Lambda@Edge – General Availability Release Candidate 1 – 7/10/2017

Lambda@Edge launched in Preview at ReInvent in 2016. Since then, we launched incremental functionalities (HTTP response generation, and CloudWatch support in Q1 2017), and we are excited to share the general availability Release Candidate 1 (RC1). It includes new features such as response generation, and network calls at origin facing hooks. We are looking forward to getting your feedback about this new version.

What has changed between Preview and RC1?

Features and Capability

<u>Preview</u>
You can write Lambda functions with the Edge-Node.js runtime. After whitelisting for the Preview access, you have the ability to associate Lambda@Edge functions with up to one distribution with the whitelisted account, and function executions up to 100 TPS globally. Limited CloudWatch logging and metrics support are also available. Functions are then replicated to AWS edge locations globally, and available for execution when a CloudFront request lands.

<u>RC1</u>
With RC1, you can create Lambda functions in the US East (N. Virginia) Region with standard Lambda Node.js-6.10 runtime. Lambda function created outside US East region cannot be associated with CloudFront distribution. To create association with CloudFront distributions, first publish a new Lambda function version. Only associations with function versions will be supported. Associations with $LATEST and aliases will not be supported.

Once an association is created, we replicate function versions to all AWS regions and edge locations for execution at the location closest to your end viewers. CloudWatch metrics for functions across all edge locations will be available in the closest replica AWS region. Lambda@Edge replica invocations will count towards Lambda concurrency limits in each region.

In addition, there is a change to the event object (as documented in following sections) that impacts how we handle case sensitivity in accessing headers.

With RC1, you will also be able to use the AWS SDK, make network calls and generate HTTP responses from scratch from Lambda@Edge functions in response to origin facing events.

Table 1 – Lambda@Edge RC1 Service limits

Item	**Lambda@Edge RC1: Origin facing**	**Lambda@Edge RC1: Viewer facing**	**Lambda@Edge Preview**
Memory	128MB	128MB	128MB
Function timeout	3 seconds	100 ms	50 ms
Minimum billing duration	50 ms	50 ms	50 ms
Concurrent executions	1000 default (upto 20K by request)	1000 default (upto 20K by request)	3 (not public)
/tmp space	0	0	0

Deployment package size	1MB	1MB	1MB
Languages	Node.js 6.10	Node.js 6.10	Node.js 4.3
Network calls	Yes	No	No
Envar	No	No	No
DLQ	No	No	No
VPC	No	No	No
XRay	No	No	No

Programming Model

The programming model for using Node.js with Lambda@Edge is the same as using Lambda's Node.js 6.10 runtime in a region. For more information, see Programming Model (Node.js). You can also include third-party packages. However, when responding to *request events*, the request object should be returned as part of callback so that CloudFront can modify the request. The same applies while responding to *response events*. The response object should be returned.

```
exports.origin_response_handler = function(event, context, callback) {
    var headers = event.Records[0].cf.response.headers;
    for (var header in headers) {
        /* your custom header logic */;
    }
    callback(null, event.Records[0].cf.response);
}

exports.origin_request_handler = function(event, context, callback) {
    var request = event.Records[0].cf.request;
    request.uri = /* your custom uri logic  */;
    var headers = request.headers;
    for (var header in headers) {
       /* your custom header logic */;
    }
     callback(null , event.Records[0].cf.request);
}
```

Event Structure

Please note that there is a change in how headers are accessed to view headers as a key and value pair, as compared to the current implementation in Preview (changes highlighted inline below).

The following example shows a request event.

Request Event

```
{
   "Records": [
      {
         "cf": {
            "config": {
               "distributionId": "EXAMPLE"
            },
            "request": {
               "uri": "/me.pic",
               "method": "GET",
               "httpVersion": "2.0",
               "clientIp": "2001:cdba::3257:9652",
               "headers": {
                  "user-agent": [
                     {
                        "key": "User-Agent",
                        "value": "Test Agent"
                     }
                  ],
                  "host": [
                     {
                        "key": "Host",
                        "value": "d2fadu0nynjpfn.cloudfront.net"
                     }
                  ]
               }
            }
         }
      }
   ]
}
```

The request event contains the following values:

- **uri (read/write):** Relative path of the content that is being requested from CloudFront. Updating this field changes the content that is being served. If you update this field, the new relative path must begin with a forward slash (**/**).
- **method (read-only):** HTTP method of the incoming request.
- **httpVersion (read-only):** HTTP version of the incoming request.
- **clientIp (read-only):** Client IP for the incoming request.
- **headers (read/write):** The headers of the incoming request. Each header is represented as an array of values. For information about restrictions on header usage, see Header Restrictions.

The following example shows a response event:

Response Event

```
{
   "Records": [
      {
         "cf": {
            "config": {
               "distributionId": "EDFDVBD6EXAMPLE "
            },
            "request": {
               "uri": "/me.pic",
               "method": "GET",
               "httpVersion": "2.0",
               "clientIp": "2001:cdba::3257:9652",
               "headers": {
                  "user-agent": [
                     {
                        "key": "User-Agent",
                        "value": "Test Agent"
                     }
                  ],
                  "host": [
                     {
                        "key": "Host",
                        "value": "d2fadu0nynjpfn.cloudfront.net"
                     }
                  ]
               }
            },
            "response": {
               "status": "200",
               "statusDescription": "HTTP OK",
               "httpVersion": "2.0",
               "headers": {
                  "user-agent": [
                     {
                        "key": "User-Agent",
                        "value": "mozilla"
                     },
                     {
                        "key": "User-Agent",
                        "value": "safari"
                     }
                  ],
                  "vary": [
                     {
                        "key": "Vary",
                        "value": "*"
                     }
                  ]
               }
            }
         }
      }
   ]
}
```

The response event contains the following values:

- **distributionId:** The ID of the distribution that's associated with the request.
- **request:** The request that the viewer submitted.
- **Status (read-only):** HTTP status code that is served to clients.
- **statusDescription (read-only):** HTTP status that is served to clients.
- **httpVersion (read-only):** HTTP version of the response.
- **Headers (read/write):** The headers of the response. Each header is represented as an array of values. For more information, see Header Restrictions.

Execution Role

Ensure the role associated with your Lambda function is assumable with *edgelambda.amazonaws.com* as well as *lambda.amazonaws.com* principals. You can either use the console to edit the trust policy of the role, or the AWS CLI to update the role policy. For more information about roles, see Modifying a Role in the *IAM User Guide*.

The following example shows a role trust policy:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "Service": "lambda.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
      },
      {
         "Effect": "Allow",
         "Principal": {
            "Service": "edgelambda.amazonaws.com"
         },
         "Action": "sts:AssumeRole"
      }
   ]
}
```

Example

The following example shows how to use Lambda@Edge for A/B testing.

```
'use strict';

exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;
    const headers = request.headers;

    if (request.uri !== '/experiment-pixel.jpg') {
        // do not process if this is not an A-B test request
        callback(null, request);
    } else {
        const experimentCookieName = 'X-Experiment-Name=';

        const groupA = 'A';
        const groupB = 'B';

        const groupAObject = '/experiment-group/control-pixel.jpg';
        const groupBObject = '/experiment-group/treatment-pixel.jpg';

        /*
         * Lambda at the Edge headers are array objects.
         * Client may send multiple Cookie headers, i.e.:
         * > GET /viewerRes/test HTTP/1.1
         * > User-Agent: curl/7.18.1 (x86_64-unknown-linux-gnu) libcurl/7.18.1
OpenSSL/1.0.1u zlib/1.2.3
         * > Cookie: First=1; Second=2
         * > Cookie: ClientCode=abc
         * > Host: example.com
         * You can access the first Cookie line by
         *    headers["cookie"][0].value
         * and the second by
         *    headers["cookie"][1].value
         *   headers["Cookie"][0].value will return "First=1; Second=2", cookie
tokens are not parsed
         * separately.
         */
        let modifiedUri = false;
        if (headers.cookie) {
            for (let i = 0; i < headers.cookie.length; i++) {
                const experimentIndex =
headers.cookie[i].value.indexOf(experimentCookieName);
                console.log(experimentIndex);
                if (experimentIndex >= 0) {
                    if (headers.cookie[i].value[experimentIndex +
experimentCookieName.length] === groupA) {
                        request.uri = groupAObject;
                        modifiedUri = true;
                    } else if (headers.cookie[i].value[experimentIndex +
experimentCookieName.length] === groupB) {
                        request.uri = groupBObject;
                        modifiedUri = true;
                    }
                }
            }
        }

        /*
         * If this is the first time the viewer is
         * requesting this or we have not found
         * the experiment cookies,
         * randomly distribute viewers between objects.
         */
        if (!modifiedUri) {
            if (Math.random() < 0.75) {
                // 75% of the viewers go to group A.
                request.uri = groupAObject;
            } else {
                request.uri = groupBObject;
            }
        }

        console.log(`Request uri set to "${request.uri}"`);
        callback(null, request);
    }
};
```

Input Test event

```
{
    "Records": [
        {
            "cf": {
                "config": {
                    "distributionId": "EXAMPLE"
                },
                "request": {
                    "uri": "/experiment-pixel.jpg",
                    "method": "GET",
                    "clientIp": "2001:cdba::3257:9652",
                    "headers": {
                        "user-agent": [
                          {
                            "key": "User-Agent",
                            "value": "Test Agent"
                          }
                        ],
                        "host": [
                          {
                            "key": "Host",
                            "value": "d123.cf.net"
                          }
                        ],
                        "cookie": [
                          {
                            "key": "Cookie",
                            "value": "SomeCookie=1, AnotherOne=A, X-Experiment-Name=B"
                          }
                        ]
                    }
                }
            }
        }
    ]
}
```