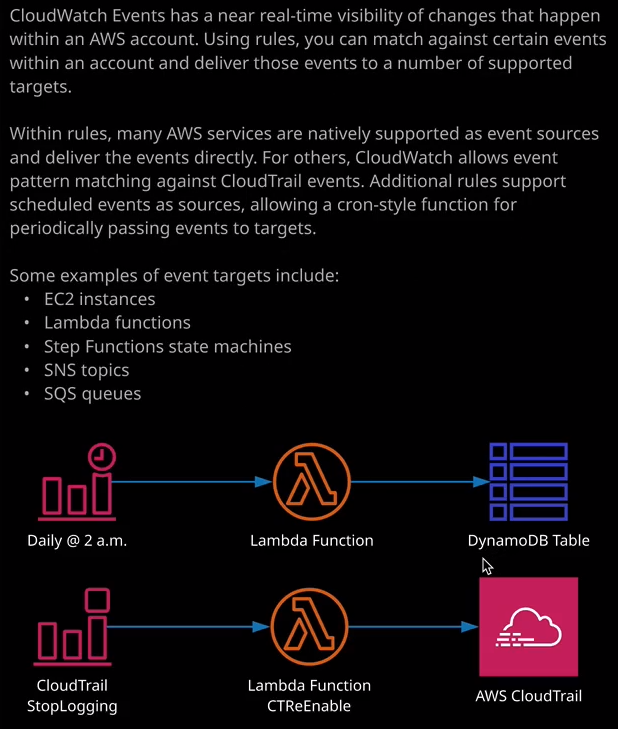
CloudWatch Events. So this is another subcomponent of CloudWatch but it's a really important one because it helps you design a lot of automated infrastructure inside of AWS. So understanding CloudWatch Events is simple enough.



**It's a product which is able to see all the events which happen inside an AWS account. So an event could be an instance stopping, a security group being changed, a CloudTrail trail being switched off, a new user a being added, pretty much anything that can happen inside an AWS account CloudWatch Events will see**. Now, in many ways, you could compare this to CloudTrail in the sense that both of them have visibility to events which occur inside of AWS. **The difference between CloudTrail and CloudWatch events is that CloudWatch events can take action based on what's happening. So CloudTrail is simply an auditing tool. It generates an overview of all of these API calls and when you configure a trail, it allows you to do something with that information. CloudWatch Events receives this overview either using the same backend infrastructure that CloudTrail uses or by getting events delivered to it directly from supported services**.

Now **CloudWatch events also has one of the crucial difference from CloudTrail, and that's that it can see these events as they occur in almost real time.** So if something happens to an EC2 instance then CloudWatch events is going to see it immediately as it happens, and it can take action based on that event. Now, **CloudWatch Events is based on rules. It's a rule that allows you to see and then, as it happens, from an event source and take an action by delivering the event to a target**.   
Now, **an event source can either be based on an event pattern. So it's essentially a pattern match against an AWS service and a lot of AWS services have been implemented to directly integrate with CloudWatch events**. You've got alot of services that directly support CloudWatch events and can put events directly into the product. But for any services which aren't directly supported, **you're able to do a pattern match based on the API events, and if you want to do that, you need to have CloudTrail enabled.** So CloudTrail is the thing that monitors API events and in order to be able to get access to all of those APIs events, CloudTrail needs to be enabled. So we've got event patterns as one possible way to initiate a rule, but we've also got **schedules**. So schedules are a way that you can initiate a rule or invoke a rule either based on a fixed amount of time, maybe every five minutes or every X hours or every X days or you could use a Cron expression, which, if you created any expressions in a Cron tab file before on a Linux machine, you'll be used to this format. It's really flexible, and it allows you to define using the Cron format, that an event will occur at a certain time period, maybe once per day, maybe once a week, maybe a couple of times a week on certain days. You've got a lot of flexibility with the Cron expression, and you can use that to specify exactly when you want this rule to be invoked. When the rule is invoked you can take a certain action.

Now, **one of the common things that I see CloudWatch Event rules being used for, especially using a schedule, is you can set it to power off EC2 instances**. So let's say I wanted to add a target and I'm going to change from Lambda function, which is the default and I'm going to change it to EC2 stop instances API call and then I can specify an instance ID. So you might use this to power off any EC2 instances which don't need to be online outside of working hours and that's a pretty common use case that I see for CloudWatch Events. CloudWatch Events is essentially the glue that allows you to receive events from sources and configure the delivery of those events to targets and those targets could be EC2 instances, Lambda functions, step function, state machines, SNS topics. It could be adding a message onto an SQS queue for future processing. It really is a super flexible service.

Now, again, at an associate level and for the associate exam, the only thing you need to know is the architecture of the product and exactly how you could implement it. You don't need to be aware of the exact implementation details. Just have some awareness that **you can create event patterns or schedules and then have those delivered to targets, maybe stopping instances, starting instances, invoking Lambda functions.** So what I would recommend is just explore this user interface and look at all of the different possible targets that you could use for these events and if you do that, you'll have enough information to be able to answer any exam questions.

Now, what I want to do quickly is to just give you a quick demonstration, because a practical demonstration will help you understand the power of this product. So I want you to imagine for a second that you're a security engineer and you've highlighted that somebody appears to be disabling a trail within CloudTrail and what you want to do is whenever that trail is disabled you want to reenable it. So this is the architecture that we're going to use. We're going to receive an event and the event that we're looking for is a stop logging API call. So that's the API call that's used when you disable a trail in CloudTrail. When we receive that event, we're going to run a Lambda function and that Lambda function is going to reenable that same disabled trail. So that's what I want to implement quickly and it's not going to be super complicated and I'll make sure that I put the source code for the Lambda function inside this lesson's folder in the course GitHub repository. So I'm going to go ahead and implement that. So the first thing I'm going to do is create a role that the Lambda function is going to use. So I'm going to go to "Services" and then "IAM." Once I'm there going to go to "Roles" and I'm going to create a role. Now, it's going to be a role that Lambda is going to be able to assume on my behalf. So I need to make sure AWS service is selected then go to Lambda and then next permissions. Now because the main topic of this lesson isn't permissions, I'm going to use some managed policies. So first I'm going to search for Lambda. I'm going to look for the AWS Lambda basic execution role. So this provides a Lambda function with the ability to log any information when it gets executed into CloudWatch logs. So this is a managed policy that you should always give to every Lambda function. So I'm going to go ahead and select that. Next I'm going to search for CloudTrail and I'm going to give this Lambda function AWS CloudTrail full access because the Lambda function is going to need to reenable any trails that get disabled. So that looks good. I'm going to go to "Next: Tags" "Next: Review." Just to verify that this role has both of those managed policies, which it does and I'm going to call the role Lambda for CloudTrail and I'll hit, "Create Role." Okay, so that's the role that's going to give the Lambda function enough permissions to be able to log to CloudWatch logs and to interact with CloudTrail. So next I need to go to Lambda and actually create the Lambda function. I'm going to go to "Create Function." I want to author it from scratch. The function name will be CloudTrail reenable select Python 2.7 as the runtime. I'll expand the permissions dropdown. I'm going to choose an execution role. I'll select use an existing role and then I'll select the Lambda for CloudTrail role that I created earlier. Once I've done that I'll go ahead and click on "Create Function." So inside the folder for this lesson inside the 08 Application Analytics Operation folder, there's a folder called operations and inside operations, there's a file called ct.reenable.py. So I'm going to open that file. I'm going to copy the contents into my clipboard and don't worry, I'll make sure I include a link in the lesson description if you want to follow along in your own environment. I'm going to come back to the console, scroll down and then I'm going to replace all of this existing Python code with the contents of that file. I'm just going to spend a couple of seconds just stepping through this. You don't have to be a developer. It is relatively simple Python code. The first three lines just import some libraries. One of them is Boto3, which allows us to interact with AWS. One of them is JSON which is just some JSON libraries and the other is the system library, which tends to be standard, so we can almost ignore those lines. Now, remember this is going to be invoked whenever an event is received of a stop logging API call. So what this is going to do is it's going to retrieve the data of the actual CloudTrail name from the event data that it receives. So when CloudTrail invokes this Lambda function it's going to be passed information about the actual event and that's going to include the name of the CloudTrail that's been disabled. So all we're going to do is retrieve the name of the CloudTrail that's been disabled, which is what this represents. Then we go to initiate a start logging API call and we're going to pass in the name of that CloudTrail. So essentially this whole function gets executed when the CloudTrail is disabled and then it runs a command which reenables the very same CloudTrail. So it's nothing complicated. So everything else looks good. I'm going to go ahead and save this function. Now that that's saved, I'm going to do the last piece of configuration. I'm going to move back to the CloudWatch console and using CloudWatch events, I'm going to configure this integration. So I'll go to events and then rules and create rule. Now, it's not going to be a scheduled event because I want this to run when a particular type of event happens, I don't want to do it on a scheduled basis. So it's going to be event pattern. Then I'm going to drop down the service name, select "CloudTrail" because I want it to be based on a CloudTrail event and then for the event type, I'm going to change it from all events to AWS API call via CloudTrail because I want a specific API call. I want the stop logging API call, which is the event that's called whenever CloudTrail logging is disabled. So I'm going to go to specific operations and then in this box, I'm going to type stop logging. So that's the name of the API call that gets called whenever you disable a CloudTrail. So that's the source part done that's going to detect whenever the stop logging API call happens against a particular trail. The next component, and now we've got the event sorted is we need to configure the Lambda function so that it gets invoked and the event detail is passed to it whenever this rule executes. So I'm going to go to "Add Target." It's going to be a Lambda function that I'm going to run, I'm going to select this dropdown and just select "CloudTrail Reenable." Now that's done, the final thing I need to do is click on "Configure Details" and give this rule a name. So I'm going to call the rule re enable logging and just make sure that it's set to be enabled. Finally, I'll hit "Create Rule" and that will configure all of the integration. The rule is now active, and any time that CloudWatch event detects this particular event occurring, it will run this rule. It will cause a Lambda function to be invoked. The Lambda function will receive the details of the CloudTrail that's being disabled, and it'll go ahead and reenable it. So the next logical thing to do is to test it. To do that, I'm going to move across to the CloudTrail console. Now, I've already set up a CloudTrail. It was the one that I used in a previous lesson to demonstrate the functionality of CloudTrail. So open up the LATrail that I created in an earlier lesson. I'll switch the status from logging on to logging off. It'll ask me to confirm it and I'll select "Continue." So the logging for this CloudTrail has now been switched off. What's going to happen in the background is the CloudWatch events rule that I just created is going to be initiated because it's going to detect this API call. It's going to take the event details, invoke the Lambda function, the Lambda function is going to act on that. So it's going to identify the CloudTrail that's been disabled. It's going to run the command to reenable the CloudTrail and so what should happen if I hit "Refresh" and what we should see is that logging has been reenabled on this trail and there we go logging is switched to on. So this is just a simple example of an automated workflow using CloudWatch events. We've configured an event. We've configured an event being detected. We've configured a pattern. We've got that event sent to a particular target, which is the Lambda function. The Lambda function accepts the event, passes the details, and then reenables the CloudTrail logging. It's a very simple example but it illustrates the power of CloudWatch Events. So if you see any questions in the exam which talk about events which talk about patterns or automatically remediating or automatically responding to certain actions, then you should default to thinking CloudWatch Events. It's a super flexible product, and in most cases it's going to be the correct answer. With that being said, this is associate level and for the associate level solutions architect exam, you're not expected to be a developer, and you're not expected to understand the implementation details. What you need to focus on is the architecture of CloudWatch Events, understand each step of what I've just done, exactly how each of the components fits together, how CloudWatch events has visibility of these account level events, how it captures it, matches it, runs a rule. The rule has the event source, the target. The target is a Lambda function it's invoked and it performs an activity. And if you can understand each of those individual components, then you'll have plenty of knowledge to answer any exam questions which involve CloudWatch events. So with that being said, that's everything I wanted to cover in this lesson. Go ahead mark this lesson as complete, and when you're ready, you can join me in the next.