

Functions in AWS Lambda

Date published: 2021-04-06

Date modified: 2024-06-03

CLOUdera

Legal Notice

© Cloudera Inc. 2024. All rights reserved.

The documentation is and contains Cloudera proprietary information protected by copyright and other intellectual property rights. No license under copyright or any other intellectual property right is granted herein.

Unless otherwise noted, scripts and sample code are licensed under the Apache License, Version 2.0.

Copyright information for Cloudera software may be found within the documentation accompanying each component in a particular release.

Cloudera software includes software from various open source or other third party projects, and may be released under the Apache Software License 2.0 (“ASLv2”), the Affero General Public License version 3 (AGPLv3), or other license terms. Other software included may be released under the terms of alternative open source licenses. Please review the license and notice files accompanying the software for additional licensing information.

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Cloudera, Cloudera Altus, HUE, Impala, Cloudera Impala, and other Cloudera marks are registered or unregistered trademarks in the United States and other countries. All other trademarks are the property of their respective owners.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Creating a Lambda function.....	5
Configuring your Lambda function.....	7
Output ports.....	12
Parameters.....	13
Cold start.....	15
Cloud storage.....	16
Configuring VPC to deploy Lambda Functions.....	18
File system for content repository.....	20
Output size constraints.....	21
DataFlow state.....	21
Configuring Kerberos.....	21
Handling failures.....	22
Testing your Lambda function.....	22
Publishing your Lambda function.....	23
Monitoring and logs.....	24
Adjusting logs levels.....	25

AWS Lambda triggers.....	26
Creating a Lambda function using CLI.....	36

Creating a Lambda function

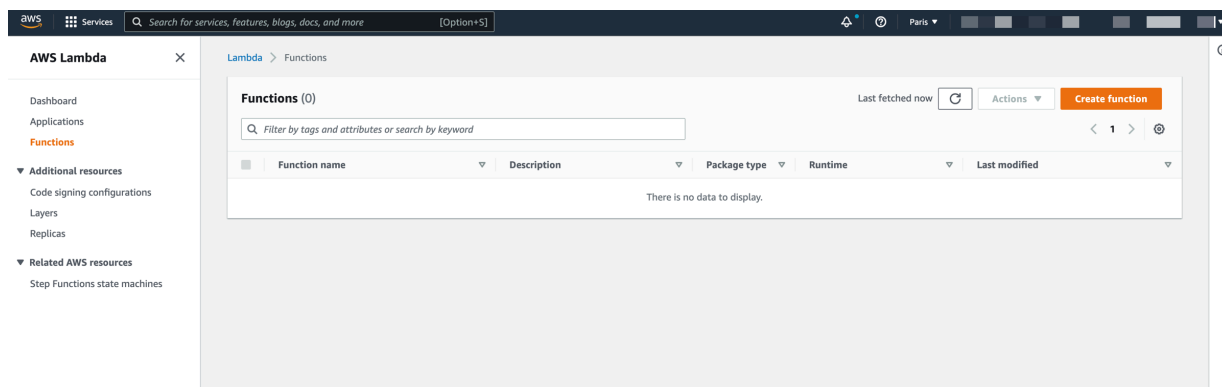
AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers.

About this task

Follow these steps to create an AWS Lambda function that is able to run :

Procedure

1. Navigate to the AWS Lambda page in your cloud account in the region of your choice.



2. Click Create function.

The Create function page opens.

3. Fill in the Basic information section:
 - a) Keep the default Author from scratch option selected.
 - b) Give your function a meaningful name for your use case.
 - c) Select Java 8 on Amazon Linux 2 for Runtime.
 - d) Choose x86_64 for Architecture.
 - e) Click Create function.

The screenshot shows the AWS Lambda 'Create function' page. The breadcrumb navigation is 'Lambda > Functions > Create function'. The main heading is 'Create function' with an 'info' link. Below it, a sub-heading says 'Choose one of the following options to create your function.' There are four selection boxes: 'Author from scratch' (selected with a blue dot), 'Use a blueprint', 'Container image', and 'Browse serverless app repository'. The 'Basic information' section contains: 'Function name' with the value 'MyFirstFunction'; 'Runtime' set to 'Java 8 on Amazon Linux 2'; and 'Architecture' set to 'x86_64'. Below this is a 'Permissions' section with a link to 'Change default execution role'. At the bottom right of the form are 'Cancel' and 'Create function' buttons.

Your function is created and its details are displayed.

4. In the Code source section of the Code tab, click Upload from Amazon S3 location .
The Upload a file from Amazon S3 modal window appears.
5. Provide the S3 URL of Function binaries in the textbox.



Note: You have downloaded the Function handler libraries as a zip of binaries from and uploaded them to an S3 bucket when getting ready to run functions. AWS Lambda will use these binaries to run the NiFi flow. For instructions, see *Downloading Lambda Function binaries and uploading to S3*.

The screenshot shows a modal window titled 'Upload a file from Amazon S3'. It has a close button (X) in the top right. Inside, there is a blue information box that says 'When you upload a new .zip file package, it overwrites the existing code.' Below this is a section for 'Amazon S3 link URL' with the instruction 'Paste an S3 link URL to your function code .zip.' A text input field contains the URL 'unctionbucket.s3.eu-west-3.amazonaws.com/naaf-aws-lambda-1.0.0-SNAPSHOT-bin.zip'. At the bottom right are 'Cancel' and 'Save' buttons.

6. Click Save.

7. Navigate to the Runtime Settings on the Code tab and click Edit to change the Handler property to: `com.cloudera.naaf.aws.lambda.StatelessNiFiFunctionHandler::handleRequest`

The details of your first function:

The screenshot shows the AWS Lambda console for a function named 'MyFirstFunction'. The 'Function overview' tab is active, displaying a card for the function with a 'Layers' section showing 0 layers. To the right, a sidebar shows metadata: Description, Last modified (3 minutes ago), Function ARN (arn:aws:lambda:eu-west-3:381358652250:function:MyFirstFunction), and Function URL. Below the overview, the 'Code' tab is selected, showing a message that the code editor does not support Java 8 on Amazon Linux 2. The 'Code properties' section lists package size (1.5 kB), SHA256 hash, and last modified date. The 'Runtime settings' section shows the runtime as 'Java 8 on Amazon Linux 2' and the handler as 'com.cloudera.naaf.aws.lambda.StatelessNiFiFunctionHandler::handleRequest'.

What to do next

You can now start configuring your Lambda specifically for your flow and for your use case.

Related Information

[Downloading Lambda Function binaries and uploading to S3](#)

Configuring your Lambda function

After you create a function, you can use the built-in configuration options to control its behavior. You can configure additional capabilities, adjust resources associated with your function, such as memory and timeout, or you can also create and edit test events to test your function using the console.

Memory and timeout in runtime configuration

You can configure two very important elements, memory and timeout on the Configuration General configuration tab of the Lambda function details screen.

The screenshot shows the AWS Lambda console interface for a function named 'MyFirstFunction'. The top navigation bar includes 'Lambda > Functions > MyFirstFunction'. The main header area displays the function name, a 'Throttle' button, a 'Copy ARN' button, and an 'Actions' dropdown menu. Below this is the 'Function overview' section, which includes a card for 'MyFirstFunction' showing 'Layers (0)', an 'Add trigger' button, and an 'Add destination' button. To the right of the card, details are listed: 'Description -', 'Last modified 11 seconds ago', and 'Function ARN arn:aws:lambda:eu-west-2:381358652250:function:MyFirstFunction'. A tabbed interface below the overview shows 'Code', 'Test', 'Monitor', 'Configuration' (selected), 'Aliases', and 'Versions'. The 'Configuration' tab is active, showing a 'General configuration' section with an 'Edit' button. The configuration table lists: 'Description -', 'Memory (MB) 1024', and 'Timeout 1 min 0 sec'. A left-hand sidebar contains a menu with 'General configuration', 'Triggers', 'Permissions', 'Destinations', 'Environment variables', 'Tags', 'VPC', and 'Monitoring and operations'.

The appropriate values for these two settings depend on the data flow to be deployed. Typically, it is recommended to start with 1536-2048 MB (1.5 - 2 GB) memory allocated to Lambda. This amount of memory may be more than necessary or less than required. The default value of 512 MB may be perfectly fine for many use cases. You can adjust this value later as you see how much memory your function needs.

While tend to perform very well during a warm start, a cold start that must source extensions (processors, controller services, and so on) may take several seconds to initialize. So it is recommended to set the Timeout to at least 30 seconds. If the data flow to run reaches out to many other services or performs complex computation, it may be necessary to use a larger value, even several minutes.

Click Edit if you want to change the default values.

See the corresponding [AWS documentation](#) to learn more about timeout duration and memory sizing, which determines the amount of resources provisioned for your function.

Runtime environment variables

You must configure the function to specify which data flow to run and add any necessary runtime configuration using environment variables.

Procedure

1. Navigate to the Configuration tab and select Environment variables from the menu on the left.
2. Click Edit on the Environment variables pane.

3. Define your environment variables as key-value pairs.

You can add one or more environment variables to configure the function. The following environment variables are supported:

Variable Name	Description	Required	Default Value
FLOW_CRN	<p>The Resource Name (CRN) for the data flow that is to be run.</p> <p>The data flow must be stored in the Catalog. This CRN should indicate the specific version of the data flow and as such will end with a suffix like /v.1.</p> <p>For more information, see <i>Retrieving data flow CRN</i>.</p>	true	--
DF_PRIVATE_KEY	<p>The Private Key for accessing the service.</p> <p>The Private Key and Access Key are used to authenticate with the Service and they must provide the necessary authorizations to access the specified data flow.</p> <p>For more information, see <i>Provisioning Access Key ID and Private Key</i>.</p>	true	--
DF_ACCESS_KEY	<p>The Access Key for accessing the service.</p> <p>The Private Key and Access Key are used to authenticate with the Service and they must provide the necessary authorizations to access the specified data flow.</p> <p>For more information, see <i>Provisioning Access Key ID and Private Key</i>.</p>	true	--
INPUT_PORT	<p>The name of the Input Port to use.</p> <p>If the specified data flow has more than one Input Port at the root group level, this environment variable must be specified, indicating the name of the Input Port to queue up the AWS Lambda notification. If there is only one Input Port, this variable is not required. If it is specified, it must properly match the name of the Input Port.</p>	false	--

Variable Name	Description	Required	Default Value
OUTPUT_PORT	<p>The name of the Output Port to retrieve the results from.</p> <p>If no Output Port exists, the variable does not need to be specified and no data will be returned. If at least one Output Port exists in the data flow, this variable can be used to determine the name of the Output Port whose data will be sent along as the output of the function.</p> <p>For more information on how the appropriate Output Port is determined, see <i>Output ports</i>.</p>	false	--
FAILURE_PORTS	<p>A comma-separated list of Output Ports that exist at the root group level of the data flow. If any FlowFile is sent to one of these Output Ports, the function invocation is considered a failure.</p> <p>For more information, see <i>Output ports</i>.</p>	false	--
DEFAULT_PARAM_CONTEXT	<p>If the data flow uses a Parameter Context, the Lambda function will look for a Secret in the AWS Secrets Manager with the same name. If no Secret can be found with that name, this variable specifies the name of the Secret to default to.</p> <p>For more information, see <i>Parameters</i>.</p>	false	--
PARAM_CONTEXT_*	<p>If the data flow makes use of Parameter Contexts, this environment variable provides a mechanism for mapping a Parameter Context to one or more alternate Secrets in the AWS Secrets Manager, in a comma-separated list.</p> <p>For more information, see <i>Parameters</i>.</p>	false	--
CONTENT_REPO	<p>The contents of the FlowFiles can be stored either in memory, on the JVM heap, or on disk. If this environment variable is set, it specifies the path to a directory where the content should be stored. If it is not specified, the content is held in memory.</p> <p>For more information, see <i>File system for content repository</i>.</p>	false	--

Variable Name	Description	Required	Default Value
EXTENSIONS_*	The directory to look for custom extensions / NiFi Archives (NARs). For more information, see <i>Providing custom extensions / NARs</i> .	false	--
DF_SERVICE_URL	The Base URL for the Service.	false	https://api.us-west-1.cdp.cloudera.com/
NEXUS_URL	The Base URL for a Nexus Repository for downloading any NiFi Archives (NARs) needed for running the data flow.	false	https://repository.cloudera.com/artifactory/cloudera-repos/
STORAGE_BUCKET	An S3 bucket in which to look for custom extensions / NiFi Archives (NARs) and resources. For more information, see <i>S3 Bucket storage</i> .	false	--
STORAGE_EXTENSIONS_DIRECTORY	The directory in the S3 bucket to look for custom extensions / NiFi Archives (NARs). For more information, see <i>S3 Bucket storage</i> .	false	extensions
STORAGE_RESOURCES_DIRECTORY	The directory in the S3 bucket to look for custom resources. For more information, see <i>Providing additional resources</i> .	false	resources
WORKING_DIR	The working directory, where NAR files will be expanded.	false	/tmp/working
EXTENSIONS_DOWNLOAD_DIR	The directory to which missing extensions / NiFi Archives (NARs) will be downloaded. For more information, see <i>Providing custom extensions / NARs</i> .	false	/tmp/extensions
EXTENSIONS_DIR_*	It specifies read-only directories that may contain custom extensions / NiFi Archives (NARs). For more information, see <i>Providing custom extensions / NARs</i> .	false	--
DISABLE_STATE_PROVIDER	If true, it disables the DynamoDB data flow state provider, even if the data flow has stateful processors.	false	--
DYNAMODB_STATE_TABLE	The DynamoDB table name where the state will be stored	false	nifi_state
DYNAMODB_BILLING_MODE	It sets the DynamoDB Billing Mode (PROVISIONED or PAY_PER_REQUEST).	false	PAY_PER_REQUEST

Variable Name	Description	Required	Default Value
DYNAMODB_WRITE_CAPACITY_UNITS	It sets the DynamoDB Write Capacity Units, when using PROVISIONED Billing Mode.	true if using PROVISIONED Billing Mode	--
DYNAMODB_READ_CAPACITY_UNITS	It sets the DynamoDB Read Capacity Units, when using PROVISIONED Billing Mode.	true if using PROVISIONED Billing Mode	--
KRB5_FILE	It specifies the filename of the krb5.conf file. This is necessary only if connecting to a Kerberos-protected endpoint. For more information, see <i>Configuring Kerberos</i> .	false	/etc/krb5.conf

4. When ready, click Save.

Related Information

[Retrieving data flow CRN](#)

[Provisioning Access Key ID and Private Key](#)

[Output ports](#)

[Parameters](#)

[File system for content repository](#)

[Custom NARs in AWS](#)

[S3 Bucket storage](#)

[Configuring Kerberos](#)

Output ports

Depending on what kind of data flow you want to run, you may or may not want to output the data.

For example, a data flow may be designed to be called in response to an object being added to S3. This data flow may fetch the data from S3, parse the data, transform it, and then push the data into Apache Kafka. For such a data flow, it does not really make sense to output any data, as the data flow itself introduces a Kafka topic as the sink of the data.

You can design a different data flow to be called in response to a message being placed onto an SNS topic. The data flow may take that notification, filter out messages that do not match some criteria, and then send any messages that do match the criteria to an Output Port. That output may then be connected to another destination through an AWS Lambda destination.

Output ports can also be used when using the API Gateway trigger to output the data that should be sent back to the client that made the API call.

No output port

If no Output Port is present in the data flow's root group, no output will be provided from the function invocation.

One output port

If one Output Port is present, and its name is "failure", any data routed to that Output Port will cause the function invocation to fail. No output will be provided as the output of the function invocation.

Two output ports, where one is failure

If two Output Ports are present, and one of them is named "failure", that Output Port will behave as the "failure" port described above. The other Output Port is considered a successful invocation. When the data flow is invoked, if a single FlowFile is routed to this Port, the contents of the

FlowFile will be emitted as the output of the function. If more than one FlowFile is sent to the Output Port, the data flow will be considered a failure, as Lambda requires a single entity to be provided as its output. In this case a MergeContent or MergeRecord Processor may be used to assemble multiple FlowFiles into a single output FlowFile.

Any port with the name "failure" (ignoring case) will be considered a Failure Port. Additionally, if the *FAILURE_PORTS* environment variable is specified, any port whose name is in the comma-separated list will be considered a Failure Port.

Two or more output ports with no failure port

If two or more Output Ports are present and none of them are failure ports, you must provide the *OUTPUT_PORT* environment variable. This environment variable must match the name of an Output Port at the data flow's root group (case is NOT ignored). Anything that is routed to the specified port is considered the output of the function invocation. Anything routed to any other port is considered a failure.

Parameters

An important step in building a data flow that you can run outside of the NiFi instance where it was built is the concept of parameterization. NiFi allows you to define Processor and Controller Service properties at runtime instead of at build time by using Parameter Contexts. The Lambda function handler allows you to specify the parameters in two ways: using environment variables or using the AWS Secrets Manager.

Environment variables

Any parameter can be specified using the environment variables of the AWS Lambda function. When configuring the Lambda function, simply add an environment variable whose name matches the name of a parameter in your Parameter Context.



Note: AWS has reserved environment variables that you cannot use for your parameters (examples: *AWS_ACCESS_KEY*, *AWS_SECRET_ACCESS_KEY*, and so on). See the full list [here](#).

AWS Secrets Manager

A more secure mechanism for storing parameters is to use the AWS Secrets Manager.

Procedure

1. Open the AWS Secrets Manager and click Store a new secret.
2. For secret type, select Other type of secret.
3. Provide your secret information, such as credentials and connection details, as key/value pairs.

Enter one or more key/value pairs to use as parameters. Each key provided to the Secret Manager maps to a parameter in the data flow's Parameter Context with the same name.

If the name of the secret matches the name of the Parameter Context in the data flow, there is no further configuration needed. The Lambda function will automatically configure the data flow to pull the specified secret's values to represent the Parameter Context.



Note:

Keep in mind that there are cases in which this does not work.

4. When you are ready, click Next.
5. Provide a name for your secret.
6. You can configure other options like tags, resource permissions, rotation schedule and others.

7. On the Review page, review your secret details, and click Store.

Example:

You developed a data flow with a Parameter Context named `AWS_CONTEXT`, and you want to deploy this data flow three times. The first time, it should use the values in the "Context_1" secret. For the second deployment, it should use values for the "Context_2" secret. For the third deployment, it should use values from the "Context_3" secret.

To accomplish this, you can specify an environment variable named `PARAM_CONTEXT_AWS_CONTEXT`. In this case, the name of the environment variable is `PARAM_CONTEXT_` followed by the name of the Parameter Context. The value of the environment variable will be the name of the secret in the AWS Secrets Manager.

For the first deployment you would use the environment variable `PARAM_CONTEXT_AWS_CONTEXT` with a value of "Context_1". For the second deployment, you would use an environment variable named `PARAM_CONTEXT_AWS_CONTEXT` with a value of "Context_2" and so on.

If your data flow contains multiple Parameter Contexts, you can also map each of them to different secrets. For example, if you have a Parameter Context named `AWS_CONTEXT` and another one named `CLOUDERA_CONTEXT`, and you wanted to map those to secrets named "Context_1" and "Cldr_Context" respectively, you could do so by adding two environment variables: `PARAM_CONTEXT_AWS_CONTEXT = "Context_1"` and `PARAM_CONTEXT_CLOUDERA_CONTEXT = "Cldr_Context"`.

Additionally, you can simply default all Parameter Contexts whose names do not map any secrets in AWS Secrets Manager to use a default secret by setting an environment variable named `DEFAULT_PARAM_CONTEXT`. The value of this environment variable should be the name of the secret to use.



Important: To use this capability, the Parameter Contexts used in the flow must have names that are compatible with AWS Secrets Manager names and Environment Variable names. To ensure that these naming conventions are followed, Parameter Contexts and Parameters should use names that start with a letter and only consist of letters, numbers, and the underscore (`_`) character. For example, instead of naming a Parameter Context `AWS Parameter Context`, use a name of `AWS_PARAMETER_CONTEXT` when building the data flow.



Note: The role that is used to launch the Lambda function must be assigned to the right policies to list secrets in the AWS Secrets Manager, as well as read the secrets.

You can also specify that multiple Secrets map to the same Parameter Context by providing a comma-separated list of Secret names. For example, if you want both the "Kafka" and "Common" Secrets to contribute parameters to a Parameter Context named "ALL_KAFKA", you would set `PARAM_CONTEXT_ALL_KAFKA = "Kafka, Common"`.



Note: Duplicate keys are not supported and will be non-deterministic. Therefore, it is important to ensure that any Secrets mapped to the same Parameter Context do not share identical key names. Otherwise, you are not able to tell which Secret's key is used for the parameter value.

The same Secret may also provide parameters to multiple different Parameter Contexts. Therefore, a configuration similar to the following is possible:

```
PARAM_CONTEXT_KAFKA = "Kafka, Common"
PARAM_CONTEXT_SOLR = "Solr, Common"
```

In this scenario, the Common Secret supplies values to both the KAFKA and SOLR Parameter Contexts.

Example

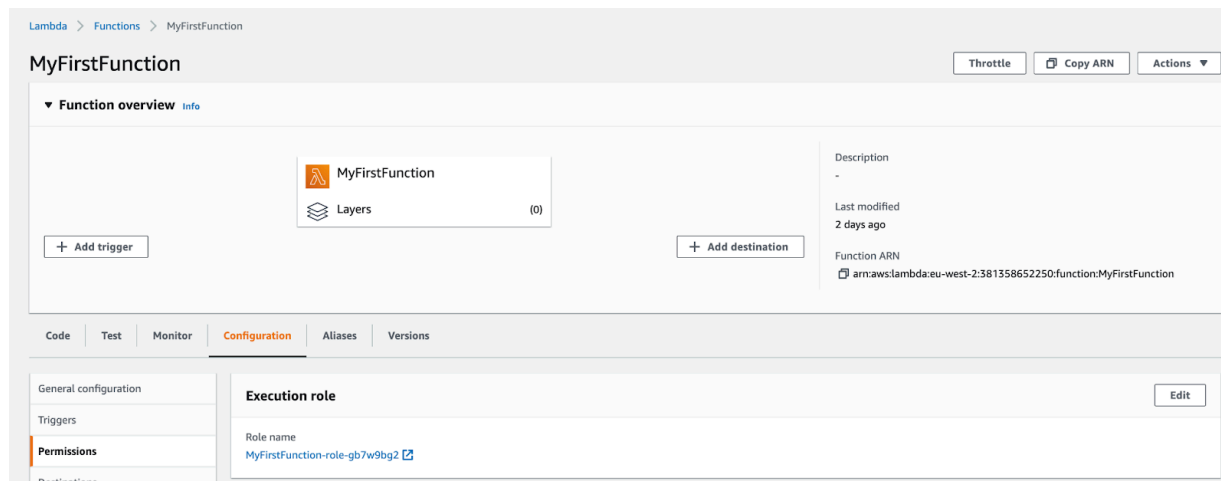
Now you can update the permissions of your Lambda:

1. Navigate to Configuration Permissions .
2. Click the arrow next to the role name.

Identity and Access Management (IAM) opens on a new browser tab.

3. Click Add permissions to attach a new policy.

You can attach a policy to grant the right permissions to the specific secret(s) that should be accessed by the Lambda.



This is an example of a policy to grant your Lambda access to your secret. Make sure to update the policy with the ARN of your secret.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetResourcePolicy",
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret",
        "secretsmanager:ListSecretVersionIds"
      ],
      "Resource": [
        "ARN:TO:YOUR:SECRET"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "secretsmanager:ListSecrets",
      "Resource": "*"
    }
  ]
}
```

Cold start

Cold start can be defined as the set-up time required to get a serverless application's environment up and running when it is invoked for the first time within a defined period.

What is a cold start?

The concept of cold start is very important when using Function as a Service solutions, especially if you are developing a customer-facing application that needs to operate in real time. A cold start is the first request that a new function instance handles. It happens because your function is not yet running and the cloud provider needs to

provision resources and deploy your code before the processing can begin. This usually means that the processing is going to take much longer than expected.

Before execution, the cloud provider needs to:

- Provision the container that will run the code
- Initialize the container / runtime
- Initialize your function

before the trigger information can be forwarded to the function's handler.

For more information about the concept of cold start, see [Operating Lambda: Performance optimization](#).

How to avoid a cold start?

Cloud providers offer the option to always have at least one instance of the function running to completely remove the occurrence of a cold start. You can read more about predictable start-up times in [Provisioned concurrency for AWS Lambda](#).

Cold start with

When a cold start happens with , the following steps are executed before the trigger payload gets processed:

- The function retrieves the flow definition from the Catalog.
- The NiFi Stateless runtime is initialized.
- The flow definition is parsed to list the required NAR files for executing the flow.
- The required NAR files are downloaded from the specified repository unless the NARs are made available in the local storage attached to the function.
- The NARs are loaded into the JVM.
- The flow is initialized and components are started.

The step that might take a significant amount of time is the download of the NAR files. In order to reduce the time required for this step as much as possible, the best option is to list the NAR files required for the flow definition and make those files available to the function.

For the related instructions, see [Cloud storage](#) on page 16.

Cloud storage

Providing custom extensions / NARs

The binary that is deployed for running AWS Lambda does not include the full NiFi deployment with all NARs / extensions to avoid very long startup times and requiring more expensive configurations for Lambda. Instead, it packages only the NARs necessary to run Stateless NiFi as well as the AWS NAR. If any other extension is needed to run the data flow, it will be automatically downloaded when the Lambda function is initialized on a cold start. The extensions will be downloaded from the Cloudera Maven Repository by default.

However, you can configure an alternate Nexus repository by setting the `NEXUS_URL` environment variable to the URL of the Nexus Repository where extensions should be downloaded. For example, to use Google's mirror of the Maven Central repository, set the `NEXUS_URL` environment variable to `https://maven-central.storage-download.googleapis.com/maven2`.



Important: Any configured URL must either be accessible through http or be served over https with a trusted certificate.

This is the recommended approach for downloading extensions that are made available by Cloudera or Apache NiFi. However, if there is a need to provide custom extensions, a simpler route is usually to use Lambda's Layers capability.

1. Create a zip file containing all NAR files that you will need.
2. In the AWS console (or through the CLI), add a layer to your Lambda function, providing the zip file as the source for the layer.
3. Tell Lambda where to look for those extensions by adding another environment variable whose name starts with `EXTENSIONS_DIR_`.

You can add multiple extensions, such as `EXTENSIONS_DIR_1`, `EXTENSIONS_DIR_2`, `EXTENSIONS_DIR_MY_CUSTOM_LIBS`. The name of the environment variable is the path where Lambda should look for the extensions. If the zip file provided as a layer does not contain any directories, the value should be `/opt`. If the zip file is to contain some directory, (for example, `extensions/nars`), the value for the environment variable would be set to `/opt/extensions/nars`.

If you add a larger layer, the Lambda configuration takes a bit longer to load on a cold start, and it takes more memory to load the additional classes. So it is not recommended to upload extremely large zip files of extensions if they are not needed.

Providing additional resources

Besides NiFi extensions, some data flows may also require additional resources to work. For example, a JDBC driver for establishing a database connection, or a CSV file for data enrichment. The recommended approach for providing such resources to your Lambda function is through Lambda Layers. You can create a zip file containing the resources that are required for your data flow. This zip file will be extracted into the `/opt` directory to be accessed by the data flow.

As each deployment of a data flow may need to reference files in a different location depending on its environment, it is a best practice to parameterize resources that need to be accessed by the data flow.

For example, if the data flow contains a `DBCPCConnectionPool Controller Service` for accessing a database, you should use a parameter for the `Database Driver Location(s)` property of the `Controller Service`. This way each deployment can specify the location of the JDBC driver independently.

1. Set the `Database Driver Location(s)` property to `#{JDBC_DRIVER_LOCATION}`.
2. In Lambda, add a layer that contains our JDBC driver, `database-jdbc-driver.jar`.
3. Add an Environment Variable named `JDBC_DRIVER_LOCATION` with a value of `/opt/database-jdbc-driver.jar`.

This way your data flow can be deployed (reused) in many different environments.

S3 Bucket storage

If the extensions or resources of your data flow exceed the size limit of Lambda Layers, you can provide these files in an S3 bucket.

1. Specify the S3 bucket name in the `STORAGE_BUCKET` environment variable.
2. Upload any extensions to a directory in the S3 bucket and specify the directory name in the `STORAGE_EXTENSIONS_DIRECTORY` environment variable.
3. Upload any data flow resources to another directory in the bucket and specify the directory name in the `STORAGE_RESOURCES_DIRECTORY` environment variable.

When your Lambda function does a cold start, it will download any extensions or resources from these locations.

The extensions are directly downloaded in the right location to be loaded in NiFi at startup. The resources are downloaded in `/tmp`. For example if `STORAGE_BUCKET = my-bucket-name`, `STORAGE_RESOURCES_DIRECTORY = my_resources` and there is an `hdfs-site.xml` file in `s3a://my-bucket-name/my_resources/hdfs-site.xml`, the file will be available in the Lambda at `/tmp/my_resources/hdfs-site.xml`.



Note: In order to pull from S3, the Execution Role for the Lambda function must have both the `ListBucket` and `GetObject` permissions for the chosen bucket.

Configuring VPC to deploy Lambda Functions

In some scenarios, it is required to run the Lambda in a specific Virtual Private Cloud (VPC). For example, if your Lambda function has to interact with services in a environment, you must run the Lambda in the same VPC where the service is and you must also fulfill some networking prerequisites for the Function to work as expected.

Before you begin

running on top of AWS Lambda require access to the Catalog deployed on the Control Plane. This means that Lambda requires outbound internet access to download the flow that is going to be run, and it also requires access to some AWS services like Security Token Service (STS) and AWS Secrets Manager.

This [article](#) explains how you can allow a managed, serverless service like AWS Lambda to have internet access. AWS Lambda cannot be deployed on a public subnet in a VPC as it is not possible to grant a Lambda function a public IP address. This means that to have internet access, you need to have at least one private subnet for each availability zone where the Lambda function is going to run and you need to configure a Network Address Translation (NAT) gateway on the public subnets to route its outbound traffic.

Step 1 - Creating VPC private subnets and route tables

A VPC can have multiple subnets, which can be public or private. Depending on the connectivity that you need, you might also have to add gateways and route tables. The Internet Gateway (IG) component enables your instances to connect to the internet through the Amazon EC2 network edge. For more information on basic AWS networking concepts, see [How Amazon VPC works](#).

A public subnet has a direct route to an internet gateway. Resources in a public subnet can access the public internet. By default, the AWS subnets are public, so the instances have public and private IPs and they have a default route table with a route to an internet gateway.

A private subnet does not have a direct route to an internet gateway. Resources in a private subnet require a NAT device to access the public internet, so it cannot have a route to an IG. An EC2 instance in a non-default subnet only has private IPs and does not have internet access.

To allow a private subnet to have internet access you need to use a NAT device, which maps multiple private IPs into a single public IP. So, if you configure the NAT device in a public subnet with a public IP, it can use the internet gateway to route the traffic and get access to the internet.

Use the following procedure to create subnets for your VPC and to create route tables that will be associated with the subnets.

1. Create one private subnet per availability domain of your VPC where you will deploy the Lambda function.
 - a. Open the Amazon VPC console.
 - b. Click Subnets in the left navigation pane.
 - c. Click Create subnet.
 - d. For the VPC ID, select the VPC where you want to add the new subnet.



Note: This should be the Environment subnet.

- e. (Optional) For Name, add a name for your subnet.
- f. Define the Availability Zone and the CIDR.
- g. Click Create subnet.

2. Create a route table that will be associated with the subnet.
 - a. Click Route tables in the left navigation pane.
 - b. Click Create route table.
 - c. (Optional) For Name, add a name for your route table.
 - d. For VPC, choose your VPC.
 - e. (Optional) To add a tag, click Add new tag and add the tag key and tag value.
 - f. Click Create route table.



Note: Follow these steps for all subnets and use a common nomenclature to identify the route table of a subnet.

Step 2 - Creating NAT gateways and configuring route tables

Now that you have your subnets, you need to create and configure the Network Address Translation (NAT) gateways that map the private IPs to a public IP and allow the instances in your private subnets to connect to services outside your VPC.

1. Create NAT gateways.
 - a. Open the Amazon VPC console.
 - b. Click NAT gateways in the left navigation pane.
 - c. Click Create NAT gateway.
 - d. (Optional) Add a name for the NAT gateway.
 - e. Define the subnet where you want to create the NAT gateway.
 - f. Define the connectivity type.



Important: Important: The subnet where the NAT gateway is deployed must be a public subnet to allow internet access, so make sure that the connectivity type is set to public. For more information about the difference between public and private NAT gateways, see [NAT gateways](#).

- g. Allocate an Elastic IP. For more information, see [Elastic IP addresses](#).
2. Configure the route tables of the private subnets created in the previous procedure.
 - a. Click Route tables.
 - b. Search for the route table.
 - c. Click Routes Edit routes .

There are two routes that need to be defined.

Route 1

- Destination: <VPC CIDR>
- Target: local
- Status: Active

Route 2

- Destination: 0.0.0.0/0
- Target: <Search for one of the NAT Gateways created>
- Status: Active

- d. Click Save Changes.

3. Associate each route table with each private subnet.
 - a. Click Subnet Associations.
 - b. Select the specific subnet.
 - c. Save the associations.
 - d. Validate that subnets have the route table with it.



Note: Follow these steps for all route tables of the private subnets.

Result

With this, the private subnet components are able to communicate with other components within the same VPC and have internet access.

File system for content repository

If your function is going to process very large files, the data may not be held in memory, which would cause errors. In that case you may want to attach a File System to your Lambda and tell it to use the File System as the Content Repository.

Configuration

If you want to [attach a File System to your Lambda](#), you are required to [connect your Lambda to the VPC](#) where the File System is provisioned.

However, by default, Lambda runs your functions in a secure VPC with access to AWS services and the internet. Lambda owns this VPC, which is not connected to your account's default VPC. When you connect a function to a VPC in your account, the function cannot access the internet unless your VPC provides access. Access to the internet is required so that it can request the Catalog and retrieve the flow definition to be executed.

In such a situation, a recommended option is to create a VPC with a public subnet and multiple private subnets (to increase resiliency across multiple availability zones). You can then create an Internet Gateway and NAT Gateway to give your Lambda access to the internet. For more information, see the [AWS Knowledge Center](#).

Concurrent access

Depending on your Lambda's configuration and the throughput of the events triggering the function, you may have multiple instances of the Lambda running concurrently and each instance would share the same File System. For more information, see this [AWS Compute Blog article](#).

To ensure each instance of the function has its own “content repository”, you can use the `CONTENT_REPO` environment variable when configuring your Lambda and give it the value of the *Local mount path* of the attached File System:

The screenshot shows the AWS Lambda console with the 'Configuration' tab selected. On the left sidebar, 'File systems' is highlighted. The main panel displays the 'File system' configuration for a specific file system (fs-b432f544). The configuration includes:

- File system ID:** fs-b432f544
- File system state:** Available
- File system ARN:** arn:aws:elasticfilesystem:eu-west-2:381358652250:file-system/fs-b432f544
- Access point ID:** fsap-02e81ba594dcc505e
- Access point lifecycle state:** Available
- Access point ARN:** arn:aws:elasticfilesystem:eu-west-2:381358652250:access-point/fsap-02e81ba594dcc505e
- Local mount path:** /mnt/fs

Buttons for 'Edit' and 'Remove' are visible in the top right corner of the configuration panel.

In this case, each instance of the function will create its content repository on the File System under the following path: `${CONTENT_REPO}/<function name>/content_repository/<epoch timestamp>-<UUID>`

You can share the same File System across multiple Lambda functions if desired depending on the IO performances you need.

Output size constraints

When you design a data flow that will be deployed within AWS Lambda, you create a Process Group that contains a single Input Port (for example named Input) and two Output Ports (for example named success and failure). If data is routed to the failure port, the entire invocation is failed. If data is routed to the success port, the contents of the FlowFile become the output of the Lambda function. This allows you to set a destination for your AWS Lambda. For example, you can send the output to an SQS queue or another Lambda function.

Consider the following when designing your flow:

- AWS Lambda limits the size of the output. While the maximum output size is configurable, the default is 6 MB. So, if the size of the FlowFile routed to success exceeds 6 MB, the result is a failure, even if you have no destination configured.
- Take into account the size constraints on any downstream system that may be used as the Lambda destination, and do not send a large FlowFile as the output of the Process Group. Instead, you should have the last processor in the data flow auto-terminate the success relationship so that there is no output FlowFile. Or you can use a ReplaceText processor with the Replacement Strategy set to Always Replace. This allows you to set the contents to some predefined text, such as: { "action": "perform dataflow", "success": true, "fileSize": \${fileSize} }

DataFlow state

By default, if your data flow contains any stateful processors (for example, ListSFTP), this state will automatically be stored in a DynamoDB table called `nifi_state`. This name can be set using the `DYNAMODB_STATE_TABLE` environment variable. If state is to be stored in DynamoDB, the role executing the Lambda function requires permissions to create and describe tables, and permissions to put and get items. If the data flow contains no stateful processors, this provider is not used, and requires no extra permissions.

The default DynamoDB billing mode is `PAY_PER_REQUEST`. To change this to `PROVISIONED`, set the `DYNAMODB_BILLING_MODE` environment variable to `PROVISIONED`, and set the `DYNAMODB_READ_CAPACITY_UNITS` and `DYNAMODB_WRITE_CAPACITY_UNITS` variables accordingly. For more information, see [Read/write capacity mode](#).

The DynamoDB state provider can be disabled even if your data flow contains stateful processors by setting the `DISABLE_STATE_PROVIDER` environment variable to `true`.



Warning: This will cause the processor state to get lost between function executions.

Configuring Kerberos

Depending on the dataflow that is used, you may need to enable Kerberos authentication in order to interact with some service(s).

To enable Kerberos, you must provide an appropriate `krb5.conf` file. This file tells the Function where the Key Distribution Center (KDC) is located and the Kerberos Realm to use.

To specify this, make the `krb5.conf` available to . You can do it by using the Lambda Layer or by placing the file in an S3 bucket. For more information, see [S3 Bucket storage](#).

Additionally, you must tell Function where it can find this `krb5.conf` file. To do this, add an environment variable to your Lambda function. The environment variable must have the name `KRB5_FILE` and the value must be the fully qualified filename of the `krb5.conf` file. For example, it might use `/opt/krb5.conf`.

In order to use Keytabs in your data flow, it is also important to provide the necessary keytab files to the Function. You can accomplish it by using a Lambda Layer or an S3 bucket. While the `krb5.conf` file is a system configuration, the keytab is not. The keytab will be configured in the data flow on a per-component basis. For example, one processor may use keytab `nifi-keytab-1` while some Controller Service makes use of keytab `hive-keytab-4`. The location of these files may change from deployment to deployment. For example, for an on-premises deployment, keytabs may be stored in `/etc/security/keytabs` while in Function, they may be made available in `/opt`.

So it is better to use Parameter Contexts to parameterize the location of the keytab files. This is a good practice for any file-based resource that will need to be made available to a deployment. In this case, because there is a parameter referencing the location, you need to provide Function with a value for that parameter. For more information, see *Parameters*.

Related Information

[S3 Bucket storage](#)

[Parameters](#)

Handling failures

You can use a failure Output Port to notify the Cloud function that an invocation has failed.

Lambda will automatically retry the invocation, by default, twice. After that, if it still fails, the default option in Lambda is to simply drop the event notification. This could result in data loss, as it means that our Lambda function will not have a chance to process the data again.

Instead, you can also choose to use a "Dead-Letter Queue" for the data:

1. On the Function details page of the AWS Console, navigate to the Configuration Asynchronous invocation tab.
2. Click Edit.

You can see an option to configure Dead-letter queue service. Use a dead-letter queue to send discarded events to an Amazon SQS queue or Amazon SNS topic. Your function's execution role requires permission to write to the queue or topic.

You can use None (the default, which drops the notification), or you can configure an SNS or SQS queue to use in the case of failure. This would give you the option to later go back and manually process the data or triage and analyze it to understand why processing failed.

For more information, see the [AWS documentation](#).

Testing your Lambda function

Once you have built and verified your dataflow and uploaded it to the catalog, you can create your Lambda function.

After creating the Lambda function and configuring all necessary Environment Variables, you should test that you have configured all of the settings correctly before publishing the Lambda.

For this, you can use the Test tab of AWS Lambda Function details screen. Lambda provides you with many options for sample data, so you can easily create an example JSON document to simulate the events that your Lambda function will handle.

Click the Test button to run the data flow. If there are problems, you can inspect the log messages to understand the problem. If the logs do not provide enough details, you may need to adjust the log levels to gain more debugging information. For more information, see *Adjusting logs levels*.

Function ARN
arn:aws:lambda:eu-west-2:381358652250:function:MyFirstFunction

Code **Test** Monitor Configuration Aliases Versions

Test event Format Save changes Test

Invoke your function with a test event. Choose a template that matches the service that triggers your function, or enter your event document in JSON.

☒ New event
☐ Saved event

Template
cloudwatch-scheduled-event

Name
MyEventName

```

1- {
2-   "id": "cdc73f9d-aed9-11e3-9d5a-835b769cd9c",
3-   "detail-type": "Scheduled Event",
4-   "source": "aws.events",
5-   "account": "123456789012",
6-   "time": "1970-01-01T00:00:00Z",
7-   "region": "us-east-1",
8-   "resources": [
9-     "arn:aws:events:us-east-1:123456789012:rule/ExampleRule"
10-  ],
11-   "detail": {}
12- }

```

The amount of time the Lambda function takes to run depends heavily on the data flow and the number of extensions it needs. Because the Lambda function may have to download some extensions from Nexus and perform initialization, a cold start may take longer time. Even 20-30 seconds is not uncommon for a data flow with several extensions, while other data flows may complete in 10 seconds.

After you run the function successfully, it may be helpful to run several additional iterations to understand how the function will perform when a cold start is not necessary. Again, this depends heavily upon the configured data flow and other services that it may interact with. A simple data flow may complete in as a short time as 10 milliseconds while a data flow that must perform complex transformations and reach out to one or more web services or databases for enrichment may take several seconds to complete.

It is important to adjust Lambda's timeout configuration on the **Configuration General configuration** tab if the cold start takes longer than the amount of time allocated.

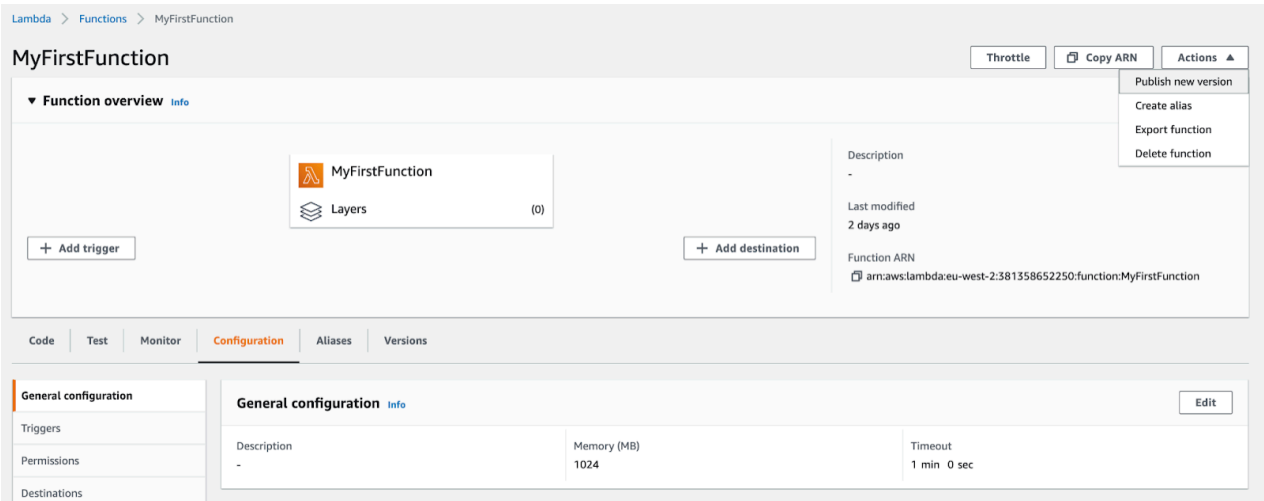
Related Information

[Adjusting logs levels](#)

Publishing your Lambda function

When the Lambda configuration is ready, you can publish a new version of the Lambda function.


For more information about Lambda function versions, see the [AWS documentation](#).



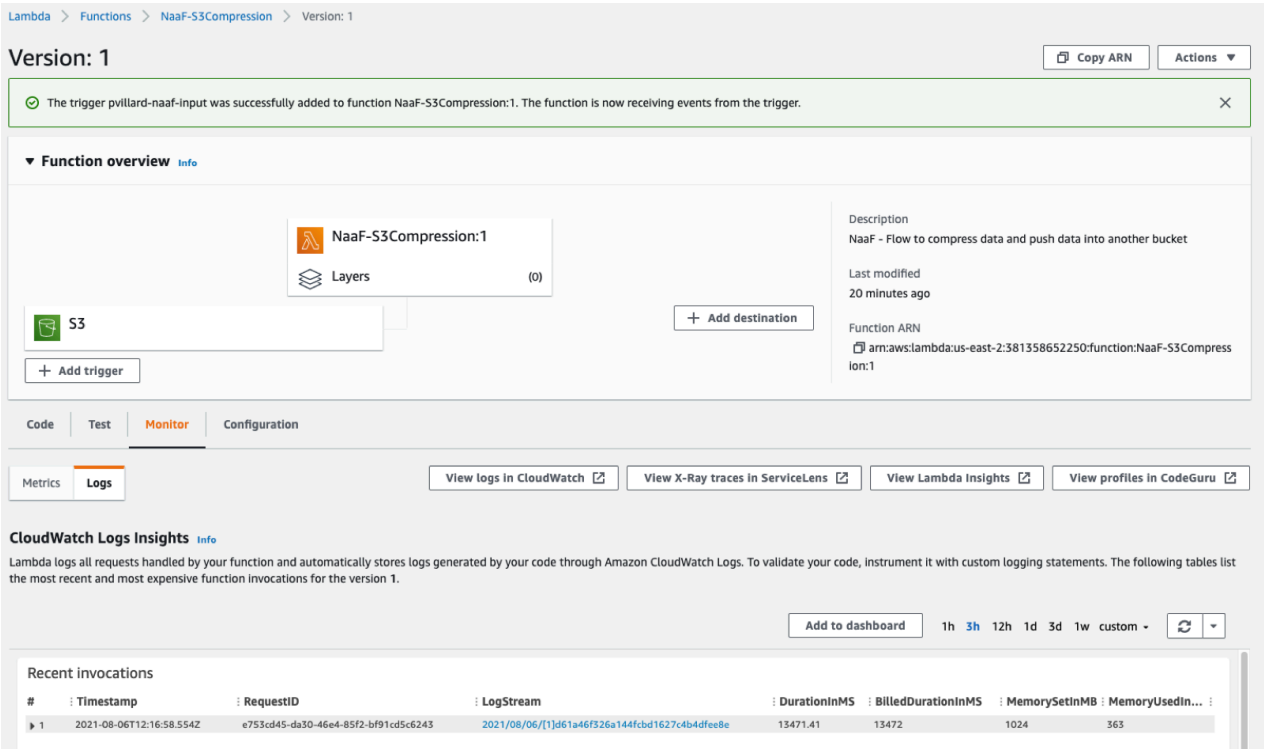
You can define the Triggers/Destinations for the newly published version of your Lambda function and everything will be ready to process the events.

Monitoring and logs

For each invocation of your Lambda, you are able to access information about the duration of the execution and the resources provisioned and used during the function’s execution.

 **Note:** With , a cold start may take a few seconds.

This view is useful in case you want to adjust the amount of memory provisioned with your function and optimize the cost charged by AWS:



From this view, you can also access the logs of the function's execution:

▶	2021-08-06T14:16:54.796+02:00	[main] INFO org.apache.nifi.groups.StandardProcessGroup - Added Connection[Id=1e2410b5-f90c-3cde-80bd-df5845e888fb, Source ID=c28026c1-ae71-3be9-bb8c-45a94ca745f3]
▶	2021-08-06T14:16:54.797+02:00	[main] INFO org.apache.nifi.groups.StandardProcessGroup - Added Connection[Id=7fba82c7-c0cd-3b6c-b50a-4b191478c756, Source ID=6b3f9082-d130f25d-197f-3dca-b715-b385]
▶	2021-08-06T14:16:54.797+02:00	[main] INFO org.apache.nifi.groups.StandardProcessGroup - Added Connection[Id=2816399e-f373-38f1-82d2-13a31f48a14d, Source ID=60d4c52c-3f07-30fe-af16-65139]
▶	2021-08-06T14:16:54.814+02:00	[main] INFO org.apache.nifi.groups.StandardProcessGroup - Added Connection[Id=8541305e-082e-313c-99e3-bb77451ffdb4, Source ID=3e5e1010-2021-08-06T14:16:54.815+02:00]
▶	2021-08-06T14:16:54.815+02:00	[main] INFO org.apache.nifi.groups.StandardProcessGroup - Updating LocalPort[Id=3e5e1010-d6e4-3e57-9478-b2a70d2a7355, type=INPUT_PORT, Source ID=3e5e1010-2021-08-06T14:16:54.894+02:00]
▶	2021-08-06T14:16:54.894+02:00	[main] INFO org.apache.nifi.stateless.flow.StandardStatelessDataFlowFactory - NiFi Stateless Engine and Dataflow created and initialize
▶	2021-08-06T14:16:54.903+02:00	[main] INFO org.apache.nifi.controller.scheduling.StatelessProcessScheduler - Starting PutS3Object[id=58cbcfbd-84d8-3fe7-a76e-0ee58d9302a5]
▶	2021-08-06T14:16:54.903+02:00	[main] INFO org.apache.nifi.controller.StandardProcessorNode - Starting PutS3Object[id=58cbcfbd-84d8-3fe7-a76e-0ee58d9302a5]
▶	2021-08-06T14:16:54.916+02:00	[main] INFO org.apache.nifi.controller.scheduling.StatelessProcessScheduler - Starting FetchS3Object[id=6b3f9082-ee4b-3c9b-a2b7-e55ff0db7816]
▶	2021-08-06T14:16:54.916+02:00	[main] INFO org.apache.nifi.controller.StandardProcessorNode - Starting FetchS3Object[id=6b3f9082-ee4b-3c9b-a2b7-e55ff0db7816]
▶	2021-08-06T14:16:54.916+02:00	[main] INFO org.apache.nifi.controller.scheduling.StatelessProcessScheduler - Starting EvaluateJsonPath[id=d130f25d-197f-3dca-b715-b385]
▶	2021-08-06T14:16:54.916+02:00	[main] INFO org.apache.nifi.controller.StandardProcessorNode - Starting EvaluateJsonPath[id=d130f25d-197f-3dca-b715-b385dc6b5676]
▶	2021-08-06T14:16:54.916+02:00	[main] INFO org.apache.nifi.controller.scheduling.StatelessProcessScheduler - Starting SplitJson[id=c28026c1-ae71-3be9-bb8c-45a94ca745f3]
▶	2021-08-06T14:16:54.916+02:00	[main] INFO org.apache.nifi.controller.StandardProcessorNode - Starting SplitJson[id=c28026c1-ae71-3be9-bb8c-45a94ca745f3]
▶	2021-08-06T14:16:54.917+02:00	[main] INFO org.apache.nifi.controller.scheduling.StatelessProcessScheduler - Starting CompressContent[id=60d4c52c-3f07-30fe-af16-65139]
▶	2021-08-06T14:16:54.917+02:00	[main] INFO org.apache.nifi.controller.StandardProcessorNode - Starting CompressContent[id=60d4c52c-3f07-30fe-af16-65139cef438]
▶	2021-08-06T14:16:55.859+02:00	[Component Lifecycle for dataflow S3Compression Thread-2] INFO org.apache.nifi.processors.aws.s3.FetchS3Object - FetchS3Object[id=6b3f9082-ee4b-3c9b-a2b7-e55ff0db7816]
▶	2021-08-06T14:16:55.860+02:00	[Component Lifecycle for dataflow S3Compression Thread-1] INFO org.apache.nifi.processors.aws.s3.PutS3Object - PutS3Object[id=58cbcfbd-84d8-3fe7-a76e-0ee58d9302a5]
▶	2021-08-06T14:16:55.874+02:00	[Component Lifecycle for dataflow S3Compression Thread-2] INFO org.apache.nifi.processors.aws.s3.FetchS3Object - FetchS3Object[id=6b3f9082-ee4b-3c9b-a2b7-e55ff0db7816]
▶	2021-08-06T14:16:55.874+02:00	[Component Lifecycle for dataflow S3Compression Thread-1] INFO org.apache.nifi.processors.aws.s3.PutS3Object - PutS3Object[id=58cbcfbd-84d8-3fe7-a76e-0ee58d9302a5]
▶	2021-08-06T14:16:57.212+02:00	[main] INFO org.apache.nifi.stateless.flow.StandardStatelessFlow - Successfully initialized components in 2297 millis (4 millis to perf)
▶	2021-08-06T14:16:57.392+02:00	[main] INFO org.apache.nifi.aws.lambda.StatelessNiFiFunctionHandler - Enqueued 798 bytes for Input Port events
▶	2021-08-06T14:16:57.533+02:00	[Run Dataflow S3Compression] INFO org.apache.nifi.processors.standard.SplitJson - SplitJson[id=c28026c1-ae71-3be9-bb8c-45a94ca745f3] Sp
▶	2021-08-06T14:16:58.039+02:00	[Run Dataflow S3Compression] INFO org.apache.nifi.processors.aws.s3.FetchS3Object - FetchS3Object[id=6b3f9082-ee4b-3c9b-a2b7-e55ff0db7816]
▶	2021-08-06T14:16:58.075+02:00	[Run Dataflow S3Compression] INFO org.apache.nifi.processors.standard.CompressContent - CompressContent[id=60d4c52c-3f07-30fe-af16-65139]
▶	2021-08-06T14:16:58.541+02:00	[Run Dataflow S3Compression] INFO org.apache.nifi.processors.aws.s3.PutS3Object - PutS3Object[id=58cbcfbd-84d8-3fe7-a76e-0ee58d9302a5]
▶	2021-08-06T14:16:58.552+02:00	[Run Dataflow S3Compression] INFO org.apache.nifi.stateless.flow.StandardStatelessFlow - Ran dataflow in 1146 millis
▶	2021-08-06T14:16:58.553+02:00	[main] INFO org.apache.nifi.aws.lambda.StatelessNiFiFunctionHandler - Successfully triggered dataflow but there were no output FlowFile
▶	2021-08-06T14:16:58.554+02:00	END RequestId: e753cd45-da30-46e4-85f2-bf91cd5c6243
▶	2021-08-06T14:16:58.554+02:00	REPORT RequestId: e753cd45-da30-46e4-85f2-bf91cd5c6243 Duration: 13471.41 ms Billed Duration: 13472 ms Memory Size: 1024 MB Max Memory

Additional monitoring capabilities may be enabled in AWS.

Adjusting logs levels

makes use of the SLF4J Simple Logger for logging purposes. You can update the log levels by adjusting the JVM's system properties.

To do so, you need to set the `JAVA_TOOL_OPTIONS` environment variable.

- To set a log level for a given logger, you can set the JVM system property named `org.slf4j.simpleLogger.log.<logger name>` to the desired log level.

For example, if you want to enable DEBUG logging for the `AwsSecretsManagerParameterValueProvider` to understand whether or not parameters are being fetched from AWS Secrets Manager, you would set the `JAVA_TOOL_OPTIONS` environment variable to `-Dorg.slf4j.simpleLogger.log.org.apache.nifi.stateless.parameter.AwsSecretsManagerParameterValueProvider=DEBUG`.

- You can also set additional logger levels by adding multiple `-D` options separated by spaces. For example, to enable DEBUG logs on both the AWS Secrets Manager Parameter Value Provider and the Stateless Bootstrap class, you would set `JAVA_TOOL_OPTIONS` to a value of:

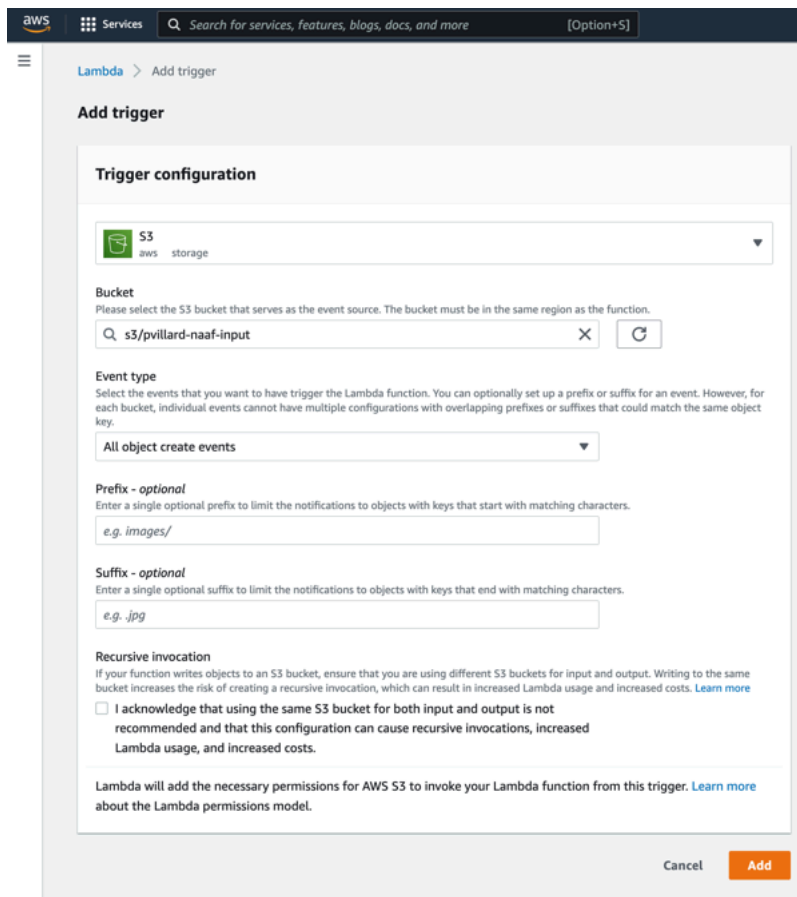
```
-Dorg.slf4j.simpleLogger.log.org.apache.nifi.stateless.parameter.AwsSecretsManagerParameterValueProvider=DEBUG
-Dorg.slf4j.simpleLogger.log.org.apache.nifi.stateless.bootstrap.StatelessBootstrap=DEBUG
```

AWS Lambda triggers

All AWS Lambda triggers are supported with . In this section you can review the most commonly used triggers and examples of the FlowFile's content that would be generated following a triggering event. You can add triggers on the Configuration Triggers tab.

S3 Trigger

This trigger can be used, for example, to start your function whenever a file is landing into AWS S3:



The screenshot shows the AWS Lambda console's 'Add trigger' page. The 'Trigger configuration' section is active, showing an S3 trigger setup. The 'Bucket' field is set to 's3/pvillard-naaf-input'. The 'Event type' is set to 'All object create events'. The 'Prefix' is set to 'e.g. images/' and the 'Suffix' is set to 'e.g. .jpg'. A checkbox for 'Recursive invocation' is checked, with a warning message below it. At the bottom, there are 'Cancel' and 'Add' buttons.

aws Services Search for services, features, blogs, docs, and more [Option+S]

Lambda > Add trigger

Add trigger

Trigger configuration

S3 aws storage

Bucket
Please select the S3 bucket that serves as the event source. The bucket must be in the same region as the function.
s3/pvillard-naaf-input X C

Event type
Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.
All object create events

Prefix - optional
Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters.
e.g. images/

Suffix - optional
Enter a single optional suffix to limit the notifications to objects with keys that end with matching characters.
e.g. .jpg

Recursive invocation
If your function writes objects to an S3 bucket, ensure that you are using different S3 buckets for input and output. Writing to the same bucket increases the risk of creating a recursive invocation, which can result in increased Lambda usage and increased costs. [Learn more](#)
☒ I acknowledge that using the same S3 bucket for both input and output is not recommended and that this configuration can cause recursive invocations, increased Lambda usage, and increased costs.

Lambda will add the necessary permissions for AWS S3 to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

Cancel Add

The FlowFile's content would look like this:

```

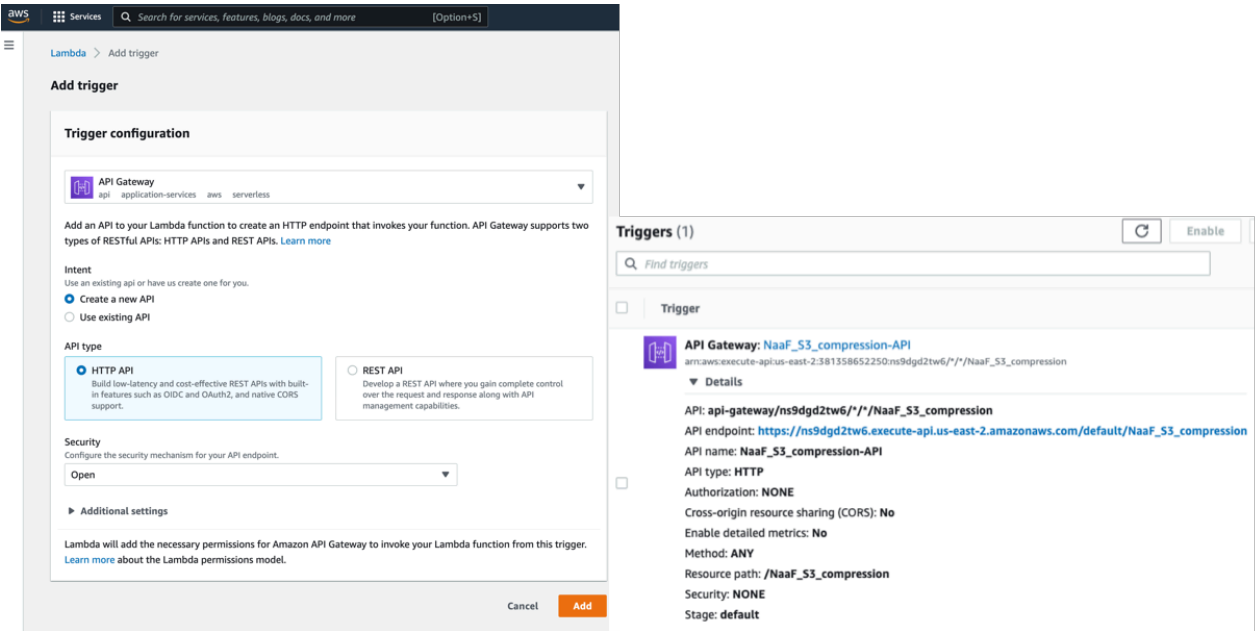
1 {
2   "Records": [
3     {
4       "eventVersion": "2.1",
5       "eventSource": "aws:s3",
6       "awsRegion": "us-east-2",
7       "eventTime": "2021-08-06T09:53:57.377Z",
8       "eventName": "ObjectCreated:Put",
9       "userIdentity": {
10        "principalId": "AWS:AIDAVRSV7HNNHFQQ0E2F4"
11      },
12       "requestParameters": {
13        "sourceIPAddress": "18.222.152.52"
14      },
15       "responseElements": {
16        "x-amz-request-id": "WBPM5W0026J3NWEH",
17        "x-amz-id-2": "Cco/DVtbHFYtb+rXQrEMUFJnZSxtebZL61KG79emsACUDqUIyrAziY3frha9AzX+RSLx9syJmL40MCn06poZv1dJQJg5TgRx"
18      },
19       "s3": {
20        "s3SchemaVersion": "1.0",
21        "configurationId": "ad0ec124-9c9f-476e-9dd1-9e513d9af822",
22        "bucket": {
23          "name": "pvillard-naaf-input",
24          "ownerIdentity": {
25            "principalId": "A339RKRO79D9V"
26          },
27          "arn": "arn:aws:s3:::pvillard-naaf-input"
28        },
29        "object": {
30          "key": "04f56bcf-cb15-4244-9379-2abc7ac0cd8d",
31          "size": 1024,
32          "eTag": "265bc4c12162ffca179a58975728a033",
33          "sequencer": "00610D06B8BE365D41"
34        }
35      }
36     ]
37   }
38 }

```

API Gateway Trigger

This trigger is used to expose an HTTPS endpoint so that clients can trigger the function with HTTP requests. This is particularly useful to build microservices. It can also be used to create an HTTPS Gateway for ingesting data into Kafka, or to provide an HTTP Gateway to many MiNiFi agents sending data into the cloud. See more information about these use cases in the *next sections*.

Configuration example:



Example of event generated:

```

1  {
2    "version": "2.0",
3    "routeKey": "ANY /NaaF_S3_compression",
4    "rawPath": "/default/NaaF_S3_compression",
5    "rawQueryString": "",
6    "headers": {
7      "accept-encoding": "gzip",
8      "content-length": "34",
9      "content-type": "application/json",
10     "date": "Fri, 06 Aug 2021 12:35:29 GMT",
11     "host": "ns9dgd2tw6.execute-api.us-east-2.amazonaws.com",
12     "x-amzn-trace-id": "Root=1-610d2c92-7651f91a5dcbab8671048936",
13     "x-forwarded-for": "3.19.60.232",
14     "x-forwarded-port": "443",
15     "x-forwarded-proto": "https"
16   },
17   "requestContext": {
18     "accountId": "381358652250",
19     "apiId": "ns9dgd2tw6",
20     "domainName": "ns9dgd2tw6.execute-api.us-east-2.amazonaws.com",
21     "domainPrefix": "ns9dgd2tw6",
22     "http": {
23       "method": "POST",
24       "path": "/default/NaaF_S3_compression",
25       "protocol": "HTTP/1.1",
26       "sourceIp": "3.19.60.232",
27       "userAgent": ""
28     },
29     "requestId": "DpPm3j83iYcEP7w=",
30     "routeKey": "ANY /NaaF_S3_compression",
31     "stage": "default",
32     "time": "06/Aug/2021:12:35:30 +0000",
33     "timeEpoch": 1628253330104
34   },
35   "body": "{\n  \"data\": \"this is my payload\"\n}",
36   "isBase64Encoded": false
37 }

```

EventBridge (CloudWatch Events) - CRON driven functions

This trigger can be used with rules including CRON expressions so that your function is executed at given moments. You can use it, for example, to execute a particular processing once a day.

aws Services Search for services, features, blogs, docs, and more [Option+S]

Lambda > Add trigger

Add trigger

Trigger configuration

EventBridge (CloudWatch Events)
aws events management-tools

Rule
Pick an existing rule, or create a new one.

☒ Create a new rule
☐ Existing rules

Rule name
Enter a name to uniquely identify your rule.

Rule description
Provide an optional description for your rule.

Rule type
Trigger your target based on an event pattern, or based on an automated schedule.

☐ Event pattern
☒ Schedule expression

Schedule expression
Self-trigger your target on an automated schedule using [Cron or rate expressions](#). Cron expressions are in UTC.

rate(1 day)

e.g. rate(1 day), cron(0 17 ? * MON-FRI *)

Lambda will add the necessary permissions for Amazon EventBridge (CloudWatch Events) to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

Cancel Add

Kinesis

This trigger can be used to process a stream of events using . The trigger configuration could be:

The screenshot shows the 'Add trigger' configuration page in the AWS Lambda console. The 'Trigger configuration' section is active, showing a dropdown for 'Kinesis' with sub-items 'analytics', 'aws', and 'streaming'. Below this, the 'Kinesis stream' field is set to 'naaf-test'. The 'Consumer - optional' dropdown is set to 'No consumer'. There is an unchecked checkbox for 'Enable trigger'. The 'Batch size' is set to '100'. The 'Starting position' is set to 'Latest'. The 'Batch window - optional' field is empty. At the bottom, there is a note about execution role permissions and 'Cancel' and 'Add' buttons.

The FlowFile's content would look like this:

```

1 {
2   "Records": [
3     {
4       "kinesis": {
5         "kinesisSchemaVersion": "1.0",
6         "partitionKey": "1422215794",
7         "sequenceNumber": "49620838107107426836706204605255346319435863317985361922",
8         "data": "ewogICJkYXRhIjogInRoZXN0bG9hZC9hZCIKfQ==",
9         "approximateArrivalTimestamp": 1628255950.71
10      },
11      "eventSource": "aws:kinesis",
12      "eventVersion": "1.0",
13      "eventId": "shardId-000000000000:49620838107107426836706204605255346319435863317985361922",
14      "eventName": "aws:kinesis:record",
15      "invokeIdentityArn": "arn:aws:iam:381358652250:role/service-role/Naaf_S3_compression-role-rvm8p2yc",
16      "awsRegion": "us-east-2",
17      "eventSourceARN": "arn:aws:kinesis:us-east-2:381358652250:stream/naaf-test"
18    }
19  ]
20 }

```

Note that the data associated with a particular record may be base 64 encoded. In this case, you may want to consider the Base64EncodeContent processor in your flow definition to decode the content.

AWS MQ

This trigger can be used to trigger your Lambda based on events received in specific queues of AWS MQ. The trigger configuration may look like this:

The screenshot shows the 'Add trigger' configuration page in the AWS Lambda console. The 'Trigger configuration' section is active, showing the following settings:

- MQ** (aws messaging multi-protocol)
- Amazon MQ broker**: Select an Amazon MQ broker. **my-naaf-mq-broker-test**
- ☒ **Enable trigger**: Enable the trigger now, or create it in a disabled state for testing (recommended).
- Batch size**: The largest number of records that will be read from your stream at once. **100**
- Batch window - optional**: The maximum amount of time to gather records before invoking the function, in seconds. **0**
- Queue name**: Enter the name of the Amazon MQ broker destination queue to consume. **naaf**
- Virtual host - optional**: Enter the path to the target virtual host. Required for RabbitMQ. **e.g. /my-host**
- Authentication**: Choose the authentication method and secret key required to access the brokers in your cluster. **BASIC_AUTH**
- Add** button: This field is required if your brokers are accessed over the Internet.
- Remove** button
- Cancel** and **Add** buttons at the bottom.

The FlowFile's content would look like this:

```

1 {
2   "eventSource": "aws:mq",
3   "eventSourceArn": "arn:aws:mq:us-east-2:381358652250:broker:my-naaf-mq-broker-test:b-7feee146-7a61-40a3-9245-d06419a5c118",
4   "messages": [
5     {
6       "messageID": "ID:b-7feee146-7a61-40a3-9245-d06419a5c118-1.mq.us-east-2.amazonaws.com-42681-1630411308974-4:1:1:1",
7       "messageType": "jms/text-message",
8       "timestamp": 1630412991976,
9       "deliveryMode": 1,
10      "correlationID": "",
11      "replyTo": "null",
12      "destination": {
13        "physicalName": "naaf"
14      },
15      "redelivered": false,
16      "type": "",
17      "expiration": 0,
18      "priority": 0,
19      "data": "ew0KICAiZGF0YSI6ICJ0aGZlIGZlIG15IHBeWxvYWQidQp9",
20      "brokerInTime": 1630412991976,
21      "brokerOutTime": 1630412991980
22    }
23  ]
24 }
```


Note that the data associated with a particular message may be base 64 encoded. In this case, you may want to consider the Base64EncodeContent processor in your flow definition to decode the content.


AWS SNS

When using the AWS SNS trigger with a particular topic, the configuration may look like this:

Lambda > Add trigger

Add trigger


Trigger configuration

 **SNS**
aws messaging notifications pub-sub push

SNS topic
Select the SNS topic to subscribe to.

arn:aws:sns:us-east-2:381358652250:naaf

Lambda will add the necessary permissions for Amazon SNS to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

 The Lambda console no longer supports disabling SNS triggers. Delete these triggers to stop further actions.

Cancel **Add**

The FlowFile's content would look like this:

```

1 {
2   "Records": [
3     {
4       "EventSource": "aws:sns",
5       "EventVersion": "1.0",
6       "EventSubscriptionArn": "arn:aws:sns:us-east-2:381358652250:naaf:d9d324e6-26c9-4c9f-bc17-787422a526e8",
7       "Sns": {
8         "Type": "Notification",
9         "MessageId": "2d8aafed-45fc-5d20-9b73-f64563caaf6",
10        "TopicArn": "arn:aws:sns:us-east-2:381358652250:naaf",
11        "Subject": null,
12        "Message": "This is my message...",
13        "Timestamp": "2021-08-31T12:43:34.806Z",
14        "SignatureVersion": "1",
15        "Signature":
16        "fjxVP2P5wt5jQicwrW0sH0RDQQAhyKaPv7NrdRcj0sy1KqFc5pXTiW4k2bzKcjtvpPaISnKuVY042B1X6FPCbWjhGe00dPKMJNFT1Zg2pFaUFYUHPXmfrBu3Nfj8sLEkgQsP9vIo0ruNil1j15kucg22Pb/
17        46adDggMyLri928I+4zZ+3nSTA3GMTt49mIeLTCzw8JUeQsEY3bCqt0sqvuBSM1NyqJYaKoiFY30YLV8ZhUPas5jbG32en1crSnX8Z58+WB0Tc16Z/ubreCKrAbTYIK1FoJBrVmze8uMHoAjUgcYDvy6T/
18        0EL0ZnH/tuueGQ8E7Meyqvvdwscsl8fg==",
19        "SigningCertUrl": "https://sns.us-east-2.amazonaws.com/SimpleNotificationService-010a507c1833636cd94bdb98bd93083a.pem",
20        "UnsubscribeUrl": "https://sns.us-east-2.amazonaws.com/?Action=Unsubscribe&SubscriptionArn=arn:aws:sns:us-east-2:381358652250:naaf:d9d324e6-26c9-4c9f-bc17-
21        787422a526e8",
22        "MessageAttributes": {
23          "naafKeyExample": {
24            "Type": "String",
25            "Value": "naafValueExample"
26          }
27        }
28      }
29    ]
30  }

```


AWS SQS

When using the AWS SQS trigger with a particular queue, the configuration may look like this:

Lambda > Add trigger

Add trigger

Trigger configuration



SQS
aws queue

SQS queue

Choose or enter the ARN of an SQS queue.

✕
↺

Batch size

The maximum number of messages to retrieve in a single batch.

Batch window

The maximum amount of time to gather records before invoking the function, in seconds.

In order to read from the SQS trigger, your execution role must have proper permissions.

☒ **Enable trigger**

Enable the trigger now, or create it in a disabled state for testing (recommended).

Cancel
Add

The FlowFile's content would look like this:

```

1 {
2   "Records": [
3     {
4       "messageId": "2f895bd2-256e-4a37-bf50-e75ecbb9706c",
5       "receiptHandle": "AQEBjVQd44ww6p0skNG+ss7ltM63PJL+IGBxeVyQ/3vd9CubJ+600jofED168mSrz21oA2fu1gqRneuVVB4c9nCoLY1y6yeaF6Mo4yqCJDECwscwU7hfhmCfYk9bNu0chUb3uIC2z1A2/
6       P+RI5hodPc4gX6h0LH5aAv6At8TP5s7dsyzlmHjCzv2QVb4oE+CAVYIVwz0rXKIj1PgzwU4Jk4s4Go7+Vravh0T5xiI5g43WSg03yx1PK1qFE8L7ESoxLtwG+fv2w9gMU10x4xcT5A/d10ZFwjWysVP/
7       pWlaingQXfzNW+oHLfscZk0EcW3xFJUHsu2GT0UgfJXx30lmqcSzeL1NAJ/9jbf/tbBosWpUHN70xuTFD5LQ6xsP",
8       "body": "This is my message...",
9       "attributes": {
10        "ApproximateReceiveCount": "1",
11        "SentTimestamp": "1630414343808",
12        "SenderId": "AROAVRSV7HNNJTVJNL3VZ:pviillard@cloudera.com",
13        "ApproximateFirstReceiveTimestamp": "1630414343814"
14      },
15      "messageAttributes": {
16        "myKeyExample": {
17          "stringValue": "myValueExample",
18          "stringListValues": [],
19          "binaryListValues": [],
20          "dataType": "String"
21        }
22      },
23      "md5OfMessageAttributes": "f170220ddb772bb53d3e06f3fb2e0529",
24      "md5OfBody": "fc0ef998398157d220c38142ae53b795",
25      "eventSource": "aws:sqs",
26      "eventSourceARN": "arn:aws:sqs:us-east-2:381358652250:naaf",
27      "awsRegion": "us-east-2"
28    }
29  ]
30 }


```

AWS MSK

When using the AWS MSK trigger with a particular Kafka cluster and topic, the configuration may look like this:

Add trigger

Trigger configuration

 **MSK**
aws cluster

MSK cluster

Select an MSK cluster.

naaf

Batch size

The maximum number of messages to retrieve in a single batch.

100

Topic name

Enter the name of a Kafka topic to consume.

naaf

Starting position - *optional*

Enter a start position to begin reading the stream.

Latest

Secret key

Choose the secret key used for SASL/SCRAM authentication of the brokers in your MSK cluster.

This field is required if your Kafka brokers are accessed over the Internet.

In order to read from the MSK trigger, your execution role must have proper permissions.

☒ **Enable trigger**

Enable the trigger now, or create it in a disabled state for testing (recommended).

Cancel

Add

The FlowFile's content would look like this:

```

1 {
2   "eventSource": "aws:kafka",
3   "eventSourceArn": "arn:aws:kafka:us-east-2:381358652250:cluster/naaf-msk-cluster/d2a665a0-84a2-473d-bb2d-8853c532cd54-7",
4   "bootstrapServers": "b-2.naaf-msk-cluster.vnvbjv.c7.kafka.us-east-2.amazonaws.com:9094,b-1.naaf-msk-cluster.vnvbjv.c7.kafka.us-east-2.amazonaws.com:9094",
5   "records": {
6     "naaf-0": [
7       {
8         "topic": "naaf",
9         "partition": 0,
10        "offset": 0,
11        "timestamp": 1630434537574,
12        "timestampType": "CREATE_TIME",
13        "value": "VGhpcyBpcyBteSBtZXNzYmWdLICE=",
14        "headers": []
15      }
16    ]
17  }
18 }

```

Note that the data associated with a particular record may be base 64 encoded (the value field in the above screenshot). In this case, you may want to consider the Base64EncodeContent processor in your flow definition to decode the content.

Creating a Lambda function using CLI

Follow these steps to deploy a Function using the AWS CLI.

Before you begin

- Your Apache NiFi flow definition is stored in the Catalog and you have the CRN corresponding to the flow version you want to execute as a function
- You have created a Machine User with the proper role and you have its Access Key and Private Key credentials
- You have installed and configured the AWS CLI on your local machine

Procedure

1. Create a role for your Lambda.

```
aws iam create-role --role-name my_function_role --assume-role-policy-document '{ "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Principal": { "Service": "lambda.amazonaws.com" }, "Action": "sts:AssumeRole" } ] }'
```

2. Attach the basic AWS Lambda policy to your role.

```
aws iam attach-role-policy --role-name my_function_role --policy-arn arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole
```

3. Copy the following file as a local file named my-function-definition.json.

```

{
  "FunctionName": "first-cli-function",
  "Description": "my first cli function",
  "Timeout": 300,
  "MemorySize": 1024,
  "Environment": {
    "Variables": {
      "DF_ACCESS_KEY": "00000000-0000-0000-0000-000000000000",
      "DF_PRIVATE_KEY": "0000000000000000000000000000000000000000000000000000000000000000",
      "FLOW_CRN": "crn:cdp:df:us-west-1:00000000-0000-0000-0000-000000000000:flow:my-flow/v.1"
    }
  },
  "Tags": {
    "tagKey": "tagValue"
  }
}

```

```

    },

    "EphemeralStorage": {
      "Size": 2048
    },
    "Code": {
      "S3Bucket": "my-bucket",
      "S3Key": "naaf-aws-lambda-1.0.0-bin.zip"
    },
    "Role": "arn:aws:iam::327162129569:role/service-role/My-Role",

    "Runtime": "java11",
    "Handler": "com.cloudera.naaf.aws.lambda.StatelessNiFiFunctionHandler
::handleRequest",
    "Publish": false,
    "PackageType": "Zip",
    "Architectures": [
      "x86_64"
    ]
  }
}

```

**Note:**

Executing `aws lambda create-function --generate-cli-skeleton` gives you additional information that you can add in this JSON payload if you want to further customize your deployment.

4. Update the three sections of the JSON file.

The JSON file is generally structured into three sections.

- a) The first section is expected to be updated with each function. It includes Function Name, Description, Timeout, Memory Size, Tags, and Environment Variables.

The Environment Variables must include the following 3 environment variables:

- `DF_ACCESS_KEY`: The Access Key
- `DF_PRIVATE_KEY`: The Private Key
- `FLOW_CRN`: The CRN of the specific version of the in the Catalog. This always ends with `/v.<number>`.

Additionally, you can use environment variables to specify parameters for the dataflow.

- b) The second section of the JSON file contains fields that will likely be updated once when customizing the template for your particular organization. You should add the ARN of the role you created during the first step.
- c) The third section of the JSON file contains fields that will likely remain the same.

5. Create the Lambda function.

```
aws lambda create-function --cli-input-json file://my-function-definition.json
```

The function is now created. You can check it on the AWS Lambda UI and test the configuration.

6. Use the AWS Lambda UI or CLI to publish a version of the function.

```
aws lambda publish-version --function-name <function name or function ARN>
```

7. Use the AWS Lambda UI or CLI to specify the appropriate trigger for your published version.