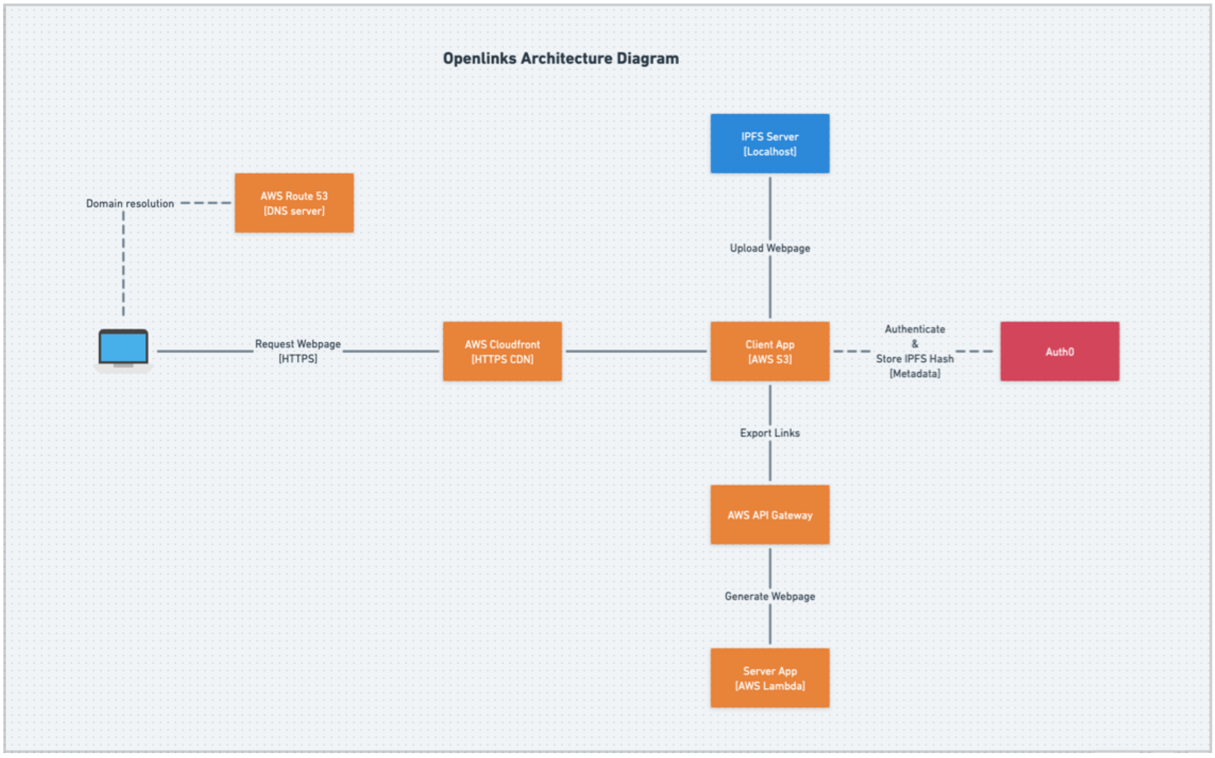
Openlinks Documentation

TL;DR: The whole application is deployed on AWS platform as follows to make this all work!



Client

The client side application is mainly built on **React (javascript)** and makes use of the following npm libraries:

- **ipfs-http-client** to communicate with local IPFS server over HTTP. Each request is handled by IPFS server, hosted on your local machine.
- **auth0-js** to use auth0 as identity and user's metadata management platform.

One challenge in communication between the application and chromium browsers was the *CORS policy* to disallow **remote** HTTP requests to **localhost**. To overcome this, the client app is served through HTTPS (TLS) on the domain, setup through Route53 and Cloudfront.

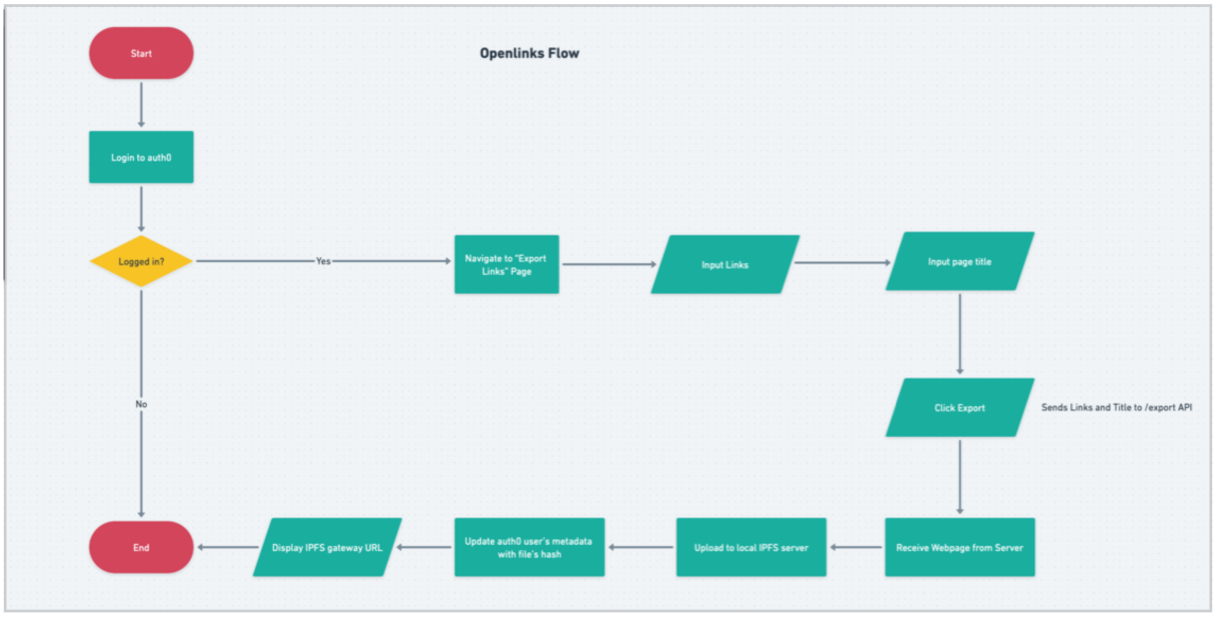
The build from react generates a single page app, which is uploaded to an S3 bucket (openlinks) on AWS. Looking at the diagram, you may realise that the S3 bucket is basically being pointed by the Cloudfront (CDN) distribution. Therefore, to put simply, the domain (openlinks.io on Route53) is configured to resolve to this Cloudfront distribution, which is then configured to point at the `openlinks` S3 bucket.

Server

As depicted in the architecture, the server is built around AWS lambda and depends on `ejs` template engine to generate a webpage. The AWS lambda resource is accessed through a CORS enabled API Gateway. The API gateway currently only exposes `/export` (REST) API.

Due to **lambda proxy integration**, the API Gateway forwards the requests to a single Lambda function as-is, which maps it to the required functionality, in this case, generating a webpage for the client app through a template engine.

Flow


Openlinks Flow
Start
Login to auth0
Logged in?
Yes
Navigate to "Export Links" Page
Input Links
Input page title
Click Export
Sends Links and Title to /export API
Receive Webpage from Server
Upload to local IPFS server
Update auth0 user's metadata with file's hash
Display IPFS gateway URL
End
No