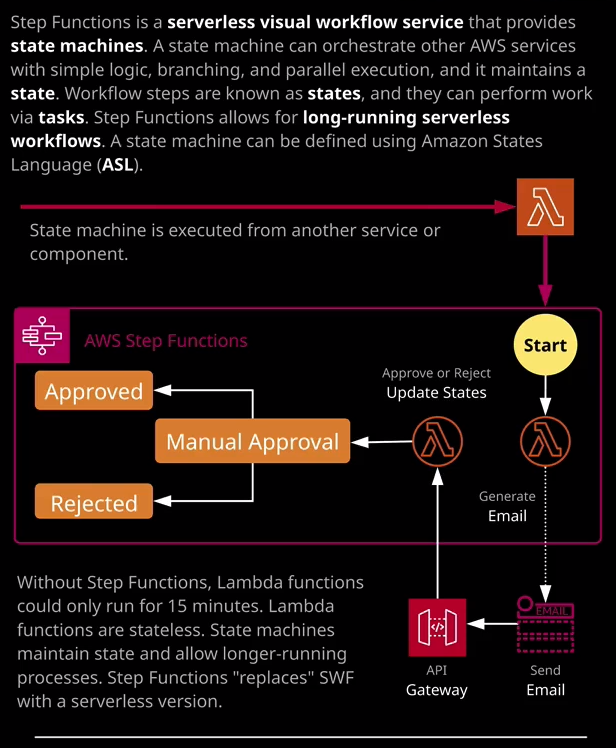
**Step Functions**

**Step functions allow you to build a serverless workflow. It coordinates other micro services to perform something bigger than they can accomplish by themselves. One Lambda function is capable of doing a lot, but it's always going to be limited to how long it can operate. Step functions address a number of key limitations with Lambda. The first is that Lambda has a fairly short runtime a maximum of 15 minutes and the second is that by design, Lambda is stateless. Now these could be considered either a benefit or a limitation, but step functions is designed to overcome the limits of using Lambda. Step functions provides an entity known as a state machine and a state machine can orchestrate other AWS services with simple logic. State machines can be started based on other events, so they can either be executed manually or executed by other AWS services and state machines can contain step by step processes, which can include user interactions and state machines could be operating for up to one year. State machines have a start and end and a flow of actions and decisions in between.** Put simply, state machines allow multiple Lambda functions to be coordinated to do things that one alone couldn't accomplish. You might want to run two functions in parallel and have their outputs fed into third or have some form of choice or multiple path architecture, depending on input or other data. Before step functions this would take a lot of complexity. You need to code this yourself and potentially have a chain of long running compute resources. You might even need to use EC2 but by using step functions, this isn't required.



**With step functions you can define these visual work flows and have them implemented in a serverless way.** Now, a very simple example of a useful state machine workflow might be a user approval process. So this might be a manual process that comes as part of a code release or a change to infrastructure. So you might have some sort of process in your company that requires manual approval, either by a manager or a member of the change team. This is the architecture that I'm going to implement to demonstrate step functions within this lesson. It's going to be a pretty simple demo. Essentially, this state machine is going to be executed. It's going to start at the starting point. It's going to execute a Lambda function that uses SES or simple email service to send an email. At this point, that state machine is going to stop operating and this email is going to be the thing that forces it to continue. This email is going to contain two links. The first is an approval action, and the second is a reject action. Both of these links connect to an API gateway. So they pass along this accept or reject through to the API gateway and this API Gateway uses another Lambda function to continue the operation of the state machine and based on the response in the email, it moves to an approved state or rejected state. So that's the process we're essentially just using step functions to provide a serverless approval process which has human interaction. Now, to make this a simple as possible rather than implement this manually, we're going to use a CloudFormation template. You've already covered CloudFormation earlier in the course, but because we've covered a lot of these components, so API gateway and Lambda it's getting less and less value for implementing these things manually in isolation. I'm going to use CloudFormation to implement this solution as a whole. I'm going to open CloudFormation. I'm going to create a stack. I'm going to upload a template file and the template file I'm going to pick is inside this lesson's file, so I'm going to come choose file. It's inside topic four\_serverless and the step functions folder and its step functionss.yaml. So I'm going to select that file, click on next, and I'm going to name the stack email approval. Now as part of this demo I'm going to create a state machine, and inside that state machine it's going to spawn a Lambda function to send me an email using SES and SES by default, operates in sandbox mode, which avoids spam email. So until the sandbox mode is removed, we need to approve individual emails and so I'm going to have to enter my personal email address into this box, which will be used to demonstrate this. So I'm going to enter my personal email address. Once I have done that I'll click on next. Everything else could be accepted as default so I'll click on next again. I'll need to scroll down to the bottom of this screen and take this acknowledgment. So whenever you create any IAM resources with CloudFormation, you are prompted explicitly to approve that. So that's what I'm doing here. I'm acknowledging that this is going to create an IAM role. So I select the box, click on Create Stack, so that will begin the process of creating a state machine. All of the actions and states inside if that state machine, the Lambda functions, that will be used. The API gateway will be used as well as the configuration to send the email. So this will take a few moments to complete. So I'm going to pause the video and I'll resume it once the stack has moved into a created status. Okay so that CloudFormation stack has created successfully and in the background I did need to approve my email being added as a wide list inside SES, but now that it has completed, I'm going to move across the Lambda console just highlight the two different Lambda functions that get created. So there's one Lambda function which sends the email and that's this one. If I open that up, it is written in node.js, wait for it to load, scroll down. Essentially, **this is just invoked by the state machine. It creates an email and sends that out to my address. So this is the first thing that gets executed inside the state machine and that just generates the email and sends it to me, which contains the approve or reject actions.**

The other email is the approval function, and that's the one that's used by API gateway when I click on any of the links in the email. So that's the thing that kicks off the state machine to continue execution. I'm not going to dwell too much on this because we've used these in previous lessons. I'm now going to move across to the step functions console because I want to step through the main architectural components that you'll need to know for the exam. So the CloudFormation templates created a single state machine, so let's open this up and have a look, and we'll start by going to definition. **Now every state machine inside step functions is configured by using ASL, which is known as Amazon states language.** So this is this. This is the language that looks very familiar to the JSON that's used in CloudFormation. That actually creates the state machine and we also see a visual representation that's based on this ASL. So when a state machine is executed it starts at the start and it runs until it completes or reaches its maximum runtime of one year or whatever time out you specify. **A state machine has an IAM role and the IAM role defines what exactly it can interact with from an AWS service perspective.** So, like other AWS services that operate on your behalf it does need an IAM role. So the state machine starts at the start and ends at the end and in between are all of the other states that you define. Now the start state points at the first real state that runs. So if you look at the first line in this ASL, it shows a start and then it names Lambda callbacks. So a Lambda callback is the first real state that runs inside the state machine. Now, states within the state machine will continue running until they complete, and then they'll move on to the next state, which is generally specified by this next directive so the state can inform the state machine what the next state to execute should be. So the first one is Lambda call back. The second one is manual approval choice state and then, depending on the choice made, one option is approved pass state and the other option is rejected pass state and a state machine continues in this way until the state completes, which has a type succeed, fail, or end, at which point the machine stops.

In this example, the first real state is this **Lambda callback state**, and this is actually a special example. **What this actually does is it invokes a Lambda function, and it passes that Lambda function what's known as a token. The state machine then just pauses. It doesn't continue, and it doesn't do anything else until it receives this token back, and some data and that data is then used to continue executing**. So in this case, we pass the token to this Lambda function. This Lambda function then sends the email using simple email service, and the token is stored in this email. We've got an accept and reject action. Each of those actions uses API gateway, passes that token in together with the action of accept or reject that calls another Lambda function that Lambda function passes that token back in and then we go to the manual approval state, and that's where we go to approve or reject it. So let's show this in practice. I'm going to go to executions and then start an execution of this state machine. Now the execution name I'm going to use is new cat, and the comment is, "can we have another cat?" Because, of course, everyone needs another cat. So this is a manual human approval action for getting another cat. I'm going to click on start execution. Now the execution has started and we can see this because the currently in progress state is highlighted in blue. So this is the Lambda callback state. This is the state which executes the Lambda function, which sends the email, and this is what the email looks like. It's essentially got an approval link and I reject link. Both of these point at the API gateway one has an action of approve and one has an action of reject and the rest of the information is just this token that I was talking about moments ago. So if I click on the approval link, it'll call API gateway but it'll pass in this approved action together with the token. So I'm going to do that, I'm going to click on approved. If we go back to the state machine, we can see that it's now completed. So it's moved all the way through each of these states but what I want to draw your attention to is this manual approval choice state. Remember when this Lambda callback was in progress and was essentially paused well it moved away from pause once I clicked the approval link. If I look at the output what was actually received was an approval choice. If we go to manual approval choice and look at its input then it got this message approved. So the task was approved by me. So the manual approval choice state takes two different actions, depending on this variable of status. Remember, the email had two different links one to approve, one to reject, and I told you how it passed in that data so approve or reject. Well, what this does is it looks for the input to the state, so the status variable and if it's approved, then it moved on to this next state, which is approved pass state. If the status is rejected, then it moves on to rejected pass state. So looking on the visual overview, it essentially diverges. It's going to branch in either goes to approved or rejected based on which link I clicked in the email and that that's what's occurred have essentially used step functions and state machines to perform a manual approval process involving a human being. Now being able to use step functions day to day is beyond what you need for this exam. I want you to be aware of the architecture because there are more and more serverless related questions popping up in the exam and I do think an awareness of step functions is really useful.

So from an architectural perspective, **know that step functions provides state machines. Know that state machines can orchestrate all that AWS services it provides simple logic, branching, and parallel execution and it maintains a state that's important. So workflow steps inside a state machine are themselves known as states and a state can be branching logic. It can be decisions or choices. It can end the execution or allow it to continue. It could be a task state, which can run a Lambda function so you can use other AWS services as part of this but essentially it allows you to have a long running workflow with multiple different branches, mergers, and parallel executions based on input to that state machine. So you'd utilize step functions when you need to use serverless components but you need this included branching logic, parallel execution, mergers. Whenever you're doing long term work flows that go beyond the 15 minutes of an individual Lambda function. Examples of this might be image resizing. Using multiple AWS services or using resizing combined with using other services to identify features in images while the resizing happens. So do those things in parallel or another example is what I've demoed in this lesson. You might have certain business processes that need manual approvals, and you want to manage them without using any long running compute such as EC2 and you can do that using step functions.**

**Now step functions are essentially a serverless form of an old AWS product called SWF or simple workflow service. So this was similar, but it required an EC2 instance to manage the process. Step functions could do most of what simple workflow service could do, but it does so in a serverless way. It does not require any long running consistent infrastructure. It's all handled by the product. So for the exam, anything that needs Lambda coordination anything longer than 15 minutes, anything needing large scale coordinations scalability, visual work flows then think of step functions. If you just performing small, isolated tasks that could be done in less than 15 minutes and that needs serverless than you can probably just use individual Lambda functions.**