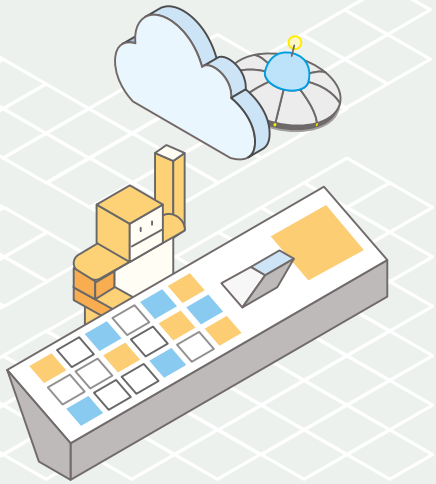
Lambda@Edge

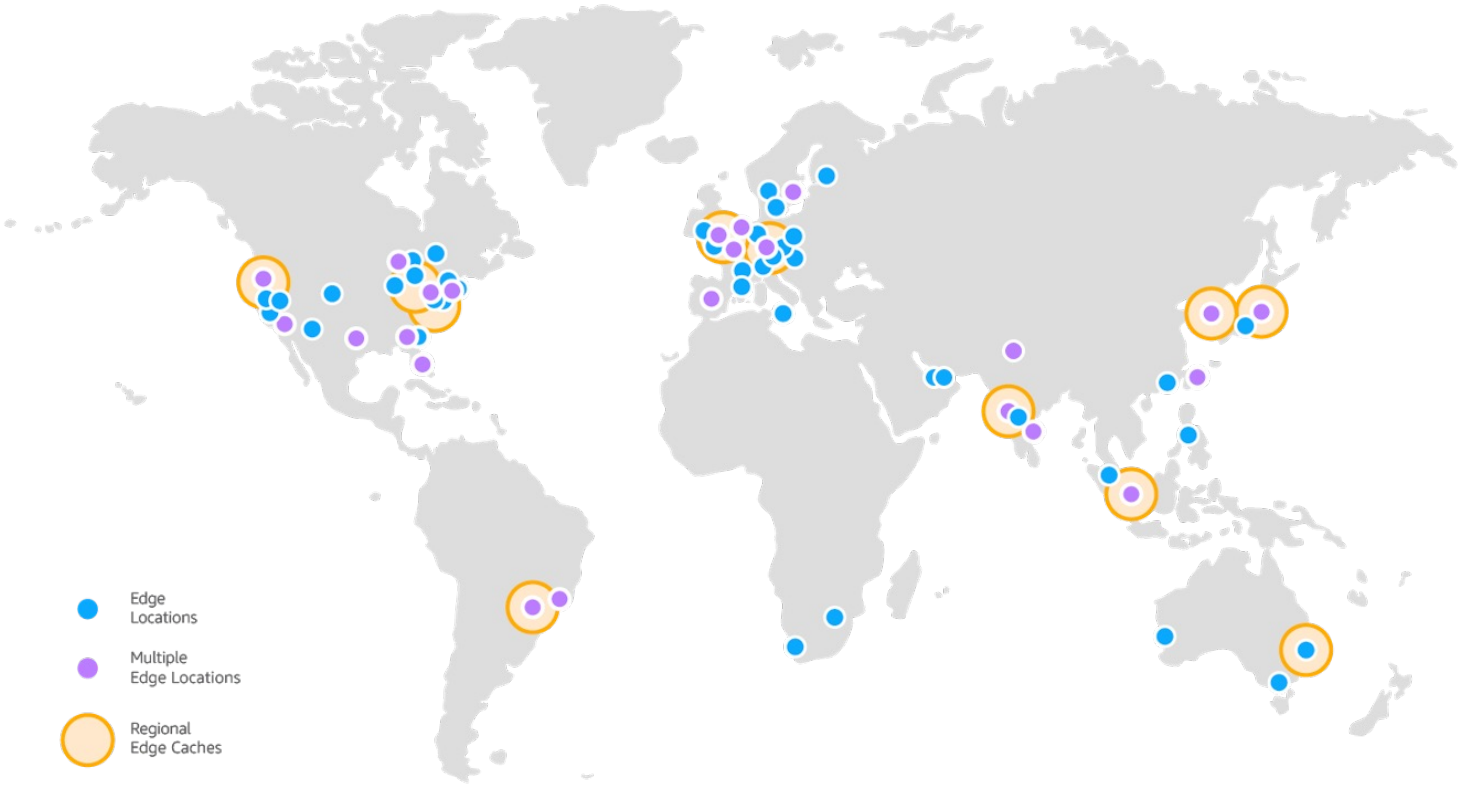
Customize your content delivery

Achraf Souk, Solutions Architect – Edge Services

17/10/2018



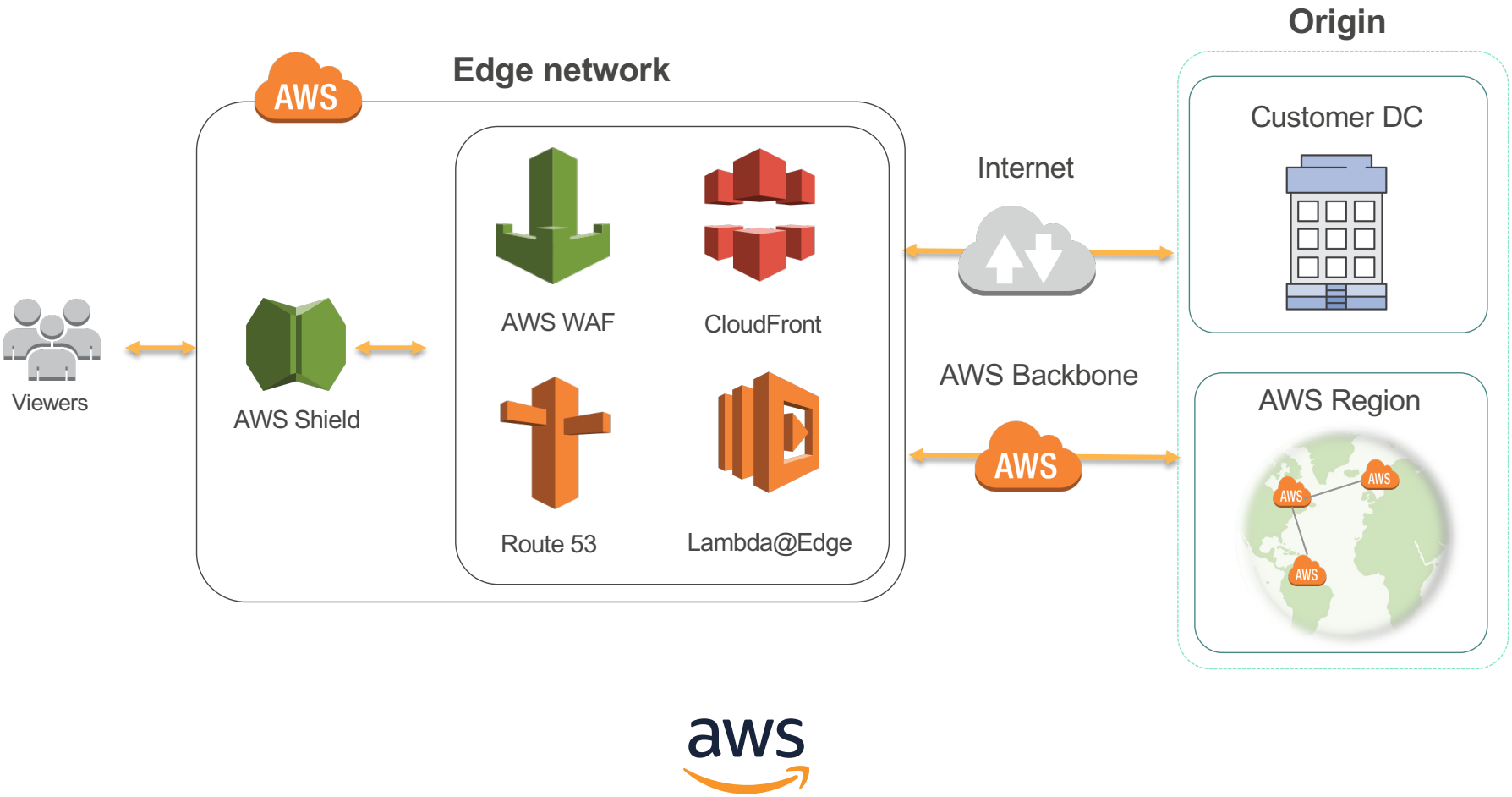
AWS Edge Network



138 PoPs
63 cities
29 countries

aws

Edge Services



aws

In the Nordics

- 3 PoPs in Stockholm, Sweden
- 1 PoP in Helsinki, Finland
- 1 PoP in Oslo, Norway
- 1 PoP in Copenhagen, Denmark

… and a region in Stockholm coming soon!

aws

Lambda is serverless

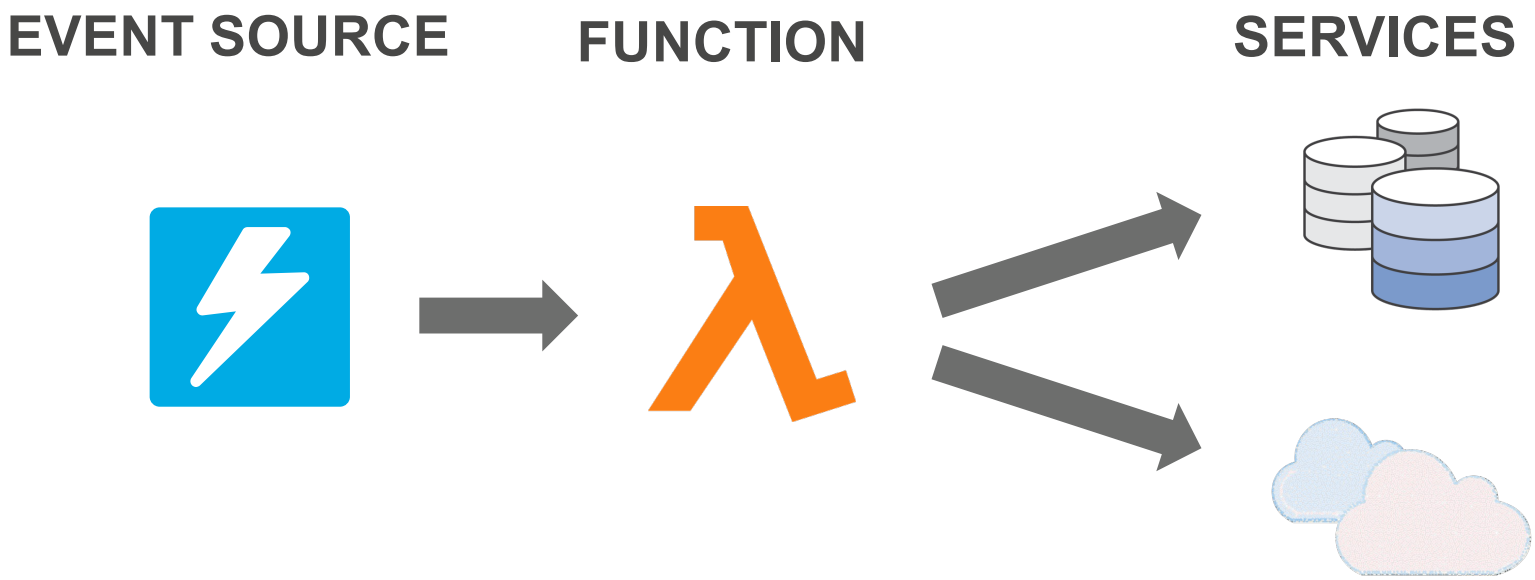
No servers to provision or manage

Scales with usage

Never pay for idle

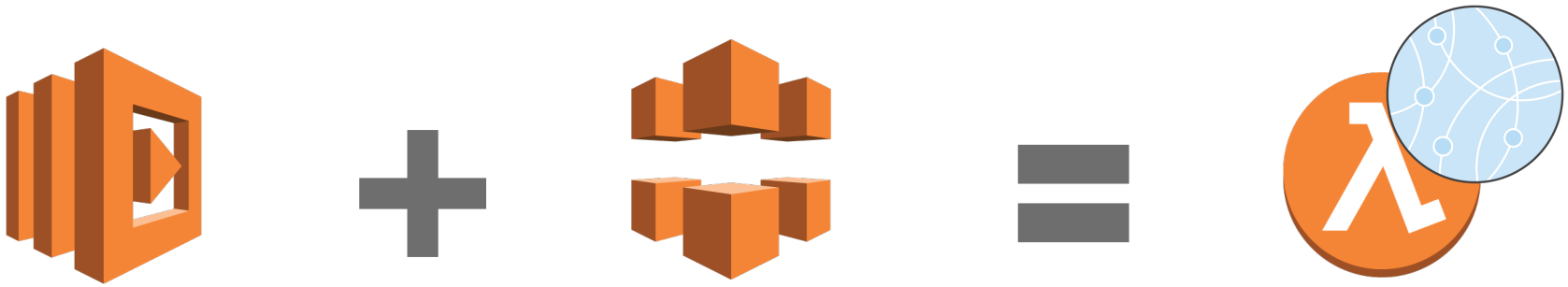
Built-in availability and fault tolerance

How it works



aws

Lambda@Edge is distributed Lambda



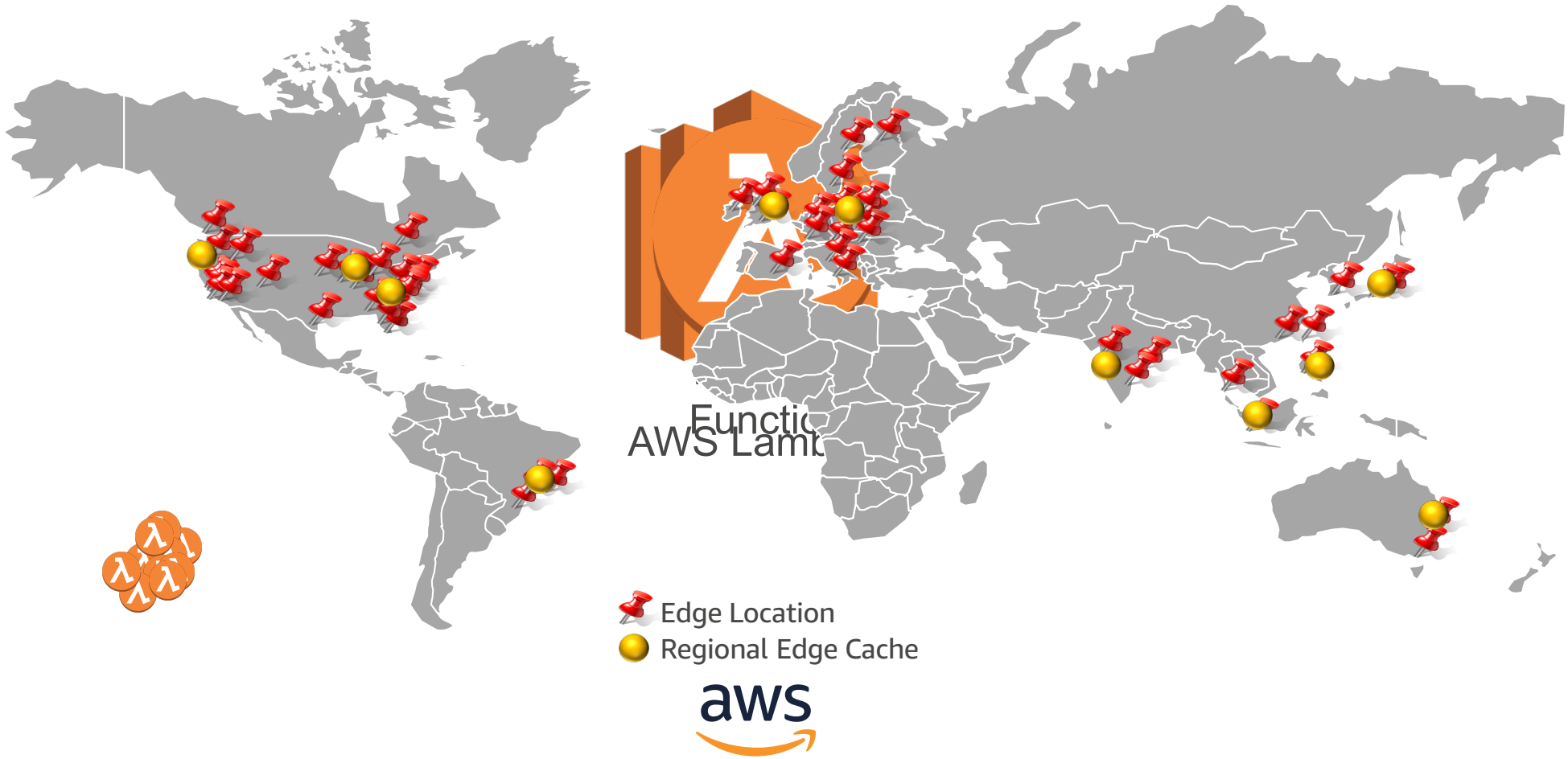
AWS Lambda

Amazon CloudFront

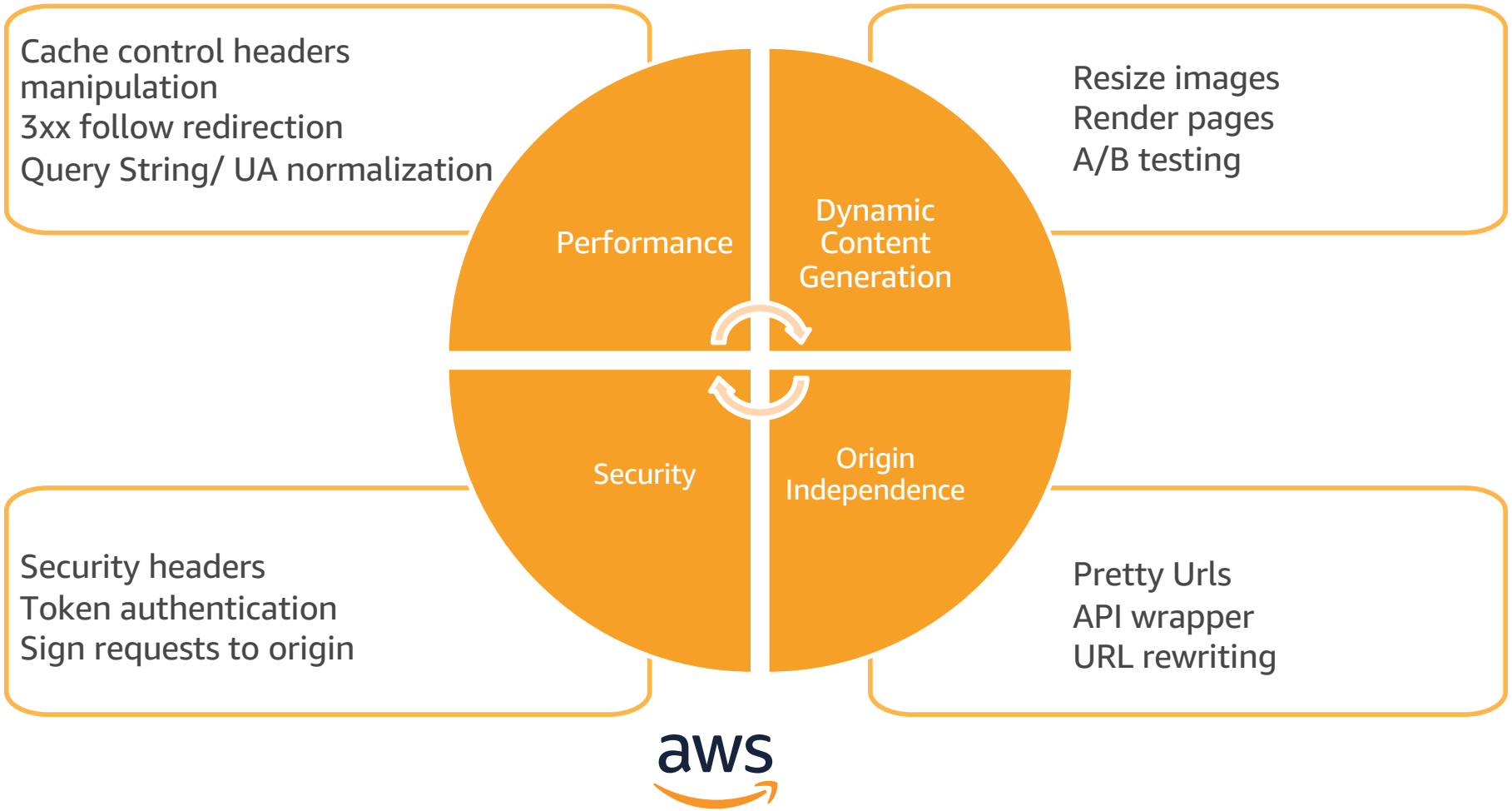
Lambda@Edge

aws

Write once, run Lambda functions globally



Use cases

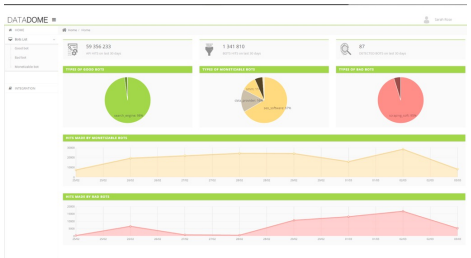

Cache control headers manipulation
3xx follow redirection
Query String/ UA normalization
Performance
Dynamic Content Generation
Resize images
Render pages
A/B testing
Security
Origin Independence
Security headers
Token authentication
Sign requests to origin
Pretty Urls
API wrapper
URL rewriting
aws

Bot management - Datadome

- DataDome is an SaaS cybersecurity solution for web and mobile applications that analyzes and manages non-human traffic in real time.

DATADOME

- DataDome uses Lambda@Edge to simplify the onboarding process for their customers, and to make their service available globally AWS edge locations.



aws

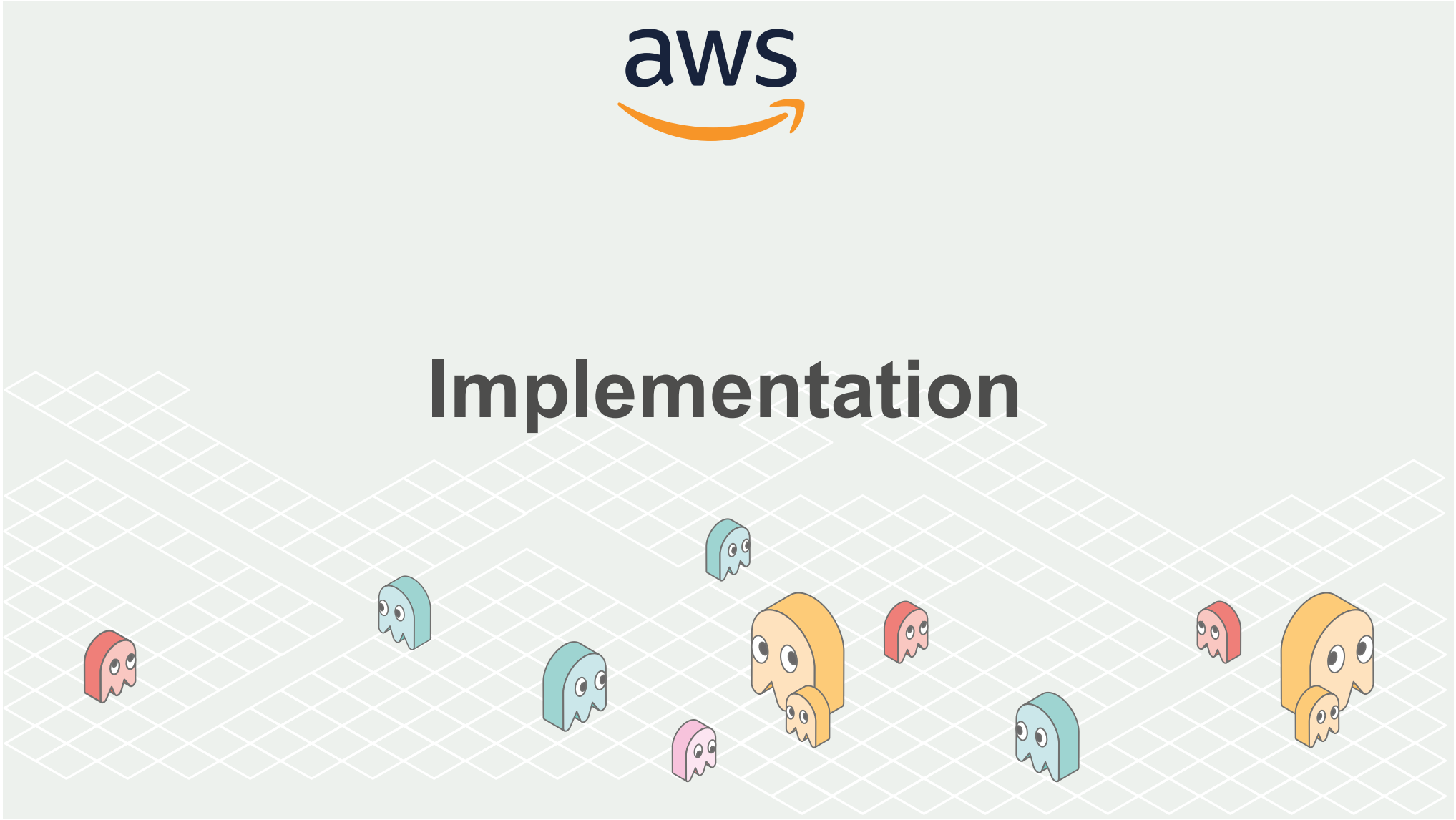
Video streaming - Blockbuster

- Blockbuster provides transactional video-on-demand services to viewers in the nordics with hundreds of thousands of subscribers in Denmark.

- Blockbuster used Lambda@Edge to solve Smart TV compatibility issues, by rewriting the manifest to a version understandable by these devices.

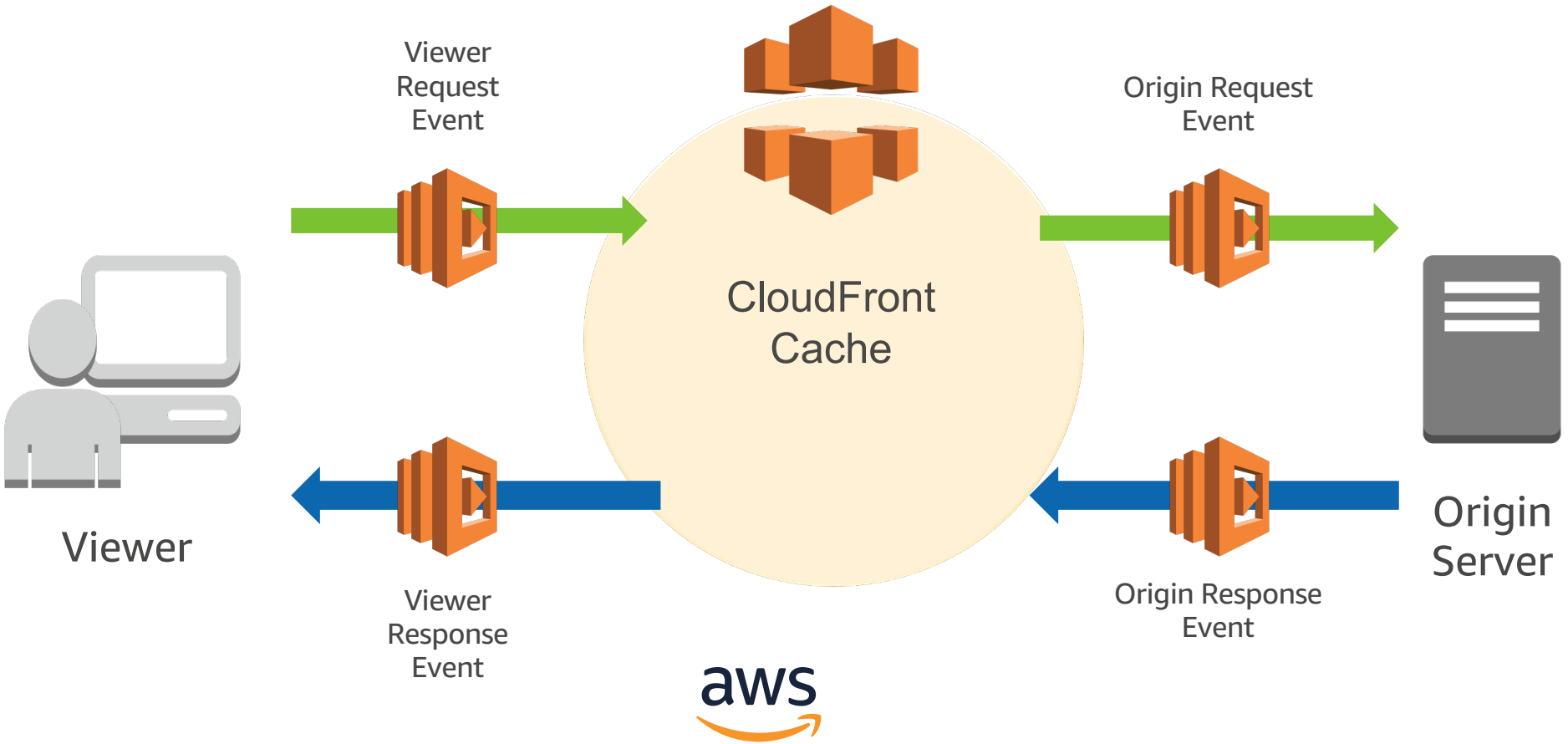
On AWS Blog

- Leveraging Lambda@Edge for AdTech: Cookie Syncing at the Edge
- Visitor Prioritization on e-Commerce Websites with CloudFront and Lambda@Edge
- Serving Private Content Using Amazon CloudFront & AWS Lambda@Edge
- Building a Serverless Subscription Service using Lambda@Edge for Publishers



Implementation

CloudFront triggers



Using Lambda@Edge

Author in N.Virginia

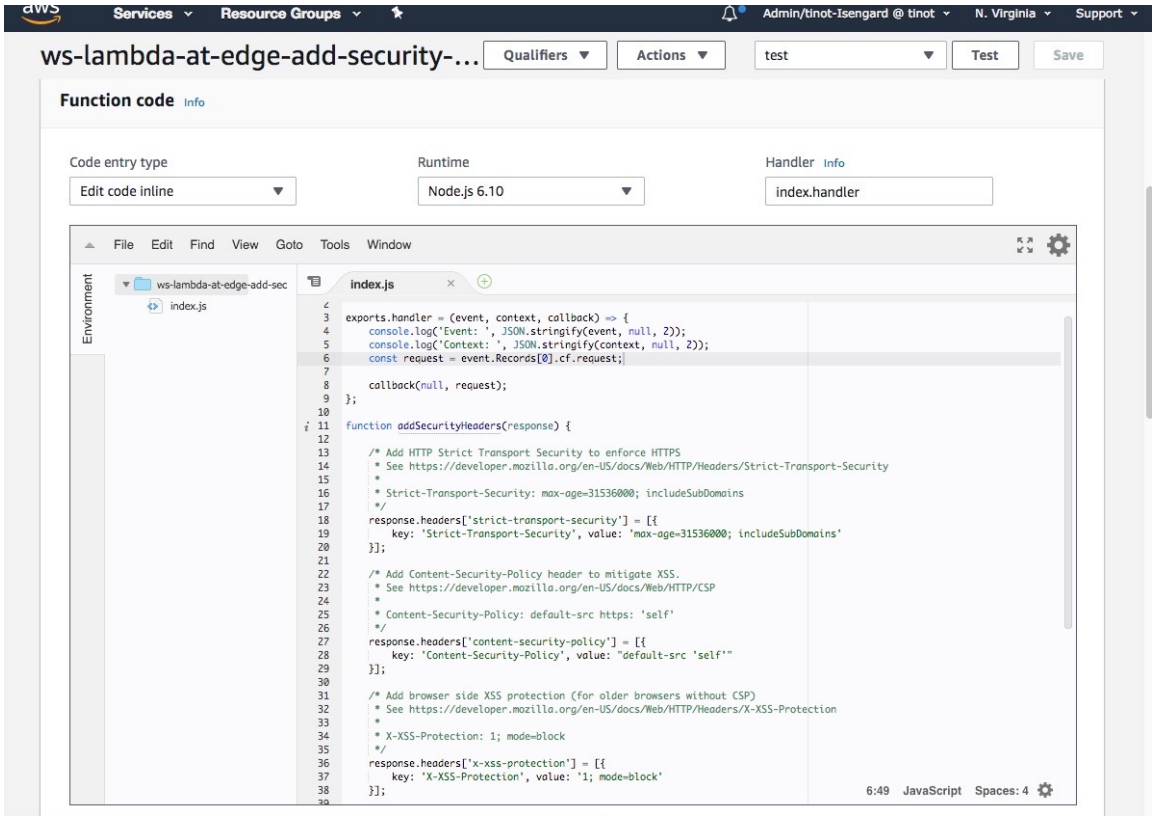
Lambda & Cloud9 IDE

NodeJS 6.10 & 8.10

CloudWatch Logs & Metrics

IAM integration

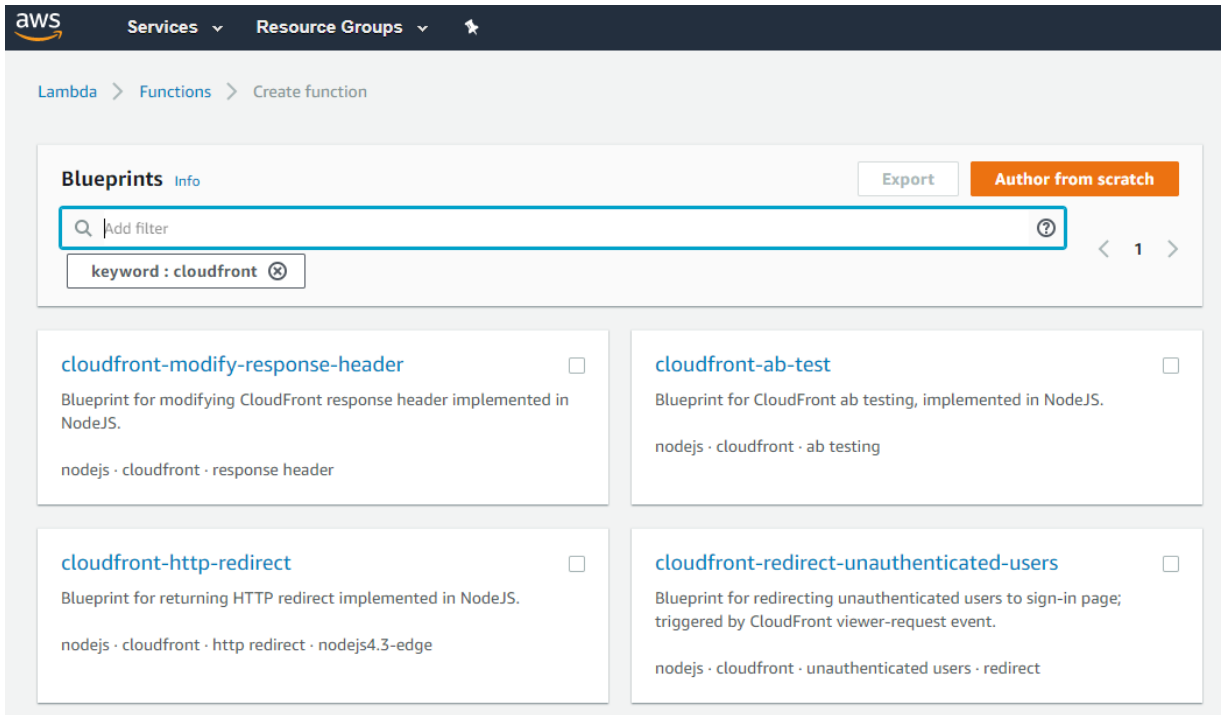
Versioning



aws

Quick start with blueprints

- Content customization based on requests attributes
- user authentication and session validation
- URL rewrites, pretty URLs
- A/B testing and cookie-based sticky sessions



Function structure

```
const LOCATION = 'http://docs.aws.amazon.com/lambda/latest/dg/lambda-edge.html'

exports.handler = (event, context, callback) => {

    const response = {
    status: '302',
    statusDescription: 'Found',
    headers: {
     location: [{
              key: 'Location',
              value: LOCATION,
              }],
        },
      };
     callback(null, response);
};
```

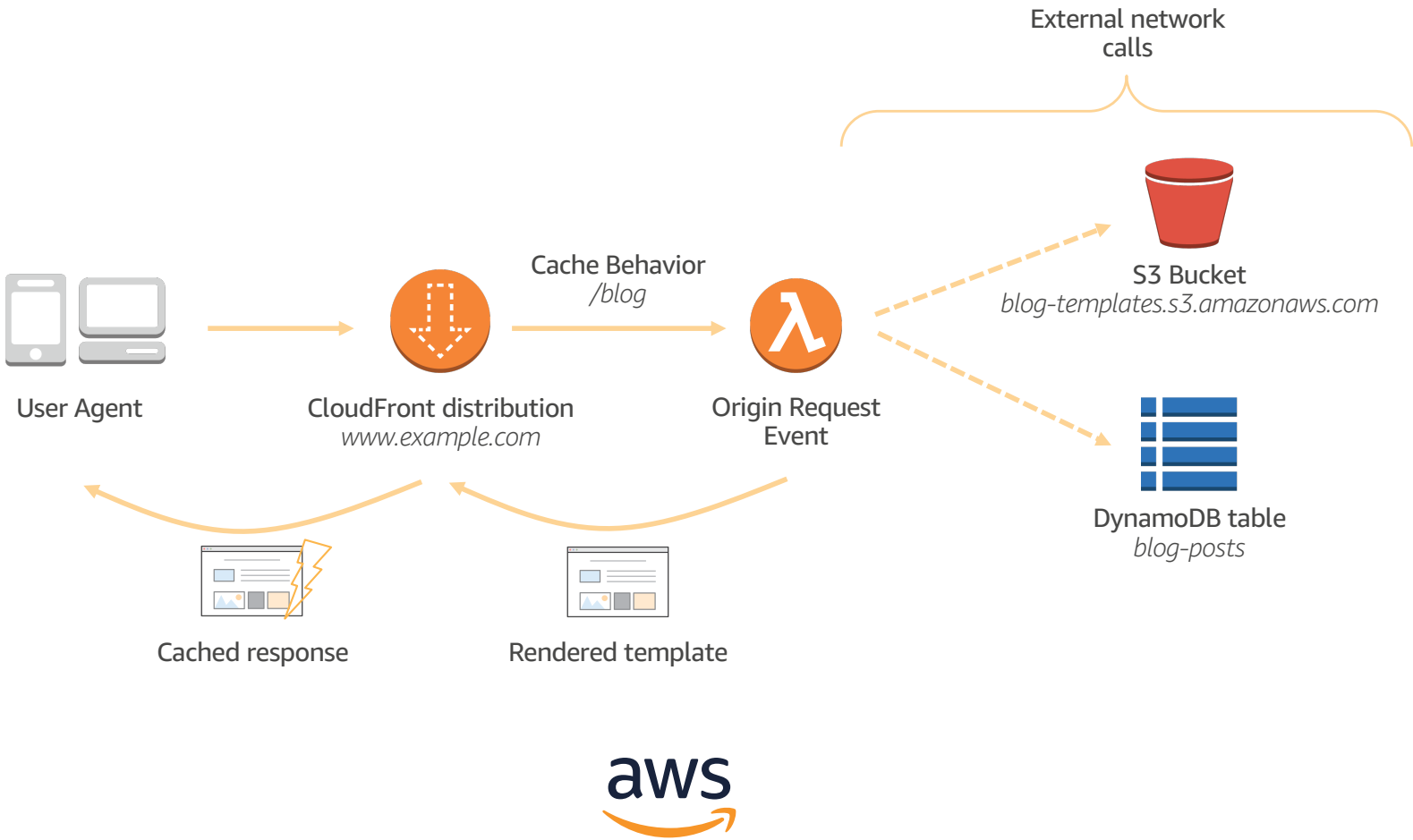
Dynamic content generation (1/3)

```
<h1>{ page.title }</h1>
{{ for section in page.sections }}
<h2>{ section.title }</h2>
<p>{ section.body }</p>
{{ endfor }}
```

```
"page": {
    "title": "Hello",
    "sections": [ {
        "title": "Introduction",
        "body": "The quick..."
    }, { ... } ]
```

aws

Dynamic content generation (2/3)



aws

Dynamic content generation (3/3)

```
const templateBucket = 'blog-templates-123456789012';
const postTable = 'blog-posts';

var AWS = require('aws-sdk');
var Mustache = require('mustache');

var s3 = new AWS.S3({region: 'us-east-1'});
var documentClient = new AWS.DynamoDB.DocumentClient({
    region: 'us-east-1'});

exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;
    const response = {
        status: '200',
        statusDescription: 'OK',
        headers: {
            'cache-control': [{
                key: 'Cache-Control',
                value: 'max-age=2628000, public'
            }],
            'content-type': [{
                key: 'Content-Type',
                value: 'text/html; charset=utf-8'
            }]}};
```

```
    const ddbParams = {
        TableName: postTable,
        Key: { slug: request['uri'].slice(1) }};

    documentClient.get(ddbParams, function(err, resp) {
        if (err) {
            callback(err, null);
            return;
        }

        const template = resp['Item']['template'];
        const data = resp['Item']['data'];
        const s3Params = {
            Bucket: templateBucket,
            Key: template };

        s3.getObject(s3Params, function(err, s3resp) {
            if (err) {
                callback(err, null);
                return;
            }
            const body = s3resp.Body.toString('utf-8');
            response.body = Mustache.render(body, data);
            callback(null, response);
        });
    });
};
```

Security Headers (1/2)

HSTS (tell your viewers to stick to HTTPS)

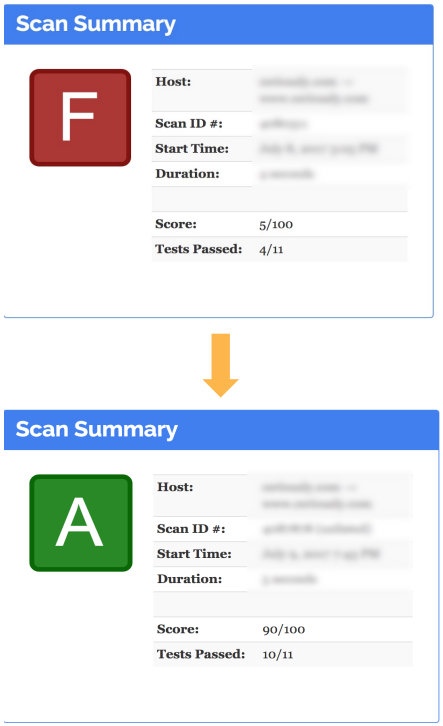
CORS (allow other domains to access your data relaxing the same-origin policy):

CSP (block content loaded from other sources, reduce XSS risk)

HPKP (pin your public key in the client, prevent MITM attacks with forged certificates):

More:

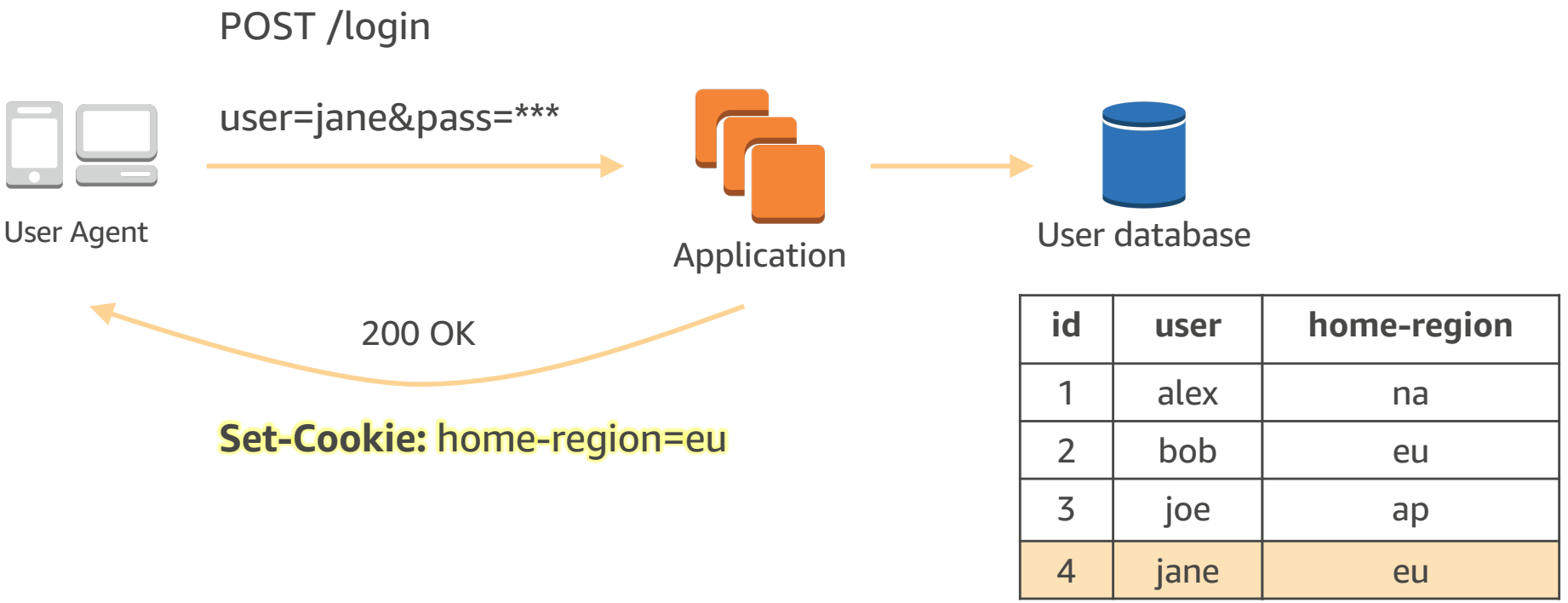
- X-Frame-Options (prevent clickjacking)
- X-Content-Type-Options (prevent MIME sniffing)
- X-XSS-Protection
- ...



Security Headers (2/2)

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

    // Enforce HTTPS with the HSTS header
    response.headers['strict-transport-security'] = [{
        key: 'Strict-Transport-Security', value: 'max-age=31536000; includeSubDomains'
    }];

    // Disallow embedded frames
    response.headers['x-frame-options'] = [{
        key: 'X-Frame-Options', value: 'DENY'
 }];

    // Remove CORS headers
    delete response.headers['access-control-allow-origin'];
    delete response.headers['access-control-max-age'];
    ...

    callback(null, response);
}
```

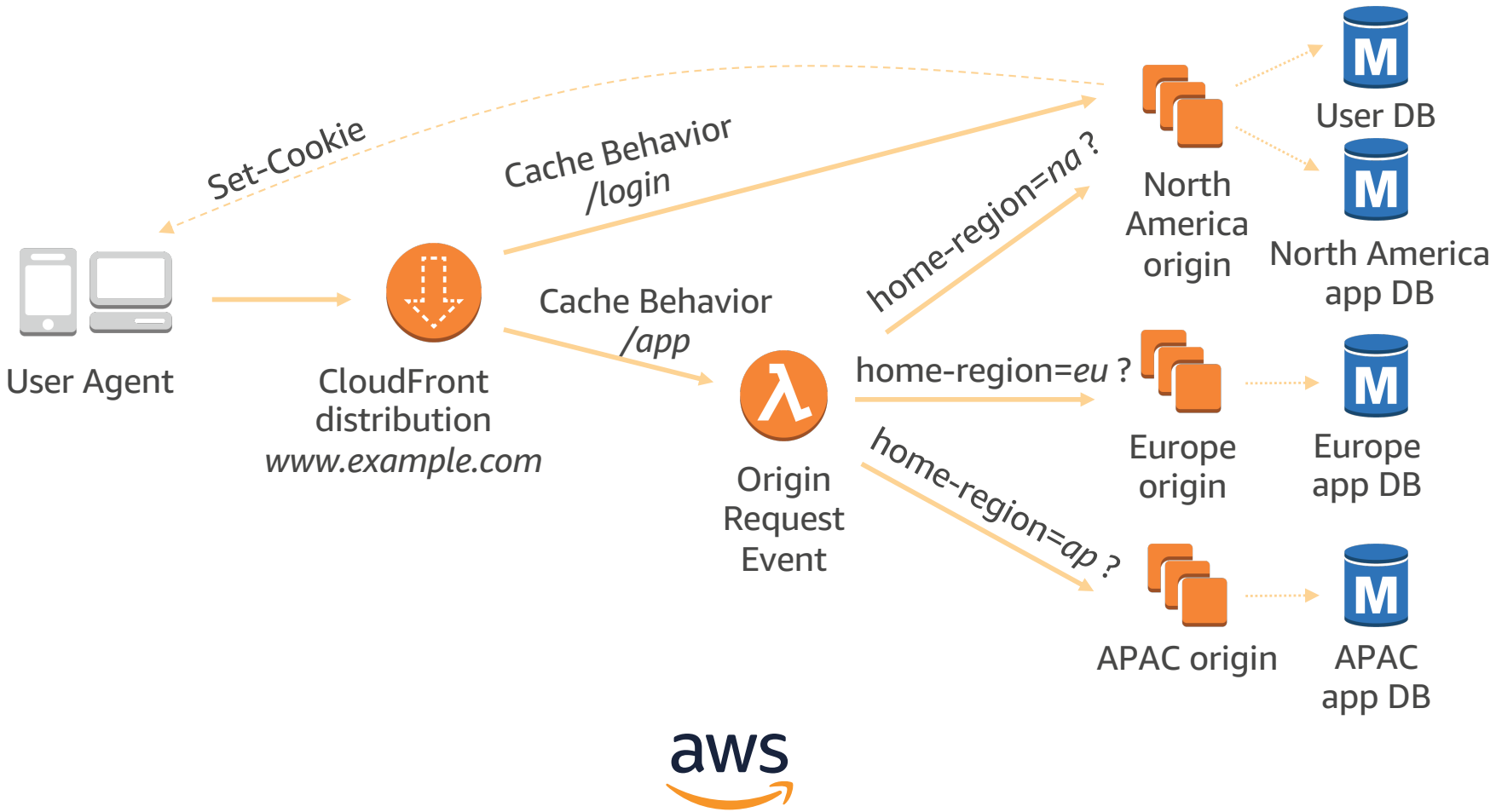
Origin selection (1/2)



id	user	home-region
1	alex	na
2	bob	eu
3	joe	ap
4	jane	eu

aws

Origin selection (1/2)




aws
7 best practices

How much does it cost?

Consider an API with 15M requests/month & a 256MB function executing in 125ms.

Lambda@Edge is charged based on the following two factors:

- Number of requests: 15M *$0.60/1M = $9
- Memory*Duration resource usage: 15M * 150ms *256MB $0,00005001$/GBS = 28,1$

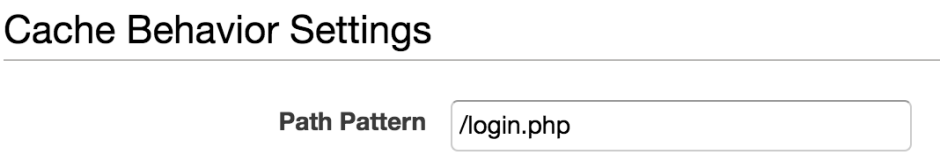
Viewer Trigger: Total = 37,1$ per month

#1 Invoke Lambda@Edge only where needed

CloudFront already provide native features:

- Device identification: CloudFront-Is-Mobile-Viewer headers
- Analytics: Access Logs delivered to S3 & WAF logs
- Access Control: signed URLs/Cookies, Geoblocking, WAF

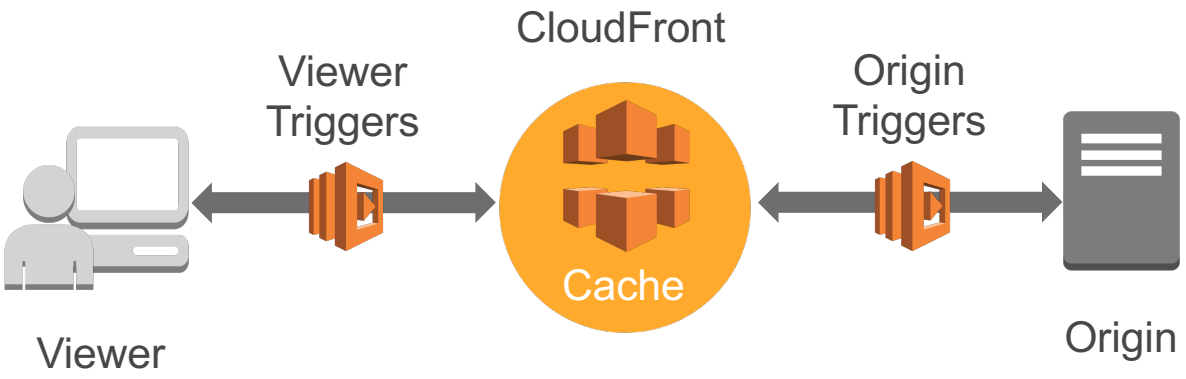
Use the most specific CloudFront behavior:



#2 Invoke Lambda@Edge only when needed

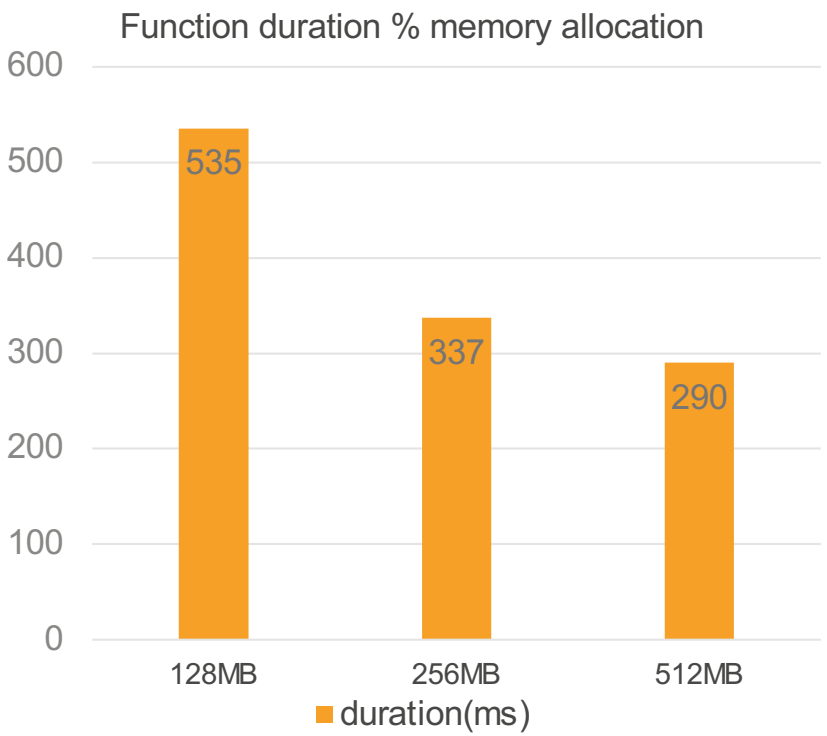
Remove it when it’s not used any more

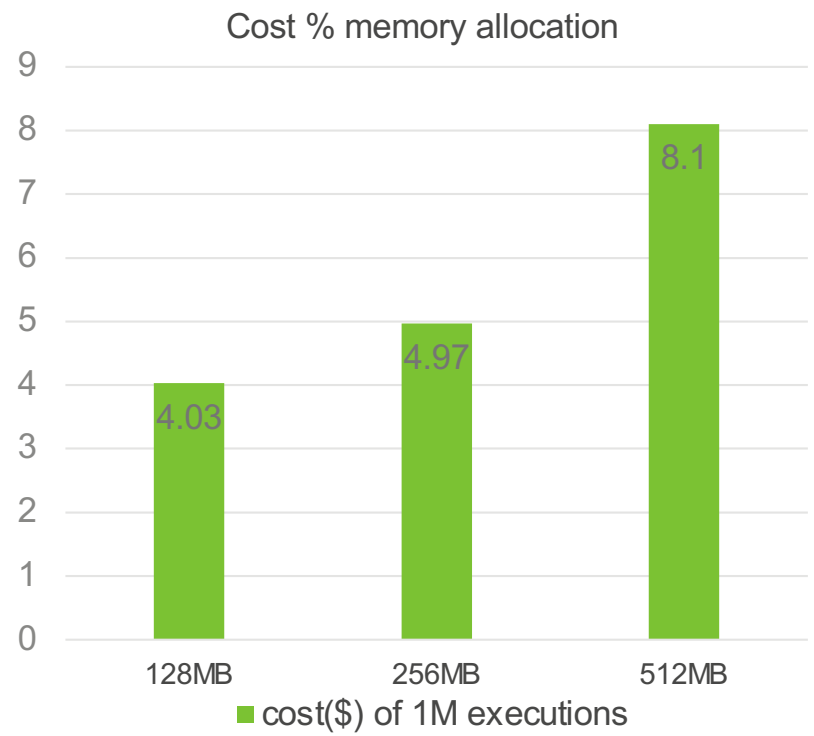
For every request or only on cache misses?



aws

#3 Optimize memory configuration





#4 Optimize function code for Node.js

Node.js 8.10 runtime

Reduce deployment package size: external dependencies, lightweight packages, minify, browserfy…

Parallelism using async calls

```
Promise.all([
            httpGet({ hostname: ”HTML template", path: ""}),
            ddbGet({ TableName: ddbTableName, Key: { name: ”achraf" } }),
]).then(responses => { …
```

#5 Leverage global variables (1/2)

```
const dns = require("dns");
let bestOrigin;
let expires = 0;
exports.handler = (event, context, callback) => {
            const request = event.Records[0].cf.request;
            getBestOrigin().then((origin) => {
                        request.origin.custom.domainName = origin;
                        headers.host[0].value = origin;
                        callback(null, request);
            });
}
```

#5 Leverage global variables (2/2)

```
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 + TTL;
                                    resolve(bestOrigin);
                        });
            });
}
```

#6 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: ”api.d.net", path: ”/”, agent: keepAliveAgent}, (resp) => {
                        let data = ’’;
                        resp.on('data', (chunk) => { data += chunk; });
                        resp.on('end', () => { resolve(data); });
            });
}
```

#7 Know the limits!

Functional:

- Blacklisted/ Read only headers
- Function size – 1MB vs 50MB
- Response size – 40K vs 1MB

Error negative Caching

Resource allocation

- Memory – 128M vs 3G
- Timeout – 5s vs 30s
- 1K concurrent execution & 10K RPS per region
- Scaling mechanism per region

To learn more about Lambda@Edge

AWS re:Invent 2017: Taking Serverless to the Edge (SRV312)

AWS re:Invent 2017: Building Serverless Websites with Lambda@Edge (CTD309)

Tutorial: Creating a Simple Lambda@Edge Function

Testing and Debugging Lambda@Edge Functions

https://github.com/aws-samples/aws-lambda-edge-workshops

Lambda@Edge Design Best Practices

Some takeaways…

- CloudFront brought storage closer to viewers, Lambda@Edge now brings compute closer to viewers.
- Lambda is serverless, Lambda@Edge is originless
- Many existing use cases for Lambda@Edge but you have the liberty to build and innovate.
- There are things that you can do to optimize your implementation in terms of performance and cost.

Thank you!

… time for Q&A