# **Learning Amazon Web Services Lambda**

# 

### Serverless computing with lambdas

Selecting transcript lines in this section will navigate to timestamp in the video

- AWS offer many cloud services like databases and working and compute. AWS Lambda is one of those offerings. When Lambda functions were announced in 2014, they revolutionized how we develop software. In this course you will learn the basics of Lambda. You will learn how to deploy code to Lambda in two ways. First, by going to the AWS console and working from there, and then by using SAM, that is a Serverless framework. You will learn how to create several expressions as you first structure this code using simple projects. I'm Marcia Villalba, an AWS Serverless Hero with 15 years of experience building backend applications. I have been designing and developing Serverless applications since 2016. Join me in this journey as we get started developing Serverless applications with AWS here in Weekend in Learning.

### Serverless computing with lambdas

Selecting transcript lines in this section will navigate to timestamp in the video

- AWS offer many cloud services like databases and working and compute. AWS Lambda is one of those offerings. When Lambda functions were announced in 2014, they revolutionized how we develop software. In this course you will learn the basics of Lambda. You will learn how to deploy code to Lambda in two ways. First, by going to the AWS console and working from there, and then by using SAM, that is a Serverless framework. You will learn how to create several expressions as you first structure this code using simple projects. I'm Marcia Villalba, an AWS Serverless Hero with 15 years of experience building backend applications. I have been designing and developing Serverless applications since 2016. Join me in this journey as we get started developing Serverless applications with AWS here in Weekend in Learning.

### Secure your AWS account

Selecting transcript lines in this section will navigate to timestamp in the video

- [Woman] The first thing we need to learn when we create a new AWS account is security. In this video you will learn what you need to do to secure your AWS account. When you first create an AWS account, a user is created at the same time. That user has complete access to all the AWS services and resources within that account. We'll call this user the Root user for this account. You can access the Root user by signing in with the email and password that you used to create the account with. It is strongly recommended that you don't use the Root user, and instead you create an IAM user and use that. Store the credentials for this Root user in a secure place and only use them if you really need them. So you might be asking, why is it not recommended to use the Root user? Well imagine that you only have the Root user and you use it for everything in your AWS management. One day someone steals your credentials and changes your password. Now you don't have access to this account anymore. How can you recover your AWS account back? This can be a tricky scenario. Sometimes customer support might be able to help you, but sometimes they can't depending on the case. Also your credit card is linked to this account, so whatever the account theft does will be billed to your card. This is not a good scenario. The best scenario is that when you create an AWS account the first thing you do is create an IAM user with admin permissions. And store the credentials for the root user in a very safe place and you don't use that account unless it's really, really needed. Then use this admin user for everything you need. You can also create other users with fewer and specific privileges for your needs. And now what happens if someone steals the credentials for the admin user? It's no problem at all. You open the safe where you stored the Root user credentials, log in to the Root user, delete the admin user, and create a new admin with a new password. That solves the problem. That much can be done.

### Basics of IAM

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] We learned previously that we need to have an admin user in our AWS account to make it secure. How do we do that? In AWS there is a service that takes care of access management. It's called IAM. And in this video we will see some of its basic features. What does IAM mean? IAM stands for Identity Access Management. AWS IAM is a service that helps you to control the access to your AWS resources. You will use IAM to control who is signed in and has permissions to use the different resources. All IAM Users have some special properties that define the permissions they have and to which users. Let's look more in detail at what is in there. IAM Users contain a couple of things, groups and policies. Groups are a great way to organize your users. If you're working in a team and you want to give the same access to many different people, you can add users to groups and then give permissions to the groups. Policies contain the permissions that a user or group has. By the default in AWS, all users and groups are created with no permissions at all. Here you can see an example of a policy. You can see that there is a statement in the policy. That statement is an array of effects, actions, and resources. Let's analyze more in detail each of these parts. The effect will define if it's allowing or denying the action. The action is the exact name of the action that is allowing or denying. In general, all actions starts with the name of the AWS service and then the specific action to the service. In this example, the action is dynamodb:Get. It means that it's only possible to get data from the dynamo table. Also you can add more actions to this list. For example, we could put dynamo:Put or dynamo:Delete. In order to only be able to add and delete elements from this table, in addition to getting. Permissions in AWS are extremely fine-grained and as a security recommendation, you are to be as fine-grained as possible. And finally, the policy contains the resource where we are giving permission. The resource can be very generic as in all the tables in dynamodb or very specific as in the case of the example, like the exact table we want from dynamodb. The more fine-grained your permissions are, the more secure your account will be.

### Creating IAM users

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Now that you have your AWS root user and you understand a bit more about AWS security and IAM permissions, let's go and create our first user. Look into your AWS Management Console with your username and password for your root user. When you see this screen, you just need to type the service that you're looking for, in this case it's IAM, the Identity and Access Management. Here you will see an URL. This is the link that your users will use to sign in. In my case, I have changed it to "learning-serverless". In your case, you will have a number that is your account number. So don't be afraid if it looks very different. It's normal if you don't change it. The first thing we are going to do is to create a group and we are going to give administrative permissions to this group and in there we are going to add the user that we are going to create. I have one group already created. You will have zero. So let's create a new group. I will call it "new-admin". You can put any name you want, just be clear on what kind of permissions the group will have in the name. It's a good practice. Then here you will have all the different permissions that you can give to that group. I will just give administrative access to this, basically all the permissions. Then we click next and then the group is created. Now let's go to Users and there you will see no users. I have two here but in your case there will be zero and we will just add a user and we will put a name to it. So let's put "lambdauser". Then we'll need to give different types of access to this user. When using AWS, there is two types of access: The programmatic access and the console access. The programmatic access means that you can access the resources in this account by the SDK, the CLI, some APIs and other kind of tools. We are going to use these programmatic access when we are working with some, so it's very important that we give this access. The console access is that you will get a username and a password and then you can access these administrative console and the things there. We are going to use the console for creating the lambdas and doing some checks in CloudWatch and in our tools, so we also need that access. After we select what kind of access we want, we just need to decide if we want the password to be autogenerated and if the users need to create a new one when they sign in. We'll put "yes" to both. Then we go to give permissions to this user and there's two ways that we can either add the user to a group or then we can start adding permissions to this user. As we already have the group, the "new-admin", I'm going to add my user there and then we'll click next and we'll get to review. There we will review the information from the user and then we will create the user. When you create the user, you will get a lot of important information back like the URL that you need to use to log in, so keep this in a safe place and then we'll get the username, access key, secret access key, password. This information is very important and it won't be shown again. So you need to download this comma separated value file, store it in a very secure place because you will need it through the whole course. The access key and the secret access key are for the programmatic access and the username and password are for the access in the console. As well in that comma separated value you will have the URL to log in. So now we have everything we need, we can log out from this console and then we can open this file. I open it in my spreadsheet and here you can see the link, so I will open that link and this is the link specific for my account. Then the account ID is the numbers that were in the URL. It should populate it automatically for you, so you don't need to worry. Then the username is lamdauser, that you can see here and then the password is this one that you find in your comma separated values. Now we'll log in and the first thing we need to do is to change the password, so we need to put the old password and then we need to create a new password. We change it. And now we are logged in. So in the following videos, we are going to start logged in into our console, so it's important that you are following through.

### Lambda overview

Selecting transcript lines in this section will navigate to timestamp in the video

- [Narrator] AWS Lambda is the main topic of this course so let's get started by learning what it is and when to use it. Lambda is referred by AWS as Serverless Computing. So what does that mean? Serverless is a new term in the software world that involves services that adhere to three criteria. You pay for the services you use. You don't need to manage infrastructure, and you can scale the service automatically up and down depending on the traffic. AWS Lambda lets you run code without provisioning or managing servers. You pay for how much you use and the servers will scale accordingly to your needs. That's why it's called Serverless Computing. So how does Lambda work? The first thing you need to do is to upload your code to AWS Lambda. We will see there in this course that they are many ways to upload your code to the platform. Lambda supports many different programming languages like JavaScript, Python, Java, C#, Go and many others. Then, set your code up to trigger from other AWS services, HTTP endpoints or some other event. We will see some examples of triggers during the course. And finally, AWS Lambda will run the code when it's triggered using the computer resources needed. What are good use case scenarios for Lambda? When you should use it? Currently, Lambda has been doctored for many different use cases. The following list is just a brief summary of the most common use cases. AWS Lambda can be used to execute code when data changes. For example, to enrich data streams or do real-time analytics of data. Lambda is a great replacement for instances for building back ends. That Hello World request, IoT or mobile back ends. Changes in your infrastructure can trigger events that execute code in Lambda for maintaining your Cloud infrastructure. For example, you can use Lambda to turn off your instances if nobody's using them. Lambda can execute on a schedule basics. Some coder perform different task. For example, if you need to send a message at 2 p.m. everyday, you can use a scheduled Lambda to send that message for you. When you should not use Lambda is a hard question to answer. The use cases for Lambda are constantly growing. There are only a few use cases where Lambda may not currently be the best fit. Extreme real-time response is when a real-time response is critical for you. Lambda may take a while to respond and if you're system needs to respond extremely fast, then Lambda may not work for you. Whenever you need complex compute with high memory or compute requirements, Lambda, however, is generic and may not be a good fit for complex compute task. Thus, such as rendering videos are require better specific hardware may not perform well. 100% reliability is needed such in life support systems that cannot fail. Third-party hardware and services may not be the best fit. We had mentioned that you pay for how much you use when using Lambda but what does that mean? Lambda pricing is based on the number of requests, the duration of each request, and the amount of memory Lambda needs to handle each request. You can check the latest AWS Lambda pricing model on their website.

### **Explore the AWS Lambda console**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Now that you know what AWS Lambda is and when to use it, let's log in to our just created IAM user and to start exploring the AWS console. When you log in the first thing you'll see is the management console, and if you go around, you will start seeing some helpers here in the build a solution, but we are not interested in that, then you will see some tutorials and then if you go up in the AWS services, you can make this extended list and you will see all the services, so here is a very long list of services. The easiest way to find something in the console is to search for it. So we are going to put Lambda here, and this will take us to the Lambda console. If you don't have any function, you may have a presentation of the service, but if you have some function you may see something a little bit different. As I have one function that I create to show you, I will not see the presentation for the Lambda service, but there you will find some important information on getting started with Lambda. If you go to dashboard here, you can see how many functions you have, that concurrency, you can have a thousand Lambdas running at the same time, with no changes to a console that's interesting, and then, how much code you have stored, I have only one function, so this is a size of one function, and there are some metrics as I have not run that in a while. There is nothing shown here, so let's go and see the function I have, written and how it looks. So if we go to functions, here you will have all the list of functions that you have in this account, in this region. So, regions are very important in AWS, there are different regions that means they are different parts of the world where the AWS data centers are. So you need to pick one and then use all the services within that region. So in this case, I will pick Ohio, you can pick the one that is closest for you so everything will be sand boxed in Ohio, all the functions I will create will be displayed in Ohio, but for example if I go to Virginia, you will see that I have a lot of functions here, and that's because I've been using Virginia before and I never use Ohio. I only have this function that I created for this course, so that's very important to have in mind. If we want to see an existing function then we click on the function and the console for Lambda will open. So the first thing you will see is the designer, and here you will see the function, and if it has layers, we are not covering layers in this course, so that's something where you can manage dependencies or add extra functionality to your Lambdas, but we are not covering that in the course, then in the left, you can see the triggers, in this case, this Lambda specifically gets triggered by API gateway, and in the right you can see all the different resources that the function can access. In this case, the function can only put logs in cloudwatch, so it doesn't have much permissions. The next thing you can see is the code of the function and this is very simple piece of code, it's one file, everything in the handler, and here we can see that there is the event object, and here we can see that it's just building an HTTP response that says hello from Lambda, and it's returning it back to API gateway. We will see a little bit more in detail how API gateway works in the following videos. After that, you can see some properties from the Lambda that we are not going to talk about during this course, but they are available here for you to experiment. One important thing if you click on the trigger, then you will see the trigger information, and here we can see that we have one API called hello world, and then it has an end point, so we can click on the endpoint and it will run the Lambda for us, so it says hello from Lambda. So now we can go up again, to the monitoring tab, there we will see some metrics, we will explore this metrics in a little bit more detail later, and we can see also some logs, we are coming to this tab with more details in a future video.

### **The AWS Lambda programming model**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Now that we are familiar with the basics of AWS Lambda, we need to look at the programming model for Lambda. What do you need to know to get started working with it? I will be using JavaScript for my examples, but the same concept applies to other languages. We can break the Lambda programming model into three basic parts. Lambda has triggers, it has a handler function, and it has a specific code for Lambda. Let's begin by looking at some examples of AWS Lambda triggers. One typical example is an HTTP request that comes into our system using API Gateway and then API Gateway triggers a Lambda function. Another example is a record in a DynamoDB table that has been modified and this can trigger a Lambda function to execute. The creation of a new file in our file storage S3 can trigger a Lambda function to execute. Also, a new message in an SQS queue, that is the normal messaging queue that AWS provides, can also trigger a Lambda function to execute. Now, let's look at the Lambda function handler. We have the handler function that will be invoke when the function is called. The handler function has an event object as an input parameter. That is the data sent during the function call. The event object changes depending on who invokes the Lambda. This means that the event object when API Gateway triggers the Lambda will be different than the event object when Lambda is triggered by Kinesis. Another input parameter is the context object. This parameter contains methods available to interact with runtime information. The last input parameter is the callback object. This parameter is not mandatory and not available in all cases. It is used to return information to the one that invokes the Lambda. So the anatomy of the Lambda function take us to the execution model of Lambda function. There are three execution models for a Lambda function, synchronous, asynchronous, and poll/stream based. Let's look first at a synchronous model. For this model, the Lambda function can respond back to its invoker. For example, this is the case of an API Gateway that is in charge of handing HTTP requests and responses. In the synchronous model, services can be invoke by other services and must wait for a reply. This is considered a blocking request because the invoking service cannot finish executing until a response is received. The asynchronous model handles requests that are non-blocking. A service can invoke another service directly or it can use another type of communication channel to queue the information. The service typically only needs to wait for confirmation, acknowledge that the request was received. For example, this is the case of S3. When a new file is created, it can trigger a Lambda function, but the entity that created the file is not blocked waiting for the Lambda function to execute. The poll/stream base model is when Lambda polls the stream, or message queue, and invokes a Lambda function synchronously. For example, SQS is poll/stream based. Messages are put into the queue and then the Lambda platform polls the queue. When there is a new message it triggers the right Lambda function.

### **Create your first AWS Lambda**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Now that you are a little bit more familiar with AWS console and the Lambda console, let's go and create our first function. So go to the Lambda service and to functions. And there, you can create a function. Here we have three options. We can author one from scratch, use an existing blueprint, or browse there serverless application repository. We are going to author one from scratch, because I think it's the best way to learn how this works. So, we will create a new function, and we need to put a name to it. So, let's put my first function. The run time is Node 10. You can pick between all the different run times that are there, but we are going to use Node 10 for this course. The next thing we need to do is to give permissions for this Lambda to do things like put messages in the log. So, here we have some options. So, we can create a new role with basic Lambda permissions. This is the easiest way to get started. We can use an existing role that we have for another Lambda that we want to reutilize, or then we can create a new role using policy templates. For this example, we are going to use the basic Lambda permissions. We are going to create a new role with those, and that is enough for our use cases. So, now we can go and create the function. This takes a few seconds, and when it's ready, it will redirect us to the function page. So, if it's in the designer, we don't have any triggers for this function. And this function has permissions to input information in CloudWatch. So, we can continue down, and we can see the function code. And this Lambda is returning and APA Gateway response. We don't have a trigger that is an HTP, so this response might not work for us. So, let's do this, console, log, event. And then console, log, hello, Lambda. So, now we are just running the Lambda. The first thing the Lambda will do is bring in the console. The event object that we are getting in, and then it will print a message called Hello, Lambda. So, we can save this. So, now you are wondering how we test this, because we don't have any triggers. So, in reality if we don't have any triggers, it's going to be very hard to execute this Lambda. But, in a test scenario, what we can do, is we can create a new test. So, let's create a new test for the Hello World function. And here we can put test event. So, basically, this will create a new test template called Hello World, and we are going to pass information in the event object as an input. So, here we are going to pass key one, value one, key two, value two, and this basically is the event object this chaser. So we can create this test, and now we can run it. So, here after you save, you can see all your saved test events. And then, if you press test, it will run with those parameters that we just configured. So here, we can see that the return result is new. That's okay, because we have not returned anything. But, if we go down to the log output, we can see this what the logs are printing. We can is in this line that the event objects is being print. Key one, key two, key three, and its values. And then, that the message that we put in the console is also being print. So this is how you can test a Lambda without a trigger.

### **Triggers for AWS Lambda**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Previously, we learned that Lambda functions can be triggered in different ways. Let's discuss the most popular triggers and learn when to use them. API Gateway is an AWS service that allows you to create and maintain APIs. You can create an API that acts as a front door for your application. API Gateway will generate an HTTP endpoint that your clients can call. For example, your clients can call the endpoint with the method Get in the path Hello and that API will be managed by API Gateway. We can configure a Lambda function to get triggered when an API receives a request to that endpoint. With this functionality it is very easy to create totally serverless web applications backend. Amazon S3 is an object storage built to store and retrieve any amount of data from anywhere, websites and mobile apps, corporate applications and data from IoT sensors and devices. For example, S3 can trigger a Lambda function when a new file is created, updated, or deleted in a specific path or with a specific extension. With this it is very easy to enrich your files when they are created or create a step in a workflow for your data. Amazon DynamoDB is a NoSQL database that delivers reliable performance at any scale. Dynamo has a feature called Dynamo Streams that can trigger a Lambda function whenever a new record in a specific table is created, updated, or deleted. This is a great functionality to manage transactions and to do stored procedures in a NoSQL database. Amazon Simple Queue Service, or SQS, is a fully managed message queuing service that enables you to decouple and scale microservices. For example, when a new message arrives to the SQS queue, it can trigger a Lambda function. Amazon Kinesis make it so easy to collect, process, and analyze real time streaming data so you can get timely insights and react quickly to new information. For example, when a new event arrives at the Kinesis stream, that can trigger a Lambda. This is a great tool to process high volumes of data in real time. These are just some examples of the most common triggers for Lambda functions. There are many more triggers that can produce really interesting applications. You can check the documentation of AWS for more information on the available triggers.

### **Introduction to API Gateway**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] API Gateway is one of the most popular triggers for AWS Lambda to create Serverless back ends. Let's look at it in more detail and explore some possible use cases. Amazon API Gateway is a fully managed service that make it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. You can create an API that acts as a front door for your applications to access data, business logic, or functionality from your back end services. Amazon API Gateway handles all the task involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, authorization, access control, monitoring, and API version management. API Gateway has no minimum fees or startup costs. You pay only for the API's cost to receive an amount of data transfer out. How does exactly API Gateway work? On the left we have all the different services that can call API Gateway from, such as Web and mobile applications, IoT devices, and anything that can call to an HTTP or WebSocket endpoint. These request are intercepted by API Gateway that will take care of managing the traffic of the request. It will take care of making sure that those request can call those APIs. It will monitor the traffic of the request and then it will send the request to the right service. Services such as AWS Lambda, EC2, DynamoDB, Kenesis, or even other public endpoints. So when you should consider using API Gateway? This is the most typical example of the use case of API Gateway: When API Gateway is triggering different Lambda functions depending on the request that is receiving. API Gateway supports WebSocket so it's possible to implement a fully serverless real-time chat with session management using Lambda, API Gateway, and DynamoDB to store the sessions. API Gateway can connect directly to DynamoDB, and perform CRUD action directly over the table without the need to have a Lamba or any server in between. It can be connected easily with Cognitive or with other Lambda to validate all incoming request to make sure the users performing those request are authorized to make them.

### **Add API Gateway as a trigger to Lambda**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Marcia] So now we tested our Lambda that didn't have a trigger, so this is how it looks. So what if we add a trigger? We can add an API Gateway, we can create a new API, and for the security, we can pick different types of security, but in our case, we just leave it open so everybody can connect. Then, just with this, we can do Add, and this will add the trigger for us. We'll create the API and then add the trigger. So now we can see, in the API Gateway, when we select it, then we have the name of the function, myfirstfunction, and if we click on it, it'll take us to the API Gateway Service. Here, you can see myfirstfunction. And if we go back to the Lambda console, we can see more information about this API here. So the first thing we can see is the API, then we can seee the endpoint, and this is important for us because if we click on this endpoint, this will trigger the Lambda. Then we have, that it doesn't have any authorization, and the method is any, so basically get, post, put, delete will all call this Lambda. And the resource path is basically myfirstfunction, so we could have back slash hello, back slash mysecondfunction or whatever we imagine, in this case, it's myfirstfunction. So if we have myfirstfunction with any method, it will trigger this Lambda. If we go back to our Lambda, we can see the code. The code we have is very simple. It's not returning anything, so if we run this API Gateway, we are getting some error. So we need to change the code to something a little bit better. We can do an HTTP response, just a simple one. So basically, we build an HTTP response that returns the statusCode 200 and a message that says Hello Lambda, and then we return this to API Gateway. So if we save this, now, I can go to the trigger and click on endpoint in my browser, and it will show us Hello Lambda. So what if we wanted to do a little bit more complicated, like pass parameters in that endpoint, so we would like to do something like this. Name Marcia, for example. Nothing happens because API Gateway doesn't know what to do with this, so we can change a little bit the code to know how to process the parameters. So as I said, the event object is specific for each of the triggers, in this case, we have the event object for API Gateway, and inside the event object, there, we have one parameter called queryStringParameters. So if we do that, then we can get information from the query string. You can also get information from the headers, from the body if it supports. So all the information that is passed through API Gateway will go to Lambda through the event object. So we want to do Hello World that is return Hello and the name of the person that is calling this. So we can do something very simple. So basically, here, I'm grabbing the name from the queryStringParameters, and in this case, if you have other queryStringParameters like age or country, they will be in a JSON, so you can just put .country, .name, .age, whatever you fancy. Then we are going to write the message. And then if the name is not known, we are going to change the message to Hello name, and this will return Hello Marcia, Hello John, Hello Laura. But if there is no name, it just will return Hello Lambda. So we leave the rest, the response as before, but we need to change in here, instead of Hello Lambda, that it shows the message, And the simple thing will return the Hello World accordingly to our specifications. So we can test this again. I'm going to the URL of before in the browser, and we can try this name Marcia, and we can see Hello Marcia, we don't need those commas, for example. Hello Marcia, Hello John, or we can remove it, and we can have Hello Lambda. So it's really easy to pass things from API Gateway to Lambda.

### **Introduction to Postman**

Selecting transcript lines in this section will navigate to timestamp in the video

- Now that we have added an API Gateway to trigger the Lambda, we test it in the browser, but the browser is very simple. It only allow us to test get. In this case, the API Gateway will trigger the Lambda when any method is executed, but we want to have a little bit more flexibility. We want to run post and the leads and put and other methods for our HTP code. For that, we can use many ways, we can curl in our terminal, or then we can use an application that allow us to do that. I will recommend to use Postman, if you're not familiar with curl. Postman is an API Development Environment and it has a lot of features, including HTP code. So, you can curl a get, a put, whatever method you want with no problem. Define the bodies, define the headers and you will see everything there. So, download Postman from this URL, install it in your computer, you might need to create an account to get started with it. And then, let's jump into the Postman user experience and see the most important features. So, when we open Postman, we can see that we have the left all the different methods that we can pull, get, post, put, delete and whatever you imagine. And then the URL here, that you can type. Then here, you can send the HTP request or you can save it so you can use them later. Here you can define the query stream parameters so in the case we will have, like before, the name being Marcia or Sean or Laura, we can put it here. Also you can have headers if you need so and if it's a post you can have a body and you can define what type of encoding it has. It can be a raw jSON or text or it can be a binary or graphQL, whatever you imagine. So, for example, let's test our URL that we can get from the browser. We will put it with get and here we can have a name. That is Marcia, and then I can send it. And this returns our response status code 200. This is time that it took to get the response. You can save the response if you want for later and here is the body of the response that it says, "Hello, Marcia." we can change the method to post and do exactly the same and it should work. So, this is a great tool to test your HTP request and response.

### **Test your function with Postman**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] We have tested the function with Postman and we have got back the results that we expected, but sometimes knowing that a function work is not that simple and we need to do a little bit more exploration to know if things are working. Monitoring and debugging is a crucial part of working with any system. You need to be able to understand the state of your system by looking at certain groups of your logs. This way you can detect possible bugs, prevent disasters, and validate your assumptions. AWS offers to its users a service called CloudWatch. CloudWatch is a monitoring and observability service built on the AWS platform. CloudWatch provides a lot of data to monitor your applications. Also, it collects the monitoring data in the forms of logs, metrics, and events. When you work with AWS, the AWS resources will be sending some metrics automatically to CloudWatch. So you can monitor them easily without doing anything in your function. CloudWatch provides CloudWatch dashboards where you can see metrics from different AWS resources. Now, let's go back to the AWS console to see the CloudWatch dashboard for the function that we have been working on. So, here we have our Lambda function and we can go to Monitoring. In this Monitoring tab you will see some metrics. It's important to adjust the timeframe because you may have stopped working this project and you may not have seen anything, so I will put the timeframe a little bit bigger so we can see some information in the metrics, but adjust your timeframe accordingly to your case. You may have been working on this continuously, so maybe one hour is enough or a couple of days or a week if you have stopped working with this course. The first metric we can see is the Invocations. The invocations basically, it counts how many times this Lambda has been called and in here you can see that we have called this Lambda one time on the 28th of August at 6:10 and then we have called this Lambda again six times on the same day at 6:25 and again at 6:45 two times and one more time at 7:00. So, this is the amount of invocations that is graphed for you. Then we can see the Duration and the duration is a little bit more tricky because it's just creating minimals and maximals and average of how long the function lasted. So, functions tend to run for a period of time from zero milliseconds to minutes. So in this graph you can see the duration. The duration is the length that a function took to run. In this case, you can see that we had six invocation at 6:25 and we can see the maximum, the average, and the minimum of these six invocations. So we have the maximum duration for one of the functions was 94.9, the average of the six invocations was 50 milliseconds point four, and the minimum that was very fast function that run in 12.4 milliseconds. So, that's how you can read the duration graph. Then next we have the Error Count and Success Rate and here you can see how many errors you had in your invocations. In our case we didn't have any errors, so everything is in the success rate. There is three graphs here that don't apply to this course, the Throttles, Iterator Age, and Dead Letter Errors, because this a little bit more advanced and we are not covering in this course, but if you have a lot of concurrency and a lot of traffic in your functions, you might be interested in paying a look at these metrics.

### **Monitor your function with CloudWatch**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Also CloudWatch provides logs for your Lambda functions. If you want to log something in your function you just need to write console log if you're using Node.js and that message will appear in your log. We have already tried that in one of our first functions, but we can see what is going on in this function, add a log, and see it in our logs. So let's go to the function code, in the Configuration, and the function we click there. Then we can see the code. If you look at this piece of code there is no console log, so nothing is going to be written in the console. So, we can, for example, print the event object. This is a good thing. When we print the event object, then you can stop this video and look at the event object and see what API Gateway is sending to Lambda. It's a good exercise for you. So let's save this. Let's run this function using the API Gateway with Postman as we have done before. Now we have trigger of the Lambda. So if we go to Monitoring and then we click on View Logs in CloudWatch a new window will appear and there we will be able to see the logs. So, when you open the logs for the first time you will see that there is a list and these are the group logs. Usually how AWS works is that it will group the logs in little batches and this depends, maybe, if you're executing the Lambda in one moment it will put them in one group or if there is a lot of executions it will start creating different groups. But it's important to look at the date to know which is the latest of the grouped logs that you need to open. So here you can see this is the latest, so we are going to open that and in there you will see the logs for the execution that we just did. So here you can see the start of the execution. So it always start with Start, then there are the logs in the middle, whatever you put in the console log, and then you have the end of the execution. And in general, there is always a report that will tell you information about the execution. So, we can see the start. There is this RequestID that is unique for this invocation and then we can see that in the end there is the same RequestID and in the report there is the same RequestID. We can see that the duration was 153 milliseconds. How much it was billed because AWS bills on bunches of 100 milliseconds. It will billed for 200 milliseconds. The amount of memory was assigned to this Lambda was 100 megabytes and the amount of memory used was 75 megabytes. This is important if you want to tweak the amount of memory that you are giving to your Lambda. If we go to the exact logs, then we can click in there and this is the event object that we just logged. I recommend that you stay here and look at all the information that is passed by the event object. You will find the core string parameters that we pass, like you can see here, but you can also find the headers, the body, the path, the method, and all kind of information that will enrich your development with Lambda.

### **Challenge**

Selecting transcript lines in this section will navigate to timestamp in the video

(whimsical instrumental music) - [Instructor] This is your first challenge of this course. This challenge will take you up 15 minutes to complete. The objective of this challenge is that you create a new lambda function with an API Gateway trigger. You want your lambda function return a result of adding two numbers together. Those input numbers you will get from the API Gateway as input parameters in a post request. Here's an example of the request and the response that we are expecting. I will show you my solution to this challenge in the next video.

### **Solution**

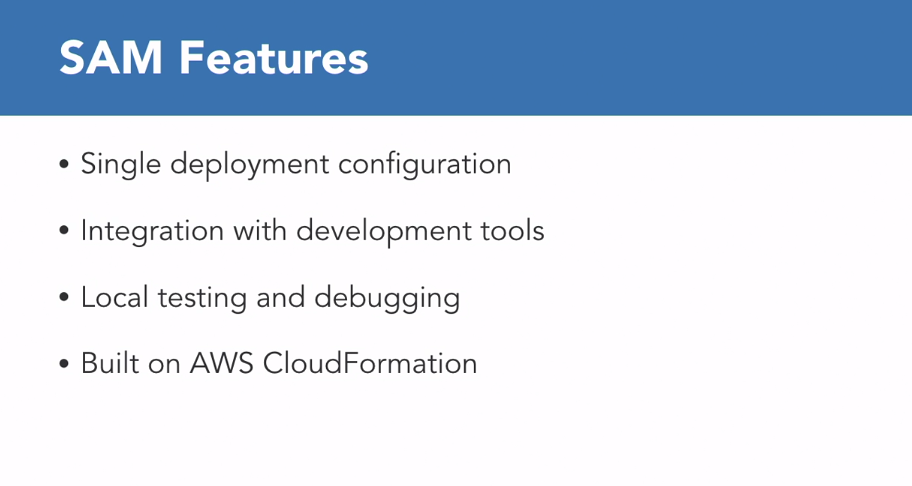
Selecting transcript lines in this section will navigate to timestamp in the video

(upbeat music) - [Instructor] So I will show you my solution to the first challenge. And let's start by creating a function. So we will author it from scratch and we will call it challenge1. We leave the runtime Node 10, and we will just select the permissions to be a new role with basic Lambda permissions. That's enough for us. Let's create the function, and after the function is created, we are going to add the API gateway and then modify the code so we can get the addition of two numbers. So we can see here in the Designer that this function doesn't have a trigger, so let's add one, an API Gateway, and we will create a new API and we will leave the endpoint Open so anybody that has the URL can trigger this Lambda. Good, so now our API Gateway will trigger this Lambda, and this Lambda has permissions to write in the logs, so we are fine. We can see here the URL, and then we can go to modify the code. So here we can see the code for the Lambda. Let's leave the response in there and add some more information. We will have two values, value1 and value2, coming as queryStringParameters on the API Gateway, and API Gateway will pass them to Lambda using the event object, so we need to get that. So we will get those parameters. We get the first parameter, value1. Then we get the second parameter in the same way. And now we have the two parameters then we can create a message. This message will be display if there is no parameters or one of the parameters is missing, so we will write, "Value 1 and Value 2 are needed", something in this lines. Then we can create a message that is the result of adding these two values if everything is right. So we can first do, if there is something in the StringParameters then this is something we can continue, and if the value1 and the value2 are not undefined, meaning that there is some information in there. So if everything is okay, we can modify the message. And we can say, "The result is", and here we can parse the value1 and add it to the value2. It's important to parse these values, if not, they will be considered strings, and also use these brackets so they will be added before putting them into the message. Then when we have that, we just need to modify the response in order to print the message. And that's it. We save, then we go to the API Gateway, we grab this URL and we test it in Postman. Let's go to Postman, let's past the URL here, the instructions says that is a post. In this case it's no problem because API Gateway will create an ANY METHOD API, so basically no matter what kind of method you pass by, it will just work if you're connecting to the right endpoint, but we will try with POST, and here we will do value1, and value2. So this just print "The result is 3", that's good. What happens now if we remove one of these? "Value 1 and Value 2 are needed." We remove both, then we still get the same error so this is good and it's doing what we are expecting it to do.

### **Introduction to AWS Serverless Application Framework (SAM)**

Selecting transcript lines in this section will navigate to timestamp in the video

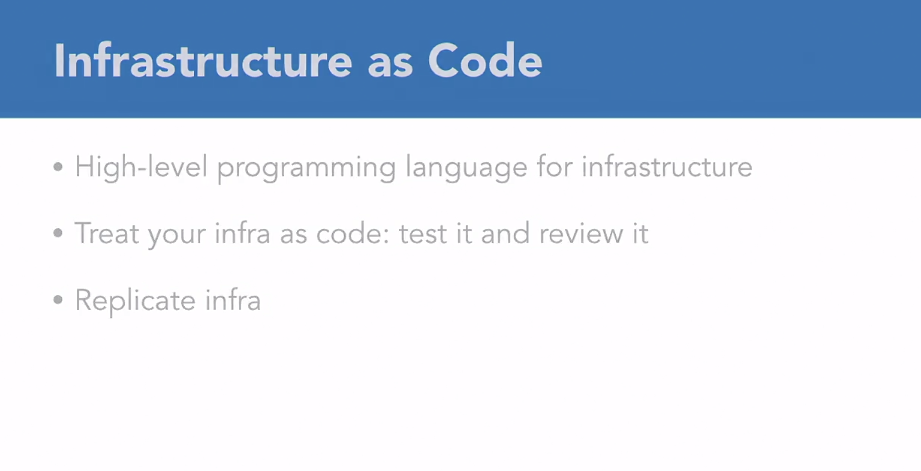
- [Instructor] Building serverless applications directly on the AWS Console is tedious and prone to errors. Usually, this is not the way we want to build our production applications. For production applications we want to have a process that can be maintain with time and also replicated in different environments such as testing, development and production. For building serverless applications for production using infrastructure as code is a recommended practice. I will discuss why this is so important later on. We can implement the infrastructure as code practice in many ways while building serverless applications. One way is to code the infrastructure directly and package the scripts and deploy them to the cloud. Another way is to use a framework that already has some tools to assist us in developing and deploying the applications. In this course, we will use SAM. SAM stands for Serverless Application Model. And it's an open-source framework for building serverless applications. SAM will help you to build your infrastructure as it has a simple notation to create functions, APIs and database tables that are quite common resources needed in serverless applications. The infrastructure is defined in a YML file. And then it can be deployed with the help of SAM to the cloud. SAM syntax will get transformed into CloudFormation during the deployment process. CloudFormation is the AWS infrastructure as code syntax for defining all the AWS resources. This is very portable within different accounts and maintainable over time. SAM comes with a rich set of features available to developers. The SAM template will contain all the definitions for the right components of the serverless applications and their configuration. All these can deployed as one single entity. SAM integrates with many AWS serverless tools like Cloud9 that is an Online IDE, AWS CodeBuild, CodeDeploy and CodePipeline that they are tools to build CI/CD pipelines. Local testing and debugging cloud applications can be hard, but SAM provides tools that make it easier. Developers who starting out with serverless loved this feature about SAM. SAM syntax is translated to CloudFormation during the packaging time. CloudFormation syntax can also be added in some SAM templates and that gives a lot of flexibility. T**o work with SAM and develop serverless applications, it is recommended that you use the AWS SAM CLI. SAM CLI is a command-line interface that supports building SAM-based applications. SAM CLI supports local development, testing and you can choose between multiple languages to work with your SAM application. In our project, we are going to use SAM and SAM CLI.**

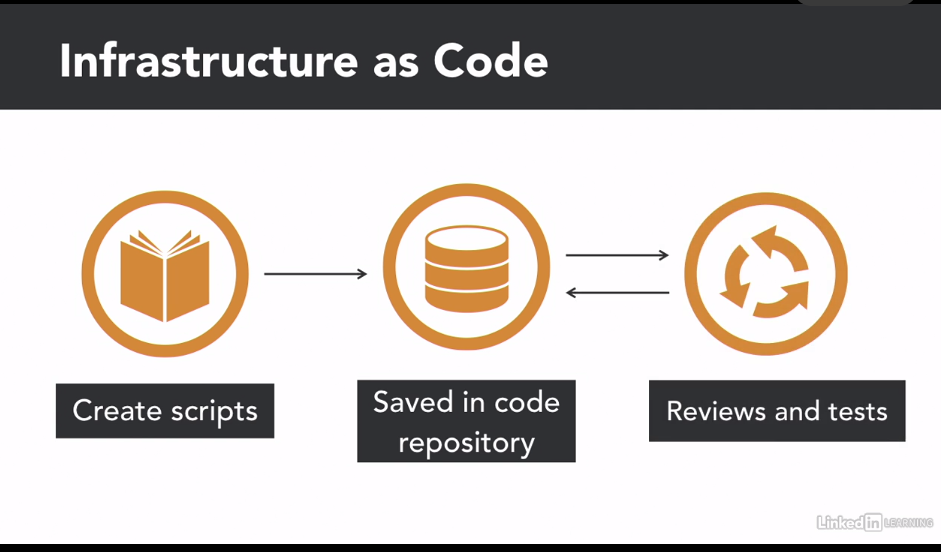


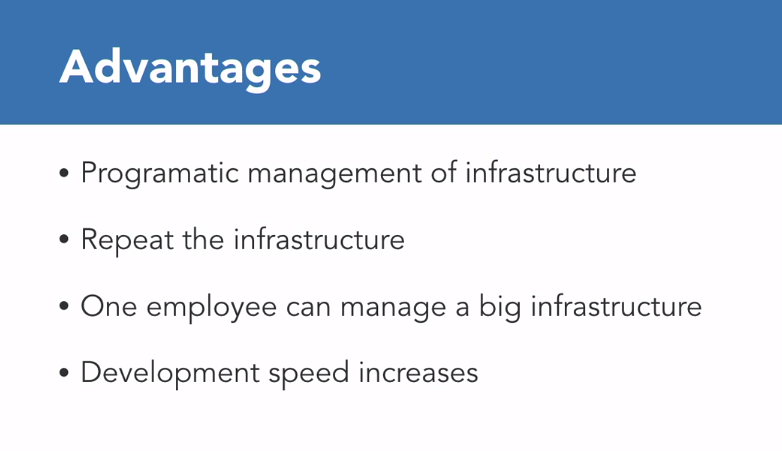
### **Why infrastructure as code is so important**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Infrastructure as code is one of the fundamental practices when working with serverless applications in production. I want to take a moment to analyze why this is so important. Let's talk by answering the question, 'What is infrastructure as code?' Infrastructure as code involves using a high-level programming language to control the infrastructure of IT systems. When your infrastructure can be treated as code, you can start applying the same techniques to the infrastructure that you apply to code, like testing, code reviews, automatic testing, and so on. Additionally, when the infrastructure is in a coded form, it can replicated many times, minimizing errors. All of this will improve the quality of your infrastructure. Infrastructure as code means that you're not going to the console of your cloud provider and different tools and type and click check boxes to create our infrastructure. It means that we are going to create scripts where the whole infrastructure gets defined. Later, these scripts will be saved in the same repository as the code is. And it will be put through the code reviews and tests. And as with code, when the scripts are changed, we will register who made the changes and why. This practice is fundamental for cloud development, microservices, and for serverless. Some advantage of infrastructure as code are the infrastructure can be managed in a programmatic way. No need for manual configurations that can generate many errors. It is possible to make infrastructure that can be repeated in many different environments, such as production, testing, and development. One employee can manage a massive infrastructure. Development speed is gained and now you can reutilize pieces of your infrastructure you had already used in other projects and evolve them. Infrastructure with more security. The chances of creating bugs gets reduced. When we are building our serverless projects with SAM, we are defining our LAN drives, APA gateways, dynamaterials, and our resources in the code. We are using YAML to find these resources. We don't need to go to the AWS control to create the resources. We can store all the code in a code repository. We can even replicate this code in multiple AWS accounts.







### **Setting up SAM**

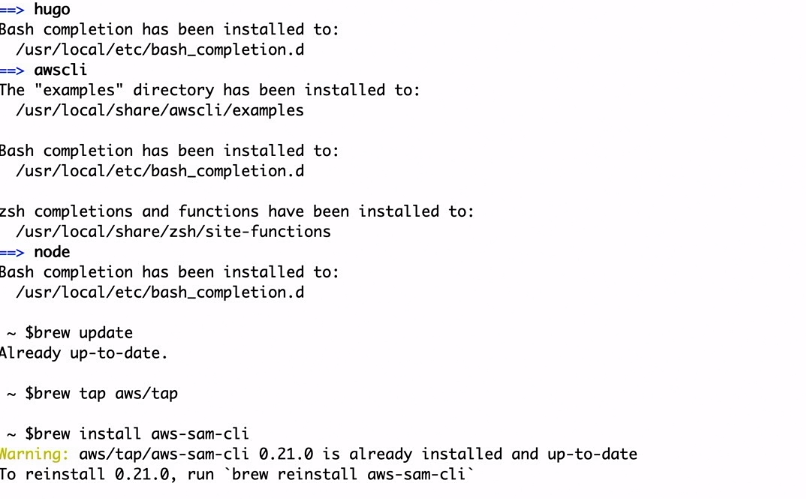
Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Now that we know a little bit more about Sam and why it's important to use a framework, let's go and install all the different products that we need for installing Sam. Some of these you might already have in your computer, so you can skip those steps, but make sure that you have them before moving onto installing Sam. The first thing you need is to have Docker. So make sure that your Docker is running, so you can see here that Docker is running. If not, then go to this URL and follow the instructions to install Docker in your computer. If you're working in a Windows environment, you can look for a little whale symbol in your notification area. If that is there, it means that Docker is running and you can access it from a terminal. The next thing you need to do is to have Python. And to make sure that you have Python, you can go to your terminal and you can run Python version. And if you have 2.7 or above, then you are fine and you can move on. If not, go to this URL, Python.org and follow the instructions to install Python in your machine. The next thing you need to have is pip. In general, if you have Python installed, you will have pip. Pip is the Python package installer and you can verify that you have it in your terminal by typing pip version and it will appear the version of pip that you have. And if you don't have it, just go to this URL and follow the installation guide. The last thing you need to have in order to install Sam is the AWS Command Line Interface or CLI. And for that you have two ways to install it using pip that you can run pip install awscli or then you can use brew by doing brew install awscli. If you want to make sure that you have installed correctly the awscli, you just do in your terminal aws version and the version will appear. So that's everything you need before installing Sam.

### **Create your first project with SAM**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Now that you have all the requirements to install Sam, let's proceed and install it. I will follow the steps of brew, but you can follow the pip installation as well. So if you want to run it with pip, you will do pip install aws-sam-cli. If you're in a Mac, you can use brew. And for that the first step is to do brew upgrade and this will take a while, while everything gets upgraded. So it's upgrading all your packages and that's important. So ev**erything that is used in brew including my awscli, so it's putting the latest version. After it completed, then we type brew update to get all the latest versions of the different libraries. I already did it, so everything is up to date. And finally, we are going to start installing, so we are going to do brew tap aws/tap and there we will be able to see Sam. So now we are in that tap and we can do brew install aws-sam-cli and this will bring the latest version of Sam in your computer.** I already have installed it, so it's already installed, but if you don't have it, then it will be installing for you. And then to make sure that you have the latest version, you can do sam version and you will get the latest version of the CLI. The next step is to configure the credentials in Sam. And for that we are going back to our AWS credential file that we downloaded when we created our user. So open that, after you open the file, you will see access key and secret access key. So you need that information for the following steps. Let's go to terminal and run aws configure. And now here, some questions will start appearing. The first one is the AWS access key. So we can go to our credentials, and copy the access key and we paste it here. Then it asked for the secret access key and we'll do the same. We'll go to the credentials file and we go and paste it here. And then we can decide if we want to have a default region, I don't know, that's up to you. We are using in the example, Ohio. If you are using one particular region, you can put it here, but I will leave it empty for now. And this you don't need to put anything as well. So if you want to make sure that everything is properly configured, opening Visual Studio Code, your credentials files, it will be in the folder inside your own user AWS credentials. And then you will see that you have configured correctly your credentials for the account. You will the access key and the secret in here. So after you have done that, then you are ready to start working with Sam.



### **Create a Lambda function with SAM**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Now that Sam is installed in our computer, we are ready to start with our first Sam project. For that, we need to create a directory, where we are going to put the files for Sam, so, we'll make a new directory called "First Sam Project" and then, I will get into it. Inside there I will create this Sam project and for that I can use the Sam CLI to initialize a project for me, so, I can type "**$sam init --runtime"** and then I can select which runtime, we are going to **use nodejs10.x** for this, and if I press "Enter", then Sam will do all of the things that you need to do in order to create a Sam application for me so, I can open this in Virtual Studio code and we can go and check it out. So if we go to the structure of the files that the "sam init." created for us, we can see, we have a template. Here is our infrastructure, there's our "README" file and our "gitignore", and then there is "hello-world", that is our application. So, if we open the template, we will see the template information. The first thing that we can see, some information about the template itself, and then we can start seeing information about the functions. Here is the timeout for the functions, Lambda functions can run for fifteen minutes, but, in this case, we are going to time it out at three seconds, so, if the function doesn't complete in three seconds, it will give a time-out error. The next thing we can see is the resources and here, we can see the function, this is the resource we have, and this is the type of this resource that is a server-less function. The template comes with a lot of help, so I recommend that you go to the links that the help is suggesting so you can see more details about each of the parts. Then these functions have properties and some of the most relevant properties are, where is the handler? It'll be in this application "app." handler method. What is the runtime? And then what are the events that trigger this lambda? By default, the lambda that comes in the "sam init" project because this is the lambda that comes here, will have an API gateway trigger so, we can see that this is the name of the API and this is the type. It will be an API. It will be an API gateway and the properties, it will be in the path "hello" and the method "get." Here, we can see that the API gateway that is created by Sam is more specific. In when we create it in the console, we have this "any" method and in this case we will get a "get" method. Now we can go to the "hello-world" folder, and inside here, we can go to the "app.js" that is another important file that is created for us. Here, it has a lot of information that you can find. For example some documentation on the event object and how everything is integrated. If you are interested you should go and check this information. It's good to know and it will bring more details on how to develop lambda. So, here we can see in line 17 that "lambdaHandler", the method that we were defining in the template. The "app.lambHandler" is this one in the app file the method lambdaHandler. So basically what this does is just create a response with status code 200 that returns "hello-world". Nothing more complicated than that. So, when we deploy this, we will get hello-world in our screen when we try it with (postman?)

### **Add an API Gateway trigger to Lambda**

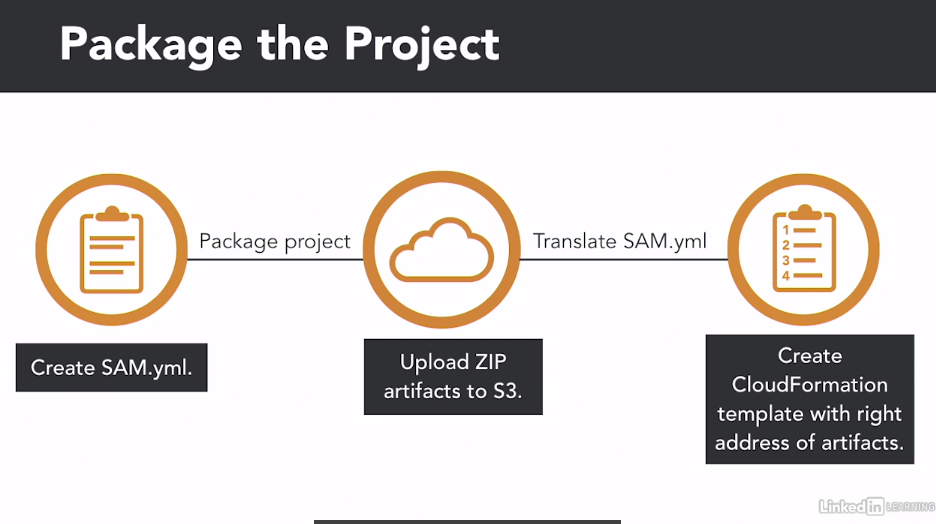
Selecting transcript lines in this section will navigate to timestamp in the video

- Now we have installed SAM and we have our SAM project that we have created in the video before. The next thing that we want to do is to create our own function and apa gateway. So what we are going to do is we are going to the template demo that we have and we are going to create our first function there. Well, second because the first one is HelloWord function but it was created automatically by the initializing of the SAM project. So we are going in resources under Helloworld function we are going to start typing our own function. We are going to collect clock function. And this function I want just to display the time so I want it very simple. If you are working with yemel its very important that you respect the indentations because if you have something not correctly indented then it might not work. Well it will not work. So the first thing we want to do for our function is to define a type and then we want to find some properties. The first property is the CodeUri and here we are going to put clock. If we look at the CodeUri for hello-world its hello-world and that means this folder where all the code tests and everything is inside this folder so we want to have similar thing for our clock function. The next thing we want to do is to find the Handler. For our case I want to have a handler file with the MEDock clock then we define the Runtime, that is nodejs and then we are going to start defining the events that will trigger this lambda. We only want one event that is an apa gateway so I will have one api ClockApi from the type Api it's an api gateway with some properties that will be the path clock and the method get. So this is the definition for our clock function. The next thing we want to do is to create the clock directory. So I just did make there clock and now it appear in our folder. And then I will go inside there and I will do npm init and this will initialize the node baggage inside that folder. So now if you go inside the folder we will have the package json so if we have any dependencies in this function they will be in there. And the next thing finally that we want to do is to create the file handler. So it's just the touch handler js and that will create a handler MEDock in here. The handler MEDock is empty so we need to start writing things on it. So the first thing we want to do is to create the clock MEDock exports. And here we need to be very careful that the handler that we define in the clock function is exactly the same name as the one that we are creating here. So we said in a template in the handler that it will be in the handler file in the MEDock clock and now we are in the handler file and we are creating the MEDock clock. And now we just write the definition for this method and we start writing our code. So the first thing I want to do is to put a message in the log so when we see the logs we know that this function runs so we can put clock function run. And then I want to display the time. For displaying the time I want to use a library const moment. This a very nice, simple library. For javascript to display that and do all kind of things with the dates so I like that library a lot. So what we need to do is to install that library in our clock project. So we are going to npm, install, save, moment. And that will download the dependencies and put them in the package json here. So then when lambda is executed it can download these dependencies. So now we can use these and for that we are going to write the message that will be just the date with the basic format. Nothing fancy but if you want to play with this library it's a very fun library to try, so go ahead and play with it. And the next thing we want to do is to create a response with a status code of 200 and a body that is JSON with the message date that we just got for a moment. So it is a very simple response. And then we just return it to apa gateway. So this lambda is extremely simple. But you should save everything and let's build it.

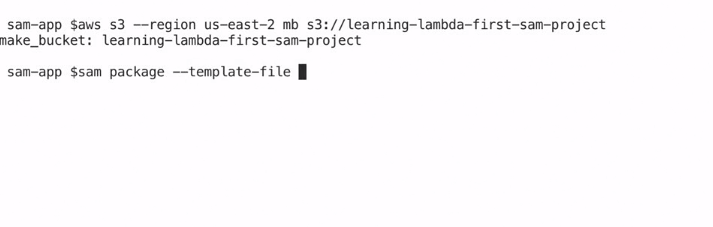
### **Package with CloudFormation**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Now that we have our code ready, we need to get it up in the cloud. For that, we need to do some things to get it up there. First we need to build the project. Then we need to package it, and finally we need to deploy it. What is involved in building the project? We want to get all the reference to third party libraries and packages into our project. We want to install all the libraries that we need. In OJS we use MPM for that. In SAM we can use **sam build**. This command copies all the project source files to a temporary subdirectory and install all the dependencies for the functions. It ignores the test codes and development dependencies. So let's go to Visual Studio code and run this command. We are going to do sam build. And this is creating the temporary source and building the HelloWorldFunction and the ClockFunction in there. So if we check in the side, we have this aws-sam folder here, and inside it has the build. And inside you can find the two functions and the template.yaml as well. Now that we have built the project, we need to package it and upload it to the cloud. Cloud formation takes one file as an input, so we need to create that file with all the resources. The packaging process will first archive all the artifacts in a ZIP and then upload that to S3. That is the AWS file storage. Then we'll translate the YAML files that are pointing to the resources and it will point those resources to the S3 package resource. But before everything, we need to have an S3 bucket created. So the command here is not as friendly as some build, but it's not that complicated after you get used to it. It's sam package then we need to define the template file. In our case it's template.yaml. Then the output file. That is the YAML file that we have all the right cloud addresses for the resources, and then the S3 bucket where all these resources live. So let's go to the terminal and start working on this. So the first thing we want to do is we want to build the bucket. So to create a bucket, what we need to do is use the AWS CLI, aws s3. Then we need to find the region where this bucket is going to be created. Imagine the region many videos ago. So if you're not sure on your region, you can check it up here. And you can get the code for that region in the URL of your AWS console. There is many regions, and everybody might have a different one. Virginia is the one that comes for the folder island. I just pick Ohio. And for Ohio, it's us-east 2. I will just copy that and paste it in the terminal. And then I need to give a name for my bucket. Then I will create main bucket, s3, and here is the name. I will call it learning-lambda-first-sam-project. You cannot have the same bucket name that I have, so you will need to use your imagination and create another one. After we press enter then the bucket is created, and we can run the package command. So for that we run sam package and then we give the template file that is the template.yaml that we have in here with our functions that define there. Then the output-template-file, and here you can give any name. I will call it pck.yml. Remember this name. What I do in general is I open a new file in here. And I copy the name of the bucket because you will need those eventually, bucket. And then I put file, so when I need them for other commands, I have them. Because then you don't need to go to the console that many times. And then we need to give the name of the S3 bucket. And here now we have it in our helper file. And that's it. Now we can press Enter, and this is building. Takes a while because it's uploading all these different files to S3. And it depends on your internet connection, how long it will take. So I will stop this video here, let it upload, and when it finish uploading, move to the next video where we are going to deploy this code.



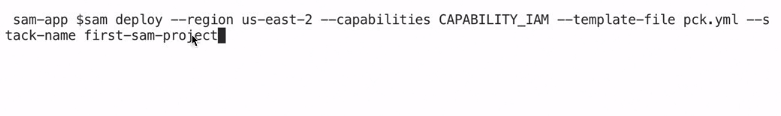




### **Deploy to AWS**

Selecting transcript lines in this section will navigate to timestamp in the video

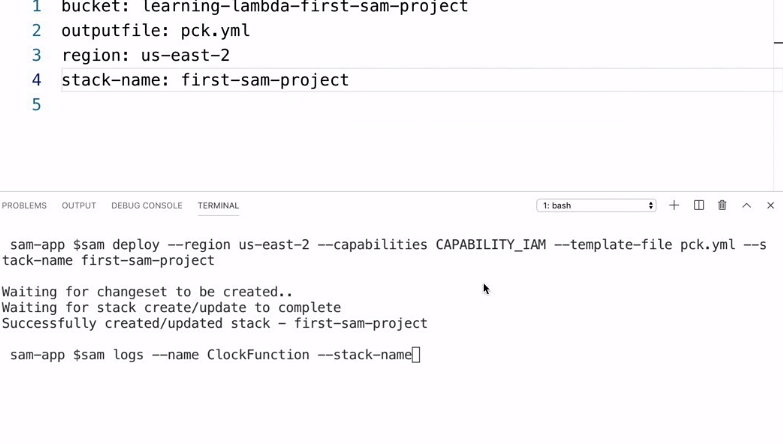
- So now we have built and we have packaged our project. The next step we need to do is to deploy it, so for that we are going to run another command. $sam deploy. First thing we do is put the region. The region is the same as the one we are working with and where we put our brackets so it's very important to remember that. Copy paste that. If you have it in your helper file or just type it there, then we need to put the capabilities and here this is the same for everybody. And then we need to put the template file. This is the output file. And the last thing we need is the stack name. In general, I use the same name as my project so then I can find them easily. first-sam-project if you see here, and here it has the same name. Press Enter, and then we wait. If we go to AWS console, we can open the service CloudFormation, and in here we will start seeing that there is one stack, first-sam-project, that is being created at the moment. If you go to resources you can see that there is some functions. First in roles, and then the functions we'll have here, an API gateways will up here as well. So, it takes a while to deploy and you can keep an eye in what is going on in CloudFormation to see what is being created. And when it's completed, then we will get back the cursor, and if we go to CloudFormation we will see that now there are 11 resources that were created with this deployment: The function, some API Gateway, and permissions and roles. All kind of things that we will take a look at it next.

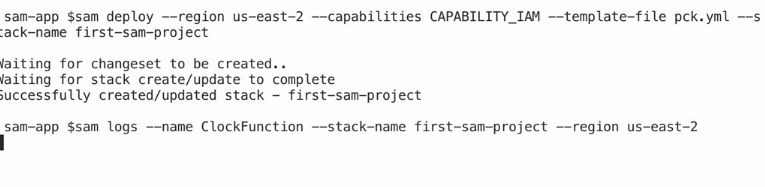


### **See function logs with SAM**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] So now that we ran our Lamda with Postman, we can check the logs for that. We can go to the Lamda page, hit Monitoring, and View logs in CloudWatch. Or, there is another way. We can use Sam for this. So let's go back to visual studio code in our terminal, and we can type **Sam logs**. And then we need to put the name of the function we want to get the logs. And that you can get from your template gamma and the resource here, Clock Function. The next thing we need is the stack name. And that one, you can either put it in your helper file here. Then I have this file here, the helper file, and things that I might need. And that one. Just remember that is the same as your project so you don't need to worry too much. Sam Project. And finally we need to have the region. And that one we also have in our helper file, or we just remember it. We click enter, and it takes a little while, and the logs will start appearing here. So we can see the Clock Function run, and it run again, so we run it a couple of times. And that's what we get. If we want to see the lives logs, we can do tail. Write the same, Sam logs, the name of the function we want to monitor, the name of the stack, the name of the region, and then you put tail at the end. And this will leave the logs open, so whenever we call the functions... so let's do that. Let's call it. And then we will start seeing the logs appearing here. Well, they appear faster than I can move. They start at START, and it takes a little while to refresh. Here we see the Clock Function run. So if we keep on executing this, the logs will tail to this. So it's a very convenient way to see the logs while you're trying to figure out a problem in your function. You don't need to run Sam logs all the time, you just leave this open. The logs will tail and you can start your code live.





### **Challenge**

Selecting transcript lines in this section will navigate to timestamp in the video

(upbeat electronic music) - [Instructor] This challenge should take around 20 minutes to complete. The objective of this challenge is that you add a new function that gets triggered with an API Gateway to our existing SAM project. Then you can build it, package it, deploy it, and test it in the Cloud. The function should take the name of a timezone, like "Americas, New York" or "Europe, Helsinki" from the HTTP request, and then return in the response the time that it is right now in that timezone. If there is no input, there is no timezone. Then the time that will be returned is the time in GMT 0. For helping you out, I recommend you that you use this library called moment-timezone that will make your challenge very simple, because the library provides a lot of helper tools for managing times with timezones. The URL to find more information about this library is on the screen. So, here's an example of the request and the response. So, this is an example when there is something in the timezone name. So, we have a GET metered, the HTTPS with the URL, back slash, convert time, and then we pass the name of the timezone, and when that's the case, we get back as a response the time, and at the end, we show the plus 8 that is the timezone difference from GMT 0. And then, if we don't have anything in the timezone, we have not specified which timezone, we will have a request that looks like this. GET and our URL, the convert time, and we will get back, as a response, the time in GMT 0 is that, plus zero zero, that is the timezone for this. You can find the solution to this challenge in the next video.

### **Solution**

Selecting transcript lines in this section will navigate to timestamp in the video

(upbeat music) - [Instructor] So, the objective of this challenge is to create a new function with an API gateway. So, let's go to the template yaml where we have defined our other functions, we have the HelloWorldFunction, we have the ClockFunction, and under the ClockFunction we can define the new function that we will call ConvertTimeFunction. Remember that we are using yaml, so you need to keep the indentation. The first thing we are going to do is to write the type, AWS::Serverless::Function, and then we are going to type some properties. The first property is the CodeUri, so where the code will be located, convert-time, we want to put it in this folder, where the Handler will be the handler.convertTime method, the Runtime will be nodejs10, and then the Events, we want it to be triggered by an API, that is an API gateway. ConvertTimeAPI, then the Type is API, and the properties are Path, that is /convert-time, and the Method is get. So, this is our definition, so the next thing we need to do is to create the convert-time folder, so let's do mkdir convert-time, and now we have this little one here, go inside, and do npm init, yes, and now we have our package.json and then we do touch handler.js, and we have our handler file. In here we need to define the method, exports, and the name of the method needs to be the same as the one in the handler definition, so I usually copy paste it, so I make sure that everything is okay, event, and this is the signature of the method, and I want to write something in the log so if this runs then this is what you'll see, we have something in the logs. So now we have these two cases we will get our timezone from the URL, so we need to grab that. So, let's do timezone, we have already passed queryString parameters into our functions from the HTTP request to the API gateway to the function in previous challenges and examples, and we are passing it with the name tz. And then we want to formatDate, I would like to call it formattedDate, so basically, if I run this, it will just return the date in the GMT zero. So now I have the basic results, so let's do this, if the timezone is null, is not null, then we are going to change the formatted date into something related to the timezone that we got back, and format. So, this line of code returns us the timezone in the timezone that we passed, so super simple, and if it's new then we want to set the timezone name to "GMT 0". Good! The next step is to create the message that we want to return, and that will be "The time in ", the timezone, " is ", the formattedDate, as simple as that, and then we want to create the response. So that should be all the code. It's important not to forget to require the library, let's call it moment, require moment-timezone, and then also to do npm install save moment-timezone in our project so we get the dependencies in our package.json, that's very important, so then we can use them in our code. So, we are getting the timezone from the queryStringParameters, we are formatting the data when there is no timezone, and if there is a timezone we just format it with the timezone, and then we will find the timezone to be GMT zero if there is nothing, we'll write a message, and then we'll put it in the response of our HTTP. So, now we need to build this. It's important that if you're building, go to the top folder, don't stay in that convert-time. Do sam build, and now it will let you, and you will see that the new function will appear. Then we want to run sam package, template-file is the template, the output-template-file, you can open your helper file and use that, and the s3-bucket, you don't need to create it again because you already created it so you can use the same, and then run that. The helper file you see in my project has a specific configuration to my AWS account, you don't need that file, that's why you won't find it in the exercise files. We need to wait for all the files to upload. So, there's three, and it depends on your internet connection, so. Good! So now there is the package, and it's ready in s3, so it can run sam deploy, region, the one that we are using, and then we can do capabilities and then the template file, that is the one that we just created there, and this tack-name, that is the name of our project that I have in my helper file. I press enter and then the deployment will start, and now we just wait for it to complete. Great! Now the application is deployed, we can go to our AWS Lambda console. Lambda, and now we should see the new function here, first-sam-project-CovertTimeFunction and a weird number, this is our function that's just been deployed, it has the code that we wrote and it has an API gateway, and we can get the URL from here and go to post and try it out. So the first thing we want to try it is with nothing, so we just run this. So the time in GMT 0 is this one, in 11:18, and then if we put some timezone, let's put America, New York, it's really quite the same as the moment timezone has, so it doesn't break, and there it's 7:18 and here is the timezone, minus four. We can write another timezone here, and this case it's Europe, Helsinki, we can remove the first one, and here you can see, and here it should be 2:18 in GMT three, so if you try different timezones then you will get different times back. So, that's the whole challenge, I hope you enjoyed it.

### **Clean up your AWS environment**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] When you finish this course, or you learn to stop working on this course for an extended period of time, I recommend that you clean up your environment. Clean up the environment means that you delete all AWS resources created during the course, including the functions, AWS gateways, everything. Cleaning up the environments will also remove the endpoints, so that if you really employ this application later on, you will get new orals for your endpoint. Cleaning everything is as easy as writing in your project terminal, aws cloudformation delete-stack and the stack-name. This will remove your confirmation stack from your AWS account. Removing all of the resources within it. When deploying this application, it's as easy as redeploying this to cloud formation as we saw in the previous video. Other resources will be recreated with different ARN, Amazon Resource Name, but the whole application will be working as before.

### **Clean up your AWS environment**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] When you finish this course, or you learn to stop working on this course for an extended period of time, I recommend that you clean up your environment. Clean up the environment means that you delete all AWS resources created during the course, including the functions, AWS gateways, everything. Cleaning up the environments will also remove the endpoints, so that if you really employ this application later on, you will get new orals for your endpoint. Cleaning everything is as easy as writing in your project terminal, aws cloudformation delete-stack and the stack-name. This will remove your confirmation stack from your AWS account. Removing all of the resources within it. When deploying this application, it's as easy as redeploying this to cloud formation as we saw in the previous video. Other resources will be recreated with different ARN, Amazon Resource Name, but the whole application will be working as before.

### **Next steps**

Selecting transcript lines in this section will navigate to timestamp in the video

- [Marcia] That bring us to the end of the course. At this point, you should be able to create Lambda functions that get triggered with API Gateway and also have the capability to start learning other triggers for Lambda. In addition, you also got introduced to AWS SAM and infrastructure as code. This isn't the end of your learning path. I encourage you to keep experimenting and learning about these topics. Check my YouTube channel where I post new serverless videos every Tuesday and don't forget to check back here often as we continue to add more AWS and serverless courses to our LinkedIn Learning library. Again, my name is Marcia Villalba and thanks for watching.