ForgeRock Coding Challenge - Don Welch

Build Instructions

None ;-)

Demo Access

Sign-in URL: https://730403596516.signin.aws.amazon.com/console
User name: `forgerock_reviewer`
Password: `*#6[J]5n*_4]`

Demo Steps

1. Head over to the S3 page and go into the `sandbox.dwelch91.org` bucket. Once in the bucket, click `Upload` and upload the `.mxf` file from the `samples/` directory into the bucket.
2. Once this is done, several things will happen over the course of a few 10's of seconds:
   a) Over in Cloudwatch logs, there will be logs for `controller`, `worker1`, `worker2`, and `worker3`. Worker 1 is setup to resize the input video, worker 2 flips the video on its side, and worker 3 always fails (by design). The controller dispatches jobs, receives progress and completion notifications, and saves the state of things in the database (seen next item).
   b) The progression of of the job and final completion result is recorded in DynamoDb in the `Jobs` table. If you take a look, you should see that worker 1 and worker 2 were running concurrently and that worker 3 is run after worker 2 is complete. This table is being used in an "append-only" style where each progress/job update is appended to the previous and sorted/accessed by a timestamp (a basic "event sourcing" pattern)
   c) Head back over to S3, and there sould be 2 output files identified by their worker numbers that can be downloaded and played in VLC, etc.

Discussion

1. Some thoughts about using Lambdas (specifically) for this sort of use-case:
   a) Lambdas are difficult to debug - you cannot (easily) use an actual debugger (it is possible to simulate the environment locally, but getting that all setup is a big chore). You are resigned to mainly using logging and exceptions to get visibility into what's occurring. Fortunately, Python makes this very easy with the logging library and `f-strings` (love those things!)
   b) Similarly, creating anything but low-level unit tests is difficult. There's a lot of AWS service behavior that would be a lot of effort to mock/stub. I pretty much stuck to testing the lower-level items like small data manipulation routines.
   c) In this toy example, I've given the Lambdas free-reign over many AWS services - obviously a bad practice. For production, you'd want to reduce the scope of the permissions (via the role) to as narrow a scope as practical.

d) Even though the effort of setting this up is more than if it were done locally, the concurrency possible (1000+ workers at a time) is hard to beat. And, you get it for free (effort, not $$$:-). You should be able to "pour" in as many files as you can into the input bucket and it should do all the jobs concurrently (not tested!) (you'd also hit some Dynamo limits if you over-did it without making some adjustments first).

2. General shortcomings:
   a) Error handling is sparse!
   b) The ability to pass-in what transform you want each worker to perform would be a big upgrade.
   c) With more effort/time, a more respectible test suite could be developed.
   d) It seems that I didn't turn on the h264 flags for the Amazon Linux compiled version of `ffmpeg` - if you drop a h264 file, ffmpeg will not have a good day.
   e) You get concurrent scaling from AWS lambdas "automatically", the way to "scale" this toy example would be to allow for more control over what runs in what order (likely would want to implement some sort of graph to model the steps, perhaps a Petri-net) and allow for customizable worker task defintions. With the appropriate workflow network/graph in place, and Lambda's concurrency, doing sequential and parallel tasks in a flexible way would be fairly strightforward to implement.

Final Steps

Please let me know when the team is done reviewing this demo so that I can delete the account and not risk incurring any significant costs.