AWS Console Deployment Guide

Deploy the Step Functions API Gateway integration example entirely through the AWS Management Console.

Prerequisites

- AWS Account with appropriate permissions
- Access to AWS Management Console

Step 1: Create DynamoDB Table

1. Navigate to **DynamoDB Console**
2. Click **Create table**
3. **Table name**: TransactionTable
4. **Partition key**: Id (String)
5. **Provisioned capacity**: Read 1, Write 1
6. Click **Create table**

Step 2: Create Lambda Functions

2.1 Stock Checker Function

1. Go to **Lambda Console** → **Create function**
2. **Function name**: StockCheckerFunction
3. **Runtime**: Node.js 18.x
4. Replace default code with:

```
function getRandomInt(max) {
  return Math.floor(Math.random() * Math.floor(max));
}


exports.lambdaHandler = async (event, context) => {
  try {
    console.log('Event:', JSON.stringify(event, null, 2));
```

```
        const stock_price = getRandomInt(100);

        return {
            statusCode: 200,
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({"stock_price": stock_price})
        };
    } catch (error) {
        console.error('Error:', error);
        return {
            statusCode: 500,
            body: JSON.stringify({ error: 'Internal server error' })
        };
    }
};
```

5. Click **Deploy**

2.2 Stock Buyer Function

1. **Function name**: StockBuyerFunction
2. **Runtime**: Node.js 18.x
3. Replace code with:

```
const crypto = require("crypto");

function getRandomInt(max) {
    return Math.floor(Math.random() * Math.floor(max)) + 1;
}

exports.lambdaHandler = async (event, context) => {
```

```
    try {
        console.log('Event:', JSON.stringify(event, null, 2));

        const body = typeof event.body === 'string' ? JSON.parse(event.body) : event.body;
        const stock_price = body.stock_price;

        const date = new Date();
        const transaction_result = {
            'id': crypto.randomBytes(16).toString("hex"),
            'price': stock_price.toString(),
            'type': "buy",
            'qty': getRandomInt(10).toString(),
            'timestamp': date.toISOString(),
        };

        return {
            statusCode: 200,
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(transaction_result)
        };
    } catch (error) {
        console.error('Error:', error);
        return {
            statusCode: 500,
            body: JSON.stringify({ error: 'Internal server error' })
        };
    }
};
```

4. Click **Deploy**

2.3 Stock Seller Function

1. **Function name**: StockSellerFunction
2. **Runtime**: Node.js 18.x
3. Replace code with:

```
const crypto = require("crypto");

function getRandomInt(max) {
    return Math.floor(Math.random() * Math.floor(max)) + 1;
}

exports.lambdaHandler = async (event, context) => {
    try {
        console.log('Event:', JSON.stringify(event, null, 2));

        const body = typeof event.body === 'string' ? JSON.parse(event.body) : event.body;
        const stock_price = body.stock_price;

        const date = new Date();
        const transaction_result = {
            'id': crypto.randomBytes(16).toString("hex"),
            'price': stock_price.toString(),
            'type': "sell",
            'qty': getRandomInt(10).toString(),
            'timestamp': date.toISOString(),
        };

        return {
            statusCode: 200,
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(transaction_result)
        };
```

```
  } catch (error) {
    console.error('Error:', error);
    return {
      statusCode: 500,
      body: JSON.stringify({ error: 'Internal server error' })
    };
  }
};
```

4. Click **Deploy**

Step 3: Create API Gateway

1. Go to **API Gateway Console**
2. **Create API** → **REST API** → **Build**
3. **API name**: StockTradingAPI
4. Click **Create API**

3.1 Create Resources and Methods

Console Steps: Create /check resource:

- Actions → Create Resource → Resource Name: check
- Actions → Create Method → GET
- Integration type: Lambda Function
- Lambda Function: StockCheckerFunction
- Click **Save**

Create /buy resource:

- Actions → Create Resource → Resource Name: buy
- Actions → Create Method → POST
- Integration type: Lambda Function
- Lambda Function: StockBuyerFunction
- Click **Save**

Create /sell resource:

- Actions → Create Resource → Resource Name: sell
- Actions → Create Method → POST
- Integration type: Lambda Function
- Lambda Function: StockSellerFunction
- Click **Save**

CLI Alternative:

```
# Get API Gateway ID and Root Resource ID
API_ID=$(aws apigateway get-rest-apis --query "items[?name=='StockTradingAPI'].id" --output text)
ROOT_ID=$(aws apigateway get-resources --rest-api-id $API_ID --query "items[?path=='/'].id" --output text)

# Create /check resource
CHECK_RESOURCE_ID=$(aws apigateway create-resource \
  --rest-api-id $API_ID \
  --parent-id $ROOT_ID \
  --path-part check \
  --query 'id' --output text)

# Create GET method for /check
aws apigateway put-method \
  --rest-api-id $API_ID \
  --resource-id $CHECK_RESOURCE_ID \
  --http-method GET \
  --authorization-type NONE

# Integrate /check with Lambda
aws apigateway put-integration \
  --rest-api-id $API_ID \
  --resource-id $CHECK_RESOURCE_ID \
  --http-method GET \
```

```
  --type AWS_PROXY \
  --integration-http-method POST \
  --uri arn:aws:apigateway:$(aws configure get region):lambda:path/2015-03-31/functions/arn:aws:lambda:$(aws configure get region):$(aws sts get-caller-identity --query Account --output text):function:StockCheckerFunction/invocations

# Create /buy resource
BUY_RESOURCE_ID=$(aws apigateway create-resource \
  --rest-api-id $API_ID \
  --parent-id $ROOT_ID \
  --path-part buy \
  --query 'id' --output text)

# Create POST method for /buy
aws apigateway put-method \
  --rest-api-id $API_ID \
  --resource-id $BUY_RESOURCE_ID \
  --http-method POST \
  --authorization-type NONE

# Integrate /buy with Lambda
aws apigateway put-integration \
  --rest-api-id $API_ID \
  --resource-id $BUY_RESOURCE_ID \
  --http-method POST \
  --type AWS_PROXY \
  --integration-http-method POST \
  --uri arn:aws:apigateway:$(aws configure get region):lambda:path/2015-03-31/functions/arn:aws:lambda:$(aws configure get region):$(aws sts get-caller-identity --query Account --output text):function:StockBuyerFunction/invocations

# Create /sell resource
SELL_RESOURCE_ID=$(aws apigateway create-resource \
```

```
  --rest-api-id $API_ID \
  --parent-id $ROOT_ID \
  --path-part sell \
  --query 'id' --output text)

# Create POST method for /sell
aws apigateway put-method \
  --rest-api-id $API_ID \
  --resource-id $SELL_RESOURCE_ID \
  --http-method POST \
  --authorization-type NONE

# Integrate /sell with Lambda
aws apigateway put-integration \
  --rest-api-id $API_ID \
  --resource-id $SELL_RESOURCE_ID \
  --http-method POST \
  --type AWS_PROXY \
  --integration-http-method POST \
  --uri arn:aws:apigateway:$(aws configure get region):lambda:path/2015-03-
31/functions/arn:aws:lambda:$(aws configure get region):$(aws sts get-caller-identity --query Account --
output text):function:StockSellerFunction/invocations

# Add Lambda permissions
aws lambda add-permission \
  --function-name StockCheckerFunction \
  --statement-id apigateway-invoke \
  --action lambda:InvokeFunction \
  --principal apigateway.amazonaws.com \
  --source-arn "arn:aws:execute-api:$(aws configure get region):$(aws sts get-caller-identity --query
Account --output text):$API_ID/*/*"

aws lambda add-permission \
```

```
  --function-name StockBuyerFunction \
  --statement-id apigateway-invoke \
  --action lambda:InvokeFunction \
  --principal apigateway.amazonaws.com \
  --source-arn "arn:aws:execute-api:$(aws configure get region):$(aws sts get-caller-identity --query
Account --output text):$API_ID/*/*"

aws lambda add-permission \
  --function-name StockSellerFunction \
  --statement-id apigateway-invoke \
  --action lambda:InvokeFunction \
  --principal apigateway.amazonaws.com \
  --source-arn "arn:aws:execute-api:$(aws configure get region):$(aws sts get-caller-identity --query
Account --output text):$API_ID/*/*"
```

3.2 Deploy API

Console Steps:

1. Actions → **Deploy API**
2. **Deployment stage**: Prod
3. **Note the Invoke URL** (e.g., https://abc123.execute-api.us-east-1.amazonaws.com/Prod)

CLI Alternative:

```
# Deploy API
aws apigateway create-deployment \
  --rest-api-id $API_ID \
  --stage-name Prod

# Get the API endpoint URL
echo "API Endpoint: https://$API_ID.execute-api.$(aws configure get region).amazonaws.com/Prod"
```

Step 4: Create IAM Role for Step Functions

1. Go to **IAM Console → Roles → Create role**
2. **AWS service → Step Functions**
3. **Role name**: StepFunctionsExecutionRole
4. Attach policies:
    - AWSStepFunctionsFullAccess
    - AmazonDynamoDBFullAccess
    - AmazonAPIGatewayInvokeFullAccess
5. Click **Create role**

Step 5: Create Step Functions State Machine

1. Go to **Step Functions Console**
2. **Create state machine**
3. **Choose authoring method**: Write your workflow in code
4. **Type**: Standard
5. **State machine name**: StockTradingStateMachine
6. Replace definition with:

```
{
  "Comment": "A state machine that does mock stock trading.",
  "StartAt": "Check Stock Value",
  "States": {
    "Check Stock Value": {
      "Type": "Task",
      "Resource": "arn:aws:states:::apigateway:invoke",
      "Parameters": {
        "ApiEndpoint": "YOUR_API_GATEWAY_ID.execute-api.YOUR_REGION.amazonaws.com",
        "Method": "GET",
        "Stage": "Prod",
        "Path": "/check",
        "RequestBody.$": "$",
        "AuthType": "NO_AUTH"
      },
      "Next": "Buy or Sell?"
```

```
    },
    "Buy or Sell?": {
        "Type": "Choice",
        "Choices": [
            {
                "Variable": "$.ResponseBody.stock_price",
                "NumericLessThanEquals": 50,
                "Next": "Buy Stock"
            }
        ],
        "Default": "Sell Stock"
    },
    "Sell Stock": {
        "Type": "Task",
        "Resource": "arn:aws:states:::apigateway:invoke",
        "Parameters": {
            "ApiEndpoint": "YOUR_API_GATEWAY_ID.execute-api.YOUR_REGION.amazonaws.com",
            "Method": "POST",
            "Stage": "Prod",
            "Path": "/sell",
            "RequestBody.$": "$.ResponseBody",
            "AuthType": "NO_AUTH"
        },
        "Next": "Record Transaction"
    },
    "Buy Stock": {
        "Type": "Task",
        "Resource": "arn:aws:states:::apigateway:invoke",
        "Parameters": {
            "ApiEndpoint": "YOUR_API_GATEWAY_ID.execute-api.YOUR_REGION.amazonaws.com",
            "Method": "POST",
            "Stage": "Prod",
            "Path": "/buy",
```

```
            "RequestBody.$": "$.ResponseBody",
            "AuthType": "NO_AUTH"
        },
        "Next": "Record Transaction"
    },
    "Record Transaction": {
        "Type": "Task",
        "Resource": "arn:aws:states:::dynamodb:putItem",
        "Parameters": {
            "TableName": "TransactionTable",
            "Item": {
                "Id": {
                    "S.$": "$.ResponseBody.id"
                },
                "Type": {
                    "S.$": "$.ResponseBody.type"
                },
                "Price": {
                    "N.$": "$.ResponseBody.price"
                },
                "Quantity": {
                    "N.$": "$.ResponseBody.qty"
                },
                "Timestamp": {
                    "S.$": "$.ResponseBody.timestamp"
                }
            }
        },
        "End": true
    }
  }
}
```

7. **Execution role**: Select StepFunctionsExecutionRole
8. Click **Create state machine**

Step 6: Update State Machine Configuration

Important: Replace placeholders in the state machine definition:

- YOUR_API_GATEWAY_ID: Get from API Gateway console (e.g., abc123def)
- YOUR_REGION: Your AWS region (e.g., us-east-1)

Step 7: Test the Solution

1. In **Step Functions Console**, select your state machine
2. Click **Start execution**
3. Input: {}
4. Click **Start execution**
5. Monitor execution flow
6. Check **DynamoDB** table for transaction records

Architecture Overview

- **Lambda Functions**: Process stock operations
- **API Gateway**: REST endpoints for stock operations
- **Step Functions**: Orchestrates the workflow
- **DynamoDB**: Stores transaction results

Cleanup

To delete resources:

1. Delete Step Functions state machine
2. Delete API Gateway
3. Delete Lambda functions
4. Delete DynamoDB table
5. Delete IAM role