Getting the most out of Edge Computing

Best practices, use cases, and tradeoffs

David Brown – Sr. Product Manager Amazon CloudFront
September 27, 2021

Agenda

Introduction to AWS Lambda@Edge and Amazon CloudFront Functions

Choosing between CloudFront Functions and Lambda@Edge

When to use Edge Computing

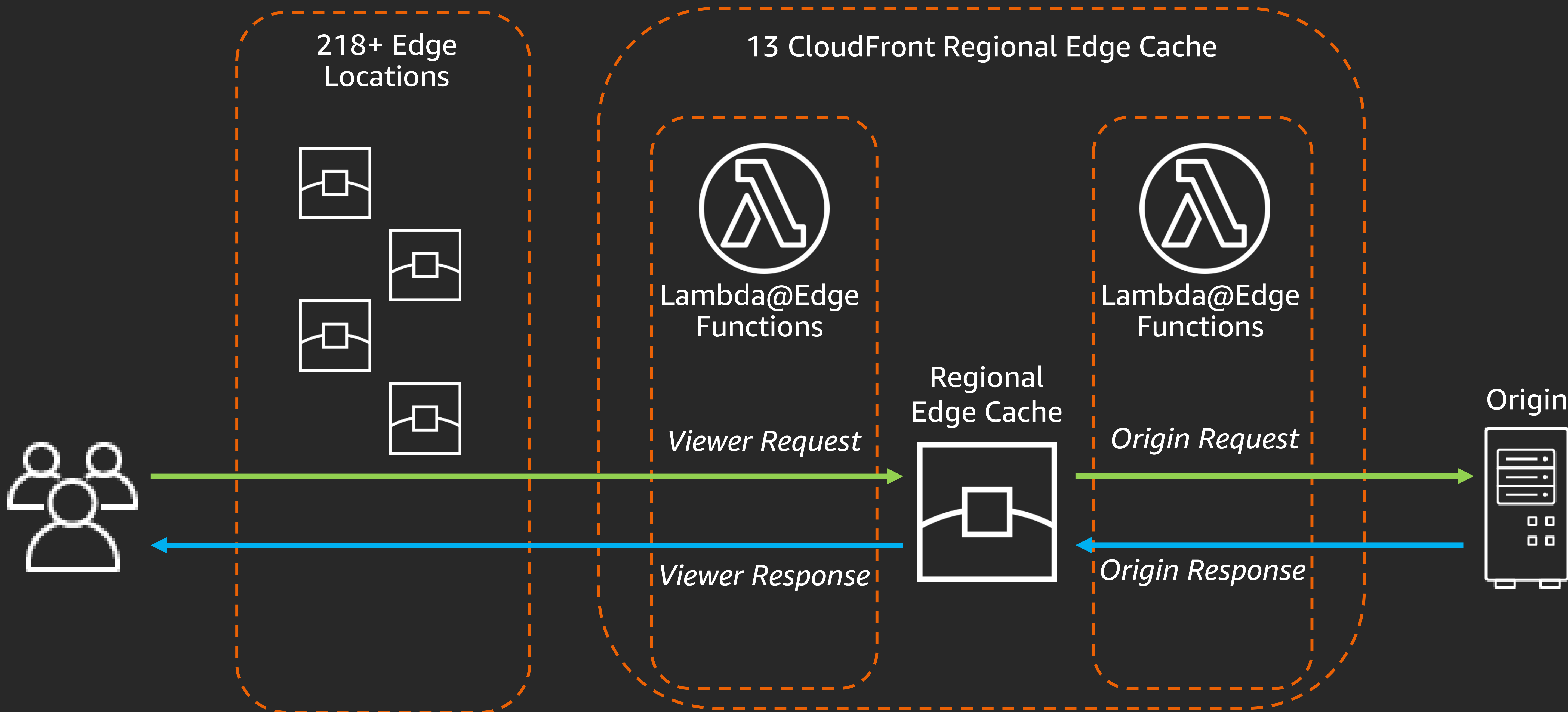
CloudFront Functions best practices

Lambda@Edge best practices

Introduction to Lambda@Edge and CloudFront Functions

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lambda@Edge execution

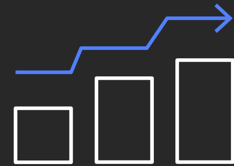


Introducing CloudFront Functions

New purpose-built serverless scripting feature for running lightweight JavaScript code at the 218+ CloudFront edge locations

Ultra Performant

Adds no perceptible latency to requests

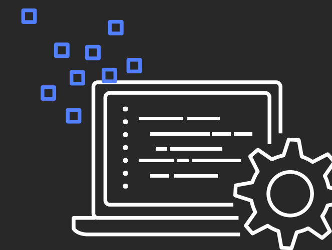


Instantly Scalable

Handle millions of requests per second

Cost Effective

Fraction of Lambda@Edge price



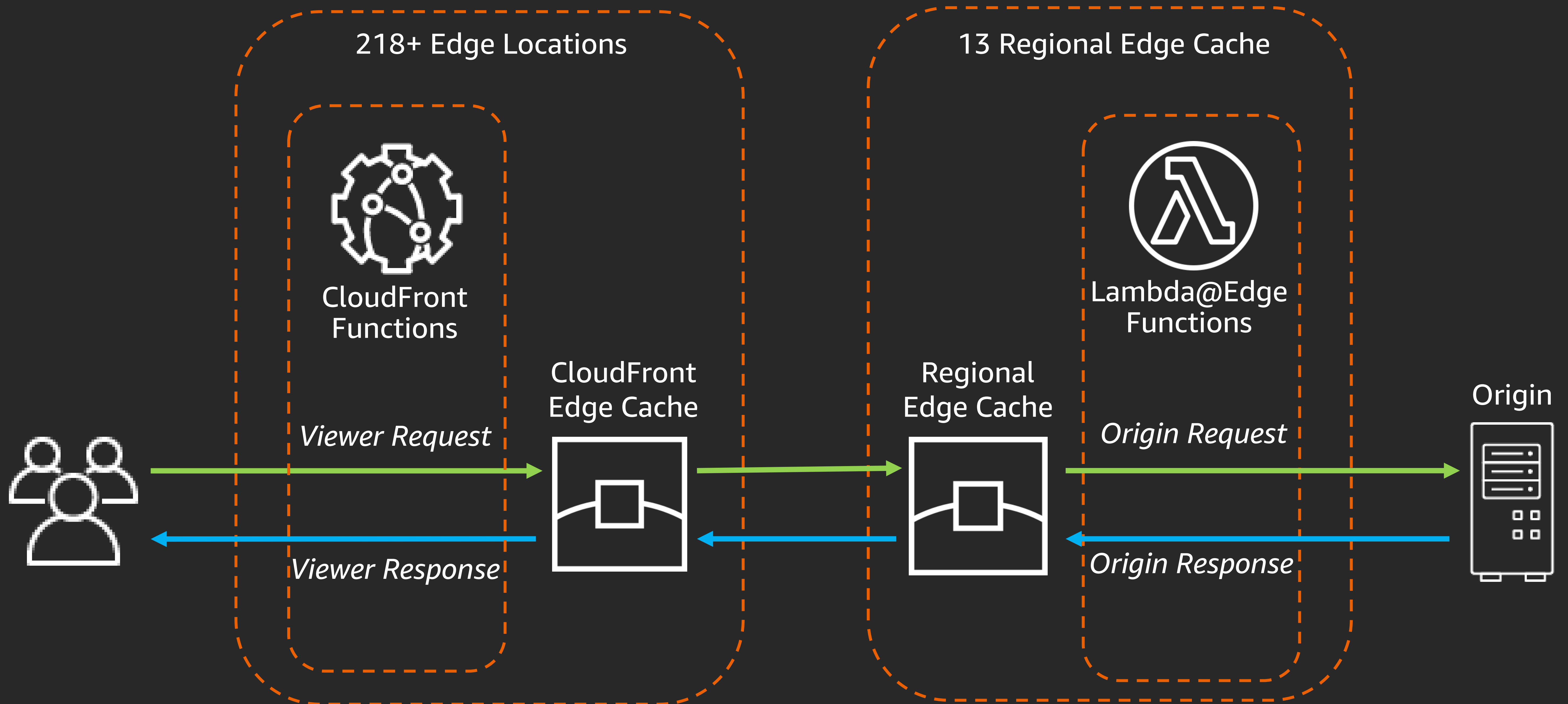
Developer Friendly

Streamlined workflow and APIs

Highly Secure

Uses the highest security standards

CloudFront Functions execution



Choosing between CloudFront Functions and Lambda@Edge

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

CloudFront Functions use cases

Ideal for high scale workloads

- **Cache key normalization** - Transform HTTP request attributes (URL, headers, cookies, query strings) to construct CloudFront cache key in a more optimal way, leading to an improved cache hit ratio.
- **Header manipulation** - Insert, modify or delete any HTTP headers (e.g. True-Client-IP, CORS, or HSTS headers).
- **URL redirects/rewrites** - Redirect users to other pages or seamlessly direct requests to different paths on the origin server.
- **Request authorization** - Create and validate user generated tokens, such as HMAC tokens or JSON web tokens (JWT).

Not ideal for complex workloads

- **Long running** - Workloads that take several milliseconds to seconds to complete.
- **Adjustable Memory or CPU** – Workloads that require large CPU or memory footprint.
- **Dependency on 3rd party libraries** – Including the AWS SDK which is required for integrations with other AWS services (e.g., S3, DynamoDB).
- **Networks calls** – Workloads that need to call external services or end points for data processing.

Continue to use Lambda@Edge for these types of workloads

CloudFront Functions vs. Lambda@Edge

	CloudFront Functions	Lambda@Edge
Runtime support	JavaScript (ECMAScript 5.1 compliant)	Node.js, Python
Execution location	218+ CloudFront Edge Locations	13 CloudFront Regional Edge Caches
CloudFront triggers supported	Viewer request Viewer response	Viewer request Viewer response Origin request Origin response
Maximum execution time	Less than 1 millisecond	5 seconds (viewer triggers) 30 seconds (origin triggers)
Pricing	Free tier available; charged per request	No free tier; charged per request and function duration

CloudFront Functions vs. Lambda@Edge

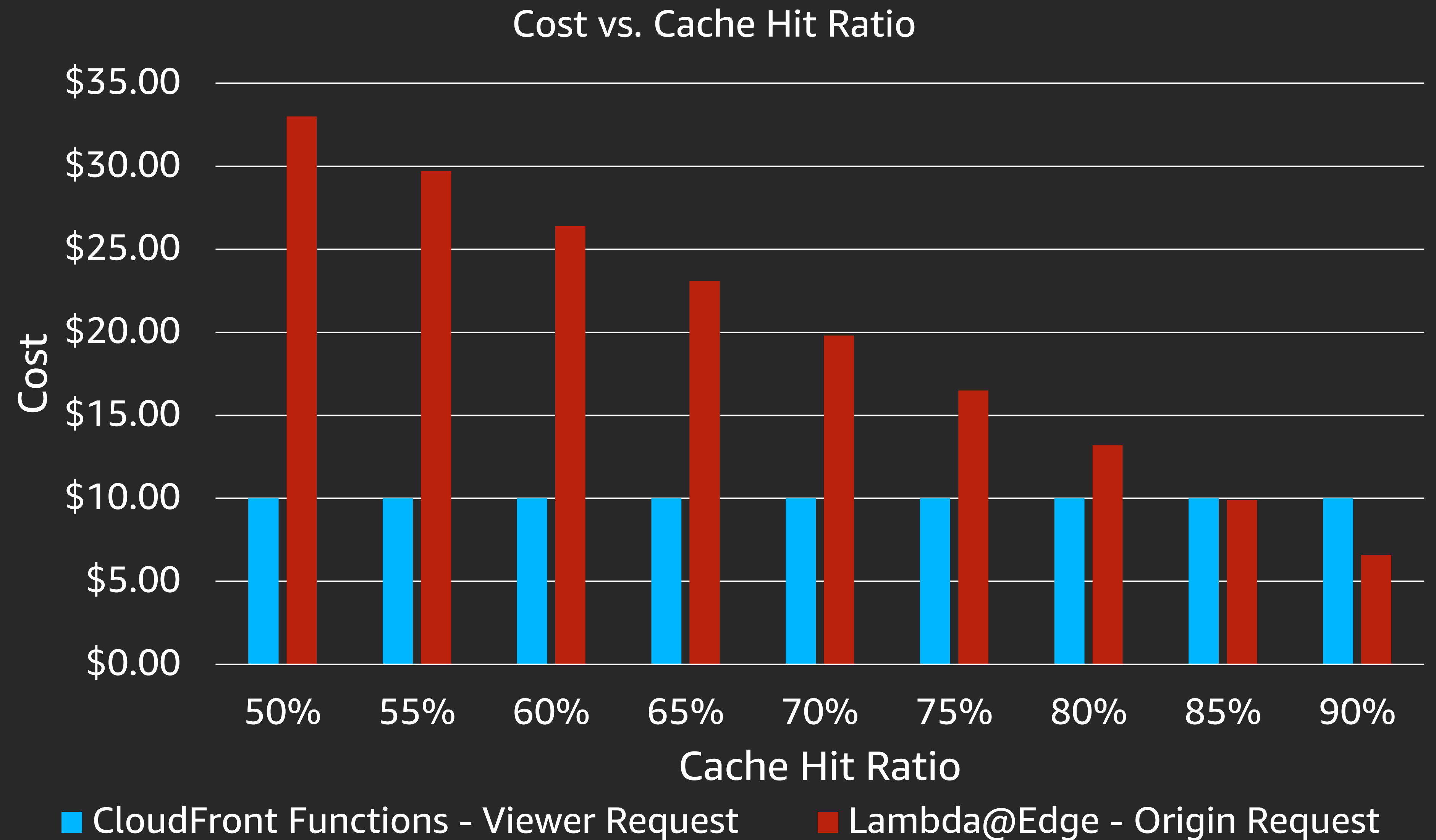
	CloudFront Functions	Lambda@Edge
Maximum memory	2MB	128MB (viewer triggers) 10GB (origin triggers)
Total package size	10 KB	1 MB (viewer triggers) 50 MB (origin triggers)
Network access	No	Yes
File system access	No	Yes
Access to the request body	No	Yes

Don't choose based on price alone

Assumptions:

- 100 million requests per month
- 10ms duration on Lambda@Edge
- Function could run as viewer request or origin request

If your cache hit ratio is high, Lambda@Edge may be more cost effective

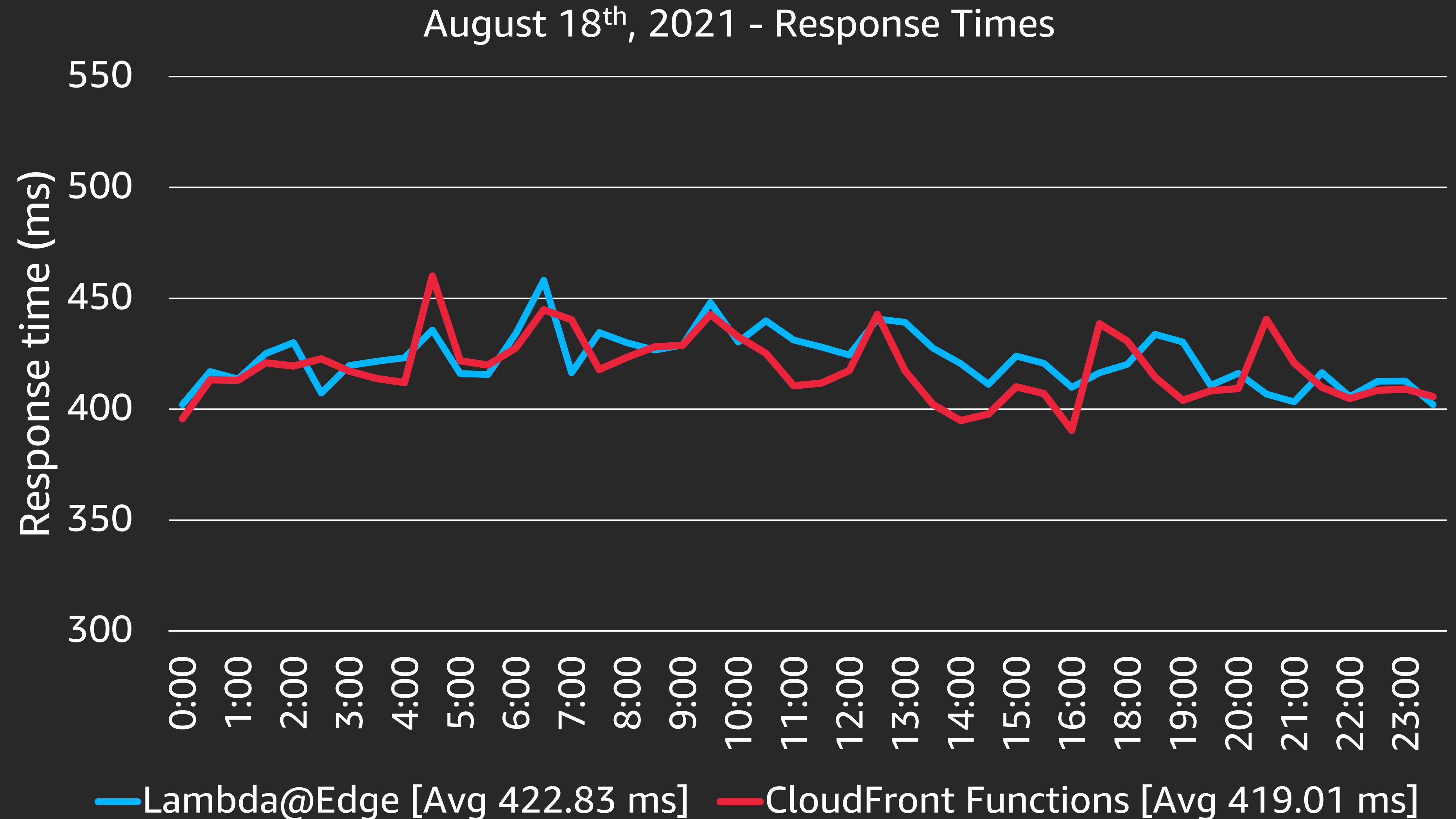


Don’t choose based on perceived performance

Assumptions:

- Function could run as viewer request or origin request
- Dynamic content (i.e., no caching)

On cache misses, Lambda@Edge and CloudFront Functions have similar performance



When to use Edge Computing

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

When to use Edge Computing

- Stateless logic
- Logic needs to run on every request
- Performance is paramount
- Origin scaling is a challenge

Simple HTTP manipulations	Dynamic content generation	Origin independence
User-Agent header normalization	Image manipulation	Pretty URLs
Adding security headers	Render pages	API wrapper
Enforcing Cache-Control headers	Redirections	User Authorization
A/B testing	SEO optimization	Bot mitigation

When to use Edge Computing

Just because you can move something to edge doesn’t mean you should

The edge is always going to be a constrained environment:

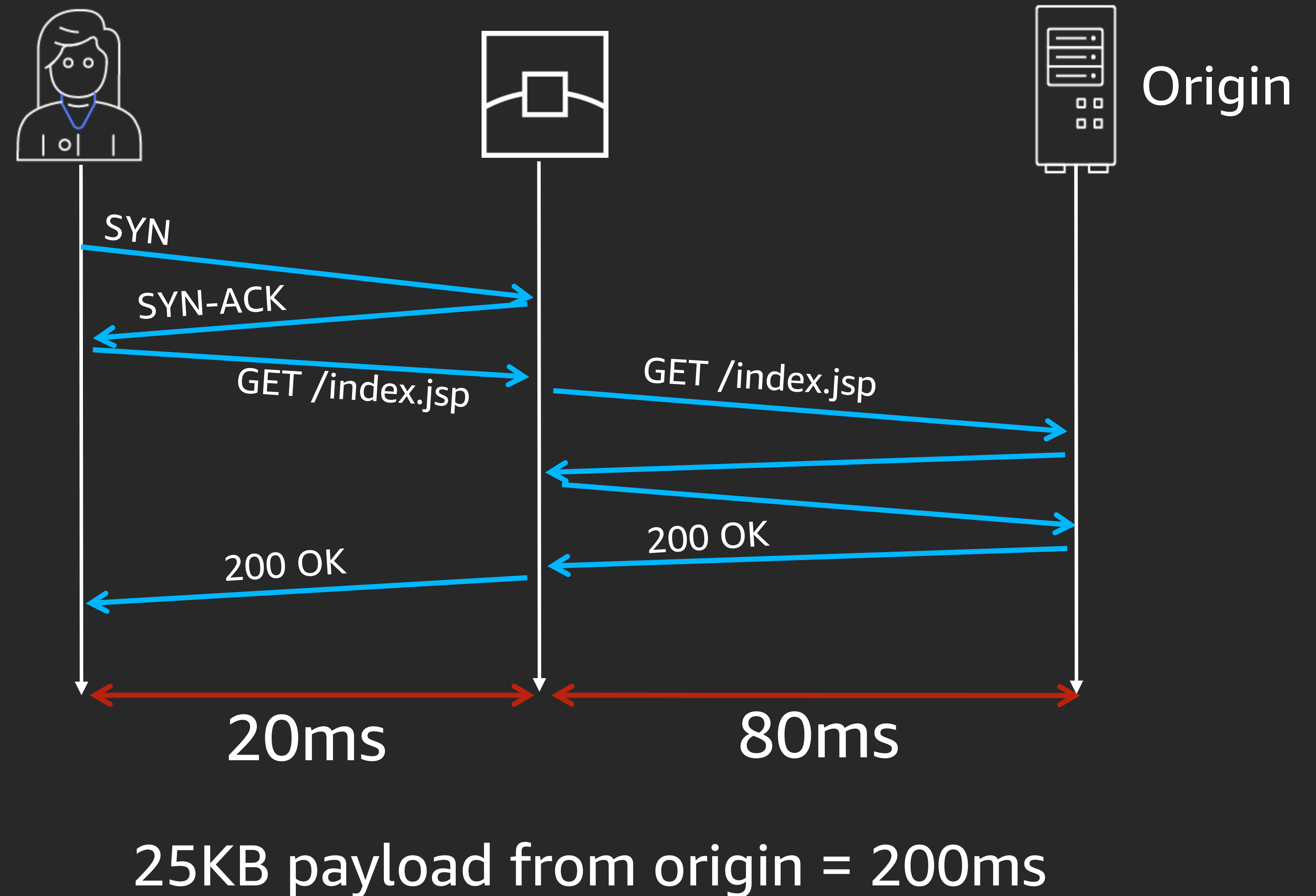
- Edge Compute power will always be less than an AWS Region
- Maintaining state will always be more difficult than in an AWS Region
- Edge Compute will add complexity to builds, deployments, and monitoring

Use case: making network calls

Use Case: Your site uses server side rendering (SSR) to construct a page on the fly.

The SSR calls an API service located in a single AWS Region to fetch 25KB of data to construct the page.

Should you use Edge Computing?



Use case: making network calls

Can you use caching within the function?

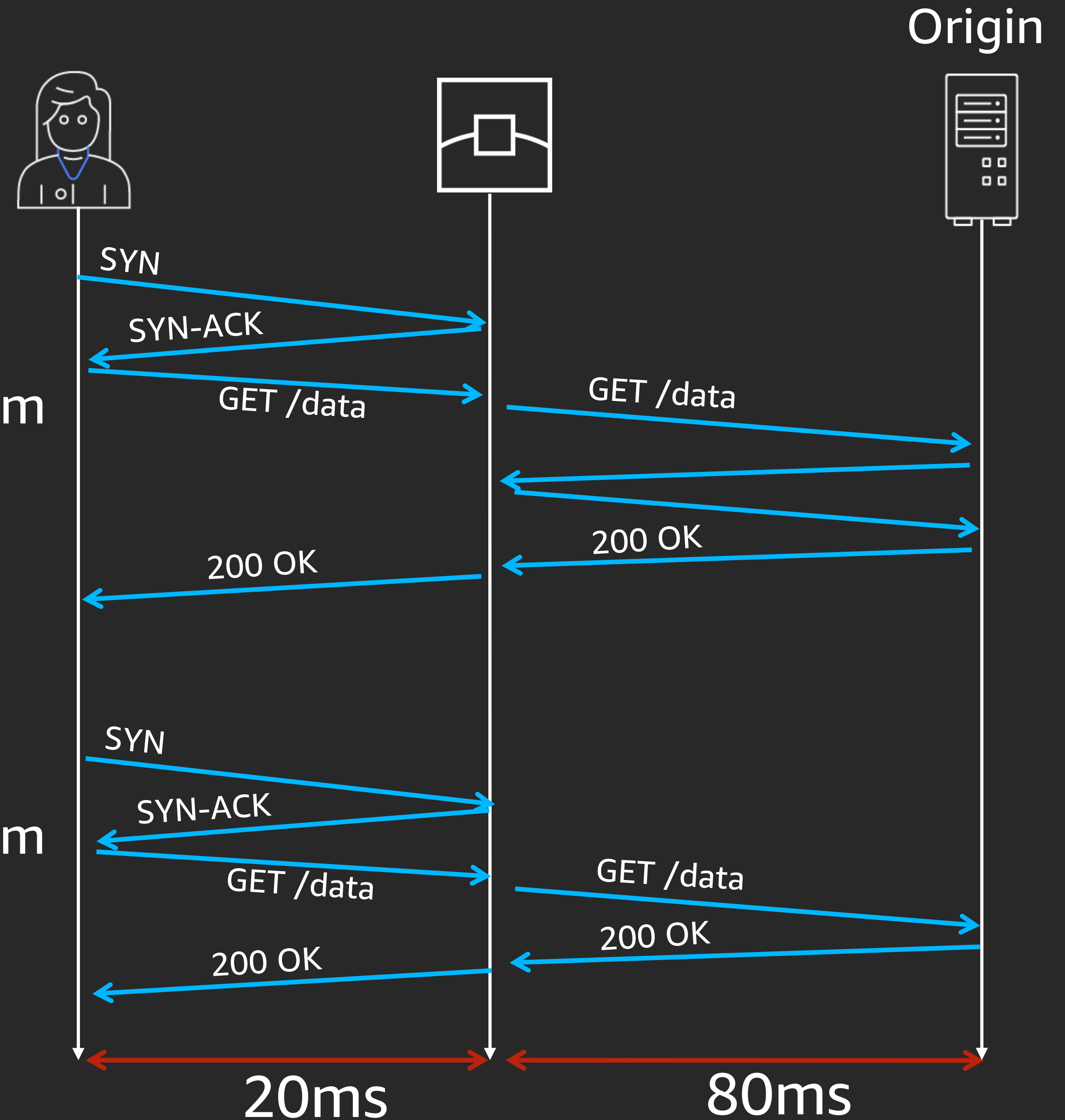
- How frequently does the data change?
- How many different responses will I get?
- How many requests for the function will I get?

```
let s3data;

function fetchData() {

  if (!s3data) {

    s3data = fetchFromS3();


    setTimeout(() => {

      s3data = undefined;

      }, 300000;)} // TTL of 5 minutes


    return redirections;

}
```

Use case: making network calls

Can you reduce the payload to under 15KB?

- TCP's Initial congestion window is 10
- Maximum transmission unit (MTU) ~1500 bytes
- Maximum data transferred in a single round trip ~15KB

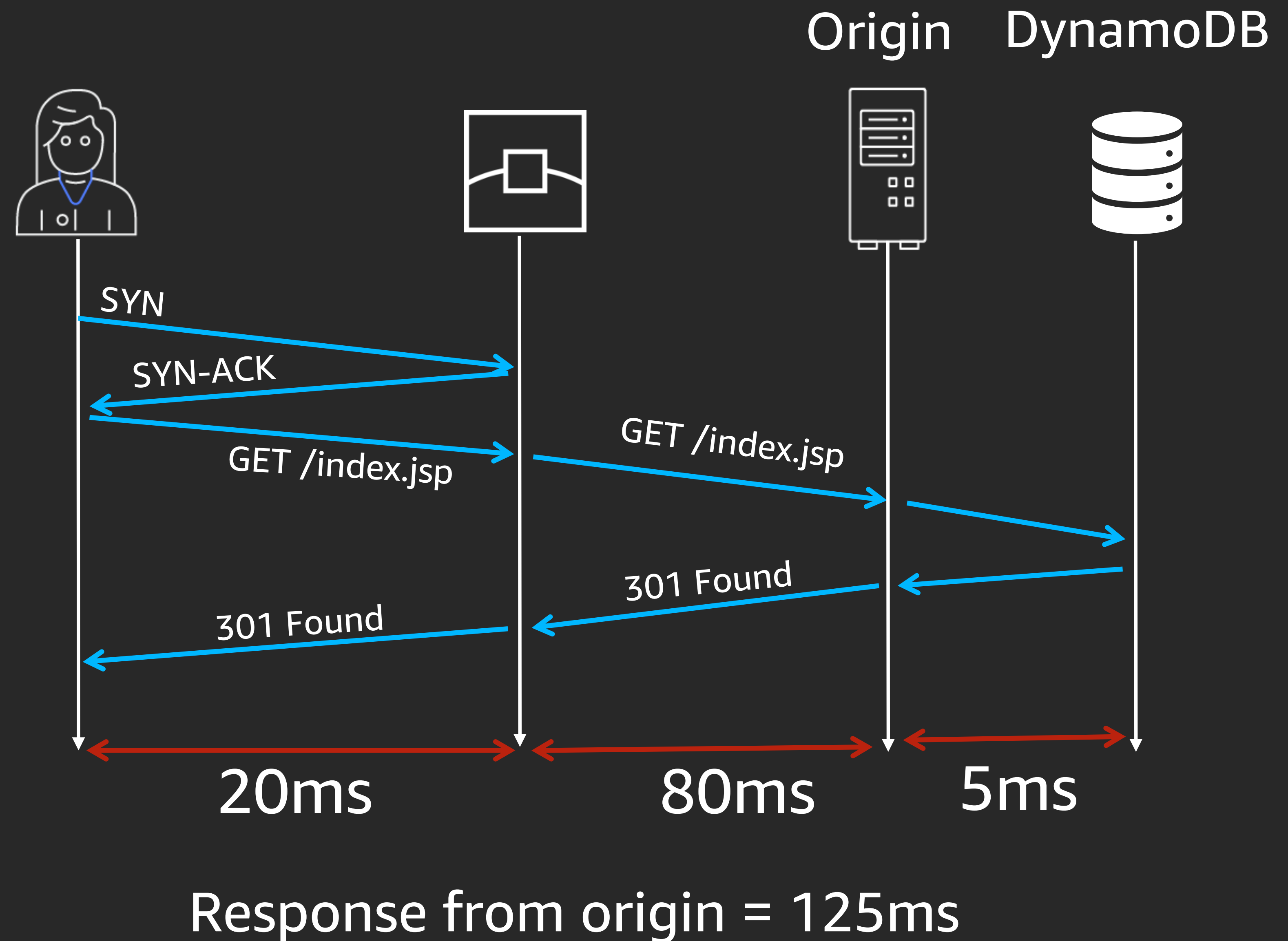


Use case: URL Redirects

Use Case: You manage a over 3,000 old URLs that should redirect to updated URLs.

To manage this you store the redirect mapping inside a DynamoDB table in a single AWS Region.

Should you use Edge Computing?



Use case: URL Redirects

Can you use DynamoDB Global Tables?

- How often are these URLs requested?
- How much does this really reduce latency?
- How much cost does this add to your overall architecture?

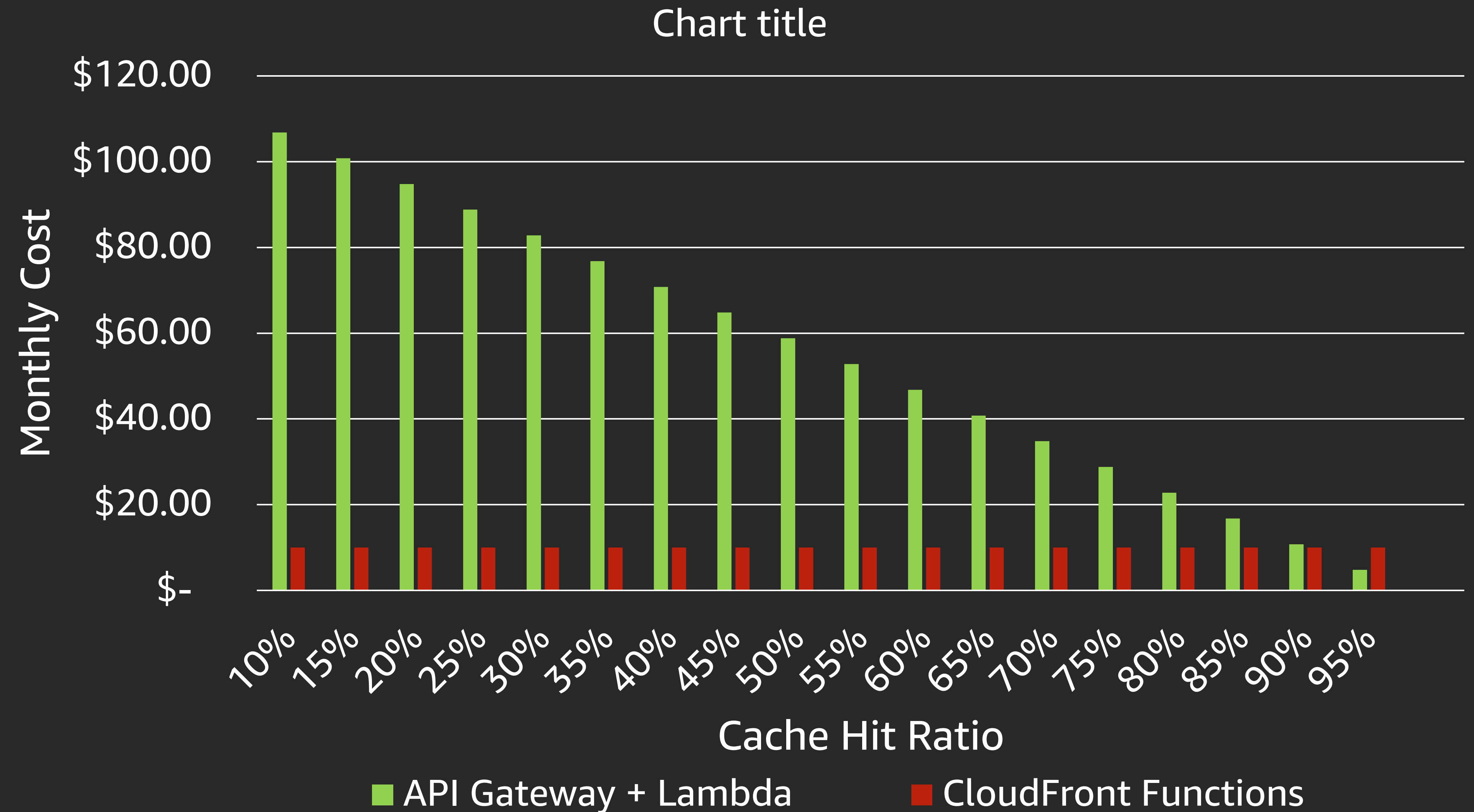
```
const AWS_REGION = process.env;
const replicatedRegions = {
        'us-east-1': true,
        'us-east-2': true,
        'us-west-2': true,
        'eu-west-2': true,
        'eu-central-1': true
};

const documentClient =
       new aws.DynamoDB.DocumentClient({
       region: replicatedRegions[AWS_REGION] ?
       AWS_REGION : 'us-east-1',
       httpOptions: {
          agent: new https.Agent({
               keepAlive: true})
       }
});
```

Use Case: URL Redirects

Assumptions:

- 100 million requests per month
- Origin uses API Gateway + Lambda to serve redirects
- API Gateway uses HTTP APIs
- 128 MB function with avg. 5 ms duration on Lambda

Caching is your friend, use it as often as possible. CloudFront can cache 3XX status codes.



Use case: user authentication

Use Case: You want to protect an HLS video stream by securing the video segments with a security token.

Should you use Edge Computing?

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:5
#EXT-X-MEDIA-SEQUENCE:2680
#EXTINF:5,
fileSequence2680.ts?token=SflKxwRJSMeKKF2QT4fwpMeJf
#EXTINF:5,
fileSequence2681.ts?token=SflKxwRJSMeKKF2QT4fwpMeJf
#EXTINF:5,
fileSequence2682.ts?token=SflKxwRJSMeKKF2QT4fwpMeJf
#EXTINF:5,
fileSequence2683.ts?token=SflKxwRJSMeKKF2QT4fwpMeJf
#EXTINF:5,
fileSequence2684.ts?token=SflKxwRJSMeKKF2QT4fwpMeJf
```

Use case: user authentication

Do you need to make a network call to validate?

- If I fetch the public key, can I cache it?
- Is the rest of validation logic stateless?
- How many requests for the function will I get?

```
{
    "keys": [{
        "alg": "RS256",
        "e": "AQAB",
        "kid": "abcdefghijklmnopqrsexample=",
        "kty": "RSA",
        "n":"lsjhglskjhgslkjgh43ljexample",
        “use": "sig"
    }, {
        "alg": "RS256",
        "e": "AQAB",
        "kid": "fgjhlkhjlkhexample=",
        "kty": "RSA",
        "n": "sgjhlk6jp98ugp98up34hpexample",
        "use": "sig"
    }]
}
```

Use case: user authentication

Is the validation stateless?

- Is the entire validation logic stateless?
- Is this validation required on every request?

If the validation is stateless and required on every request, edge computing is a good option

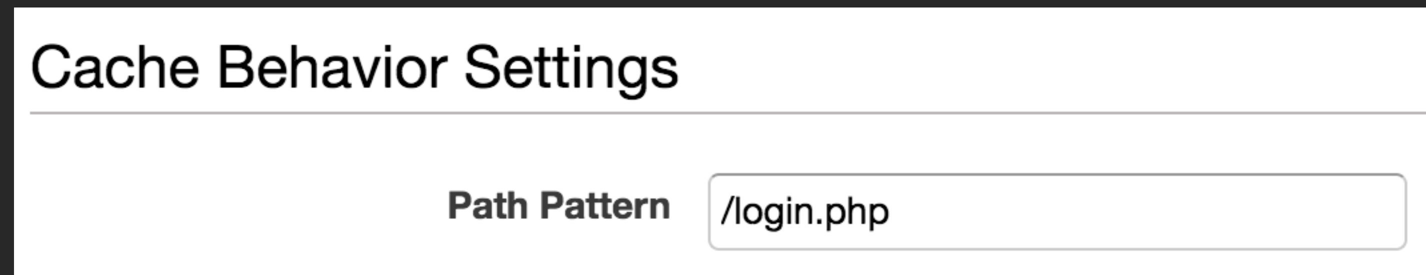
```
header:
    {
        "alg" : "HS256",
        "typ" : "JWT"
    }
payload:
    {
        "sub": "1234567890",
        "name": "John Doe",
        "iat": 1516239022
    }
secret: MySuperSecretKet
```

CloudFront Functions best practices

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Invoke functions when you need it

- For every request or only on cache misses?
- Use the most specific CloudFront behavior:



- Caching will always be faster than edge computing

Global variables aren't reused

```
var expires = 0;

function handler(event) {
    var req = event.request;
    var host = req.headers.host.value;
    var now = Date.now();


    if (now < expires)
        return req;

    expires = now + 30000;
    req.querystring['expired'] = true;
    return req;
}
```

```
function handler(event) {
    var req = event.request;
    var host = req.headers.host.value;
    var now = Date.now();
    var expires = 0;

    if (now < expires)
        return req;

    expires = now + 30000;
    req.querystring['expired'] = true;
    return req;
}
```

Regular expression is expensive

```
function handler(event) {
    var req = event.request;
    var host = req.headers.host.value;
    var HOST_REGEX = /.*\.example.com$/g;

    if(HOST_REGEX.test(host))
        return req;

    req.headers.host.value = 'example.com';
    return req;
}
```

```
function handler(event) {
    var req = event.request;
    var host = req.headers.host.value;


    if(host.includes('example.com')
        return req;

    req.headers.host.value = 'example.com';
    return req;
}
```

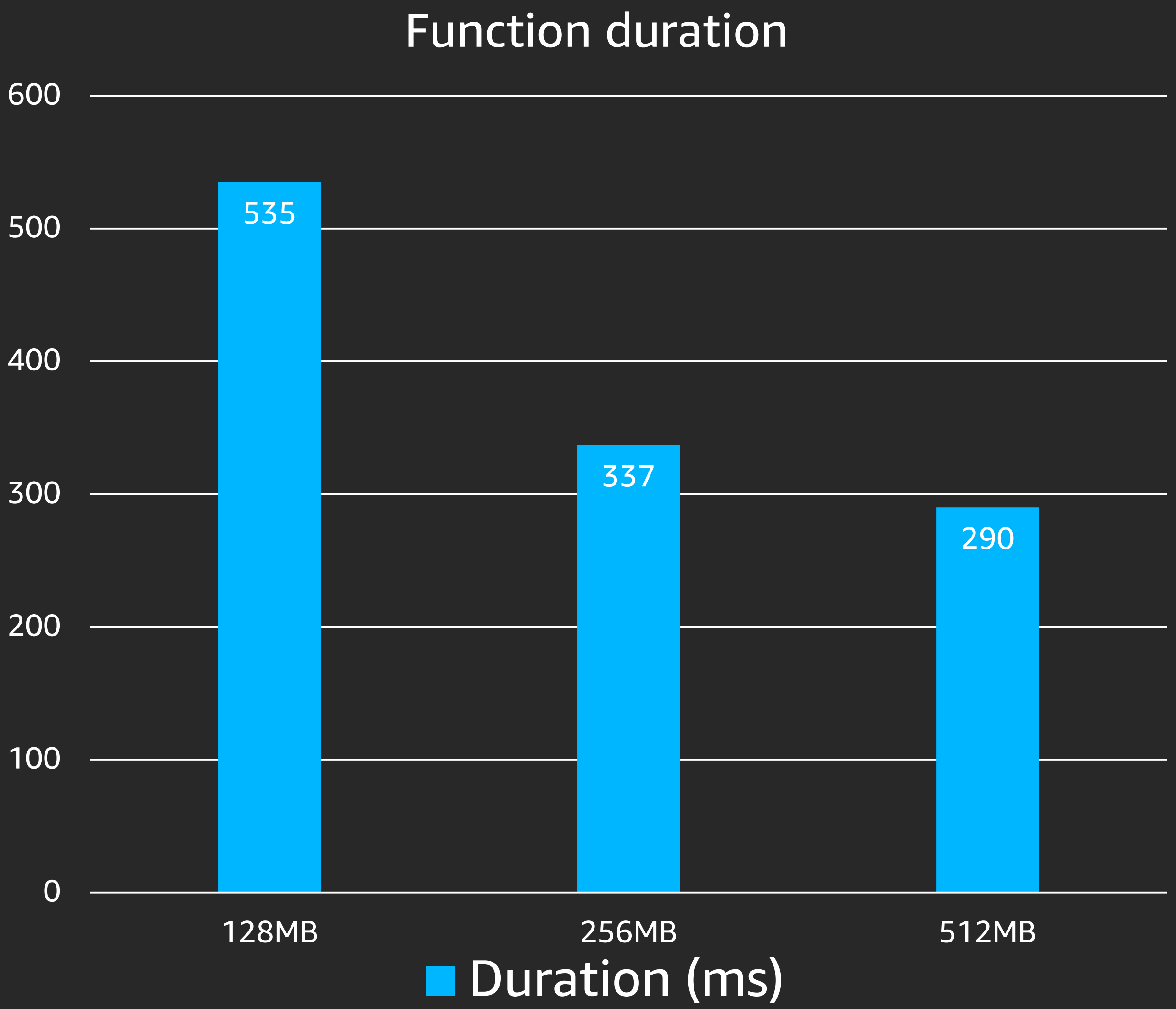
Don't let limits limit you

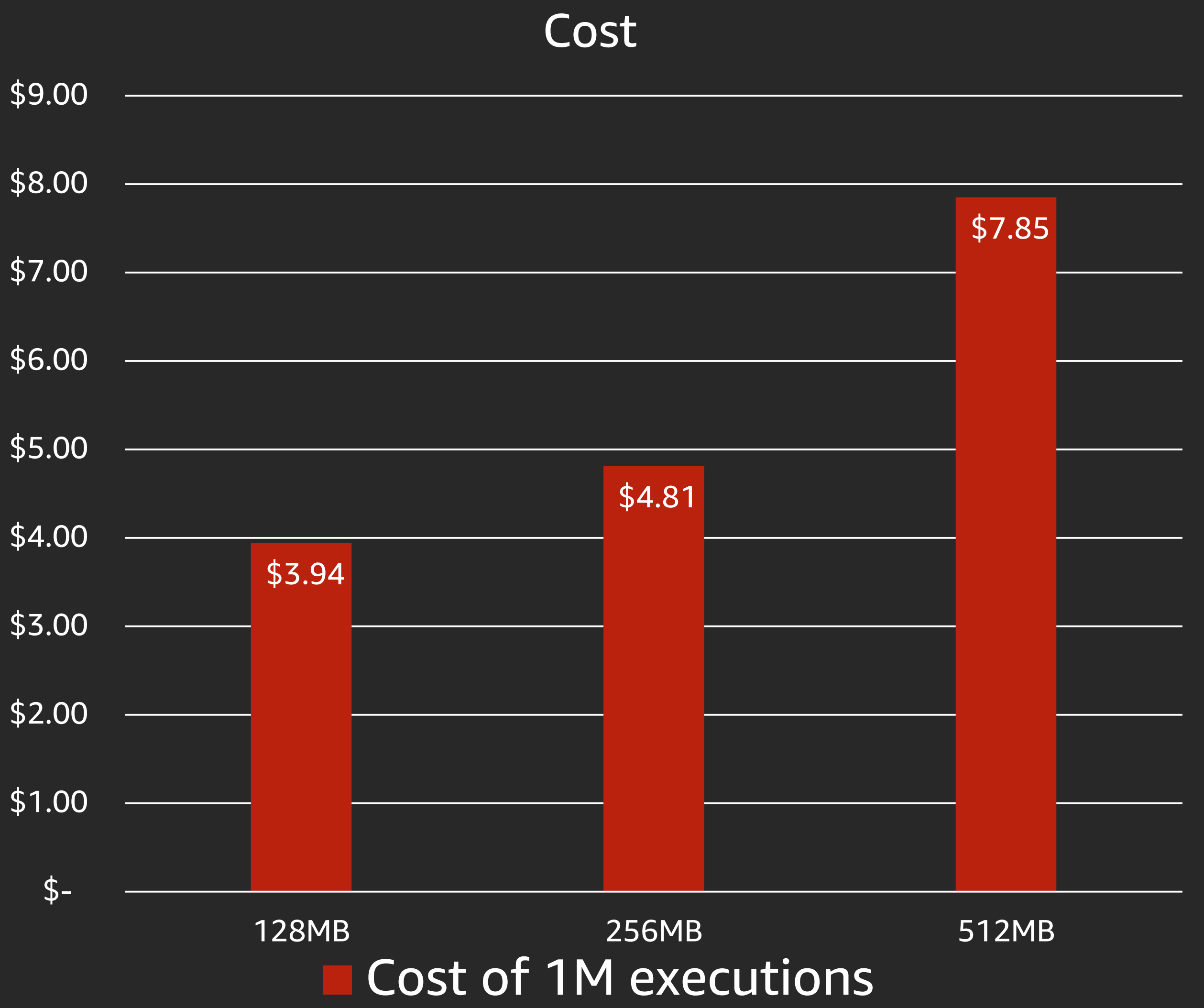
- The >1ms execution limit is CPU clock time NOT wall clock time
  - That's still 2.5 million operations per 1ms of CPU clock time!
- The 10KB function size limit is total size, don't forget to minify
  - You can can easily fit 20KB or more into function if you minify
- When it doubt, test
  - Compute utilization is your friend, use it when testing

Lambda@Edge best practices

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Choose the optimal memory configuration





Use global variables

```
const dns = require('dns');

let bestOrigin;
let expires = 0;

exports.handler = async (event) => {
  let req = event.Records[0].cf.request;

  getBestOrigin().then((origin) => {
    req.origin.domainName = origin;
    req.headers.host[0].value = origin;
    return req;
  });
}
```

```
function getBestOrigin() {
  const now = Date.now();

  if (now < expires)
    return Promise.resolve(bestOrigin);

  return new Promise((resolve, reject)=>{
    dns.resolveCname(DNS_HOST, (err,addr)=>{
      bestOrigin = addr[0];
      expires = now + 30000;
      resolve(bestOrigin);
    });
  });
}
```

Use parallelism and make async calls

```
let responses = await Promise.all([
        httpGet({ hostname: 'HTML template', path: '' }),
        ddbGet({ TableName: ddbTableName, Key: { name: 'mytable' } })
]);
```

Optimize external network calls

```
const http = require('https');
const keepAliveAgent = new http.Agent({ keepAlive: true, keepAliveMsecs: 2000 });
exports.handler = (event, context, callback) => {
        http.get({ hostname: 'hello.com', path: '/', agent: keepAliveAgent }, (resp) => {
                let data = '';
                resp.on('data', (chunk) => { data += chunk; });
                resp.on('end', () => { resolve(data); });
        });
}
```

Thank you!

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Please complete the session survey in the mobile app.

© 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved.