How Benefit Cosmetics Uses Serverless

Jason Collingwood @ Benefit
November 21, 2018

Founded by twin sisters in San Francisco well before the city became the focal point of tech, Benefit has been a refreshing and innovative answer to cosmetics customers for over 40 years. The company is a major player in this competitive industry, with a presence at over 2,000 counters in more than 30 countries and online. In recent years, Benefit has undergone a swift digital transformation, with a popular eCommerce site in addition to their brick-and-mortar stores.

When I started with Benefit, the dev team's priority was to resolve performance issues across our stack. After some quick successes, the scope opened up to include exploring how we could improve offline business processes, as well. We started with our product scorecard, which involved measuring:

- In-site search result ranking.
- Product placement and mentions across home and landing pages.
- How high we appeared within a given category.

We needed to capture all this information on several different sites and in a dozen different markets. If you can believe it, we'd been living in a chaotic, manually updated spreadsheet and wasting thousands of hours per year gathering this information. There had to be a better way.

Automating Applications

To monitor a large number of sites in real time, a few SaaS options exist, but the costs can be hard to justify. Moreover, most solutions are aimed at end-to-end testing and don't offer the kind of customization we needed. With our needs so well-defined it wasn't very much work to write our own web scraper and determine the direction we needed to take.
The huge number of pages to load, though, meant that scaling horizontally was a must. Checking thousands of pages synchronously could take multiple days, which just wasn't going to cut it when we needed daily reports!

"Well, let's look into this serverless thing."

Web monitors and testers are a classic case for serverless. The service needs to be independent of our other infrastructure, run regularly, and NOT be anyone's full-time job to manage! We didn't have the time nor people to spend countless hours configuring resources- and really didn't want to be patching servers to keep it running a year in the future.

How it Works

We use Selenium and a headless Chrome driver to load our pages and write the results to a DynamoDB table. Initially, we tried to use PhantomJS but ran into problems when some of the sites we needed to measure couldn't connect correctly. Unfortunately, we found ourselves confronted with a lof of "SSL Handshake Failed" and other common connection timeout/connection refused request errors.
The hardest part of switching to the ChromeDriver instead of PhantomJS is that it's a larger package, and the max size for an AWS Lambda's code package is 50 mb. We had to do quite a bit of work to get our function, with all its dependencies, down under the size limit.

The Trouble of Complexity

At this point, even though we now had a working Lambda, we weren't completely out of the woods. Hooking up all the other services proved to be a real challenge. We needed our Lambdas to connect to DynamoDB, multiple S3 buckets, Kinesis streams, and an API Gateway endpoint. Then, in order to scale we needed to be able to build the same stack multiple times. The Serverless Application Model (SAM) offers some relief from rebuilding and configuring stacks by hand in the AWS console, but the YAML syntax and the specifics of the AWS implementation make it pretty difficult to use freehand. For example, a timer to periodically trigger a Lambda is not a top-level element nor is it a direct child of the Lambda. Rather, it's a 'rule' on a Lambda. There are no examples of this in the AWS SAM documentation.
At one point, we were so frustrated that we gave up and manually zipped up the package and uploaded via the AWS Console UI… at every change to our Lambdas! Scaling a lot of AWS services is simple, but we needed help to come up with a deployment and management process that could scale.

How Stackery Helps

It's no surprise that when people first see the Stackery Operations Console, they assume it's just a tool for diagramming AWS stacks. Connecting a Lambda to DynamoDB involves half a dozen menus on the AWS console, but Stackery makes it as easy as drawing a line.
Stackery outputs SAM YAML, meaning we don't have to write it ourselves, and the changes show up as commits to our code repository so we can learn from the edits that Stackery makes. It was very difficult to run a service even as simple as ours from scratch and now it's hard to imagine ever doing it without Stackery. But if we ever did stop using the service, it's nice to

know that all of our stacks are stored in our repositories, along with the SAM YAML I would need to deploy those stacks via CloudFront.

Results

With the headaches of managing the infrastructure out of the way, it meant we could focus our efforts on the product and new features. Within a few months were able to offload maintenance of the tool to a contractor. A simple request a few times a day starts the scanning/scraping process and the only updates needed are to the CSS selectors used to find pertinent elements.
Lastly, since we're using all of these services on AWS, there's no need to setup extra monitoring tools, or update them every few months, or generate special reports on their costs. The whole toolkit is rolled into AWS and best of all, upkeep is minimal!