When to Go Serverless

NOTE: This content will be available as a video course soon! We are making it available as a Learning Path resource in the meantime to help you prepare for the newly updated AWS Certified Solutions Architect - Associate (SAA-C03) exam.

Introduction

Hello, and welcome to this course, which will help you evaluate when to go serverless using technologies such as AWS Lambda, and when a more traditional architecture using Amazon EC2 instances may make more sense based on your workload's performance optimization requirements.

Before we get started, I would like to introduce myself. My name is Danny Jessee, and I am one of the trainers here at Cloud Academy, specializing in AWS–Amazon Web Services–and AWS certifications. Feel free to contact me with any questions using the details shown on the screen, or you can always get in touch with us here at Cloud Academy by sending an email to support@cloudacademy.com, where one of our Cloud experts will reply to your question.

Who should attend this course?

This course is intended for anyone who may ever need to answer the question, "Should I go serverless?" This includes anyone responsible for migrating existing applications to the AWS Cloud, or any solutions architect looking to decide between EC2 instances and a serverless implementation when designing anything from new cloud-based application architectures to microservices and cron jobs.

Learning Objectives

By the end of this course, you'll know what to consider when choosing between EC2 instances and serverless services for your cloud-based application architectures. Depending on your specific requirements, either approach may be a better option for a given scenario. This course will provide a framework to help you evaluate when to go serverless or when to use EC2 instances instead.

Prerequisites

To get the most out of this course, you should have a basic understanding of Amazon EC2 as well as the various serverless service offerings in AWS. To learn more about all of the serverless services in AWS, I encourage you to check out this course:

A Survey of Serverless
https://cloudacademy.com/course/survey-serverless-2399/

Feedback

Here at Cloud Academy, we strive to keep our content current to provide the best training available. If you have any feedback, positive or negative, or if you notice anything that needs to be updated or corrected for the next release cycle, please reach out to us at support@cloudacademy.com. Thank you!

When to Go Serverless

Hello, and welcome to this lecture, where I will discuss how to choose between using traditional EC2 instances or serverless services for your cloud-based compute workloads. And in this course I'm going to be focusing on AWS Lambda as my serverless service of choice. Now Lambda allows you to deploy and run custom code functions on demand, without needing to provision or maintain any servers of your own. But you should know that in addition to Lambda, AWS offers around a dozen different serverless services that fall into three different categories, including serverless compute, serverless application integration, and serverless databases, each with their own specific use cases within a given architecture. And to learn much more about all of the different serverless services offered by AWS, I encourage you to check out this course:

A Survey of Serverless
https://cloudacademy.com/course/survey-serverless-2399/

So I'm going to begin by discussing the main differences between EC2 and serverless services in general, again with an emphasis on AWS Lambda. After that, I'll explain when you might want to go serverless, or when sticking with EC2 may be a better option instead. It's important to remember that there are no hard and fast rules here, and sometimes your decision may come down to personal preference or just your general willingness to maintain and administer servers. As you'll see, there are scenarios when an EC2 deployment will be more cost-effective than a serverless deployment, but other times when the opposite will be true. So it's important to understand all of the tradeoffs between EC2 and serverless architectures to know what questions to ask when deciding whether or not to go serverless.

So let's start by quickly discussing EC2 instances and when it makes sense to use them. EC2 instances are, of course, your tried and true virtual servers running in the AWS Cloud. They've been around for a very long time and have significantly reduced the burden that's typically associated with provisioning servers in an on-premises environment. You can choose from a wide variety of instance types that can be optimized for compute, storage, or memory, going up to dozens or even hundreds of vCPU, hundreds of gigabytes of memory, and gigabit-level network

performance. But at the end of the day, these EC2 instances are still servers and bring along with them all of the typical server maintenance and administration tasks you would expect with a legacy on-premises server. And this includes everything from configuring storage and networking, to scaling and load balancing multiple servers for high availability and fault tolerance, to patching and updating the underlying operating system as well as any other installed applications. And all of these things carry an associated cost on top of the amount AWS is billing you to run the instances themselves.

Now speaking of billing, when it comes to your EC2 instances, you're either going to have some sort of reserved instance, where you're paying a fixed cost for your instance to be up and running for a predefined amount of time, such as one year or three years, or you'll be paying an on-demand or spot instance rate for however long your instance is up and running. And knowing if you need to always have an instance up and running is a key factor when deciding whether or not to go serverless. Depending on the nature of your workloads, it may or may not make sense to pay for an EC2 instance that is up and running at all times to respond to any requests. Because keep in mind, you're always going to be paying for that running instance, even if it's just sitting idle for extended periods of time.

So one of the big advantages of serverless computing is, just as the name suggests, you aren't responsible for any servers: from your architecture's standpoint, it is server-less. Now of course your code is still running on a server somewhere, but AWS is going to be fully responsible for the upkeep, maintenance, and security of that server. You'll never have any access to it, nor will you ever need to worry about configuring or administering it. And as an added bonus, serverless services are inherently highly available. So unlike EC2 instances, which require you to use Elastic Load Balancing with an Auto Scaling Group to achieve high availability, services like AWS Lambda are highly available without any additional complexity or cost.

Now when it comes to billing for services like Lambda, you're only ever going to pay for what you use. So if you deploy some code to a running EC2 instance that doesn't get executed for two months, you're still going to pay the full amount for that instance just as if it was being fully utilized the entire time. But with Lambda, you won't pay anything until your code executes again. And even then, you're only going to pay for exactly how much memory you've allocated to your function, as well as any ephemeral storage you need above 512 MB. And this billing is down to the millisecond level of execution time. Now that being said, there are some hard limits within Lambda, like a 15 minute timeout and 10 GB memory limit that can't be exceeded. And we'll talk more about this shortly.

When to use EC2

So as you've probably figured out, there are plenty of situations where running your workloads on EC2 instances will be your only option, or at the very least, a clearly preferable option to using a serverless service such as Lambda. And these situations include the following:

- When your workload is long-running, memory intensive, has steady and predictable levels of demand, or if it requires more processing power, memory, storage, concurrent executions, or execution time than Lambda is capable of providing.

Now this situation is probably the most obvious, as it won't be technically possible to execute your code any other way than with EC2 instances. And as I mentioned earlier, these instances can scale up to hundreds of vCPU and hundreds of gigabytes of memory. So if your workloads require resources at this scale, or they require a runtime that isn't currently supported by Lambda, you're going to want to stick with a traditional EC2 implementation and right-size your instances to ensure you have the appropriate level of compute resources available at all times to meet demand.

- The next situation would be if your workload has a lot of external dependencies, or otherwise requires access to the underlying operating system of the server where it is running.

Because services like Lambda abstract away the underlying server and operating system, you won't be able to install any other software on the server where your code is running. Now Lambda does allow you to upload what are called deployment packages, which can take the form of a zip file or a container image that contains your code along with all of its dependencies. But there are some important limitations with these deployment packages: if you're using a zip file, you're limited to a file size of 50 MB, which includes any dependencies or custom runtimes you may need. And in the case of container images, your image is limited to 10 GB in size and generally speaking, if you aren't already using a containerized architecture, you probably aren't going to switch to one just to take advantage of these Lambda deployment packages. But if you are already leveraging containers, you may want to take advantage of AWS Fargate instead, which is a serverless service that allows you to deploy containerized applications without having to manage servers or provision infrastructure.

- And finally, if you're migrating an existing workload from an on-premises environment and you don't have the resources or budget to refactor it into a serverless architecture, you'll probably want to use EC2 instances as part of a "lift and shift" migration approach.

And this one's pretty self-explanatory. If you're doing a lift and shift migration from an on-premises server, you're going to need a server in the cloud. But it's important to remember that this wouldn't prevent you from still going serverless at some point in the future.

When to Go Serverless

So we've seen some examples of when it makes sense to use EC2 instances. Now let's talk about when a serverless implementation might make more sense. So obviously if you aren't subject to any of the constraints I mentioned earlier, such as needing an extreme amount of processing

power or memory, and you don't otherwise need access to an underlying server, you probably have a good candidate for a serverless architecture. Using a serverless architecture allows you to focus on solving your business problems without having to spend any time provisioning or managing infrastructure. Some other situations where you may want to consider adopting a serverless approach include:

- When you're developing a brand-new, greenfield application and you want to adopt the most modern approach possible.

Cloud-native applications often have event-driven architectures that leverage microservices, and both of these lend themselves to the use of serverless technologies that include not only Lambda, but services like the Amazon Simple Notification Service and Simple Queue Service, or SNS and SQS as well. It's even possible to create what are called Function URLs that allow you to expose HTTPS endpoints for your Lambda functions without needing to leverage additional services like API Gateway to deploy secure, highly available microservices. And you can also leverage the AWS Serverless Application Model, or SAM, to define your serverless application's resources using CloudFormation and create more robust, repeatable deployments.

- The next situation is if demand levels for your workloads are unpredictable, but you know they're still within the limits supported by Lambda.

In these cases, a serverless architecture makes sense because you never have to pay for any idle EC2 instances. If you're unable to accurately forecast demand, it becomes difficult to determine the best EC2 instance family to choose for your workloads. And choosing too large of an instance could prove to be a very expensive mistake if it just ends up sitting idle most of the time.

- And finally, Lambda makes it easy to maintain multiple concurrent versions of your functions.

It even allows you to direct a subset of traffic to two different function versions at the same time to support testing, making it easy to quickly iterate, update, and roll back changes to your applications in a way that would require significantly more effort with an EC2 instance.

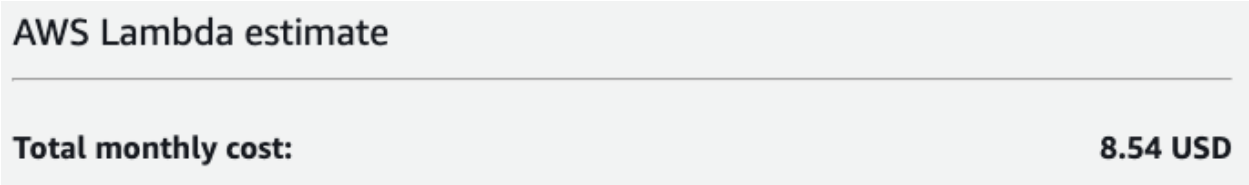
Now we've spent a lot of time talking about Lambda so far in this course, and that's because as one of the flagship serverless services, it's such an integral part of so many serverless solution architectures. So if you're interested in learning more about AWS Lambda and how to leverage it within your own serverless architectures, I encourage you to check out this course:

Understanding AWS Lambda to Run & Scale Your Code
https://cloudacademy.com/course/understanding-aws-lambda-to-run-scale-code/

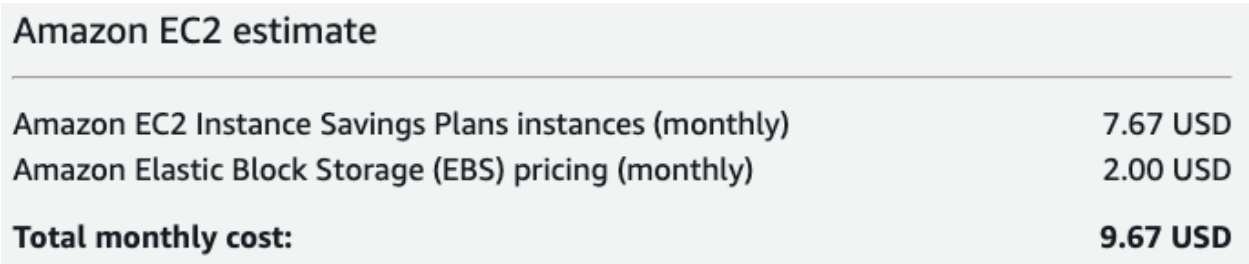
Example Cost Comparisons

Now before I wrap things up, I want to illustrate by way of example just how much your costs can vary between an EC2 and a serverless architecture depending on your application's performance requirements. And the idea here is just to show that you can't necessarily assume a serverless architecture will always be less expensive than one that leverages EC2 instances, or vice versa. Now of course, costs alone shouldn't dictate whether or not a serverless implementation is more appropriate for your use case. Just know that one option may be more expensive than another, and it's always worth running the numbers to see just how much these costs may differ. So for these examples I'm going to be using the AWS Pricing Calculator to estimate my costs, which is available at https://calculator.aws. And I would encourage you to do the same for your use cases as well. Now obviously AWS pricing does change over time, but the prices I'm showing you here were current at the time of this recording.

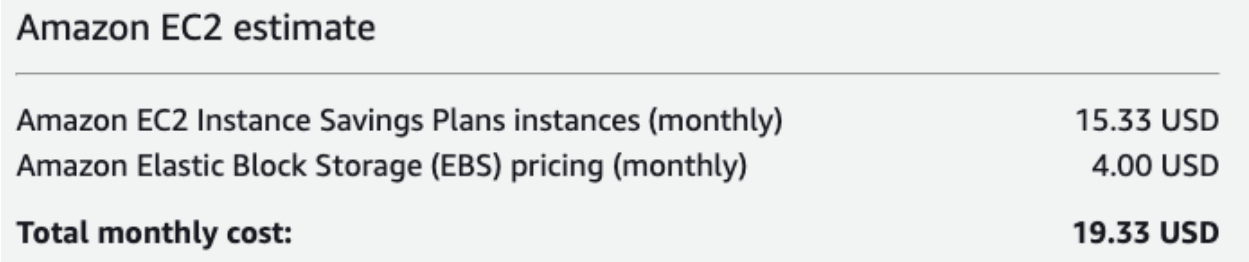
So let's start by comparing a workload we want to deploy as a Lambda function with unpredictable access patterns, but we know it gets executed a total of 1 million times per month. And let's say this function requires 1 gigabyte of memory and 1 gigabyte of ephemeral storage. And let's say it takes a total of 500 milliseconds to execute each time. So using the AWS pricing calculator, we can see that when operating outside of the free tier, this function will cost us $8.54 per month.



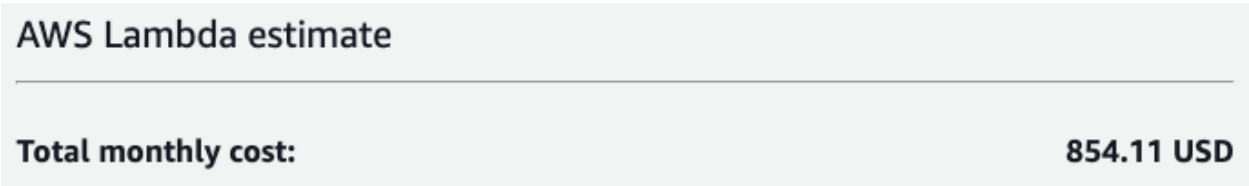
Now let's say we want to deploy this workload on an EC2 instance instead. And let's assume we can use a relatively modest t4g.small instance with a single 20 gigabyte EBS volume. With just the one instance, and using an EC2 Instance Savings Plan, this architecture will cost us $9.67 per month, which is pretty close to what we'd be paying to use Lambda.



Of course if we want to achieve high availability, we would need to add a second EC2 instance, which doubles our total cost to just over $19 per month. And keep in mind this still doesn't include any kind of load balancing, or any additional costs you might incur during prolonged spikes in demand since we are using a burstable instance type here.



So that was an example where a serverless approach saves a little money. But what if we massively scale up the demand? Let's say instead of 1 million requests per month, we now have 100 million requests per month. Well since our Lambda function is billed based on usage, and we've increased our usage by a factor of 100, you'll see the cost for this function has also increased by a factor of 100, putting us now at just over $854 per month.



But let's say we can better predict the demand for these 100 million requests, and maybe we've determined that we can meet this demand by always running 4 t4g.xlarge EC2 instances. So by right-sizing our instances, even factoring in the cost of frequent snapshots for our EBS volumes, this implementation will only cost us about $282 per month, and that's a huge savings.

So again, the idea here isn't to show that a particular architecture will always be the most cost-effective option, but you do see how the costs of serverless and EC2 architectures can scale very differently. And if you are able to accurately forecast demand and right-size your instances, then an EC2 instance-based architecture may be your best option in terms of both cost and performance. EC2 instances also enable you to take advantage of savings plans and reserved instances that can further reduce your costs. But keep in mind that you will be responsible for managing and administering these servers, as well as configuring things like elastic load balancing and auto scaling. So it's important to keep those costs in mind, as well as the additional time and labor required for server administration.

Final Thoughts

So to wrap things up here, there are no hard and fast rules when it comes to choosing a serverless architecture versus a more traditional EC2 instance-based approach. But if you're looking to avoid the time and energy associated with provisioning and maintaining infrastructure, then a serverless architecture is probably best for you. This is especially true if you're building a new architecture from scratch. Serverless architectures are also ideal for:

- rapid prototyping,
- event-driven applications,
- applications with low levels of demand that are not frequently accessed, as well as
- cron jobs that only need to be run on a fixed schedule and don't need to be available at any other time.

On the other hand, you'll want to go with an EC2 instance-based architecture if you need to have root-level access to the underlying server operating system and complete control over your infrastructure. Keep in mind, however, that you'll always need to have at least one instance up and running, even if that instance is just sitting idle most of the time. And if you want your application to be highly available, you'll need at least two instances running across different availability zones.

Serverless services are inherently highly available and don't require you to pay for idle or unused resources. But there will always be cases where memory and performance requirements may exceed the capabilities of services like Lambda. In these cases you'll have no choice but to leverage EC2 and accept the associated costs of updating, patching, and securing your instances.

Summary

Hello, and welcome to the final lecture in this course, where I want to quickly summarize what we've covered.

In this course, we provided a framework to evaluate when it's best to use a serverless architecture with technologies such as AWS Lambda, and when a traditional deployment using Amazon EC2 instances may make more sense based upon your workload's performance optimization requirements. Serverless architectures are great for modern greenfield applications, as well as any workloads that don't have steady, high levels of demand that might otherwise be better suited for EC2 instances. They're also ideal for anyone looking to avoid the hassle and expense associated with provisioning, maintaining, and updating server infrastructure. But EC2 instances will always have their place, especially for compute and memory-intensive workloads or anything that otherwise requires you to have access to the underlying server operating system. We also looked at the costs associated with implementing some example workloads with varying levels of demand and showed how in certain cases, right-sizing EC2 instances can offer significant cost savings over a serverless implementation using Lambda.

That brings me to the end of this lecture, and to the end of this course. You should now be able to make an informed decision about when to go serverless based on your specific requirements.

Feedback on our courses here at Cloud Academy is valuable to both us as trainers and any students looking to take the same course in the future. If you have any feedback, positive or negative, it would be greatly appreciated if you could email support@cloudacademy.com.

Thank you so much for your time and best of luck with your continued learning!

Danny Jessee, AWS Certification Specialist
danny.jessee@cloudacademy.com | @dannyjessee | linkedin.com/in/dannyjessee