1. Project
   1. Create a RESTful web interface that will receive a request with a number and it will return that number of entries in the fibonacci sequence
   2. Negative and non integer requests will respond with an error
   3. It needs to be able to scale to support as many users as need.
   4. It needs to be easy to deploy
   5. We will need to have a load balancer in-front of the web service to allow for scaling.
2. Plan
   1. Using python and flask we will create a standalone web service that will receive and respond to the web request.
   2. We are not going to calculate the value as the request comes in. Since whatever request comes in, necessarily solves all requests that are less than that, we might as just solve for all the available values to us.
   3. ~~Our maximum request will have to be less the max value of a signed integer, 9,223,372,036,854,775,807. We will not return any sequence that has a last value greater than that.~~ The only limit to the size of the integer in python is ram.
   4. ~~F(93) is the first value that has 20 digits in it. We can expect to only have to do F(92), with is about 7,000,000,000,000,000,000 so less than our max.~~
   5. ~~We will store all entries in an array. We will just walk the array for the length we need then return the json~~ We are going to keep the local cache as a file and store the bitcount offset in the redis array. That way we don’t take up ridiculous amount of ram for each web server. They will only be limited by the disk size.
3. Implementation
   1. There will be the main parts. The load balancer and the webservice.
   2. WebService
      1. First populate the array with some amount of numbers
      2. Then start responding to requests using the array
      3. Any time we need to get more just go to the redis server and add it to the cache.
   3. Load Balancer
      1. Used to run multiple instances of the webservice.
   4. Redis populator
      1. We will have a client running the the back ground that will just sit and spit out numbers adding them to redis.
   5. Updater
      1. Since we will have multiple web service instances running on the same server we can only have one instance updating the local cache. That will be start on startup and just update the cache as needed.
4. Docker
   1. There will be docker images for the redis cluster clients, web service, testing, load balancing.
5. ~~RPM’s I also want to provide RPM to do the install.~~

Notes:

Length and estimation calculators can be found here.

http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/Fibonacci/fibFormula.html#calcLog