AWS Finland Meetup

CloudFront authentication and Lambda@Edge

Uri Savelchev, 24.05.2022

zalando

Agenda

1. About Zalando
2. Problem statement
3. History
4. CloudFront
5. Solution Design
6. Details and Caveats
7. Why it is cool?
8. Q&A

This is Zalando. The Starting Point for Fashion.

zalando

We take the lead in European fashion.

14.3bn

Euro GMV

almost 49m

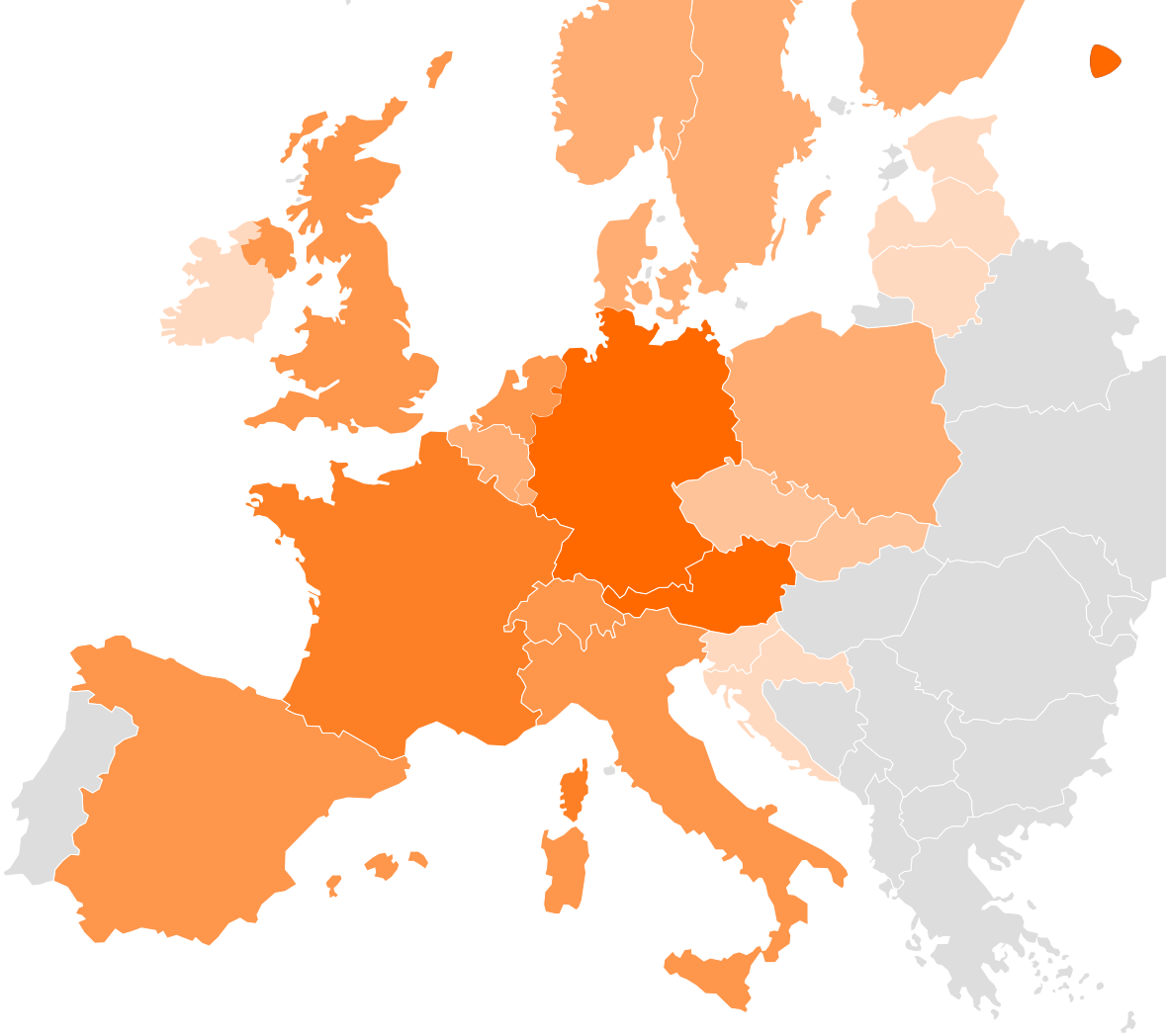
Active Customers

>5,800

Brands

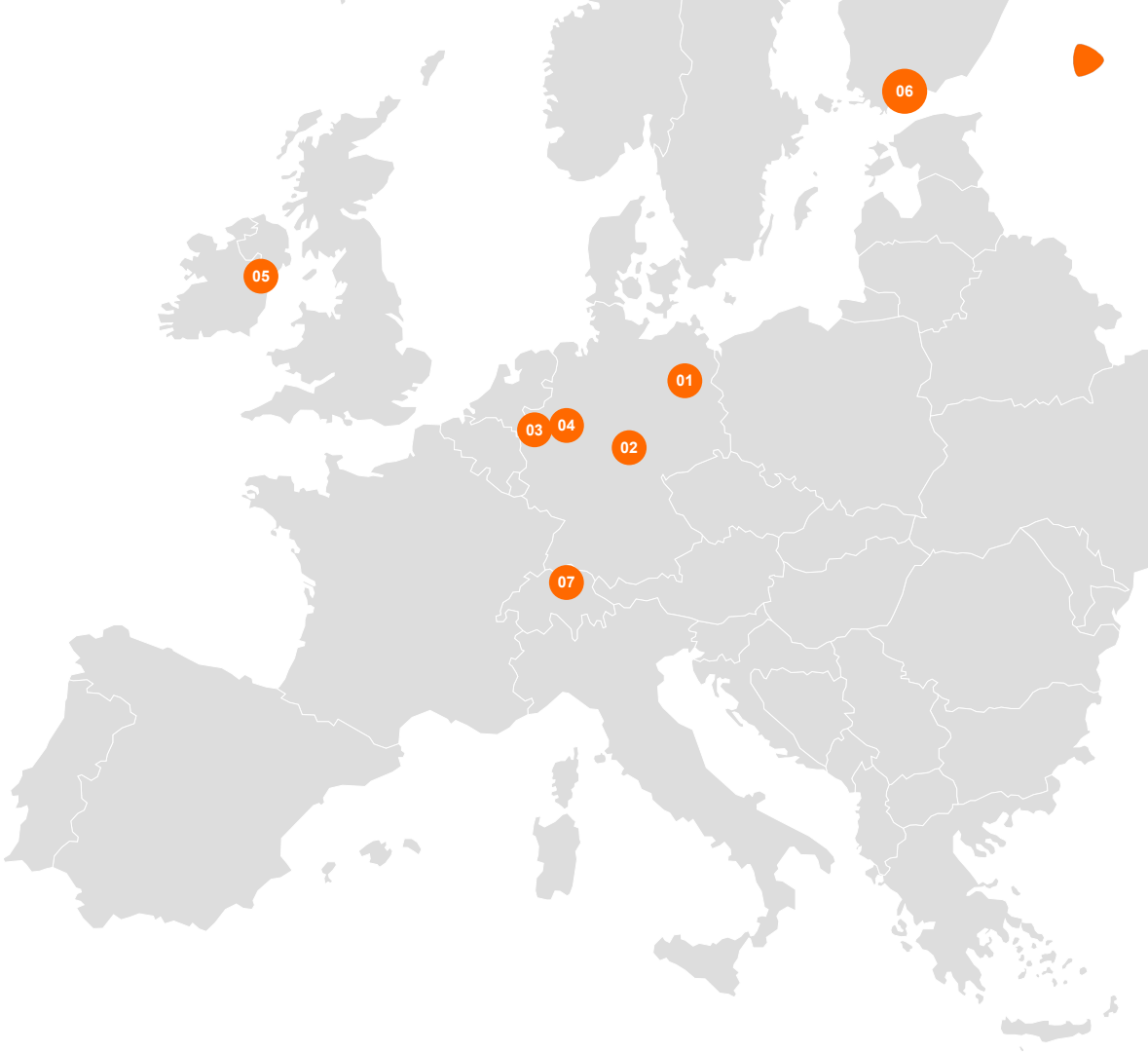
Bringing fashion
to 23 countries

- 2008-2009
- 2010
- 2011
- 2012-2013
- 2018
- 2021



Zalando offices

01 Berlin Headquarters
02 Erfurt Tech Office
03 Mönchengladbach Tech Office
04 Dortmund Tech Hub
05 Dublin Tech Hub
06 Helsinki Tech Hub
07 Zurich Tech Hub



Zalando Helsinki

Founded in
2015

160+ employees
39 nationalities

Location
Kamppi

Company language
English

01

Connected Retail

Connected Retail offers physical retailers a chance to connect to the Zalando platform and sell products directly to our growing online customer base.

02

Customer Fulfillment

In Helsinki, the Customer Fulfillment teams are building products to streamline a wide range of processes and material flows in Zalando's fulfillment network.

03

Digital Experience

The Digital Experience teams build the actual customer experience of the Zalando Fashion Store. We are responsible for some of the most prestigious real estate on Zalando - the Home screen of Zalando Fashion Store

04

Recommerce

The Helsinki Recommerce Engineering team is focusing on integrating the pre-owned fashion experience into the main Zalando site and apps.

Be part of our journey. The Starting Point for Fashion.

zalandohelsinki.com

Problem

Need to serve static web content for internal users:

- E2E functional test results
- Load test results
- Domain specific reports
- …

The content is **internal** and it should not be visible outside Zalando, so we need authentication.

The content does **not contain sensitive information** and can be shared to anyone in the company, so no special authorization or roles are required.

The content is static but it is **not persistent**, it is updated regularly or occasionally.

The content may be **quite different in terms of its size**.

The very first idea:

“Let’s use S3”

It is robust and durable, it can store any amount of data. It supports data aging via its “storage lifecycle policy”.

It is reasonably cheap.

We actively use S3 in our apps in Zalando, our CI/CD solution supports uploading to S3 out of the box.

S3 supports static website hosting.

S3 supports authentication, is the problem solved?

S3 works with AWS authentication only, that means the user has to authenticate with Amazon for the specific AWS account. We want it to work with our Platform IAM system (OAuth2).

S3 may be slow when you need to access many small files.

S3 seems to be a proper storage for the content,

but we have something else to serve it to users

History
First solution

NGINX with a Lua script to go through OAuth2 authentication flow, based on the Cloudflare script

`aws s3 sync` to synchronize content in S3 to a local K8s volume, running as sidecar container.

It worked, but…

The content size tends to grow up, that makes difficult to use local storage.

Using persisted volumes (EBS) has its own problems.

Can we serve from S3 **directly**?

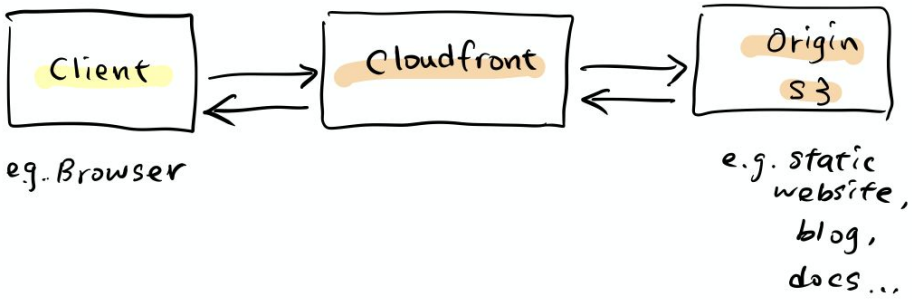
AWS CloudFront

CloudFront is the AWS managed CDN. Being connected to many large internet providers it provides fast and effective delivery of a content.

AWS S3 is the most popular Origin for CloudFront distributions.

For low-profile distribution CloudFront is *almost* free (Free tier is 1TB/month per account).

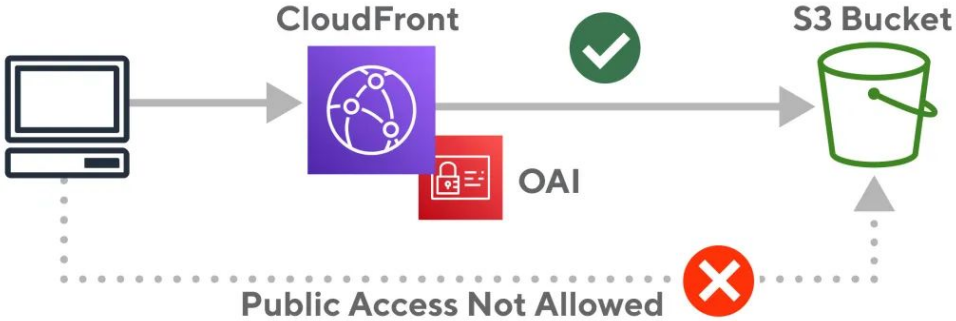
In 2021 Zalando migrated our CDNs to CloudFront.



What if we use **CloudFront?**

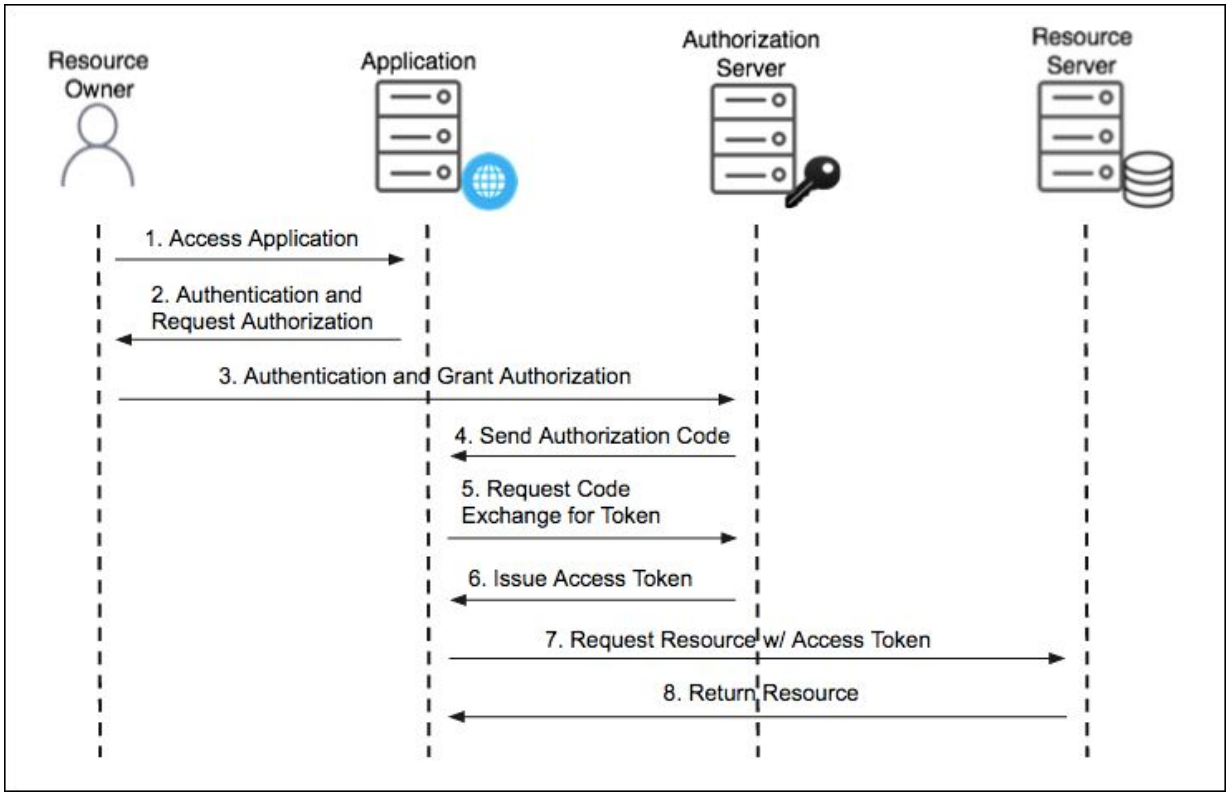
CloudFront supports access authorization via signed URLs and signed cookies. The public access is blocked.

We need some code to handle the OAuth2 authentication and then to issue the signed cookies for a limited time.



OAuth2
Authorization Code Flow

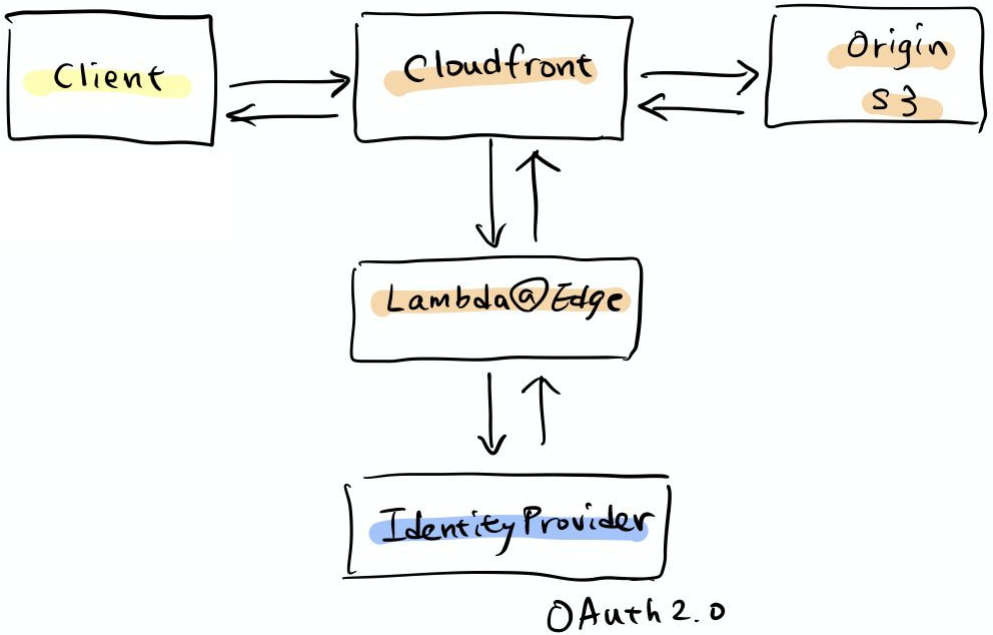
The common user authentication flow for web applications. The access token is never sent to the client browser, so the access is limited by the client application logic (server side).



The obvious choice is to deploy the code as **Lambda@Edge**

There is a number of published articles with ready blueprints to deploy CloudFront and Lambda@Edge together, e.g. the nice one from Ernest Chiang.

To be run at edge a Lambda has to be deployed to **us-east-1**. Unfortunately at the time of creating the solution (Spring 2021) deploying to non-european regions was not supported for us.



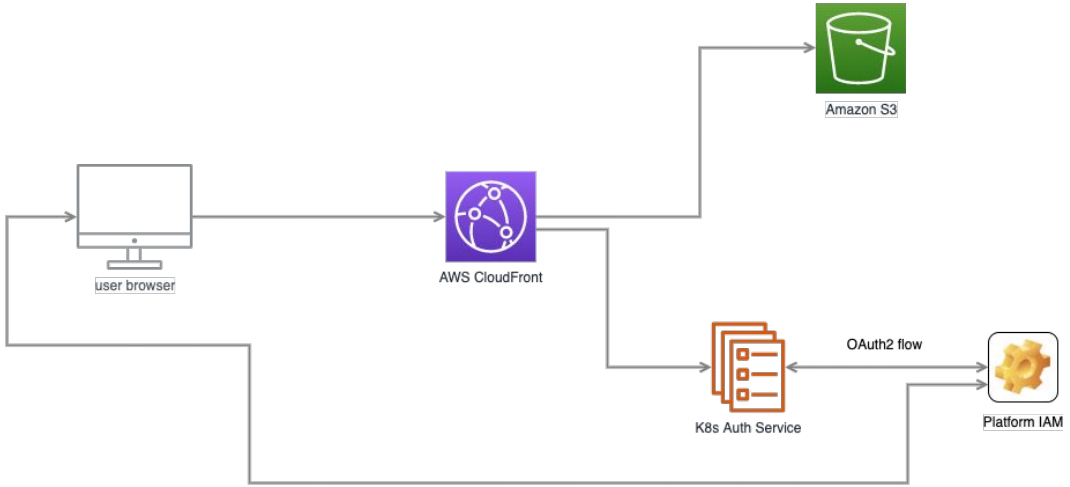
Had to resort* to a plain Kubernetes service

* Now we can deploy Lambda@Edge, so I'm going to migrate it to Lambda

Solution

A Kubernetes service goes through the OAuth2 authorization code flow and then issues signed cookies for CloudFront

- the service is responsible only for the OAuth2 flow, the rest is handled by CloudFront
- when the cookies are expired, CloudFront has to return error 403. We set a custom page for it, saying the session is expired and asking to click-n-go to the Auth URL



01 **Domain magic**
Cookies may be set for super-domain using the “Lax” policy

02 **Specific URL for redirect**
Use something like `/index.html` to avoid looping

03 **Different cache behaviors**
Disable caching for the Auth Origin

04 **Think about invalidations**
In our case we invalidate root document every time it is updated in S3. Also keep the CloudFront distribution ID somewhere - you need it to trigger invalidations.

05 **Use a mainstream language**
Signed cookie generation is trivial, but picky about details, use some ready-to-use code, if possible.

```
Origins:
- Id: SourceS3Bucket
  S3OriginConfig:
    OriginAccessIdentity: !Sub "origin-access-identity/cloudfront/${OriginAccessIdentity}"
  DomainName: !Sub "${SourceBucketName}.s3.eu-central-1.amazonaws.com"
- Id: AuthOrigin
  CustomOriginConfig:
    OriginProtocolPolicy: match-viewer
  DomainName: !Ref OAuth2AppDNSName
CacheBehaviors:
- PathPattern: /
  TargetOriginId: AuthOrigin
  CachePolicyId: 4135ea2d-6df8-44a3-9df3-4b5a84be39ad # Managed-CachingDisabled
  ViewerProtocolPolicy: redirect-to-https
- PathPattern: /expired
  TargetOriginId: AuthOrigin
  CachePolicyId: 658327ea-f89d-4fab-a63d-7e88639e58f6  # Managed-CachingOptimized
  ViewerProtocolPolicy: redirect-to-https
DefaultCacheBehavior:
  TargetOriginId: SourceS3Bucket
  CachePolicyId: 658327ea-f89d-4fab-a63d-7e88639e58f6  # Managed-CachingOptimized
  ViewerProtocolPolicy: redirect-to-https
  TrustedKeyGroups:
  - !Ref TrustedKeyGroup
CustomErrorResponses:
- ErrorCode: 403
  ResponseCode: 200
  ResponsePagePath: /expired
```

```
cookies = generate_signed_cookies("https://*", CLOUDFRONT_KEY_ID, PRIVATE_KEY,
    PRIVATE_KEY_PASSWORD)
expiration = int(time.time()) + MAX_CF_COOKIE_AGE
logging.debug("Setting %d cookies: expires=%d, domain=%s" % (len(cookies),
    expiration, BASE_DOMAIN))

for cn,cv in cookies.items():
    logging.debug("Cookie: %s -> %s" % (cn, cv))
    response.set_cookie(cn, cv, expires=expiration, domain=BASE_DOMAIN,
        samesite='Lax', secure=True, httponly=True)
```

Why do I think the solution is cool?

01 **Simplicity**
We need to validate the user and issue a cookie, the rest is handled for us by AWS

02 **Effectiveness**
Whatever is the content size, it is served fast

03 **Cost effective**
While your traffic is under 1TB/month, you pay Amazon for invalidations only. The authentication service is very light and does not consume much resources.

04 **Robust**
The authentication service is very simple and it has only dependency - the Zalando OAuth2 service

05 **Customizable**
If I need to add authorization rules, I can extend the authentication service and I don’t need to touch CloudFront part at all

Thank You