connect

Dogfooding Workers at Cloudflare

CLOUDFLARE

connect

Rita Kozlov

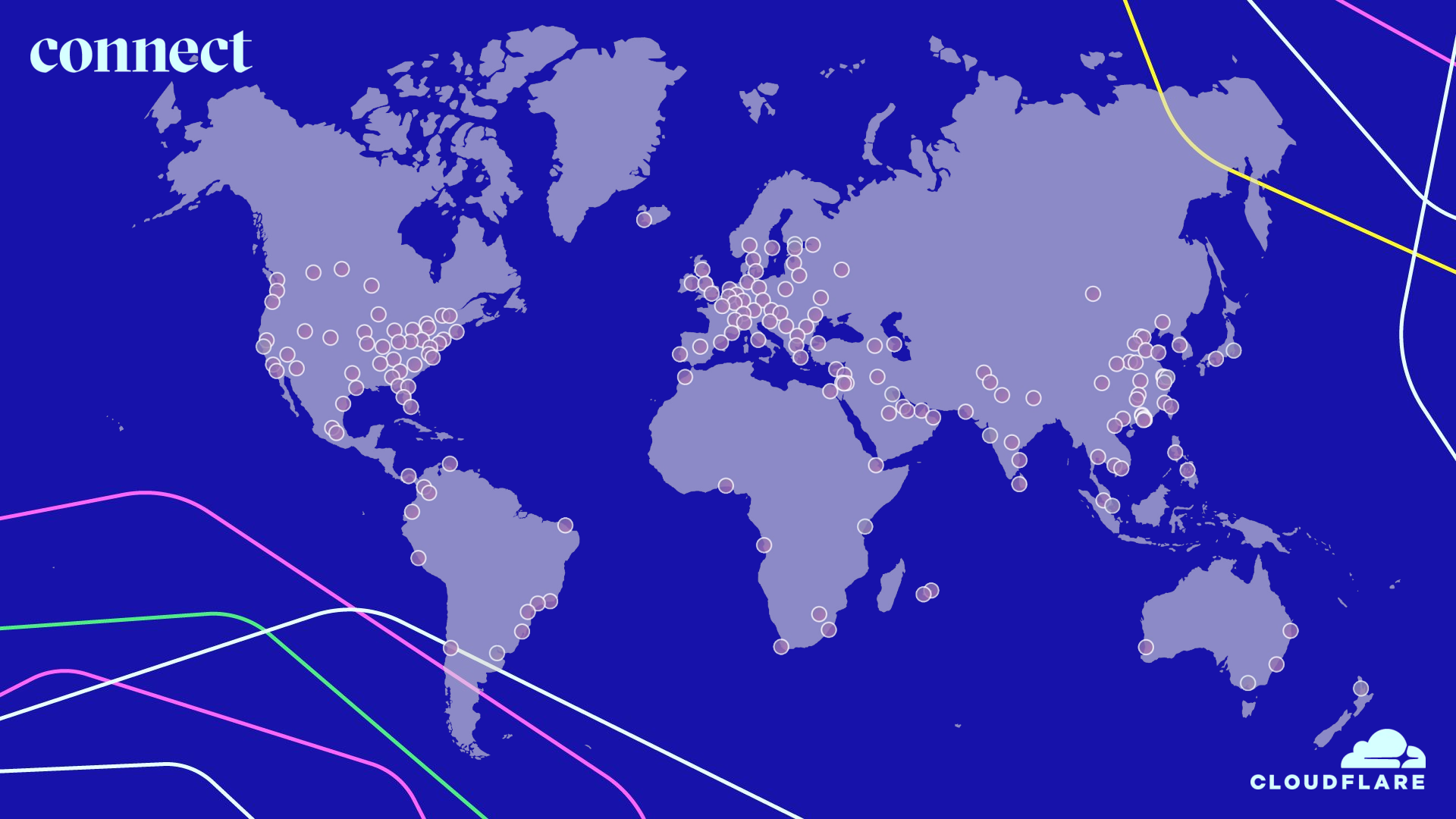
Product Manager, Workers

CLOUDFLARE

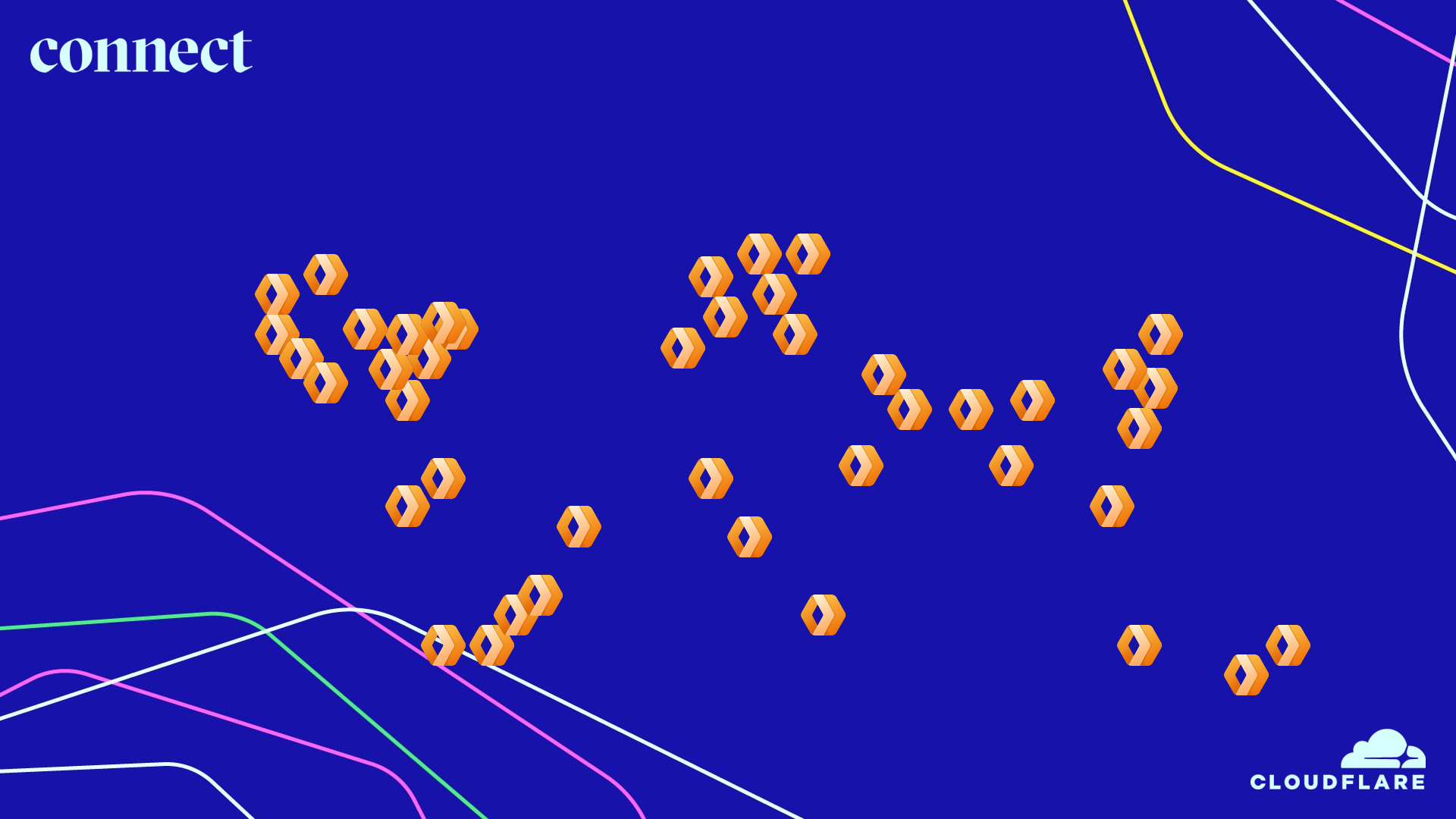
connect

Cloudflare Workers = Cloudflare's Serverless Platform

CLOUDFLARE


connect
CLOUDFLARE

connect



CLOUDFLARE

connect

Serverless

CLOUDFLARE

connect

Serverless at Global Scale

CLOUDFLARE

connect

How do I get started with Cloudflare Workers?

CLOUDFLARE

connect

Gaul’s law

CLOUDFLARE

connect

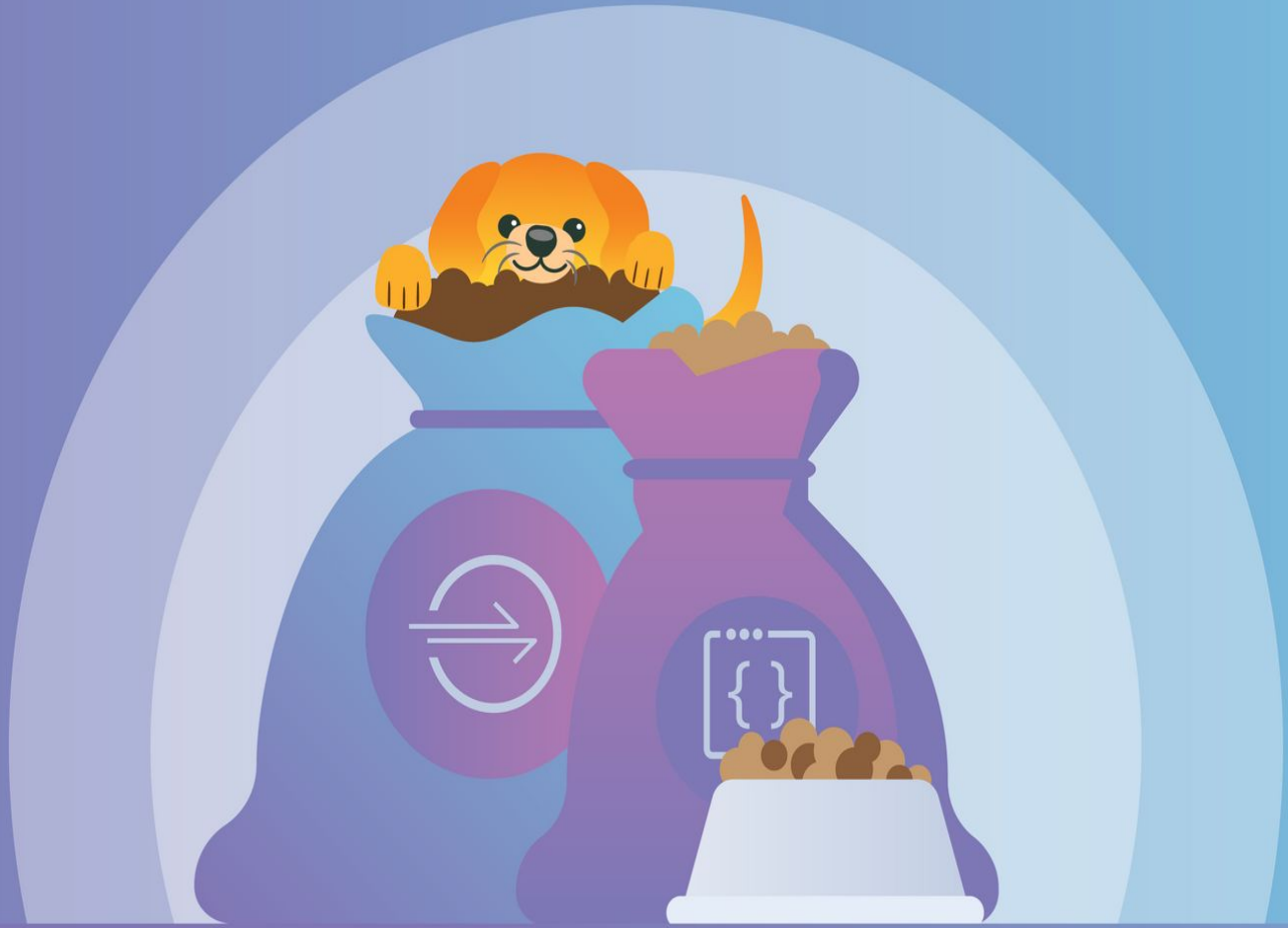
Designing complex systems is hard

CLOUDFLARE

connect

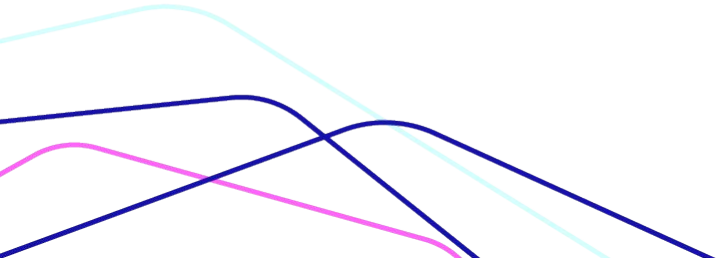
Start simple!

CLOUDFLARE



Cloudflare’s Journey into Workers

- **Augment:** Deprecating old TLS
- **Enhance:** Access on Workers
- **Greenfield:** Workers.dev reservation system



CLOUDFLARE

connect

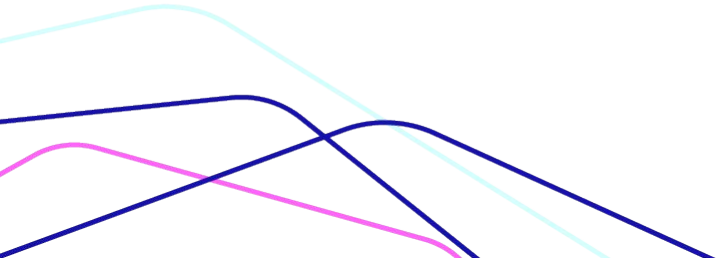
Case study #1: Deprecating old TLS

CLOUDFLARE

Deprecating old TLS

Requirement

- PCI: need to disable TLS < 1.2



CLOUDFLARE

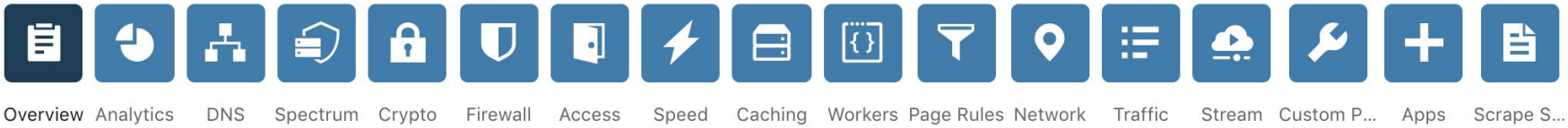
Deprecating old TLS

Requirement

- PCI: need to disable TLS < 1.2

Challenge

- Cloudflare has many different services



CLOUDFLARE

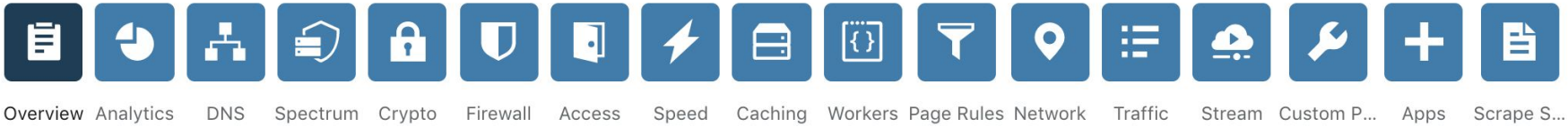
Deprecating old TLS

Requirement

- PCI: need to disable TLS < 1.2

Challenge

- Cloudflare has many different services
- Every service — owned by a different team, different stack

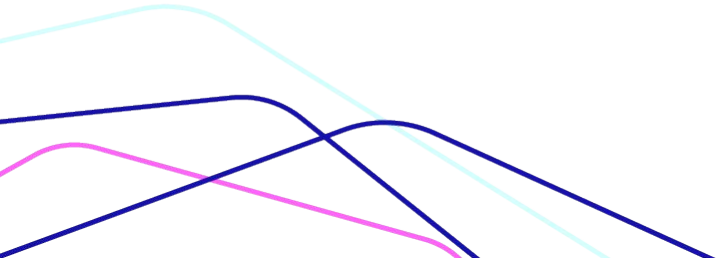


CLOUDFLARE

Deprecating old TLS

With Workers

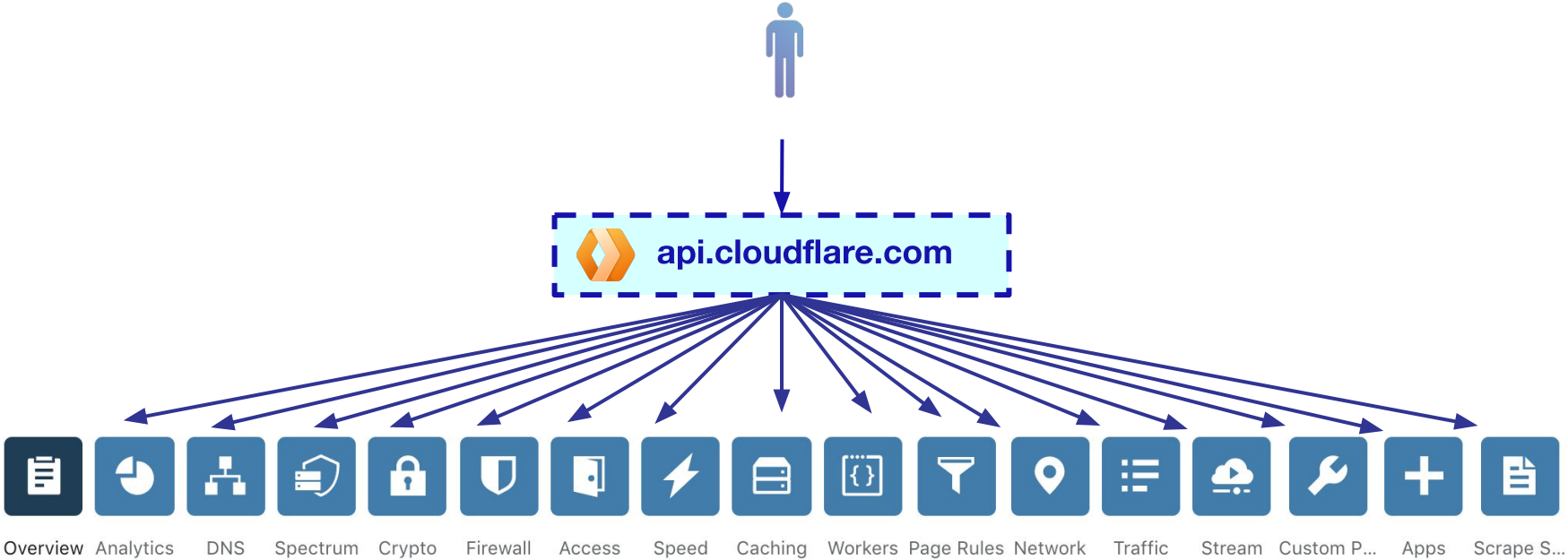
- Cloudflare Workers sits between the eyeball, and the origin(s)
- Set up a proxy
- Disable old TLS at the Worker level



CLOUDFLARE

Deprecating old TLS

Solution



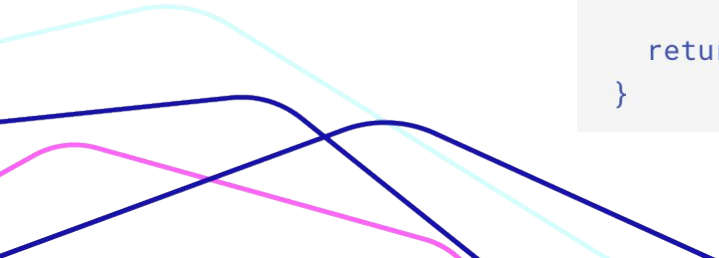
CLOUDFLARE

Deprecating old TLS

Super simple!

You can do this too!

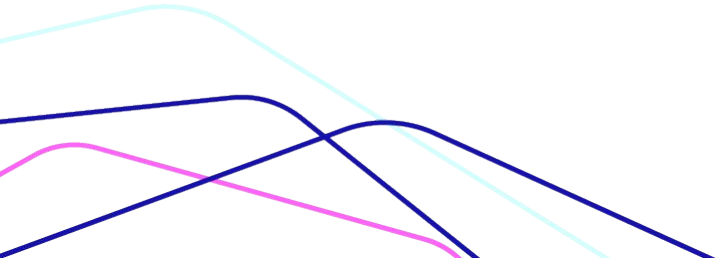
```
addEventListener('fetch', event => {
  event.respondWith(sslBlock(event.request))
})

async function sslBlock(request) {
  // Find TLS version - This will appear as "undefined" in the Preview
  // For specific cipher blocking, you can inspect request.cf.tlsCipher
  let tlsVersion = (request.cf || {}).tlsVersion

  // Allow only TLS versions 1.2 and 1.3
  if ((tlsVersion != 'TLSv1.2') && (tlsVersion != 'TLSv1.3')){
    return new Response("Please use TLS version 1.2 or higher.",
    { status: 403, statusText: "Forbidden" })
  }

  return fetch(request)
}
```



CLOUDFLARE

Deprecating old TLS

- Easy to unify many different services
- No performance hit



CLOUDFLARE

connect

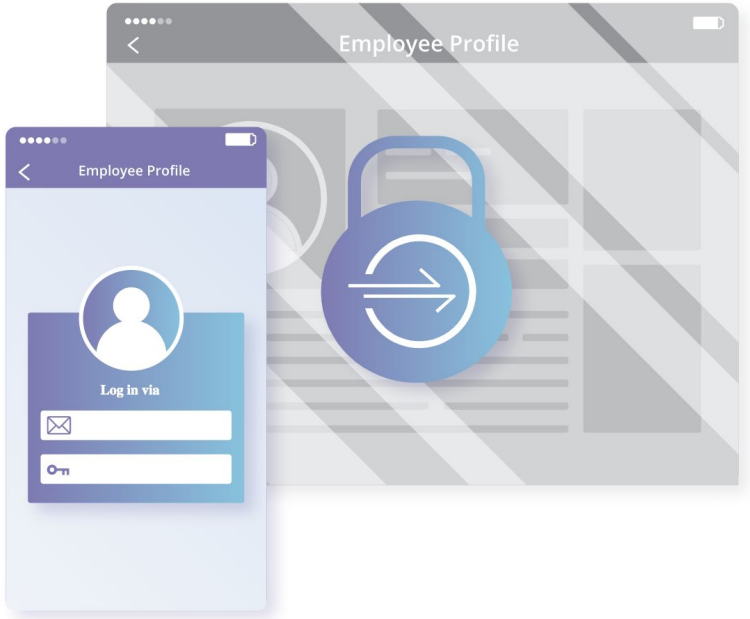
Case study #2: Access on Workers

CLOUDFLARE

Access on Workers

What is Cloudflare Access?

A perimeter-less access control solution for cloud and on-premise applications.

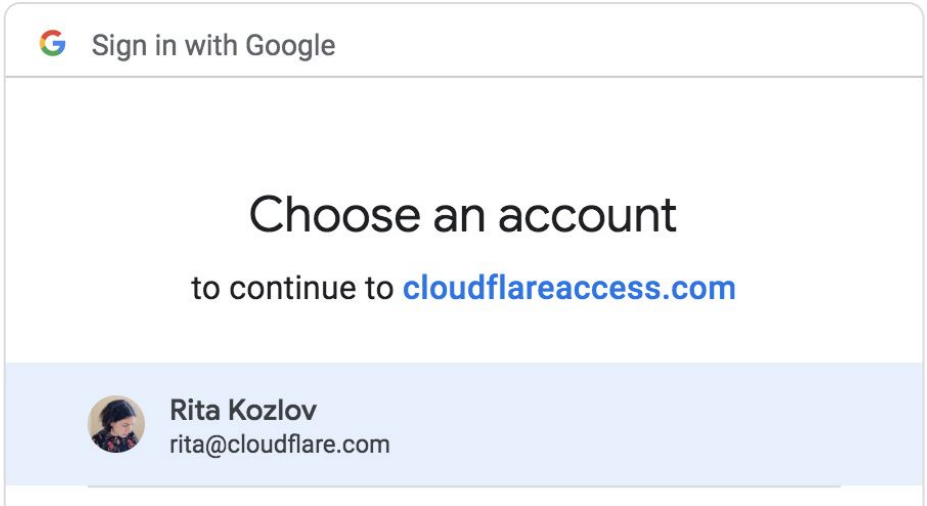


CLOUDFLARE

Access on Workers

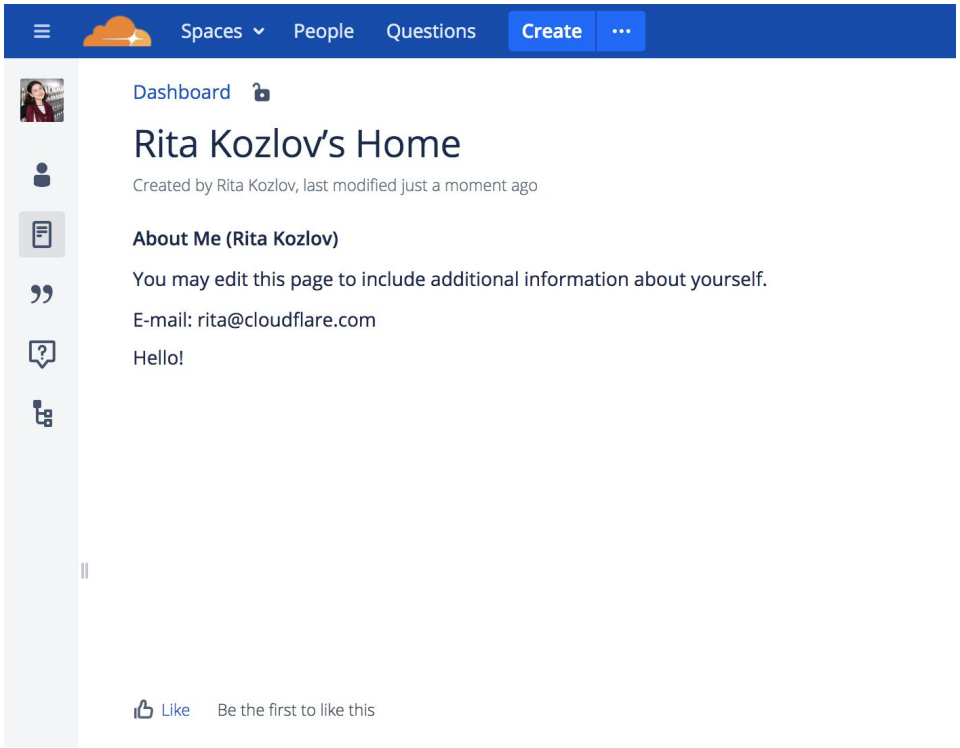
CLOUDFLARE

Access on Workers



CLOUDFLARE

Access on Workers



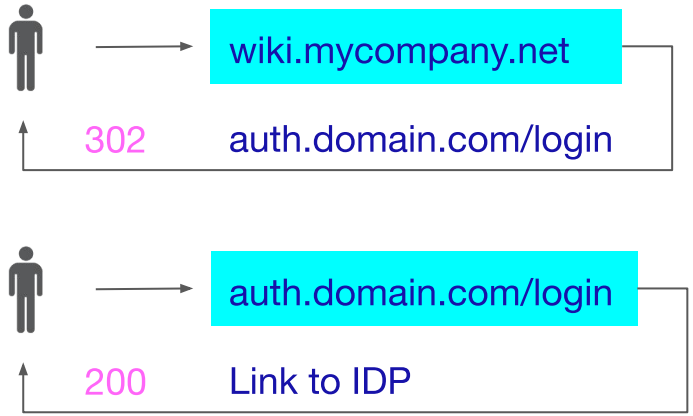
CLOUDFLARE

Access on Workers

How does it work?

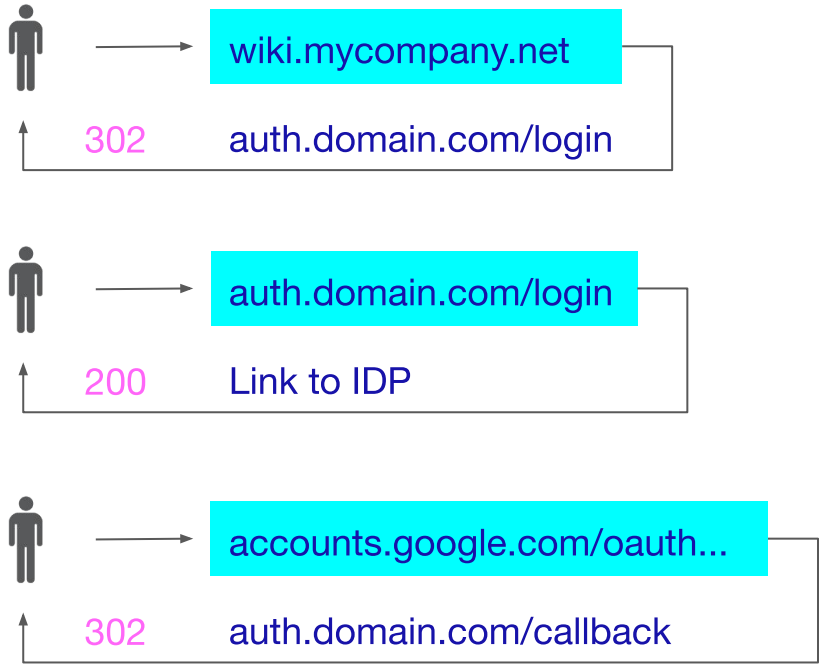
Access on Workers

How does it work?



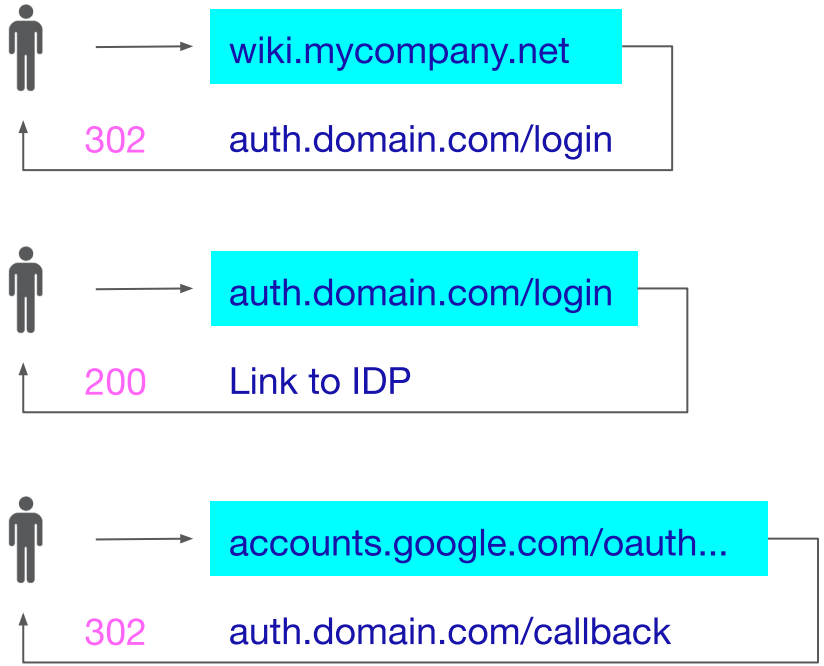
Access on Workers

How does it work?



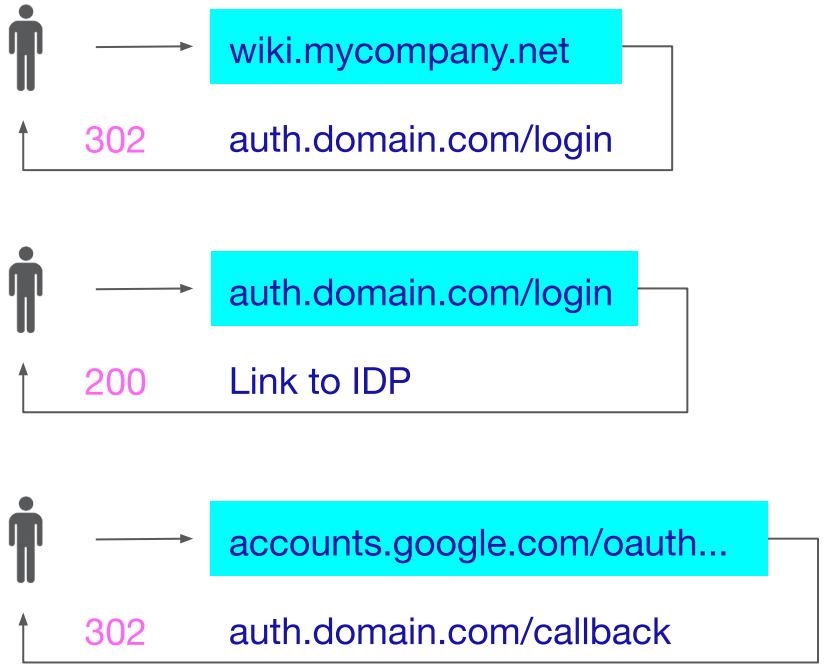
Access on Workers

How does it work?



Access on Workers

How does it work?

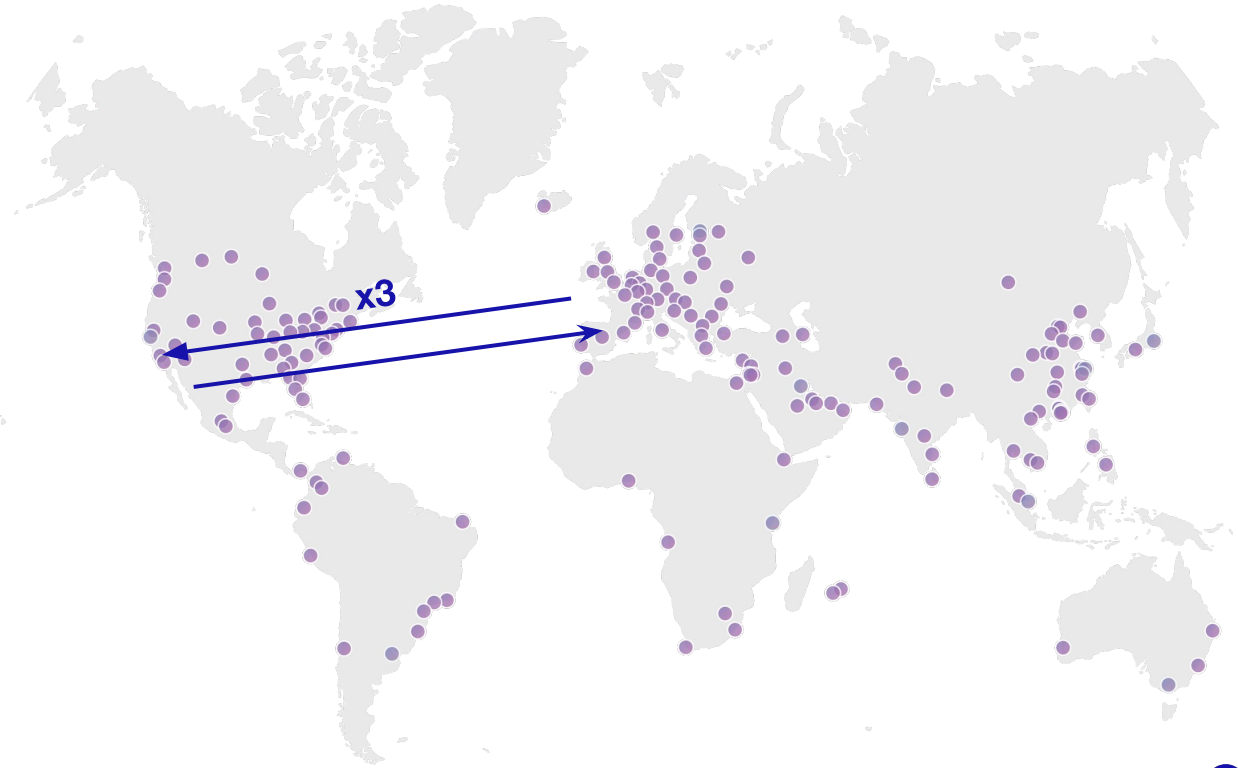


Access on Workers

Challenge

For every action you take, you have to go all the way back to one of our core data centers

- At least 3 times for login path



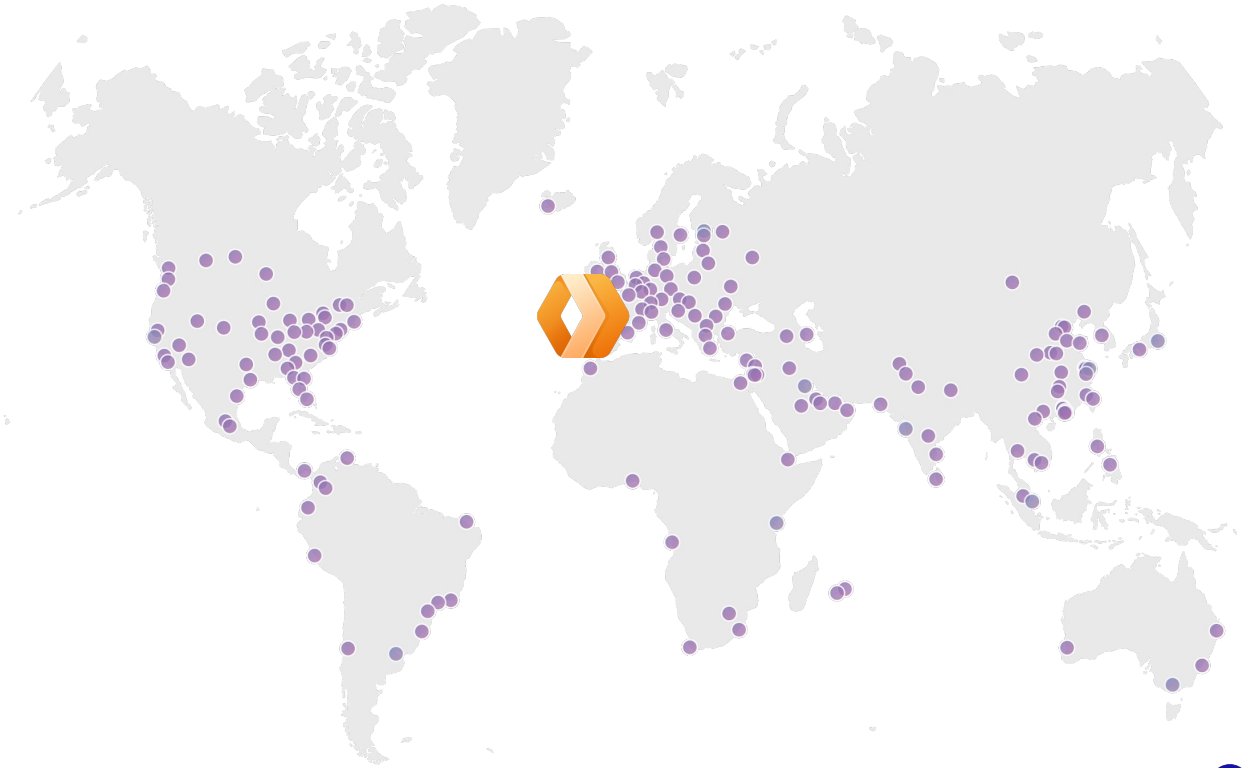
CLOUDFLARE

Access on Workers

With Workers:

Do authentication at the edge — all the needed information is in the JWT

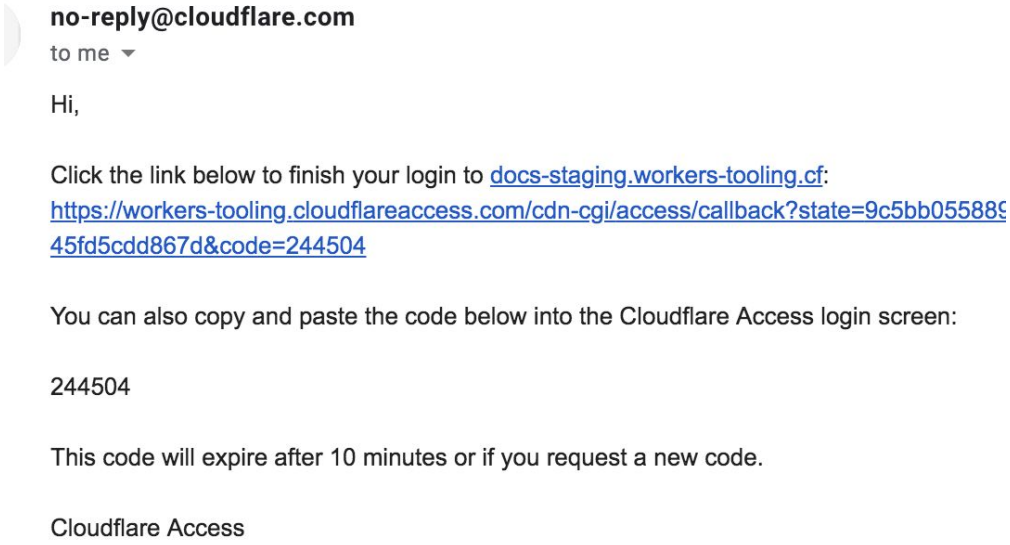
- Now only bottleneck is identity provider



CLOUDFLARE

Access on Workers

Single-time nonce



no-reply@cloudflare.com
to me

Hi,

Click the link below to finish your login to docs-staging.workers-tooling.cf:
https://workers-tooling.cloudflareaccess.com/cdn-cgi/access/callback?state=9c5bb055889
45fd5cdd867d&code=244504

You can also copy and paste the code below into the Cloudflare Access login screen:

244504

This code will expire after 10 minutes or if you request a new code.

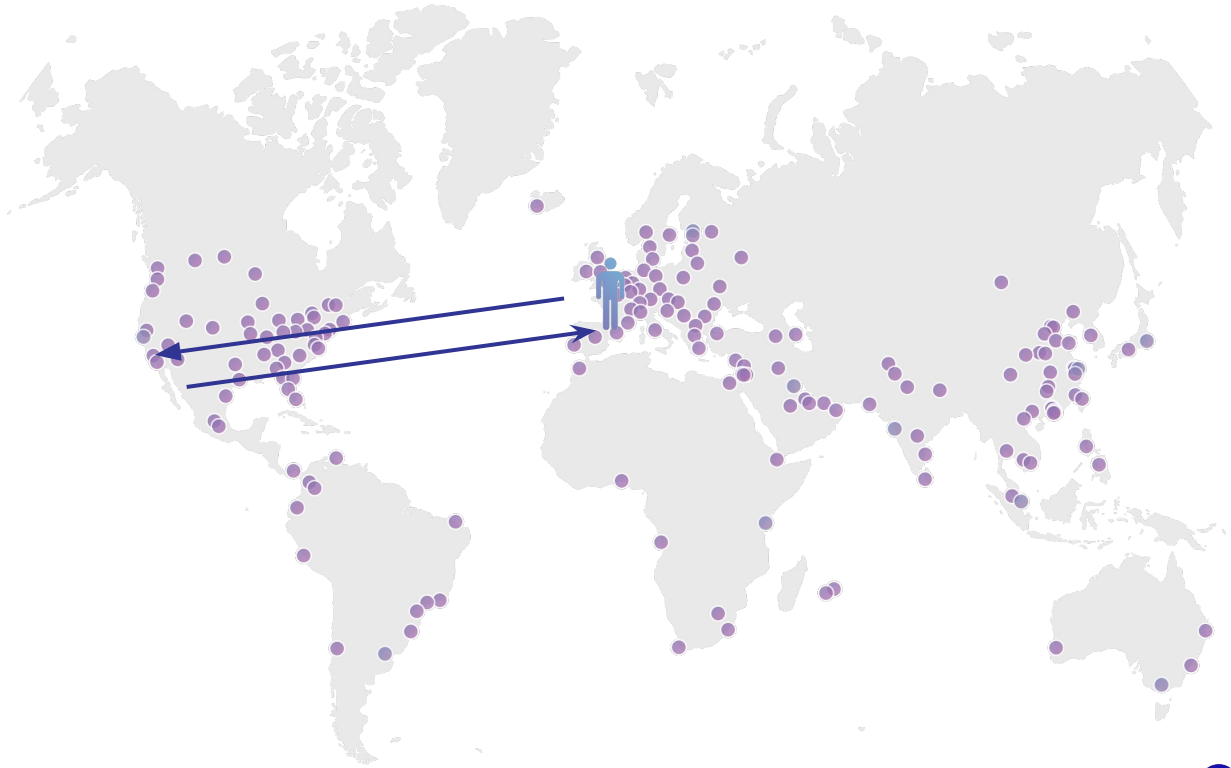
Cloudflare Access

CLOUDFLARE

Access on Workers

Challenge

If you use the single time nonce, the nonce would be generated in a single location



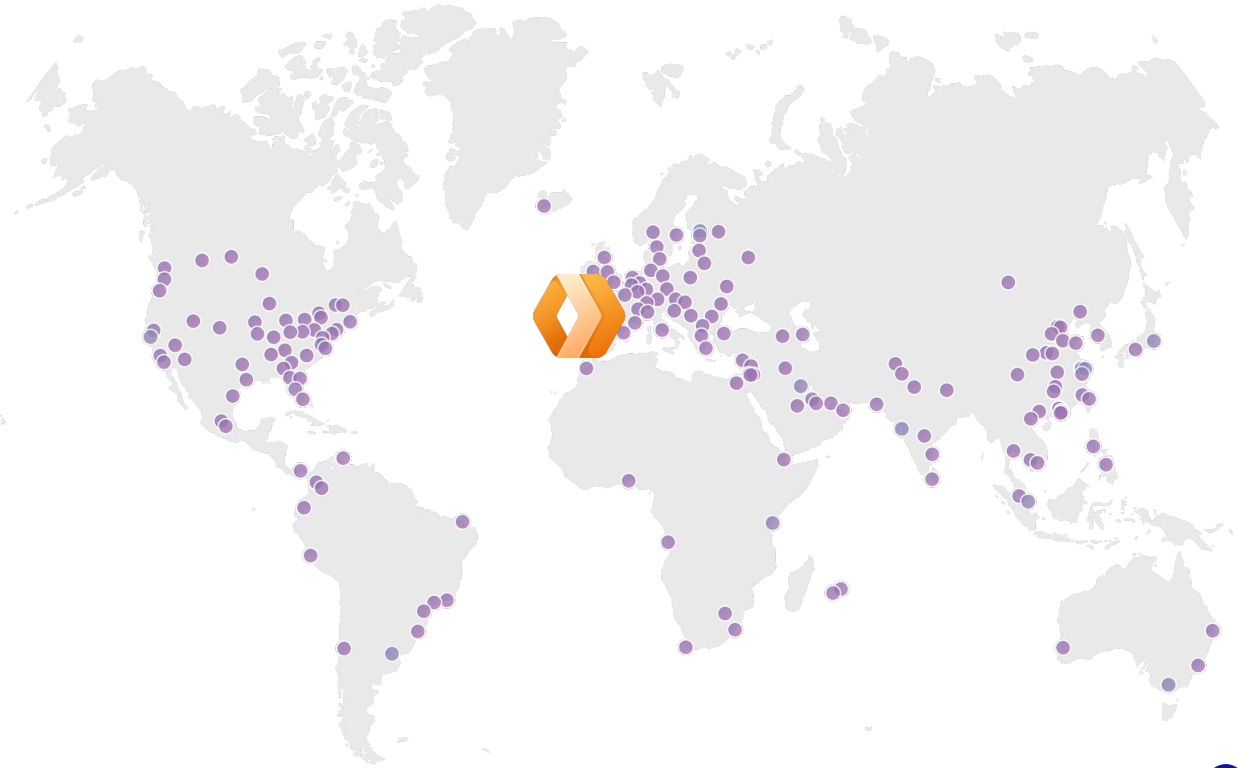
CLOUDFLARE

Access on Workers

With Workers

Generate the nonce and store in KV

- Originally used cache API, but when users travel, they might connect to a different PoP



CLOUDFLARE

Access on Workers

Approach: Worker per endpoint

- Split out logic
  - /login
  - /callback
  - etc
- Lower risk deployments
- Easier to work individually

CLOUDFLARE

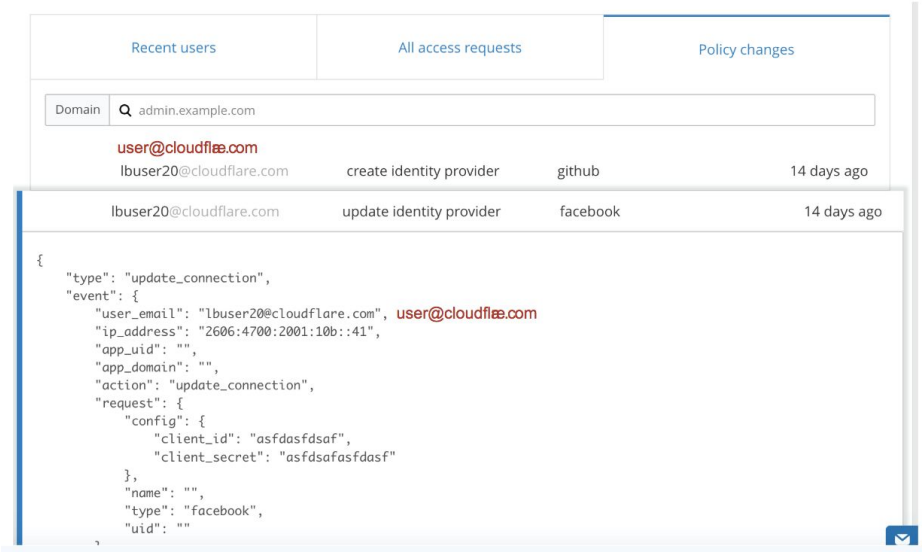
Access on Workers

Approach: logging

Asynchronous logging with

`WaitUntil()`

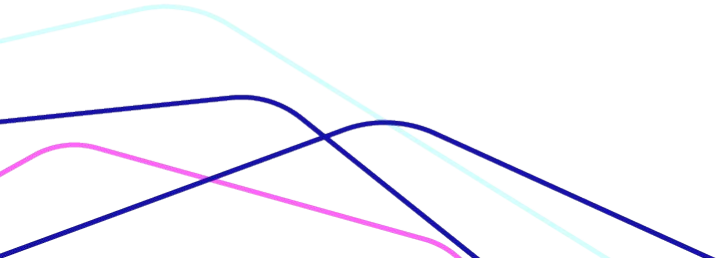
- Audit logs
- Sentry logs



CLOUDFLARE

Access on Workers

- Improved performance
- Improved reliability (reduced points of failure)
- Try different approaches



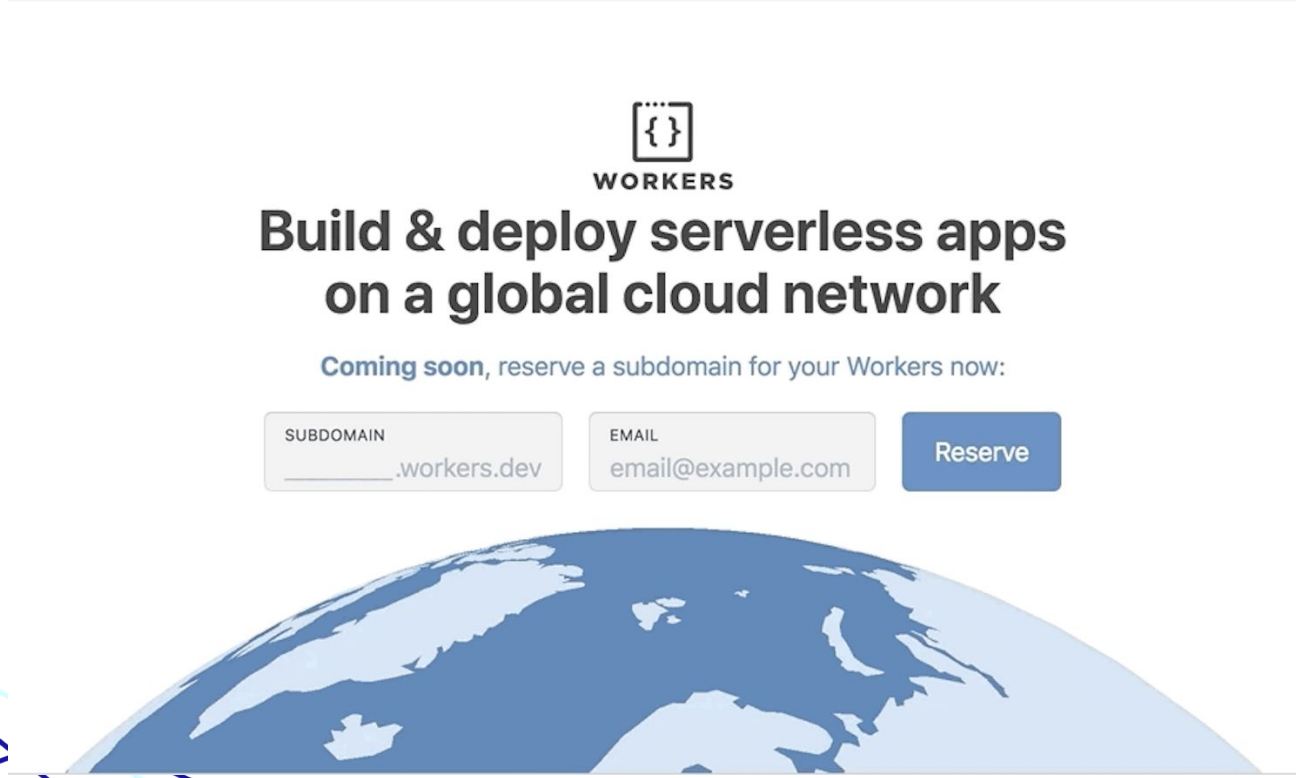
CLOUDFLARE

connect

Case study #3: Reservations for workers.dev

CLOUDFLARE

Reservations for workers.dev



CLOUDFLARE

Reservations for workers.dev

Requirements

- Limit reservations to one per email address.
- We only want to allow a single reservation per subdomain— need a reliable uniqueness constraint within the datastore on write.
- Blacklist a few key subdomains

CLOUDFLARE

Reservations for workers.dev

Requirements

- Limit reservations to one per email address.
- We only want to allow a single reservation per subdomain— need a reliable uniqueness constraint within the datastore on write.
- Blacklist a few key subdomains

Challenges

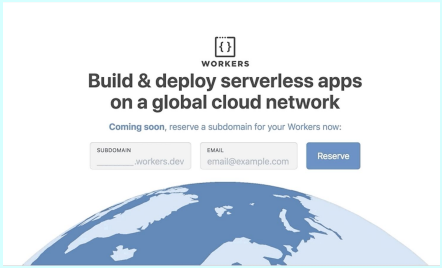
- Didn’t know how many sign-ups we were going to get
- Didn’t want something cumbersome to maintain long term
- Needed something *quickly*

CLOUDFLARE

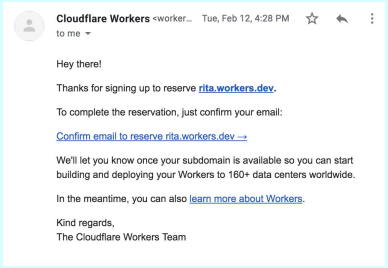
Reservations for workers.dev

Flow

1 Reserve subdomain



2 Send email



3 Confirmed!

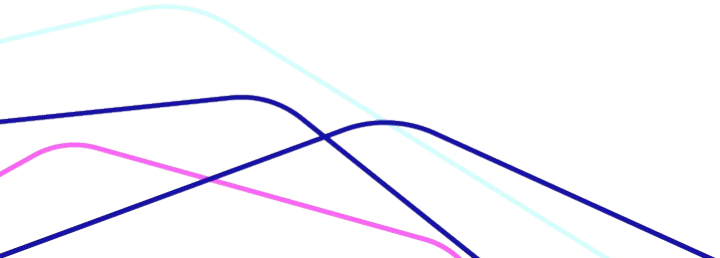
rita.workers.dev reserved by rita@cloudflare.com

CLOUDFLARE

Reservations for workers.dev

Two Workers

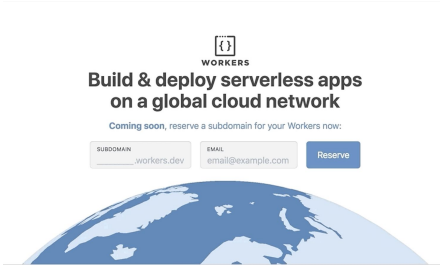
- Reserve subdomains
- Validate email



CLOUDFLARE

Reservations for workers.dev

Reserve subdomain Worker



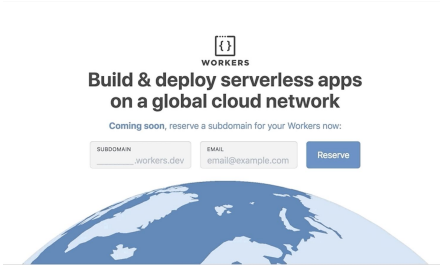
Firestore

reservations	
subdomain: rita	email: rita@cloudflare.com

CLOUDFLARE

Reservations for workers.dev

Reserve subdomain Worker



Firestore

reservations	
subdomain: rita	email: rita@cloudflare.com

- Auth to GCP
- Check subdomain not taken
- Check subdomain not blacklisted
- Write to Firestore

CLOUDFLARE

Reservations for workers.dev

Assemble data into JSON

```
const fs = require('fs')
const path = require('path')
const YAML = require('yaml-js')

// Service Definition for Cloud Firestore can be found here:
// https://github.com/googleapis/googleapis/blob/master/google/firestore/firestore_v1.yaml
// Service Account Config should be the JSON file you saved in the last step
let [serviceDefinitionPath, serviceAccountConfigPath] = process.argv.slice(2)

let serviceDefinition = YAML.load(fs.readFileSync(serviceDefinitionPath))
let serviceAccountConfig = require(path.resolve(serviceAccountConfigPath))

// JWT spec at https://developers.google.com/identity/protocols/OAuth2ServiceAccount#jwt-auth
let payload = {
  aud: `https://${serviceDefinition.name}/${serviceDefinition.apis[0].name}`,
  iss: serviceAccountConfig.client_email,
  sub: serviceAccountConfig.client_email,
}

let privateKey = serviceAccountConfig.private_key
let privateKeyID = serviceAccountConfig.private_key_id
let algorithm = 'RS256'
let url = `https://firestore.googleapis.com/v1beta1/projects/${serviceAccountConfig.project_id}/databases/(default)/documents`

// The object we want to send to KV
let FIREBASE_JWT_CONFIG = {
  payload,
  privateKey,
  privateKeyID,
  algorithm,
  url,
}

// Write out to JSON file to send to KV
fs.writeFileSync('./config/metadata.json', JSON.stringify(FIREBASE_JWT_CONFIG))

console.log('Worker metadata file created at', metadataFilename)
```

Auth to GCP using JWT

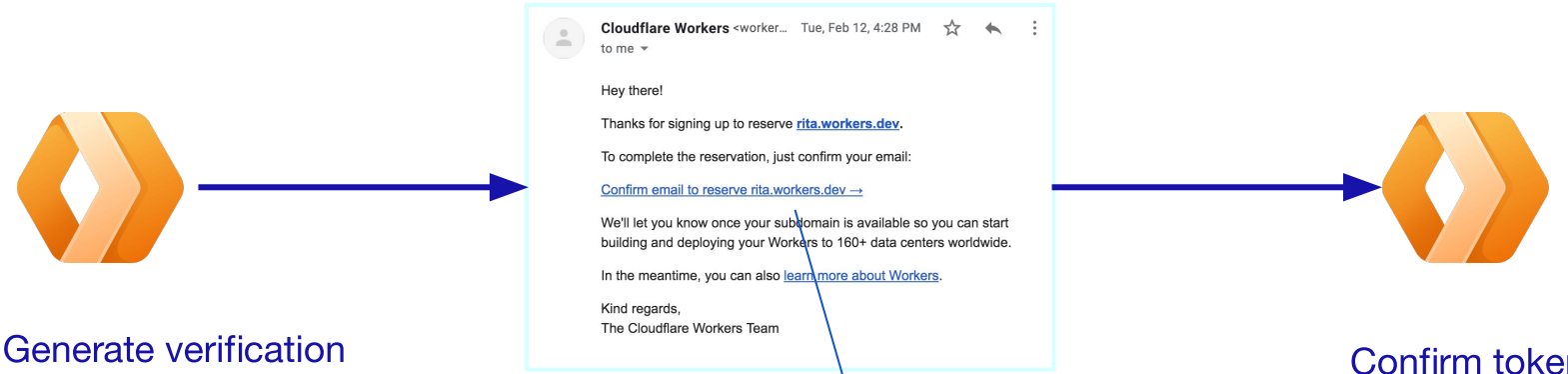
```
import jose from 'node-jose';

/**
 * Generate a Google Cloud API JWT
 *
 * @param config - the JWT configuration
 */
export default async function generateJWT(config) {
  const iat = new Date().getTime() / 1000;
  let payload = {
    ...config.payload,
    iat: iat,
    exp: iat + 3600
  };

  const signingKey = await jose.JWK.asKey(
    config.privateKey.replace(/\\n/g, '\n'),
    'pem'
  );

  const sign = await jose.JWS.createSign(
    { fields: { alg: config.algorithm, kid: config.privateKeyID } },
    signingKey
  )
    .update(JSON.stringify(payload), 'utf8')
    .final();

  const signature = sign.signatures[0];
  return [signature.protected, sign.payload, signature.signature].join('.');
}
```

https://blog.cloudflare.com/api-at-the-edge-workers-and-firestore/

CLOUDFLARE

Reservations for workers.dev

Verify email Worker



- Generate verification token & store in Firestore
- Use Mandrill API to send email

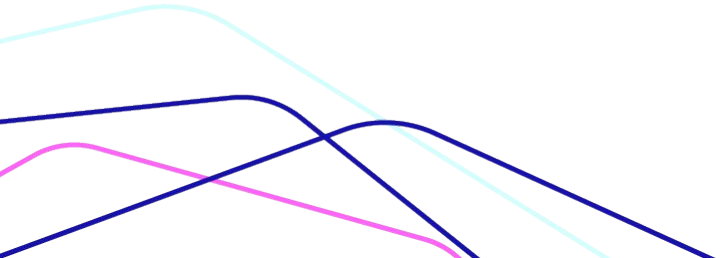
https://workers.dev/?**email**=rita%40cloudflare.com&**token**=ClO6kdDKSc8dDlVVnqJK&subdomain=rita

Confirm token

CLOUDFLARE

Reservations for workers.dev

- Successful launch
- Seamlessly scalable
- No double-bookings!
- No need to maintain
- Can be used with any APIs / cloud providers



CLOUDFLARE

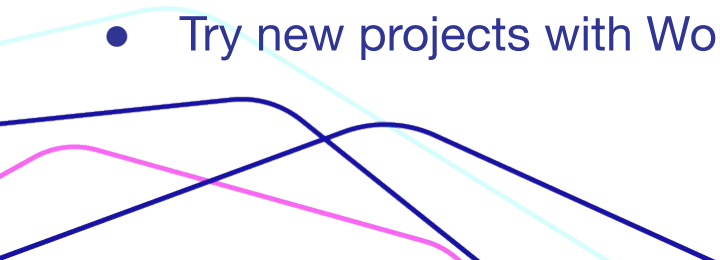
connect

What did we learn?

CLOUDFLARE

What did we learn?

- Start small!
  - Move functionality to the edge deliberately
- Break up endpoints into multiple Workers
  - Deploy frequently!
  - Work in small, independent teams
- Workers allow us to move quickly
  - Constantly iterate — higher engineering velocity
- Try new projects with Workers



CLOUDFLARE

connect

Thank you!
Questions?
rita@cloudflare.com

CLOUDFLARE