



Serverless x GenAI BKK Workshop

Generative AI at scale: Serverless workflows for enterprise-ready apps

[Pre-requisite] Enable foundation model access in Amazon Bedrock

[Pre-requisite] Configuring the front-end application

► Playground

▼ Use cases

▼ Building a RAG pipeline

Testing the need for RAG

Building the RAG Pipeline

Running the Pipeline

Testing the RAG Inference

High level Code Walkthrough

Summary

► Document extraction and summarization

► Workshop Cleanup

▼ AWS account access

[Open AWS console \(us-west-2\)](#)

[Get AWS CLI credentials](#)

Exit event

[Event dashboard](#) > [Use cases](#) > [Building a RAG pipeline](#) > Building the RAG Pipeline

Building the RAG Pipeline

In this section, you'll create a workflow in Step Functions Workflow Studio to process the input documents required for RAG interactions. To build it quick, we have placed 4 documents in the S3 bucket with the name containing `ragpipelinebucket`.

In the workflow

- You will first identify the format of the document using a Lambda function.
- You will then use choice state to choose the right Lambda function to process the different file format.
- After the file is processed, you will move the processed file to another location.
- You will repeat the above steps for every single file in the S3 bucket using Step Functions distributed map

Why distributed map?

RAG process is generally done for a collection of objects. The collection can be 100s of 1000s of files. Distributed map is a purpose built iterator that can iterate millions of items in a collection at an unparallel concurrency, run a business logic on these items and support configurations such as batching, concurrency and failure tolerance. If you want to learn more about distributed map, try this [workshop](#) later.

Creating the Workflow

1. Navigate to [AWS Step Functions](#) in your AWS console. Make sure you are in the correct region.
2. If you are not on the State machines page, choose State machines on the left side hamburger menu icon and then select **Create state machine**
3. On the Choose a template overlay, choose the **Blank** template, and select **Select**.
4. Choose **JSONPath** as query language on the right side of the workflow configuration.

Workflow Definition >

The top level state machine properties for this workflow. [Learn more](#)

State machine query language

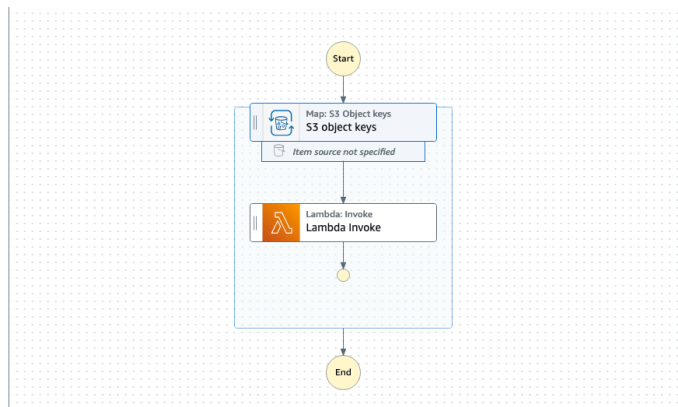
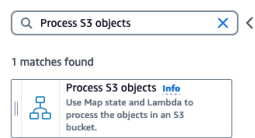
☐ JSONata - recommended
All states and fields will require valid JSONata expressions for queries and data transformations.

☒ JSONPath
New states will default to JSONPath. You can convert to JSONata on a state-by-state basis.

Comment - optional
A human-readable description of the state machine.

A description of my state machine

Create the distributed map to iterate on all files



2. Configure the Distributed Map state with the following values:

Setting	Value	Comments
State name	RAG Pipeline Ingestion	
Processing mode	Distributed	
Item source	Amazon S3	
S3 item source	S3 object list	
S3 bucket	Choose Get bucket and prefix at runtime from state input	
Bucket Name	\$.input_bucket	
Prefix	\$.prefix	
Additional Configurations	Expand	
Modify items with ItemSelector - optional	{ "sourcebucket.\$": "\$.input_bucket", "Key.\$": "\$\$.Map.Item.Value.Key" }	
Set concurrency limit	1	Because of workshop restrictions, you will run the process in serial though Distributed map can run at a concurrency of 10K!
Child execution type	Express	In general, if an iteration can complete in 5 minutes, prefer Express workflow to save on cost

Configuring steps to create the vector embeddings

1. Select the **Lambda Invoke** state within the Distributed Map state. Configure the state with the following values.

Setting	Value
State name	File Validations
Function name	rag-pipeline-ingestion-file-validation:\$LATEST

► Expand to see the workflow

2. In the States browser on the left, search for **Choice** and drag the **choice** flow onto the canvas within the Distributed Map under the existing Lambda state. Configure the state with the

following values:

- Enter the Choice **State name** as `Is Media File?`
- Edit the **Rule 1** under the Choice Rules tab select **Add Conditions**.
- Enter conditions as shown below and click **Save conditions**.

Conditions for rule #1

Choice rules contain conditional statements, which are used to evaluate one or more node values (called variables) in your state's JSON input. [Learn more](#)

Simple
Evaluates a single conditional statement.

Not

Variable

Operator

Value

`$.ismediafile`

is equal to

Boolean constant

true

Must use JsonPath.

- Don't change anything on the **Default Rule**

► Expand to see the workflow

Ensure that the Choice state is nested within the Map state to ensure that the decision logic is applied to each individual item being processed in the map iteration, rather than to the entire collection at once.

► Incorrect Placement

- Search for **AWS Lambda** and drag the *Invoke* state onto the canvas within the Distributed Map under the choice true condition. Configure the state with the following values:

Setting	Value	comment
State name	Split and Embedding - Audio Files	
Function name	rag-pipeline-ingestion-process-audio-files:\$LATEST	This Lambda function creates the vector embeddings for the audio files and stores in OpenSearch

► Expand to see the workflow

- Search for **AWS Lambda** and drag the *Invoke* state onto the canvas within the Distributed Map under the Default rule condition. Configure the state with the following values:

Setting	Value	Comment
State name	Split and Embedding - Other Files	
Function name	rag-pipeline-ingestion-process-files:\$LATEST	This Lambda function creates the vector embeddings for non audio files and stores in OpenSearch

► Expand to see the workflow

Configuring steps to move the processed file to another bucket

- Search for **S3 CopyObject** and drag onto the canvas inside Distributed Map under **Split and Embedding - Other Files** Lambda

Setting	Value
State name	Copy Processed Files To Archive Location

Add the below details under **API Parameter**.



```

1 {
2   "Bucket.$": "$.archive.archivebucket",
3   "CopySource.$": "$.processedfiles",
4   "Key.$": "$.archive.filename"
5 }

```

- a. Click on **Input/Output** tab, Select **Add original input to output using ResultPath** option. Make sure 1st dropdown has **Combine original input with result** and in value enter \$.output

☒ **Add original input to output using ResultPath - optional**
 By default, a state sends its task result as output. With a ResultPath, you can pass the original input combined with Task results. [Info](#)

Combine original input with result ▼

\$.output

Must use valid JSONPath syntax.

2. Search for S3 DeleteObject and drag onto the canvas below the copy object.

Setting	Value
State name	Delete Processed Files From Source Location

Add the below details under **API Parameter**

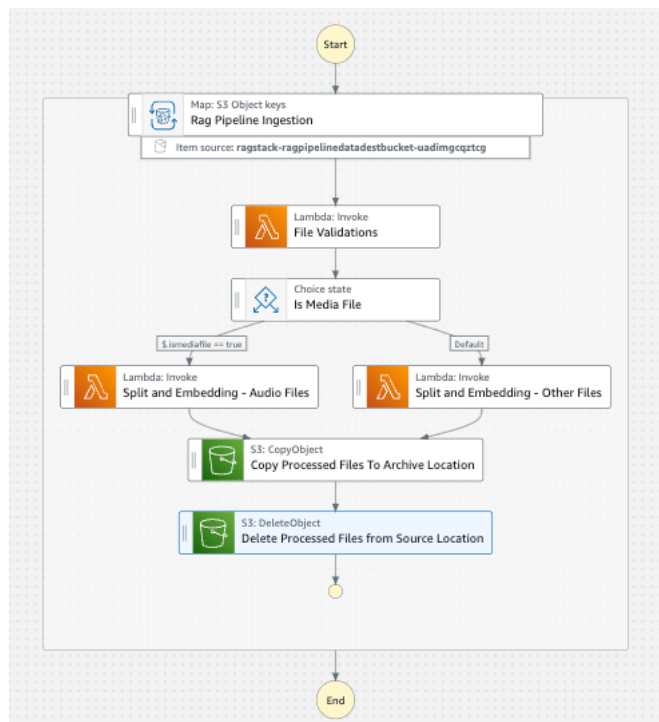
```

1 {
2   "Bucket.$": "$.archive.sourcebucket",
3   "Key.$": "$.archive.filename"
4 }

```



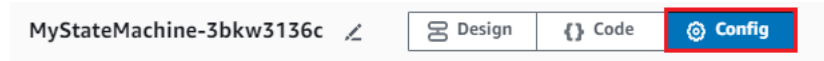
3. Select the **Split and Embedding - Audio Files** Lambda invoke, Go to **Next State** under configuration. From the dropdown select the **Copy Processed Files To Archive Location** step.
4. Verify if your workflow looks like the one below





- If you run into issues, expand here to get the code to create the workflow. Make sure to replace {ACCOUNT} with correct account number

Creating the workflow

1. Select the **Config** tab next to the state machine name at the top of the page



2. Edit the **State machine name**: RagPipelineStateMachine 
3. Make sure to select standard workflow.
4. For the **Execution role**, choose an existing role containing RAGPipelineIngestionProcessing  in its name
5. Select All in the Log level for **Logging**
6. Select **Create**

Congrats! You have successfully built a workflow for RAG pipeline. In the next section, you will run the RAG pipeline.

[Previous](#)[Next](#)