LEAVING THE CDN BEHIND:

BUILDING A JAVASCRIPT SDK IN A SERVERLESS WORLD
WHO ARE YOU AGAIN?

- Daniel Brain / @bluepnume
- Still hacking away on the Checkout platform at PayPal!
- Last year: mainly working on open-sourcing our suite of cross-domain component libraries (google "paypal cross domain")
- This year: leading the effort to deliver up our sdks and components in much more interesting ways than boring old CDNs :)
So, let’s talk about SDKs
WHAT IS AN SDK, AND HOW DO YOU BUILD ONE?

- Traditional web SDKs wrap some existing API or interface, and make it super-simple to build against platforms in your language of choice:

  ```bash
  curl api.paypal.com/payment -d '{ "amount": "5.99" }'
  ...
  ```

  gets turned into an SDK ...

  ```javascript
  await paypal.createPayment({ amount: '5.99' });
  ```

- Pretty boring – and often these kinds of SDKs can just be autogenerated based off of an API spec or definition.
WE DID SOMETHING BETTER . . .

- Our next generation client SDK focused more on building UI that slots directly into third-party web-pages and native apps.
- The focus was on building native-feeling components, which could fit in seamlessly into your apps.
But... that’s not what I’m here to talk about today.
Let’s talk about how we package and distribute SDKs
Distributing SDKs? Pretty boring right? There are only really two basic ways of distributing a client-side javascript sdk, in a NodeJS platform:

1. You compile and bundle everything, then publish to NPM

2. You compile and bundle everything, then publish to a CDN

But both of these approaches have some downsides
NPM IS COOL, BUT:

- You end up supporting vast numbers of minor versions, each with the potential to regress in future – because you’re at the mercy of people deciding to manually upgrade npm dependencies.

- That becomes problematic when you have SDKs rendering UI, which needs to stay up-to-date and consistent with other experiences that are constantly being updated on your site.

- Take PayPal: we’re constantly adding new funding methods into new versions of our SDK. So NPM becomes tricky.
CDNS ARE A LITTLE BETTER:

- You have the choice of either doing versioned urls, or a single "evergreen" url. Evergreen makes introducing new fixes and features easy, but:
  - 1. Keeping backwards compatibility is painful. Every change added to the public interface has to live forever, including implicit things that people end up relying on.
  - 2. The script grows pretty big – which is no good for performance. It’s very easy to end up with a "one size fits all" bundled script which balloons in size over time.
So is there a better way?
WHAT'S THE ANSWER?

- Here’s what we came up with:
  
  **Before:** pre-bundled, static script, both on CDN and NPM
  
  ```html
  <script src="paypalobjects.com/checkout.js" />
  ```
  
  **After:** let’s do it live!
  
  ```html
  <script src="paypal.com/js/v4
          ?components=buttons,hosted-fields
          &client-id=xyz
          &locale=fr_FR" />
  ```
THE BASICS:

There’s no single bundled script; everything is webpack’d and minified on-demand on the server-side, then cached on our edge servers.

URL params are passed in the `<script>` tag, which dictates exactly which features (and components) are compiled in and shipped.

Each web-page integrating our SDK gets *only the specific code they need, customized exactly for them, on demand*.

```html
<script src="paypal.com/js/v4
    ?components=buttons,hosted-fields
    &client-id=xyz
    &locale=fr_FR" />
```
Let’s look at some specific use-cases
The SDK is only bundled with the exact localization content, images, even the specific code, needed for the specific site using it. Result = smaller javascript payload for everyone!
**BUT HOW?**

- Webpack + DefinePlugin + UglifyJS make this super-easy:

  ```javascript
  new webpack.DefinePlugin({
    __LOCALE__: req.query.locale
  })
  ```

  ```javascript
  if (__LOCALE__ === 'en_US') {
    content = require('./en_US.json');
  } else if (__LOCALE__ === 'fr_FR') {
    content = require('./fr_FR.json');
  }
  ```
LAZY-LOAD ANYTHING NOT ON THE HOT RENDER PATH

- Lazy loading! The core script only needs to contain enough code to do the first render.
- Everything else can be pulled in lazily, based on which components the client wants to render immediately, and which will be rendered later.
Webpack + import() has your back!

import('./lazyFeature')

  .then(({ lazyFeature }) =>
    lazyFeature.activate();

This automatically splits your code into jsonp bundles which are lazily required!
The SDK can now embed dynamic data pulled from GraphQL microservices, without any ajax calls slowing down renders!

**Before**

SDK ON CDN

GRAPHQL AJAX ENDPOINT

**After**

/JS/V4

EMBEDDED GRAPHQL DATA
BUT HOW?

- Webpack + DefinePlugin are all you need!

```javascript
query {
  MERCHANT_NEEDS_VENMO,
  MERCHANT_NEEDS_CREDIT
}

if (__GQL_CONFIG__.MERCHANT_NEEDS_VENMO) {
  // Include Venmo code, assets, content
}

if (__GQL_CONFIG__.MERCHANT_NEEDS_CREDIT) {
  // Include Credit code, assets, content
}
```
We’ve set things up so our SDK can dynamically pull data in from any GraphQL microservice that we deploy on the fly:
INLINED CONFIG ALSO LEADS TO MUCH FASTER RENDERS

- Not having to do Ajax calls for config, speeds up the render path:

  Before

  ![Before Image]

  (ajax call to fetch config)

  After

  ![After Image]
Configuration from the server side can dictate rendering decisions on client-by-client basis, to enable as many funding sources as possible and improve conversion and buyer experience:
HOW ABOUT VERSIONING?

- The resulting SDK is evergreen; new features, bug-fixes, security fixes are enabled automatically without anyone re-integrating.
- Backwards incompatible changes can be made, and old features deprecated, by persisting the date when clients integrate.
- Clients can manually opt-in to newer versions with a `version` url parameter.

```html
<script src="paypal.com/js/v4?version=2018/07/25"/>
```
Webpack + DefinePlugin + UglifyJS make this super-easy:

```javascript
if (__FEATURES__.DEPRECATED_FEATURE) {

  // Code for old, deprecated feature

  // Now newer integrations won’t
  // get the old code in their
  // js payload!
}
```
**LIVE DEPLOYS**

- Deploys are triggered by npm publish (check out `grabthar` on npm to try this yourself).

- No need to ship `dist/`, just `src/`! Server-side web pack handles the rest.

**Before**

```
PAYPAL ENGINEERING
  ↓
NPM RUN BUILD
  ↓
NPM
  ↓
CDN
  ↓
MERCHANTS
```

**After**

```
PAYPAL ENGINEERING
  ↓
NPM
  ↓
/JS/V4
  ↓
MERCHANTS
```
We’re currently beta-testing this, but we’ve had excellent results so far.

Come try it with us! We’re hiring!