Chapter 1: Getting Started with AWS
Welcome to Identity and Access Management

IAM users sign-in link:
https://ta-books.sigin.aws.amazon.com/console

IAM Resources

- Users: 5
- Roles: 20
- Groups: 2
- Identity Providers: 0
- Customer Managed Policies: 7

Security Status

- Activate MFA on your root account
- Create individual IAM users
- Use groups to assign permissions
- Apply an IAM password policy
- Rotate your access keys

4 out of 5 complete.
Add user

Set user details
You can add multiple users at once with the same access type and permissions. Learn more

User name*

☐ Add another user

Select AWS access type
Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more

Access type*

☐ Programmatic access
Enables an access key ID and secret access key for the AWS API, CLI, SDK, and other development tools.

☐ AWS Management Console access
Enables a password that allows users to sign-in to the AWS Management Console.

AWS Lambda
Run code without thinking about servers. Pay for only the compute time you consume.

Get started with AWS Lambda
Building Applications with AWS Lambda
- Event Source Mapping
- Supported Event Sources
- Deploying Lambda-based Applications

AWS Account

1. User
2. Amazon S3
   - Source bucket
   - Notification configuration
3. AWS Lambda
   - Execution role
4. Lambda function
   - Access policy

Custom app

1. Amazon Kinesis
   - Stream
2. Execution role
3. AWS Lambda
   - Event source mapping
   - Lambda function
**Monthly compute charges**

The monthly compute price is $0.00001667 per GB-s and the free tier provides 400,000 GB-s.

Total compute (seconds) = 3M * (1s) = 3,000,000 seconds

Total compute (GB-s) = 3,000,000 * 512MB/1024 = 1,500,000 GB-s

Total compute - Free tier compute = Monthly billable compute GB-s

1,500,000 GB-s - 400,000 free tier GB-s = 1,100,000 GB-s

**Monthly compute charges** = 1,100,000 * $0.00001667 = $18.34
**Monthly request charges**

The monthly request price is $0.20 per 1 million requests and the free tier provides 1M requests per month.

Total requests – Free tier requests = Monthly billable requests

3M requests – 1M free tier requests = 2M Monthly billable requests

**Monthly request charges = 2M * $0.2 / M = $0.40**
## AWS Lambda

<table>
<thead>
<tr>
<th>Region Name</th>
<th>Region</th>
<th>Endpoint</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>US East (N. Virginia)</td>
<td>us-east-1</td>
<td>lambda.us-east-1.amazonaws</td>
<td>HTTPS</td>
</tr>
<tr>
<td>US East (Ohio)</td>
<td>us-east-2</td>
<td>lambda.us-east-2.amazonaws</td>
<td>HTTPS</td>
</tr>
<tr>
<td>US West (N. California)</td>
<td>us-west-1</td>
<td>lambda.us-west-1.amazonaws</td>
<td>HTTPS</td>
</tr>
<tr>
<td>US West (Oregon)</td>
<td>us-west-2</td>
<td>lambda.us-west-2.amazonaws</td>
<td>HTTPS</td>
</tr>
<tr>
<td>Asia Pacific (Seoul)</td>
<td>ap-northeast-2</td>
<td>lambda.ap-northeast-2.amazonaws</td>
<td>HTTPS</td>
</tr>
<tr>
<td>Asia Pacific (Singapore)</td>
<td>ap-southeast-1</td>
<td>lambda.ap-southeast-1.amazonaws</td>
<td>HTTPS</td>
</tr>
<tr>
<td>Asia Pacific (Sydney)</td>
<td>ap-southeast-2</td>
<td>lambda.ap-southeast-2.amazonaws</td>
<td>HTTPS</td>
</tr>
<tr>
<td>Asia Pacific (Tokyo)</td>
<td>ap-northeast-1</td>
<td>lambda.ap-northeast-1.amazonaws</td>
<td>HTTPS</td>
</tr>
<tr>
<td>EU (Frankfurt)</td>
<td>eu-central-1</td>
<td>lambda.eu-central-1.amazonaws</td>
<td>HTTPS</td>
</tr>
<tr>
<td>EU (Ireland)</td>
<td>eu-west-1</td>
<td>lambda.eu-west-1.amazonaws</td>
<td>HTTPS</td>
</tr>
<tr>
<td>EU (London)</td>
<td>eu-west-2</td>
<td>lambda.eu-west-2.amazonaws</td>
<td>HTTPS</td>
</tr>
</tbody>
</table>

### Blank Function

Define the trigger and deploy your code by stepping through our wizard.

- custom

### kinesis-firehose-syslog-to-json

An Amazon Kinesis Firehose processor that converts input records from RFC3164 Syslog format to JSON.

- nodejs · kinesis-firehose

### alexa-skill-kit-sdk-factskill

Demonstrate a basic fact skill built with the ASK NodeJS SDK

- nodejs · alexa

### config-rule-change-triggered

An AWS Config rule that is triggered by configuration changes to EC2 instances. Checks instance types.

- nodejs4.3 · config

### dynamodb-process-stream

An Amazon DynamoDB trigger that logs the updates made to a table.

- nodejs · dynamodb

### microservice-http-endpoint

A simple backend (read/write to DynamoDB) with a RESTful API endpoint using Amazon API Gateway.

- nodejs · api-gateway
Congratulations! Your Lambda function "firstLambda" has been successfully created. You can now click on the "Test" button to input a test event and test your function.

```javascript
const exports = module.exports = { 
  handler: (event, context, callback) => { 
    // TODO implement 
    callback(null, 'Hello from Lambda'); 
  } 
};
```
Methods

RESOURCE ACTIONS
Create Method
Create Resource
Enable CORS
Edit Resource Documentation

API ACTIONS
Deploy API
Import API
Edit API Documentation
Delete API

/foo
✓
ANY
DELETE
GET
HEAD
OPTIONS
PATCH
POST
PUT
Choose the integration point for your new method.

**Integration type**
- Lambda Function
- HTTP
- Mock
- AWS Service

**Use Lambda Proxy integration**

**Lambda Region**

You are about to give API Gateway permission to invoke your Lambda function:

arn:aws:lambda:eu-central-1:186706155491:function:firstLambda

**OK**  **Cancel**
Method Request

Auth: NONE
ARN: arn:aws:execute-api:eu-central-1:186706155491:haaqakx41h/*/G

Integration Request

Type: LAMBDA
Region: eu-central-1

Method Response

HTTP Status: 200
Models: application/json => Empty

Integration Response

HTTP status pattern: - 
Output passthrough: Yes
Request: /foo
Status: 200
Latency: 90 ms
Response Body

"Hello from Lambda"

Response Headers

{"X-Amzn-Trace-Id":"Root=1-589a16ee-45e548c29c2a96e41d65979b","Content-Type":"application/json"}

Logs

Execution log for request test-request
Tue Feb 07 18:50:22 UTC 2017 : Starting execution for request: test-invoke-request
Tue Feb 07 18:50:22 UTC 2017 : Method request path: {}
Tue Feb 07 18:50:22 UTC 2017 : Method request query string: {}
Tue Feb 07 18:50:22 UTC 2017 : Method request headers: {}
Tue Feb 07 18:50:22 UTC 2017 : Method request body before transformations:

CloudWatch metrics at a glance (last 24 hours)
Chapter 2: Exploring the Serverless Framework

The way cloud should be.

Serverless is your toolkit for deploying and operating serverless architectures. Focus on your application, not your infrastructure.
Service Information
service: blog
stage: dev
region: eu-central-1
stack: blog-dev
api keys: None
endpoints:
POST - https://58pletzc5d.execute-api.eu-central-1.amazonaws.com/dev/articles
GET - https://58pletzc5d.execute-api.eu-central-1.amazonaws.com/dev/articles/{id}
PUT - https://58pletzc5d.execute-api.eu-central-1.amazonaws.com/dev/articles
DELETE - https://58pletzc5d.execute-api.eu-central-1.amazonaws.com/dev/articles
GET - https://58pletzc5d.execute-api.eu-central-1.amazonaws.com/dev/articles
functions:
hello: blog-dev-hello
createArticle: blog-dev-createArticle
readOneArticle: blog-dev-readOneArticle
updateArticle: blog-dev-updateArticle
deleteArticle: blog-dev-deleteArticle
readAllArticles: blog-dev-readAllArticles

Service Information
service: simple
stage: dev
region: eu-central-1
stack: simple-dev
api keys: None
endpoints:
GET - https://igknh40tnh.execute-api.eu-central-1.amazonaws.com/dev/hello
POST - https://igknh40tnh.execute-api.eu-central-1.amazonaws.com/dev/articles
GET - https://igknh40tnh.execute-api.eu-central-1.amazonaws.com/dev/articles/{id}
PUT - https://igknh40tnh.execute-api.eu-central-1.amazonaws.com/dev/articles
DELETE - https://igknh40tnh.execute-api.eu-central-1.amazonaws.com/dev/articles
GET - https://igknh40tnh.execute-api.eu-central-1.amazonaws.com/dev/articles
functions:
hello: simple-dev-hello
C:\Users\admin\Desktop\programming-aws-lambda-master\javascript\blog-app\sls invoke local -f hello

event is { awsRequestId: 'id',
  invokeId: 'id',
  logGroupName: '/aws/lambda/simple-dev-hello',
  logStreamName: '2019/09/22/[HEAD]13370884ca4ed8b77c427af260',
  functionName: 'simple-dev-hello',
  isDefaultVersion: true,
  functionVersion: 'HEAD',
  memoryLimitInMB: '1024',
  succeeded: [Function: succeed],
  failed: [Function: fail],
  done: [Function: done],
  getRemainingTimeInMills: [Function: getRemainingTimeInMills] }


{
  "statusCode": 200,
  "ev": "",
  "rt": 6000,
  "fn": "simple-dev-hello",
  "aid": "id"
}

region: eu-central-1
stack: simple-dev
api keys:
  None
endpoints:
  GET - https://nb2gqgav6i.execute-api.eu-central-1.amazonaws.com/dev/hello
  POST - https://nb2gqgav6i.execute-api.eu-central-1.amazonaws.com/dev/articles
  GET - https://nb2gqgav6i.execute-api.eu-central-1.amazonaws.com/dev/articles/{id}
  PUT - https://nb2gqgav6i.execute-api.eu-central-1.amazonaws.com/dev/articles
  DELETE - https://nb2gqgav6i.execute-api.eu-central-1.amazonaws.com/dev/articles
  GET - https://nb2gqgav6i.execute-api.eu-central-1.amazonaws.com/dev/articles

functions:
  hello: simple-dev-hello
  createArticle: simple-dev-createArticle
  readOneArticle: simple-dev-readOneArticle
  updateArticle: simple-dev-updateArticle
  deleteArticle: simple-dev-deleteArticle
  readAllArticles: simple-dev-readAllArticles

{
  "statusCode": 200,
  "ev": {},
  "rt": 5998,
  "fn": "simple-dev-hello",
  "aid": "ca5f1156-ef0d-11e6-bdf4-df482d5dd82e"
}
```json
{
  "statusCode": 200,
  "ev": {
    "Foo": "bar"
  },
  "rt": 6000,
  "fn": "simple-dev-hello",
  "aid": "id"
}
```

```
C:\Users\admin\Desktop\programming-aws-lambda-master\javascript\blog-app> sls invoke local -f hello -p event.json
event is {  
  "foo": "bar"
}
context is {  
  "awsRequestId": 'id',
  "invokedId": 'id',
  "logGroupName": '/aws/lambda/simple-dev-hello',
  "logStreamName": '2018/09/22/[HEAD/13370a84ca4ed8b77c427af268]',
  "functionVersion": 'HEAD',
  "isDefaultVersion": true,
  "functionName": 'simple-dev-hello',
  "memoryLimitInMB": '1024',
  "succeed": [Function: succeed],
  "fail": [Function: Fail],
  "done": [Function: done],
  "getRemainingTimeInMillis": [Function: getRemainingTimeInMillis] }
{
  "statusCode": 200,
  "body": '{"ev":{"foo":"bar"},"rt":6000}''
}
```

```
config ........................ Configure Serverless
config credentials .......... Configure a new provider profile for the Serverless Framework
create ........................ Create new Serverless service
deploy ........................ Deploy a Serverless service
deploy function ............. Deploy a single function from the service
deploy list ................. List deployed version of your Serverless Service
deploy list functions ....... List all the deployed functions and their versions
info .......................... Display Information about the service
install ........................ Install a Serverless service from GitHub or a plugin from the Serverless registry
invoke ........................ Invoke a deployed function
invoke local ................ Invoke function locally
logs .......................... Output the logs of a deployed function
metrics ........................ Show metrics for a specific function

C:\Users\admin\Desktop\programming-aws-lambda-master\javascript\blog-app> sls logs -f hello
START RequestId: 2e2f66f9-6e1f-11e8-bc0f-2dedd2183214 Version: $LATEST
2018-09-22 19:39:13.849 [100:00] 2e2f66f9-6e1f-11e8-bc0f-2dedd2183214
event is {  
  "foo": "bar"
}
done: [Function], succeed: [Function], fail: [Function],
logGroupName: '/aws/lambda/simple-dev-hello',
logStreamName: '2018/09/22/[HEAD/13370a84ca4ed8b77c427af268]',
functionName: 'simple-dev-hello',
```

```
memoryLimitInMB: '1024',
functionVersion: $LATEST',
invokedId: '2e2f66f9-6e1f-11e8-bc0f-2dedd2183214',
awsRequestId: '2e2f66f9-6e1f-11e8-bc0f-2dedd2183214',
```
Make a test call to your method with the provided input

Path
No path parameters exist for this resource. You can define path parameters by using the syntax `{myPathParam}` in a resource path.

Query Strings

```javascript
{hello}
param1=value1&param2=value2
```

Headers

```javascript
{hello}
Use a colon (:) to separate header name and value, and new lines to declare multiple headers. eg. Accept:application/json.
```

Stage Variables

No stage variables exist for this method.
Lambda: Simple Dev-Hello
Chapter 3: Building a Serverless Application

C:\Users\admin\Desktop\programming-aws-lambda-master\javascript\blog-app>sls deploy
Serverless: Packaging service...
Serverless: Excluding development dependencies...
Serverless: Creating Stack...
Serverless: Checking Stack create progress...

Serverless: Stack create finished...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading artifacts...
Serverless: Uploading service .zip file to S3 (5.96 KB)...
Serverless: Validating template...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...

Serverless: Stack update finished...

Service Information

service: blog
stage: dev
region: eu-central-1
stack: blog-dev
api keys:
  None
endpoints:

functions:
  createArticle: blog-dev-createArticle
  readArticle: blog-dev-readArticle
  updateArticle: blog-dev-updateArticle
  deleteArticle: blog-dev-deleteArticle
C:\Users\admin\Desktop\programming-aws-lambda-master\javascript\blog-app>ls invoke -f createArticle
{
    "statusCode": 200,
    "body": "{\"ev\":{},\"rt\":5929}"
}
endpoints:
POST - https://9owl38dug8.execute-api.eu-central-1.amazonaws.com/dev/articles
GET - https://9owl38dug8.execute-api.eu-central-1.amazonaws.com/dev/articles
PUT - https://9owl38dug8.execute-api.eu-central-1.amazonaws.com/dev/articles

Pretty Raw Preview JSON

```
1 |
2 | "message": "Created article."
3 |
```
Unable to import module 'articles/create': Error
  at Function.Module._resolveFilename (module.js:325:15)
  at Module._load (module.js:720:25)
  at Module.require (module.js:335:17)
  at require (internal/module.js:12:17)
  at Object.<anonymous> (/var/task/articles/model.js:13:14)
  at Module._load (/var/task/articles/model.js:489:26)
  at Object.<anonymous> (/var/task/articles/model.js:416:10)
  at Module._load (/var/task/articles/model.js:345:32)
  at Module.require (module.js:335:17)
  at Module.load (/var/task/articles/model.js:300:12)
  at Module.require (module.js:335:17)
END RequestId: e0e9d4d8-6ed0-11ea-a516-74af0b6a0b02 Duration: 6.02 ms
Billed Duration: 100 ms
Memory Size: 1824 MB
Max Memory Used: 39 MB

START RequestId: 315d0763-6d41-11ea-a516-f7a2f0b8a0b02 Version: $LATEST
Unable to import module 'articles/create': Error
  at Function.Module._resolveFilename (module.js:325:15)
  at Module._load (module.js:720:25)
  at Module.require (module.js:335:17)
  at require (internal/module.js:12:17)
  at Object.<anonymous> (/var/task/articles/model.js:13:14)
  at Module._load (/var/task/articles/model.js:489:26)
  at Object.<anonymous> (/var/task/articles/model.js:416:10)
  at Module._load (/var/task/articles/model.js:345:32)
  at Module.require (module.js:335:17)
  at Module.load (/var/task/articles/model.js:300:12)
  at Module.require (module.js:335:17)
END RequestId: 315d0763-6d41-11ea-a516-f7a2f0b8a0b02 Duration: 2.30 ms
Billed Duration: 100 ms
Memory Size: 1824 MB
Max Memory Used: 39 MB

START RequestId: 315d0763-6d41-11ea-a516-f7a2f0b8a0b02 Version: $LATEST
Unable to import module 'articles/create': Error
  at Function.Module._resolveFilename (module.js:325:15)
  at Module._load (module.js:720:25)
  at Module.require (module.js:335:17)
  at require (internal/module.js:12:17)
  at Object.<anonymous> (/var/task/articles/model.js:13:14)
  at Module._load (/var/task/articles/model.js:489:26)
  at Object.<anonymous> (/var/task/articles/model.js:416:10)
  at Module._load (/var/task/articles/model.js:345:32)
  at Module.require (module.js:335:17)
  at Module.load (/var/task/articles/model.js:300:12)
  at Module.require (module.js:335:17)
END RequestId: 315d0763-6d41-11ea-a516-f7a2f0b8a0b02 Duration: 6.08 ms
Billed Duration: 100 ms
Memory Size: 1824 MB
Max Memory Used: 35 MB

START RequestId: e2f9a59a-6e46-11ea-a990-b303bf9c4835 Version: $LATEST
2018-09-12 12:25:37,045 (ms:19)
e2f9a59a-6e46-11ea-a990-b303bf9c4835 Hello world
END RequestId: e2f9a59a-6e46-11ea-a990-b303bf9c4835 Duration: 3.22 ms
Billed Duration: 100 ms
Memory Size: 1824 MB
Max Memory Used: 21 MB

START RequestId: e2f9a59a-6e46-11ea-a990-b303bf9c4835 Version: $LATEST
2018-09-12 12:25:37,844 (ms:19)
e2f9a59a-6e46-11ea-a990-b303bf9c4835 Hello world
END RequestId: e2f9a59a-6e46-11ea-a990-b303bf9c4835 Duration: 3.22 ms
Billed Duration: 100 ms
Memory Size: 1824 MB
Max Memory Used: 21 MB

functions:
createArticle: blog-dev-createArticle
readArticle: blog-dev-readArticle
updateArticle: blog-dev-updateArticle
deleteArticle: blog-dev-deleteArticle

endpoints:
POST - https://90wl38dug8.execute-api.eu-central-1.amazonaws.com/dev/articles
GET - https://90wl38dug8.execute-api.eu-central-1.amazonaws.com/dev/articles
PUT - https://90wl38dug8.execute-api.eu-central-1.amazonaws.com/dev/articles
endpoints:
POST - https://9owl38dug8.execute-api.eu-central-1.amazonaws.com/dev/articles
GET - https://9owl38dug8.execute-api.eu-central-1.amazonaws.com/dev/articles
PUT - https://9owl38dug8.execute-api.eu-central-1.amazonaws.com/dev/articles

functions:
createArticle: blog-dev-createArticle
readArticle: blog-dev-readArticle
updateArticle: blog-dev-updateArticle
deleteArticle: blog-dev-deleteArticle

C:\Users\admin\Desktop\programming-aws-lambda-master\javascript\blog-app>npm init -y
wrote to C:\Users\admin\Desktop\programming-aws-lambda-master\javascript\blog-app\package.json:

{
    "name": "blog-app",
    "version": "1.0.0",
    "description": "",
    "main": "handler.js",
    "scripts": {
        "test": "echo \"Error: no test specified\" && exit 1"
    },
    "keywords": [],
    "author": "",
    "license": "ISC"
}
Serverless: Packaging service
Serverless: Excluding development dependencies...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading artifacts...
Serverless: Uploading service .zip file to S3 (6.48 MB)...
Serverless: Validating template...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...
............
Serverless: Stack update finished...

Service Information

service: Blog
stage: dev
region: eu-central-1
stack: blog-dev
api keys:
  None
endpoints:
  POST - https://jdonclv412.execute-api.eu-central-1.amazonaws.com/dev/articles
  GET - https://jdonclv412.execute-api.eu-central-1.amazonaws.com/dev/articles
  PUT - https://jdonclv412.execute-api.eu-central-1.amazonaws.com/dev/articles

functions:
  createArticle: blog-dev-createArticle
  readArticle: blog-dev-readArticle
  updateArticle: blog-dev-updateArticle
  deleteArticle: blog-dev-deleteArticle

C:\Users\adminDesktop\programming-aws-lambda-master\javascript\blog-app>sls deploy

C:\Users\adminDesktop\programming-aws-lambda-master\javascript\blog-app>sls invoke -f createArticle -p articles/event.json

{  
   "statusCode": 200  
}
### Website Speed Test - Page Load Results

**MN, USA**

http://s3-frontend.s3.amazonaws.com/central-1.amazonaws.com/

<table>
<thead>
<tr>
<th>AGENT / LOCATION</th>
<th>FIRST VISIT</th>
<th>REPEAT VISIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MN, USA</td>
<td>11.6 seconds</td>
<td>496.0 milliseconds</td>
</tr>
<tr>
<td>NY, USA</td>
<td>26.4 seconds</td>
<td>6.3 seconds</td>
</tr>
<tr>
<td>London, UK</td>
<td>2.4 seconds</td>
<td>379.0 milliseconds</td>
</tr>
<tr>
<td>CA, USA</td>
<td>15.0 seconds</td>
<td>765.0 milliseconds</td>
</tr>
<tr>
<td>FL, USA</td>
<td>10.9 seconds</td>
<td>286.0 milliseconds</td>
</tr>
<tr>
<td>Hong Kong, China</td>
<td>5.8 seconds</td>
<td>424.0 milliseconds</td>
</tr>
<tr>
<td>Montreal, Canada</td>
<td>10.4 seconds</td>
<td>410.0 milliseconds</td>
</tr>
<tr>
<td>Frankfurt, Germany</td>
<td>1.3 seconds</td>
<td>314.0 milliseconds</td>
</tr>
<tr>
<td>CO, USA</td>
<td>11.6 seconds</td>
<td>427.0 milliseconds</td>
</tr>
<tr>
<td>Brisbane, AU</td>
<td>31.4 seconds</td>
<td>446.0 milliseconds</td>
</tr>
<tr>
<td>TX, USA</td>
<td>13.8 seconds</td>
<td>774.0 milliseconds</td>
</tr>
<tr>
<td>Amsterdam, Netherlands</td>
<td>2.0 seconds</td>
<td>448.0 milliseconds</td>
</tr>
<tr>
<td>Tel-Aviv, Israel</td>
<td>27.0 seconds</td>
<td>918.0 milliseconds</td>
</tr>
<tr>
<td>VA, USA</td>
<td>9.9 seconds</td>
<td>282.0 milliseconds</td>
</tr>
<tr>
<td>Amazon-US-East</td>
<td>10.2 seconds</td>
<td>10.3 seconds</td>
</tr>
<tr>
<td>Shanghai, China</td>
<td>Temporarily Unavailable</td>
<td>Temporarily Unavailable</td>
</tr>
</tbody>
</table>

**Monitor Website Performance for a month**
Test as frequently as once per minute for 30 days

---

### Amazon CloudFront Getting Started

Either your search returned no results, or you do not have any distributions. Click the button below to create a new CloudFront distribution. A distribution allows you to distribute content using a worldwide network of edge locations that provide low latency and high data transfer speeds (learn more)

**Create Distribution**
Website Speed Test - Page Load Results

<table>
<thead>
<tr>
<th>AGENT / LOCATION</th>
<th>FIRST VISIT</th>
<th>REPEAT VISIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>MN, USA</td>
<td>4.4 seconds</td>
<td>461.0 milliseconds</td>
</tr>
<tr>
<td>NY, USA</td>
<td>4.3 seconds</td>
<td>566.0 milliseconds</td>
</tr>
<tr>
<td>London, UK</td>
<td>1.2 seconds</td>
<td>318.0 milliseconds</td>
</tr>
<tr>
<td>CA, USA</td>
<td>6.0 seconds</td>
<td>317.0 milliseconds</td>
</tr>
<tr>
<td>FL, USA</td>
<td>5.0 seconds</td>
<td>283.0 milliseconds</td>
</tr>
<tr>
<td>Hong Kong, China</td>
<td>147.0 milliseconds</td>
<td>156.0 milliseconds</td>
</tr>
<tr>
<td>Montreal, Canada</td>
<td>3.7 seconds</td>
<td>467.0 milliseconds</td>
</tr>
<tr>
<td>Frankfurt, Germany</td>
<td>1.2 seconds</td>
<td>326.0 milliseconds</td>
</tr>
<tr>
<td>CD, USA</td>
<td>3.2 seconds</td>
<td>435.0 milliseconds</td>
</tr>
<tr>
<td>Brisbane, AU</td>
<td>9.8 seconds</td>
<td>495.0 milliseconds</td>
</tr>
<tr>
<td>TX, USA</td>
<td>4.1 seconds</td>
<td>404.0 milliseconds</td>
</tr>
<tr>
<td>Amsterdam, Netherlands</td>
<td>977.0 milliseconds</td>
<td>154.0 milliseconds</td>
</tr>
<tr>
<td>Tel-Aviv, Israel</td>
<td>25.8 seconds</td>
<td>645.0 milliseconds</td>
</tr>
<tr>
<td>VA, USA</td>
<td>3.2 seconds</td>
<td>277.0 milliseconds</td>
</tr>
<tr>
<td>Amazon-US-East</td>
<td>10.7 seconds</td>
<td>10.3 seconds</td>
</tr>
<tr>
<td>Shanghai, China</td>
<td>39.8 seconds</td>
<td>398.0 milliseconds</td>
</tr>
<tr>
<td>Brussels, Belgium</td>
<td>10.0 seconds</td>
<td>11.3 seconds</td>
</tr>
</tbody>
</table>

MONITOR WEBSITE PERFORMANCE FOR A MONTH
Test as frequently as once per minute for 30 days

View Waterfall

Server Responses:
- 13 success
- 0 client
- 0 server
- 0 connection
Chapter 4: Programming AWS Lambda with Java
Create a new AWS Lambda Java project
Create a new AWS Lambda Java project in the workspace

Project name: demo
Maven configuration
  Group ID: com.amazonaws.lambda
  Artifact ID: demo
  Version: 1.0.0
  Package name: com.amazonaws.lambda.demo

Lambda Function Handler
Each Lambda function must specify a handler class which the service will use as the entry point to begin execution. Learn more about Lambda Java function handler.

Class Name: LambdaFunctionHandler
Input Type: S3 Event

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

Preview:
```java
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.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;
    }
}
```

---

13 errors, 1 warning, 4 others
Description | Resource | Path | Location | Type
---|---|---|---|---
Java Build Path Problems (1 item) | | demo | Build path | JRE System
Java Problems (13 items) | | demo | Build path | JRE System
package com.amazonaws.lambda.demo;

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

public class LambdaFunctionHandler implements RequestHandler<String, String> {

    @Override
    public String handleRequest(String input, Context context) {
        context.getLogger().log("Input: " + input);
        // TODO: implement your handler
        return input;
    }
}
AWS Toolkit Preferences

Global Configuration

Default Profile: default

This credential profile will be used by default to access all AWS regions that are not configured with a region-specific account.

Profile Details:
Sign up for a new AWS account or manage your existing AWS security credentials.

Profile Name: default
Access Key ID: AKIAJILYCLT65EB6RMIA
Secret Access Key: **********************
Session Token: 

Optional configuration:

Credentials file:
The location of the credentials file where all your configured profiles will be persisted.
Credentials file: /home/test/.aws/credentials

Automatically reload accounts when the credentials file is modified in the file system.

Timeouts:
Connection Timeout (ms)
20000
Socket Timeout (ms)
20000

See Network connections for more ways to configure how the AWS Toolkit connects to the Internet.

Get help or provide feedback on the AWS Java Development Forum
Or directly contact us via aws-eclipse-feedback@amazon.com

Restore Defaults  Apply  Cancel  Apply and Close
Select Target Lambda Function

Choose the region and the target AWS Lambda function you want to create or update for your local lambda handler.

Select Lambda Handler and Target Region

Select the Handler: com.amazonaws.lambda.demo.LambdaFunctionHandler

Select the AWS Region: EU (Frankfurt)

Select or Create a Lambda Function:

- [ ] Choose an existing Lambda function: Not Found
- [x] Create a new Lambda function: MyJavaFunction
Function Configuration
Configure this Lambda function and upload to AWS.

Basic Settings
Name: MyJavaFunction
Description: The description for the function (optional)

Function Role
Select the IAM role that AWS Lambda can assume to execute the function on your behalf:
Learn more about Lambda execution roles.
IAM Role: lambda_basic_execution [Create]

Function Versioning and Alias
You can publish a new immutable version and an alias to that version whenever you have a new revision of the Lambda function. Learn more about Lambda function versioning and aliasing.
 Publish new version
 Provide an alias to this new version
 Choose an existing function alias: Not Found
 Create a new function alias: beta

S3 Bucket for Function Code
S3 Bucket: lambda-function-bucket-eu-central-1-146179 [Create]

Upload Lambda zip file with encryption to protect data at rest by using Amazon S3 master-key or by using AWS KMS master-key. Learn more about Amazon S3 encryption.

None [ ] Amazon S3 master-key [ ] AWS KMS master-key

KMS Key: aws/lambda [Create]

Advanced Settings
Memory (MB): 512
Timeout (s): 15

< Back [ ] Next > [ ] Cancel [ ] Finish
package com.amazonaws.lambda.demo;

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

public class LambdaFunctionHandler implements RequestHandler<String, String> {

@Override
public String handleRequest(String input, Context context) {
    context.getLogger().log("Input: " + input);

    // TODO: implement your handler
    return "Remaining time [ms]: " + context.getRemainingTimeInMillis();
}
}

package com.amazonaws.lambda.demo;

import com.amazonaws.services.lambda.runtime.Context;
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 S3FunctionHandler implements RequestHandler<S3Event, String> {

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

    public S3FunctionHandler() {}

    // Test purpose only.
    S3FunctionHandler(AmazonS3 s3) {
        this.s3 = s3;
    }
}
```java
package com.amazonaws.lambda.demo;

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

public class S3FunctionHandler implements RequestHandler<S3Event, Object> {
    
    @Override
    public Object handleRequest(S3Event input, Context context) {
        context.getLogger().log("Input: " + input);

        // TODO: implement your handler
        return null;
    }
}
```
package com.amazonaws.lambda.demo;

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

public class S3FunctionHandler implements RequestHandler<S3Event, Object> {

    @Override
    public Object handleRequest(S3Event input, Context context) {
        for (S3EventNotificationRecord rec : input.getRecords()) {
            context.getLogger().log("Event Name: " + rec.getEventName() + "\n");
            context.getLogger().log("Event Source: " + rec.getEventSource() + "\n");
            S3ObjectEntity s3object = rec.getS3().getObject();
            context.getLogger().log("S3 Object Key: " + s3object.getKey() + "\n");
        }

        // TODO: implement your handler
        return null;
    }
}
Log output

The area below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. Click here to view the CloudWatch log group.

```
START RequestId: 41ad660fc-727e-11e8-9b82-a302e4c0456b Version: $LATEST
Event Name: ObjectCreated:Put
Event Source: aws:s3
S3 Object Key: HappyFace.jpg
END RequestId: 41ad660fc-727e-11e8-9b82-a302e4c0456b
REPORT RequestId: 41ad660fc-727e-11e8-9b82-a302e4c0456b Duration: 494.25 ms Billed Duration: 500 ms Memory Size: 512 MB Max Memory Used: 58 MB
```

Designer

Add triggers
Click on a trigger from the list below to add it to your function.

- S3
- AWS IoT
- CloudWatch Events
- CloudWatch Logs
- CodeCommit
- Cognito Sync Trigger
- DynamoDB
- Kinesis
- SNS
- SNS CloudWatch

AWS CloudFormation
AWS IoT
AWS Key Management Service
AWS Lambda
AWS X-Ray
Amazon CloudWatch
Amazon CloudWatch Events
<table>
<thead>
<tr>
<th>Time (UTC +0:00)</th>
<th>Message</th>
<th>Show in stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-06-17</td>
<td>START RequestId: 1d95061e-727b-11e6-bc22-d8d366b63a00 Version: $LATEST</td>
<td>![Image]</td>
</tr>
<tr>
<td>22:09:34</td>
<td>Input: com.amazonaws.services.lambda.runtime.events.S3Event@bd8de86a</td>
<td>![Image]</td>
</tr>
<tr>
<td>22:09:35</td>
<td>END RequestId: 1d95061e-727b-11e6-bc22-d8d366b63a00</td>
<td>![Image]</td>
</tr>
<tr>
<td>22:09:35</td>
<td>REPORT RequestId: 1d95061e-727b-11e6-bc22-d8d366b63a00 Duration: 610 ms Billed Duration: 700 ms Memory</td>
<td>![Image]</td>
</tr>
<tr>
<td>22:14:45</td>
<td>START RequestId: d7915635-727b-11e6-a75d-b33a36d3989f Version: $LATEST</td>
<td>![Image]</td>
</tr>
<tr>
<td>22:14:45</td>
<td>Input: com.amazonaws.services.lambda.runtime.events.S3Event@7a187f15</td>
<td>![Image]</td>
</tr>
<tr>
<td>22:14:45</td>
<td>END RequestId: d7915635-727b-11e6-a75d-b33a36d3989f</td>
<td>![Image]</td>
</tr>
<tr>
<td>22:14:45</td>
<td>REPORT RequestId: d7915635-727b-11e6-a75d-b33a36d3989f Duration: 48.49 ms Billed Duration: 100 ms Memory</td>
<td>![Image]</td>
</tr>
<tr>
<td>22:32:03</td>
<td>START RequestId: 41ad0a6c-727b-11e6-9b02-a302e4a0456b Version: $LATEST</td>
<td>![Image]</td>
</tr>
<tr>
<td>22:32:04</td>
<td>Event Name: ObjectCreated:Put</td>
<td>![Image]</td>
</tr>
<tr>
<td>22:32:04</td>
<td>Event Source: aws:s3</td>
<td>![Image]</td>
</tr>
<tr>
<td>22:32:04</td>
<td>S3 Object Key: HappyFace.jpg</td>
<td>![Image]</td>
</tr>
<tr>
<td>22:32:04</td>
<td>END RequestId: 41ad0a6c-727b-11e6-9b02-a302e4a0456b</td>
<td>![Image]</td>
</tr>
<tr>
<td>22:32:04</td>
<td>REPORT RequestId: 41ad0a6c-727b-11e6-9b02-a302e4a0456b Duration: 492.25 ms Billed Duration: 500 ms Memory</td>
<td>![Image]</td>
</tr>
</tbody>
</table>

2018-06-18
| 14:24:09       | START RequestId: 43664381-7303-11e6-8b35-e3e6e3789df Version: $LATEST | ![Image]        |
| 14:24:10       | Event Name: ObjectCreated:Put                                         | ![Image]        |
| 14:24:10       | Event Source: aws:s3                                                  | ![Image]        |
| 14:24:10       | S3 Object Key: smiling-cat.jpg                                         | ![Image]        |
| 14:24:10       | END RequestId: 43664381-7303-11e6-8b35-e3e6e3789df                   | ![Image]        |
| 14:24:10       | REPORT RequestId: 43664381-7303-11e6-8b35-e3e6e3789df Duration: 757.83 ms Billed Duration: 800 ms Memory | ![Image]        |
Create a new Serverless Java project

You can create a new Serverless Java project either from a Blueprint or an existing Serverless template file.

Project name: JavaBlog

Maven Configuration:
- Group ID: com.serverless
- Artifact ID: demo
- Version: 1.0.0
- Package name: com.serverless.demo

Select a Blueprint:
- article
  This is a Blueprint for creating an article API in API Gateway. It will create two Lambda functions, PutArticle and GetArticle, for creating and retrieving an article. It will also create an S3 bucket for hosting the article content and a DynamoDB table for storing article metadata.

Select a Serverless template file:

Import: [Browse]
package com.serverless.dynamo.function;

import java.io.ByteArrayInputStream;

public class PutArticle implements RequestHandler<ServerlessInput, ServerlessOutput> {
    // DynamoDB table name for storing article metadata.
    private static final String ARTICLE_TABLE_NAME = System.getenv("ARTICLE_TABLE_NAME");
    // DynamoDB table attribute name for storing article id.
    private static final String ARTICLE_TABLE_ID_NAME = "id";
    // DynamoDB table attribute name for storing the bucket name where holds the article's content.
    private static final String ARTICLE_TABLE_BUCKET_NAME = "bucket";
    // DynamoDB table attribute name for storing the bucket object key name that contains the article's content.
    private static final String ARTICLE_TABLE_KEY_NAME = "key";

    @Override
    public ServerlessOutput handleRequest(ServerlessInput serverlessInput, Context context) {
        // Using builder to create the clients could allow us to dynamically load the region from the AWS_REGION environment variable. Therefore we can deploy the Lambda functions to different regions without code change.
        AmazonDynamoDB dynamoDb = AmazonDynamoDBClientBuilder.standard().build();
        // Amazon S3
        AmazonS3 s3 = AmazonS3ClientBuilder.standard().build();
        ServerlessOutput output = new ServerlessOutput();

        try {
            String keyName = UUID.randomUUID().toString();
            String content = serverlessInput.getBody();
            s3.putObject(new PutObjectRequest{
                ARTICLE_BUCKET_NAME,
                keyName,
                new ByteArrayInputStream(content.getBytes(StandardCharsets.UTF_8)),
                new ObjectMetadata())
            );
        } catch (Exception e) {
            // Map<String, AttributeValue> attributes = convert(serverlessInput.getQueryStringParameters());
            // attributes.put(ARTICLE_TABLE_ID_NAME, new AttributeValue().withS(UUID.randomUUID().toString()));
            // attributes.put(ARTICLE_TABLE_BUCKET_NAME, new AttributeValue().withS(ARTICLE_BUCKET_NAME));
            // attributes.put(ARTICLE_TABLE_KEY_NAME, new AttributeValue().withS(keyName));

            dynamoDb.putItem(new PutItemRequest{
                withTableName(ARTICLE_TABLE_NAME),
                withItem(attributes)
            });
            output.setErrorCode("200");
        }
        return output;
    }
}
package com.serverless.demo.model;

import java.util.Map;

public class ServerlessInput {

    private String resource;
    private String path;
    private String httpMethod;
    private Map<String, String> headers;
    private Map<String, String> queryStringParameters;
    private Map<String, String> pathParameters;
    private Map<String, String> stageVariables;

    private String body;
    private RequestContext requestContext;
    private Boolean isBase64Encoded;

    public String getResource() {
        return resource;
    }

    public void setResource(String resource) {
        this.resource = resource;
    }

    public ServerlessInput withResource(String resource) {
        setResource(resource);
        return this;
    }

    public String getPath() {
        return path;
    }

    public void setPath(String path) {
        this.path = path;
    }

    public ServerlessInput withPath(String path) {
        setPath(path);
        return this;
    }

    public String getHttpMethod() {

```java
public ServerlessOutput handleRequest(ServerlessInput serverlessInput, Context context) {
    // Using builder to create the clients could allow us to dynamically load the region from the AWS_REGION env
    // variable. Therefore we can deploy the Lambda functions to different regions without code change.
    AmazonDynamoDB dynamoDb = AmazonDynamoDBClientBuilder.standard().build();
    AmazonS3 s3 = AmazonS3ClientBuilder.standard().build();
    ServerlessOutput output = new ServerlessOutput();
    try {
        String keyName = UUID.randomUUID().toString();
        String content = serverlessInput.getBody();
        s3.putObject(new PutObjectRequest{
            ARTICLE_BUCKET_NAME,
            keyName,
            new ByteArrayInputStream(content.getBytes(StandardCharsets.UTF_8)),
            new ObjectMetadata()
        });
        Map<String, AttributeValue> attributes = convert(serverlessInput.getQueryStringParameters());
        attributes.putIfAbsent(ARTICLE_TABLE_ID_NAME, new AttributeValue().withS(UUID.randomUUID().toString()));
        attributes.put(ARTICLE_TABLE_BUCKET_NAME, new AttributeValue().withS(ARTICLE_BUCKET_NAME));
        attributes.put(ARTICLE_TABLE_KEY_NAME, new AttributeValue().withS(keyName));
        dynamoDb.putItem(new PutItemRequest()
            .withTableName(ARTICLE_TABLE_NAME)
            .withItem(attributes));
        output.getStatusCode(200);
        output.setBody(“Successfully inserted article “ + attributes.get(ARTICLE_TABLE_ID_NAME).getS());
    }
    if (serverlessInput.getQueryStringParameters() == null || serverlessInput.getQueryStringParameters().get(ARTICLE_TABLE_ID_NAME) == null) {
        throw new Exception(“Parameter " + ARTICLE_TABLE_ID_NAME + " in query must be provided!”);
    }
    Map<String, AttributeValue> item = dynamoDb.getItem(new GetItemRequest()
        .withTableName(ARTICLE_TABLE_NAME)
        .withKey(key))
        .getItem();
```
String s3ObjectKey = item.get(ARTICLE_TABLE_KEY_NAME).getS();

"Description": "Simple article service."

"Parameters": {
  "ArticleBucketName": {
    "Type": "String",
    "Default": "serverless-blueprint-article-bucket",
    "Description": "Name of S3 bucket used to store the article content. If left blank, AWS CloudFormation would manage this resource."
  },
  "ArticleTableName": {
    "Type": "String",
    "Default": "serverless-blueprint-article-table",
    "Description": "Name of DynamoDB table used to store the article metadata. If left blank, AWS CloudFormation would manage this resource."
  },
  "ReadCapacity": {
    "Type": "Number",
    "Description": "Read capacity for the DynamoDB blog table."
  },
  "WriteCapacity": {
    "Type": "Number",
    "Description": "Write capacity for the DynamoDB blog table."
  }
}

"GetArticle": {
  "Type": "AWS::Serverless::Function",
  "Properties": {
    "Handler": "com.serverless.demo.function.GetArticle",
    "Runtime": "java8",
    "CodeUri": "./target/demo-1.0.0.jar",
    "Policies": [
      "AmazonDynamoDBReadOnlyAccess",
      "AmazonS3ReadOnlyAccess"
    ],
    "Environment": {
      "Variables": {
        "ARTICLE_TABLE_NAME": { "Ref": "ArticleTableName" },
        "ARTICLE_BUCKET_NAME": { "Ref": "ArticleBucketName" }
      }
    }
  },
  "Events": {
    "GetResource": {
      "Type": "Api",
      "Properties": {
        "Path": "/",
        "Method": "get"
      }
    }
  }
}
"PutArticle": {
  "Type": "AWS::Serverless::Function",
  "Properties": {
    "Handler": "com.serverless.demo.function.PutArticle",
    "Runtime": "java8",
    "CodeUri": "/target/demo-1.0.0.jar",
    "Policies": [
      "AmazonDynamoDBFullAccess",
      "AmazonS3FullAccess"
    ]
  }
}

"ArticleTable": {
  "Type": "AWS::DynamoDB::Table",
  "Properties": {
    "AttributeDefinitions": [
      {
        "AttributeName": "id",
        "AttributeType": "S"
      }
    ],
    "KeySchema": [
      {
        "AttributeName": "id",
        "KeyType": "HASH"
      }
    ],
    "ProvisionedThroughput": {
      "ReadCapacityUnits": {
        "Ref": "ReadCapacity"
      },
      "WriteCapacityUnits": {
        "Ref": "WriteCapacity"
      }
    },
    "TableName": {"Ref": "ArticleTableName"}
  }
}"
"ArticleBucket": {
  "Type": "AWS::S3::Bucket",
  "Properties": {
    "BucketName": {"Ref": "ArticleBucketName"}
  }
}

"Default": "serverless-blueprint-article-bucket-23648817236827282",

![Image of AWS console]

![Image of stack deployment]

![Image of stack events]

"Deploy Serverless application to AWS"

**Fill in stack template parameters**

Provide values for template parameters.

- **ReadCapacity**: 3 (Read capacity for the DynamoDB blog table)
- **ArticleBucketName**: serverless-blueprint-article-bucket-23648817236827282 (Name of S3 bucket used to store the article content. If left blank, AWS CloudFormation would manage this resource)
- **ArticleTableName**: serverless-blueprint-article-table (Name of DynamoDB table used to store the article metadata. If left blank, AWS CloudFormation would manage this resource)
- **WriteCapacity**: 3 (Write capacity for the DynamoDB blog table)
### JavaBlog-devstack - EU (Frankfurt)

<table>
<thead>
<tr>
<th>Event</th>
<th>Stack Name</th>
<th>Status</th>
<th>Created</th>
<th>Created Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>JavaBlog-devstack</td>
<td>CREATE COMPLETE</td>
<td>Tue Jun 19 01:46:22 IST 2018</td>
<td>Tue Jun 19 01:47:36 IST 2018</td>
</tr>
</tbody>
</table>

**Events**

<table>
<thead>
<tr>
<th>Event Time</th>
<th>State</th>
<th>Resource Type</th>
<th>Logical ID</th>
<th>Physical ID</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tue Jun 19 01:46:22 IST 2018</td>
<td>CREATE_IN_PROGRESS</td>
<td>AWS::DynamoDB::Table</td>
<td>PutArticlePutResourcePermissions</td>
<td>JavaBlog-devstack-PutArticlePutResourcePermissions</td>
<td>Resource creation initiated</td>
</tr>
<tr>
<td>Tue Jun 19 01:46:21 IST 2018</td>
<td>CREATE_COMPLETE</td>
<td>AWS::Lambda::Permission</td>
<td>PutArticlePutResourcePermissions</td>
<td>serverless-blueprint-article-tat</td>
<td></td>
</tr>
</tbody>
</table>

**Resources**

- **AWS::DynamoDB::Table**
  - Logical ID: PutArticlePutResourcePermissions
  - Physical ID: serverless-blueprint-article-tat
  - Reason: Resource creation initiated

**Parameters**

- **ServerlessRestApi**
  - Logical ID: 20c8556x2
  - Physical ID: 20c8556x2
  - Reason: Resource creation initiated

**Outputs**

- **ServerlessRestApi**
  - Logical ID: 20c8556x2
  - Physical ID: 20c8556x2
  - Reason: Resource creation initiated
Chapter 5: Programming AWS Lambda with Python
**S3 trigger**

**Bucket**
Please select the S3 bucket that serves as the event source. The bucket must be in the same region as the function.

![Bucket dropdown](serverless-python-lambda-bucket-78940987654321)

**Event type**
Select the events that you want to 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.

![Event type dropdown](Object Created (All))

**Prefix**
Enter an optional prefix to limit the notifications to objects with keys that start with matching characters.

![Prefix field](e.g. images/)

**Filter pattern**
Enter an optional filter pattern.

![Filter pattern field](e.g. .jpg)

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

- **Enable trigger**
  Enable the trigger now, or create it in a disabled state for testing (recommended).
```python
from __future__ import print_function
import json
import urllib
import boto3

print('loading function')

s3 = boto3.client('s3')

def lambda_handler(event, context):
    #print("Received event: " + json.dumps(event, indent=2))
    # Get the object from the event and show its content type
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.unquote_plus(event['Records'][0]['s3']['object']['key'].encode('utf8'))
    try:
        response = s3.get_object(Bucket=bucket, Key=key)
        print("CONTENT TYPE: " + response['ContentType'])
        return response['ContentType']
    except Exception as e:
        print(e)
        print('Error getting object {} from bucket {}. Make sure they exist and your bucket is in the same region as this function.
    raise
```

```python
s3 = boto3.client('s3')
bucket = event['Records'][0]['s3']['bucket']['name']
key = urllib.unquote_plus(event['Records'][0]['s3']['object']['key'].encode('utf8'))
response = s3.get_object(Bucket=bucket, Key=key)
print("CONTENT TYPE: " + response['ContentType'])
return response['ContentType']
except Exception as e:
    print(e)
    print('Error getting object {} from bucket {}. Make sure they exist and your bucket is in the same region as this function.
    raise
```
Configure test event

A function can have up to 10 test events. The events are persisted so you can switch to another computer or web browser and test your function with the same events.

- Create new test event
- Edit saved test events

Event template

- S3 Put
- S3 Delete

AWS
- CloudFront Modify Response Header
- CloudFront AB Test
- AWS Config Change Triggered Rule
- CodeCommit
- API Gateway Authorizer
- AWS Config Change Triggered Rule - Oversized
- CloudFormation Create Request
- SES Email Receiving
- Rekognition S3 Request
- CloudFront HTTP Redirect
- API Gateway AWS Proxy
- Kinesis Firehose streams as source
- Scheduled Event
- CloudWatch Logs
- SNS
- AWS Batch Get Job Request
Congratulations! Your Lambda function “PyFunS3” has been successfully created and configured with serverless-python-lambda-bucket-78940967654321 as a trigger. You can now click on the "Test" button to input a test event and test your function.

**Execution result: failed**

The result returned by your function execution:
```
stack_trace:
  [  
    "<lambda_function.py",  
    "$2",  
    "lamba_handler",  
    "raise e"
  ],
  
  error_type: "ClientError",
```

**Summary**

- **Code SHA**: 256
- **Duration**: 670.77 ms
- **Billed duration**: 700 ms
- **Resources configured**: 1.28 GB
- **Max memory used**: 46 MB

**Log output**

The area below shows the log entries in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. Click here to view the CloudWatch log group.

```
START RequestId:6902009-73d3-11e8-8763-699b843a141 Version: SLATEST
An error occurred (AccessDenied) when calling the GetObject operation: Access Denied
Error getting object happyFace.png from bucket sourceBucket. Make sure they exist and your bucket is in the same region as this function.
An error occurred (AccessDenied) when calling the GetObject operation: Access Denied: ClientError
Traceback (most recent call last):
  File "<lambda_function.py", line 24, in lambda_handler
```

**CloudWatch > Logs Groups > (aws-lambda/PyFunS3) 2016/06/19 [SLATEST]6902009-73d3-11e8-8763-699b843a141**

No older events found at the moment. No newer events found at the moment.
Serverless: Successfully generated boilerplate for template: "aws-python"

Admin@Admin:~/Desktop/admin/serverless/programming-aws-lambda-master/python$ sls invoke local -f hello
{
  "body": "{\"input\": {}, \"message\": \"Go Serverless v1.0! Your function executed successfully!\"\"},
  "statusCode": 200
}
Admin@Admin:~/Desktop/admin/serverless/programming-aws-lambda-master/python$ sls deploy
Serverless: Packaging service...
Serverless: Excluding development dependencies...
Serverless: Creating Stack...
Serverless: Checking Stack create progress...
......
Serverless: Stack create finished...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading artifacts...
Serverless: Uploading service .zip file to S3 (12.49 KB)...
Serverless: Validating template...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...
.............
Serverless: Stack update finished...

Service Information
service: pyblog
stage: dev
region: eu-central-1
stack: pyblog-dev
api keys:
  None
endpoints:
  None
functions:
  hello: pyblog-dev-hello

Functions:
hello: pyblog-dev-hello
Admin@Admin:~/Desktop/admin/serverless/programming-aws-lambda-master/python$ sls invoke -f hello
{
  "body": "{"input": {}, "message": "Go Serverless v1.0! Your function executed successfully!"},
  "statusCode": 200
}
Admin@Admin:~Desktop/admin/serverless/programming-aws-lambda-master/python$ sls deploy
Serverless: Packaging service...
Serverless: Excluding development dependencies...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading artifacts...
Serverless: Uploading service .zip file to S3 (12.52 KB)...
Serverless: Validating template...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...

Serverless: Stack update finished...

**Service Information**

**service:** pyblog
**stage:** dev
**region:** eu-central-1
**stack:** pyblog-dev
**api keys:**
  None
**endpoints:**
  *POST* - https://jppj97nlo7.execute-api.eu-central-1.amazonaws.com/dev/create
**functions:**
  create: pyblog-dev-create

---

Admin@Admin:~Desktop/admin/serverless/programming-aws-lambda-master/python$ sls invoke -f create
```
{
  "body": "{"message": "Created new article"},
  "statusCode": 200
}
```
Admin@Admin:~/Desktop/admin/serverless/programming-aws-lambda-master/python$ sls deploy
Serverless: Packaging service...
Serverless: Excluding development dependencies...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading artifacts...
Serverless: Uploading service .zip file to S3 (12.69 KB)...
Serverless: Validating template...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...

..............
Serverless: Stack update finished...

Service Information
service: pyblog
stage: dev
region: eu-central-1
stack: pyblog-dev
api keys:
None
endpoints:
  POST - https://jppj97nlo7.execute-api.eu-central-1.amazonaws.com/dev/create
functions:
  create: pyblog-dev-create

Admin@Admin:~/Desktop/admin/serverless/programming-aws-lambda-master/python$ sls invoke -f create -p event.json
{
  "body": "{
    "ResponseMetadata": {
      "RetryAttempts": 0,
      "HTTPStatusCode": 200,
      "RequestId": "742VSAUSNQ0Q9FIELDD73VMPRVRWKQN50SAMJ3F66Q9A5
UAA3h",
      "HTTPheaders": {
        "x-amzn-requestid": "742VSAUSNQ0Q9FIELDD73VMPRVRWKQN50SAMJ3F66Q9A5
UAA3h",
        "server": "Server",
        "connection": "keep alive",
        "x-amz-crc32": "Z74SQuF1A3v",
        "data": "Sat, 31 Jan 2018 17:16:37 GMT",
        "content-type": "application/x-amz-json-1.1"
      },
    "statusCode": 200
  }
}
Admin@Admin:~/Desktop/admin/serverless/programming-aws-lambda-master/python
$ sls deploy
Serverless: Packaging service...
Serverless: Excluding development dependencies...
Serverless: Uploading CloudFormation file to S3...
Serverless: Uploading artifacts...
Serverless: Uploading service .zip file to S3 (14.27 KB)...
Serverless: Validating template...
Serverless: Updating Stack...
Serverless: Checking Stack update progress...

Serverless: Stack update finished...

**Service Information**

**service:** pyblog
**stage:** dev
**region:** eu-central-1
**stack:** pyblog-dev
**api keys:** None
**endpoints:**
  - GET - https://jppj97nlo7.execute-api.eu-central-1.amazonaws.com/dev/articles/{id}

**Functions:**
- create: pyblog-dev-create
- read: pyblog-dev-read
- update: pyblog-dev-update
- delete: pyblog-dev-delete

**POST Request**

![POST Request](https://jppj97nlo7.execute-api.eu-central-1.amazonaws.com/dev/articles)

**Authorization**
- Headers [1]
  - Body
  - Pre-request Script
  - Tests

**Request Body**

```
{"article_id": "7b8f5f5e:6e67-4e97-88b7-2341b20f494f", "text": "Hello from Postman"}
```

**GET Request**

![GET Request](https://jppj97nlo7.execute-api.eu-central-1.amazonaws.com/dev/articles/6945daca-7609-11ea-ae2b-06fd3e45f6b1)

**Request Headers**
- Type
  - No Auth

**Request Body**

```
{"text": "Hello from Postman", "article_id": "6945daca-7609-11ea-ae2b-06fd3e45f6b1"}
```
Chapter 6: Programming AWS Lambda with C#
Publish AWS Serverless Application

Profile

Account profile to use: default
Region: EU (Frankfurt)

Build Settings

Configuration: Release
Framework: netcoreapp1.0

CloudFormation Settings

Stack Name: cs-serverless-stack
S3 Bucket: cs-bucket-1565378277319u2

Save settings to aws-lambda-tools-defaults.json for future deployments.
**Edit Template Parameters**

These are parameters associated with your AWS CloudFormation template. You may review and proceed with the default parameters or make customizations as needed.

**BlogTableName**: CsBlogTable
Name of DynamoDB table used to store the blog post. If left blank a new table will be created.

**ReadCapacity**: 1
Read capacity for the DynamoDB blog table.

**ShouldCreateTable**: true
If true then the DynamoDB blogging table will be created with the CloudFormation stack.

**WriteCapacity**: 1
Write capacity for the DynamoDB blog table.