AWS re:Invent

NET304-R

Lambda@Edge best practices

Lee Atkinson
Principal Solutions Architect
Amazon Web Services

David Brown
Senior Product Manager
Amazon CloudFront
Amazon Web Services

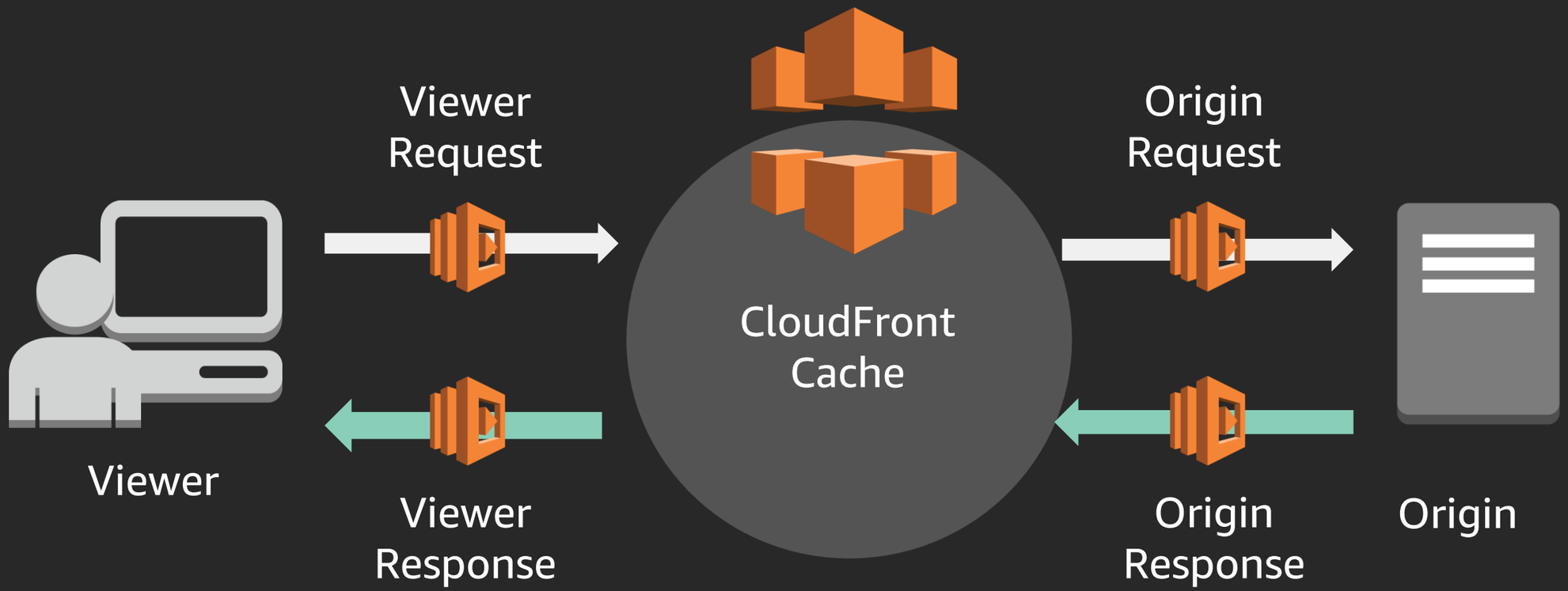
AWS re:Invent

aws

Lambda@Edge use cases

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

Amazon CloudFront and Lambda@Edge



How much does it cost?

Consider an API with 15M requests/month & 128MB Lambda@Edge function executing in 2ms. Viewer request event is configured on CloudFront.

Lambda@Edge is charged based on:

- Number of requests: 15M*,6$/1M = 9$
- Memory*Duration resource usage: 15M * 50ms * 128MB * 0,00005001$/GBS = 4,7$

Total cost is 13,7$/month

Why bother optimizing?

Is Lambda@Edge the right solution for you?

#1: Consider all the available options

- CloudFront already provides native features:
  - Device identification: CloudFront-Is-Mobile-Viewer headers
  - Analytics: CloudFront access logs delivered to Amazon S3 & AWS WAF logs
  - Access Control: CloudFront signed URLs/cookies, geoblocking, AWS WAF
- Leverage responsive web design

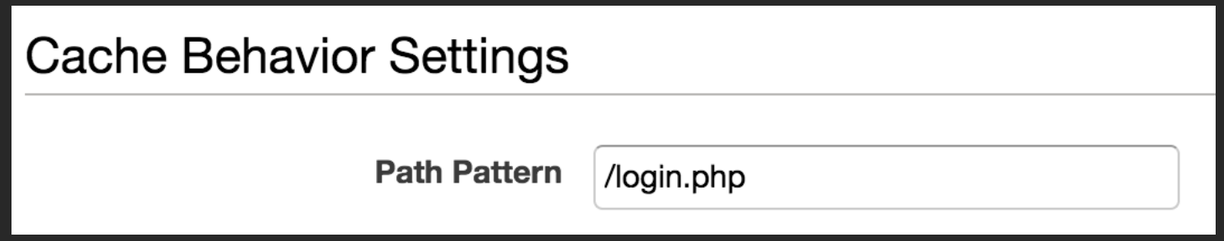
- Some logic is better off on the origin!

Optimizing Lambda@Edge configuration

#2: Invoke Lambda@Edge only when you need it

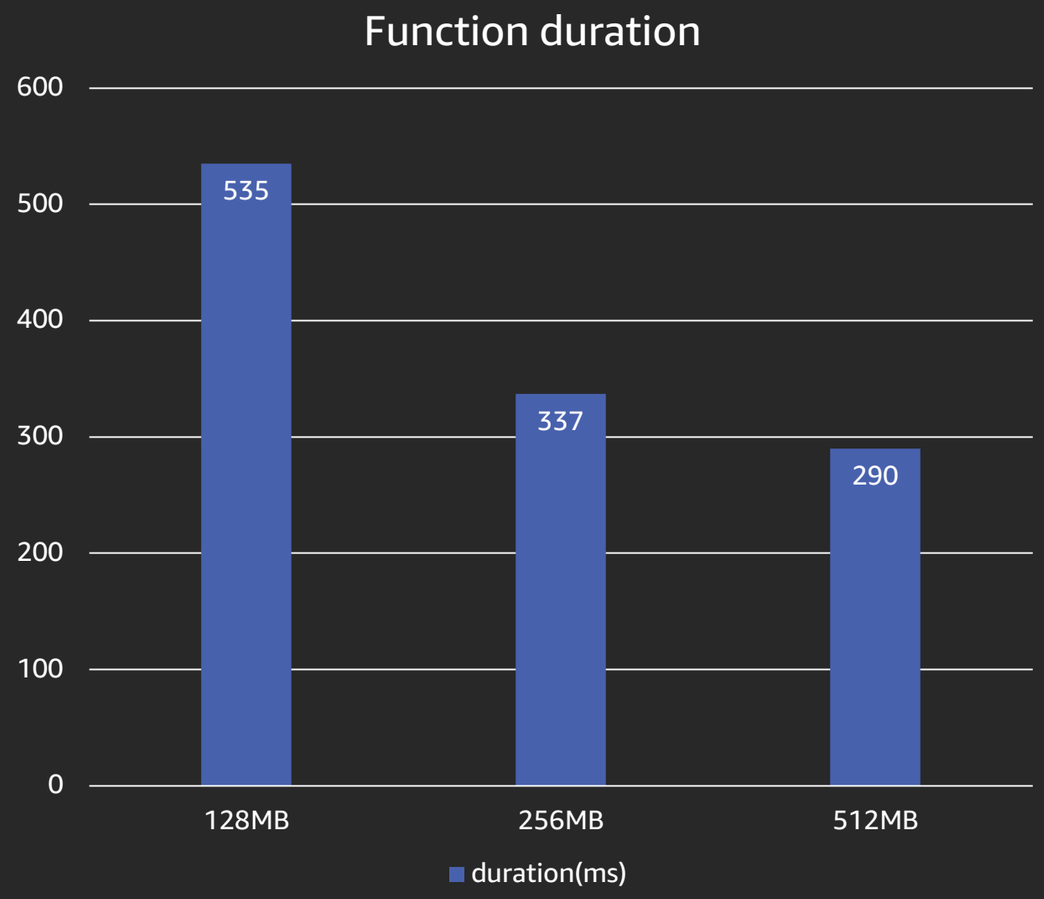
- For every request or only on cache misses?

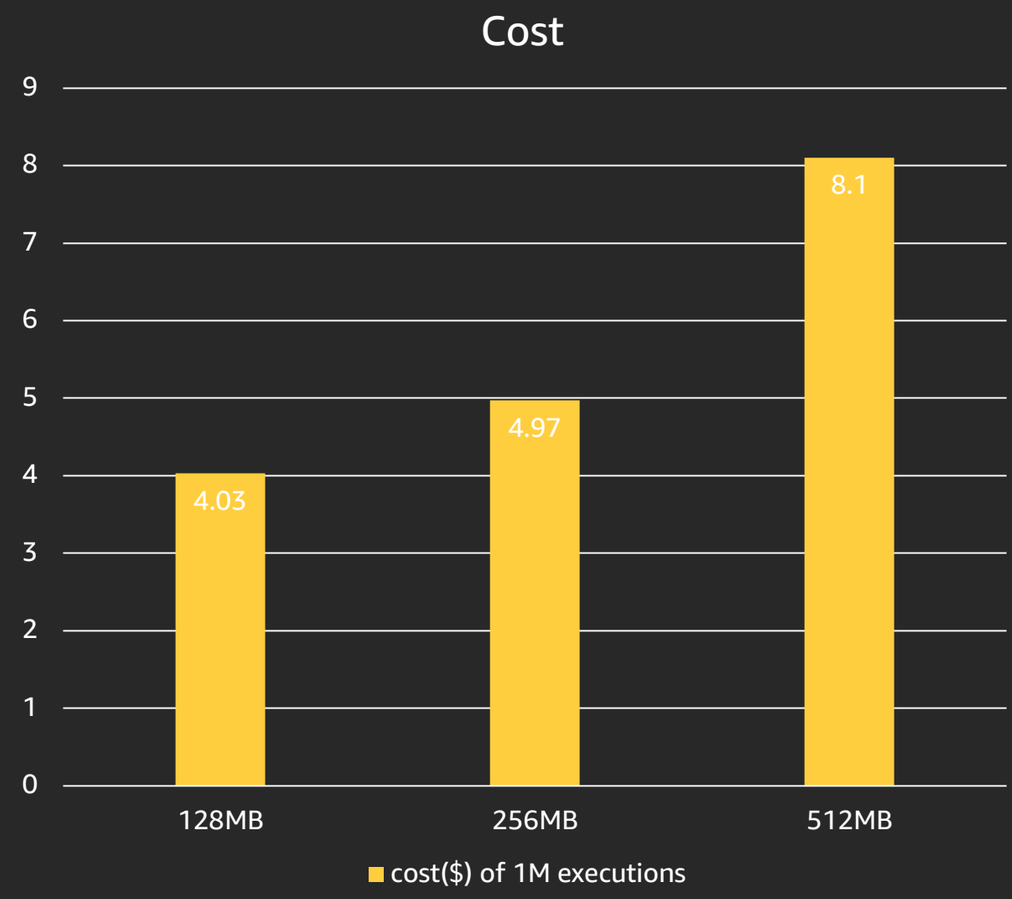
- Use the most specific CloudFront behavior:



- Remove it when it’s not used anymore

#3: Choose the optimal memory configuration





Optimizing Lambda@Edge code

#4: Optimize function code

- Reduce deployment package size
- Parallelism using async calls

```
// Node.js

let responses = await Promise.all([

        httpGet({ hostname: 'HTML template', path: '' }),

        ddbGet({ TableName: ddbTableName, Key: { name: 'mytable' } })

]);
```

#5: Leverage global variables (1/2)

```
// Node.js
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)

```
// Node.js
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

```
// Node.js
const http = require('https’);


exports.handler = (event, context, callback) => {
       http.get({ hostname: 'hello.com', path: '/’ }, (resp) => {
              let data = ’’;
              resp.on('data', (chunk) => { data += chunk; });
              resp.on('end', () => { resolve(data); });
       });
}
```

#6: Optimize external network calls

```
// Node.js
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); });
       });
}
```

Know the limits!

#7: Know the limits!

Functional:

- Blacklisted/read-only headers
- Function size—1MB vs 50MB
- Response size—40K vs 1MB

Resource allocation

- Memory—128M vs 3G
- Timeout—5s vs 30s
- 1K concurrent execution region
- Scaling mechanism

Additional resources

- https://aws.amazon.com/lambda/edge/
- https://aws.amazon.com/blogs/networking-and-content-delivery/lambdaedge-design-best-practices/

- https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/lambda-edge-testing-debugging.html

- AWS Blog: Cookie syncing for AdTech; Visitor prioritization for e-commerce; Paywall for publishers

Learn networking with AWS Training and Certification

Resources created by the experts at AWS to help you build and validate networking skills

Free digital courses cover topics related to networking and content delivery, including Introduction to Amazon CloudFront and Introduction to Amazon VPC

Validate expertise with the
AWS Certified Advanced Networking - Specialty exam

Visit aws.amazon.com/training/paths-specialty

aws training and certification

Thank you!

AWS re:Invent

aws

Please complete the session survey in the mobile app.