Immersion Day

Getting Started with AWS Lambda

August 2016

Rev 2016-08-19

Table of Contents

Overview

This lab provides an introduction to AWS Lambda, and makes use of Amazon CloudWatch and Amazon S3.

AWS Lambda

AWS Lambda is a serverless compute service that runs your code in response to events and automatically manages the underlying compute resources for you. You can use AWS Lambda to extend other AWS services with custom logic, or create your own back-end services that operate at AWS scale, performance, and security. AWS Lambda can automatically run code in response to multiple events, such as modifications to objects in Amazon S3 buckets or table updates in Amazon DynamoDB.

Lambda runs your code on high-availability compute infrastructure and performs all the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code and security path deployment, and code monitoring and logging. All you need to do is supply the code.

Amazon S3

Amazon Simple Storage Service (Amazon S3), provides developers and IT teams with secure, durable, highly-scalable object storage. Amazon S3 is easy to use, with a simple web services interface to store and retrieve any amount of data from anywhere on the web. With Amazon S3, you only pay for the storage you actually use. There is no minimum fee and no setup cost.

Amazon S3 can be used alone or together with other AWS services such as Amazon Elastic Compute Cloud (Amazon EC2), Amazon Elastic Block Store (Amazon EBS), and Amazon Glacier, as well as third party storage repositories and gateways. Amazon S3 provides cost-effective storage for a wide variety of use cases including cloud applications, content distributions, backup and archiving, disaster recovery and big data analytics.

Amazon CloudWatch

Amazon CloudWatch is a monitoring service for AWS cloud resources and the applications you run on AWS. You can use Amazon CloudWatch to collect and track metrics, collect and monitor log files and set alarms. Amazon CloudWatch can monitor AWS resources such as Amazon EC2 instances as well as custom metrics generated by your applications and services and any log files your applications generate. You can use Amazon CloudWatch to gain system-wide visibility into resource utilization, application performance and operational health. You can use these insights to react and keep your application running smoothly.

Handling S3 Events using the AWS Lambda Console

In this section, you will create an Amazon S3 bucket, configure it as the Lambda event source, and create an Amazon S3 Lambda function.

Create a S3 Bucket

In this lab, we will use a S3 bucket and connect a Lambda function to react to events generated when objects are modified within it.

1. Login and access the AWS Management Console at:

   https://console.aws.amazon.com/console/home

2. Click **Services** at the top of the console and then click **S3**, located under the **Storage & Content Delivery** section.

3. Click the blue **Create Bucket** button.

4. In the **Create a Bucket –Select a Bucket Name and Region** page, enter the following:

   a. **Bucket Name**: Choose a unique name for your bucket (e.g., *lambda-bkt-{yourname}*). Your bucket name must be unique across all regions and be all lowercase.
   b. **Region**: Set the Region that is appropriate to your location. US Standard is a good choice in the absence of a compelling reason to choose another region.

5. Select Create.

Create an S3 Lambda Function

1. On the AWS Management Console, select **Service**, and then select **Lambda**, located under the **Compute** section.

2. In the AWS Lambda Console, select **Get Started Now**.

   Note: The console shows the Get Started Now page only if you do not have any Lambda functions created. If you have functions already created within your account, you will see the Lambda > Functions page. On the list page, choose Create a Lambda function to go to the Lambda > New function page.)

3. In the **Lambda > New Function** page, enter the details below:
    a. **Step 1: Select blueprint**
    Under the **Select blueprint** section, use the filter or page forward to select the **s3-get-object** blueprint. This is nodejs blueprint that is written in JavaScript and there is a similar blueprint, **s3-get-object-python** that uses Python.

    b. **Step 2: Configure triggers**
    Select the bucket you just created in the **Bucket** field.
    Set **Event Type** to **Object Created (All)**.
    Leave the **Prefix** and **Suffix** fields empty. These are used to filter the objects that will invoke this Lambda function.
    Click the **Enable Trigger** checkbox and click **Next**.

    c. **Step 3: Configure function**
    In the **Configure function** section, in the **Name** field, provide a unique name for your function (e.g., *S3Function-{yourname}*).
    The **Description** field allows you provide a description that will be shown in the console to explain the function's purpose. The default value can be left as is.
    The **Runtime** field specifies the engine that will be used to run the code you provide.
    Leave the **Code Entry Type** field at **Edit code incline**. The other options are to upload a zip file or specify a file in S3 that contains your code.
    Here you can view the sample code in the inline code editor. The code can be edited here if required, but no changes are necessary for this lab.
    In the **Lambda function handler and role** section, leave **Handler** as **index.handler**. This specifies which function should be invoked when the function is executed.
    Leave **Role** as **Create new role from template(s)** and provide a unique **Role name** (e.g., *lambdaexecution-{yourname}*).
    In **Policy templates**, leave the default **S3 object read-only permissions**. This section creates a Role for the Lambda function to run within, and grants permissions to allow the function to access your S3 bucket.
    In the **Advanced settings** section, leave the **Memory (MB)** value at 128, and increase the **Timeout** to 5 seconds.
    Leave the **VPC** section as **No VPC**. This setting allows you to run your Lambda function to access resources that are only accessible within a private network segment.

    d. **Step 4: Review**
    Review the function details and then click **Create Function**.

Upload a file to an Amazon S3 bucket to trigger a Lambda event

Now that you have connected your Lambda function to an Amazon S3 bucket event, you can upload a file to trigger a call to the Lambda function.

1. Return to the S3 Management Console, by selecting **Service** and then **S3.**

2. Select the bucket you created earlier.

3. Click the blue **Upload** button.

4. In the **Upload – Select Files and Folders** page, select **Add Files** and select any file from your workstation that you want to be uploaded.

   Note: For the purposes of this lab, ensure that the name of the file you upload contains only alphanumeric characters and no spaces. (e.g. myimage.png)

5. Select **Start Upload**.

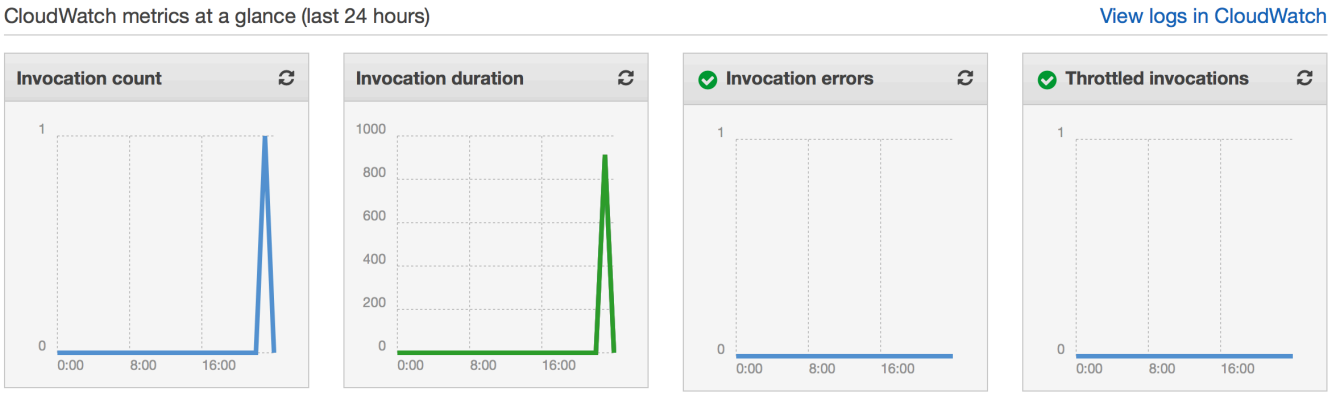
Monitor Lambda S3 functions through Amazon CloudWatch

Now that you have uploaded a file that triggered your AWS Lambda function to run, you can use Amazon CloudWatch to view the monitoring data for your AWS Lambda function.

1. Return to the Lambda console by selecting **Services**, and then click **Lambda.**

2. In the **Lambda > Functions** page, under **Function name**, select your Lambda Function

3. Select the **Monitoring** tab. There are four graphs that can be viewed: **Invocation count**, **Invocation duration**, **Invocation errors** and **Throttled invocations**.



a. **Invocation count** measures the number of times a function is invoked in response to an event or invocation API call. This includes successful and failed invocations, but does not include throttled attempts. This equals the billed requests for the function. AWS Lambda only sends these metrics to CloudWatch if they have a non-zero value.

b. **Invocation duration** measures the elapsed wall clock time from when the function code starts executing as a result of an invocation to when it stops executing. The maximum data point value possible is the function timeout configuration value. The billed duration will be rounded up to the nearest 100 milliseconds. AWS Lambda only sends these metrics to CloudWatch if they have a non-zero value.

c. **Invocation errors** measures the number of invocations that failed due to errors in the function. Failed invocations may trigger a retry attempt that succeeds which includes:
    i. Handling exceptions (e.g., `context.fail(error)`)
    ii. Unhandled exceptions causing the code to exit
    iii. Out of memory exceptions
    iv. Timeouts
    v. Permissions errors

    This does not include invocations that fail due to rates exceeding the default concurrent limits (error code 429) or failures due to internal service errors (error code 500) invocation.

d. **Throttled invocations** measures the number of Lambda function invocation attempts that were throttled due to invocation rates exceeding the customers concurrent limits (error code 429). Failed invocations may trigger a retry attempt that succeeds.

4. Click **View logs in CloudWatch** above the graphs. This takes you into the CloudWatch Management Console.

5. This will display all of the Log Groups available for your Lambda function. Select the first Log Stream available to view the logs generated by your most recently invoked Lambda function.

   The Event Data contains information about the function which includes the Request Id, the duration it took in milliseconds, the billed duration (rounded to the nearest 100ms),

the Memory Size of the Function and the Maximum Memory that the function used. The log data may take a minute to appear.
The sample function itself fetches the file from S3 and logs what type of data the file contains. This function could easily be modified to resize images or any other application specific logic.

