Cost and Time-to-market Estimates

Serverless empowers fast and robust feature development

With Serverless technology, scalability is out of the box. This has a positive shift on what a developer, and an organization, get to prioritize.

Small and agile teams can deliver robust features with minimal overhead and cost. Developers shift their focus away from maintenance and scaling, and put the bulk of their energy toward building features that drive outcomes.

We're going to briefly talk about culture shifts on serverless teams, and then get right into the use cases.

Serverless Team Culture

If a team has an idea for a feature, they do not have to request and wait for provisioned resources. It takes a developer a few minutes to get a secure, auto-scaling service running on a serverless backend (such as Lambda). This ultimately empowers teams to prototype quickly and try new ideas.

When moving to production, a great deal of the normal load of caring for a production applications is reduced or completely removed. You don't need to worry about OS level patching, scaling thresholds, or network connectivity. It becomes a natural step for the developer who built the feature to be responsible for it. In other words, "you built it, you own it."

This creates a team culture of flexibility, autonomy, and empowerment.

Serverless Use Cases

In this paper, we will outline and explain some key use cases for serverless:

- Image processing
- Workflow automation
- Scheduled tasks
- Web and mobile apps and APIs
- IoT

We will also show costs of implementation, and estimate time-to-market for each use case.

Cost estimate disclaimer: *The below price calculations should be used purely as estimates, and are not guaranteed. Many things can affect price, such as serverless backend provider, code optimization, and memory requirements. For reference, our price estimates are based on AWS US-EAST-1.*

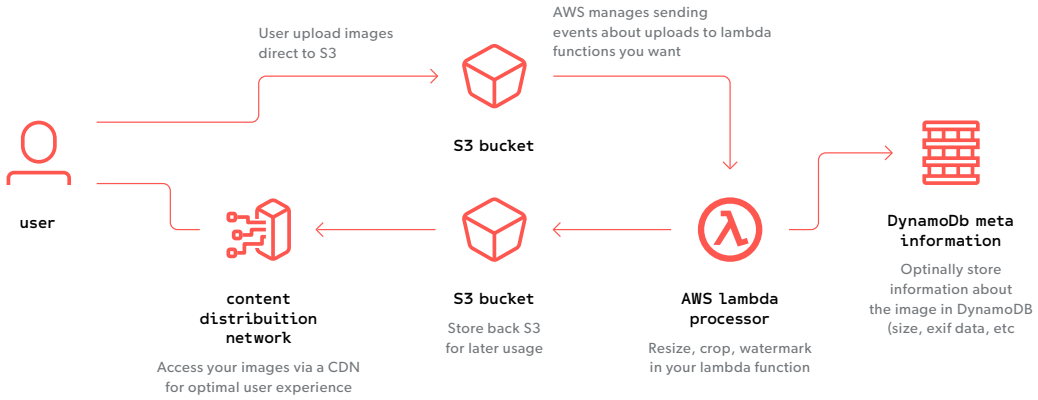
Image Processing

Traditionally, building an image processing pipeline would require teams to have their own queue and autoscaling workers, fueled by a service like Redis or SQS. This would be a multi-part system that would need to be actively managed.

With serverless, however, a single system can handle the entire pipeline. This is not only much easier to maintain, but frees up time to be spent on useful optimizations or differentiation.

Image Processing Example & Time Estimate

For our serverless image processing example, let's say the client uploads an image to S3. An S3 event triggers a Lambda, which manipulates the image (resizes or crops) and puts it back in S3, where other clients can request it. Here's what this architecture looks like:



The image resizing architecture is one of the “hello worlds” of the Serverless ecosystem. It demonstrates, even in simplicity, the power of event driven computing and managed services (S3, AWS Lambda, etc).

We have seen time and time again production-worthy systems be built in days by a pair of developers. Even full-blown, customized media management solutions can be built in weeks by a very lean team.

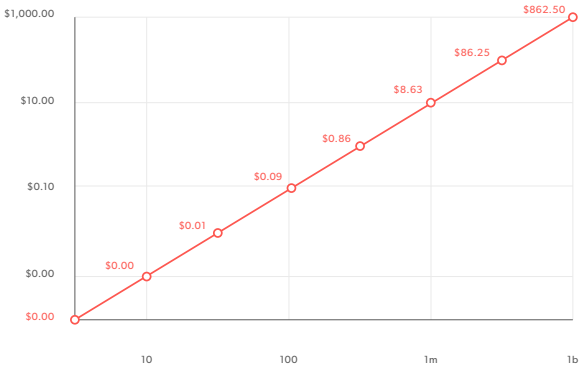
Cost Analysis

Core Assumptions Made

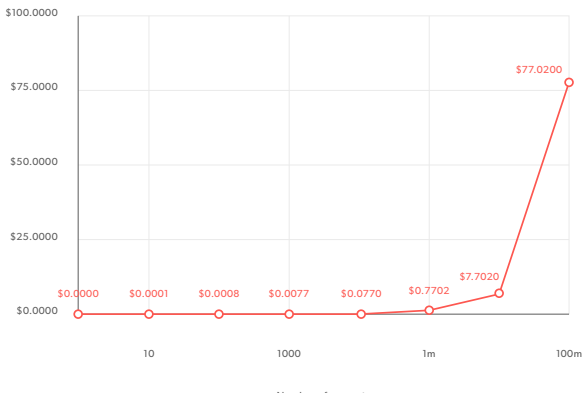
- Lambda runs for 300ms at 1024MB Memory Setting
- Operating on 100kb images

The biggest cost driver for this of system is going to be bandwidth of CloudFront (egress traffic) and the duration and requirements of resizing or processing your images. Storage costs of course grow over time, with the amount of data building up in S3. That said, it is marginal with consideration.

Cost / Number of image requests



Upload and resize cost / Number of requests



Storage cost / Upload count

Workflow Automation

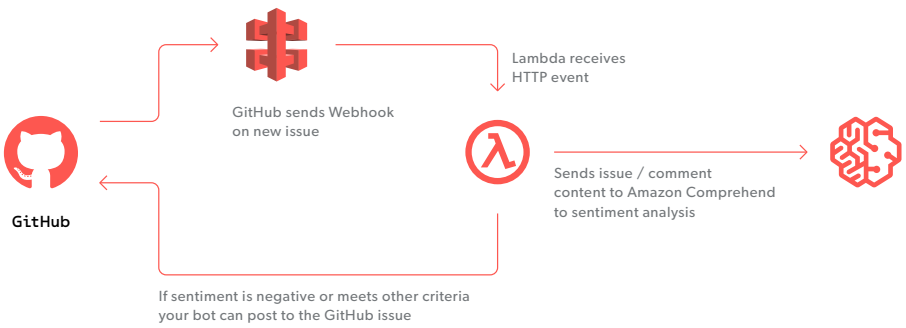
Serverless enables teams to effectively automate their operations without needing to worry about scalability. Multiple CRONs can run at once without needing to provision additional servers; everything can run on serverless compute, such as Lambda.

There are many great use cases for this: CI/CD pipelines, helpdesk reply time checks to track SLAs, monitoring alerts that get sent to Slack, GitHub hooks for checking Issues processes are properly followed, and more.

Workflow Automation Example & Time Estimate

Let's talk about using GitHub webhooks, and leveraging them to automate various checks or workflows. It's common to check if a pull request builds correctly, matches your formatting requirements, folks have signed your contributor agreement, etc.

For a bit of a different example, say you want to make sure that folks are behaving appropriately and kindly within your comments. In a distributed culture, it's easy to forget that there are humans on the other side of the keyboard, so we can write a bot to gently remind them if a comment gets a little too negative.



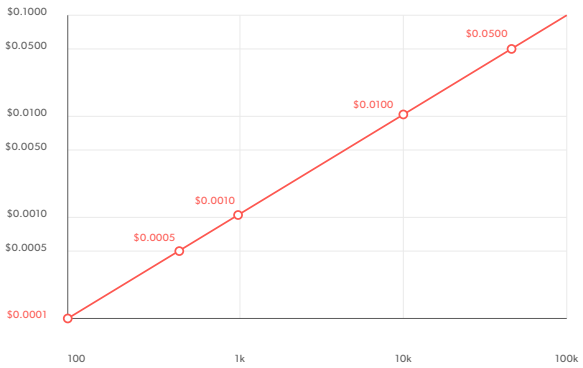
Cost Analysis

Core Assumptions Made

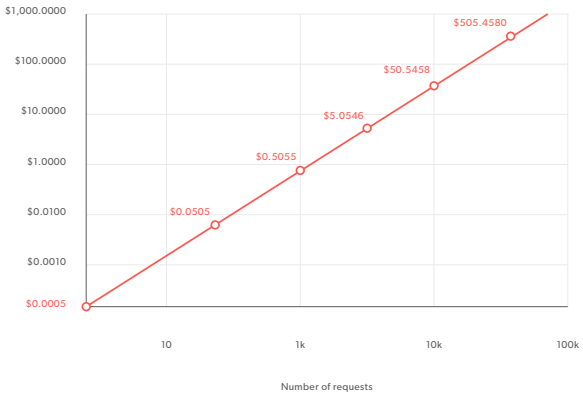
- Lambda runs for 200ms at 512MB Memory Setting

Within this architecture, the main cost driver is going to be the actual character count being sent to Amazon Comprehend. If we assume a reasonable 500 character body being sent to Amazon comprehend, we end up paying about **$0.0008** for each request to come from GitHub, through our API Gateway, to our Lambda, and getting sentiment analysis and sent back to GitHub.

Cost per request based on
Comprehend character count



Request count cost
(500 character body)



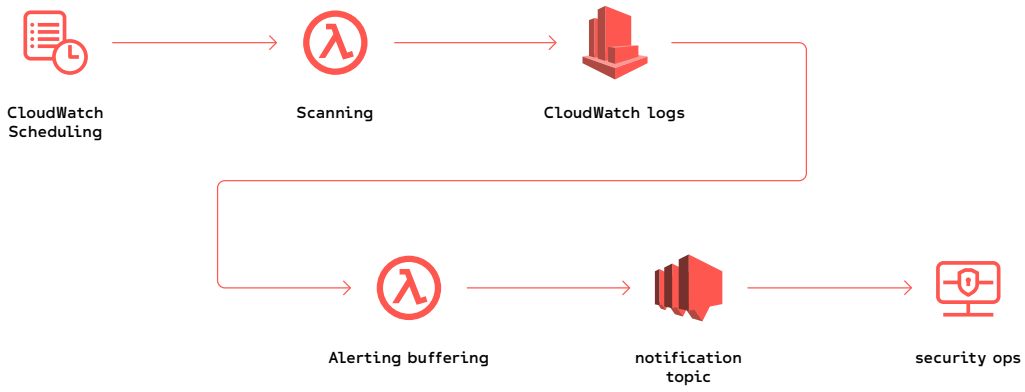
Scheduled Tasks

There are lots of options for running scheduled tasks, and probably the most common is using Docker + CRON. However, maintenance on a Docker instance can snowball fast: you have to get the server and the Docker instance set up and configured, then keep it running on an ongoing basis.

With Serverless, you can create tasks on the fly that scale and largely don't need active maintenance. It makes a serverless backend, such as Lambda, perfect for regular tasks like security checks, cleaning out dev and sandbox accounts resources, verifying data integrity, and more.

Scheduled Tasks Example & Time Estimate

A very common use of the scheduled functions is to automate security checks, security operations teams can run checks at controllable schedules (perhaps daily production, and weekly for lower risk environments), without managing any infrastructure and maintenance overhead on their own utils.



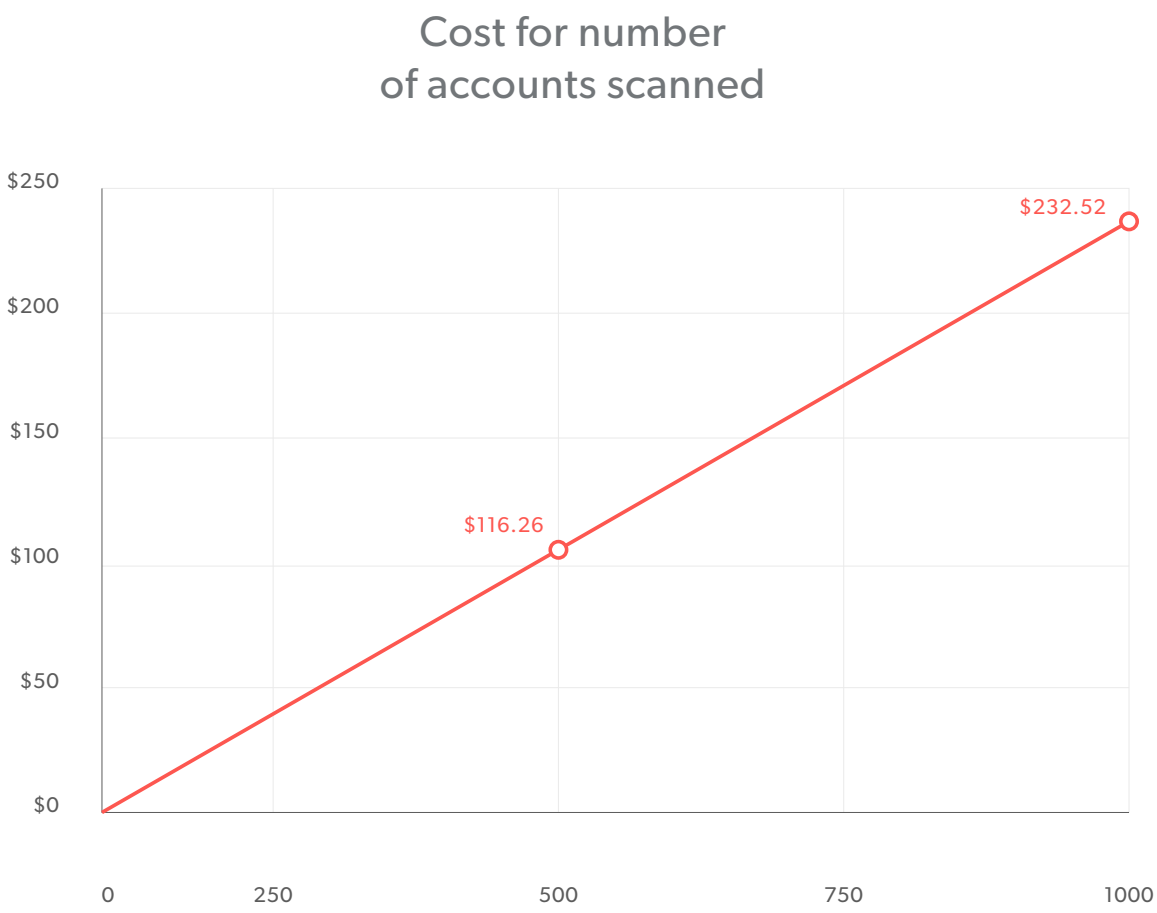
Cost Analysis

Core Assumptions Made

- Lambda runs for 60,000ms at 512MB Memory Setting once a day

- Writes 100kb of results per run
- Send 100 alerts a day per AWS account

The main cost driver in this situation is going to be the memory and duration of the AWS Lambda function running the scans, as well as the output volume of data into CloudWatch logs.



Web and Mobile Apps and APIs

APIs are one of the most common serverless architectures use cases for a good reason—it's just easy to build and maintain architectures in microservice patterns.

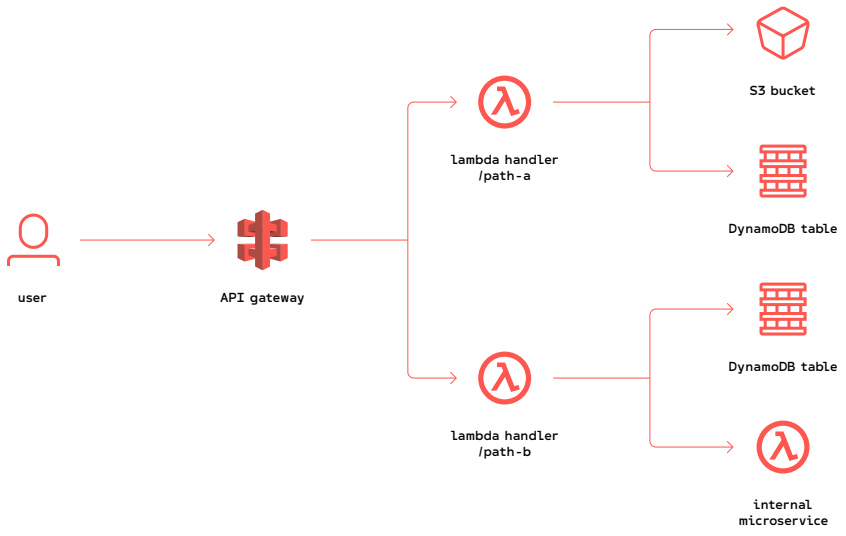
The pay-for-what-you-use billing model has great economics. The auto-scaling, zero-administration aspect of serverless prevents you from having to run a server or maintain the infrastructure for services that are otherwise lightly used, and may barely justify the cost of running.

Apps and APIs Example & Time Estimate

Say for instance you want one of your "services" to act on `/path-a`, and this service stores information about large chunks of freeform text stored in S3 (keywords, sentiment, etc.). We can quickly lookup information in DynamoDB about what keywords a user cares about, and then when we find what we are looking for, we can pull the full text straight from S3.

Another path, `/path-b`, uses another dynamodb table to store information and metadata on using some other internal microservice one of your teams has produced. Instead of needing to expose that internal microservice to your API directly, you can do a Lambda-to-Lambda call.

This lets you skip the cost and latency of API gateway, since you are a trusted caller of the service, and enables a whole new fabric of microservices within your organization.

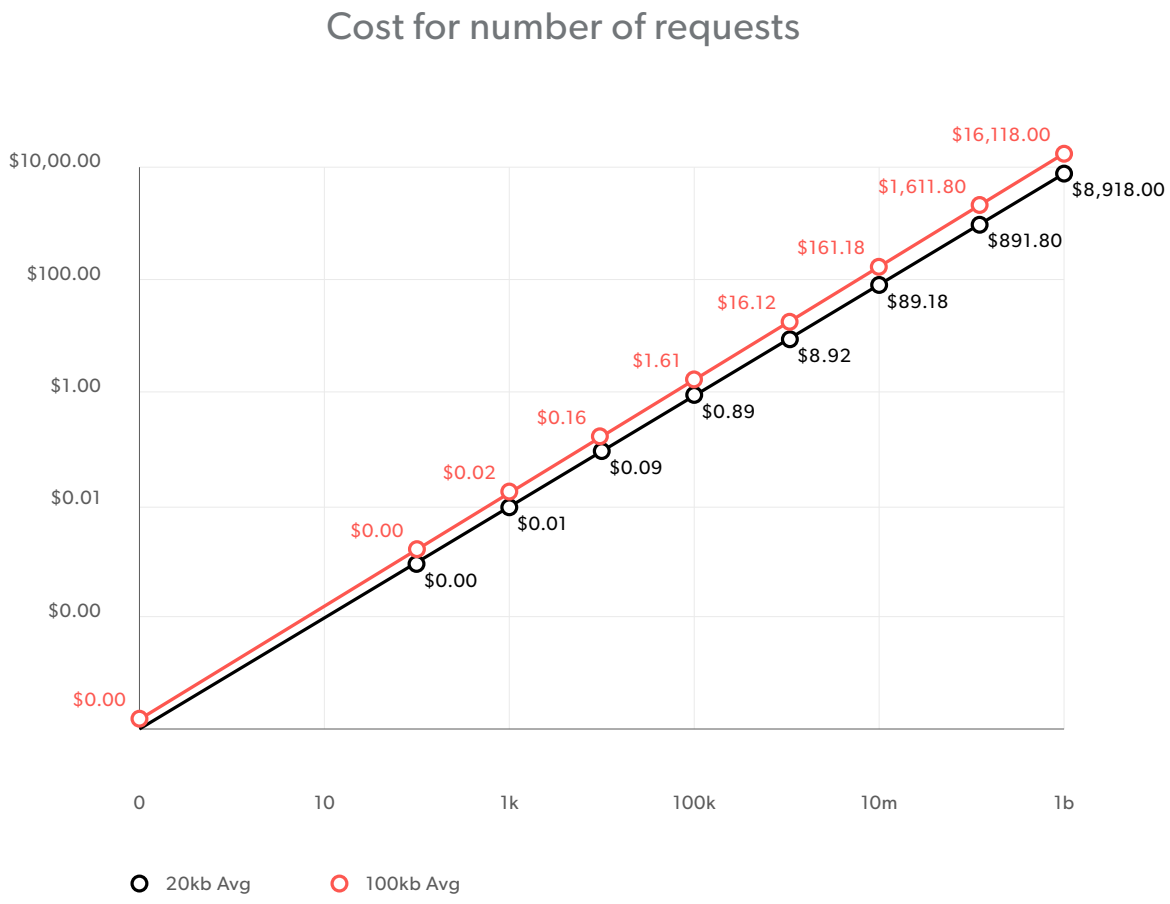


Cost Analysis

Core Assumptions Made

- Each lambda runs in 200ms at 512mb Memory
- Each request writes once and reads twice from DynamoDB
- Each request is 20kb or 100kb on average

Main cost driver in this situation is simply the number of API Gateway requests paired with the average size of the response payload.



IoT

One of the strongest cases we have seen for serverless is in the IoT space.

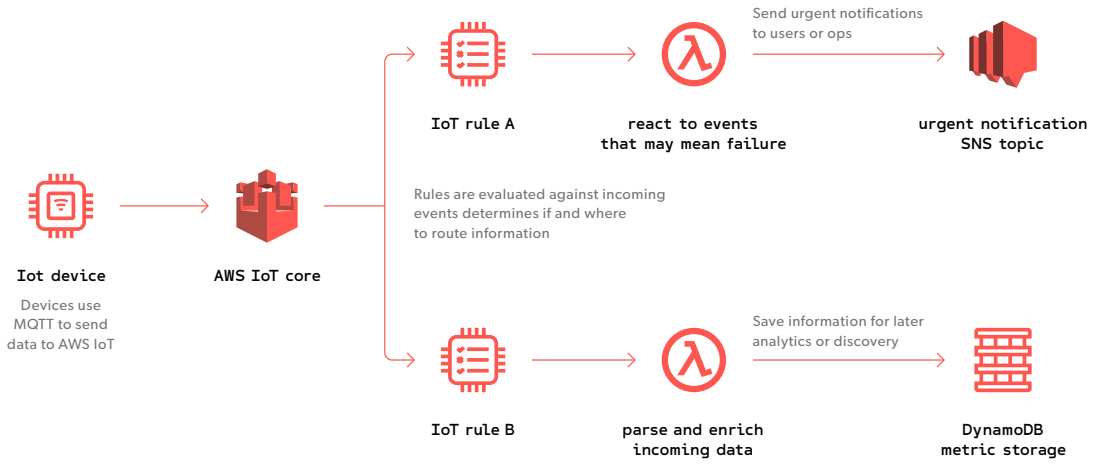
With events being a natural product of devices collecting data and interacting with the real world, being able to ingest and react to these events with event driven compute is a match made in operations heaven.

IoT Example & Time Estimate

In our example, let's assume you have an IoT device monitoring the PH balance and moisture of a plant's soil. We want to do a couple things: monitor the data over time, and alert ourselves if we drop below some threshold for moisture or PH balance goes out of whack.

While doing this all on some small device would be trivial, imagine doing this when managing tens of thousands of plants across many homes—any problem at sufficient scale starts to get interesting.

One of the powers of serverless is that in many cases, solving for the simple singular use case gets us a 90% (or even full) solution in the at scale case. This is especially true in IoT workloads.



Cost Analysis

Core Assumptions Made

- Every message triggers 1 rule and lambda
- Each device is sending 1 message a minute
- Each message writes once to dynamodb for storage
- Every 100,000 messages triggers an SNS alert

Cost Drivers: device count and chattiness