AWS Lambda is an [event-driven](https://en.wikipedia.org/wiki/Event-driven_programming), [serverless computing](https://en.wikipedia.org/wiki/Serverless_computing) platform provided by [Amazon](https://en.wikipedia.org/wiki/Amazon.com) as a part of the [Amazon Web Services](https://en.wikipedia.org/wiki/Amazon_Web_Services). It is a computing service that runs code in response to [events](https://en.wikipedia.org/wiki/Event_(computing)) and automatically manages the computing resources required by that code.

AWS Lambda lets you run code without provisioning or managing servers. You pay only for the compute time you consume - there is no charge when your code is not running.

With Lambda, you can run code for virtually any type of application or backend service - all with zero administration. Just upload your code and Lambda takes care of everything required to run and scale your code with high availability. You can set up your code to automatically trigger from other AWS services or call it directly from any web or mobile app.

Benefits

NO SERVERS TO MANAGE

AWS Lambda automatically runs your code without requiring you to provision or manage servers. Just write the code and upload it to Lambda.

CONTINUOUS SCALING

AWS Lambda automatically scales your application by running code in response to each trigger. Your code runs in parallel and processes each trigger individually, scaling precisely with the size of the workload.

SUBSECOND METERING

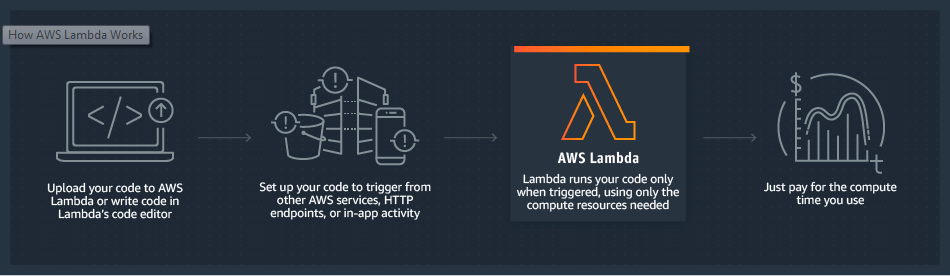
With AWS Lambda, you are charged for every 100ms your code executes and the number of times your code is triggered. You don't pay anything when your code isn't running.

You can now configure your **AWS Lambda functions to run up to 15 minutes per execution.** Previously, the maximum execution time (timeout) for a Lambda function was 5 minutes. Now, it is easier than ever to perform big data analysis, bulk data transformation, batch event processing, and statistical computations using longer running functions.

With AWS Lambda, you pay only for what you use. You are charged based on the number of requests for your functions and the duration, the time it takes for your code to execute.

<https://aws.amazon.com/lambda/pricing/>

**How AWS Lambda works**



AWS Lambda natively supports Java, Go, PowerShell, Node.js, C#, Python, and Ruby code, and provides a Runtime API which allows you to use any additional programming languages to author your functions

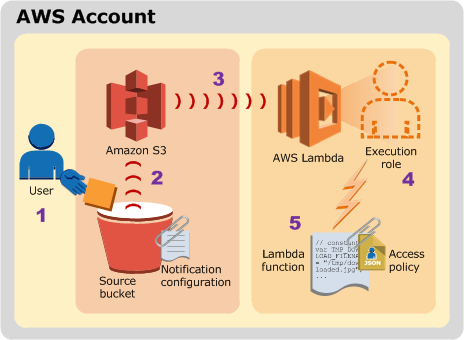
Supported Event Sources::

* [**Amazon S3**](https://docs.aws.amazon.com/lambda/latest/dg/invoking-lambda-function.html#supported-event-source-s3) **--- Will focus on this today**
* [Amazon DynamoDB](https://docs.aws.amazon.com/lambda/latest/dg/invoking-lambda-function.html#supported-event-source-dynamo-db)
* [Amazon Kinesis Data Streams](https://docs.aws.amazon.com/lambda/latest/dg/invoking-lambda-function.html#supported-event-source-kinesis-streams)
* [Amazon Simple Notification Service](https://docs.aws.amazon.com/lambda/latest/dg/invoking-lambda-function.html#supported-event-source-sns)
* [Amazon Simple Email Service](https://docs.aws.amazon.com/lambda/latest/dg/invoking-lambda-function.html#supported-event-source-ses)
* [Amazon Simple Queue Service](https://docs.aws.amazon.com/lambda/latest/dg/invoking-lambda-function.html#supported-event-source-sqs)
* [Amazon Cognito](https://docs.aws.amazon.com/lambda/latest/dg/invoking-lambda-function.html#supported-event-source-cognito)
* [AWS CloudFormation](https://docs.aws.amazon.com/lambda/latest/dg/invoking-lambda-function.html#supported-event-source-cloudformation)
* [Amazon CloudWatch Logs](https://docs.aws.amazon.com/lambda/latest/dg/invoking-lambda-function.html#supported-event-source-cloudwatch-logs)
* [Amazon CloudWatch Events](https://docs.aws.amazon.com/lambda/latest/dg/invoking-lambda-function.html#supported-event-source-cloudwatch-events)
* Many more..

The following example illustrates how this model works.

**Example – Amazon S3 Pushes Events and Invokes a Lambda Function**

Suppose that you want your AWS Lambda function invoked for each *object created* bucket event. You add the necessary event source mapping in the bucket notification configuration.



The diagram illustrates the flow:

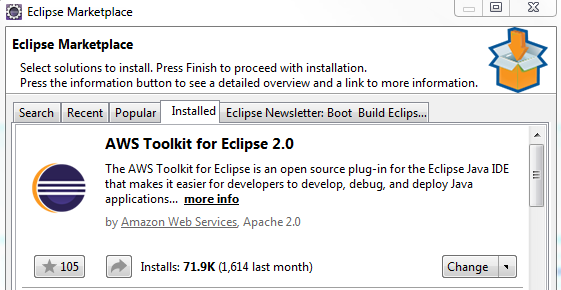
1. The user creates an object in a bucket.
2. Amazon S3 detects the object created event.
3. Amazon S3 invokes your Lambda function according to the event source mapping described in the bucket notification configuration.
4. AWS Lambda verifies the permissions policy attached to the Lambda function to ensure that Amazon S3 has the necessary permissions.
5. Once AWS Lambda verifies the attached permissions policy, it executes the Lambda function. Remember that your Lambda function receives the event as a parameter.

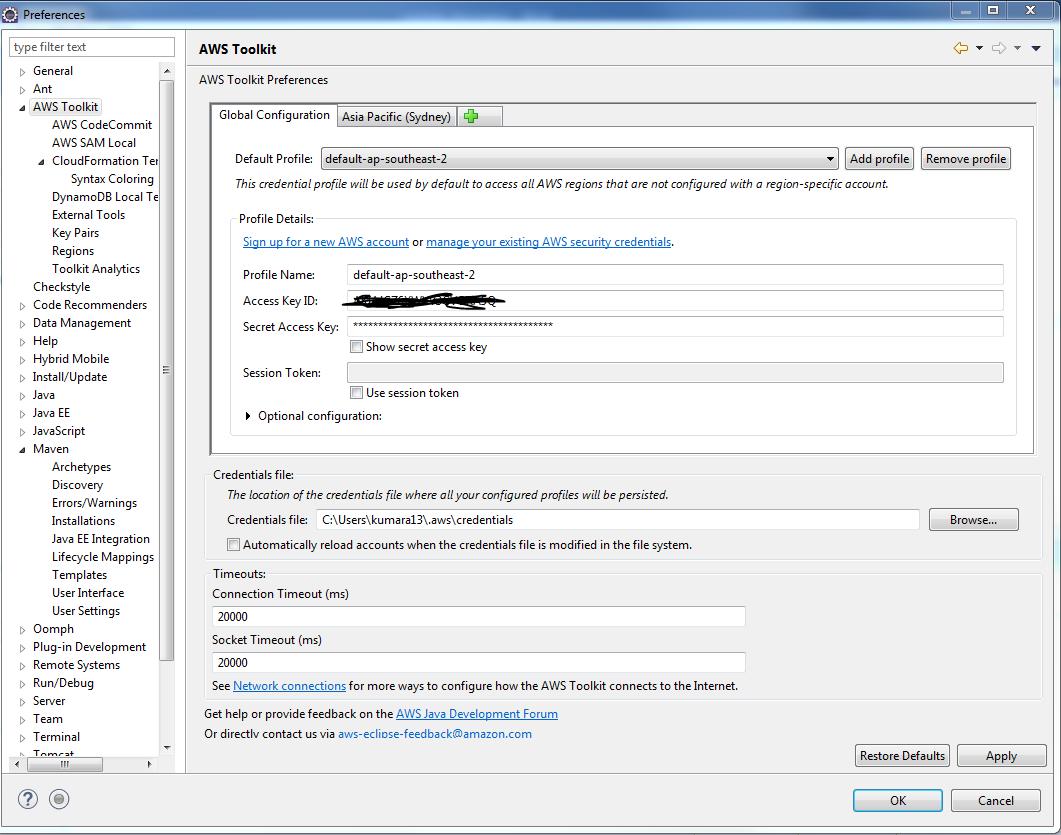
**Agenda for Today’s discussion**

1. How to integrate the AWS account with Eclipse using AWS Toolkit.
2. How to write, deploy and trigger the Lambda function in java.
3. How to use the local environment variables in the Lambda function.
4. How to use the Configurable parameters in AWS Parameter store and use in the Lambda function.
5. How to build the gitlab pipeline for the lambda function.
6. How to write the logs in the Cloudwatch and check into the logDNA

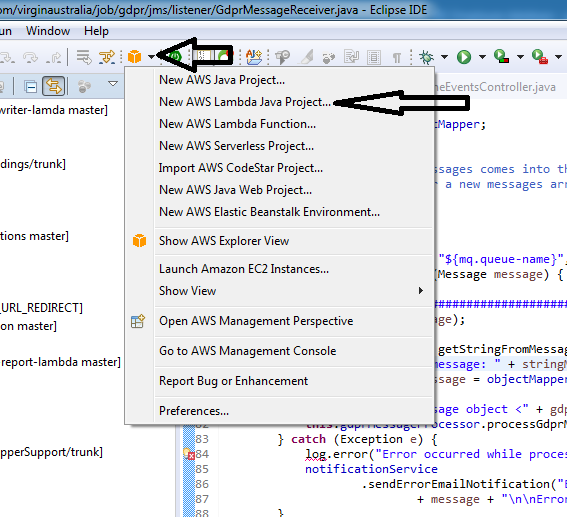
AWS plugin for eclipse can be found here

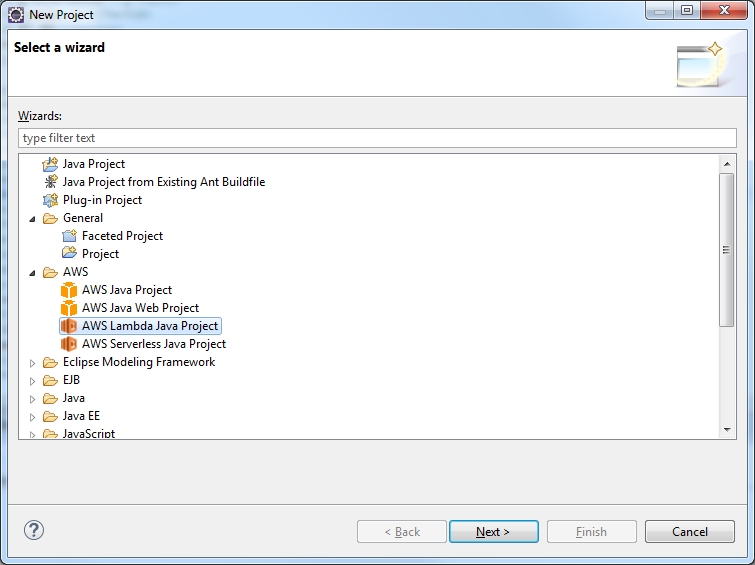
<https://docs.aws.amazon.com/toolkit-for-eclipse/v1/user-guide/setup-install.html>



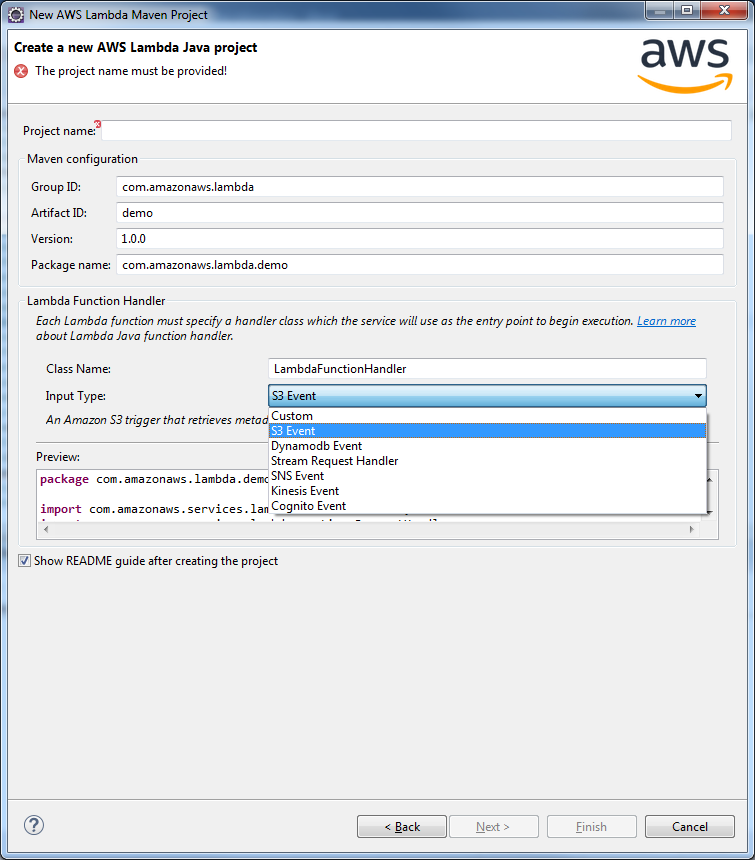


After it’s installed you can create a new lambda project by going here





Configure the Lambda handler for the specific event for which you wanted to setup the trigger.



Sample JAVA CODE

**package** com.amazonaws.lambda.demo;

**import** com.amazonaws.services.lambda.runtime.Context;

**import** com.amazonaws.services.lambda.runtime.RequestHandler;

**import** com.amazonaws.services.lambda.runtime.events.S3Event;

**import** com.amazonaws.services.s3.AmazonS3;

**import** com.amazonaws.services.s3.AmazonS3ClientBuilder;

**import** com.amazonaws.services.s3.model.GetObjectRequest;

**import** com.amazonaws.services.s3.model.S3Object;

**public** **class** LambdaFunctionHandler **implements** RequestHandler<S3Event, String> {

**private** AmazonS3 s3 = AmazonS3ClientBuilder.standard().build();

**public** LambdaFunctionHandler() {}

// Test purpose only.

LambdaFunctionHandler(AmazonS3 s3) {

**this**.s3 = s3;

}

@Override

**public** String handleRequest(S3Event event, Context context) {

context.getLogger().log("Received event: " + event);

// Get the object from the event and show its content type

String bucket = event.getRecords().get(0).getS3().getBucket().getName();

String key = event.getRecords().get(0).getS3().getObject().getKey();

**try** {

S3Object response = s3.getObject(**new** GetObjectRequest(bucket, key));

String contentType = response.getObjectMetadata().getContentType();

context.getLogger().log("CONTENT TYPE: " + contentType);

**return** contentType;

} **catch** (Exception e) {

e.printStackTrace();

context.getLogger().log(String.format(

"Error getting object %s from bucket %s. Make sure they exist and"

+ " your bucket is in the same region as this function.", key, bucket));

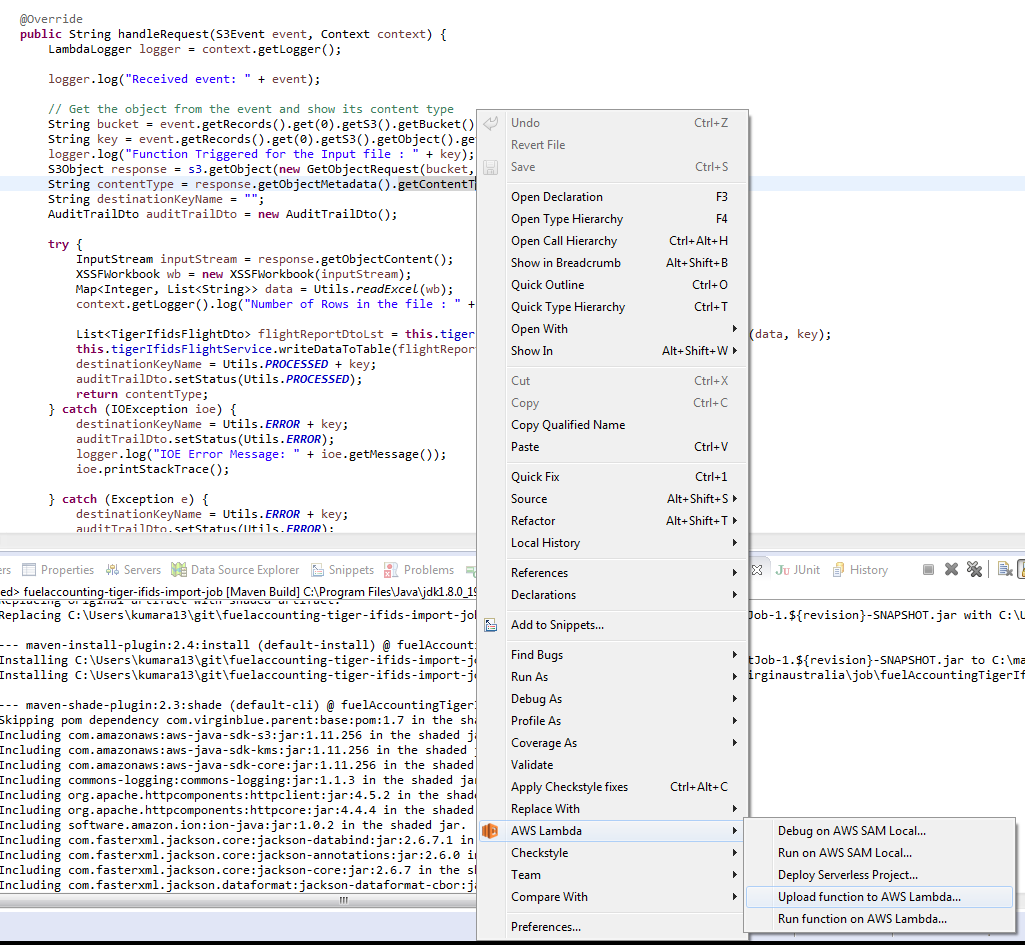
**throw** e;

}

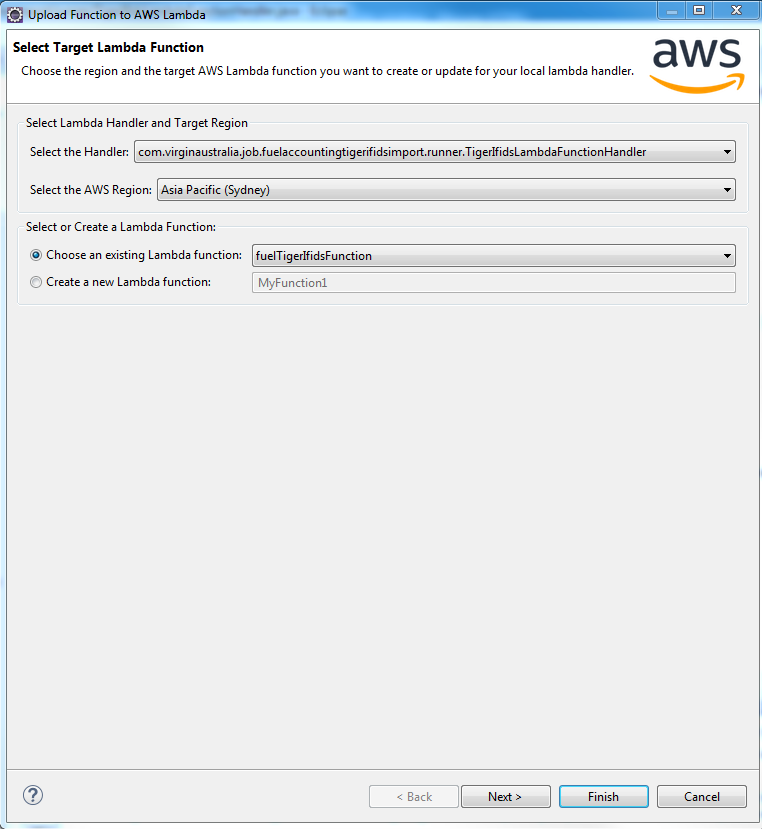
}

}

After implementing the logic into the AWS Lambda function, you can upload your Main handler directly through Eclipse

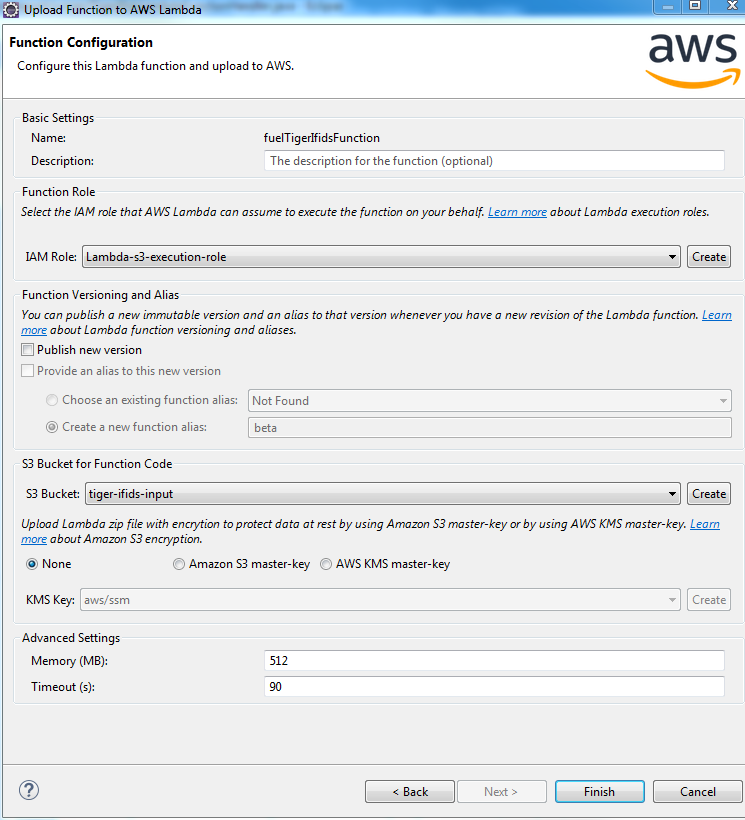


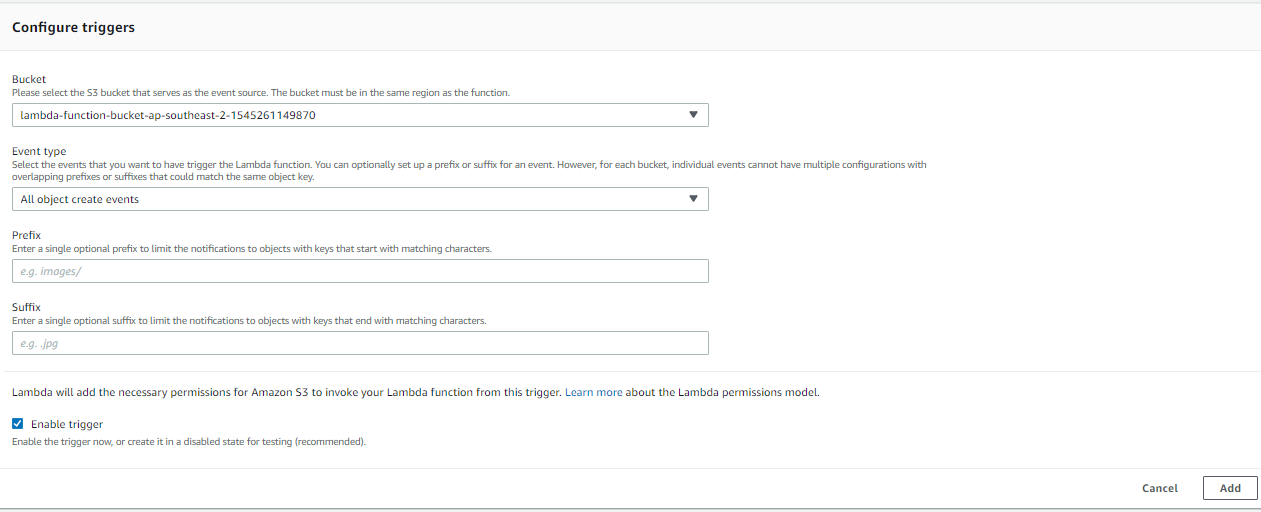
Create the new Lambda function or upload the code against the existing function setup earlier.

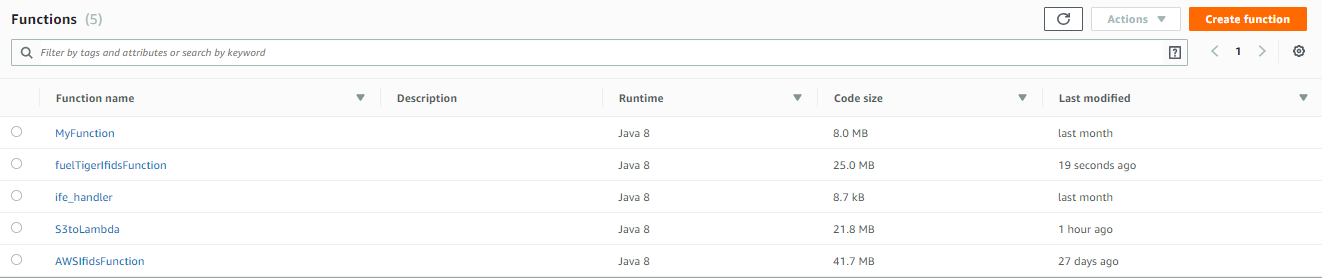


Select the IAM role which should have the permission stated below and also setup the S3 bucket which will hold the function code.

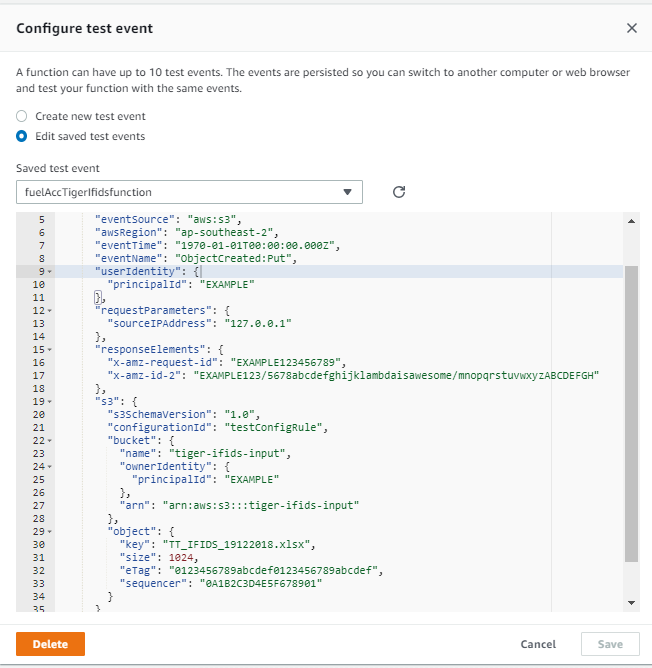








For executing the function you can either setup the trigger or configure the test events like this



GIT URL of the Fuel accounting project

<https://git.virginaustralia.com/itdev/jobs/fuelaccounting-tiger-ifids-import-job.git>

Log DNA URL

<https://va.logdna.com/f83bc64e9a/logs/view>