

SERVERLESS ORDER FLOW WITH AWS LAMBDA & STEP FUNCTIONS



SERVERLESS ORDERFLOW PROJECT DOCUMENTATION

BY KHUSHI NANDWANI

INTRODUCTION

In the modern cloud ecosystem, serverless architecture empowers developers to build and deploy applications without managing infrastructure. This hands-on project walks you through creating a simplified serverless order-processing system using AWS Lambda and Step Functions. You'll build three Lambda functions that simulate inventory checking, payment processing, and order status updating. These are orchestrated via AWS Step Functions to create a seamless order workflow.

PREREQUISITES

- AWS Account with administrative access
- Basic knowledge of AWS Lambda & IAM
- Familiarity with JavaScript (Node.js)
- Understanding of JSON structure and API workflows



PROJECT OVERVIEW

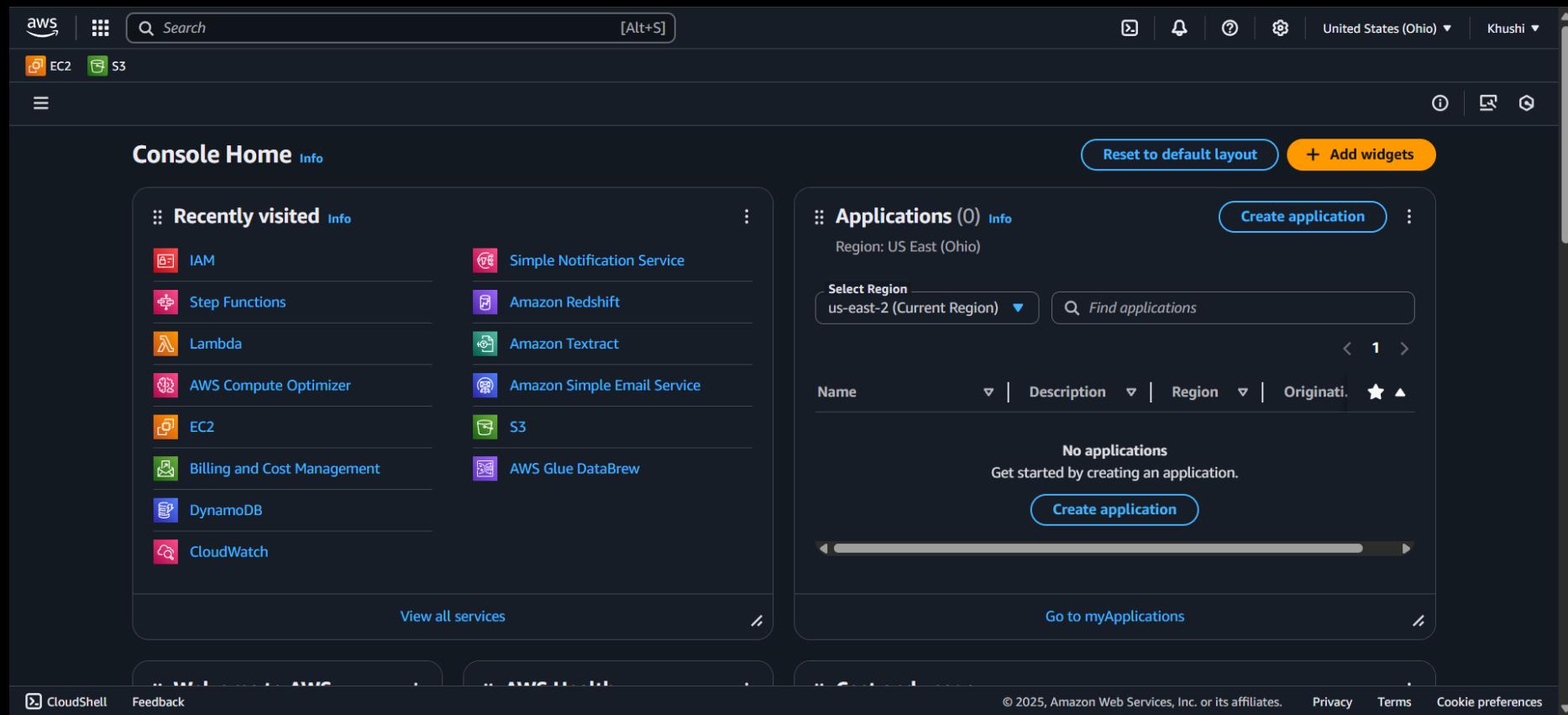
This project implements an automated serverless order flow using AWS services. You will:

- Create three Lambda functions:
 - checkInventory
 - processPayment
 - updateOrderStatus
- Use AWS Step Functions to orchestrate these tasks in sequence.
- Execute the state machine with sample input to simulate the end-to-end process.
- Visualize execution and debug via graphical and tabular representations.
- Clean up resources to avoid unnecessary billing.

📍 STEP-BY-STEP INSTRUCTIONS

I . CREATE LAMBDA FUNCTION: CHECKINVENTORY 🔎

1. Log in to your AWS Console



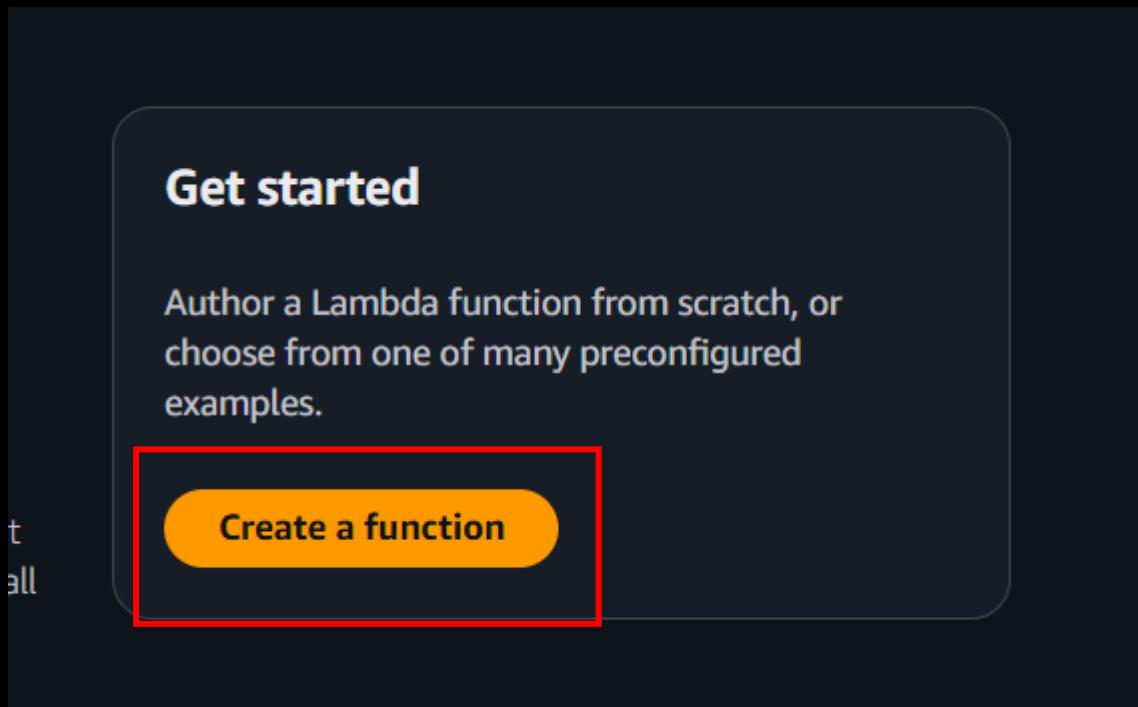
2. Navigate to AWS Lambda Service

The screenshot shows the AWS Lambda service homepage. At the top, there's a navigation bar with the AWS logo, a search bar, and various icons. Below the navigation bar, the word "Compute" is visible. The main heading is "AWS Lambda", followed by the tagline "lets you run code without thinking about servers." A paragraph explains that users pay only for compute time and can run code for virtually any application or backend service with zero administration. To the right, there's a "Get started" section with a "Create a function" button. Below this, there's a "How it works" section for Node.js, showing a snippet of code:

```
1 exports.handler = async (event) => {  
2   console.log(event);  
3   return 'Hello from Lambda!';  
4 };  
5
```

There are tabs for ".NET", "Java", "Node.js", "Python", "Ruby", and "Custom runtime". A "Run" button and a link to "Next: Lambda responds to events" are also present. At the bottom, there are links for "CloudShell", "© 2025, Amazon Web Services, Inc. or its affiliates.", "Privacy", "Terms", and "Cookie preferences".

3. Click on "Create a Function"



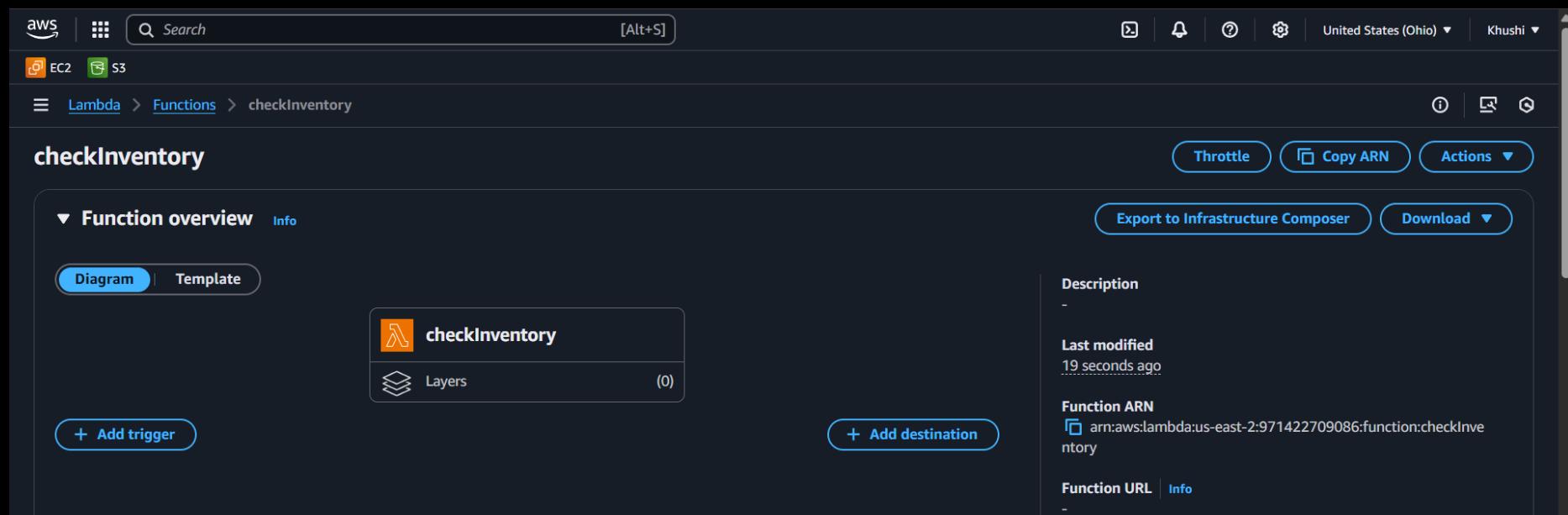
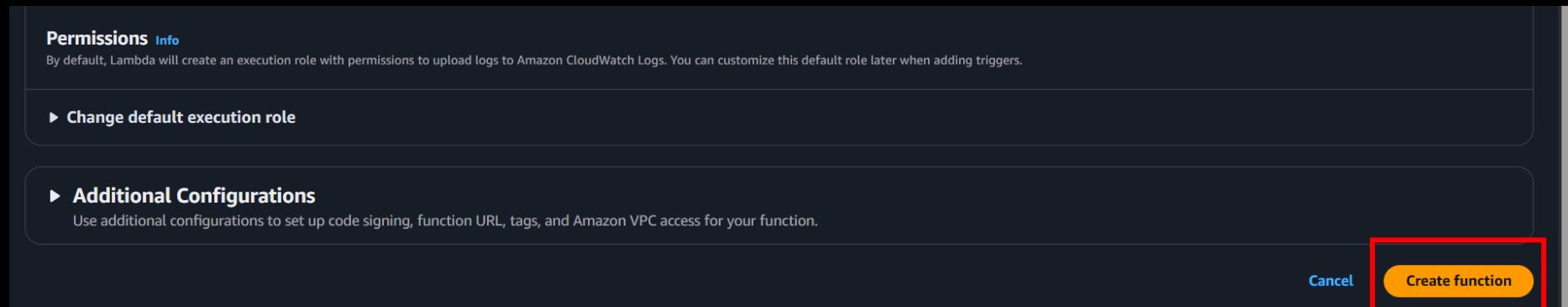
4. Name it checkInventory

The screenshot shows the AWS Lambda 'Create function' wizard. At the top, there's a navigation bar with the AWS logo, a search bar, and links for EC2 and S3. Below that, the path 'Lambda > Functions > Create function' is shown. The main section is titled 'Create function' with a 'Info' link. It says 'Choose one of the following options to create your function.' Three options are available: 'Author from scratch' (selected), 'Use a blueprint', and 'Container image'. The 'Basic information' step is currently active. It asks for a 'Function name' and provides a placeholder 'checkInventory'. A note below states: 'Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).'

5. Select runtime: Node.js 20.x

The screenshot shows the 'Runtime' configuration step. A red box highlights the 'Runtime' section, which includes a 'Info' link and a note: 'Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.' Below is a dropdown menu showing 'Node.js 22.x' as the selected option. To the right of the dropdown is a blue circular icon with a white letter 'C' and a downward arrow. The 'Architecture' section is also visible, with a note: 'Choose the instruction set architecture you want for your function code.' It shows two options: 'arm64' and 'x86_64', with 'x86_64' selected.

6. Click “Create Function”



7. In the code editor, paste the checkInventory Lambda code

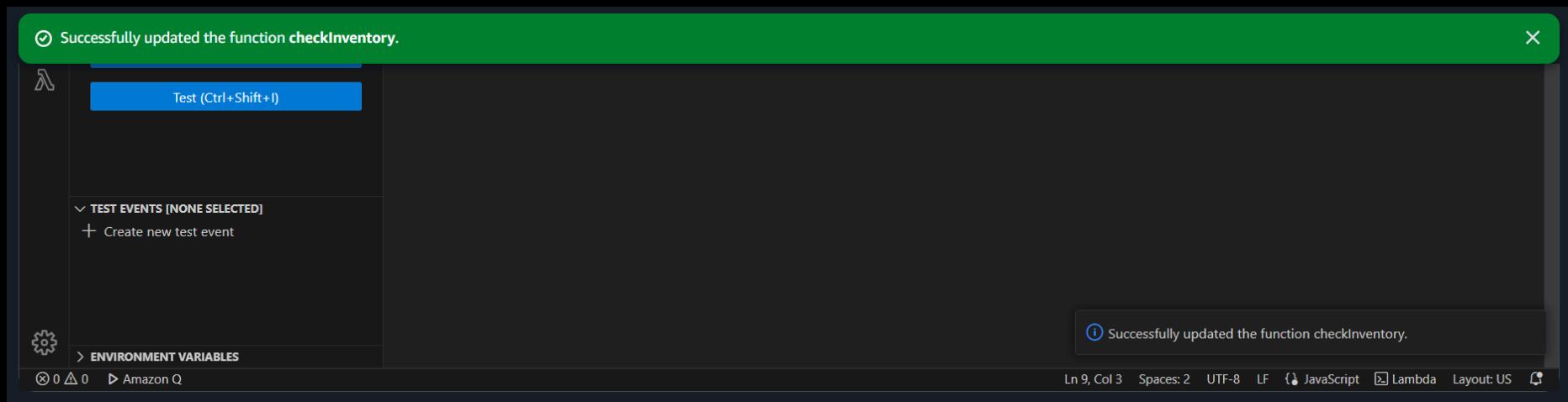
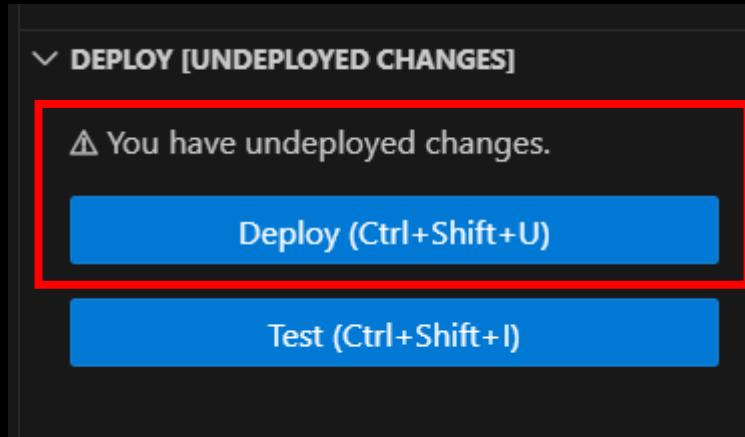
The screenshot shows a GitHub repository interface. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The main area displays the file structure under 'serverless-orderflow / lambdas'. The 'checkInventory.js' file is selected and shown in the code editor. The code is as follows:

```
1 export const handler = async (event) => {
2     console.log("Checking inventory for order:", event.orderId);
3     // Simulate inventory check
4     const inStock = Math.random() < 0.9;
5     return {
6         orderId: event.orderId,
7         inStock: inStock
8     };
9 };
```

The screenshot shows the AWS Lambda Functions interface. At the top, there are navigation links for EC2 and S3. Below that, the path is shown as Lambda > Functions > checkInventory. The main area is titled "Code source" with an "Info" tab. On the left, there's an "EXPLORER" sidebar showing a file tree under "CHECKINVENTORY" with "index.mjs" selected. A message indicates "You have undeployed changes." with buttons for "Deploy (Ctrl+Shift+U)" and "Test (Ctrl+Shift+I)". Below that, there's a section for "TEST EVENTS [NONE SELECTED]" with a "+ Create new test event" button. The right side is the code editor for "index.mjs", displaying the following JavaScript code:

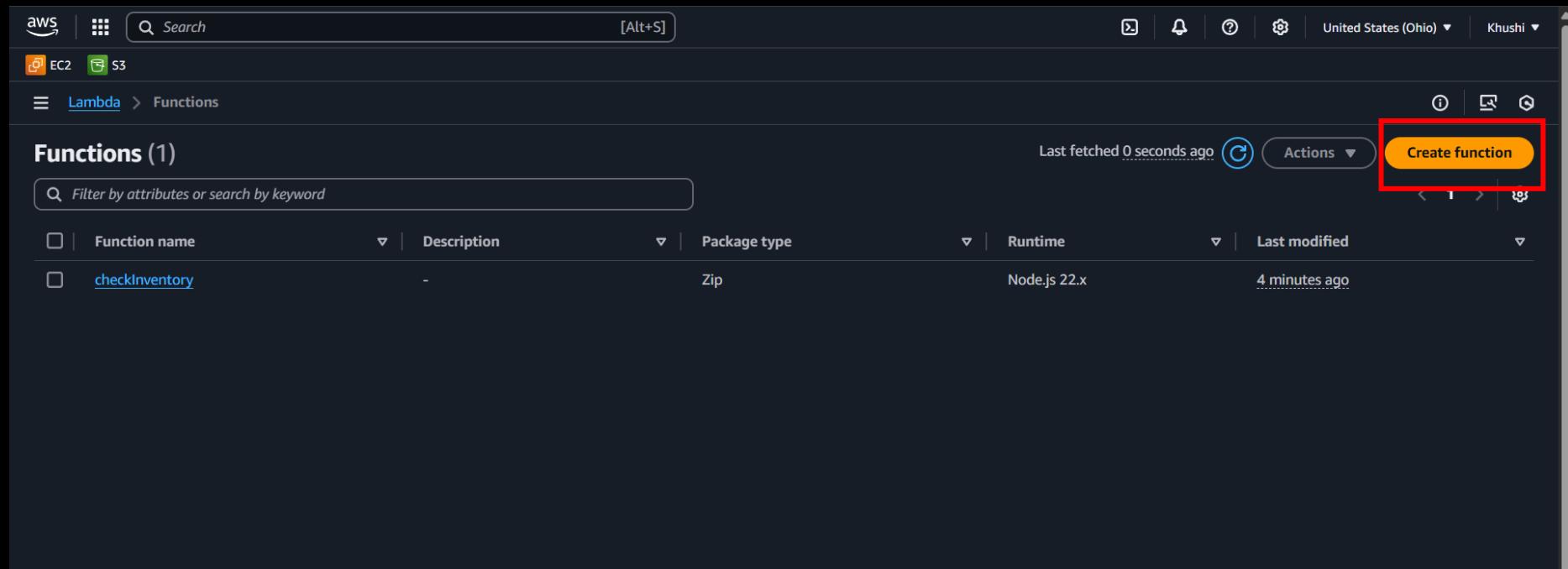
```
1 export const handler = async (event) => {
2   console.log("Checking inventory for order:", event.orderId);
3   // Simulate inventory check
4   const inStock = Math.random() < 0.9;
5   return {
6     orderId: event.orderId,
7     inStock: inStock
8   };
9 };
```

8. Click on “Deploy”



II. CREATE LAMBDA FUNCTION: PROCESSPAYMENT

1. Go back to Functions and click "Create Function"



2. Name the function processPayment

The screenshot shows the AWS Lambda 'Create function' wizard. At the top, there's a navigation bar with the AWS logo, a search bar, and links for EC2 and S3. Below that, the path 'Lambda > Functions > Create function' is shown. On the right, there are several small icons and dropdown menus for region and user.

Create function Info

Choose one of the following options to create your function.

Author from scratch
Start with a simple Hello World example.

Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.

Container image
Select a container image to deploy for your function.

Basic information

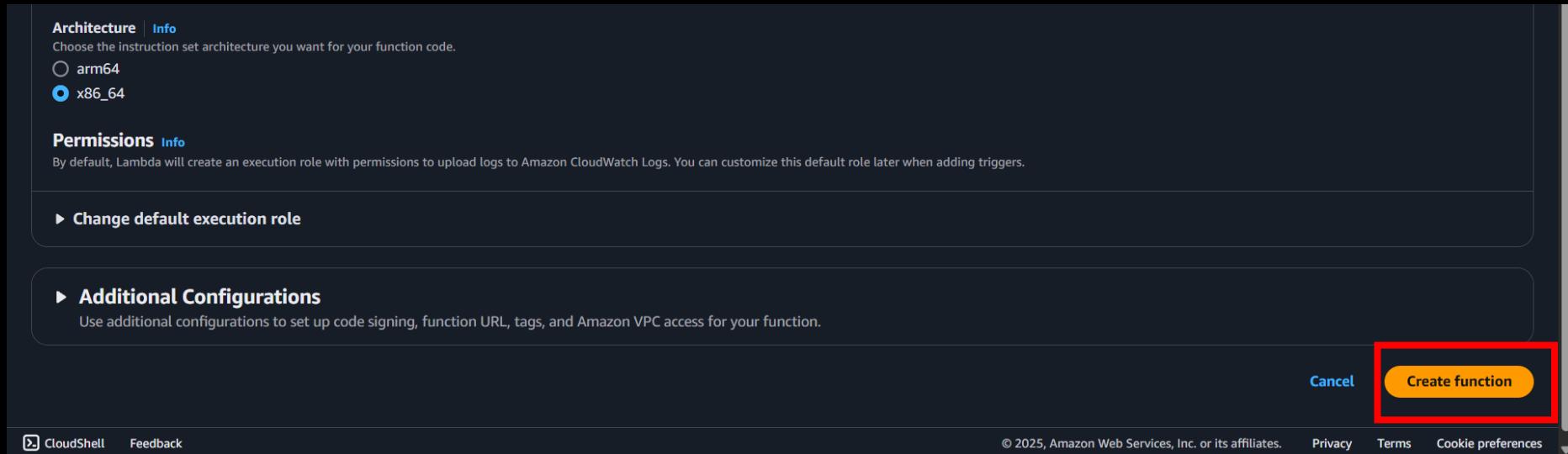
Function name
Enter a name that describes the purpose of your function.

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

3. Runtime: Node.js 20.x (default)

The screenshot shows the 'Runtime' configuration step. It has a 'Runtime' tab selected, with an 'Info' link next to it. A note says: 'Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.' Below this is a dropdown menu set to 'Node.js 22.x'. To the right of the dropdown is a blue circular icon with a white 'C' inside, which likely represents a copy or refresh function.

4. Click “Create Function”



The screenshot shows the AWS Lambda console interface. At the top, there's a navigation bar with links for AWS Home, EC2, S3, Lambda, Functions, and processPayment. A search bar and a help icon are also present. On the right, it shows the region as United States (Ohio) and a user named Khushi.

In the main area, a green success message states: "Successfully created the function processPayment. You can now change its code and configuration. To invoke your function with a test event, choose 'Test'." Below this, the function name "processPayment" is displayed in bold.

The "Function overview" section is expanded, showing:

- Diagram** (selected tab): A small preview of the function's architecture.
- Template**: A link to view the CloudFormation template.
- Layers**: 0 layers listed.
- Add trigger**: A button to add triggers for the function.
- Add destination**: A button to add destinations for the function.

To the right, there are several buttons: Throttle, Copy ARN, Actions, Export to Infrastructure Composer, and Download.

The "Description" field is empty. The "Last modified" field shows the function was last modified "in 3 seconds". The "Function ARN" field contains the value: arn:aws:lambda:us-east-2:971422709086:function:processPayment. The "Function URL" field is also present but empty.

5. Paste the code in the editor

The screenshot shows a GitHub repository interface for a project named 'serverless-orderflow'. The left sidebar displays a tree view of files: 'main' (selected), 'lambdas' (expanded, showing 'checkInventory.js', 'processPayment.js' (selected), and 'updateOrderStatus.js'), 'state-machine', '.gitignore', and 'README.md'. The right main area shows the contents of 'processPayment.js'. The code is as follows:

```
1 export const handler = async (event) => {
2     console.log("Processing payment for order:", event.orderId);
3     // Simulate payment processing
4     const paymentSuccessful = Math.random() < 0.9;
5     return {
6         orderId: event.orderId,
7         paymentSuccessful: paymentSuccessful
8     };
9 };
```

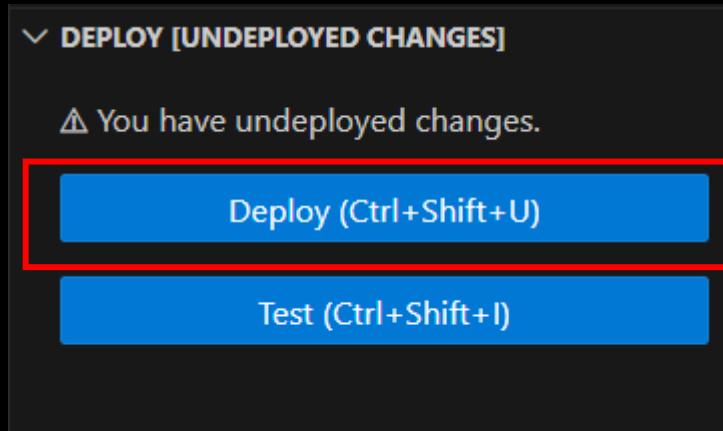
The screenshot shows the AWS Lambda Functions console interface. At the top, there are navigation links for EC2 and S3, and a search bar with the placeholder "Search". On the right side of the header, there are icons for notifications, help, and account settings, along with the text "United States (Ohio)" and "Khushi". Below the header, the breadcrumb navigation shows "Lambda > Functions > processPayment".

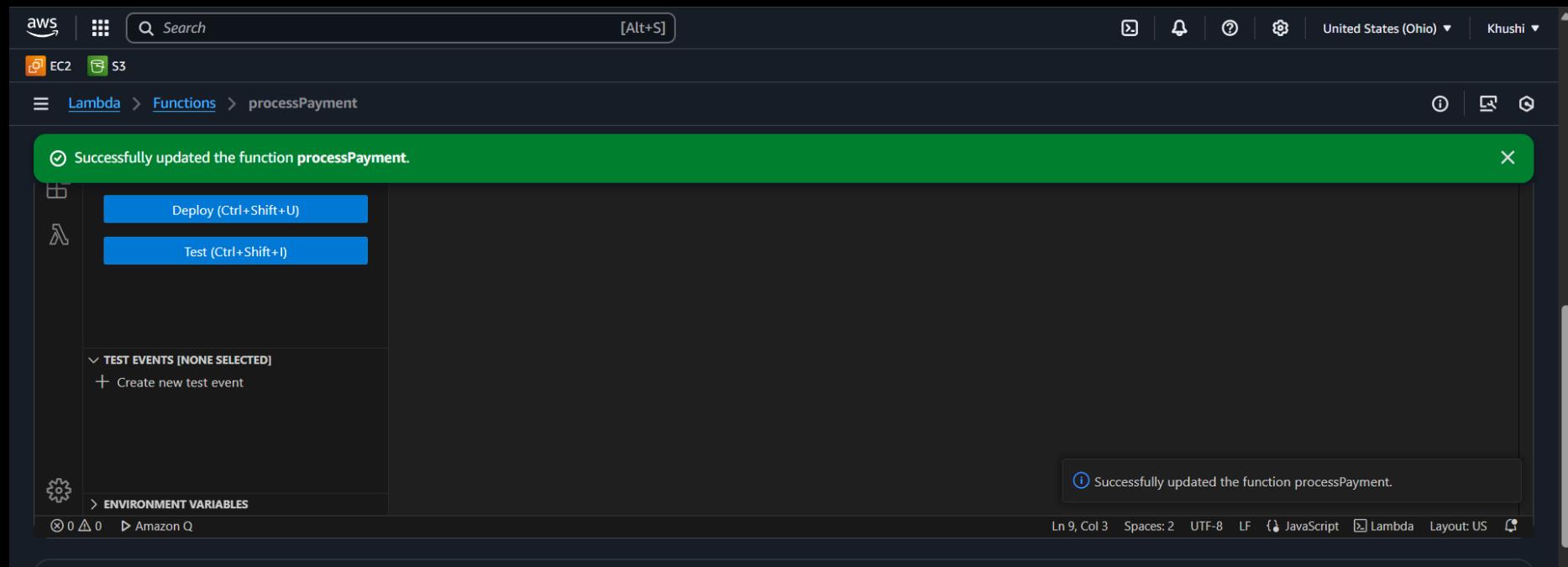
The main area is titled "Code source" with an "Info" link. It features a code editor with a dark theme, displaying the file "JS index.mjs". The code is as follows:

```
JS index.mjs > ...
1 export const handler = async (event) => {
2   console.log("Processing payment for order:", event.orderId);
3   // Simulate payment processing
4   const paymentSuccessful = Math.random() < 0.9;
5   return {
6     orderId: event.orderId,
7     paymentSuccessful: paymentSuccessful
8   };
9 };
```

To the left of the code editor is the "EXPLORER" sidebar. Under the "PROCESSPAYMENT" project, the file "index.mjs" is listed. Below the project list, there is a section titled "DEPLOY [UNDEPLOYED CHANGES]" which says "You have undeployed changes." with buttons for "Deploy (Ctrl+Shift+U)" and "Test (Ctrl+Shift+I)". At the bottom of the sidebar, there is a section titled "TEST EVENTS [NONE SELECTED]" with a "+ Create new test event" button.

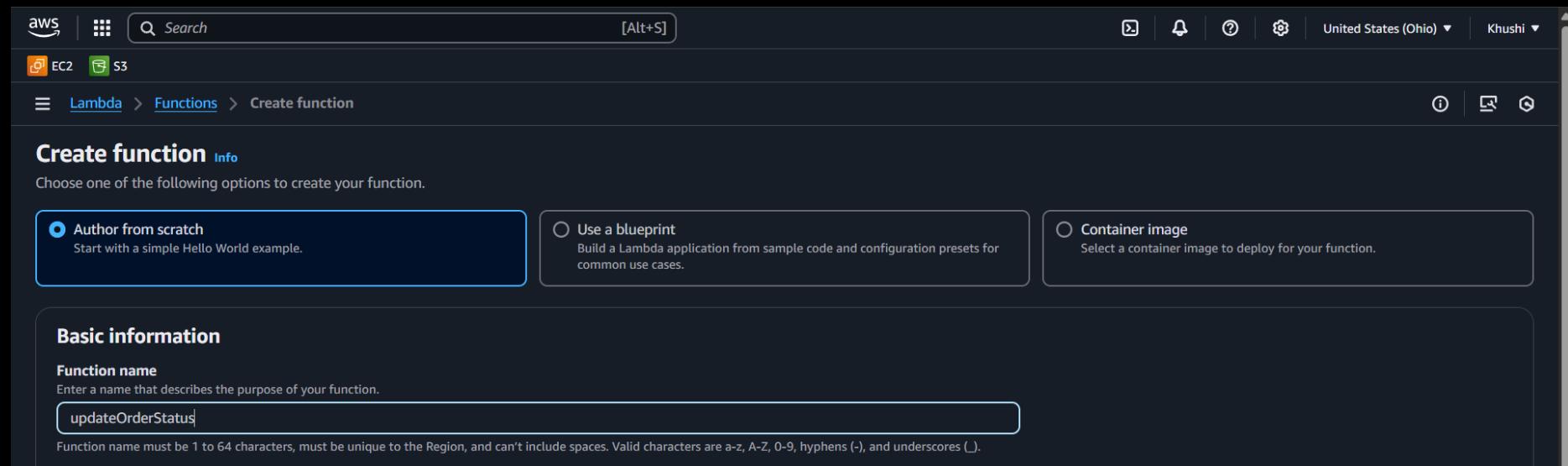
6. Click on “Deploy”





III. CREATE LAMBDA FUNCTION: UPDATEORDERSTATUS

1. Create another Lambda function named updateOrderStatus



2. Leave all the other settings as default

The screenshot shows the 'Runtime' section with 'Node.js 22.x' selected. In the 'Architecture' section, 'x86_64' is chosen. Under 'Permissions', it notes that Lambda will create an execution role with CloudWatch Logs permissions. Two buttons are visible: 'Change default execution role' and 'Additional Configurations'.

Runtime | [Info](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Node.js 22.x ▾ [C](#)

Architecture | [Info](#)
Choose the instruction set architecture you want for your function code.
 arm64
 x86_64

Permissions [Info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▶ Change default execution role

▶ Additional Configurations
Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

3. Click “Create Function”

The screenshot shows the 'Additional Configurations' section with a note about setting up code signing, function URL, tags, and Amazon VPC access. At the bottom right, the 'Create function' button is highlighted with a red box.

▶ Additional Configurations
Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

[Cancel](#) [Create function](#)

CloudShell Feedback © 2025, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

The screenshot shows the AWS Lambda console interface. At the top, there's a navigation bar with icons for AWS, search, and various services like EC2 and S3. The main title is "Lambda > Functions > updateOrderStatus". A green success message box says: "Successfully created the function updateOrderStatus. You can now change its code and configuration. To invoke your function with a test event, choose 'Test'." Below the message, the function name "updateOrderStatus" is displayed in bold. There are buttons for "Throttle", "Copy ARN", and "Actions". On the left, under "Function overview", there are tabs for "Diagram" (which is selected) and "Template". The "Diagram" view shows a single Lambda function icon with the name "updateOrderStatus", a "Layers" section (0 layers), and buttons for "+ Add trigger" and "+ Add destination". On the right, there are sections for "Description" (empty), "Last modified" (7 seconds ago), "Function ARN" (arn:aws:lambda:us-east-2:971422709086:function:updateOrderStatus), and "Function URL" (Info). There are also "Export to Infrastructure Composer" and "Download" buttons.

4. Paste the code in the editor

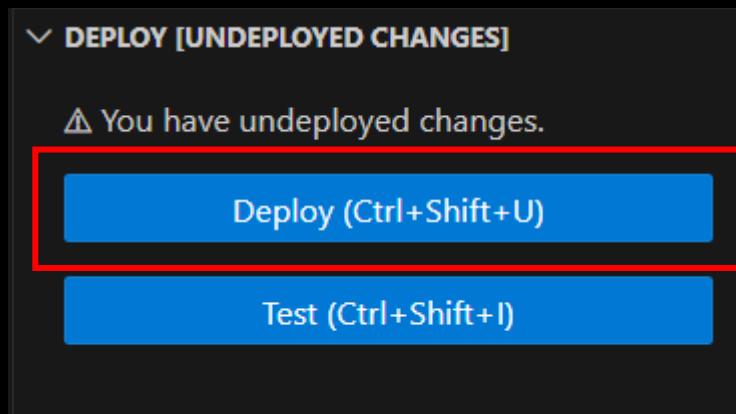
The screenshot shows a GitHub repository interface for a project named 'serverless-orderflow'. The repository has a dark theme. On the left, there's a sidebar with a 'Files' section showing a tree view of files. The 'updateOrderStatus.js' file is selected and highlighted with a blue border. The main area displays the content of this file:

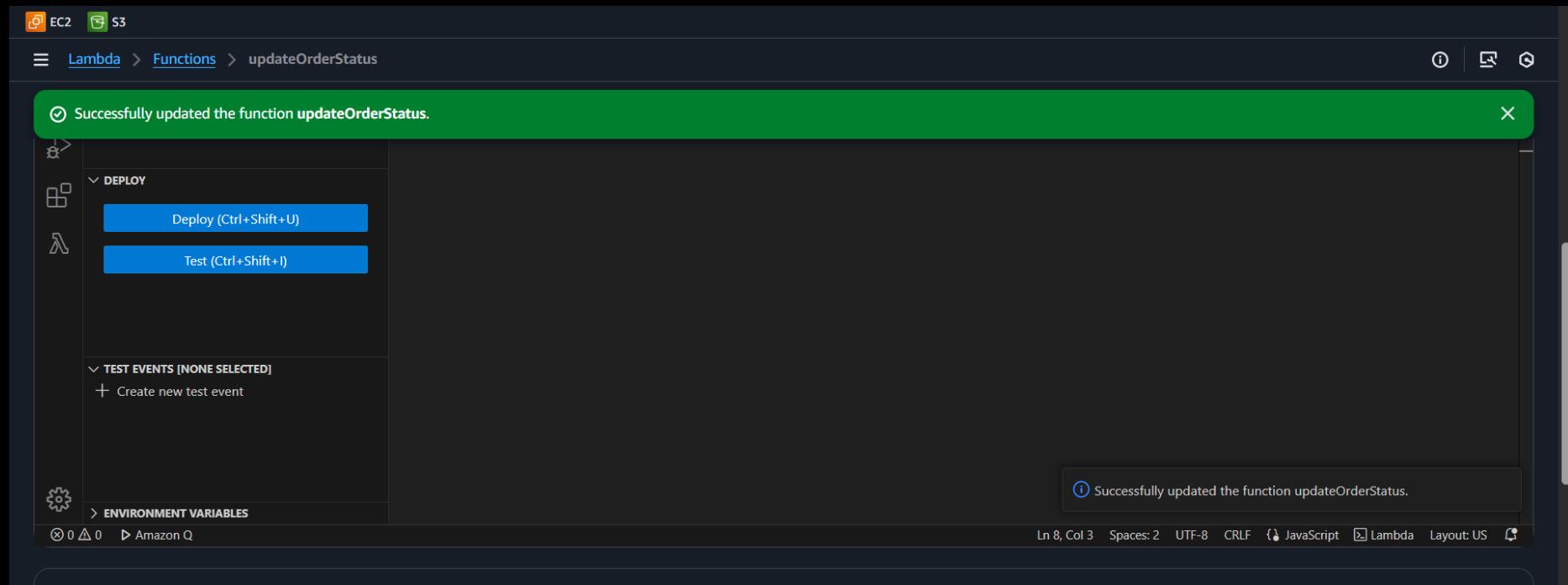
```
1 export const handler = async (event) => {
2     console.log("Updating status for order:", event.orderId);
3     // Simulate order status update
4     return {
5         orderId: event.orderId,
6         status: "Completed"
7     };
8 };
```

The screenshot shows the AWS Lambda Functions console interface. At the top, there are navigation links for EC2 and S3, followed by the Lambda menu, Functions, and the specific function name, updateOrderStatus. Below this is a toolbar with various icons and a search bar containing the text "updateOrderStatus". On the left side, there's a sidebar with sections for EXPLORER, UPDATEORDERSTATUS (containing a file named index.mjs), DEPLOY [UNDEPLOYED CHANGES] (with a Deploy button), and TEST EVENTS [NONE SELECTED] (with a Create new test event button). The main area displays the code editor with the file index.mjs open. The code is as follows:

```
JS index.mjs > ...
1 export const handler = async (event) => {
2   console.log("Updating status for order:", event.orderId);
3   // Simulate order status update
4   return {
5     orderId: event.orderId,
6     status: "Completed"
7   };
8 };
```

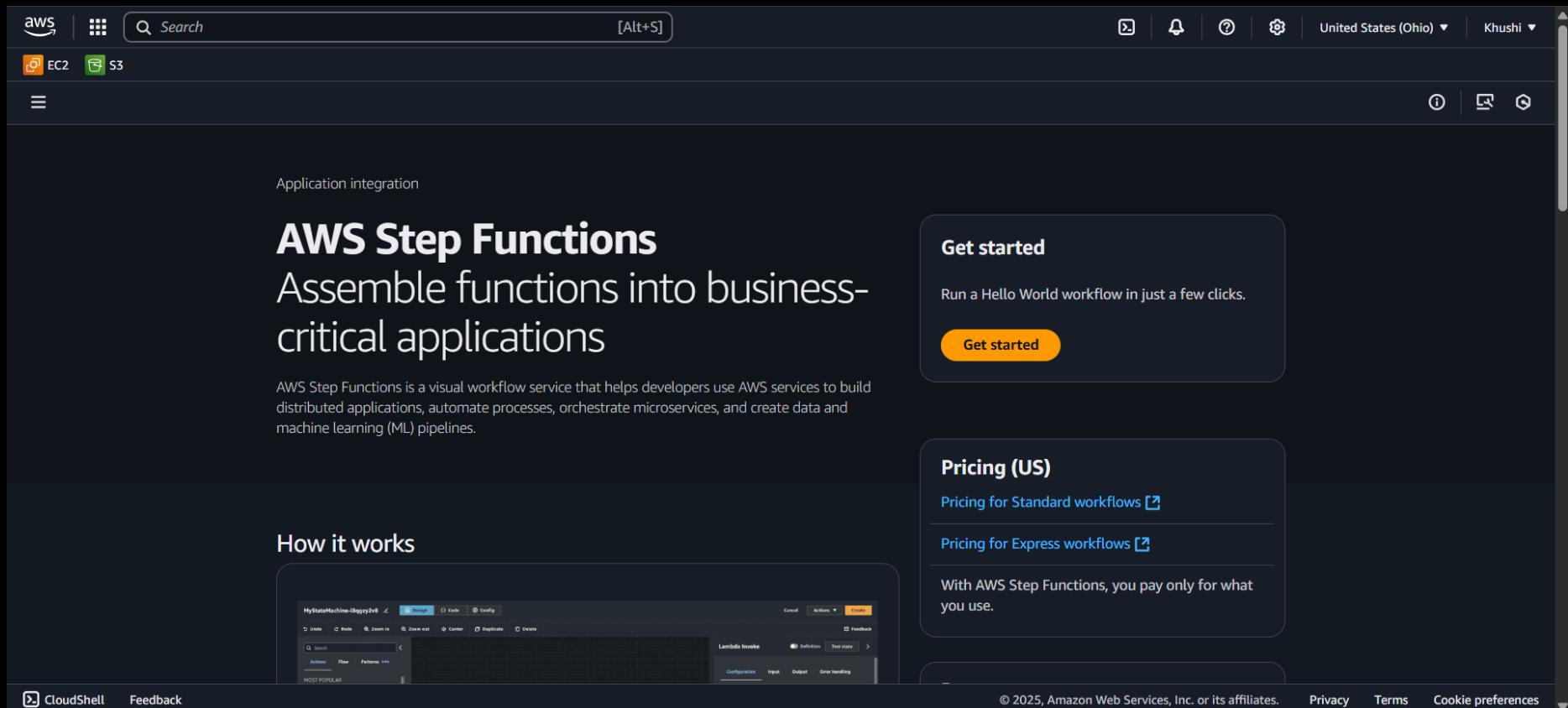
5. Click on “Deploy”



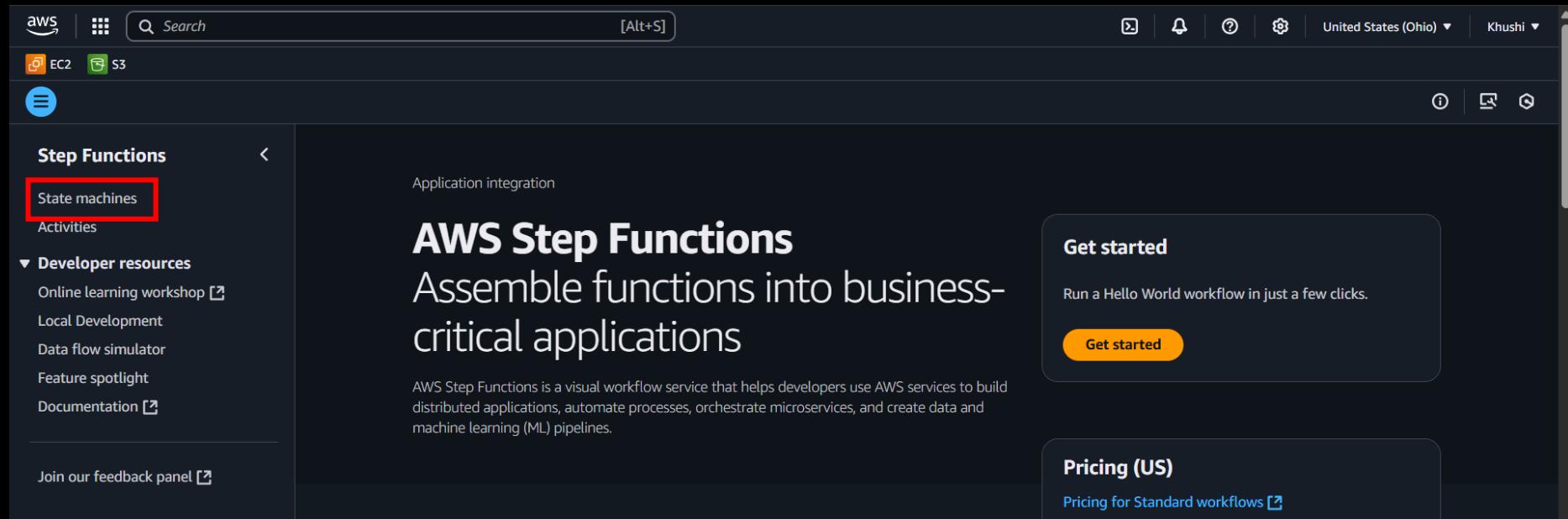


IV. CONFIGURE AWS STEP FUNCTIONS

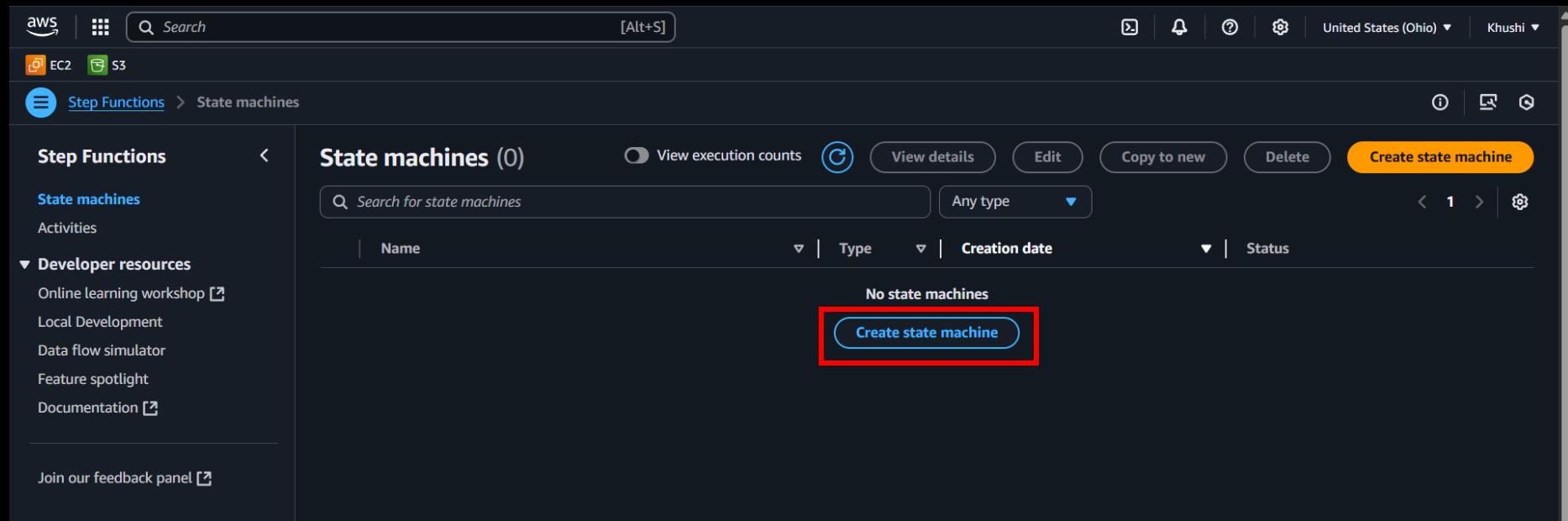
1. Go to AWS Step Functions from the Console



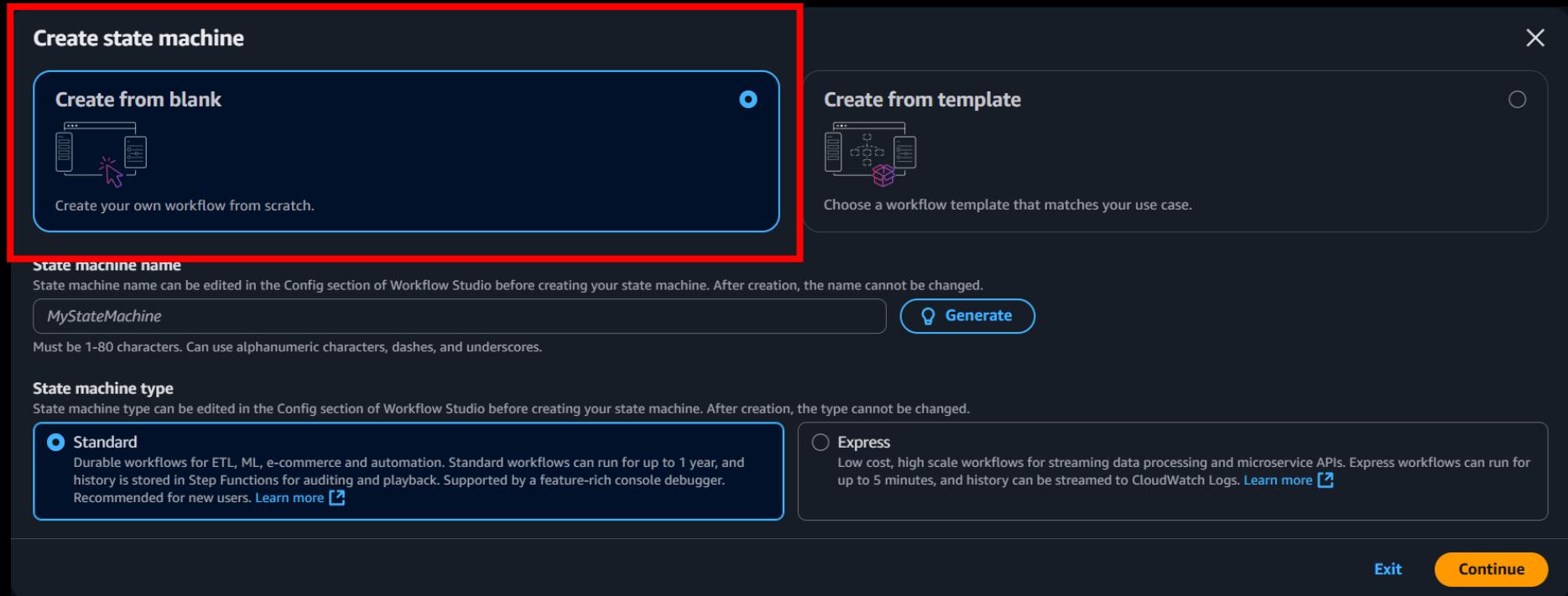
2. In the sidebar, click on “State Machines”



3. Click “Create State Machine”



4. Choose “Blank Template” and click “Select”



5. Give your state-machine a name and click on continue

State machine name
State machine name can be edited in the Config section of Workflow Studio before creating your state machine. After creation, the name cannot be changed.

[Generate](#)

Must be 1-80 characters. Can use alphanumeric characters, dashes, and underscores.

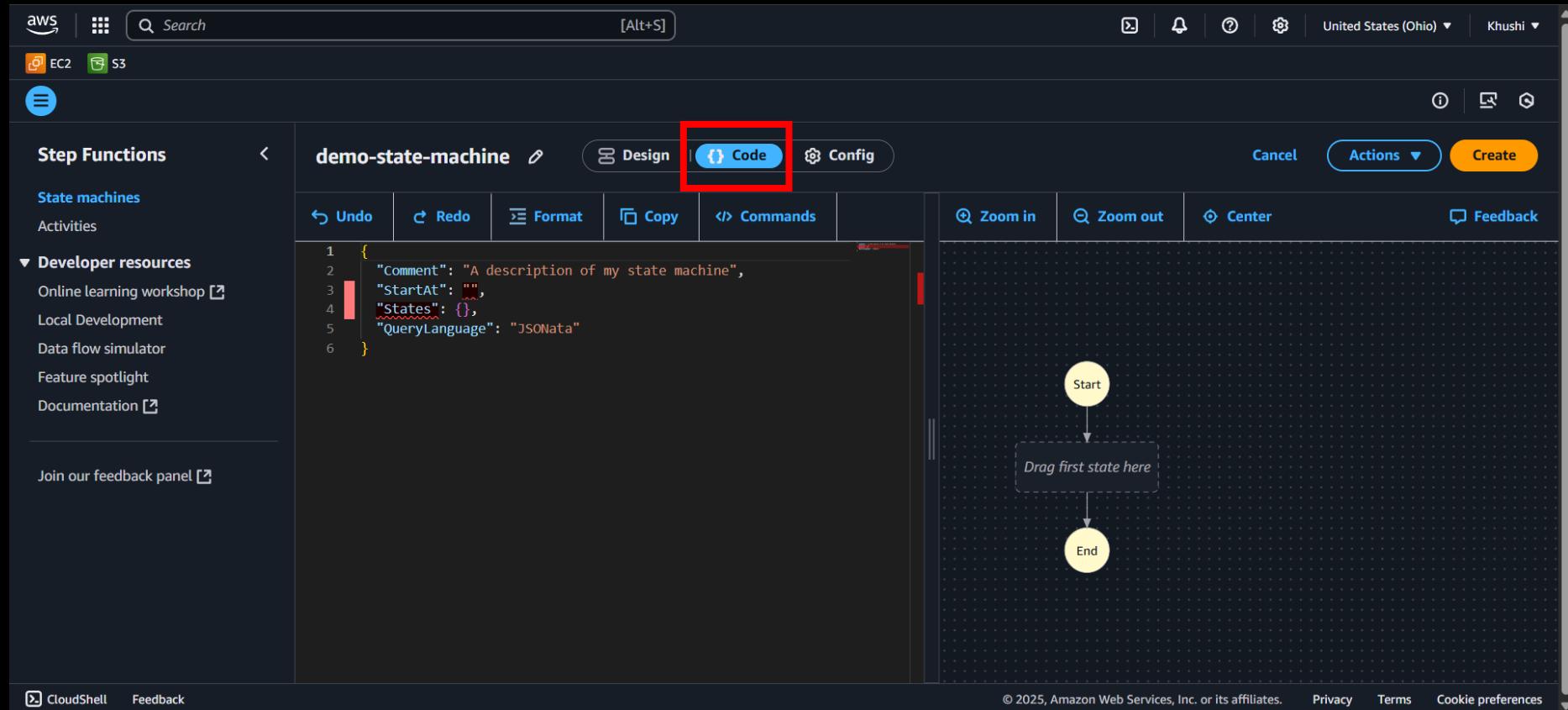
State machine type
State machine type can be edited in the Config section of Workflow Studio before creating your state machine. After creation, the type cannot be changed.

Standard
Durable workflows for ETL, ML, e-commerce and automation. Standard workflows can run for up to 1 year, and history is stored in Step Functions for auditing and playback. Supported by a feature-rich console debugger.
Recommended for new users. [Learn more](#)

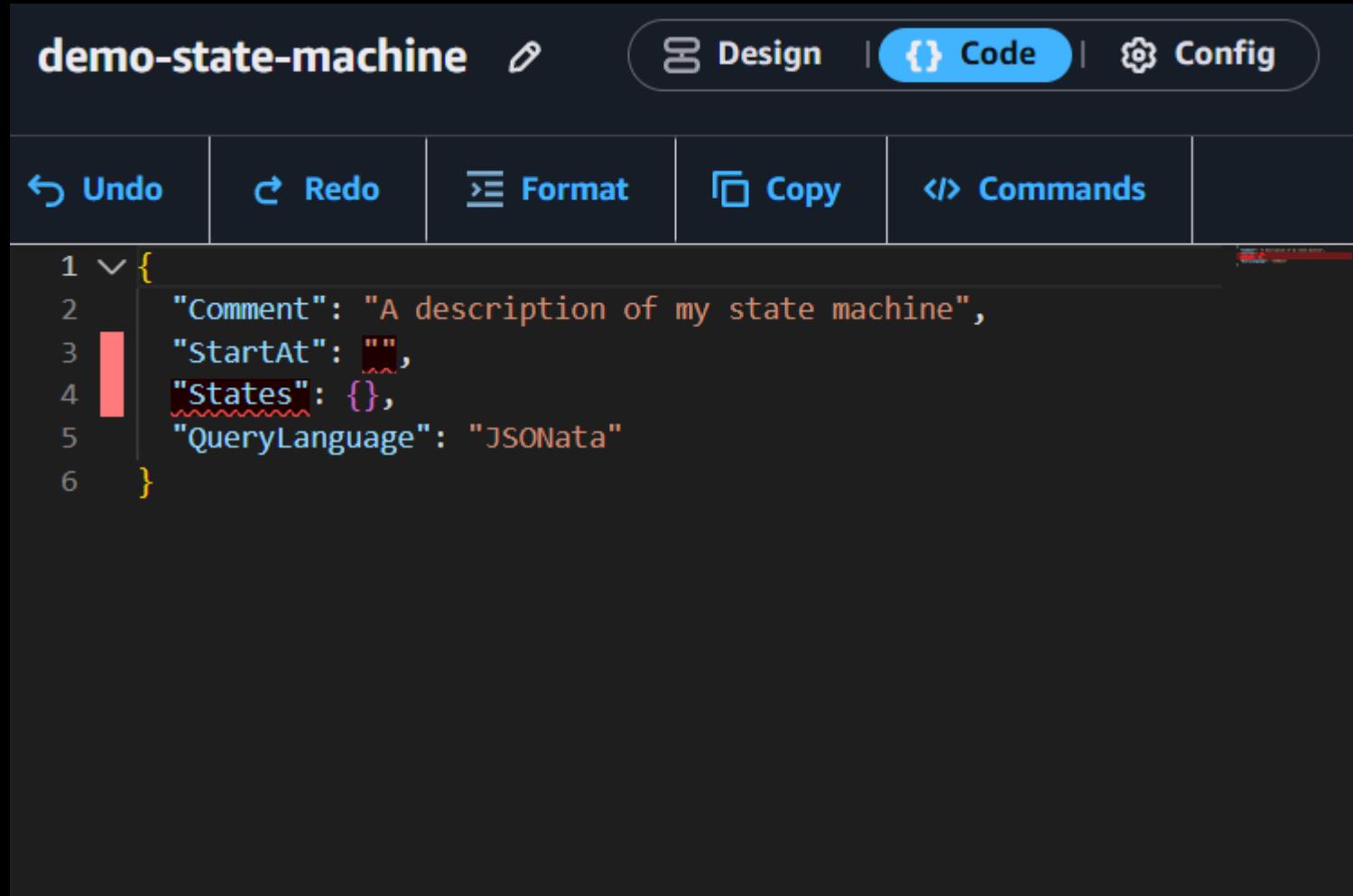
Express
Low cost, high scale workflows for streaming data processing and microservice APIs. Express workflows can run for up to 5 minutes, and history can be streamed to CloudWatch Logs. [Learn more](#)

[Exit](#) [Continue](#)

6. Go to the “Code” tab

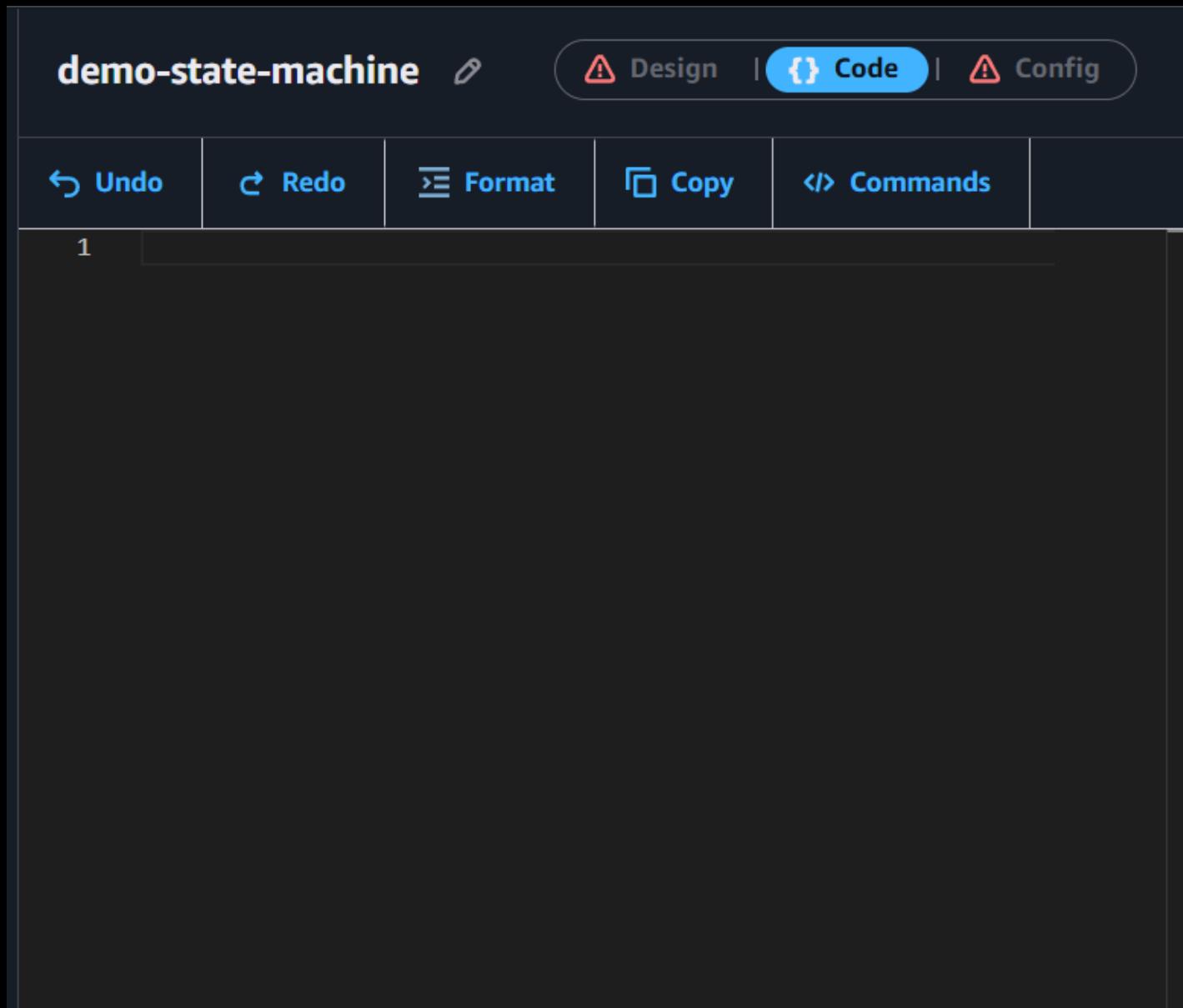


7. Remove default code

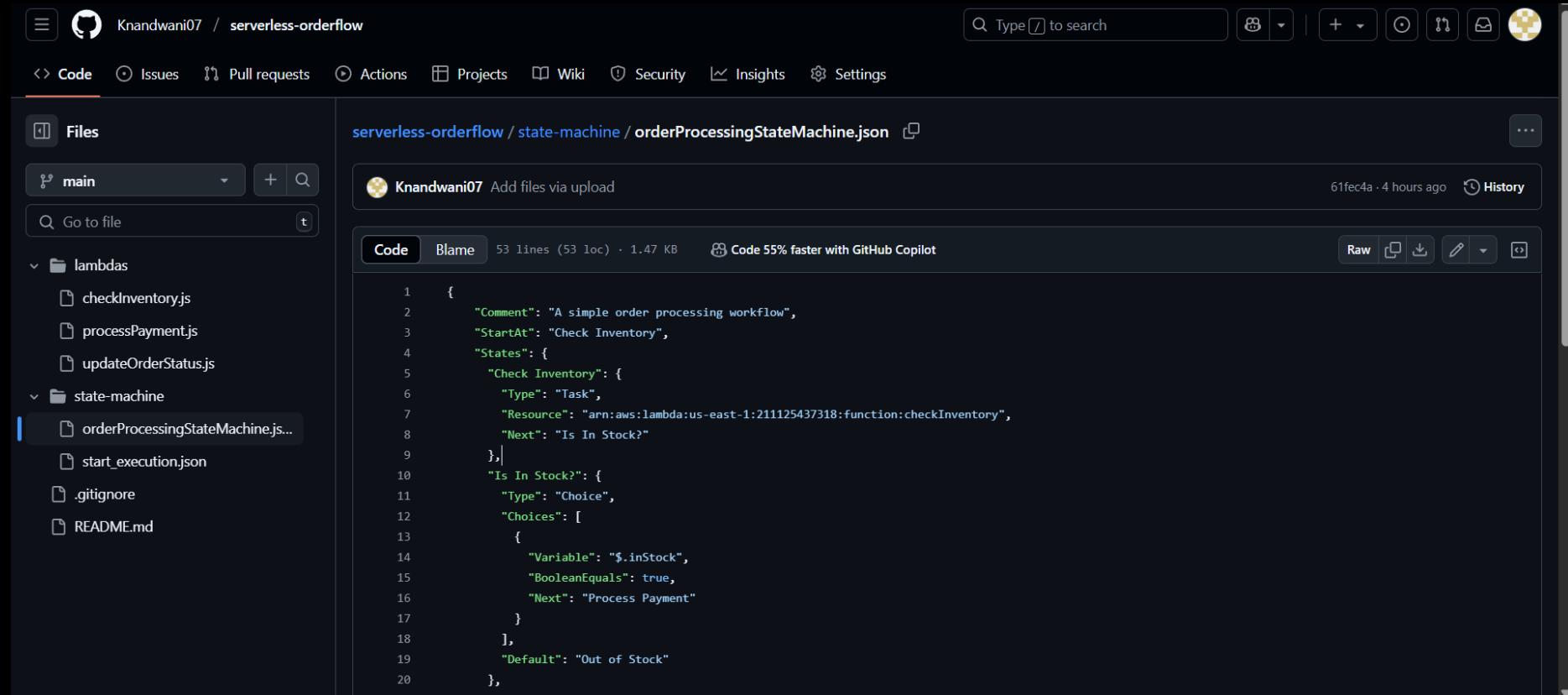


The screenshot shows a software interface for managing state machines. At the top, there's a navigation bar with tabs: 'Design', 'Code' (which is highlighted in blue), and 'Config'. Below the tabs is a toolbar with buttons for 'Undo', 'Redo', 'Format', 'Copy', and 'Commands'. The main area is a code editor displaying the following JSON configuration:

```
1  {
2      "Comment": "A description of my state machine",
3      "StartAt": "",
4      "States": {},
5      "QueryLanguage": "JSONNata"
6  }
```



8. Paste your Step Functions JSON code



The screenshot shows a GitHub repository interface for the 'serverless-orderflow' project. The repository name is 'Knardwani07 / serverless-orderflow'. The 'Code' tab is selected in the navigation bar. On the left, the file tree shows a 'main' folder containing 'checkInventory.js', 'processPayment.js', and 'updateOrderStatus.js'. Below 'main' is a 'lambdas' folder with the same three files. Under 'lambdas' is a 'state-machine' folder containing 'orderProcessingStateMachine.json' (which is highlighted) and 'start_execution.json'. There is also a '.gitignore' file and a 'README.md' file. The 'orderProcessingStateMachine.json' file is open in the main editor area. The code content is as follows:

```
1  {
2      "Comment": "A simple order processing workflow",
3      "StartAt": "Check Inventory",
4      "States": {
5          "Check Inventory": {
6              "Type": "Task",
7              "Resource": "arn:aws:lambda:us-east-1:211125437318:function:checkInventory",
8              "Next": "Is In Stock?"
9          },
10         "Is In Stock?": {
11             "Type": "Choice",
12             "Choices": [
13                 {
14                     "Variable": "$.inStock",
15                     "BooleanEquals": true,
16                     "Next": "Process Payment"
17                 }
18             ],
19             "Default": "Out of Stock"
20         },
21     }
}
```

Screenshot of the AWS Step Functions console showing the "Code" tab for a state machine named "demo-state-machine".

Left Sidebar:

- Step Functions** (selected)
- State machines**
- Activities**
- Developer resources**
 - Online learning workshop
 - Local Development
 - Data flow simulator
 - Feature spotlight
 - Documentation
- Join our feedback panel**

Top Bar:

- Search bar
- [Alt+S]
- Icons for EC2 and S3
- Notification bell
- Region: United States (Ohio)
- User: Khushi

Code Tab:

The code editor displays the state machine definition:

```

1  [
2    "Comment": "A simple order processing workflow",
3    "StartAt": "Check Inventory",
4    "States": {
5      "Check Inventory": {
6        "Type": "Task",
7        "Resource": "arn:aws:lambda:us-east-1:211125437318:function:check-inventory",
8        "Next": "Is In Stock?"
9      },
10     "Is In Stock?": {
11       "Type": "Choice",
12       "Choices": [
13         {
14           "Variable": "$.inStock",
15           "BooleanEquals": true,
16           "Next": "Process Payment"
17         }
18       ],
19       "Default": "Out of Stock"
20     },
21     "Process Payment": {
22       "Type": "Task",
23       "Resource": "arn:aws:lambda:us-east-1:211125437318:function:process-payment"
24     }
25   }
26 }
```

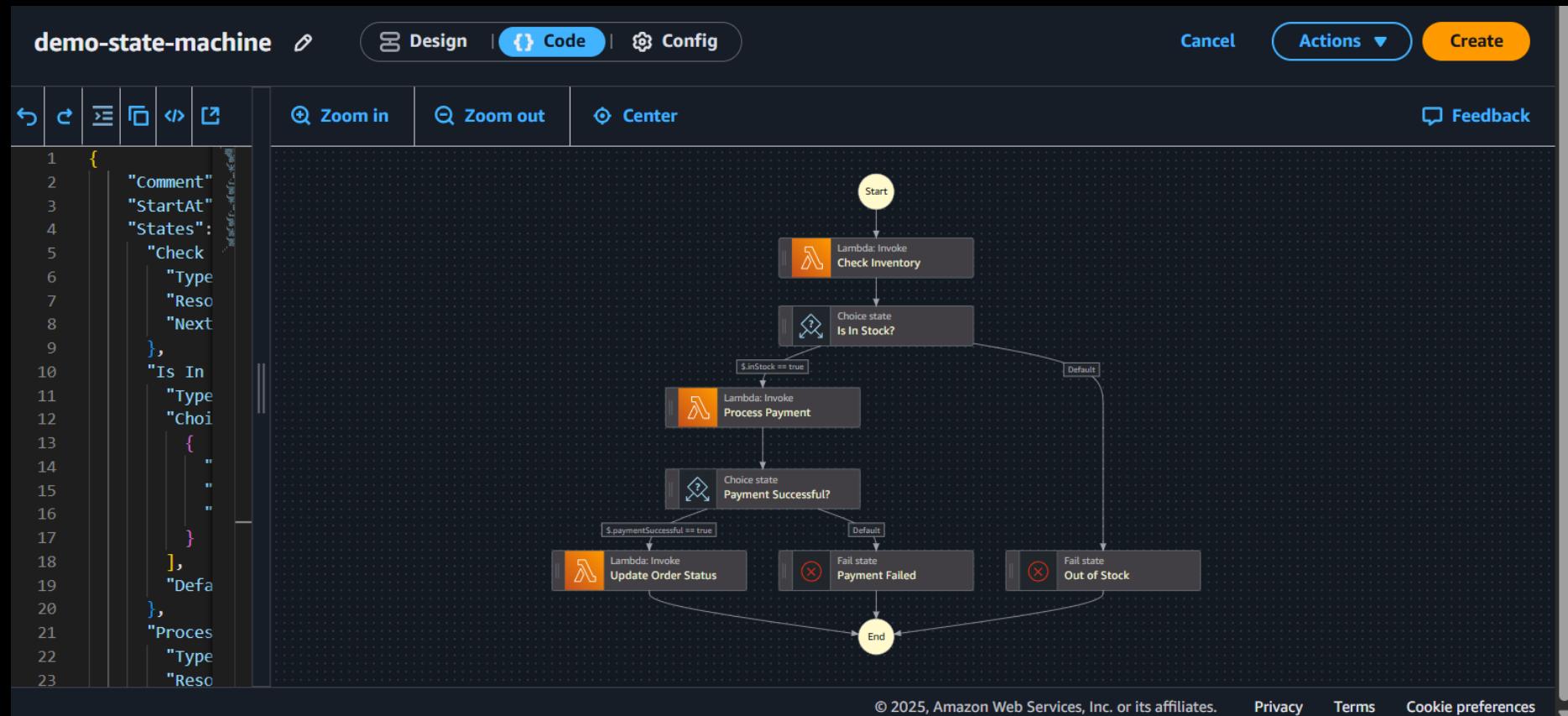
Right Panel:

```

graph TD
    Start((Start)) --> Lambda1[Lambda: Invoke Check Inventory]
    Lambda1 --> Choice{Choice state Is In Stock?}
    Choice -- $inStock == true --> Lambda2[Lambda: Invoke Process Payment]
    Choice -- $inStock == false --> OutOfStock[Out of Stock]
    
```

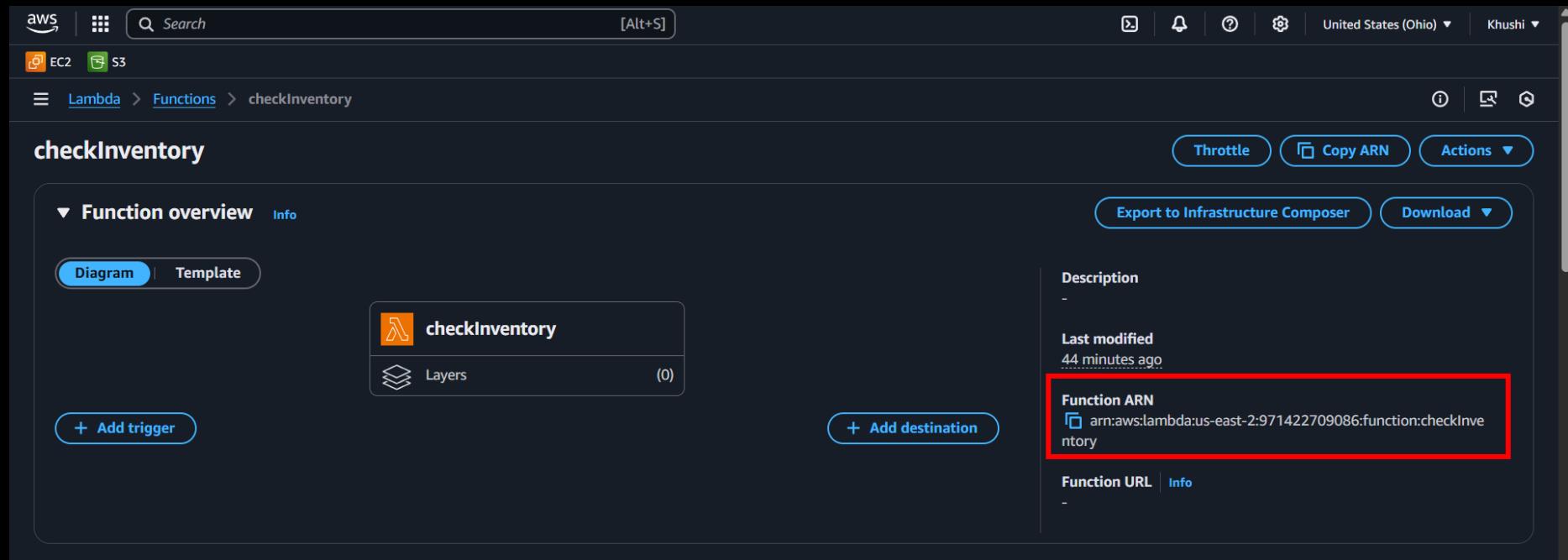
The diagram illustrates the workflow:

- The process begins at the **Start** state.
- An **Lambda: Invoke Check Inventory** step is executed.
- A **Choice state Is In Stock?** decision point follows. If $\$inStock == \text{true}$, it proceeds to the **Lambda: Invoke Process Payment** step. If $\$inStock == \text{false}$, it proceeds to the **Out of Stock** state.



9. Replace the placeholder ARN values with actual Lambda ARNs

- o Copy ARN from each Lambda function



- Replace only the account ID and function names if other parts match

The screenshot shows a code editor interface for a state machine named "demo-state-machine". The top navigation bar includes "Design", "Code" (which is selected), and "Config" tabs. A red banner at the top right says "Workflow not created". Below the tabs is a toolbar with "Undo", "Redo", "Format", "Copy", "Commands", and "View docs" buttons.

BEFORE:

```
1 {  
2   "Comment": "A simple order processing workflow",  
3   "StartAt": "Check Inventory",  
4   "States": {  
5     "Check Inventory": {  
6       "Type": "Task",  
7       "Resource": "arn:aws:lambda:us-east-1:211125437318:function:checkInventory",  
8       "Next": "Is In Stock?"  
9     },  
10    "Is In Stock?": {  
11      "Type": "Decision",  
12      "Choices": [ {  
13        "Variable": "$.inStock",  
14        "Value": true,  
15        "Next": "Send Order"  
16      }, {  
17        "Variable": "$.inStock",  
18        "Value": false,  
19        "Next": "Notify Customer"  
20      } ]  
21    }  
22  }  
23 }  
24 }
```

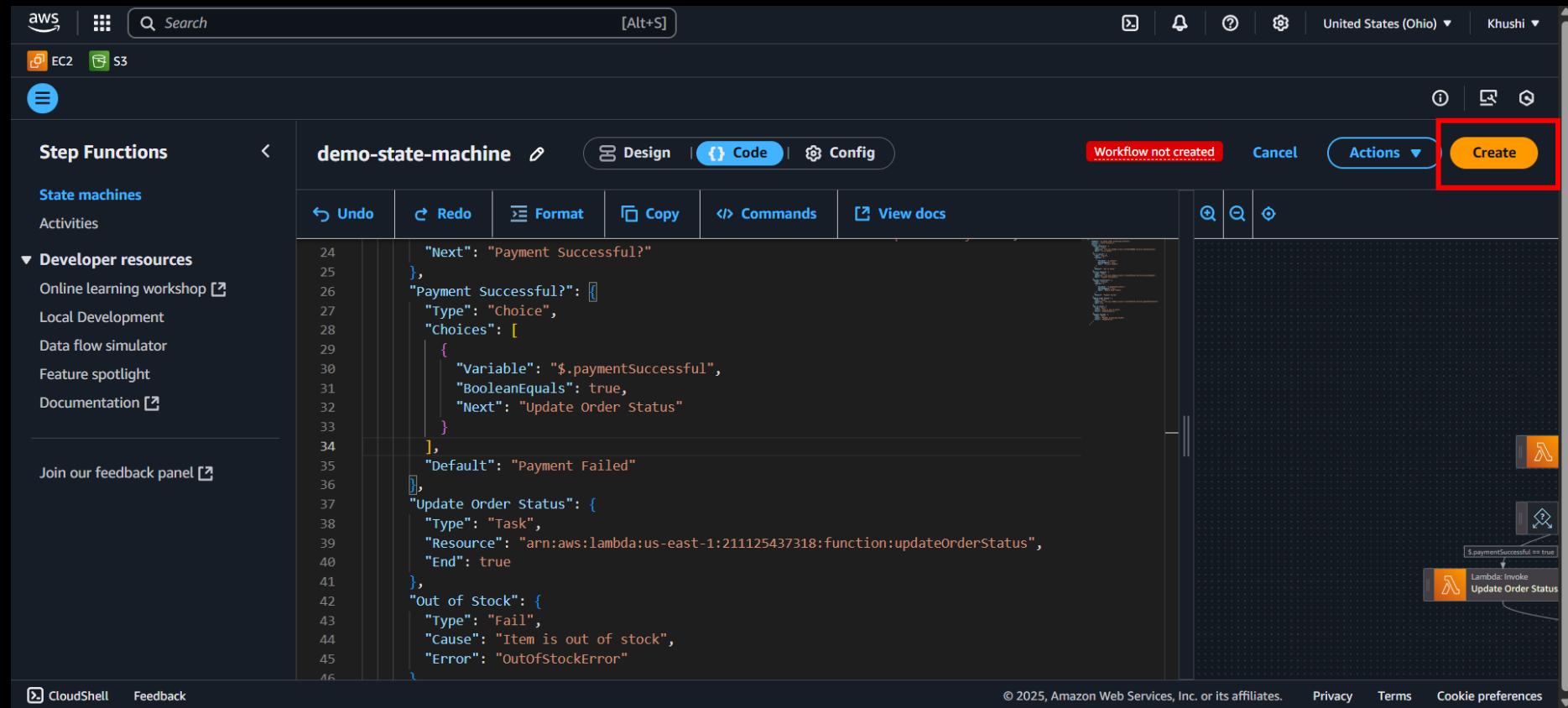
AFTER:

```
1 {  
2   "Comment": "A simple order processing workflow",  
3   "StartAt": "Check Inventory",  
4   "States": {  
5     "Check Inventory": {  
6       "Type": "Task",  
7       "Resource": "arn:aws:lambda:us-east-2:971422709086:function:checkInventory",  
8       "Next": "Is In Stock?"  
9     },  
10    "Is In Stock?": {  
11      "Type": "Decision",  
12      "Choices": [ {  
13        "Variable": "$.inStock",  
14        "Value": true,  
15        "Next": "Send Order"  
16      }, {  
17        "Variable": "$.inStock",  
18        "Value": false,  
19        "Next": "Notify Customer"  
20      } ]  
21    }  
22  }  
23 }  
24 }
```

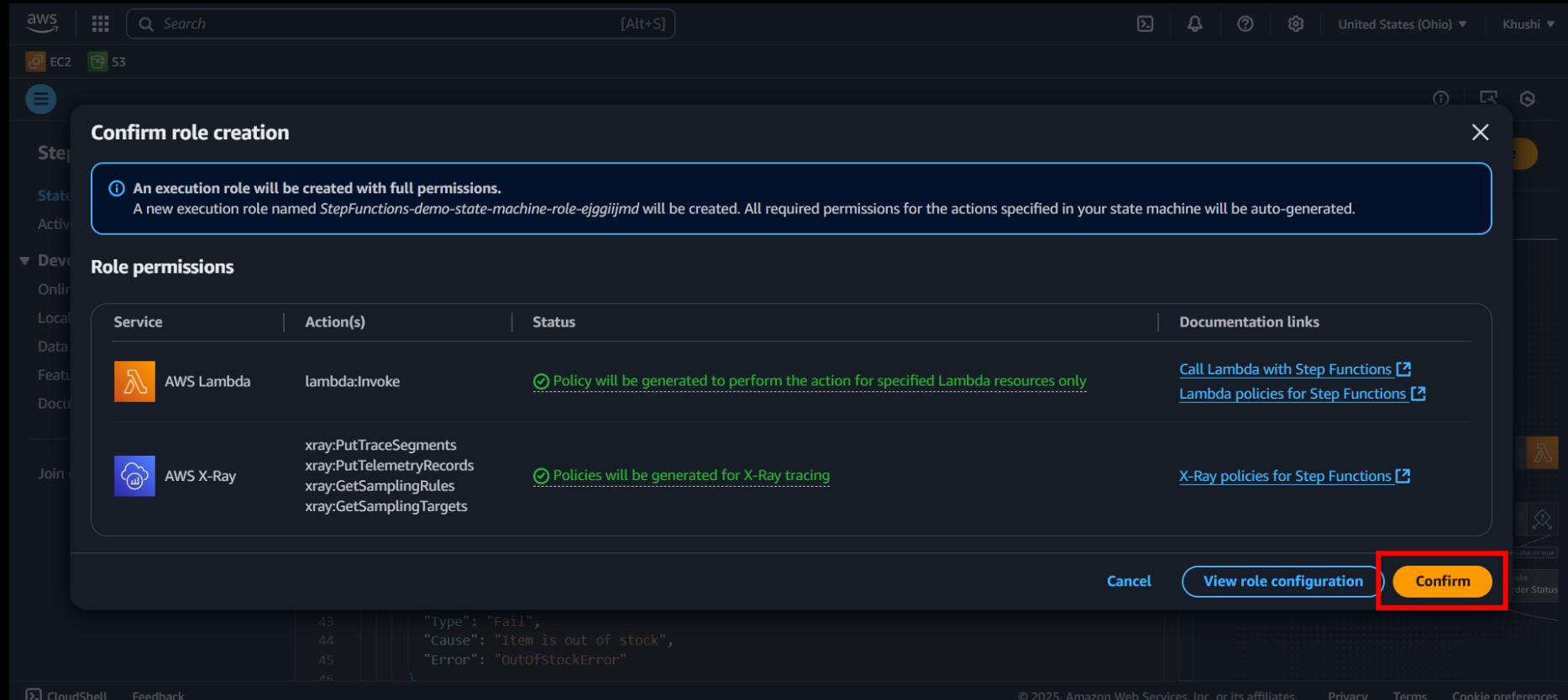
The "Resource" field in the "Check Inventory" state has been updated from "arn:aws:lambda:us-east-1:211125437318:function:checkInventory" to "arn:aws:lambda:us-east-2:971422709086:function:checkInventory". The "StartAt" state remains "Check Inventory". The "Is In Stock?" state is a decision state with two branches based on the value of "\$.inStock".

Change for the other 2 functions too!

10. Click “Create State Machine”

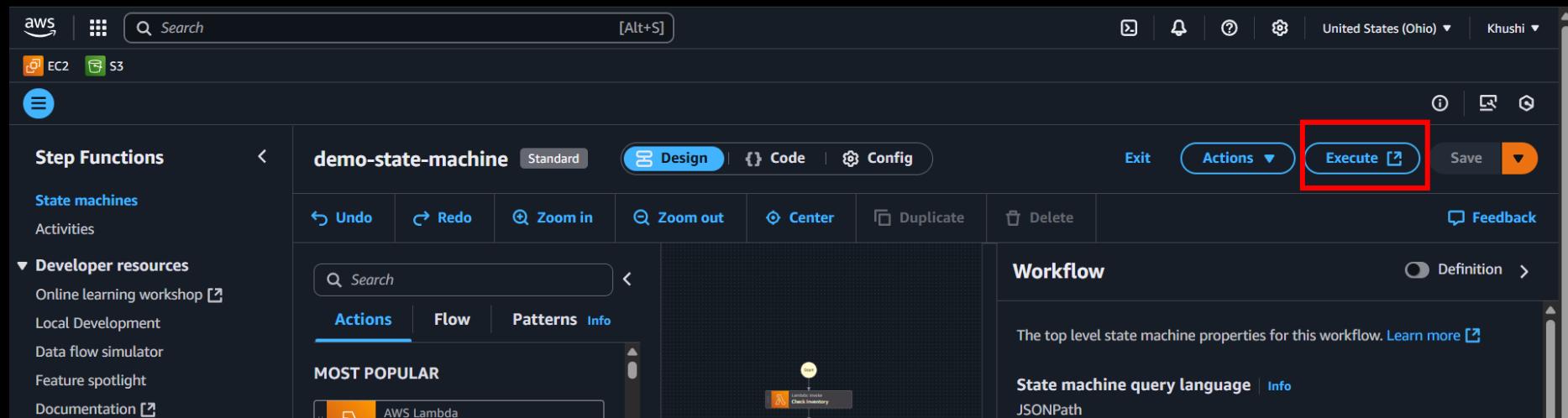


11. When prompted, confirm IAM role creation

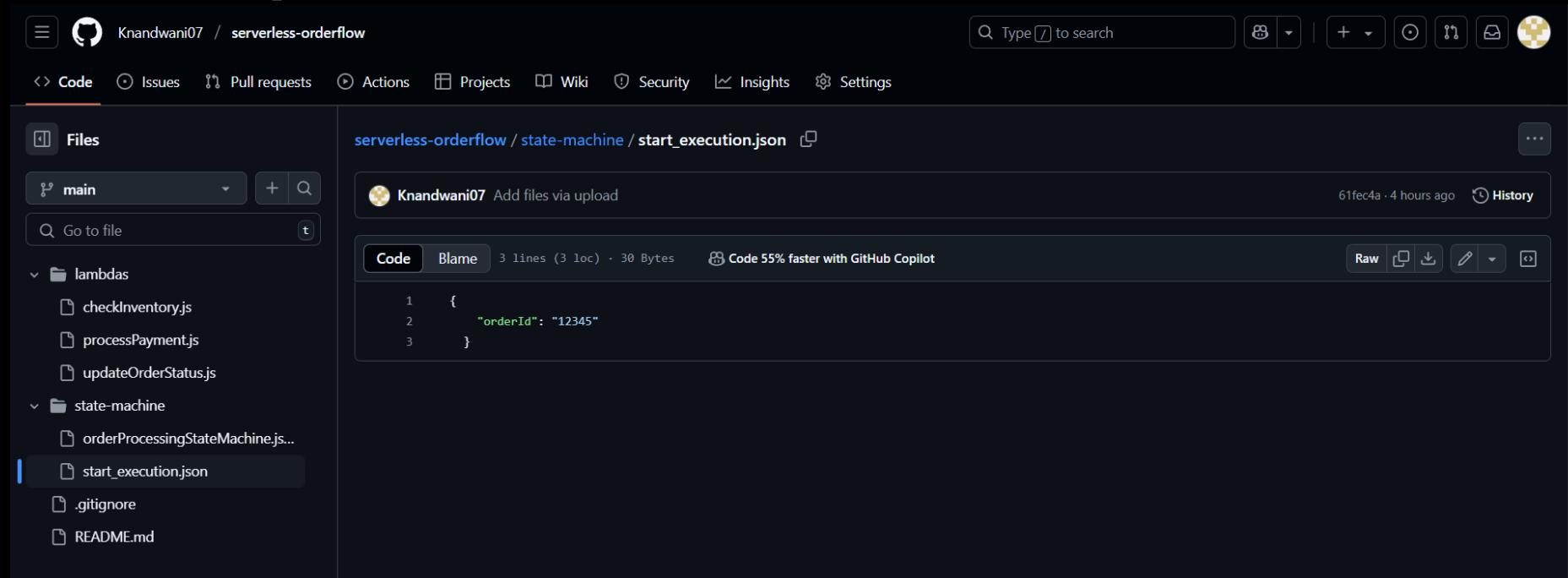


V. EXECUTE THE STATE MACHINE 🚀

1. Click On “Execute”



2. Enter JSON input



The screenshot shows a GitHub repository named "serverless-orderflow". The "Code" tab is selected. On the left, the file tree shows "main" and "lambdas" folders containing "checkInventory.js", "processPayment.js", and "updateOrderStatus.js". The "state-machine" folder contains "orderProcessingStateMachine.js..." and "start_execution.json", which is currently selected. The main pane displays the contents of "start_execution.json":

```
1  {
2    "orderId": "12345"
3 }
```

Input - optional

Enter input values for this execution in JSON format

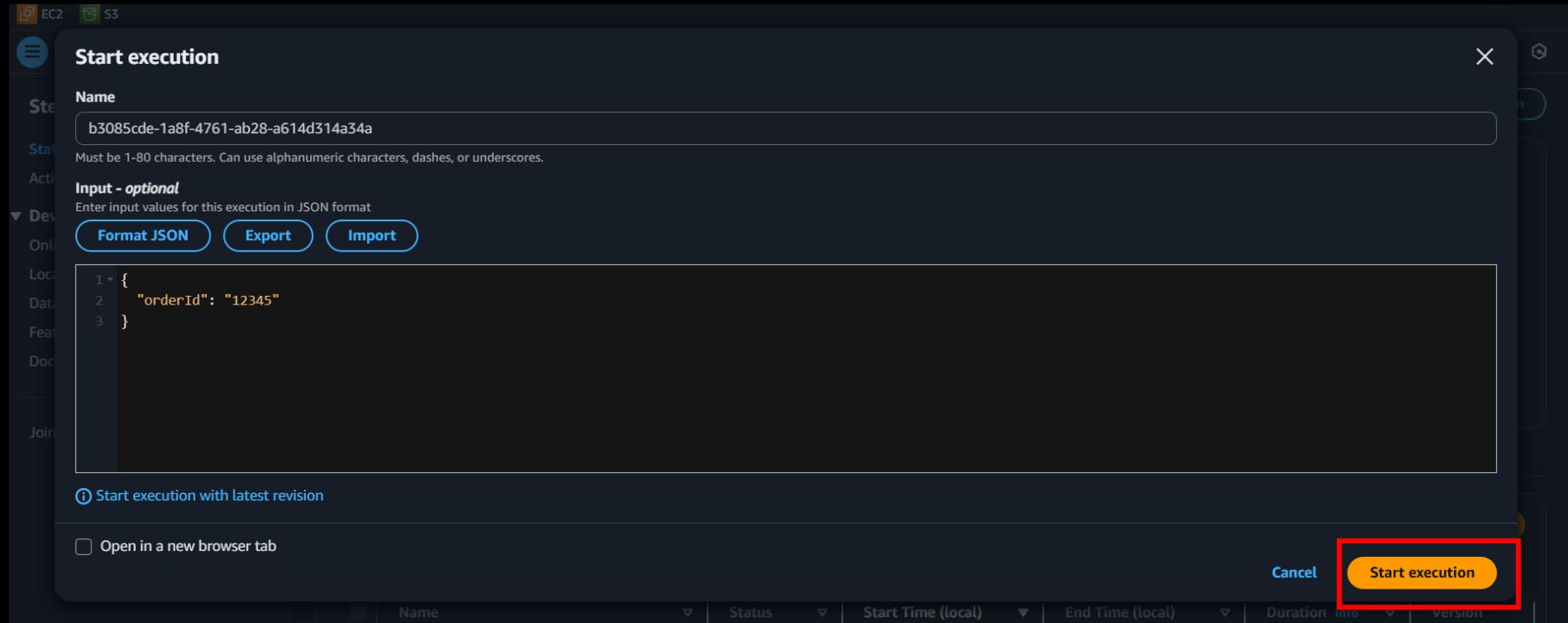
Format JSON

Export

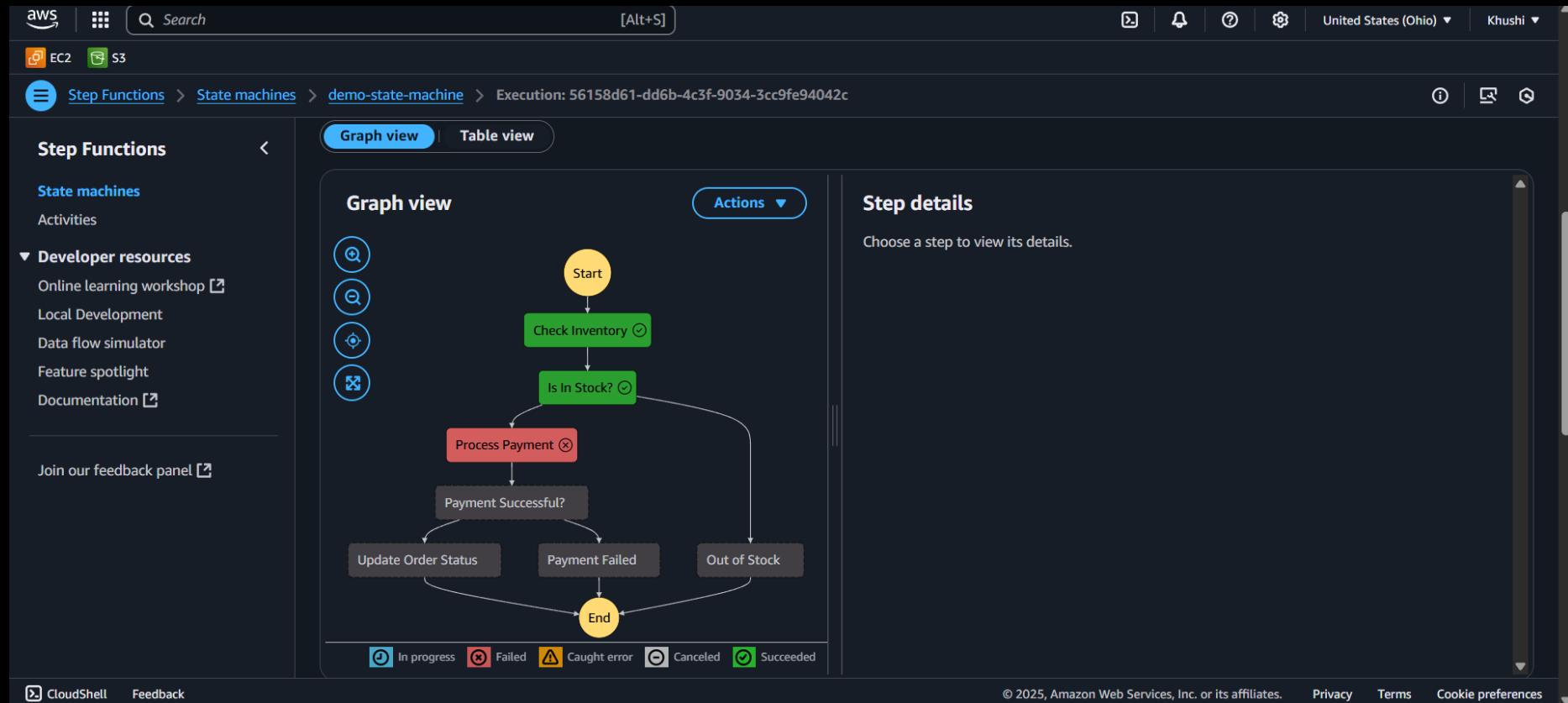
Import

```
1 {
2   "orderId": "12345"
3 }
```

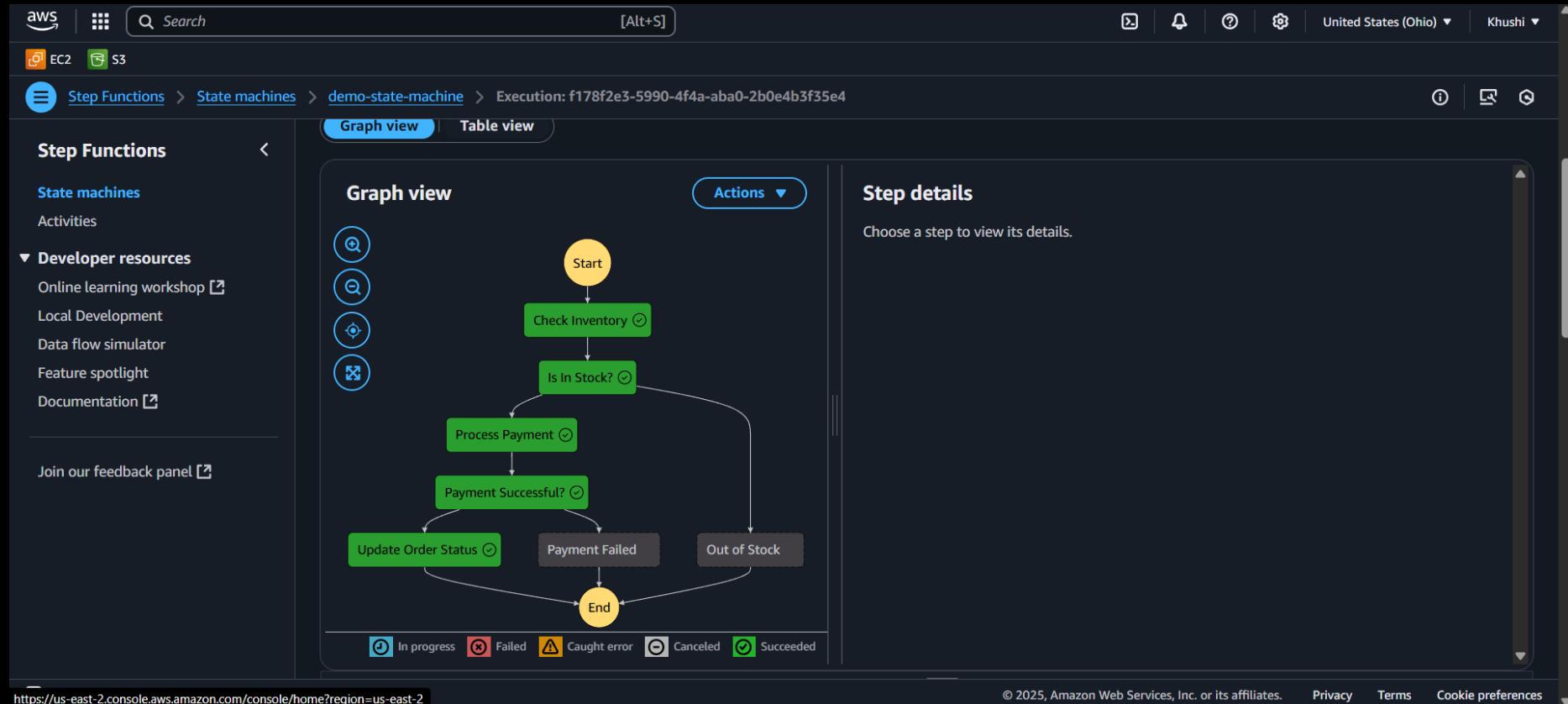
3. Click “Start Execution”



4. Observe execution flow via graph and table views

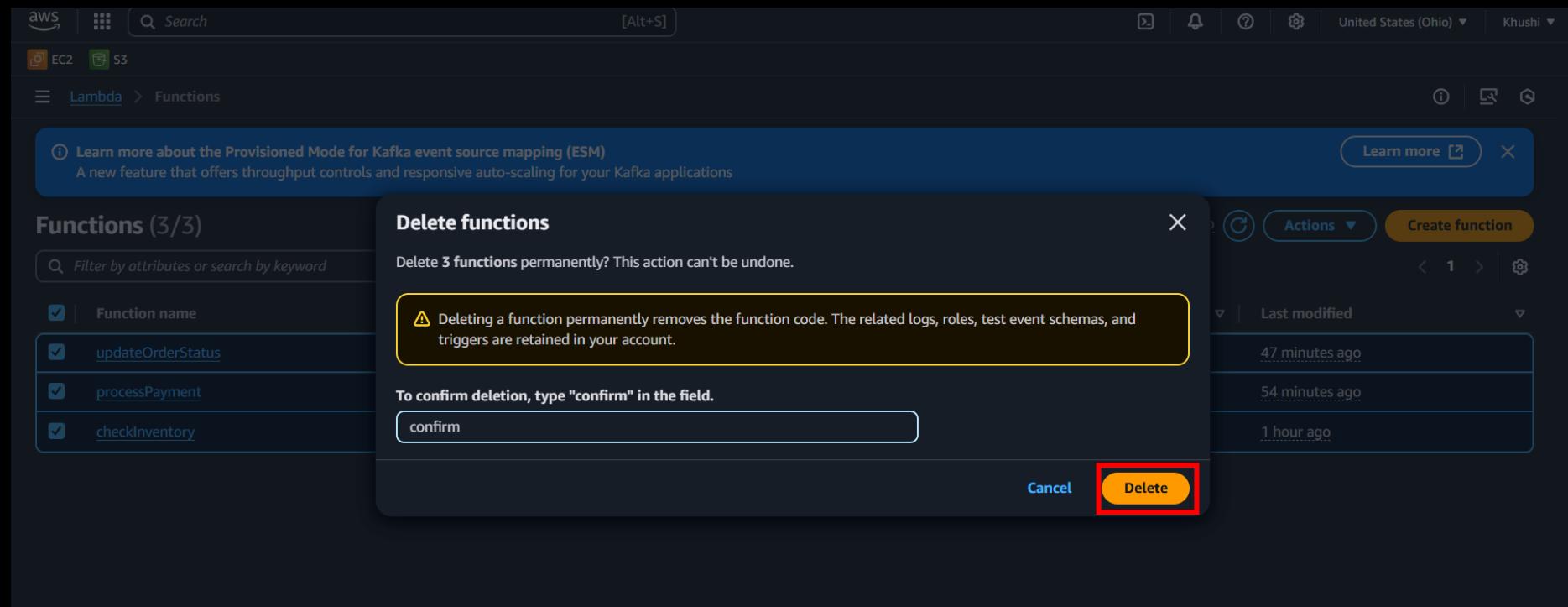


5. You may run additional executions to test different scenarios

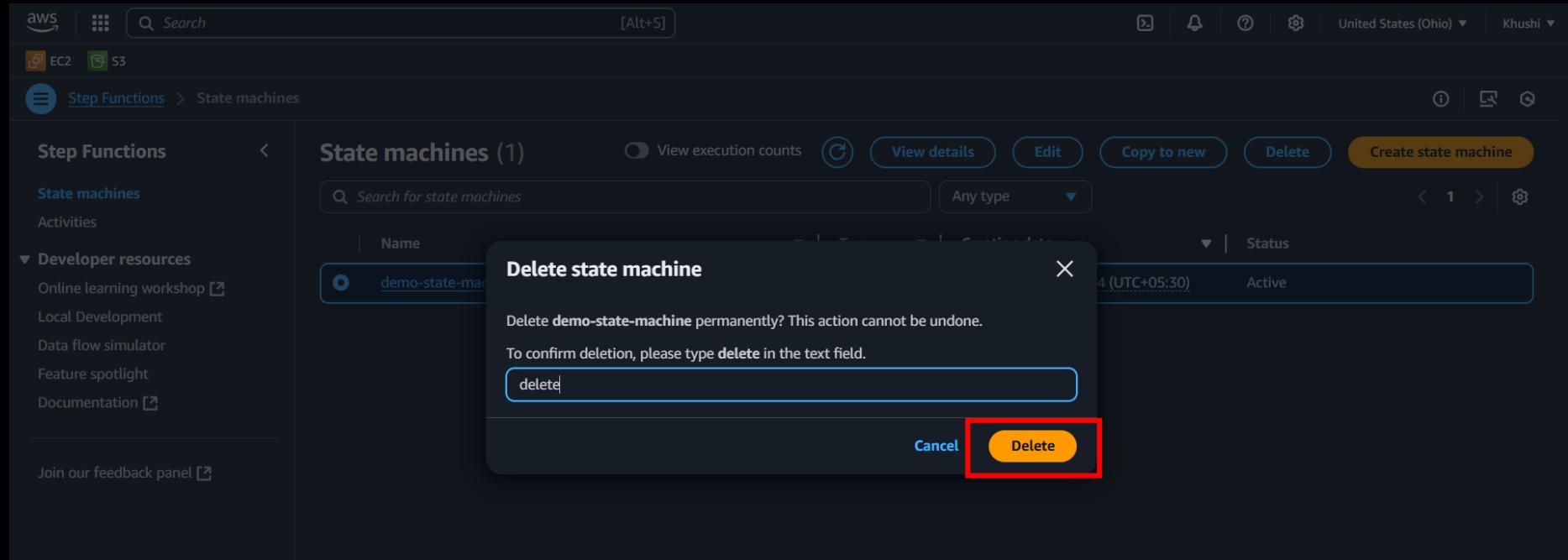


VI. DELETION & CLEANUP 🪢

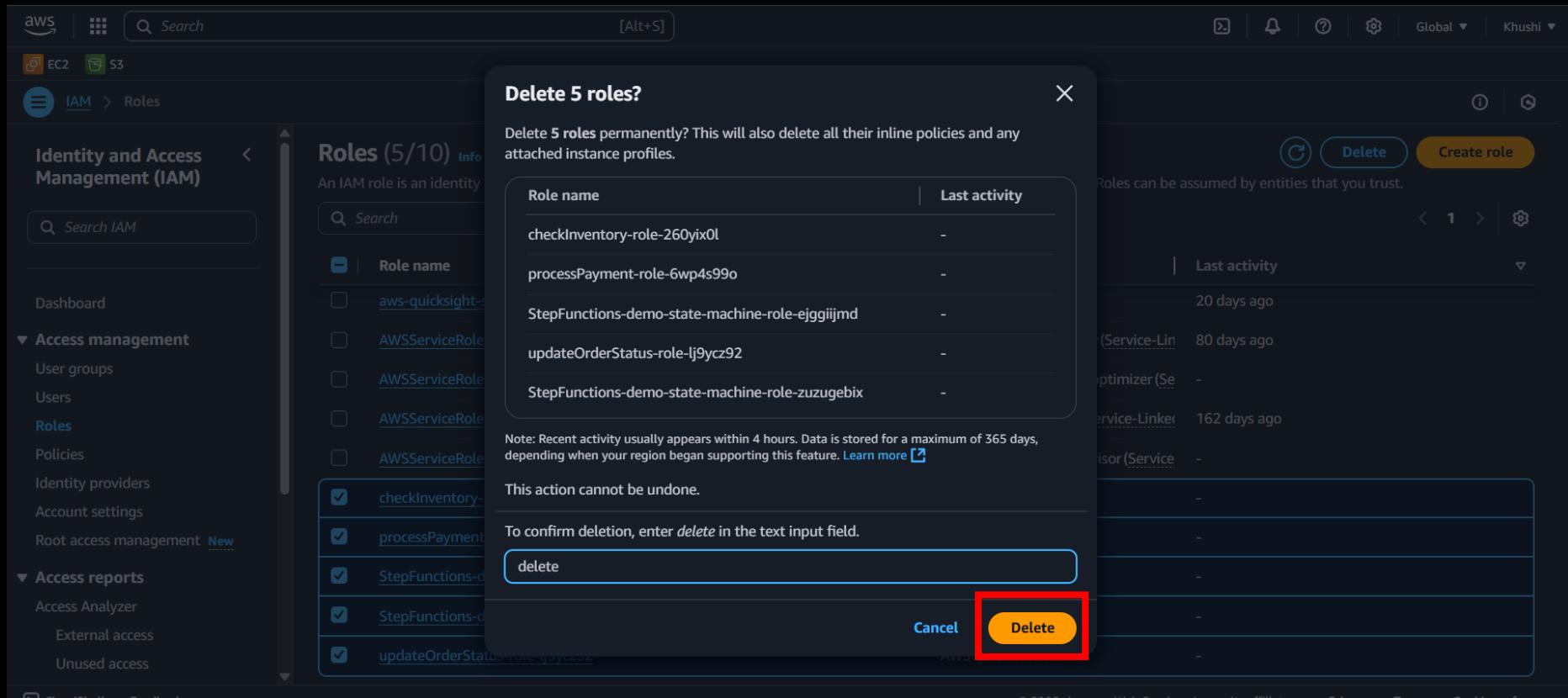
1. Delete all three Lambda functions



2. Delete the state machine from AWS Step Functions



3. Remove IAM roles that were automatically created



❖ CONCLUSION

This project provided a practical walkthrough of designing a serverless order pipeline using AWS Lambda and Step Functions. From defining isolated microservices to integrating them into a cohesive state machine, you now have a blueprint for creating scalable, event-driven architectures. Cleaning up resources post-testing ensures a lean and cost-effective deployment strategy.

“Automate the repetitive, so you can focus on the innovative.”