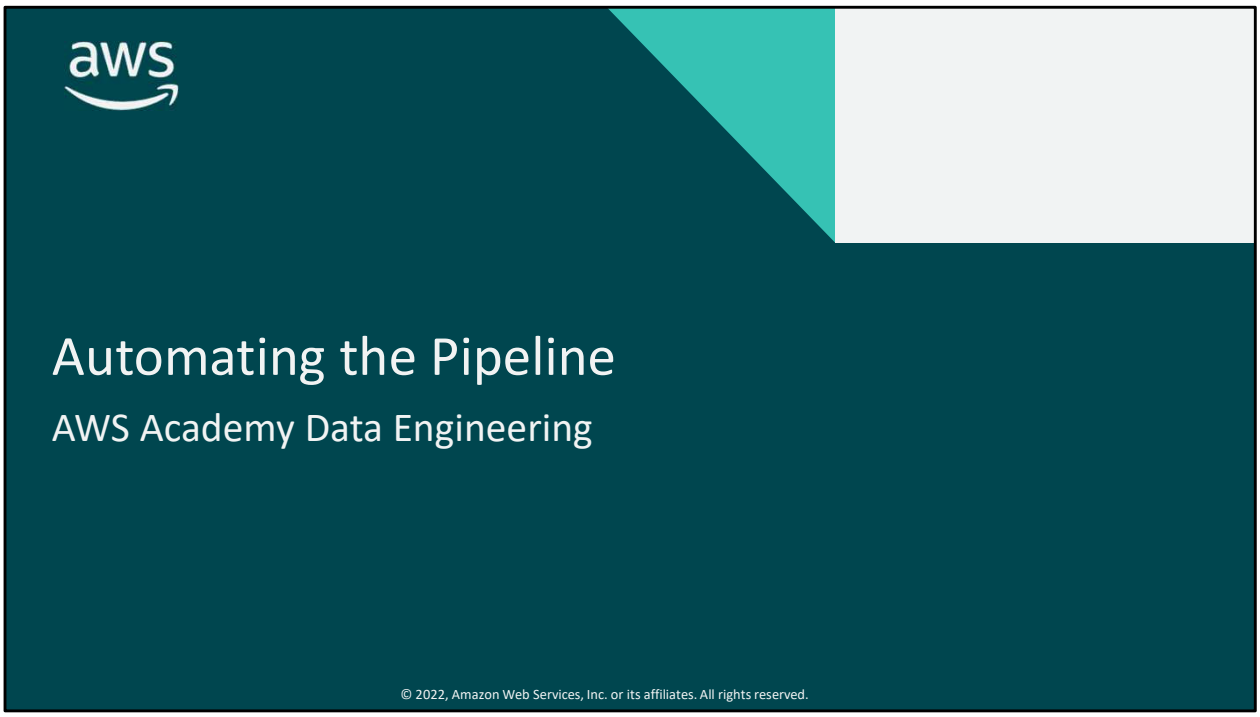


|Slide number 1
|Instructor notes
|
|Student notes
Welcome to the Automating the Pipeline module. This module focuses on automating infrastructure deployments, the continuous integration and continuous deployment (CI/CD) pipeline, and using AWS Step Functions to support data analytics and machine learning (ML) workloads.

|Slide number 2
|Instructor notes
|
|Student notes
This introduction section describes the content of this module.

Module objectives

This module prepares you to do the following:

- Identify the benefits of automating your pipeline.
- Understand the role of CI/CD and how it applies to your pipeline.
- Examine the states of AWS Step Functions.
- Use Step Functions to build and automate a data pipeline.

|Slide number 3
|Instructor notes
|
|Student notes
This module discusses the benefits of automating your pipeline by using infrastructure as code and Step Functions. You will learn about flow states in Step Functions and how to use them to control and direct your workflows. You will also examine the elements of the Step Functions Workflow Studio.

Module overview

Presentation sections

- Automating infrastructure deployment
- CI/CD
- Automating with Step Functions

Lab

- Building and Orchestrating ETL Pipelines by Using Athena and Step Functions

Knowledge checks

- Online knowledge check
- Sample exam question

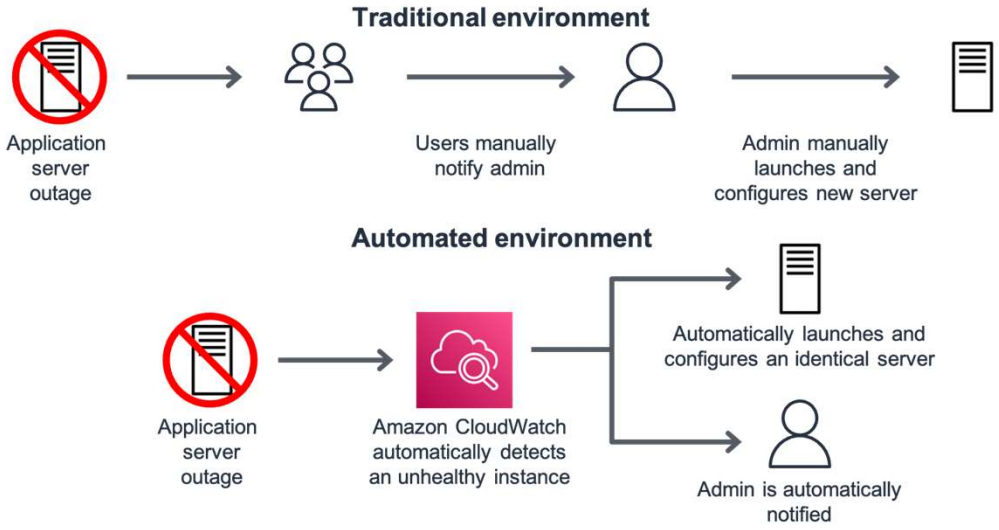
|Slide number 4
|Instructor notes
|Each module has an introduction, content sections, and a wrap-up. The wrap-up for this module contains a sample exam question for you to review with the students.
|
|Student notes
The objectives of this module are presented across multiple sections.

You will also complete a hands-on lab that uses Step Functions and Amazon Athena to build and orchestrate an extract, transform, and load (ETL) pipeline.

The module wraps up with a sample exam question and an online knowledge check that covers the presented material.

|Slide number 5
|Instructor notes
|This section reiterates the concepts that the Securing and Scaling the Data Pipeline module introduced. This module focuses on the repeatability and reusability that infrastructure as code provides.
|
|Student notes
This section discusses the benefits of automating the infrastructure deployment in support of pipelines.

Automate your environment



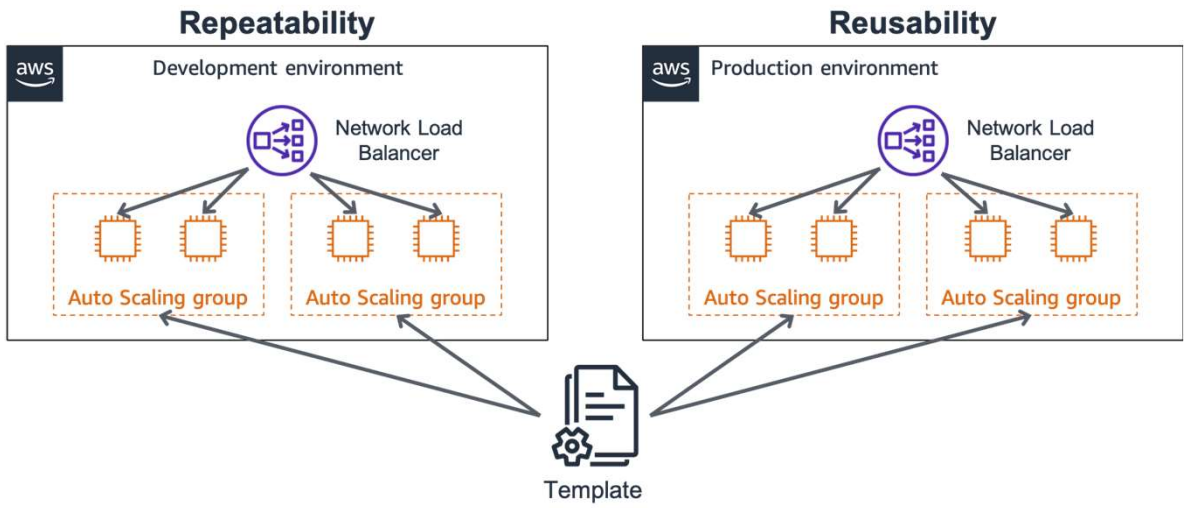
|Slide number 6
|Instructor notes
|This slide focuses on how an automated environment can respond quickly and without manual input when a failure occurs. You might pause here and ask students how this could be applied to infrastructure that supports a data analytics pipeline.
|
|Student notes
A standard best practice is to automate your environment—including the provisioning, decommission, and configuration of resources—to ensure that your system is stable, consistent, and efficient.

In the past, the systems administrator would be manually notified when the application server crashed. The administrator then needed to manually launch a new server.

Instead of following that traditional pattern, a best practice is to configure Amazon CloudWatch to automatically detect a crash. When that crash is detected, the administrator is notified, and a new server with the same configuration is launched in parallel. All of these steps happen at the same time, without human intervention.

AWS offers built-in monitoring and automation tools at virtually every layer of your infrastructure. Take advantage of these resources to ensure that your infrastructure can respond quickly to changes. You can automate detection of unhealthy resources and launching replacement resources. You can even be notified when resources are changed. By removing manual processes, you can improve your system's stability and consistency, as well as the efficiency of the organization.

Infrastructure as code



|Slide number 7
|Instructor notes
|This slide reinforces the benefits of repeatability and reusability that are achieved through infrastructure as code. You might point out to students that both the development environment and production environment have been deployed using the same template. Changes to that template could easily be implemented in production after testing in development.
|
|Student notes
In the Securing and Scaling the Data Pipeline module, you learned about infrastructure as code. You also learned about services, such as AWS CloudFormation and AWS Cloud Development Kit (AWS CDK), that can assist you in automating your pipeline. Let's review some benefits of using the infrastructure as code model.

Repeatability is a great advantage of IaC, giving you the ability to deploy an exact replication of the same environment over and over again. For example, you can use a CloudFormation template to roll out a development environment, and then test automatic scaling and the applications. Reusability means that, after a successful test, you can use that exact template to launch resources that have been thoroughly tested in the development environment into a new production environment.

If you build infrastructure with code, you gain the benefits of repeatability and reusability while you build your environments. In the example shown, a single template is used to deploy Network Load Balancers and Auto Scaling groups that contain Amazon Elastic Compute Cloud (Amazon EC2) instances. Network Load Balancers distribute traffic evenly across targets.

With one template—or a combination of templates—you can build these same complex environments repeatedly. When you do this with AWS, you can even create environments that depend on conditions—meaning that what you build is specific to the context where you created it. For instance, you can design a template to use different Amazon Machine Images (AMIs) based on whether the template is launched into the development environment or the production environment.

The repeatability and reusability benefits of infrastructure as code can help you to deploy consist infrastructure packages that are tailored to support your data analytics and ML workloads.

Key takeaways: Automating infrastructure deployment

- Automating your environment can help you ensure that your system is stable, consistent, and efficient.
- Repeatability and reusability are two key benefits of infrastructure as code.

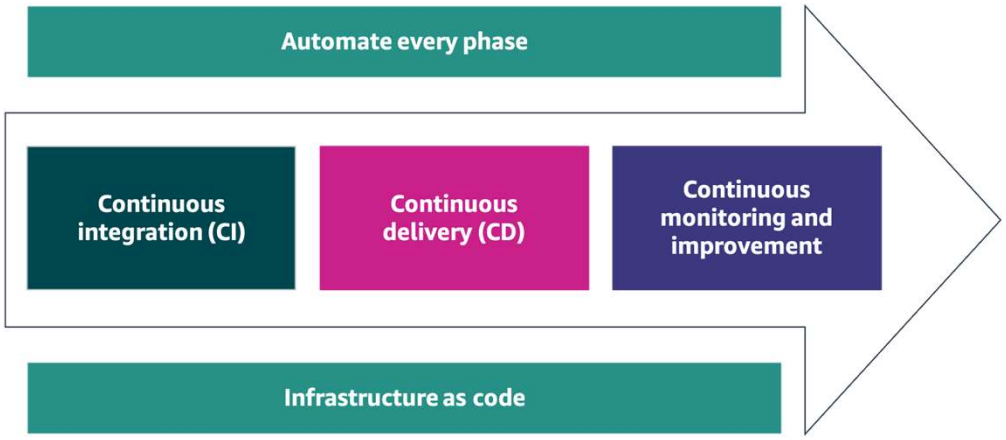
|Slide number 8
|Instructor notes
|
|Student notes
Here are a few key points to summarize this section.

Automating your environment can help you ensure that your system is stable, consistent, and efficient. Such consistency is particularly beneficial in running repetitive workloads that require identical processing methods.

Repeatability and reusability are two key benefits of infrastructure as code. By using templates to deploy infrastructure, you can ensure that production environments mirror development environments.

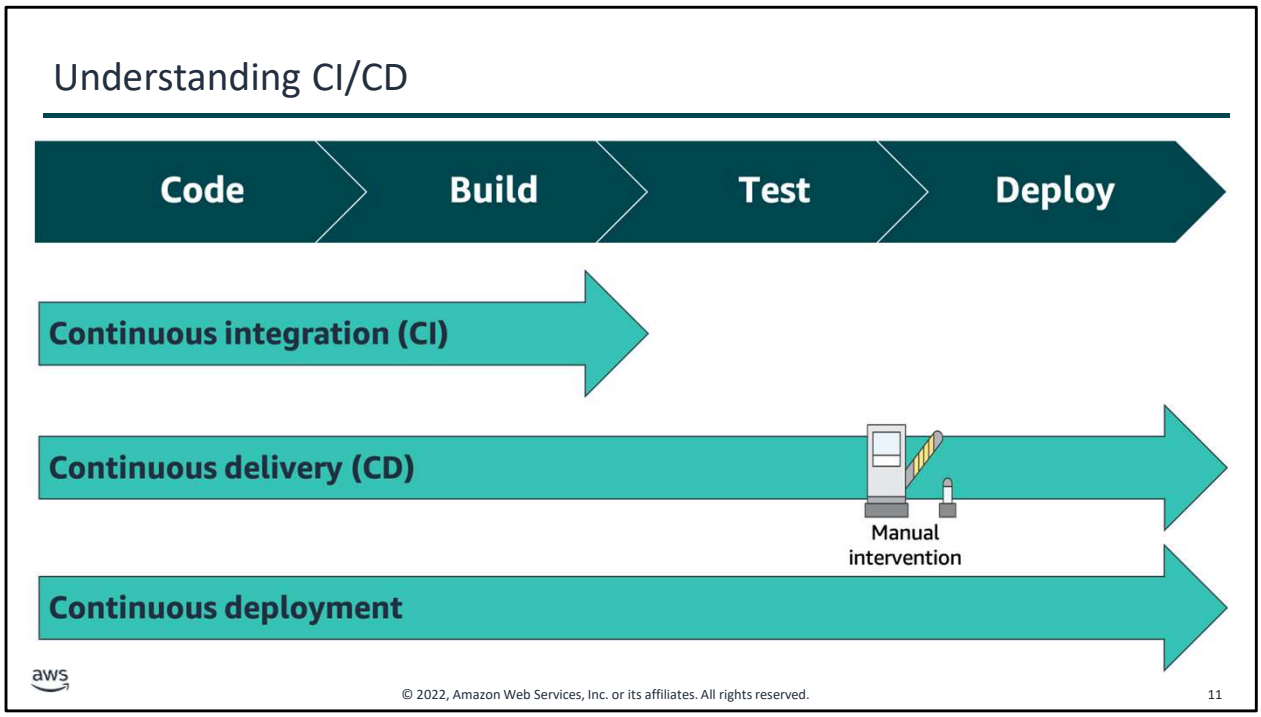
|Slide number 9
|Instructor notes
|
|Student notes
This section discusses the continuous integration and continuous delivery (CI/CD) process.

CI/CD as part of DevOps



|Slide number 10
|Instructor notes
|
|Student notes
What is DevOps? DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market. Continuous integration and continuous delivery are two DevOps best practices.

Continuous integration, referred to as CI, is a software development practice in which developers regularly merge their code changes into a central repository. *Continuous delivery*, or CD, is a software development practice where code changes are automatically built, tested, and prepared for production release. CD expands on CI by deploying all code changes to a testing environment, production environment, or both after the build stage is complete. When CD is properly implemented, developers always have a deployment-ready build artifact that has passed through a standardized test process. Code that has been deployed requires continuous monitoring and improvement.



|Slide number 11
|Instructor notes
|
|Student notes
CI/CD is a DevOps practice that spans the develop (that is, code, build, and test) and deploy stages of the software development lifecycle.

With *continuous integration,* each change is verified with an automated build and test process. In the past, developers worked in isolation for an extended period and only merged their code changes into the mainline (or parent) branch after their feature was completed. Batching changes made it difficult to merge the business logic and also the test logic. However, continuous integration practices have made teams more productive and help them to develop new features faster. Continuous integration requires teams to write automated tests that improve the quality of the software that is being released. Teams must also write tests that reduce the time that it takes to validate that the new version of the software is good.

Although there are different definitions of continuous integration, it is commonly thought that CI stops at the build stage.

As previously mentioned*, continuous delivery* extends continuous integration to include testing out to production-like stages and running verification testing against those deployments. Continuous delivery can extend all the way to the production deployment, though some form of manual intervention occurs between when code is checked in and when that code is released to customers.

Continuous delivery is a big step forward over continuous integration because it helps teams gain a greater level of certainty that their software will work in production.

Continuous deployment is the automated release of software to customers. This automation extends from check-in through production without human intervention. Continuous deployment reduces the time that it takes to deliver business value (for example, new features or updates) to customers. In turn, development teams can get customer feedback quickly on those new releases. This fast customer feedback loop helps you to iterate and release valuable software updates quickly to customers.

Key takeaways: CI/CD

- CI/CD spans the develop and deploy stages of the software development lifecycle.
- Continuous delivery improves on continuous integration by helping teams gain a greater level of certainty that their software will work in production.

|Slide number 12
|Instructor notes
|
|Student notes
Here are a few key points to summarize this section.

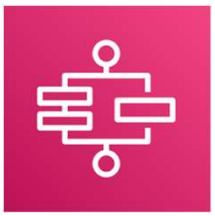
CI/CD spans the develop and deploy stages of the software development lifecycle.

Continuous delivery improves on continuous integration by helping teams gain a greater level of certainty that their software will work in production.

|Slide number 13
|Instructor notes
|This section discusses the Step Functions service and focuses on the flow states that are available to direct and control workflows. Students are introduced to elements of Step Functions Workflow Studio, which they will use in the lab that is associated with this module.
|
|Student notes
This section examines AWS Step Functions and how to use its elements and interface to build pipelines.

AWS Step Functions

- Uses visual workflows to coordinate components of distributed applications and microservices
- Automatically initiates and tracks each step, and retries when errors occur
- Logs the state of each step
- Integrates with Amazon Athena



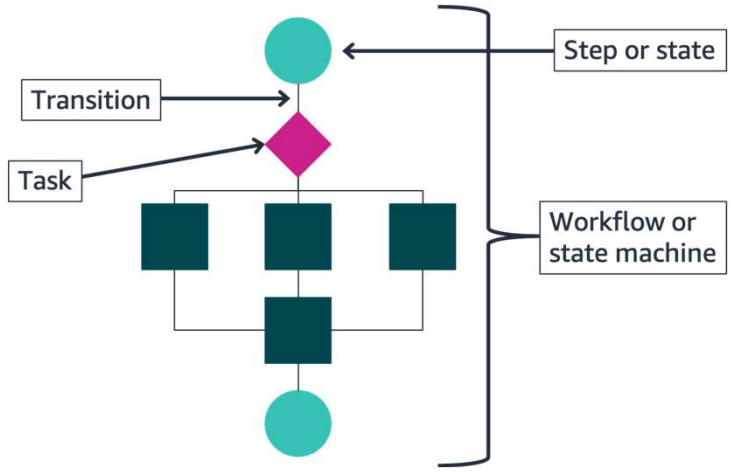
|Slide number 14
|Instructor notes
|
|Student notes

AWS Step Functions can help you coordinate components of distributed applications and microservices by using visual workflows. Step Functions provides a reliable way to coordinate components and step through the functions of your application. The service offers a graphical interface where you can visualize the components of your application as a series of steps. Step Functions automatically initiates and tracks each step—and it also retries when errors occur—so your application runs in order and as expected. Step Functions logs the state of each step, so you can diagnose and debug problems quickly.

Step Functions is integrated with Amazon Athena, so you can use Step Functions to start and stop a query run and get query results. Using Step Functions, you can run one-time or scheduled data queries, and retrieve results from your S3 data lakes. Athena is serverless, so you don’t need to set up or manage infrastructure. You pay only for the queries that you run.

To integrate Step Functions with Athena, use the provided Athena service integration APIs.

How Step Functions works

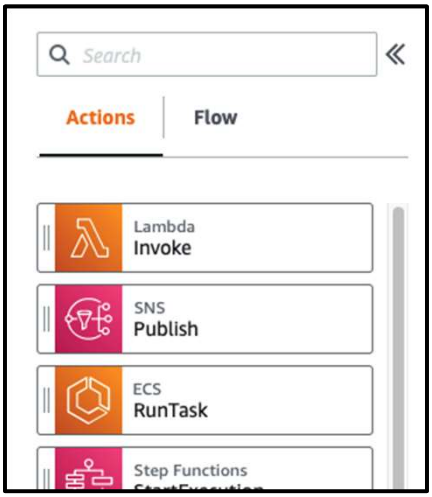


|Slide number 15
|Instructor notes
|
|Student notes
With Step Functions, you define a *workflow* as a series of *steps* and *transitions* between each step. A workflow is also called a *state machine*, and *states* are elements in a state machine. A state machine is an object that has a set number of operating conditions that depend on the object's previous condition to determine output. This means that individual states can make decisions based on their input, perform actions, and pass output to other states.

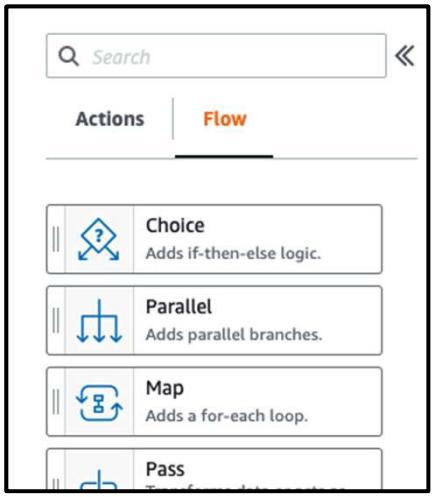
Tasks do all the work in your state machine. A task performs work by using an activity or a Lambda function, or by passing parameters to the API actions of other services. Each state machine starts with an input. The state transforms the input and passes the output to the next state.

In the Step Functions console, you can visualize your state machine and get near real-time information on your state machine tasks.

Step Functions interface: States browser



Actions panel

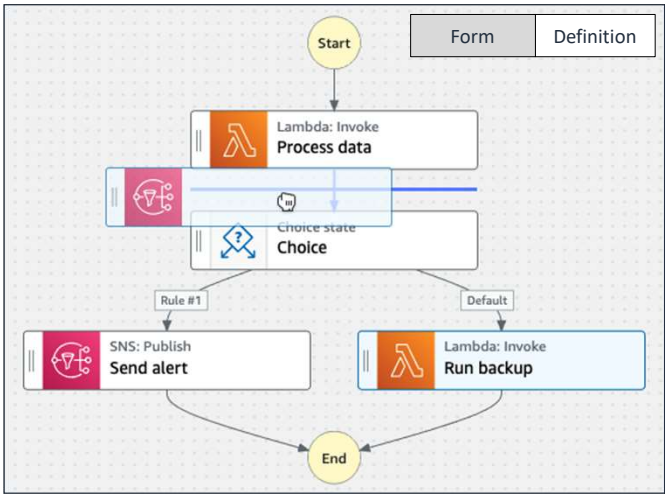


Flow panel

|Slide number 16
|Instructor notes
|
|Student notes
Located in the Step Functions Workflow Studio interface, the States browser is where you select states to drag and drop into your workflow graph. The **Actions** panel provides a list of AWS APIs, and the **Flow** panel provides a list of flow states. You can search all states by using the search field at the top of the States browser.

Step Functions interface: Canvas

- From the **Actions** and **Flow** panels, drag and drop states to the canvas.
- When a new state is added to your workflow, the code for that state is generated automatically.



|Slide number 17
|Instructor notes
|
|Student notes
After you choose a state from the **Actions** or **Flow** panel to add to your workflow, drag and drop the state to the workflow graph in the canvas area. A line shows where it will be placed in your workflow. After you add the new state to your workflow, its code is automatically generated.

The image in this slide shows the canvas of the Workflow Studio interface. Within the workflow, you can drag and drop states to move them to different places in the workflow. If your workflow is complex, you might not be able to view all of it in the canvas panel. Use the controls at the top of the canvas to zoom in or out. To view different parts of a workflow graph, you can drag the workflow graph in the canvas.

Step Functions interface: Inspector

|Slide number 18
|Instructor notes
|
|Student notes
After you have added a state to your workflow, you will want to configure it. Choose the state that you want to configure, and its configuration options appear in the **Inspector** panel.

To see the workflow code, choose the **Definition** panel. The code that is associated with the selected state is highlighted.

Let's look at the different state types that are available and the functions that they perform within the Step Functions service.

State types

Task	Single unit of work performed by a state machine
Pass	Passes input to output, without performing work
Choice	Adds branching logic to a state machine
Parallel	Creates parallel branches for running states
Wait	Delays from continuing for a specified
Map	Dynamically iterates steps
Succeed	Stops running a state successfully
Fail	Stops running a state and marks it as a failure

|Slide number 19
|Instructor notes
|
|Student notes

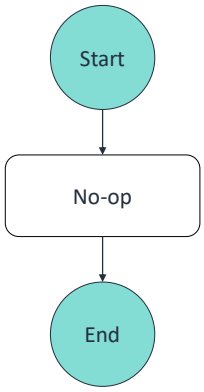
Flow states are available for you to direct and control your workflow. All of them take input from the previous state, and many let you filter the input from the preceding state. Then, they output to the state that follows.

Each state has a different function:

- **Task:** A single unit of work performed by a state machine. All work in a state machine is done by tasks using an activity or Lambda function.
- **Pass:** Pass input to output. (Optional) You can add fixed data into the output.
- **Choice:** Add branching logic to your workflow. In the **Configuration** tab of the **Inspector** panel, you can configure rules to determine which state the workflow will transition to.
- **Parallel:** Add parallel branches of to your workflow.
- **Wait:** Pause the workflow for a certain amount of time or until a specified time or date.
- **Map:** Dynamically iterate steps for each element of an input array. Unlike a Parallel flow state, a Map state will process the same steps for multiple entries of an array in the state input.
- **Succeed:** Stop your workflow with a success.
- **Fail:** Stop your workflow with a failure.

Pass state

```
"No-op": {
   "Type": "Pass",
   "Result": {
      "x-datum": 0.381018,
      "y-datum": 622.2269926397355
   },
   "ResultPath": "$.coords",
   "Next": "End"
}
```



|Slide number 20
|Instructor notes
|
|Student notes
A Pass state ("Type": "Pass") passes its input to its output, without performing work. Pass states are useful when constructing and debugging state machines.

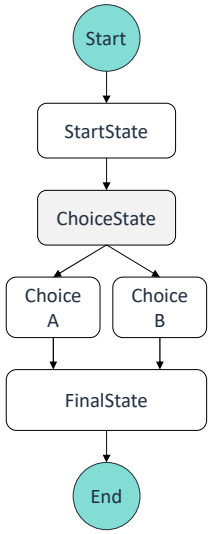
The slide shows an example of a state machine that consists of a single Pass state. When this state machine runs, it injects some fixed data, which might be useful for testing purposes.

Choice state

```
"ChoiceState": {
   "Type" : "Choice",
   "Choices": [
      {
         "Variable": "$.foo",
         "NumericEquals": 1,
         "Next": "Choice A"
      },
      {
         "Variable": "$.foo",
         "NumericEquals": 2,
         "Next": "Choice B"
      }
   ],
   "Default": "DefaultState"
}
```

Choice rule

Choice rule



|Slide number 21
|Instructor notes
|Reiterate that the Choice state is used to handle which logical route the workflow should follow next. Students will use Choice states in this module's lab.
|
|Student notes
A Choice state ("Type": "Choice") adds branching logic to a state machine. A Choice state must have a **Choices** field whose value is a non-empty array. Each element in the array is an object called a Choice Rule.

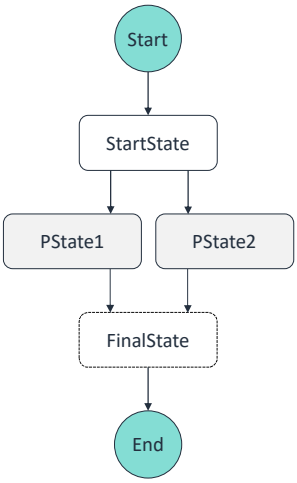
A Choice Rule contains the following:

- **Comparison:** Two fields that specify an input variable to be compared, the type of comparison, and the value to compare the variable to.
- **Next field:** The value of this field must match a state name in the state machine.

In the example on the slide, the first choice rule checks whether the numerical value is equal to 1. The second choice rule checks whether the numerical value is equal to 2.

Parallel state

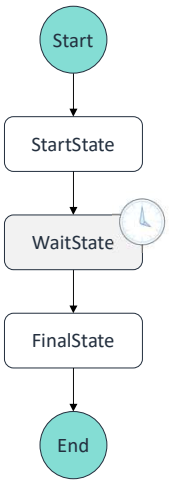
```
"LookupCustomerInfo": {
   "Type": "Parallel",
   "Branches": [
      {
         "StartAt": "PState1",
         "States": {
            "PState1": {…}
         }
      },
      {
         "StartAt": "PState2",
         "States": {
            "PState2": {…}
         }
   }
   ],
   "Next": "NextState"
}
```



|Slide number 22
|Instructor notes
|
|Student notes
The Parallel state ("Type": "Parallel") creates parallel branches in your state machine.
A Parallel state causes Step Functions to run each branch—starting with the state that is named in that branch's StartAt field—as concurrently as possible. The workflow will wait until all branches reach a terminal state before it processes the Parallel state's Next field.

Wait state

```
"wait_ten_seconds": {
   "Type": "Wait",
   "Seconds": 10,
   "Next": "NextState"
}
```

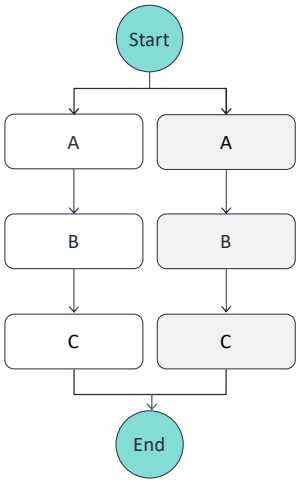


|Slide number 23
|Instructor notes
|
|Student notes
A Wait state ("Type": "Wait") delays the state machine from continuing for a specified time. You can choose either a relative time (which is specified in seconds from when the state begins) or an absolute time (which is specified as a timestamp). You can use a wait state as a form of timer for your workflow.

In the example on the slide, the Wait state introduces a 10-second delay into the state machine.

Map state

```
"Validate-All": {
   "Type": "Map",
   "InputPath": "$.detail",
   "ItemsPath": "$.shipped",
   "MaxConcurrency": 0,
   "Iterator": {
      "StartAt": "Validate",
      "States": {
         "Validate": {
            "Type": "Task",
            "Resource": "arn:aws:lambda:us-east-
               1:123456789012:function:ship-val",
            "End": true
         }
      }
   }
```



|Slide number 24
|Instructor notes
|
|Student notes
You can use the Map state ("Type": "Map") to run a set of steps for each element of an input array. While the Parallel state processes multiple branches of steps by using the same input, a Map state will process the same steps for multiple entries of an array in the state input.

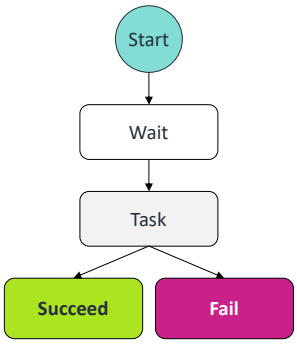
Succeed and Fail states

Succeed state

```
"SuccessState": {
   "Type": "Succeed"
}
```

Fail state

```
"FailState": {
   "Type": "Fail",
   "Cause": "Invalid response.",
   "Error": "ErrorA"
}
```



|Slide number 25
|Instructor notes
|
|Student notes

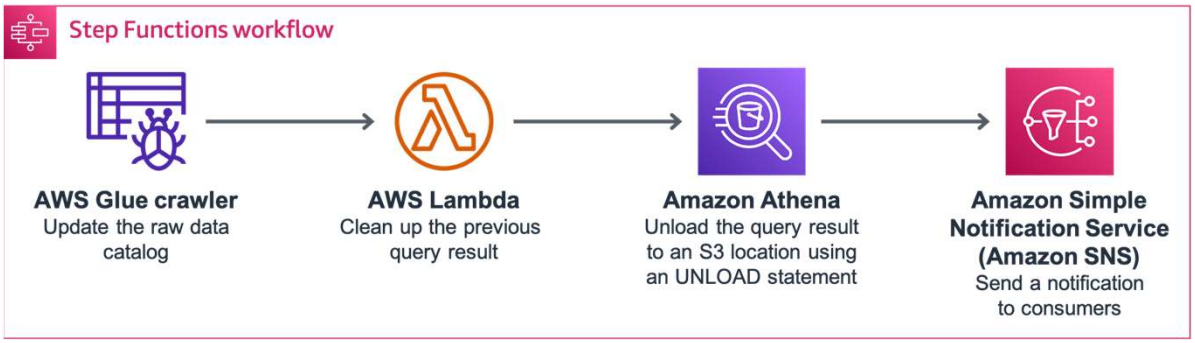
A Succeed state ("Type": "Succeed") stops running a state successfully. The Succeed state is a useful target for Choice state branches that do nothing except stop running the state. Because Succeed states are terminal states, they don’t have a Next field, and they don't need an End field.

A Fail state ("Type": "Fail") stops running the state machine and marks it as a failure. The Fail state requires the Type field (which specifies the state’s type). In addition, the Fail state allows the following optional fields:

- **Cause:** Provides a custom failure string that can be used for operational or diagnostic purposes.
- **Error:** Provides an error name that can be used for error handling (retry/catch), operational, or diagnostic purposes.

Because Fail states always exit the state machine, they don’t have a Next field, and they don’t require an End field. The slide provides an example.

Simplifying ETL pipelines with Step Functions



|Slide number 26
|Instructor notes
|
|Student notes

Step Functions is integrated with Athena to facilitate building workflows that include Athena queries and data processing operations. This integration helps you create repeatable and scalable data processing pipelines as part of a larger business application. You can visualize the workflows on the Athena console.

In the example on the slide, the raw data is in a tab-separated values (TSV) format, and the data is ingested daily. As information is ingested, the AWS Glue crawler is used to provide updates to the raw data catalog. You can then use the Athena UNLOAD statement to convert the data into Parquet format. After conversion, you send the location of the Parquet file as an Amazon Simple Notification Service (Amazon SNS) notification. You can use Amazon SNS to notify downstream applications to take further action. One common example is to initiate a Lambda function that uploads the Athena transformation result into Amazon Redshift.

Key takeaways: Automating with Step Functions

- With Step Functions, you can use visual workflows to coordinate the components of distributed applications and microservices.
- You define a *workflow*, which is also referred to as a *state machine*, as a series of steps and transitions between each step.
- Step Functions is integrated with Athena to facilitate building workflows that include Athena queries and data processing operations.

|Slide number 27
|Instructor notes
|
|Student notes
Here are a few key points to summarize this section.

With Step Functions, you can use visual workflows to coordinate the components of distributed applications and microservices.

You define a workflow, also referred to as a state machine, as a series of steps and transitions between each step.

Step Functions is integrated with Athena to facilitate building workflows that include Athena queries and data processing operations.

|Slide number 28
|Instructor notes
|
|Student notes
You will now complete a lab. The next slide summarizes what you will do in the lab, and you will find the detailed instructions in the lab environment.

Lab introduction: Building and Orchestrating ETL Pipelines by Using Athena and Step Functions

- In this lab, you will use Step Functions to build an ETL pipeline that uses Amazon S3, an AWS Glue Data Catalog, and Athena to process a large dataset.
- You will design the workflow so that if AWS Glue tables don't exist, the workflow will invoke additional Athena queries to create them.
- Open your lab environment to start the lab and find additional details about the tasks that you will perform during this lab.

|Slide number 29
|Instructor notes
|
|Student notes
Access the lab environment through your online course to get additional details and complete the lab.

Debrief: Building and Orchestrating ETL Pipelines by Using Athena and Step Functions

- Why did you need to configure routing logic in the Step Functions workflow?
- When and how did you implement Snappy compression and ensure that the data would be stored in Parquet format? What is the benefit of storing the data this way?
- What is the purpose of creating a state machine in a service such as Step Functions?

|Slide number 30
|Instructor notes
|Q1 - **Example strong response:** First, we added a Choice state so that the state machine would follow the appropriate path depending on whether AWS Glue tables were found in the “Get lookup query results” task. Then, we added another Choice state to the workflow inside a Map state. This Choice was used to determine whether a particular Parquet table existed in the AWS Glue database. Depending on whether the table name matched the “*data_parquet” condition, a different rule was invoked.
|Q2 - **Example strong response:** We accomplished this when we defined a new table. The table was defined in an Athena StartExecutionQuery task. The table definition (which was defined in the QueryString API Parameter) specified format=‘PARQUET’ and parquet_compression=‘SNAPPY’. Snappy compression optimizes the data storage, with the goal of reducing overall storage use while retaining the ability to accomplish high-speed processing. Parquet is a data storage format that is optimized to handle tables with potentially hundreds of columns.
|Q3 - **Example strong response:** You can use state machines to automate time-consuming tasks that you might otherwise need to do manually and repeatedly. Therefore, state machines can improve productivity. They also help to enforce repeatability, which should reduce errors. For example, the type of ETL pipeline that was created in this lab could be reused to accelerate data processing across many projects. Someone could duplicate the state machine and make some minor modifications to adapt it for new data processing projects.

|

|Student notes

Your instructor might review these questions with you, or you might review them on your own. Use this opportunity to extend your thinking about the tasks that you performed during the lab.

|Slide number 31
|Instructor notes
|
|Student notes
This section summarizes what you have learned and brings the Automating the Pipeline module to a close.

Module summary

This module prepared you to do the following:

- Identify the benefits of automating your pipeline.
- Understand the role of CI/CD and how it applies to your pipeline.
- Examine the states of Step Functions.
- Use Step Functions to build and automate a data pipeline.

|Slide number 32
|Instructor notes
|This is a good opportunity to use an online group or discussion board to ask students to reflect on what they have learned. You might ask the students to recall a point from the module that aligns to one of the listed objectives. This provides a good segue to the knowledge check and sample exam question.
|
|Student notes
This module discussed the benefits of automating your pipeline by using infrastructure as code and Step Functions. You examined the role of CI/CD in pipeline deployments. You learned about flow states in Step Functions and how to use them to control and direct your workflows. You also examined the elements of the Step Functions Workflow Studio.

Module knowledge check

- The knowledge check is delivered online within your course.
- The knowledge check includes 10 questions based on material presented on the slides and in the slide notes.
- You can retake the knowledge check as many times as you like.

|Slide number 33
|Instructor notes
|
|Student notes
Use your online course to access the knowledge check for this module.

Sample exam question

A data engineer is using Step Functions to automate a pipeline to handle an analytics workload. They are using the Workflow Studio interface to drag and drop components into the workflow. However, the engineer isn't sure if the auto-generated code will perform the tasks and processing that are required. Which element of the Step Functions interface can they use to examine the code for accuracy?

Identify the key words and phrases before continuing.

The following are the key words and phrases:

- **Step Functions**
- **Auto-generated code**
- **Examine the code**

aws

|Slide number 34

|Instructor notes

|The key words section is animated to be revealed on click.

|

|Student notes

The question notes that the data engineer is using the Step Functions Workflow Studio to build their workflow, and that they want to review the code that is being auto-generated for them.

Sample exam question: Response choices

A data engineer is using **Step Functions** to automate a pipeline to handle an analytics workload. They are using the Workflow Studio interface to drag and drop components into the workflow. However, the engineer isn't sure if the **auto-generated code** will perform the tasks and processing that are required. Which element of the Step Functions interface can they use to **examine the code** for accuracy?

Choice	Response
A	Use the States browser to examine the code.
B	Use the **Form** panel to view the code.
C	Review the definition area on the **Inspector** panel.
D	Run the Step Functions workflow and examine the output.

|Slide number 35
|Instructor notes
|
|Student notes

Use the key words that you identified on the previous slide, and review each of the possible responses to determine which one best addresses the question.

Sample exam question: Answer

The correct answer is C.

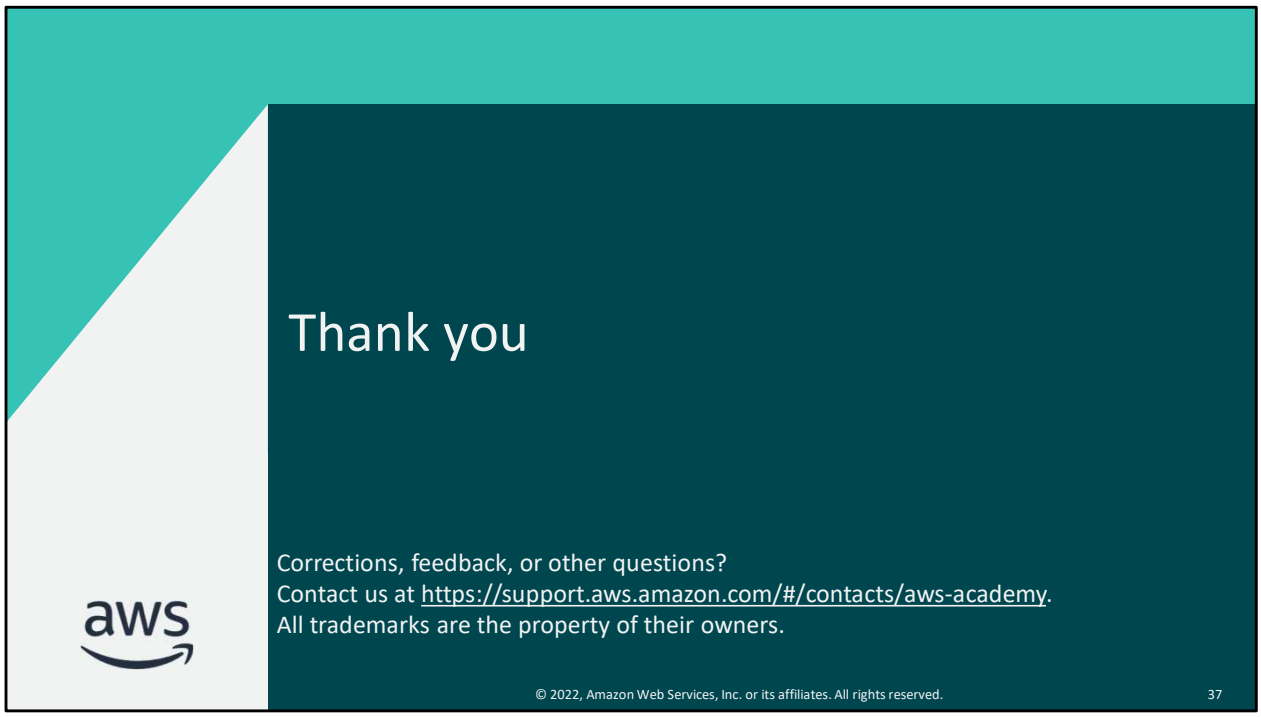
Choice	Response
C	Review the definition area on the **Inspector** panel.

|Slide number 36
|Instructor notes
|
|Student notes
Choice A (Use the States browser to examine the code) doesn't provide a method to view the code that is associated with the states and actions that are available to select.

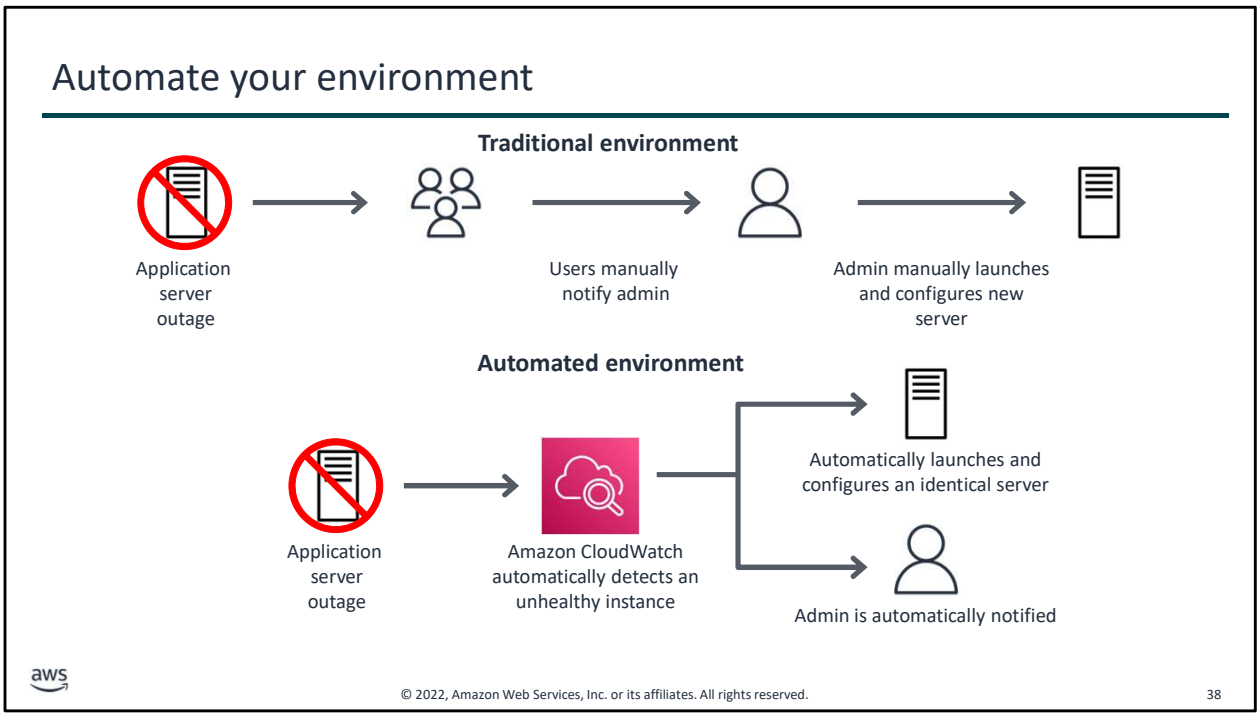
Choice B (Use the drag-and-drop elements of the **Form** panel) would provide the ability to select the individual states and actions that are part of the workflow. However, the **Form** panel doesn't present any information regarding the code that is generated.

Choice D (Run the Step Functions workflow and examine the output) will only run the workflow and present the output.

Of the options available, using the definition area on the **Inspector** panel is the only one that enables a user to examine and verify the code that is auto-generated through the Step Functions canvas. C is the correct answer.



|Slide number 37
|Instructor notes
|
|Student notes
That concludes this module. The Content Resources page of your course includes links to additional resources that are related to this module.



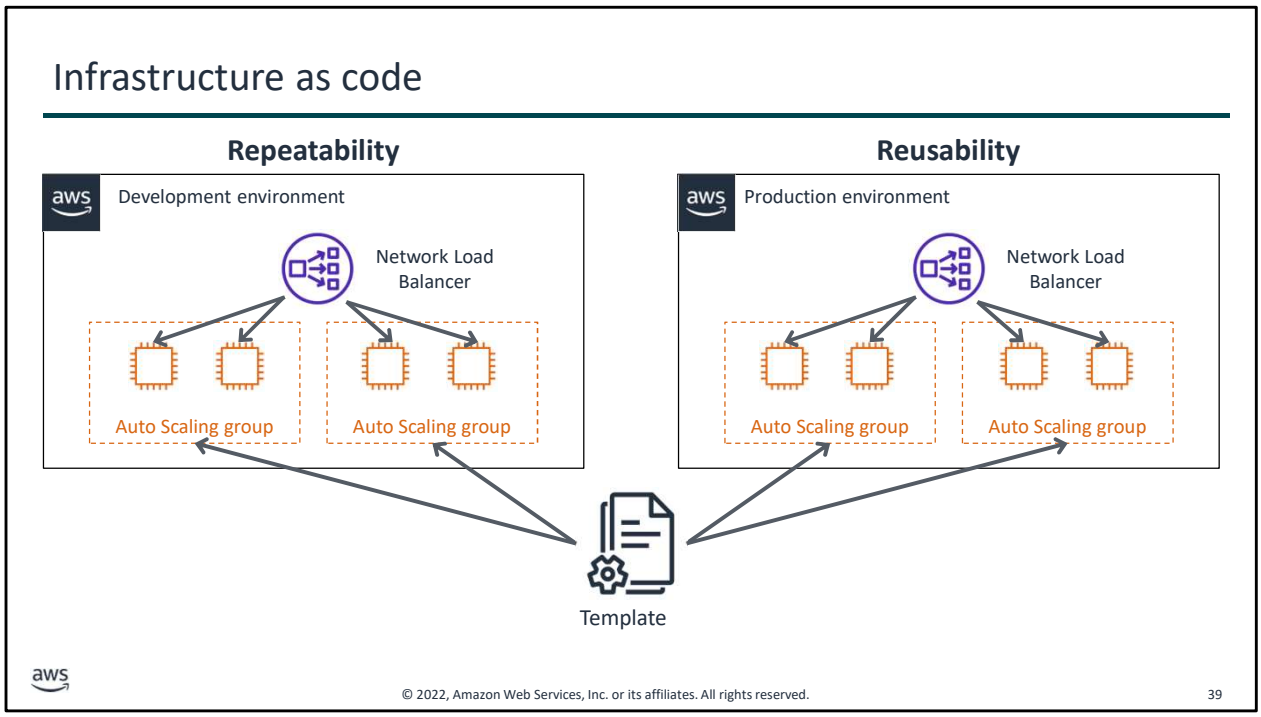
|Slide number 38
|Instructor notes
|This slide focuses on how an automated environment can respond quickly and without manual input when a failure occurs. You might pause here and ask students how this could be applied to infrastructure that supports a data analytics pipeline.
|
|Student notes
A standard best practice is to automate your environment—including the provisioning, decommission, and configuration of resources—to ensure that your system is stable, consistent, and efficient.

In the past, the systems administrator would be manually notified when the application server crashed. The administrator then needed to manually launch a new server.

Instead of following that traditional pattern, a best practice is to configure Amazon CloudWatch to automatically detect a crash. When that crash is detected, the administrator is notified, and a new server with the same configuration is launched in parallel. All of these steps happen at the same time, without human intervention.

AWS offers built-in monitoring and automation tools at virtually every layer of your

infrastructure. Take advantage of these resources to ensure that your infrastructure can respond quickly to changes. You can automate detection of unhealthy resources and launching replacement resources. You can even be notified when resources are changed. By removing manual processes, you can improve your system's stability and consistency, as well as the efficiency of the organization.



|Slide number 39

|Instructor notes

|This slide reinforces the benefits of repeatability and reusability that are achieved through infrastructure as code. You might point out to students that both the development environment and production environment have been deployed using the same template. Changes to that template could easily be implemented in production after testing in development.

|

|Student notes

In the Securing and Scaling the Data Pipeline module, you learned about infrastructure as code. You also learned about services, such as AWS CloudFormation and AWS Cloud Development Kit (AWS CDK), that can assist you in automating your pipeline. Let's review some benefits of using the infrastructure as code model.

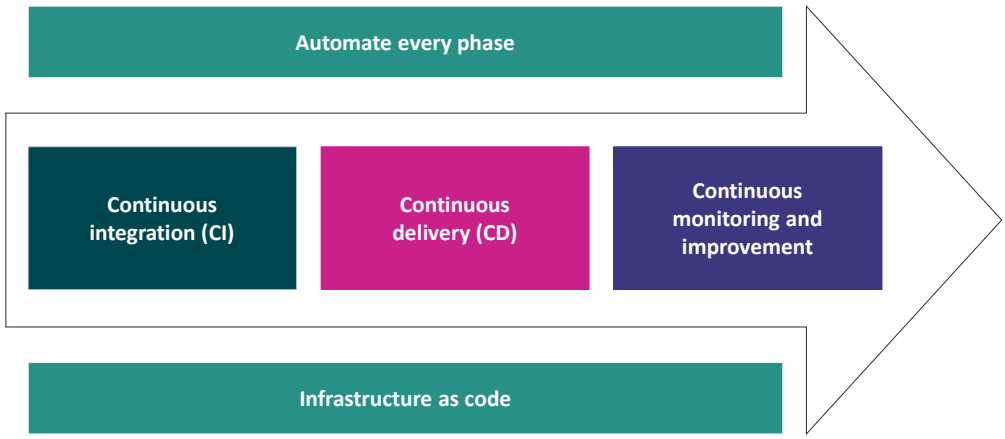
Repeatability is a great advantage of IaC, giving you the ability to deploy an exact replication of the same environment over and over again. For example, you can use a CloudFormation template to roll out a development environment, and then test automatic scaling and the applications. Reusability means that, after a successful test, you can use that exact template to launch resources that have been thoroughly tested in the development environment into a new production environment.

If you build infrastructure with code, you gain the benefits of repeatability and reusability while you build your environments. In the example shown, a single template is used to deploy Network Load Balancers and Auto Scaling groups that contain Amazon Elastic Compute Cloud (Amazon EC2) instances. Network Load Balancers distribute traffic evenly across targets.

With one template—or a combination of templates—you can build these same complex environments repeatedly. When you do this with AWS, you can even create environments that depend on conditions—meaning that what you build is specific to the context where you created it. For instance, you can design a template to use different Amazon Machine Images (AMIs) based on whether the template is launched into the development environment or the production environment.

The repeatability and reusability benefits of infrastructure as code can help you to deploy consist infrastructure packages that are tailored to support your data analytics and ML workloads.

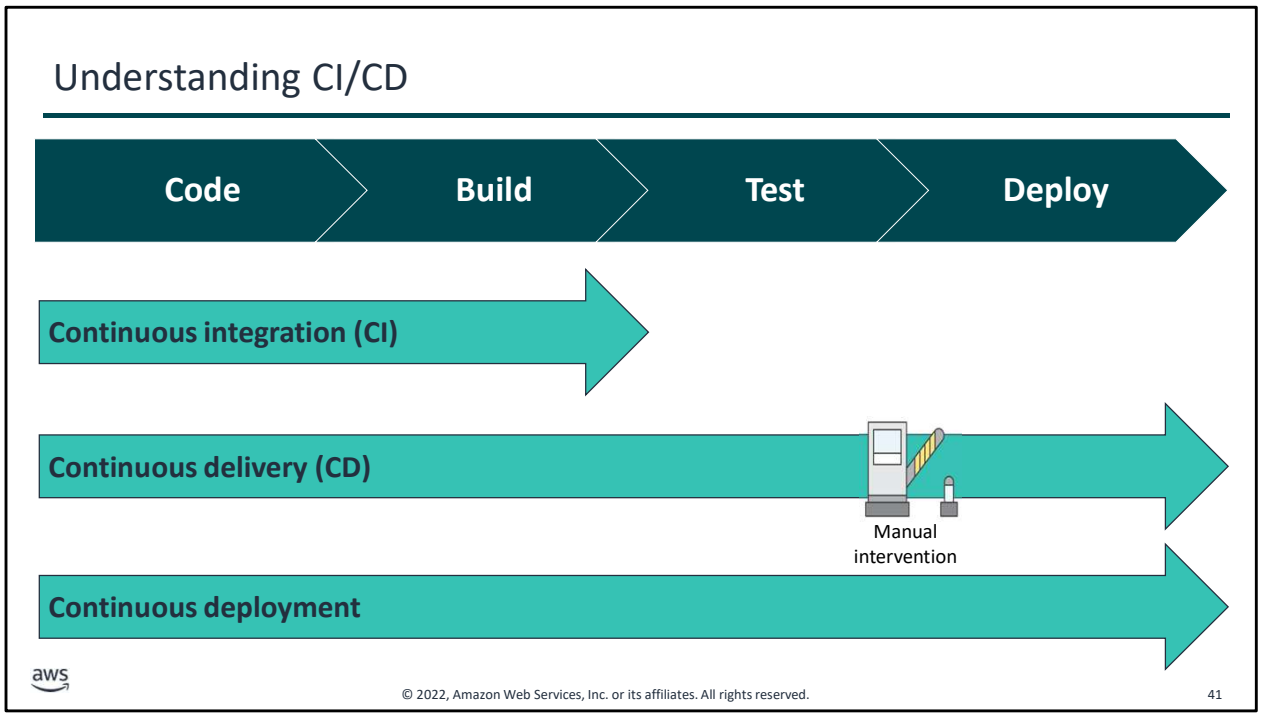
CI/CD as part of DevOps



|Slide number 40
|Instructor notes
|
|Student notes

What is DevOps? DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market. Continuous integration and continuous delivery are two DevOps best practices.

Continuous integration, referred to as CI, is a software development practice in which developers regularly merge their code changes into a central repository. *Continuous delivery*, or CD, is a software development practice where code changes are automatically built, tested, and prepared for production release. CD expands on CI by deploying all code changes to a testing environment, production environment, or both after the build stage is complete. When CD is properly implemented, developers always have a deployment-ready build artifact that has passed through a standardized test process. Code that has been deployed requires continuous monitoring and improvement.



|Slide number 41
|Instructor notes
|
|Student notes
CI/CD is a DevOps practice that spans the develop (that is, code, build, and test) and deploy stages of the software development lifecycle.

With *continuous integration* each change is verified with an automated build and test process. In the past, developers worked in isolation for an extended period and only merged their code changes into the mainline (or parent) branch after their feature was completed. Batching changes made it difficult to merge the business logic and also the test logic. However, continuous integration practices have made teams more productive and help them to develop new features faster. Continuous integration requires teams to write automated tests that improve the quality of the software that is being released. Teams must also write tests that reduce the time that it takes to validate that the new version of the software is good.

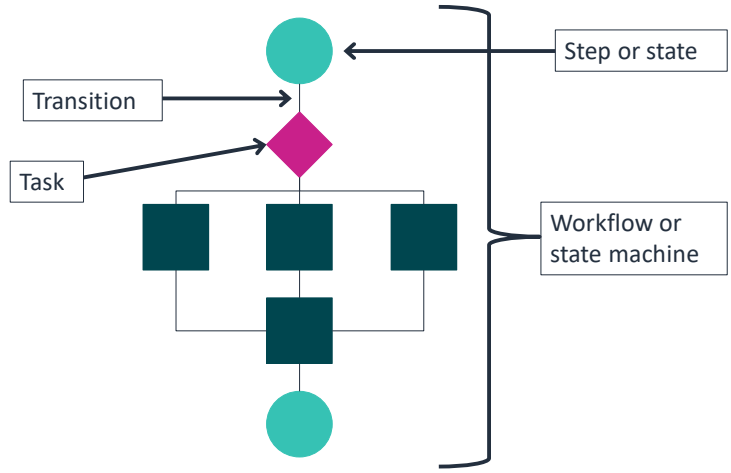
Although there are different definitions of continuous integration, it is commonly thought that CI stops at the build stage.

As previously mentioned*, continuous delivery* extends continuous integration to include testing out to production-like stages and running verification testing against those deployments. Continuous delivery can extend all the way to the production deployment, though some form of manual intervention occurs between when code is checked in and when that code is released to customers.

Continuous delivery is a big step forward over continuous integration because it helps teams gain a greater level of certainty that their software will work in production.

Continuous deployment is the automated release of software to customers. This automation extends from check-in through production without human intervention. Continuous deployment reduces the time that it takes to deliver business value (for example, new features or updates) to customers. In turn, development teams can get customer feedback quickly on those new releases. This fast customer feedback loop helps you to iterate and release valuable software updates quickly to customers.

How Step Functions works

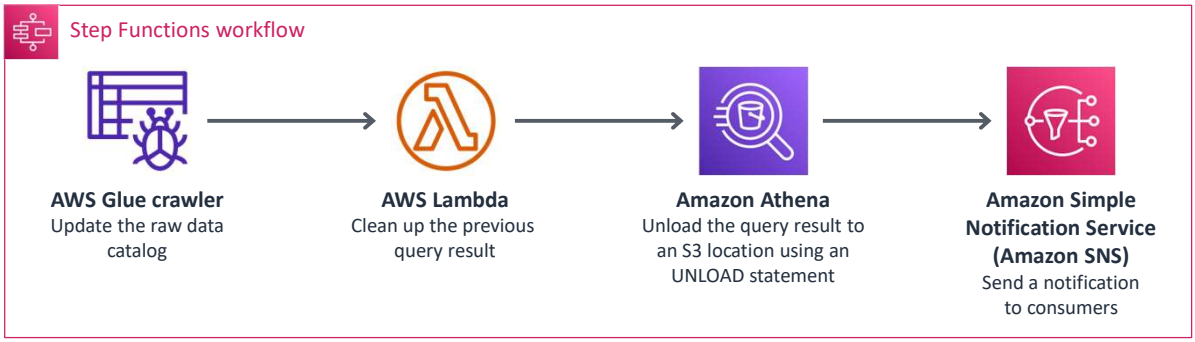


|Slide number 42
|Instructor notes
|
|Student notes
With Step Functions, you define a *workflow* as a series of *steps* and *transitions* between each step. A workflow is also called a *state machine*, and *states* are elements in a state machine. A state machine is an object that has a set number of operating conditions that depend on the object's previous condition to determine output. This means that individual states can make decisions based on their input, perform actions, and pass output to other states.

Tasks do all the work in your state machine. A task performs work by using an activity or a Lambda function, or by passing parameters to the API actions of other services. Each state machine starts with an input. The state transforms the input and passes the output to the next state.

In the Step Functions console, you can visualize your state machine and get near real-time information on your state machine tasks.

Simplifying ETL pipelines with Step Functions



|Slide number 43
|Instructor notes
|
|Student notes

Step Functions is integrated with Athena to facilitate building workflows that include Athena queries and data processing operations. This integration helps you create repeatable and scalable data processing pipelines as part of a larger business application. You can visualize the workflows on the Athena console.

In the example on the slide, the raw data is in a tab-separated values (TSV) format, and the data is ingested daily. As information is ingested, the AWS Glue crawler is used to provide updates to the raw data catalog. You can then use the Athena UNLOAD statement to convert the data into Parquet format. After conversion, you send the location of the Parquet file as an Amazon Simple Notification Service (Amazon SNS) notification. You can use Amazon SNS to notify downstream applications to take further action. One common example is to initiate a Lambda function that uploads the Athena transformation result into Amazon Redshift.