PRISM CODE

PRISMCODE INFO SOLUTIONS PVT LTD

Amazon Lambda

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. AWS Lambda executes your code only when needed and scales automatically, from a few requests per day to thousands per second. You pay only for the compute time you consume - there is no charge when your code is not running. With AWS Lambda, you can run code for virtually any type of application or backend service - all with zero administration. AWS Lambda runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring and logging. All you need to do is supply your code in one of the languages that AWS Lambda supports.

You can use AWS Lambda to run your code in response to events, such as changes to data in an Amazon S3 bucket or an Amazon DynamoDB table; to run your code in response to HTTP requests using Amazon API Gateway; or invoke your code using API calls made using AWS SDKs. With these capabilities, you can use Lambda to easily build data processing triggers for AWS services like Amazon S3 and Amazon DynamoDB, process streaming data stored in Kinesis, or create your own back end that operates at AWS scale, performance, and security.

When Should I Use AWS Lambda?

AWS Lambda is an ideal compute platform for many application scenarios, provided that you can write your application code in languages supported by AWS Lambda, and run within the AWS Lambda standard runtime environment and resources provided by Lambda.

When using AWS Lambda, you are responsible only for your code. AWS Lambda manages the compute fleet that offers a balance of memory, CPU, network, and other resources. This is in exchange for flexibility, which means you cannot log in to compute instances, or customize the operating system on provided runtimes. These constraints enable AWS Lambda to perform operational and administrative activities on your behalf, including provisioning capacity, monitoring fleet health, applying security patches, deploying your code, and monitoring and logging your Lambda functions.

Creating a Lambda Function

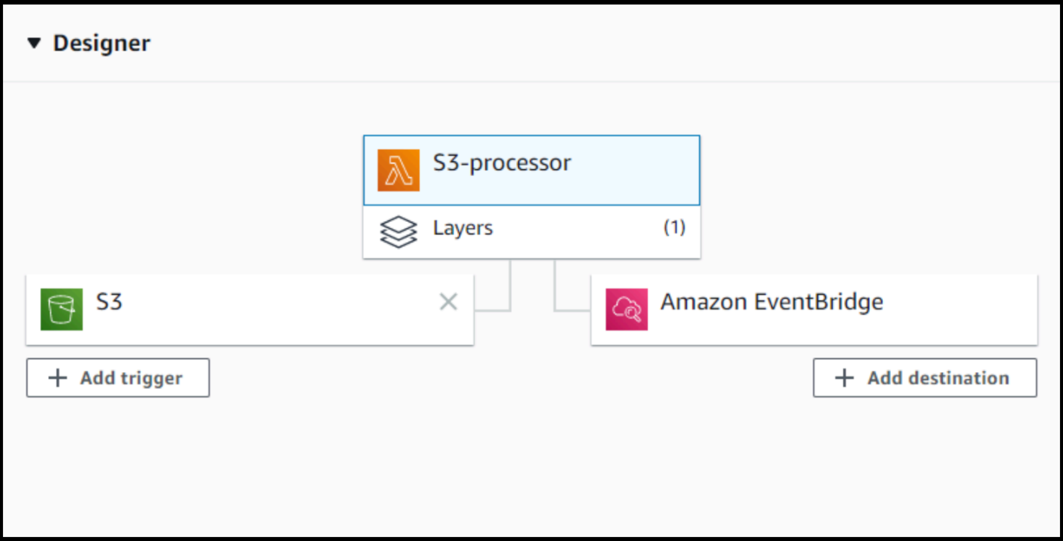
To create a Lambda function

- Open the AWS Lambda console.
- Choose Create a function.
- For Function name, enter **my-function**.
- Choose Create function.

Lambda creates a Node.js function and an execution role that grants the function permission to upload logs. Lambda assumes the execution role when you invoke your function, and uses it to create credentials for the AWS SDK and to read data from event sources.

Using the Designer

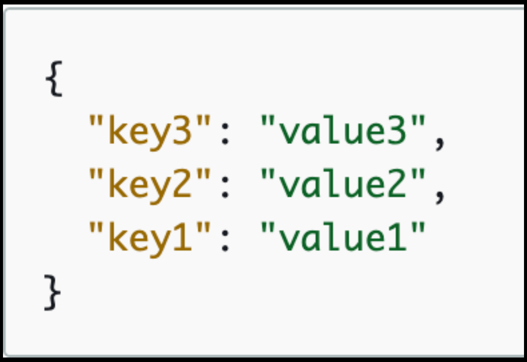
The **Designer** shows an overview of your function and its upstream and downstream resources. You can use it to configure triggers, layers, and destinations.



Choose my-function in the designer to return to the function's code and configuration. For scripting languages, Lambda includes sample code that returns a success response. You can edit your function code with the embedded AWS Cloud9 editor as long as your source code doesn't exceed the 3 MB limit.

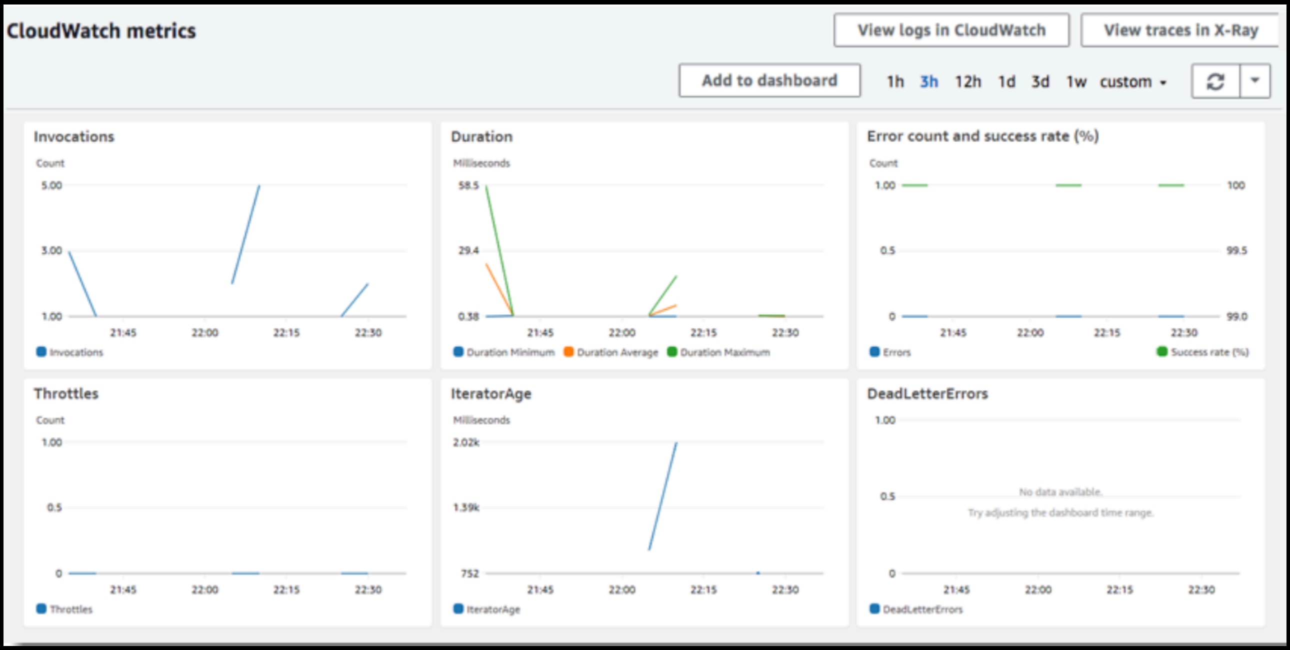
Invoking a Lambda Function

1. In the upper right corner, choose Test.
2. In the Configure test event page, choose Create new test event and in Event template, leave the default Hello World option. Enter an Event name and note the following sample event template:



```json
{
  "key3": "value3",
  "key2": "value2",
  "key1": "value1"
}
```

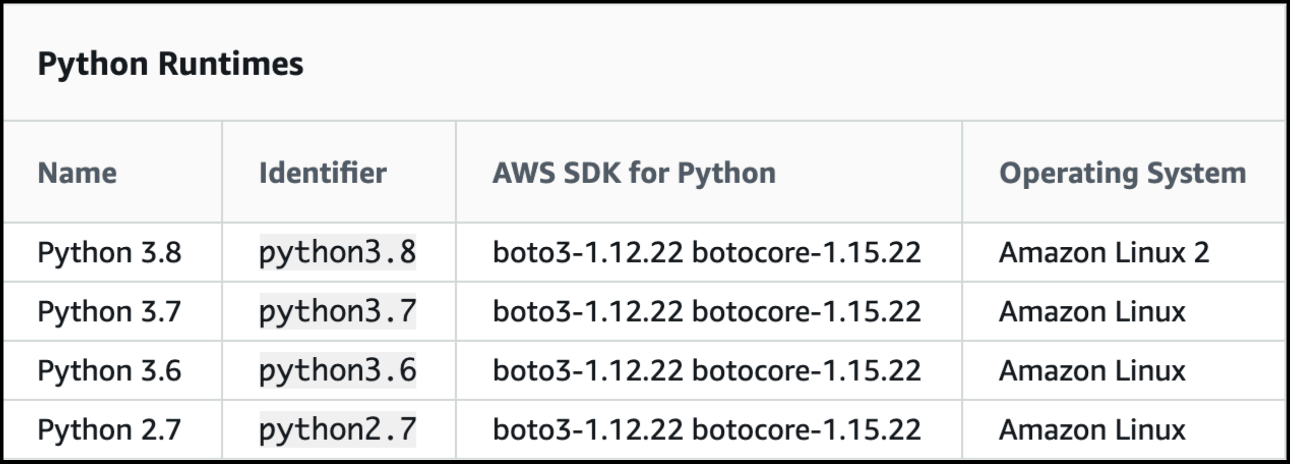
3. You can change key and values in the sample JSON, but don't change the event structure. If you do change any keys and values, you must update the sample code accordingly.
4. Choose Create and then choose Test. Each user can create up to 10 test events per function. Those test events are not available to other users.
5. AWS Lambda executes your function on your behalf. The handler in your Lambda function receives and then processes the sample event.
6. Upon successful execution, view results in the console.
7. The Execution result section shows the execution status as succeeded and also shows the function execution results, returned by the return statement.
8. The Summary section shows the key information reported in the Log output section (the REPORT line in the execution log).
9. The Log output section shows the log AWS Lambda generates for each execution. These are the logs written to CloudWatch by the Lambda function. The AWS Lambda console shows these logs for your convenience.
10. Note that the Click here link shows logs in the CloudWatch console. The function then adds logs to Amazon CloudWatch in the log group that corresponds to the Lambda function.
11. Run the Lambda function a few times to gather some metrics that you can view in the next step.
12. Choose Monitoring. This page shows graphs for the metrics that Lambda sends to CloudWatch.



Building Lambda Functions with Python

You can run Python code in AWS Lambda. Lambda provides runtimes for Python that execute your code to process events. Your code runs in an environment that includes the SDK for Python (Boto 3), with credentials from an AWS Identity and Access Management (IAM) role that you manage.

Lambda supports the following Python runtimes.



Python Runtimes

Name	Identifier	AWS SDK for Python	Operating System
Python 3.8	`python3.8`	boto3-1.12.22 botocore-1.15.22	Amazon Linux 2
Python 3.7	`python3.7`	boto3-1.12.22 botocore-1.15.22	Amazon Linux
Python 3.6	`python3.6`	boto3-1.12.22 botocore-1.15.22	Amazon Linux
Python 2.7	`python2.7`	boto3-1.12.22 botocore-1.15.22	Amazon Linux

Lambda functions use an execution role to get permission to write logs to Amazon CloudWatch Logs, and to access other services and resources. If you don't already have an execution role for function development, create one.

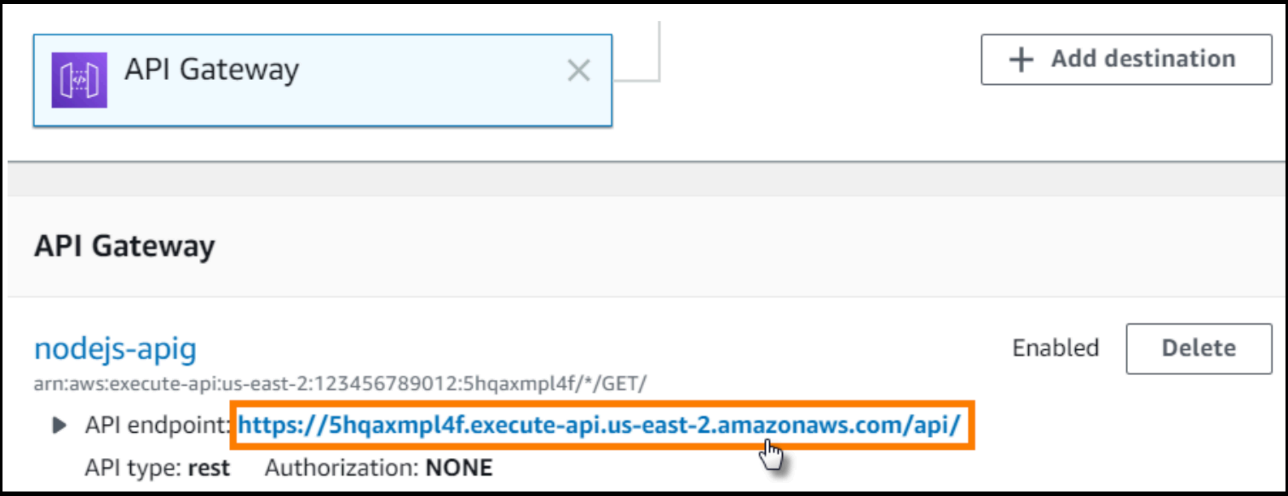
Using AWS Lambda with Amazon API Gateway

You can create a web API with an HTTP endpoint for your Lambda function by using Amazon API Gateway. API Gateway provides tools for creating and documenting web APIs that route HTTP requests to Lambda functions. You can secure access to your API with authentication and authorization controls. Your APIs can serve traffic over the internet or can be accessible only within your VPC.

To add a public endpoint to your Lambda function

1. Open the Lambda console **Functions page.**
2. Choose a function.
3. Under Designer, choose Add trigger.
4. Choose **API Gateway.**
5. For API, choose Create an API.
6. For **Security**, choose **Open.**
7. Choose **Add.**

With the **API Gateway** trigger selected in the designer, choose the endpoint to invoke the function with API Gateway.



arn:aws:execute-api:us-east-2:123456789012:5hqaxmpl4f/*/GET/
API endpoint: https://5hqaxmpl4f.execute-api.us-east-2.amazonaws.com/api/
API type: rest Authorization: NONE

API Gateway APIs are comprised of stages, resources, methods, and integrations. The stage and resource determine the path of the endpoint:
API Path Format

- **/prod/ –** The prod stage and root resource.
- **/prod/user –** The prod stage and user resource.
- **/dev/{proxy+} –** Any route in the dev stage.
- **/ –** (HTTP APIs) The default stage and root resource.

A Lambda integration maps a path and HTTP method combination to a Lambda function. You can configure API Gateway to pass the body of the HTTP request as-is (custom integration), or to encapsulate the request body in a document that includes all of the request information including headers, resource, path, and method.

Amazon API Gateway invokes your function **synchronously** with an event that contains a JSON representation of the HTTP request. For a custom integration, the event is the body of the request. For a proxy integration, the event has a defined structure.

API Gateway waits for a response from your function and relays the result to the caller. For a custom integration, you define an integration response and a method response to convert the output from the function to an HTTP response. For a proxy integration, the function must respond with a representation of the response in a specific format.

The Lambda runtime serializes the response object into JSON and sends it to the API. The API parses the response and uses it to create an HTTP response, which it then sends to the client that made the original request.

Resources in your API define one or more methods, such as GET or POST. Methods have an integration that routes requests to a Lambda function or another integration type. You can define each resource and method individually, or use special resource and method types to match all requests that fit a pattern. A proxy resource catches all paths beneath a resource. The **ANY** method catches all HTTP methods.