AWS Lambda is a compute service that lets us run code without provisioning or managing servers. AWS Lambda executes our code only when needed and scales automatically, from a few requests per day to thousands per second. We pay only for the compute time we consume, i.e. there is no charge when our code is not running. With AWS Lambda, we can run code for virtually any type of application or backend service - with zero administration. AWS Lambda runs our 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 we need to do is supply our code in one of the languages that AWS Lambda supports (as of now - Node.js, Java, C# and Python).

**When Should we Use AWS Lambda**?

AWS Lambda is an ideal compute platform for many application scenarios, provided that we can write our application code in languages supported by AWS Lambda (Node.js, Java, C# and Python) and run within the AWS Lambda standard runtime environment and resources provided by Lambda.

When using AWS Lambda, we are responsible only for our 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 we cannot login to compute instances or customize the operating system or language runtime. These constraints enable AWS Lambda to perform operational and administrative activities on our behalf, including provisioning capacity, monitoring fleet health, applying security patches, deploying our code, and monitoring and logging our Lambda functions.

If we need to manage our own compute resources, Amazon Web Services also offers other compute services to meet our needs.

Amazon Elastic Compute Cloud (Amazon EC2) service offers flexibility and a wide range of EC2 instance types to choose from. It gives us the option to customize operating systems, network and security settings, and the entire software stack, but we are responsible for provisioning capacity, monitoring fleet health and performance and using Availability Zones for fault tolerance.

Elastic Beanstalk offers an easy-to-use service for deploying and scaling applications onto Amazon EC2 in which we retain ownership and full control over the underlying EC2 instances.

**What is AWS Lambda?**

Amazon explains that AWS Lambda (λ) as a 'serverless' compute service, which implied the developers do not have to worry about which AWS resources to launch or how will they manage them, they just put the code on lambda and it runs, it is as simple as that. It helps us to focus on core-competency i.e. App Building or the code.

**Where will I use AWS Lambda?**

AWS Lambda executes our backend code by automatically managing the AWS resources. When we say ‘manage’, it includes launching or terminating instances, health checkups, auto scaling, updating or patching new updates etc.

**So, how does it work?**

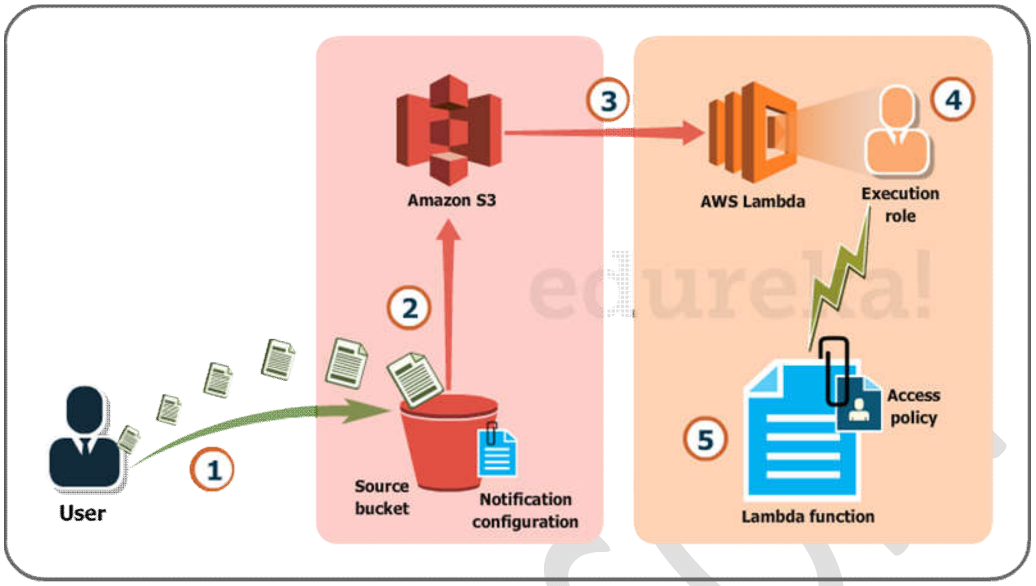
The code that we want Lambda to run is known as a Lambda function. Now, as we know a function runs only when it is called, isn’t it? Here, Event Source is the entity which triggers a Lambda Function and then the task is executed.

Let us take an example to understand it more clearly.
Suppose we have an app for image uploading. Now when we upload an image, there are a lot of tasks involved before storing it, such as resizing, applying filters, compression etc.

So, this task of uploading of an image can be defined as an Event Source or the ‘trigger’ that will call the Lambda Function and then all these tasks can be executed via the Lambda function.

In this example, a developer just has to define the event source and upload the code.

Let's understand this example with actual AWS resources.



Over here, we will be uploading images in the form of objects to an S3 bucket. This uploading an image to the S3 bucket will become an event source or the 'trigger'.

The whole process, as we can see in the diagram, is divided into 5 steps. Let us understand each one of them.

1. User uploads an image (object) to a source bucket in S3 which has notification attached to it for Lambda.
2. The notification is read by S3 and it decides where to send that notification.
3. S3 sends the notification to Lambda, this notification acts as an invoke call of the lambda function.
4. Execution role in Lambda can be defined by using IAM (Identity and Access Management) to give access permission for the AWS resources, for this example here it would be S3.
5. Finally, it invokes the desired lambda function which works on the object which has been uploaded to the S3 bucket.

If we were to solve this scenario traditionally, along with development, we would have hired people for managing the following tasks:

- Size, provision and scale up group of servers
- Managing OS updates
- Apply security patches and
- Monitor all this infrastructure for performance and availability.

This would have been an expensive, tedious and tiresome task, therefore the need for AWS Lambda is justified. AWS Lambda is compatible with Java, C#, Node.JS and Python, so we can upload our file in a zip, define an event source and we are set.

We now know – How Lambda works and what Lambda does.

Now, let us understand-

1) Where to use Lambda?
2) What purpose does Lambda serve, that other AWS Compute services don't?

If we were to architect a solution to a problem, we should be able to identify where to use Lambda.

So, as an architect we have the following options to execute a task:

- AWS EC2
- AWS Elastic Beanstalk
- AWS OpsWorks
- AWS Lambda

Let's take the above use case as an example and understand why we chose Lambda to solve it.

AWS OpsWorks and AWS Elastic Beanstalk are used to deploy an app, so our use case is not to create an app, but to execute a back-end code.

**Then why not EC2?**

If we were to use EC2, we would have to architect everything i.e. load balancer, EBS volumes, software stacks etc. In lambda we don't have to worry about anything, just insert our code and AWS will manage the rest.

For example, in EC2 we would be installing the software packages on our virtual machine which would support our code, but in Lambda we don't have to worry about any VM, just insert plain code and Lambda will execute it for us.

But, if our code will be running for hours and we expect a continuous stream of requests, we should probably go with EC2, because the architecture of Lambda is for a sporadic kind of workload, wherein there will be some quiet hours and some spikes in the number of requests as well.

For example, logging the email activity for say a small company, would see more activity during the day than in the night, also there could be days when there are less emails to be processed and sometimes the whole world could start emailing us. In both the cases, Lambda is at our service.

Considering this use case for a big social networking company, where the emails are never ending because it has a huge user base, Lambda may not be the apt choice.

**Limitations of AWS Lambda**

Some limitations are hardware specific and some are bound by the architecture.

Hardware limitations include the disk size, which is limited to 512 MB, the memory can vary between 128 MB and 1536 MB. Then there are some other such as the execution timeout can be maximized to just 5 minutes, our request body payload can be no more than 6 MB. The request body payload is like the data that you send with a “GET” or “PUT” request in HTTP, whereas the request body would be the type of request, the headers etc.

Actually, these are not limitations, but are the design boundaries which have been set in the architecture of Lambda so if our use case does not fit these, we can always have the other AWS compute services at our disposal.

Let us now cover the expense part as well.

Let us take the above use case as an example and understand why we chose Lambda to solve it.

AWS OpsWorks and AWS ElasticBeanstalk are used to deploy an app, so our use case is not to create an app, but to execute a back-end code.

**Pricing in AWS Lambda**

Like most of the AWS services, AWS Lambda is also a pay per use service, meaning we only pay what we use, therefore we are charged on the following parameters

1. The number of requests that we make to our lambda function
2. The duration for which our code executes.

**Requests**

We are charged for the number of requests that we make across all our lambda functions.
AWS Lambda counts a request each time it starts executing in response to an event source or invoke call, including test is invoked from the console.

Let us look at the prices:
First 1 million requests, every month are for free.
0.20$ per million requests thereafter.

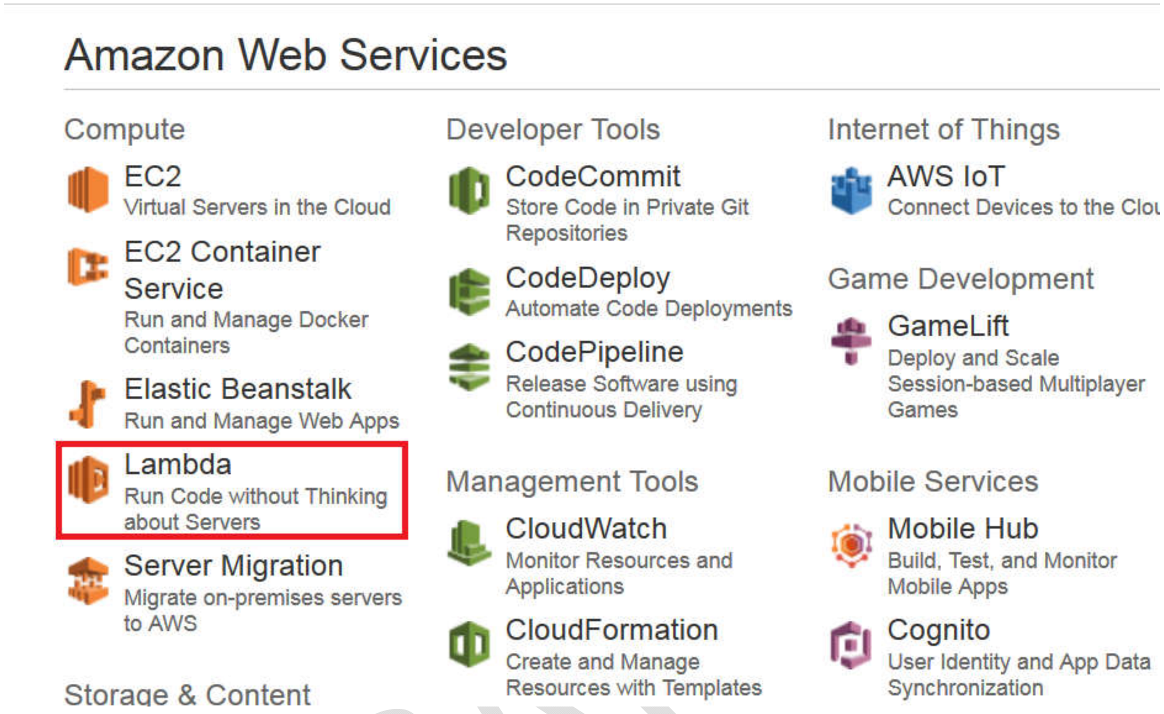
**Duration**

Duration is calculated from the moment our code starts executing till the moment it returns or terminates, it is rounded up to the nearest 100ms.
The price depends on the amount of memory we allocate to our function, we are charged $0.00001667 for every GB-second used.

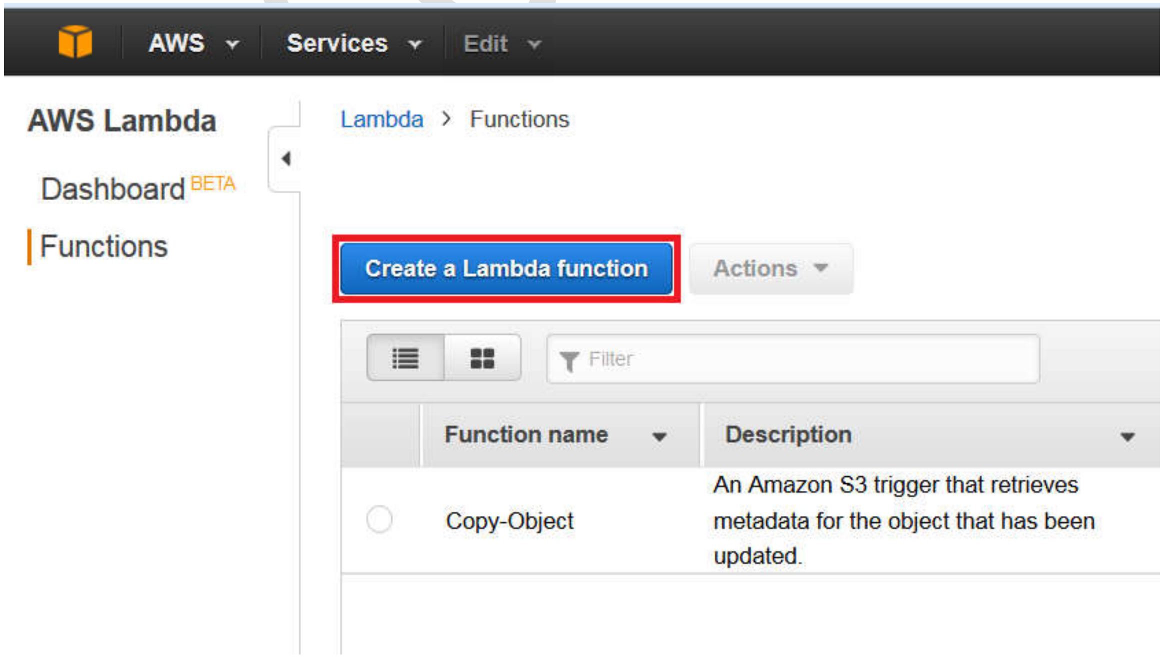
* Source: AWS official website

Let us create a Lambda function which will log “An object has been added” once we add an object to a specific bucket in S3.

**Step1:** From the AWS Management Console under compute section, select AWS Lambda.

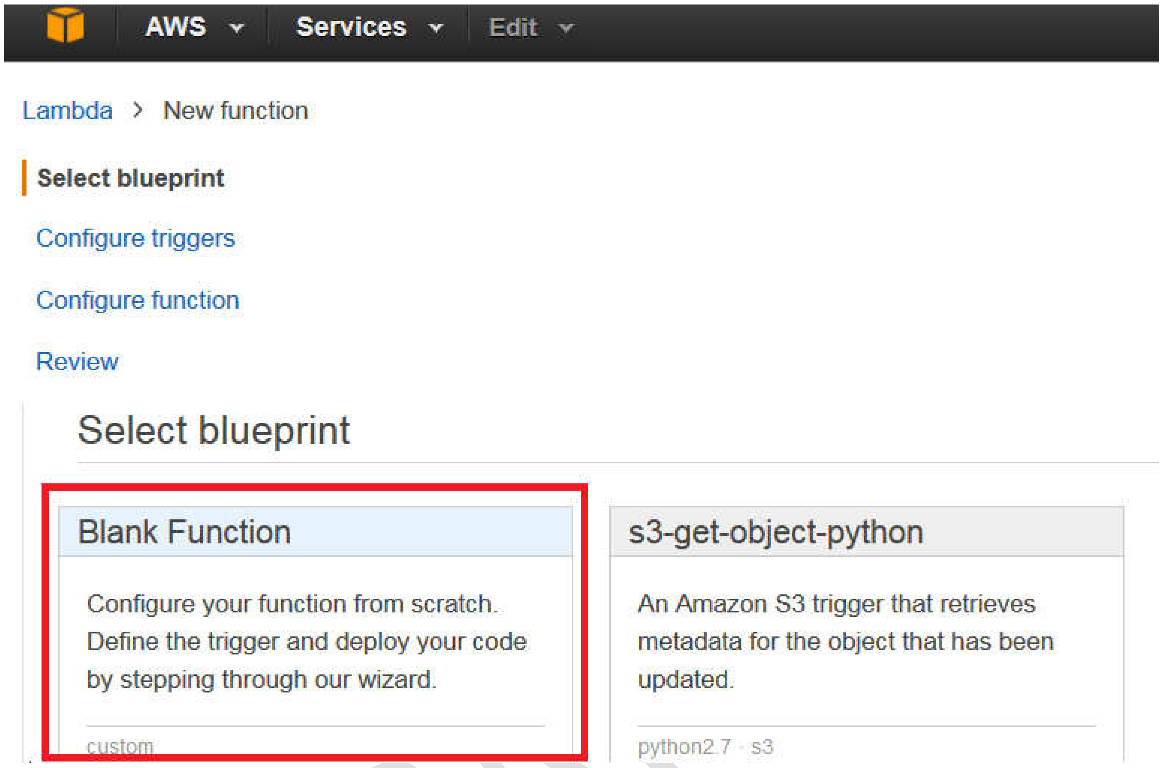

Amazon Web Services
Compute
EC2
Virtual Servers in the Cloud
EC2 Container Service
Run and Manage Docker Containers
Elastic Beanstalk
Run and Manage Web Apps
Lambda
Run Code without Thinking about Servers
Server Migration
Migrate on-premises servers to AWS
Storage & Content
Developer Tools
CodeCommit
Store Code in Private Git Repositories
CodeDeploy
Automate Code Deployments
CodePipeline
Release Software using Continuous Delivery
Management Tools
CloudWatch
Monitor Resources and Applications
CloudFormation
Create and Manage Resources with Templates
Internet of Things
AWS IoT
Connect Devices to the Clou
Game Development
GameLift
Deploy and Scale Session-based Multiplayer Games
Mobile Services
Mobile Hub
Build, Test, and Monitor Mobile Apps
Cognito
User Identity and App Data Synchronization

**Step2:** On the AWS Lambda Console, click on “Create a Lambda function”.



| | Function name | Description |
|---|---|---|
| ○ | Copy-Object | An Amazon S3 trigger that retrieves metadata for the object that has been updated. |

**Step3:** On the next page, we have to select a blueprint. For example, we will be selecting the blank function for our use-case.



Select blueprint

Blank Function

Configure your function from scratch. Define the trigger and deploy your code by stepping through our wizard.

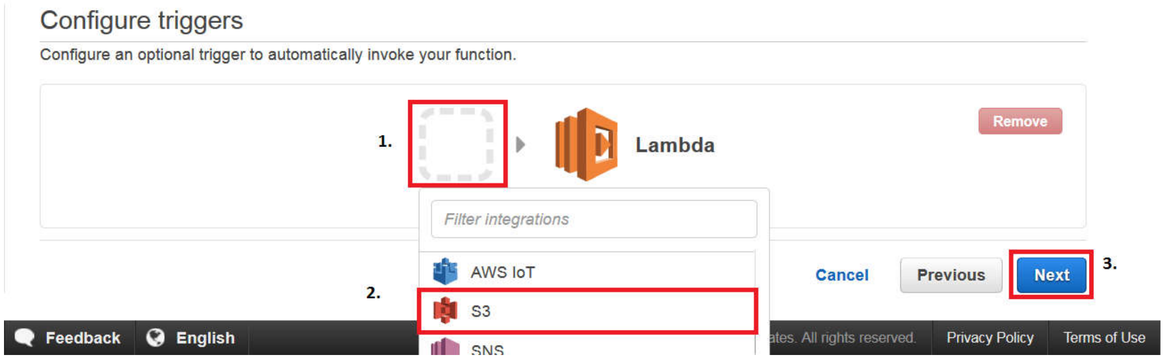
custom

s3-get-object-python

An Amazon S3 trigger that retrieves metadata for the object that has been updated.

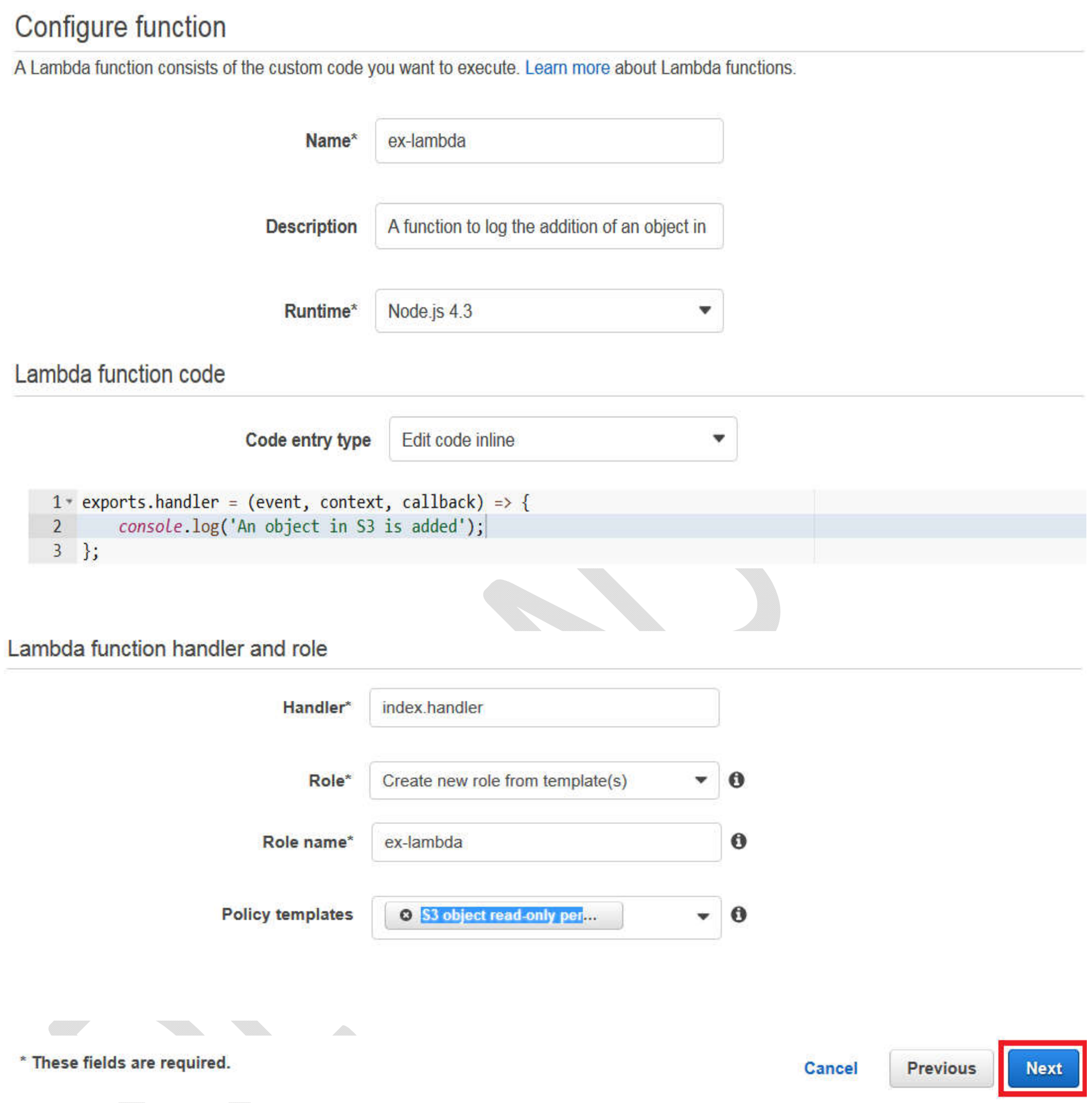
python2.7 · s3

**Step4:** On the next page we will be (1) setting a trigger, since we are going to work on S3, (2) select the S3 trigger and then (3) click next.


Configure triggers

Configure an optional trigger to automatically invoke your function.

**Step5:** On the configuration page, fill in the details. After that, fill the handler and role, leave the advanced settings as it is, in the end click next.



Configure function

A Lambda function consists of the custom code you want to execute. Learn more about Lambda functions.

Name* ex-lambda

Description A function to log the addition of an object in

Runtime* Node.js 4.3

Lambda function code

Code entry type Edit code inline

```
exports.handler = (event, context, callback) => {
    console.log('An object in S3 is added');
};
```

Lambda function handler and role

Handler* index.handler

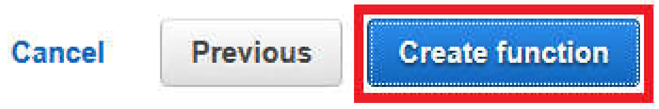
Role* Create new role from template(s)

Role name* ex-lambda

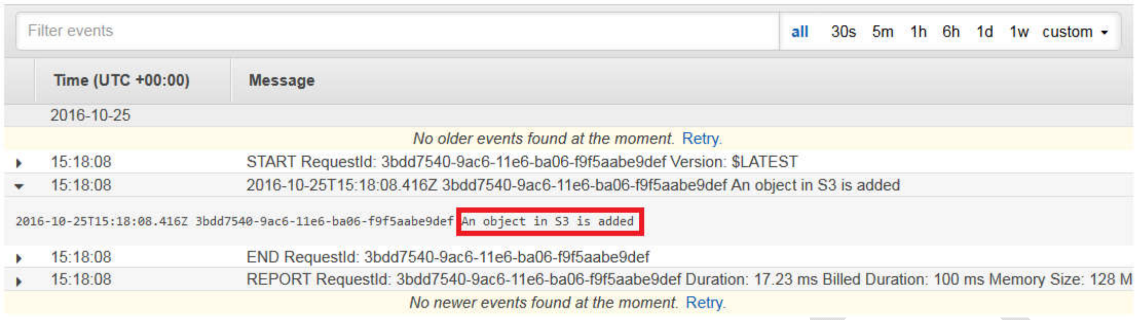
Policy templates S3 object read-only per...

* These fields are required.

**Step6:** On the next page, review all the information, and click on “Create function”.



**Step7:** Now, since we created the function for S3 bucket, the moment we add a file to our S3 bucket, we should get a log for the same in CloudWatch, which is a monitoring service from AWS.



```
Time (UTC +00:00)   Message
2016-10-25
                    No older events found at the moment. Retry.
15:18:08            START RequestId: 3bdd7540-9ac6-11e6-ba06-f9f5aabe9def Version: $LATEST
15:18:08            2016-10-25T15:18:08.416Z 3bdd7540-9ac6-11e6-ba06-f9f5aabe9def An object in S3 is added
2016-10-25T15:18:08.416Z 3bdd7540-9ac6-11e6-ba06-f9f5aabe9def An object in S3 is added
15:18:08            END RequestId: 3bdd7540-9ac6-11e6-ba06-f9f5aabe9def
15:18:08            REPORT RequestId: 3bdd7540-9ac6-11e6-ba06-f9f5aabe9def Duration: 17.23 ms Billed Duration: 100 ms Memory Size: 128 M
                    No newer events found at the moment. Retry.
```