My Co-op with SMS Conversations

Author: Klaida Azizi

Claps: 158

Date: Jan 9

This blog post is about what I learned from my co-op project at Klaviyo during Fall 2022.

New Beginnings

I have a somewhat atypical background for a software engineer. My undergraduate degree is in Chemistry and I did three years of radiochemistry research before deciding I wanted to switch to tech. I enrolled in Northeasternâ€™s Align program (Masters in Computer Science for people with non-tech background). Towards the end of the program, I landed a Software Engineering co-op at Klaviyo, and joined the SMS Conversations team.

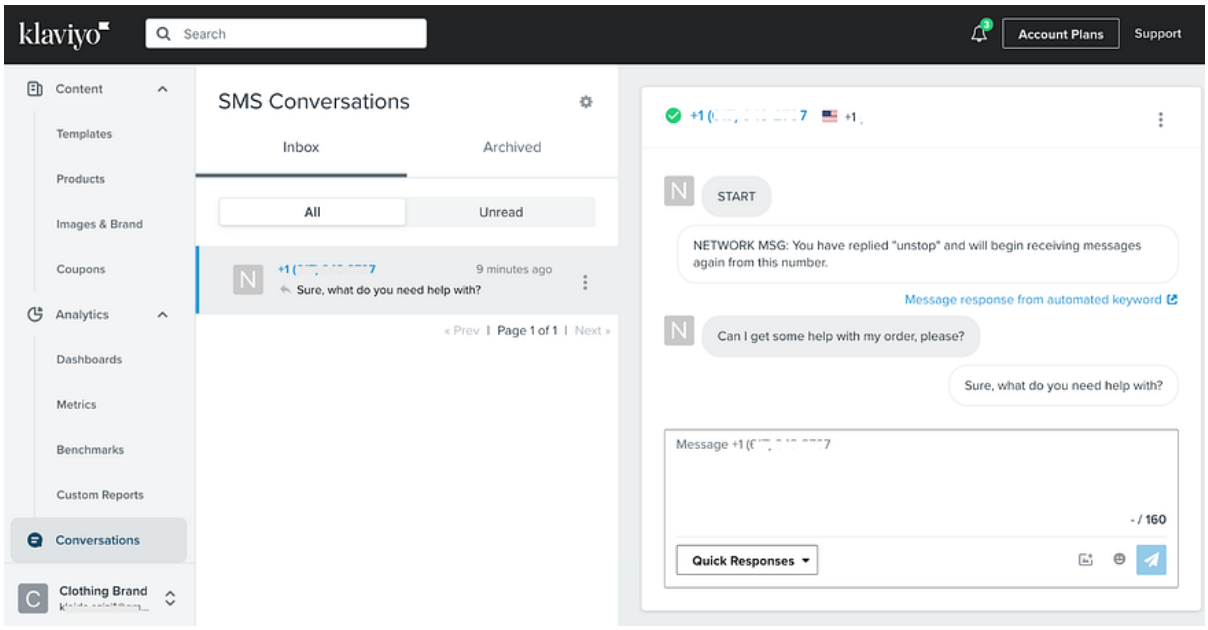
Considering my minimal tech experience before Klaviyo, I went into the co-op eager to get out of my comfort zone. I ended up responsible for building a load testing tool and running a series of load tests, something I had no prior experience with. Hereâ€™s what I learned.

Feature Background

Black Friday and Cyber Monday are the highest volume days of the year for Klaviyo as our customers promote their products via email and SMS. I learned that every year we do load testing to make sure our platform is ready to handle the heightened traffic. The team I joined, SMS Conversations, is responsible for our relatively new two-way text messaging feature. Our infrastructure processes inbounded messages from consumer to brand.

I was asked to write a tool to load test our inbound SMS pipeline by simulating inbound messages to our system. I started the project in July with the idea that we would begin load testing in September, giving us time to run repeat tests and address problems.

Inbound messages are categorized into seven main types: subscribe, unsubscribe, help, info, unrecognized, reaction, and custom keyword. As an example, suppose you as a consumer are interested in a brand. You text â€œSTARTâ€ to the brandâ€™s phone number. This is an example of a *subscribe* keyword. Later you receive a promotion for a product and have a question about it. You text your question, this is classified as an *unrecognized* message type, and a human is able to answer either through the Klaviyo UI or via integrated customer service software. Later, if you decide to leave the list, you send â€œSTOP,â€ an example of an *unsubscribe* keyword.

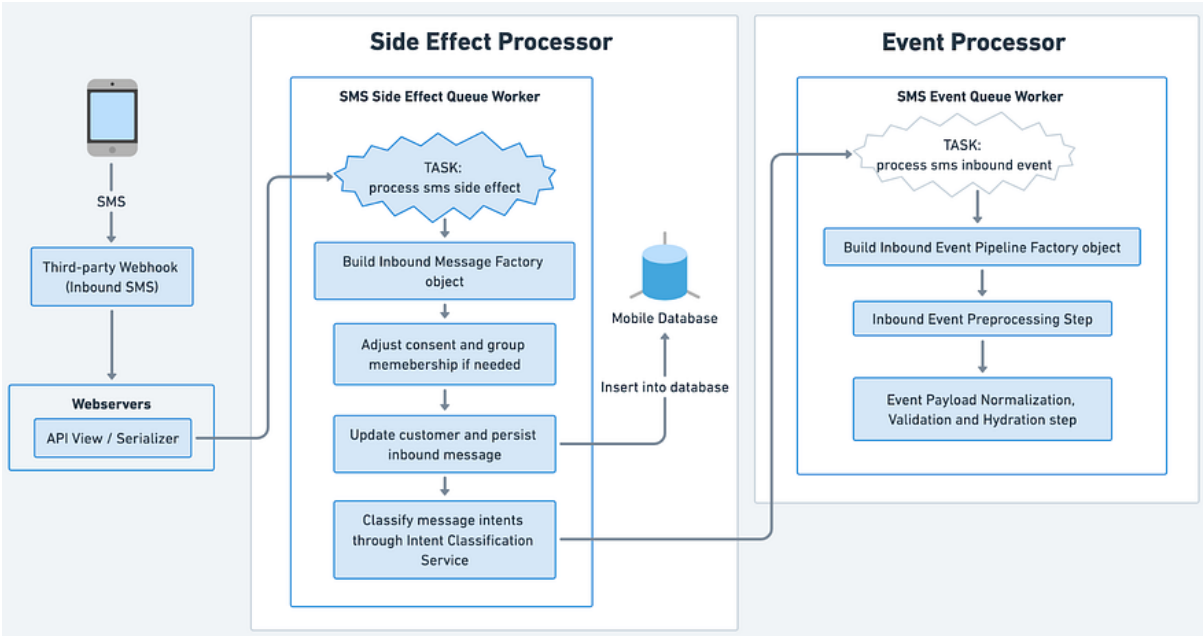


A customer/brand interaction through SMS Conversations

Our two-way messaging feature was new for BFCM 2021 when we saw a peak of 50,000 inbound messages per hour. When I started working on the load testing tool, regular traffic was peaking at 20,000 inbound messages per hour. The idea was that the load testing tool would simulate higher volumes of traffic higher than we expected for BFCM 2022, and do so with synthetic inbound messages payloads similar to real traffic. We wanted the test messages to be injected in such a way that we exercised as much of the production pipeline as possible.

Technical Background

We use third-party vendors to send and receive SMS messages. The third-party vendors communicate with our servers via a webhook, which is fired to indicate different events such as *received new inbound SMS.* After a new inbound message is received and validated, the rest of the inbound business logic is processed asynchronously. First, the message type is determined. If itâ€™s non-conversational, the processing pipeline triggers an automated response. Otherwise, our intent classification service helps to further classify different intentions, such as *coupon issue* (â€œMy coupon isnâ€™t working!â€) or *general love* (â€œI love your brand!â€). The important point for load testing is that different message types get processed differently.



Simplified diagram of the inbound SMS processing pipeline system

Working with my mentor, we decided our best entry point was to call the same webhook as the third-party vendors.

Load Test Script

The (good!) advice I received was to keep the first version simple. We followed a common pattern here and built a custom Django management command that we could manually run from what we call an on-demand. Hereâ€™s an example from the first version:

```
bin/django inbound_sms_loadtest --company-ids=WeGSit --message-count=12000
```

The only parameters were *company_ids*, the ids of the companies to receive the messages, and *message_count*, the number of synthetic inbound messages to send. We created fictional companies to be the message recipients, and added new controls in our production system to allow specified companies to be excluded from side effects such as default email forwarding of *unrecognized* messages. At a further step, in order to not overload third party vendors or get ourselves rate-limited, we scrutinized all the lookups that we do with third-party vendors (e.g. looking up phone numbers) and refactored the code so that we could add ways to skip these lookups for specified load test companies. I was already learning that there are gotchas even with simple load tests, and that itâ€™s not necessarily easy or appropriate to exercise every part of a pipeline.

We used the Python Faker library to generate payload data that would mimic inbound message payloads. Our initial version of the load test tool was a single threaded process that:

- Sent inbound messages from a single profile (sender phone number).
- Evenly distributed messages across all message types.
- Completed all sends for messages of one type before moving on to the next.

Load Testing

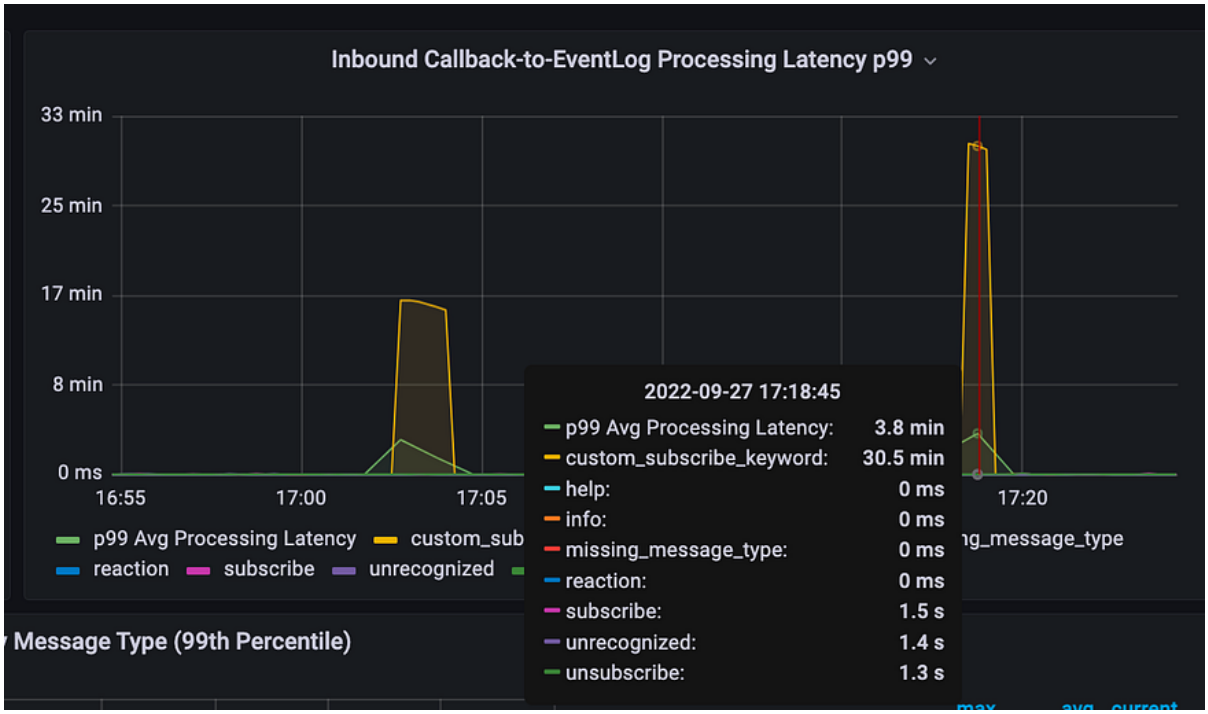
We scheduled our first load test at the end of September with the plan to form an initial idea of our system health two months ahead of Black Friday. For BFCM 2022, our product analytics team predicted a peak hour of around 120,000 inbound messages. Our aim was to work our way up to conducting a number of load tests above that volume before BFCM.

We needed to monitor what was going on in our system. I learned about Grafana and Sentry, both of which we use heavily at Klaviyo and were used during all load tests.

Load Test #1

The goal for this initial test was to hit a total of 12,000 calls to our inbound message event API, and assuming success, step up to another 36,000 calls by running the command in three parallel sessions.

This largely worked, but when we looked at Grafana, we discovered latency issues isolated to certain message types.



Grafana graph showing inbound message processing latency. Two main spikes were observed for custom keyword messages, one of 17 minutes and the other 30 minutes.

We investigated and determined that using a single test sender phone number and sending the same inbound message type in rapid succession triggered a race condition in downstream code. Our team decided this was not representative of production use. We took two actions:

- Opened a bug with the team that owned the code with the race condition.
- Adjusted the load test tool to randomize the order of messages types.

This first test opened my eyes to the subtlety of load testing. You might find problems in the system, you might find problems with the load testing process, and you need to apply judgment.

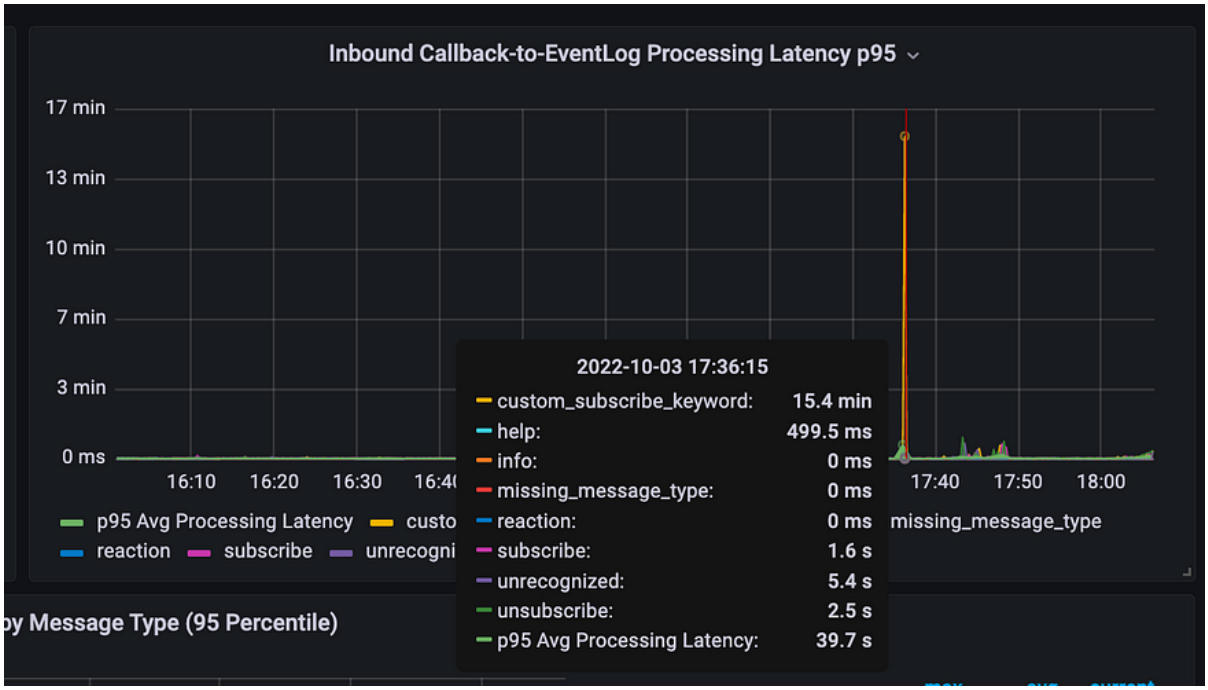
We introduced a flag to randomize the order of message types instead of sending a batch of one type followed by a batch of the next type. Example:

```
bin/django inbound_sms_loadtest --company-ids=WeGSit --message-count=20000
```

Load Test #2

The plan for this test was to run three load test sessions of 20,000 inbound messages each in a stepwise pattern within an hour, so once 7,000 messages were generated from the first one, the second 20,000 messages would be run, and so on. We wanted to balance testing up to half our predicted traffic without overwhelming dependent systems.

Randomizing the order of message types fixed part of the latency issue identified above. But not the whole thingâ€¦



Grafana graph showing inbound message processing latency. A spike of 15 minutes latency was observed for custom-keyword messages.

The custom subscribe keyword latency issues still appeared and a Sentry (exception) that seemed to be related to the latency was also fired off. We investigated and believed this was still related to the race condition identified above and it would largely go away with traffic originating from different sender phone numbers. We made two decisions:

- Run another test right away where we simply excluded custom keyword messages, knowing we could later expand our load testing to support multiple sender numbers.
- Increase load to the predicted BFCM amount to give other problems a chance to surface.

We introduced a parameter to skip certain message types. Example:

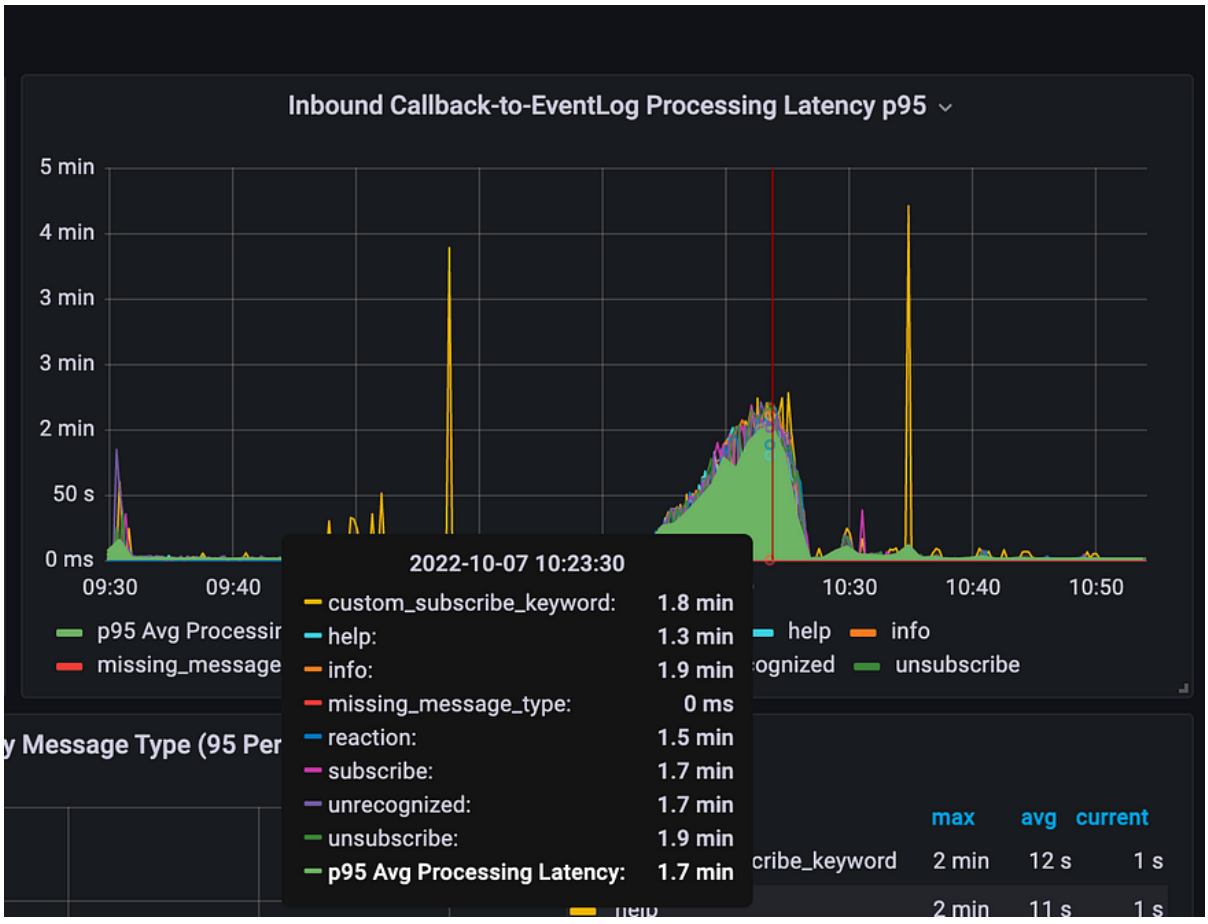
```
bin/django inbound_sms_loadtest --company-ids=WeGSit --message-count=30000
```

Load Test #3

This one was important because we wanted to see what, if anything, would break when we shifted to our BFCM prediction of 120,000 inbound messages in an hour. We ran four parallel sessions of 30,000 messages each.

The test was successful overall. The biggest error we saw this time around was increased latency caused by our intent classification service not keeping up. We worked with the Data Science team who determined the root caused and solved it.

We also noticed additional latency across all message types. Here we determined three culprits: subscription calls made for subscribe keywords, sustained CPU spikes, and database write latency.



Grafana graph showing inbound message processing latency. Spikes of around 1.5 minutes were observed for all message types and around 4 minutes for custom-keyword messages.

We gave ourselves the following actions items:

- Enhance the load test script to support messages from multiple sender numbers. (The subscription issue mentioned above was likely due to all these subscribe keywords being for the same â€œperson.â€)
- Scale out our side effects and intent classification boxes before each load test in order to handle the high volume. (Which would in fact be the configuration on BFCM.)
- Ask the team responsible for subscriptions to investigate the subscription call latencies. (To confirm or reject our theory that it was due not to a general problem but because we were trying to subscribe the same fake person over and over.)

We enhanced the load testing script to support multiple sender numbers. The way this worked was to specify a group (a Klavyo concept, like a list of people), and the script would use everyone in that group. We also moved from single-threaded to multithreaded since most of the scriptâ€™s running time was waiting on API responses.

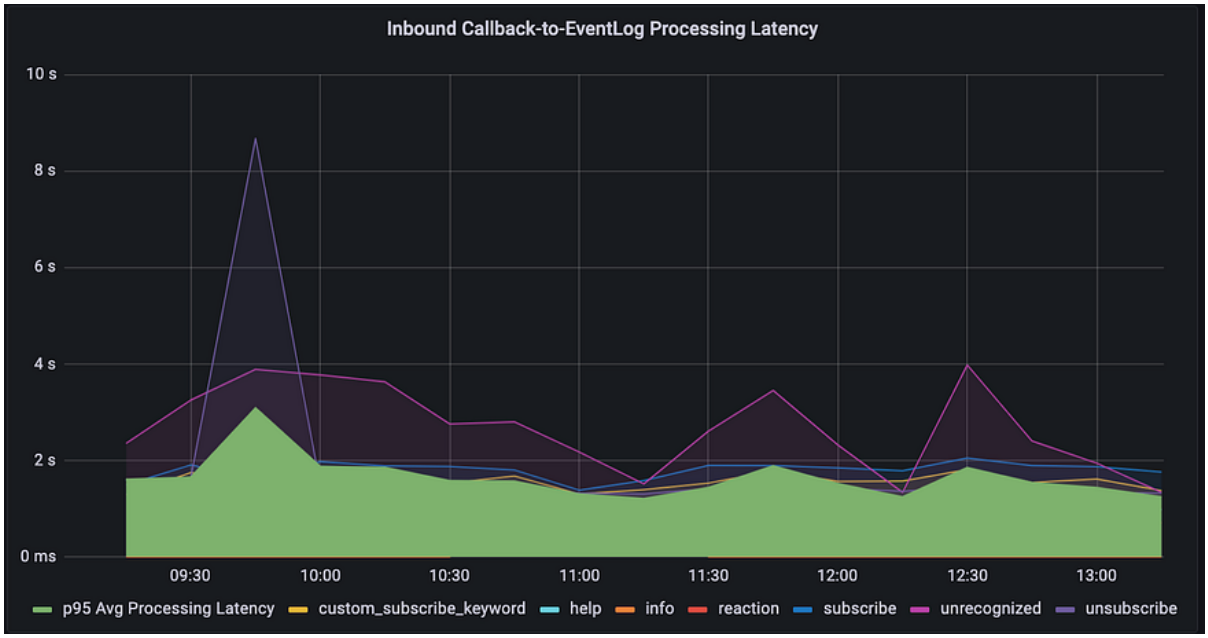
Hereâ€™s an example. You can see the parameters for group identifier and number of parallel worker threads.

```
bin/django inbound_sms_loadtest --company-ids=Tz3JZ8 --group-id=TfYtSr --m
```

Load Test #4

This load test used the tool with all the enhancements described above including multiple sender numbers, and was performed after scale-out of the clusters responsible for side effects and intent classification.

The test successfully simulated 160,000 inbound messages. It surpassed the volume of inbound messages predicted for BFCM 2022. We stopped seeing Sentries (exceptions) and there were no concerning CPU spikes.

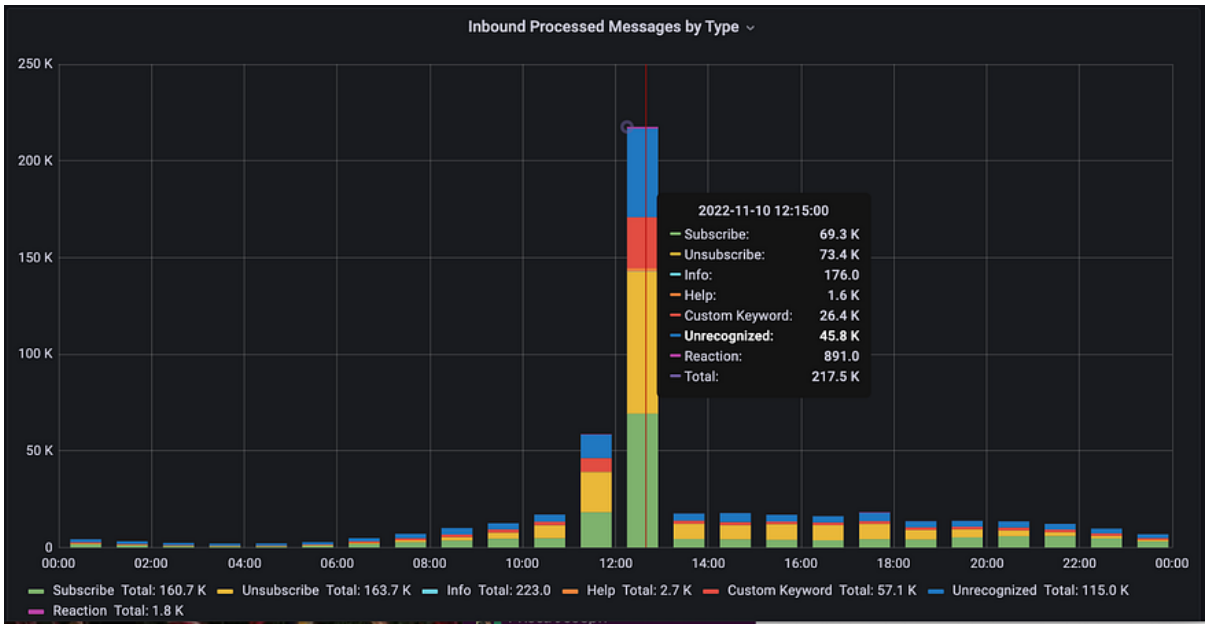


Grafana graph showing inbound message processing latency. No significant latency spikes were observed.

We did not have any action items other than to test with even greater volume.

Load Test #5

We tested 210,000 inbound messages with systems scaled out. The test was successful and we saw a max throughput of 211 messages per second. We saw no latency spikes and no Sentries. This was twice our predicted BFCM volume which left us confident we could handle BFCM.

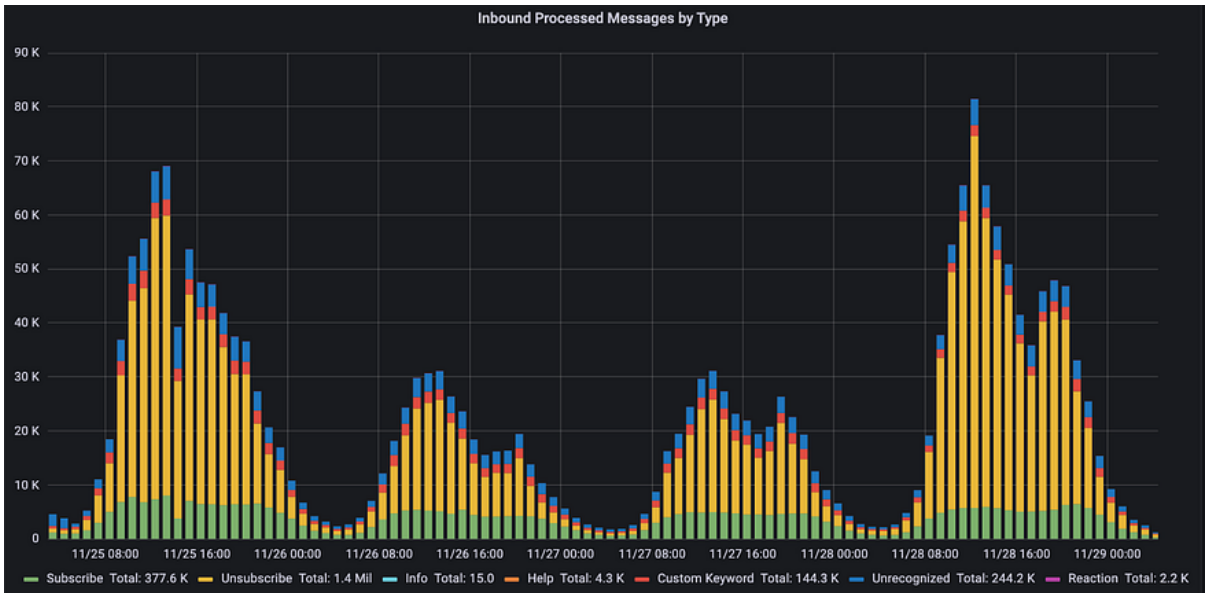


Grafana graph showing the total number of inbound messages received within an hour. An hourly peak of 217,500 inbound messages was obtained as a result of load test plus production traffic.

BFCM 2022

Here were the hours we saw with the most actual traffic over BFCM 2022:

1. Cyber Monday at 12:00â€“13:00: 81,564 inbound messages and max throughput of 52 messages/sec
2. Black Friday at 13:00â€“14:00: 69,013 messages and max throughput of 50 messages/sec
3. Black Friday at 12:00â€“13:00: 68,101 messages and a max throughput of 29 messages/sec



Grafana graph showing the distribution of total inbound messages received from Black Friday to Cyber Monday.

The overall number of inbound messages received for the 5 day period was 2,826,165. Our system successfully handled these inbound messages with no major latency or other issues.

What I Learned

Here were the takeaways from my co-op with Klaviyo:

Technical and Process

- **â€œSimplicity â€” the art of maximizing the amount of work not done â€” is essential.â€** This is one of the 12 principles of the Agile methodology that our SMS Conversations team proudly follows. Even though we suspected we would need to simulate traffic that was similar to production, we started with the bare minimum, and let our learnings guide what to do next. Throughout my entire co-op, Iâ€™ve noticed that we, as a company, stay true to our CEOâ€™s favorite motto: â€œ*We are only 1% done*.â€ Weâ€™re always looking to improve.
- **Team and cross-team collaboration.** This was crucial to getting the load test done on time for BFCM. The load testing tool was included in our team sprint and engineers were quick to help with tickets, treating it as a team project. As described above, we needed to reach out to other teams too, and they in turn quickly picked up assigned tickets and resolved issues, recognizing that our work was also serving to load test their systems.
- **Distinction between load test limitations and system bottlenecks.** Throughout load testing, it was important to have a clear distinction between issues with the load testing process and issues due to inadequate system performance. This was challenging because it wasnâ€™t always clear, so team effort went into recognizing which category each problem fell into.
- **Testing coordination and cadence management.** We scheduled a regular cadence of weekly load tests. This gave us enough time to work on action items between tests. As importantly, it let us add features one at a time, in order to change as few variables as possible between tests.

Personal

- **Take advantage of new opportunities.** Being new at Klaviyo and in the tech industry, I had a lot of on the job learning. Looking back on my first days, I felt inspired because as a co-op, nobody made me feel like I couldnâ€™t do this, and the team had full faith in my ability to learn. This made me feel enthusiastic about the project and willing to tackle it.
- **Be conscious of roadblocks.** I hit challenges with my local dev environment in the beginning of my co-op. I was new to Klaviyo and to our software and couldnâ€™t solve them. I was frustrated and wanted to move on to real work. My manager had some useful advice: Try to solve an issue by yourself, but if after twenty minutes you canâ€™t get unblocked, then reach out for help.
- **Itâ€™s okay to ask questions.** I felt very comfortable asking questions whenever I was confused about a concept or part of the code, and everyone was good about responding. The team encouraged this, and my mentor also set up working sessions so I could understand the architecture of our system.
- **Be a team player.** Through numerous one-on-ones with almost everybody on my team, I realized that the best way forward was together. Not only did I get acquainted with my co-workers and learn how to efficiently collaborate, but the project itself was improved by the contribution of multiple people.
- **You can make a big impact no matter your position.** As a co-op, I didnâ€™t expect to be working on a brand new tool from scratch, or to be given responsibility for how to go about it. In the end, the tool and the load testing made our team confident going into BFCM. I appreciated getting to contribute to something important.