Olga Skobeleva
olga@cloudflare.com
Solutions Engineer

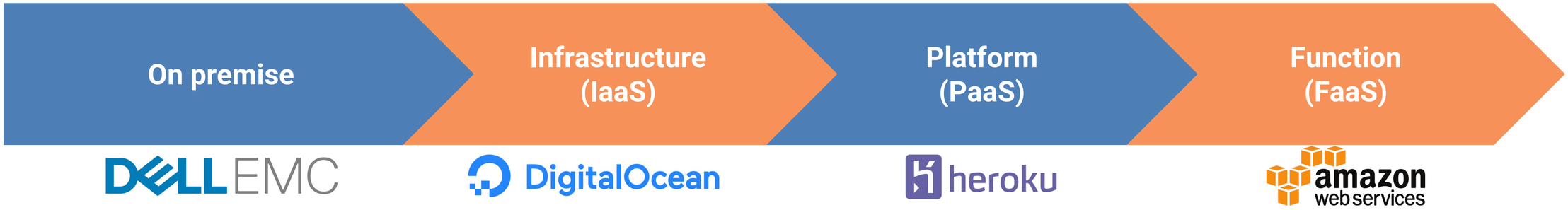
Security: the serverless future

How serverless in changing everything

Talking points

- Introduction to serverless: the future of compute
- Is FaaS what the Cloud was meant to be?
- Tools: Cloudflare Workers and other
- Examples
- Demo
- Conclusions

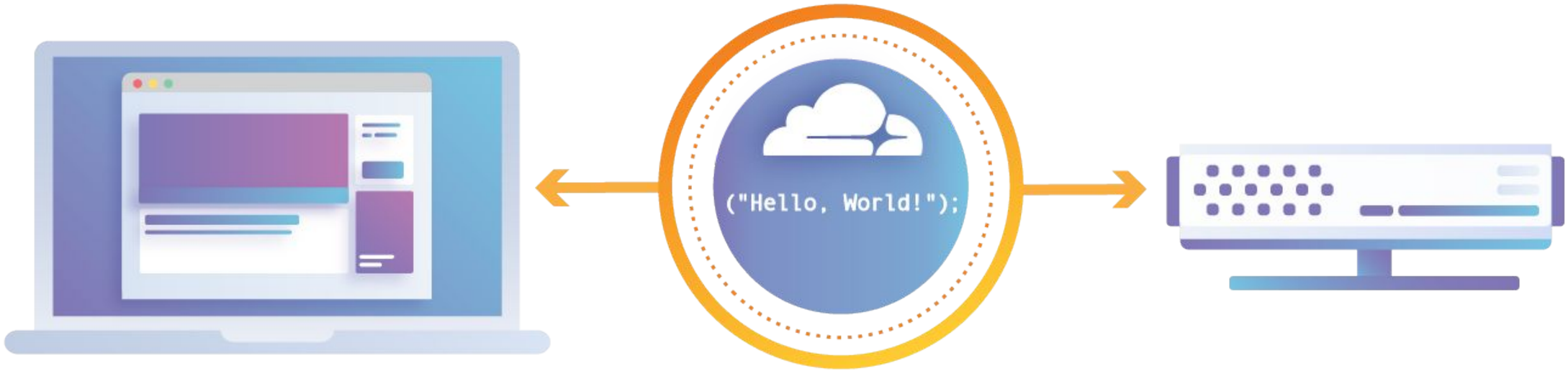
The story so far...



Just write code!

The next revolution will be
network based serverless.

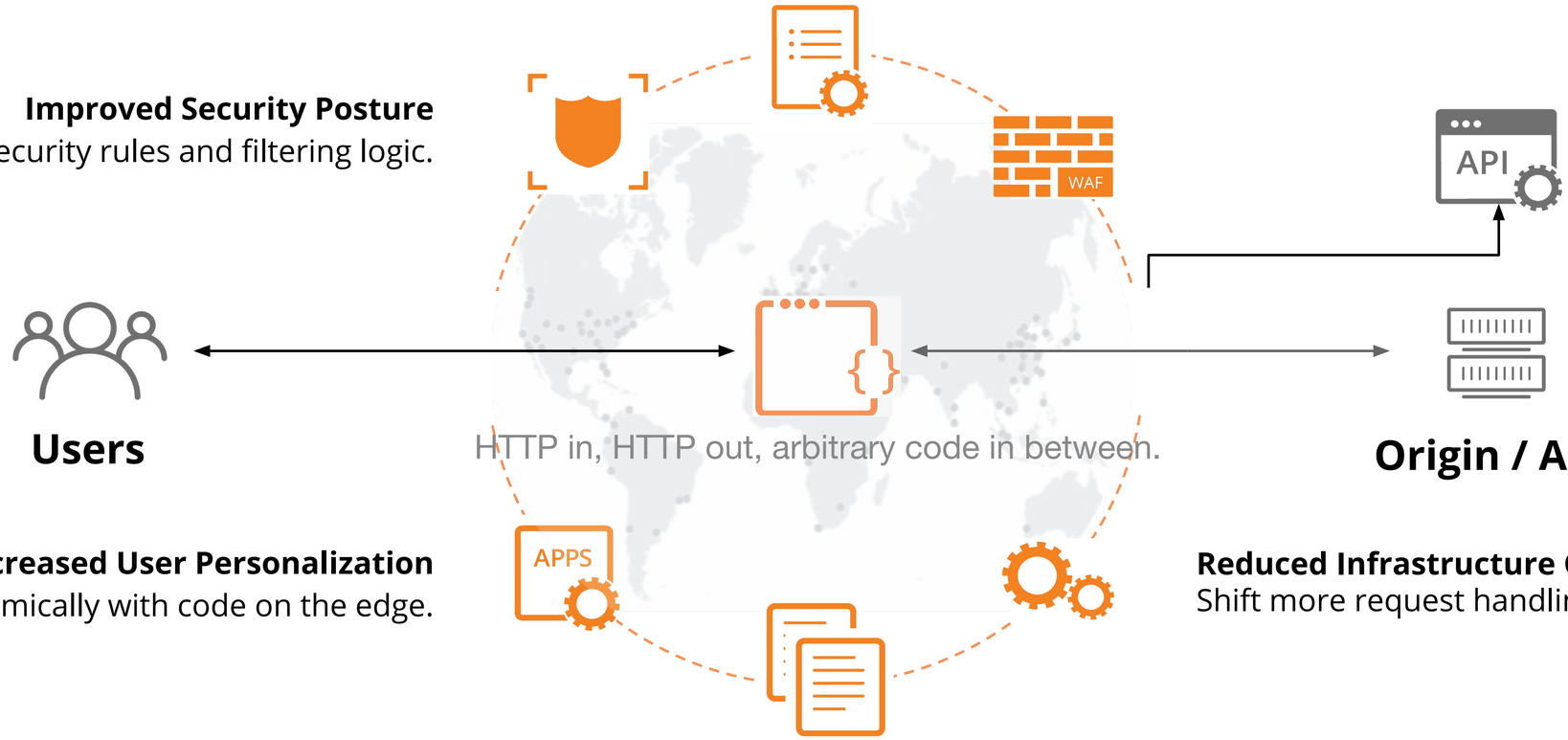
The third place to run the code



What is Cloudflare Workers?

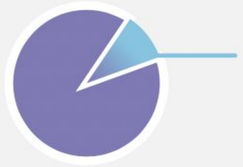
The Network is the Computer™

Improved Security Posture
Apply custom security rules and filtering logic.



Increased User Personalization
Respond dynamically with code on the edge.

Reduced Infrastructure Costs
Shift more request handling to the edge.



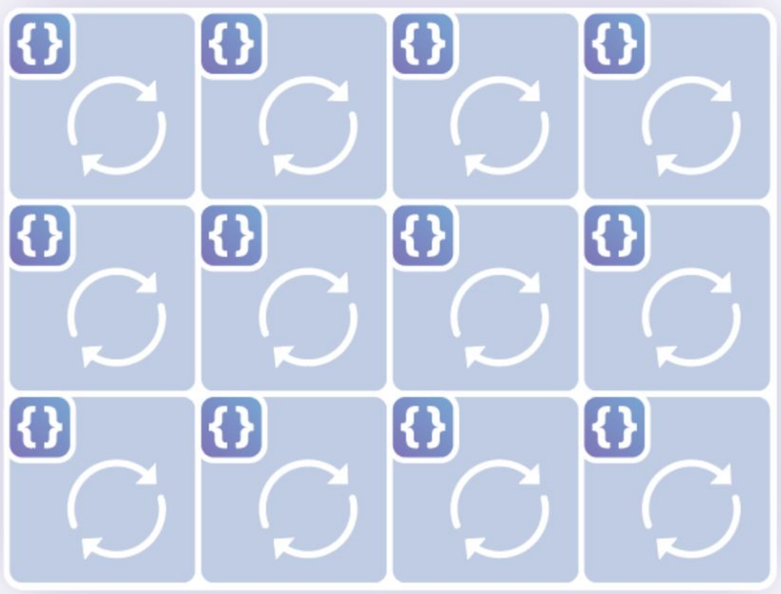
10% of all Internet requests

8 million Internet properties

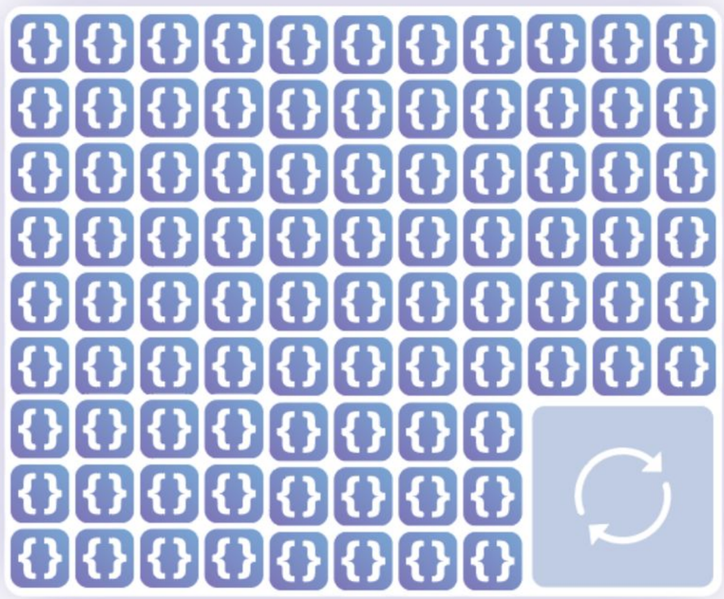
2.8 billion people worldwide

CLOUDFLARE

VM vs. Isolate

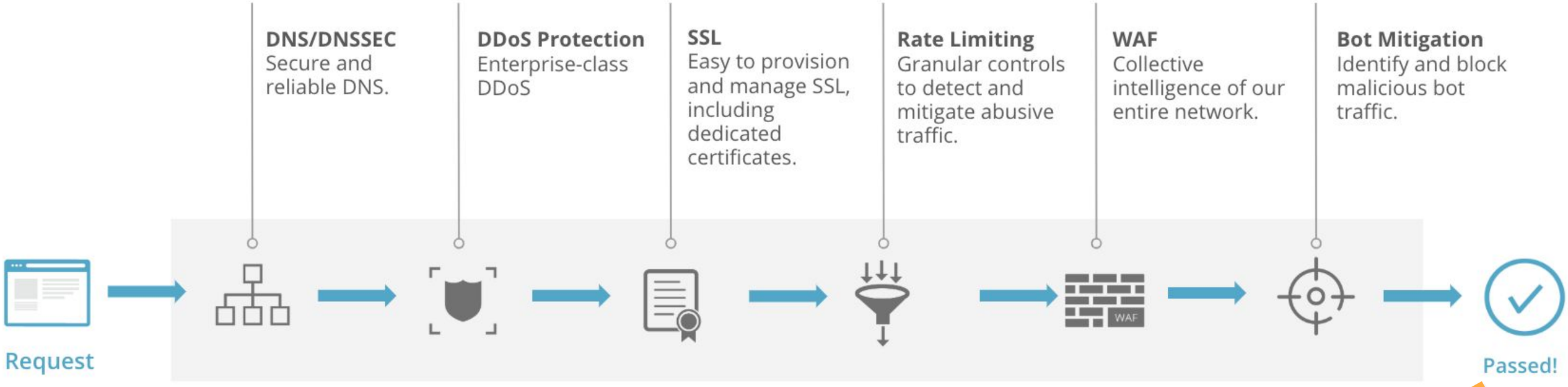


Virtual machine



Isolate model

Cloudflare Security: Life of a Request



Orbit
Secure and authenticated connection between an IoT device and origin.

Spectrum
Protect TCP applications and ports from volumetric DDoS attacks and data theft.

Access
Secure, authenticate, and monitor user access to any domain, application, or path on Cloudflare.

Argo Tunnel
Create an encrypted tunnel between an application's origin server and the nearest data center without opening a public inbound port.

Workers
Run JavaScript Service Workers to customize and configure apps on the edge.

Use cases today

1

Basic Routing / Header Modification
Let's them swap to and benefit from Cloudflare where they might not have been able to before.

Set Custom TTLs or Cache Keys
Let's them more effectively use the cache to improve user experience and reduce costs.

2

Advanced Routing / Rewrites / Caching
Even more effective use of the cache to improve user experience and further reduce costs and operational overhead.

Authorization / Authentication
By checking for things like existence of a cookie or header Cloudflare can respond faster to unauthed users making things faster and reducing load.

3

Microservice Built on Cloudflare
By building a service on Cloudflare Workers (e.g. a full auth service) orgs get to improve dev velocity with quick deploys / isolated logic.

Application Built on top of Cache
Cache heavy applications become much simpler to architect and maintain becoming

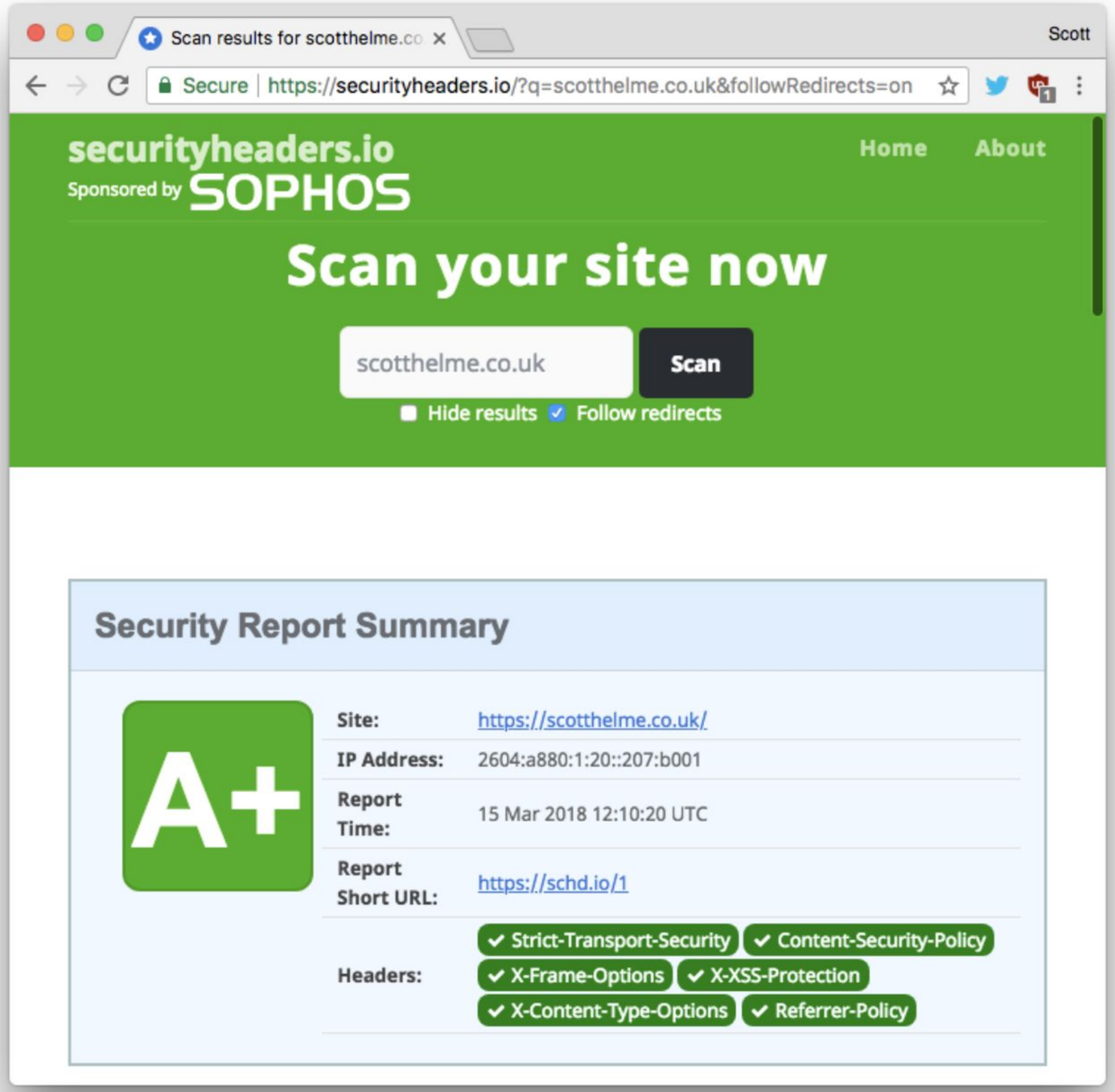
4

Multiple Serverless Microservices or Apps not necessarily Tied to Core CF
By using Workers, customers benefit from the developer productivity gains and platform speed of serverless.

Inject Security Headers on the fly

Deploy quickly and easily any security header of your choosing at the edge regardless the hosting solution.

Scott Helme
Security Researcher and founder of the popular securityheaders.com and report-uri.com, free tools to help people deploy better security.



https://securityheaders.com/
https://scotthelme.co.uk/security-headers-cloudflare-worker/

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


let securityHeaders = {
    "Content-Security-Policy" : "upgrade-insecure-requests",
    "Strict-Transport-Security" : "max-age=1000",
    "X-Xss-Protection" : "1; mode=block",
    "X-Frame-Options" : "DENY",
    "X-Content-Type-Options" : "nosniff",
    "Referrer-Policy" : "strict-origin-when-cross-origin",
}

let sanitiseHeaders = {
    "Server" : "My New Server Header!!!",
}

let removeHeaders = [
    "Public-Key-Pins",
    "X-Powered-By",
    "X-AspNet-Version",
]
```

```
!newHdrs.get("Content-Type").includes("text/html")) {
        return new Response(response.body , {
            status: response.status,
            statusText: response.statusText,
            headers: newHdrs
        })
    }

    Object.keys(securityHeaders).map(function(name,
index) {
        newHdrs.set(name, securityHeaders[name]);
    })

    Object.keys(sanitiseHeaders).map(function(name,
index) {
        newHdrs.set(name, sanitiseHeaders[name]);
    })

    removeHeaders.forEach(function(name){
        newHdrs.delete(name)
    })

    return new Response(response.body , {
        status: response.status,
        statusText: response.statusText,
        headers: newHdrs
    })
}
```

Identifying and alerting on data loss

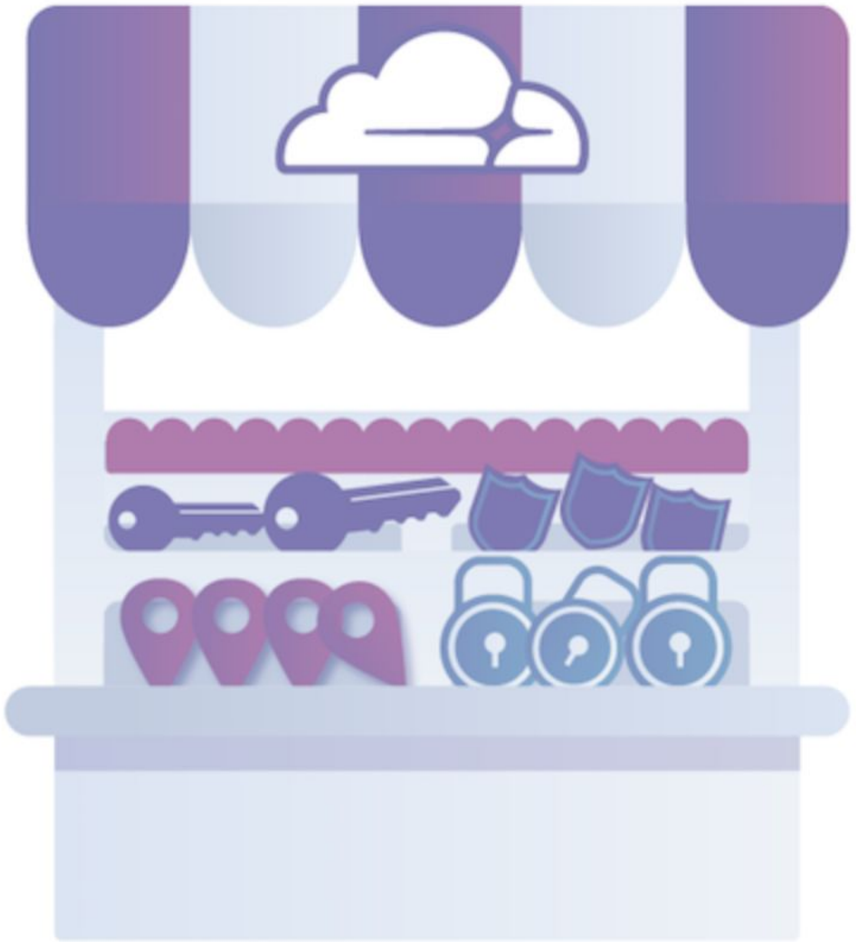
Since Cloudflare Workers sits between your infrastructure and the public for any endpoints exposed to the Internet, Workers can be used as a way of alerting you of canary data leaving.

https://blog.cloudflare.com/dogfooding-edge-workers/

Identifying pwned passwords

This year Troy Hunt launched his Pwned Password v2 service which has an API handled.

The following simple code can check if a password exists in Troy's database without sending the password to Troy.



https://blog.cloudflare.com/using-cloudflare-workers-to-identify-pwned-passwords/

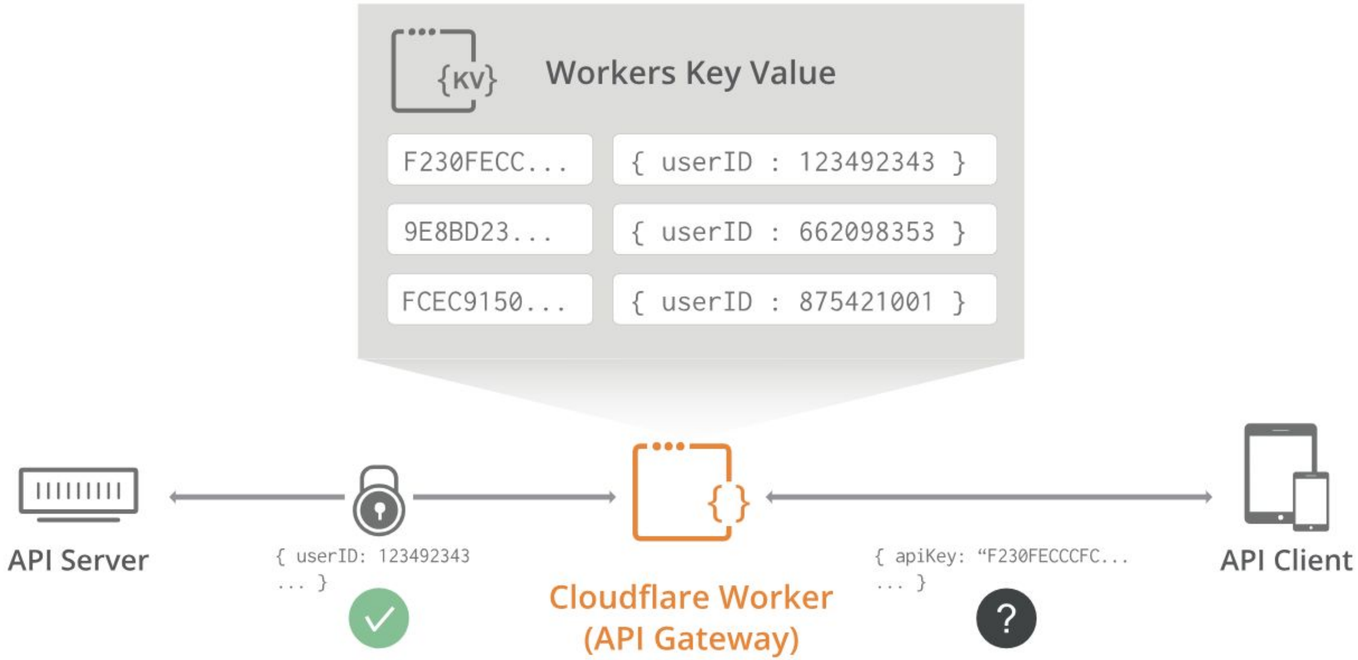
```
addEventListener('fetch', event => {
event.respondWith(fetchAndCheckPassword(event.request))
 })
  async function fetchAndCheckPassword(req) {
   if (req.method == "POST") {
     try {
       const post = await req.formData()
       const pwd = post.get('password')
       const enc = new TextEncoder("utf-8").encode(pwd)
        let hash = await crypto.subtle.digest("SHA-1", enc)
       let hashStr = hex(hash).toUpperCase()

       const prefix = hashStr.substring(0, 5)
       const suffix = hashStr.substring(5)
        const pwndpwds = await
fetch('https://api.pwnedpasswords.com/range/' + prefix)
       const t = await pwndpwds.text()
       const pwnd = t.includes(suffix)
        let newHdrs = new Headers(req.headers)
       newHdrs.set('Cf-Password-Pwnd', pwnd?'YES':'NO')
        const init = {
         method: 'POST',
         headers: newHdrs,
         body: post
        }
         return await fetch(req.url, init)
       } catch (err) {
         return new Response('Internal Error')
       }
     }

   return await fetch(req)
}
 function hex(a) {
  var h = ""
  var b = new Uint8Array(a)
  for(var i = 0; i < b.length; i++){
    var hi = b[i].toString(16)
    h += hi.length === 1?"0"+hi:hi
  }
  return h
}
```

Enforce custom authorization and authentication

Store session tokens or other user information to quickly ensure the right user has access to the right content. Move the authentication layer out of applications and into the network itself to improve site speed while reducing the threat of application-level authentication vulnerabilities.

Access token validation

When you use a proxy as an API Gateway, your access tokens get validated at the edge data center closest to the customer before the request is securely forwarded to your origin.



Access token validation

This request stores a blob of JSON identifying this token in a Worker KV namespace $NAMESPACE_ID with a key of $TOKEN_ID:

```
curl
https://api.cloudflare.com/client/v4/accounts/$ACCOUNT
_ID/storage/kv/$NAMESPACE_ID/values/$TOKEN_ID \
-X PUT \
-H "X-Auth-Key: $CLOUDFLARE_AUTH_KEY" \
-H "X-Auth-Email: $CLOUDFLARE_AUTH_EMAIL" \
-d '{
```

The Worker code, which runs on every request, will check if the token the user provides matches one you have stored. A single line of code (TOKEN_STORE.get()) pulls the JSON stored above from the KV

```
async function handleRequest(request) {
  const token = request.headers.get('Authorization')
  if (!token)
    return new Response("An Authorization header is
required", {status: 401})

  const tokenInfo = await TOKEN_STORE.get(token, "json")
  if (!tokenInfo)
    return new Response("Invalid token", {status: 403})
```

DEMO

Conclusions

- Serverless is the future
- If you’re not ready to go full serverless, start offloading single functions

Thank you!