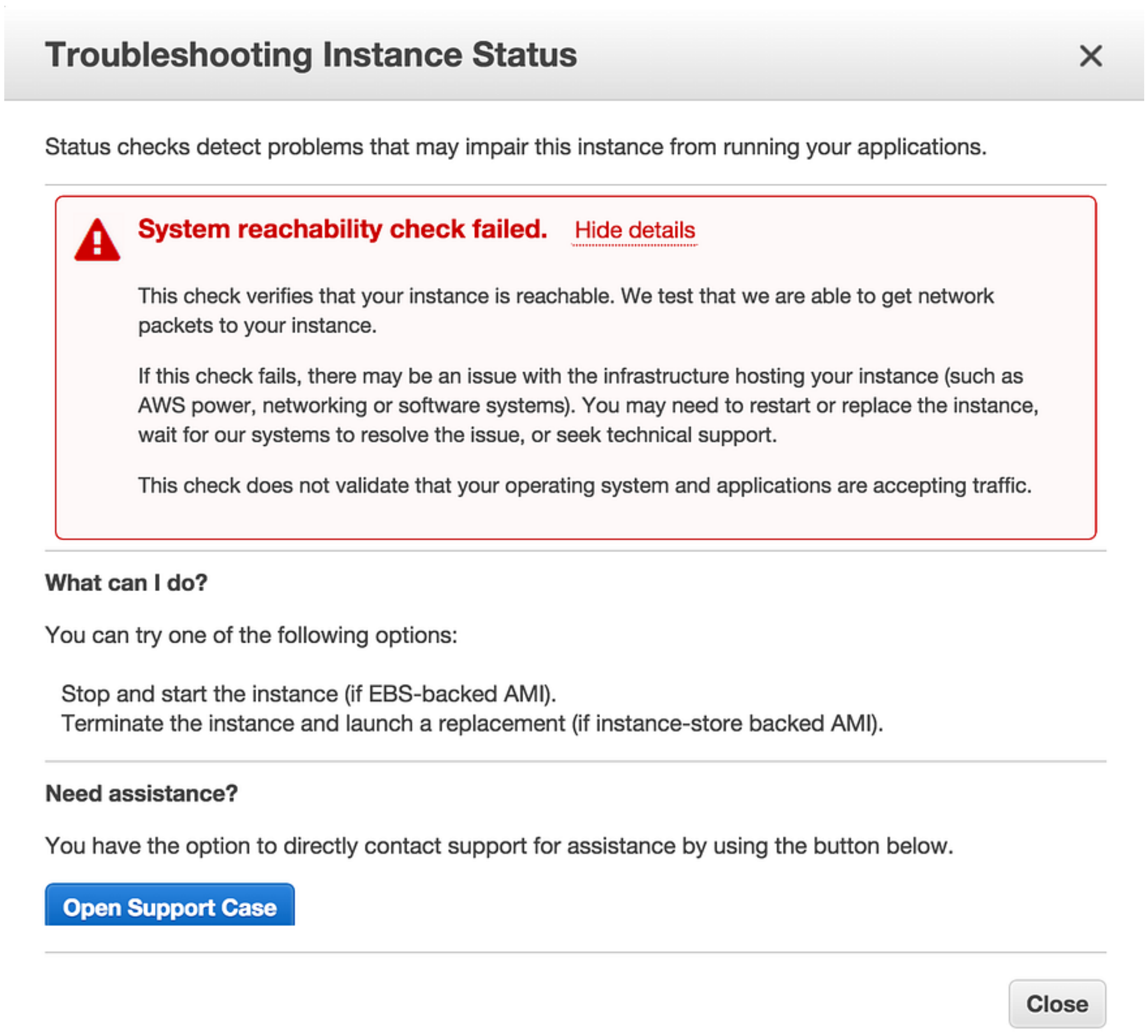
This bottleneck, the sheer number of whisper database files, was and remains a pain point for Klaviyo due to autoscaling. We heavily utilize AWS autoscaling groups for Celery instances and as a result, we launch and terminate thousands of instances a day. Our base collectd system metrics are namespaced by hostname, and our hostnames include the instance id. This results in a very large amount of whisper files for hosts that have been terminated. Our solution to this at the time was to have a cron job automatically remove whisper files for autoscaling workloads after a certain time period.

> our crontab for reaping whisper files

While its not elegant, it remains our solution to this day, and because we rely much more heavily on application instrumentation via statsd than we do for collectd metrics, itâ€™s acceptable to maintain a few days worth of collectd metrics for workloads that highly scale.

Black Friday Cyber Monday 2016 Problems

On Black Friday 2016 we had some problems with our instrumentation. After a long day breaking every record at Klaviyo for our systems the Graphite+Statsd instance became unresponsive. Attempts to reboot the instance failed due to apparent EBS block device mounting issues, which in hindsight was probably a bad mount point in /etc/fstab or corrupted EBS volume. Flying blind without metrics during the largest time of the year is not ideal. The impact of this was that:


Troubleshooting Instance Status

Status checks detect problems that may impair this instance from running your applications.

System reachability check failed.

This check verifies that your instance is reachable. We test that we are able to get network packets to your instance.

If this check fails, there may be an issue with the infrastructure hosting your instance (such as AWS power, networking or software systems). You may need to restart or replace the instance, wait for our systems to resolve the issue, or seek technical support.

This check does not validate that your operating system and applications are accepting traffic.

What can I do?

You can try one of the following options:

Stop and start the instance (if EBS-backed AMI).
Terminate the instance and launch a replacement (if instance-store backed AMI).

Need assistance?

You have the option to directly contact support for assistance by using the button below.

- Our team worked late to rebuild the instance and update the other instances to point to the new instanceâ€™s IP address
- The metrics data on the original instance were lost, which prevented comparison of key performance indicators for BFCM
- Cassandra visibility was lost entirely, because restarting the cluster to update the IP address took a full day, and due to the sensitive nature of Cyber Weekend, was not performed.

On Cyber Monday, the new Graphite instanceâ€™s infrastructure was stable, but we were dropping metric values as evidenced by inconsistent graphs and reported UDP packet receive errors from `netstat -su`. We were also fighting log files filling the disk on the graphite server due to a bug in the version we were running at the time, and could not do an upgrade during Cyber Monday.

At this point Klaviyo had learned a few key things about this stack that fall into two main categories: Visibility and Operational Burden.

Visibility is critical, and becomes the most unreliable when needed the most

- Engineering relied heavily on StatsD + Graphite + Grafana metrics, and losing visibility was unacceptable, especially on our biggest days

- StatsD metrics volume correlates directly with our platform workload and capacity planning needed to consider that
- The instance was dropping UDP StatsD packets at peak volume, when visibility is most important, as reported by instance level metrics during peak volume, resulting in metrics inaccuracies

Operational Burden

- We were running an outdated version of graphite-web (Ubuntu Trusty provided up to 0.9.12, with https://github.com/graphite-project/graphite-web/issues/608 fixing a bug that spammed our log files in 0.9.15, first available in Ubuntu Xenial).
- The legacy single EC2 instance is a single point of failure
- Replacing a failed EC2 instance was a 10+ step manual process involving manual AWS interaction to replace the instance, migrate the EBS volume, application configuration updates and deploy, notifying the entire engineering team, and restarting every Cassandra node (hours to days long).
- There were no backups for dashboards in Grafana, as the instance relied on a local SQLite database

Early 2017

When I joined Klaviyo in August 2017, Klaviyo had successfully tackled the graphite upgrade by rebuilding the instance on Ubuntu Xenial and converting it into a single node AWS Autoscaling Group (ASG). This addressed the short term concern with the filesystem filling up due to logs from an upstream bug, and greatly assisted with instance replacement. However these actions did not address the remaining issues:

- Performance bottlenecks in UDP network throughput and Disk IO
- Instance failures required large manual effort to update the endpoint address across our fleet, most crucially Cassandra due to the time required to restart the cluster
- A lack of backups for Grafana dashboards that engineering relied upon

There was also an additional problem that started in 2017: brand new metric keys that were added to our systems would take a long time, between hours to days, before being visible in Grafana. This impacted the engineering teamâ€™s ability to gain visibility and apply performance tuning to key areas of our platform in production.

During the ramp up to the holiday period, Klaviyo relied heavily on production measurements to assess performance characteristics. By measuring production weâ€™re able to see the real world performance characteristics of our platform with our production data sets. The downside to measuring production is that we will never see BFCM scale traffic prior to the big day. To tackle this, we annually perform synthetic load testing in development to help prepare us.

Black Friday Cyber Monday 2017 Load Testing

A big part of the ramp up for BFCM at Klaviyo is performing load tests, which Klaviyo will be describing in further detail in an upcoming blog post. In mid 2017, Klaviyo had invested a lot of engineering time into a reproducible infrastructure stack in our development account called ksin10 (Klaviyo stack in 10 minutes). Included in this stack was the same single instance StatsD Graphite layout that we ran in production.

Naturally, by having the same graphite layout as production, we were going to see the same performance concerns. About halfway through 2017 load testing, we were noticing our StatsD counters (for example, events processed / sec) did not make sense when compared with our RabbitMQ message rate for event processsing; or our other platform metrics. We then re-discovered that we were dropping statsd packets, and since statsd is UDP based we did not notice until we saw the load testing metric discrepancy. An interesting discovery from this load testing was that UDP packet loss occurs in two ways:

- UDP packet received errors, as reported by netstat within the instance, which we had noticed during 2016
- UDP packets silently dropping within EC2 due to reaching an undocumented maximum UDP packet/sec throughput that varies slightly for various EC2 instance types

The former failure is possible to tune for within the instance to an extent, but the latter is outside of our control and required infrastructure layout changes. These load testing issues lead to a rebuild of the entire stats infrastructure to address this concern as well as the other concerns learned from BFCM 2016.

Brubeck split-out

The first priority when rebuilding our graphite stack was to unblock the load testing efforts that were underway. While we could derive total system throughput for event processing from other metrics, any attempt to measure slower code paths was not trustworthy due to the packet loss.

To quickly resolve this issue for load testing, it meant splitting off the Brubeck (StatsD) daemon from the Graphite and Grafana instance and onto its own instances. From an infrastructure perspective, this was relatively simple: create a new AMI just for running brubeck and have the instances configure brubeckâ€™s carbon backend IP on startup as part of the EC2 user-data cloud init configuration, with the layout looking like this: