Task UFC-346: Caching Multi-Stage Builds

tl;dr

We now build and push each stage of the multi-stage prod builds, so that they can all be cached properly.

Had to switch to using `node:14-alpine` and `postgres:11` (i.e. non-specific versions) because of a bug in docker/BuildKit that makes it not pull from Docker Hub if a registry mirror is in place. That is, GCR has a registry mirror, only the non-patch versions of node/postgres are in there, so we need to use those versions since BuildKit is stupid and won't go to Docker Hub if the image isn't in the mirror.

This is likely to prove problematic in the future.

The solution (if it were to break again) would be to just pull each `FROM` image individually. Yeah, *super* jank.

The Slack Comments

I feel like the cache images aren't being properly used with the new production images. They always seem to run... basically all the steps, particularly the `npm` install steps. At least for the backend/marketing images, I suspect it has to do with the `mv` command for the production `node_modules`, but that wouldn't explain why the frontend is also always installing everything.Test it locally by specifying `--cache-image`.OMFG, it's cause of the multi-stage builds. The first stage can't be cached because only the second stage is pushed.FUCKRelevant discussion: https://github.com/moby/moby/issues/34715Seems like Docker BuildKit can help with this? Look into it...
Relevant: https://www.docker.com/blog/advanced-dockerfiles-faster-builds-and-smaller-images-using-buildkit-and-multistage-builds/ (edited)

Today

--
https://pythonspeed.com/articles/faster-multi-stage-builds/

--
Allow the kubails image attribute to accept an array? Then have it be like "backend-base", "backend"? Then change the name of the first stage to match? E.g. instead of build-env, it'd be backend-base? Then the build and push commands build and push the stages separately so that they can be cached separately?

--
https://github.com/moby/moby/issues/39003

--

https://stackoverflow.com/questions/54574821/docker-build-not-using-cache-when-copying-gemfile-while-using-cache-from

--
OK, so my understanding is that caching should now be done as follows:- Use BuildKit. This can be done by setting `DOCKER_BUILDKIT=1` before calling any docker commands.
- Add `--build-arg BUILDKIT_INLINE_CACHE=1` to all `docker build` commands
- Build each stage separately
- Push each stage separately
- Use `--cache-from` to include the first stage image when building the second stageSo now we need to modify Kubails to accommodate all this. (edited)

--
Make sure to write a doc on ^And include a section on why BuildKit doesn't play nicely with Cloud Build's mirror (https://github.com/moby/buildkit/issues/1972) and why we need to use node:14-alpine and postgres:11 instead of specific versions (cause the mirror has those but not our specific patches).Also, TECH DEBT: How we inject the database password.

Postgres Password

We implemented the management of the Postgres password as a 'secret' (i.e. encrypted) file that is stored in the root of the repo, decrypted during build time, and then injected into `kubails.json` before generating the manifests (ala how we handle the Stripe webhook secret).

This is pretty jank. But I mostly chose this route because it didn't require giving the database it's own service. Which actually wouldn't have worked, in the end, because multiple services (backend, postgres, migration-job, backup-job) need the password.

So... really, the only janky part is that we inject it into the kubails config. We should probably just have a Kubails command that handles that for us.