AWS serverless costs

Serverless architectures help teams deploy to production faster but also introduce an entirely new cost structure. Use this step-by-step example to know what to expect on a bill.

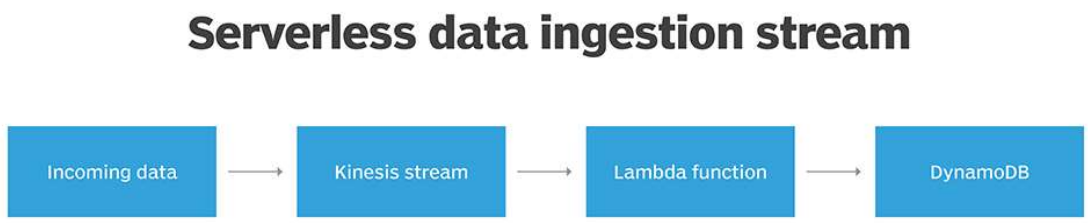
Serverless architectures can help you quickly set up an API endpoint and the logic behind it, as well as persist data, without the need to launch or maintain a server. These architectures can also significantly reduce the time it takes to push applications to production.

Many Amazon cloud services, such as API Gateway, Lambda, Kinesis and DynamoDB, are essential components in serverless architectures. But, while these architectures can negate the cost of server maintenance, there are cases where serverless costs can quickly add up. Therefore, it's important to understand how cost factors into a serverless platform -- especially at scale -- to avoid a nasty AWS billing surprise.

Let's take a look at a common serverless pattern -- a data ingestion stream -- which might consist of the following components:

- Incoming data, which includes system and application metrics, log data and real-time data, such as social media feeds, incoming transactions and ad clicks;
- Kinesis Data Streams ingests any volume of incoming data, stores it temporarily and then makes it available for consumer applications. Kinesis retains data for a default of 24 hours and up to seven days. You can have multiple clients both write data into Kinesis and consume data from the service;

- Lambda functions process incoming data from Kinesis. For example, you can set up Lambda functions to perform calculations, adjust the data format and send it to permanent storage; and
- DynamoDB, which stores data permanently, where other applications can access it.



Let's walk through a few data ingestion scenarios, and break down serverless costs, service by service.

Amazon Kinesis

In Kinesis, you pay proportionally for the frequency of incoming records and for the amount of data in those records.

One unit of capacity is called a *shard*, which can handle up to 1,000 transactions per second and a 1 Mbps ingress. If you need to ingest data at a higher rate, you can add more shards to your Kinesis stream. Each shard costs about $11 per month.

AWS also charges per Kinesis PUT Payload Unit, which is measured in 25 KB blocks. The more data you ingest into Kinesis, the more you pay.

AWS Lambda

In AWS Lambda, you pay for the number of function executions, consumed compute resources and data transfers. AWS charges for function executions at $0.20 per million, and it charges $0.00001667 per gigabytes per second (GBps) of compute resources consumed.

As the memory allocation or execution time per function increases, so too will the consumed compute usage. In many cases, higher memory allocation results in faster executions, which can decrease execution time and thus cost. Therefore, you have to find the right balance among memory, execution time and cost. Try different memory allocations, optimize your code and test -- a lot.

Amazon DynamoDB

In DynamoDB, you pay for the number of read and write transactions -- also called *read/write capacity units* -- that your table supports and also for the amount of data stored in your tables. Each write capacity unit costs $0.47, while each read capacity unit costs $0.09. AWS bills for data storage hourly at $0.25 per GB.

Read/write capacity units measure the number of transactions per second (TPS) and whether your reads are strongly consistent or eventually consistent.

Amazon CloudWatch Logs

When you write software, you must also write log outputs for informational, debugging or monitoring purposes. Lambda functions don't run on a server that you control, so they write log outputs to CloudWatch Logs.

CloudWatch Logs charges customers according to the amount of data ingested and stored. Data ingestion costs $0.50 per GB, and data storage costs $0.03 per GB-month. In most low-volume scenarios, you probably won't incur significant charges. However, once you execute transactions at scale, CloudWatch Logs costs can really add up.

Sample real-time application costs

Serverless cost scenarios will vary wildly, but let's take a look at some sample workloads to contextualize prices.

Let's say you have a Lambda function with 128 MB of allocated memory. This function receives incoming events from a Kinesis stream and takes an average of 100 ms to execute. Each execution results in 1 KB of data sent to CloudWatch

Logs and a single write to DynamoDB. Each item that writes to DynamoDB is greater than or equal to 1 KB, which means it consumes one write capacity unit.

This is a very modest scenario in terms of allocated resources, as it demonstrates the lowest possible memory allocation in a Lambda function and the lowest billable execution time. The DynamoDB write capacity units and CloudWatch Logs output per execution also carry modest costs.

Here is a look at the accrued costs, which are broken down by usage type:

Sample serverless applications cost comparison

USAGE TYPE	100 TPS	500 TPS	1,000 TPS	5,000 TPS
Kinesis shards	$10.80	$10.80	$10.80	$54.00
Kinesis million PUT payload units	$3.63	$18.14	$36.29	$181.44
Lambda executions	$51.84	$259.20	$518.40	$2,592.00
Lambda GB-second	$55.31	$276.54	$553.07	$2,765.35
DynamoDB write capacity units	$47.00	$235.00	$470.00	$2,350.00
DynamoDB indexed data storage GB	$64.80	$324.00	$648.00	$3,240.00
CloudWatch Logs GB ingested	$129.60	$648.00	$1,296.00	$6,480.00
CloudWatch Logs GB archived	$7.78	$38.88	$77.76	$388.80
	▼	▼	▼	▼
Total TPS costs	**$370.75**	**$1,810.56**	**$3,610.32**	**$18,051.59**

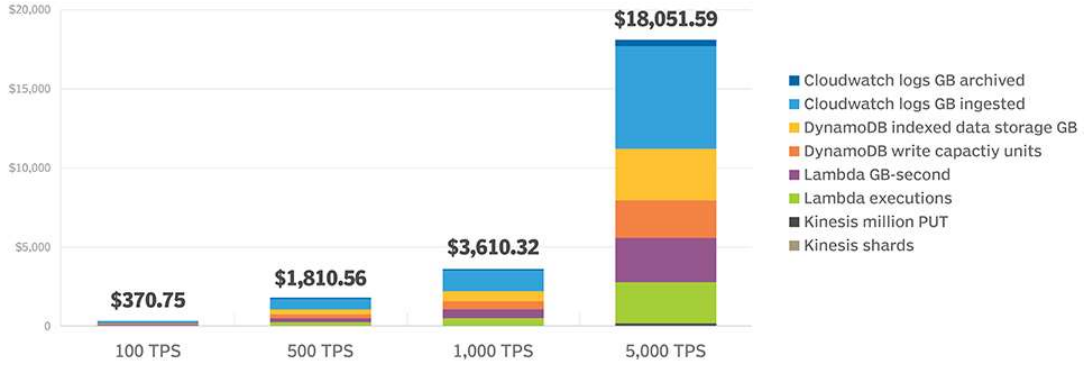
While AWS has a generous free tier, it only gives you about 3 TPS for Lambda executions and 5 TPS in DynamoDB. That might be enough for test environments or even some low-volume applications, but in many cases, that free tier will be quickly exceeded. For that reason, the following calculations do not include the free tier.

If we double Lambda memory allocation to 256 MB, cost increases by about 15%. And, if we use 128 MB of Lambda memory allocation with a doubled payload of 2 KB, we see that cost increase by about 80% compared to a 1 KB payload.

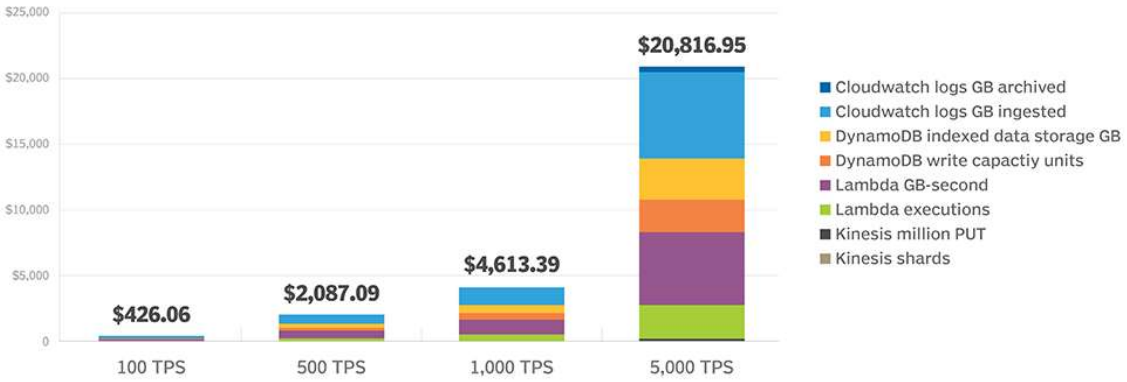
Serverless application cost comparison

While serverless offers some benefits over cloud VMs, it can be difficult to track usage and costs. Get a handle on the real cost of serverless apps with this service-by-service breakdown of monthly charges.

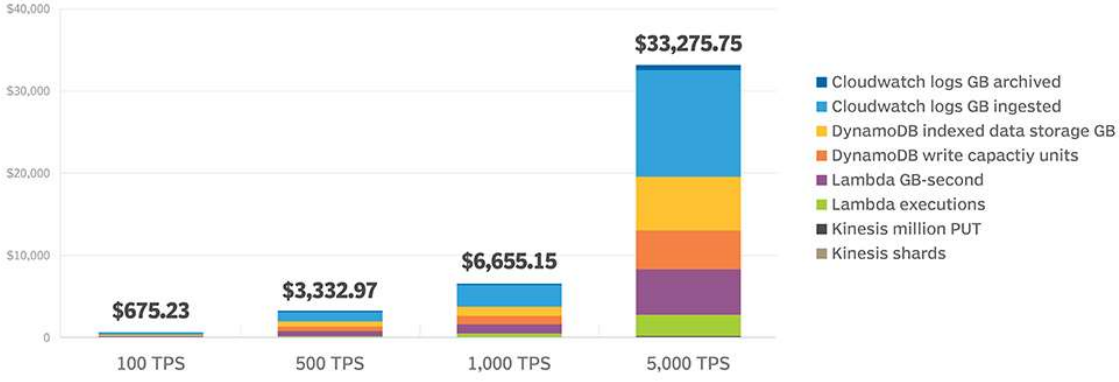
128 MB memory, 100 ms function execution, 1 KB payload



256 MB memory, 100 ms function execution, 1 KB payload



128 MB memory, 100 ms function execution, 2 KB payload



The amount of processed data has a more profound effect on the final AWS price tag than Lambda memory allocation. Keep this in mind when you design your serverless architecture.

If 100 TPS, 1,000 TPS or 5,000 TPS sound like high execution rates, think again. It's common to see architectures grow and eventually ingest thousands of records per second. If you plan to launch and grow a successful application, you should consider the possibility of seeing hundreds -- or even thousands -- of executions per second.

Reduce payload, optimize memory

First, to avoid the aforementioned high data processing and storage costs, reduce your record payload size as much as possible.

Additionally, Lambda GBps can easily become expensive. Find the right amount of memory to optimize execution times, but keep these factors in mind:

- Eventually, the cost of extra memory will exceed the cost reduction gained from faster execution.
- At some point, execution time will reach its floor and will not speed up even when you add extra memory.
- Tools, such as AWS X-Ray, can help you debug performance issues.
- In some cases, you won't be able to optimize execution times. For example, you can't control the time it takes to call an external API.

Use batch processing, reduce log output

Lambda also lets you integrate functions with a number of AWS events. In this case, you can trigger a Lambda function for each documented Kinesis record. But you also have the option to process records in batches and specify a batch size.

Batch processing lowers the number of Lambda executions and, most likely, the amount of GBps consumed. Thus, this method reduces potential serverless costs, but it also adds some latency to record processing. That extra latency

might only be a few milliseconds, seconds or even minutes -- it depends on the batch size, Lambda execution time and incoming data rate.

But, if you use batches between Kinesis and Lambda, it can complicate error handling. Lambda polls the stream to get records from Kinesis, and then it executes the appropriate Lambda function. If the function fails, Lambda retries the execution until it succeeds. If you have a Lambda function that processes events in batches, make sure to delegate error handling to that function.

Batch processing can reduce serverless costs, but make sure to test this strategy at scale, measure the latency and have a plan to handle errors.

Additionally, you should reduce output to CloudWatch Logs as much as possible. CloudWatch Logs can easily result in 30% of your cost in this architecture, which can add up to thousands of dollars per year. On the other hand, thorough logs can result in quick troubleshooting and issue resolution. So, it's important to reach a good balance and be conscious about the amount of data that gets printed in logs.

Use DynamoDB Auto Scaling, Reserved Capacity

While our cost calculations assume a constant data ingestion rate, many applications experience a varying load. In this case, use DynamoDB Auto Scaling to adjust read/write capacity units when usage increases or decreases to save money.

But keep in mind that DynamoDB might not scale fast enough. In some situations, a sudden usage spike might trigger an Auto Scaling event. But the table capacity cannot increase as quickly as needed, which results in DynamoDB throttling exceptions. Perform load tests that simulate the steepest usage spikes before you rely entirely on DynamoDB Auto Scaling.

If you're familiar with EC2 Reserved Instances, DynamoDB offers a similar purchase plan, in which you pay upfront for reserved capacity. You can opt for one- or three-year terms, and you can save around 60% and 90%, respectively.

For 100 write capacity units with on-demand pricing, it would cost you around $47 per month or $564 per year. You could, however, pay $150 in advance for a one-year term or $180 for a three-year term.

Also, you can use Reserved Capacity among multiple tables. So, if at some point you don't need as much capacity for a particular table, AWS calculates cost based on the total purchased Reserved Capacity in your AWS account or even across linked accounts.