Building SMS at Klaviyo

Author: Noah Durell

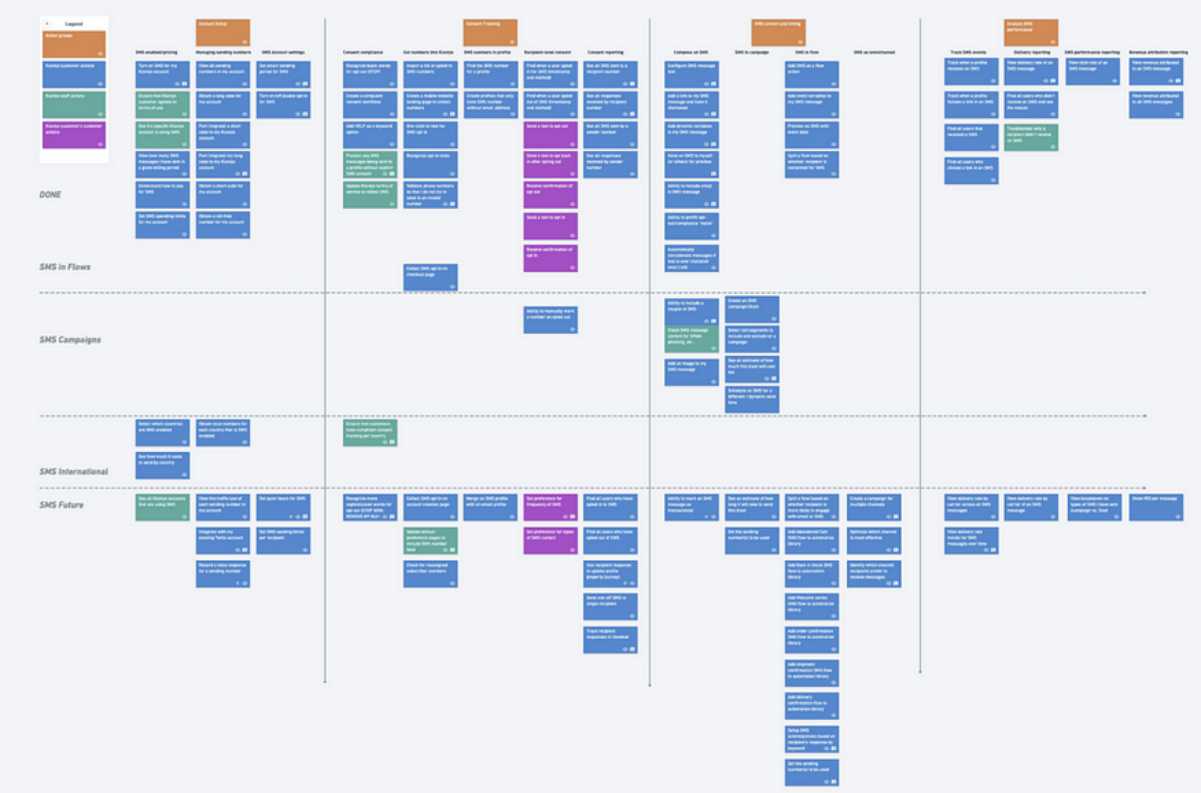
Claps: 230

Date: Oct 17, 2020

I joined Klaviyo back in January of 2019 as a senior software engineer. At that time, Klaviyo did not really have a Mobile team at all. There was a product manager, Andrea, and a designer, Gui, speccing out some work, and another engineer, Chris, revamping some parts of our Push Notification pipeline. Klaviyo had a mobile app and iOS SDK that hadnâ€™t been updated in several years. I had just come off a project at another company where I led a rebuild of an iOS app for online notarizations. I knew there was a lot of work to do at Klaviyo but I donâ€™t think I ever could have predicted how things ended up. All I can say now is Iâ€™ve had a blast working on what Iâ€™ve done so far and am very excited to see what the team weâ€™ve built (now 8 people!) is going to do in the coming year. In this post Iâ€™d like to talk about some of the work we did to build out Klaviyoâ€™s SMS product.

SMS Beginnings

When I first heard about the SMS project, Iâ€™ll admit I was a little surprised. As I mentioned above, my expectation was that I was going to help rebuild Klaviyoâ€™s mobile app and SDKs. Donâ€™t get me wrong, SMS seemed like a really fun project, but also it felt a bit out of my comfort zone. From what I could tell most of the work would be changes to Klaviyoâ€™s backend system and to the frontend web app. Also my initial interpretation of the project was that it was an experiment or sort of POC to allow a few customers to send transactional SMS from a Klaviyo Flow. As the project started to evolve, I could see that it was much bigger than I realized. I remember one day getting this diagram outlining the SMS Project from our PM, Andrea, and seeing that the scope of the project had grown dramatically.



The Original SMS Roadmap Diagram.

The initial plan had the team moving through a series of milestones of increasing complexity. For example our first milestone was simply to produce enough code to trigger a single static SMS to a specific number. Later goals included triggering an SMS from a Klaviyo flow or being able to create an SMS campaign. There were also milestones about creating facilities for billing customers and managing consent. It was definitely cool to see the entire project laid out for me. I was also impressed at how organized the plan was. It was clear the product team had done their research. This was comforting because at other companies Iâ€™ve been on projects that were seen as â€œcriticalâ€ for the company, but only later after finishing it I would find out that there was no real demand for it. To me nothing is more discouraging than finding you spent a lot of time on energy on something that no one needs or wants.

Vendor Evaluation

To kick the SMS project off, we started by looking at different vendors we could use to relay SMS from Klaviyo systems to handsets. This is similar to the approach we take with Email and we felt it would allow us to complete the project more quickly and abstract away the complexities of dealing with different SMS carriers. Below were some of the criteria we used to evaluate the vendors:

- Should be reliable and able to scale to meet our needs.
- API should be easy to use and provide a Python SDK
- Support sub accounts or at least some mechanism to segment our customersâ€™ traffic.
- Provide debugging and monitoring tools so we could trace issues back to the vendor, if necessary
- Support sending SMS/MMS over short code, long code, and toll free numbers.
- Requires minimal human intervention to get things done, like acquiring sending numbers, or if we have to do large SMS sends on short notice.

Though we met with several vendors, we were able to quickly narrow it down to 2 because many of them could not support all the requirements above. Once the final contenders were identified, we built some simple Django management commands to try sending SMS. This also gave us an excuse to prank our co-workers with funny animated gifs, though probably not all of them appreciated it. After doing more in-depth analysis on each of the vendors and having several meetings with them, we were able to decide on a vendor. One of the biggest factors for us was that we felt the APIs of one of the alternate vendors would be too cumbersome for us to deal with. Specifically, we felt setting up sub accounts, though doable, to be very confusing with the alternate.

Building an SMS Service Layer

Now that we had decided on a vendor, we were able to get our first big SMS projects underway. We started by building out some of the initial data models and also spinning up our own database instance to hold this data. At Klaviyo, getting a new database in production is actually pretty simple. Although we hadnâ€™t used it much before this, We were able to write some simple Terraform configuration to spin up an Aurora MySQL database in AWS. This was cool because we knew the new database would have all the recommended configurations we like with our other database instances. Next, we created several Django models to store SMS customer configuration information to information about their sub accounts and their sending phone numbers. The plan was to eventually allow customers to allocate numbers for their account. For some types of numbers (like short codes) we expected manual intervention would be involved. So a change request model was added that would be used to monitor the status of a phone number allocation.

Having worked with a lot of different vendors in the past, one thing I learned was it is sometimes necessary to swap them out. Even though in this case we were pretty confident about this particular vendor, we decided to create a generic interface that actually made the API calls to the vendor. Our services would call into that interface to do things like add phone numbers, send messages, and perform other operations. We called this the SMSClient interface, which was an abstract class. The implementation of this interface simply wrapped calls to the vendorâ€™s python SDK.

Link Shortening

Since customers would be paying by the message segment, which was based on character count, there was a need to shorten links within messages to keep segment count small. Unfortunately this was not a service provided by our vendor (or any vendor we evaluated), so we decided to roll our own link shortening service. An added benefit of doing this meant we could attribute clicks on our customerâ€™s SMS messages to later conversion events. We used this opportunity to learn about Klaviyoâ€™s microservice automations, specifically Kubernetes (running on EKS in this case).

Klaviyo already had some templates for creating services on EKS. We were able to quickly spin up a new service running on Python 3 and Django 2 (which we affectionately named lil-clicky). The service added two endpoints: one for shortening links and another for redirecting shortened links to their original url. The shortening endpoints take the original message (along with some metadata) and shorten all the links it finds in them. When a user clicks on the shortened link, it gets routed to and endpoint in the lil-clicky service which reads the code from the link and looks up the redirect. Lil-clicky also logs the click back into the Klaviyo event processing system so it can be used for attribution later.

SMS Consent

In addition to the models we added for account settings, we also created a model to store consent for SMS. Consent is needed before our customers can send messages to their customers. This one ended up being a little more controversial than we initially anticipated. Traditionally, information like this has been managed by the Reporting, Segmentation, and Profiles (RSP) team at Klaviyo, but at the time they were busy with other projects. Since we had to store additional information about the consent but felt the existing datastores could not handle this additional data, we added a new table in the new MySQL database we created earlier to store this data. This too was wrapped in a service that allowed callers to perform CRUD operations on profile consents. Other than some changes to the underlying models this service has not changed too much since it was created.

Updating And Building New UIs

For the SMS project we would initially need two new and updated UIs. One that would enable users to modify the content of their SMS and one that would enable users to configure their SMS settings. To achieve the UI for updating SMS content we were able to reuse some of work we had recently built for configuring push notifications. The UI would essentially detect whether it was dealing with an SMS message or a push notification and adjust the display accordingly. It would also add some additional functionality for shortening links, adding the company prefix and the opt out language.

For the SMS settings page, everything had to be written from scratch. We added a new package in our frontend monorepo that would hold all our new settings cards. Each card had a specific purpose, like adding and removing a phone number, controlling settings for how SMS attribution and smart sending work, modifying default SMS prefixes and opt out language, and controlling keywords. Each of the cards required new CRUD APIs, which we made using Django Rest Framework (DRF). This gives us access to convenient serializers to ensure request payloads are correct. It also makes it easier to format request responses back to the client. On the frontend we used redux to consolidate a lot of the state for the page. We load all the settings data when the page and it trickles down via redux to each of the cards. Similarly, when we can update state from a single endpoint that affects the display of other settings cards. For example, when the user sets up their SMS account for the first time.

Integrating with the Flows Codebase

For the initial release, the plan was to be able to send SMS from Klaviyo Flows like email and push notifications. During this time the mobile team had grown by one, with the addition of our summer intern, Vittoria. Since we were treading over some well worn paths, we were actually able to do a lot of this work. Flows are configured by connecting conditional blocks and actions together. When the conditions are correct (or evaluated), the action connected to it is fired. To get an SMS to fire we created an SMSAction and SMS flow message. When the action is fired, the flows code loads up the message and provides the context to render it. This required creating a new table to store all the messages including the message body and settings.

Flow push notifications had previously run on the same hardware that sends emails. For Flow SMS, we decided to take the opportunity to split out this work onto separate nodes. This had also been a request of the Email team for a long time. To do this, we spun up both evaluation and sending queue worker nodes (in EC2). Again, this was very easy to do using Terraform by adding a few entries to our queue worker configuration file. Separate celery tasks were created as well to

run on our new nodes. Then these tasks would simply call into the SMS sending services we had developed, which in turn called out to our vendor.

In addition to these, there were many places we had to change to account for the new message channel. This included UI changes to support adding SMS Actions to flows. Updating the UI that shows analytics around flow sends (e.g. how many SMS were sent, how many clicked). We also had to account for Flow SMS sends being skipped or failed. Flows has a convenient way of handling this by mapping exceptions thrown during sending to skipped reasons. The skipped reasons then gets translated into human readable strings for the customer.

Handling Incoming SMS and Events

One of the final pieces we put together for the initial SMS Flows project was the event processing pipeline. For this project we were able to collaborate with the Events team. That team had a comprehensive set of classes we could extend to build out the pipeline. Most events related to SMS come from webhooks that are called by our vendor with a JSON payload. The payload came in two flavors: outbound (SMS Received, SMS Clicked, SMS Failed, etcâ€¦) and inbound (SMS Sent, i.e. when an end user sends an incoming message). The webhook does some minimal validation on the payload and then passes it off to a celery task which processes the payload. Payload processing involves several steps including:

- Validation â€” additional validation of the payload to ensure it has the expected data
- Hydration â€” loading data related to the payload
- Normalization â€” mapping data and values to our internal representations
- Consent Updates â€” determining if the payload should update consent
- Attribution â€” determining if the payload will affect attribution
- Event log â€” writing the event to our event log database
- Timeline Update â€” writing the event to the customer timeline.

Although it appeared complicated at first it was fairly straightforward to wire-up. Also once this was done, we would have nice stats on how often the steps happened and how long they took. With this last piece in place customers could now get insight into how their Flow SMS performed.

In addition to the event processing above, the pipeline would also handle responding to inbound SMS from end users. This meant analyzing the consents on the message and determining an appropriate response. For example if a user texts in STOP, we interpret that to mean they are opting out and we send them a message back letting them know as much. This code was later moved to a separate task to decrease the complexity of the pipeline.

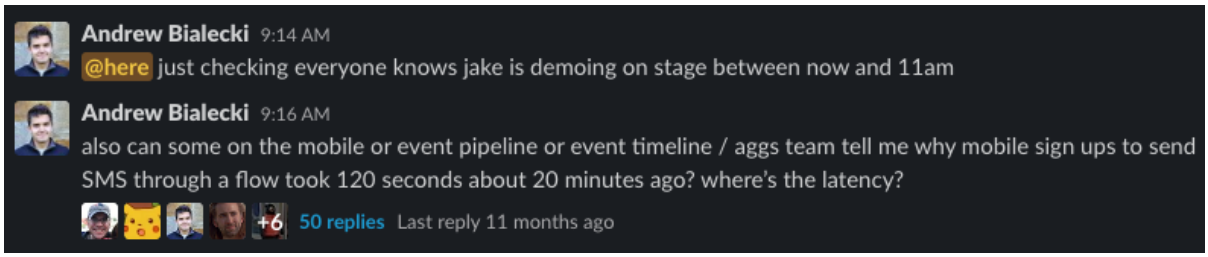
The Road to Klaviyo:BOS

At this point the Mobile team had shown they could deliver on the milestones for SMS. We had done several internal demos of the projects described above. Klaviyo hosts an annual conference called Klaviyo:BOS where customers from around the world come to learn about Klaviyo and new features built by the team leading up to Black Friday/Cyber Monday (BFCM). Given the state of our project, we decided to showcase SMS in a live demo during the conference Keynote. The plan was to set up an account ahead of time configured with SMS. During the keynote, we would post a link to a page with our sign up for so the audience could opt in for SMS. The new special Klaviyo Form would open up a messaging app on a mobile device with a preconfigured JOIN message. Once the audience sent, these would trigger a consent event which, in turn, would trigger a flow to send a message back to the audience. That message would have a link the users

could click which would demonstrate link tracking on SMS Sends. It would be one of the biggest load tests we have done so far with SMS. During the test we would also show how stats updated as texts came in, messages were sent out, and links were clicked. Although we were excited it was a bit nerve wracking to have delivered all this by a set date in front of an audience of 800 users!.

Fortunately by this time we had hired an additional teammate, Jordan, to help us get ready. He was able to quickly onboard and start attacking bugs. Probably the biggest thing we had to deliver was the new Klaviyo form that would allow users to click to consent. Part of the challenge there was to craft a link that would open up on every phone (both Android and iOS). Once we had a working link structure, we worked with the Content team to build out the form. Luckily this was a newer codebase and was fairly straightforward to get working. Next, although we had some basic keyword handling in the event pipeline, this demo would require some more advanced features there. After a few iterations, this too was ready. Finally we made it such that Flows could be triggered by consent events.

Now that the pieces were in place we tried doing some bigger tests with lots of Klaviyo team members to make sure the system wouldnâ€™t fall over. In general these tests actually went ok. There were bugs here and there, but we fixed them quickly. As the date of the conference approached we increased the size of the tests, we even gave a â€œlunch and learnâ€, where we asked the entire company to send in texts all at once. So far everything was workingâ€¦until a few days before the conference. During a rehearsal at the conference venue, some folks noticed there were delays with the flows being triggered. It would happen randomly and it wasnâ€™t clear what was going wrong. There was talk of modifying or even canceling the demo. On the morning of the keynote demo, our CEO, Andrew Beliecki, put together a bigger test and he hit the bug.



Screen capture of Slack discussion the day of our SMS demo at Klaviyo: BOS

There was concern this would impact the efficacy of the demo and there wasnâ€™t much time to fix it. When we arrived on the scene, engineers all around the company were trying to figure out what was happening. After tracing through all the pathways, they were able to conclude that flow latency is a function of triggering/evaluation queue depth and that by avoiding a time of high utilization we were able to minimize the impact to the demo.

Despite the drama of the morning though everything worked out very well at the Keynote. People were impressed how quickly they could set up with SMS and configure flows. I also didnâ€™t hear anything about the latency. Our product manager, Andrea also held a more in-depth session on SMS that was very well attended. There she shared the plans for when it would all be available for customers. For me, it was very exciting to see that so many people were interested in the work we had spent almost 5 months on.

6:02 5GE

Shopify Plus Community

Invite Invite Inv

See All Friends

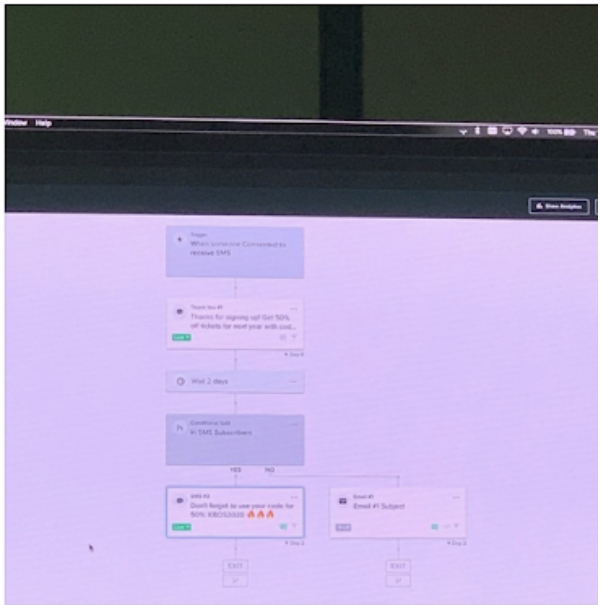
Miguel Madrid is at **Boston Convention & Exhibition Center.**

7 hrs · Boston, MA ·

Huge announcement from Klaviyo: SMS

Just saw a live demo, and it's just what you'd expect to see on a Klaviyo flow. Email and SMS are finally living together on the same platform. They also announced customer analytics with some crazy query capabilities.

Coming this October 🔥🔥🔥🔥



64 37 Comments

Releasing SMS and Future Work

After Klaviyo:BOS, momentum on the project really picked up and everything felt like a blur. Content team was generous enough to loan us one of their engineers and we quickly released more features so that customers could start to use SMS in Flows. This included attribution, message pricing and metering, phone number consent collection in Klaviyo forms, keyword management, integration with stripe for phone number subscriptions, and balance auto-refill. With these features we were able to have a limited set of customers start using SMS on November 12th, 2019 (before BFCM!). Afterwards we had an awesome team dinner and took this picture.

Mobile Team Dinner Celebrating SMS GA

Since that time, we have continued to round out the SMS offering. We now support campaigns, support for mms, dynamic coupons, and we are laser focused on deliverability. Thereâ€™s also lots more to do in the coming months and years with SMS including international support, two way messaging, and improving our compliance systems. If itâ€™s not clear already, Klaviyo is a great place to build impactful products. As part of the Mobile team, you get to collaborate across the engineering organization and beyond with the whole company. Even though at first I was nervous that I wasnâ€™t the right person for these tasks, I now realize the importance of working on the right project rather than pigeon-holing myself into particular technologies. If this sounds like an interesting project to you, drop me a line sometime!