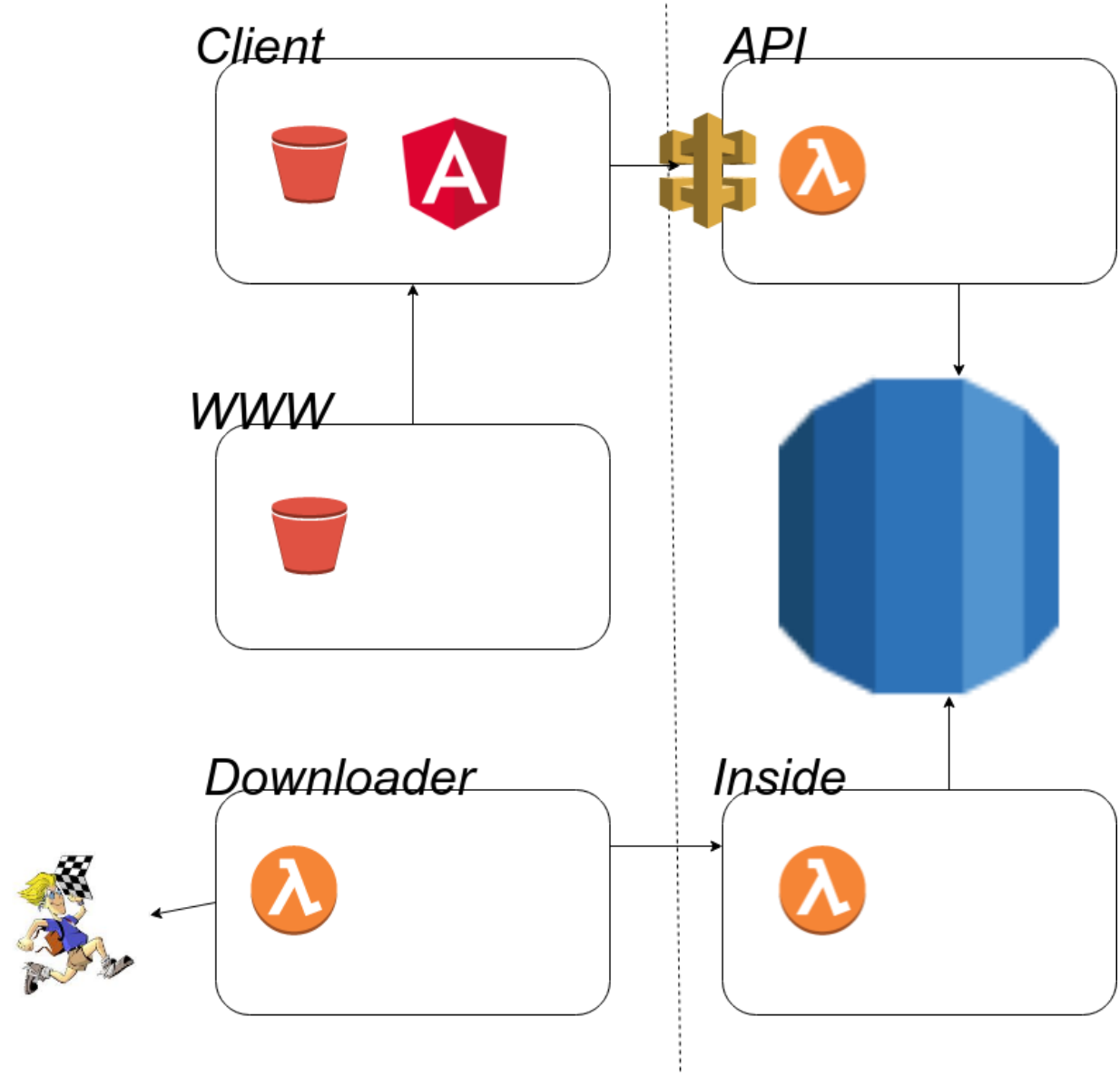
Extended Stats Serverless Developer Documentation

John Farrell
January 12 2019

System Architecture

Extended Stats Serverless is a (mostly) serverless architecture. It consists of a number of large components which mostly correspond to CloudFormation stacks. The downloader is implemented by the Downloader and Inside stacks. The site is implemented by the Client and API stacks.



The client stack implements extstats.drfriendless.com, and the API stack implements api.drfriendless.com. The client consists of HTML (written with Mustache) and JavaScript (written as Angular). The files produced for the client are uploaded to S3 and served from there via the CloudFront CDN. The Angular makes HTTPS calls to the API, which is implemented as an API Gateway proxying Lambdas and an Express server running on an EC2. There are several blog posts at blog.drfriendless.com about why there's an EC2 in there. That's the bit that's not serverless.

AWS Bills

Like it or not, and I don't very much, this project costs money to run, and it's my money. Furthermore it's an unbounded amount of money, as there are costs for every call to the API, all the data transferred out, and so on. This means that as the site becomes used more, costs will increase, but also that bugs which cause too much resource usage will cost me money as well. I mention this on the blog a lot.

Anyway, this means that I need vet any code that's deployed in a way that will cost me money. That will probably slow the development process down, for which I apologise.

Developer Engagement

Nevertheless, I hope to build the system so that other developers can experiment with their own stats ideas. It's a fun thing to do. My intent is that API will be callable by third parties, so people will be able to create components that are equivalent to the Angular components in Client without necessarily being on the site. Furthermore, I'm trying to ensure that Angular is not the only JavaScript technology that can be used.

This means I will need to document the API calls! Watch this space…

API Gateway allows me to require an API key for an API to be called. I have turned this option on, and have given myself an API key with a large daily quota. I can generate further API keys for other developers, with a quota that I'm comfortable with. My intent is to eventually lock the API keys to authentication, so that API keys can't be stolen, but I haven't got to that yet.

Repositories

ExtendedStatsServerless

https://github.com/DrFriendless/ExtendedStatsServerless

This is the main repo which contains all of the stacks. Any code which is shared between stacks (or even client apps) gets put into one of the libraries below. This repo has got quite big and unwieldy.

Core

https://github.com/DrFriendless/extstats-core
https://www.npmjs.com/package/extstats-core

Interfaces used by the API, in TypeScript. Importantly, this package specifies what the API will accept and what it returns.

Angular

https://github.com/DrFriendless/extstats-angular
https://www.npmjs.com/package/extstats-angular

Angular component library for use in client apps. As I discover reusable parts of the code, I move them into here.

Vega

https://github.com/DrFriendless/extstats-vega
https://www.npmjs.com/package/extstats-vega

Common code for dealing with Vega. I'm not sure this abstraction is achieving a lot so it might eventually go away. Vega is wonderful, but complex to integrate with Angular.

Datatable

https://github.com/DrFriendless/extstats-datatable
https://www.npmjs.com/package/extstats-datatable

When I was doing the first pages I wanted a nice data table, but I had to settle for the only one that I could get to work with Angular 6. I cloned the repo so I could develop it to suit my own purposes, but I haven't got back to that yet.

The table component will need to solve the problem of there being thousands of results being displayed 20 per page. It has a pager, but I would like to add a search box that took you to the page of results containing the text you were searching for.

The API

query

URL: https://api.drfriendless.com/v1/query
HEADERS: x-api-key, Content-Type: application/json
POST

This is the primary API for retrieving data from the database. Usually you will just need to call it once to populate a page. The body of the call is an instance of GeekGameQuery. The format key specifies what type the result should be, and hence what data it should contain.

The implementation is at:
https://github.com/DrFriendless/ExtendedStatsServerless/blob/master/api/mysql-rds.ts

I did consider doing this with GraphQL, but I felt that the patterns that were applicable to this site didn't warrant a more general graph query capability.

The Database

I use a relational database (MySQL) to store the data. I very seriously considered a NoSQL database, but the nature of the site is to slice the data in various ways, which very much matches the relational model. The schema definition is here:

https://github.com/DrFriendless/ExtendedStatsServerless/blob/master/misc/schema.sql