Bulk Image Processing w/ Kubernetes
A post-mortem on a successful project

Mike Newswanger
$ whoami

Mike Newswanger

@mnewswanger

elastic
- Overview of the project
- Project architecture
- Implementation plan
- What went right?
- What went wrong?
Project Overview - The Company

- E-commerce company selling restaurant equipment
- Over 9,000,000 images across web properties (~2TB)
- Infrastructure collocated on bare metal
Guetzli is a JPEG encoder that aims for excellent compression density at high visual quality.

github.com/google/guetzli
Is it worth it?
Project Overview - Is it worth it?

- Local Proof-of-Concept using container
- Collect metrics around the process
- Extrapolate
Project Overview - Is it worth it?

Image Sizes

- 100
- 80
- 25
Project Overview - Is it worth it?

Processing Time

#VelocityConf
Project Overview - Is it worth it?

- Performance is $O(n)$ relative to pixel count
- Memory consumption is $O(n)$ relative to pixel count
- Pixel count is $O(n^2)$ relative to single-side dimension
Decision Time
Project Overview - Decision Time

- Trade CPU cycles for gains in bandwidth and storage
- Overhead is sustainable over time
- Initial batch work is too much for existing hardware
- Overview of the project
- Project architecture
- Implementation plan
- What went right?
- What went wrong?
Project Architecture - Existing Process

- Photo team acquires and edits image for the site
- Photo gets uploaded to internal CDN
- Photo gets dynamically resized on request and cached
- Photo gets cached at public CDN
Well, that’s not going to work
Project Architecture - Existing Process

- Photo team acquires and edits image for the site
- Photo gets uploaded to internal CDN
- Photo gets dynamically resized on request and cached
- Photo gets cached at public CDN
Project Architecture - Ideal workflow

- Photo team acquires and edits image for the site
- Photo gets uploaded to internal CDN
- Job gets created to optimize image and clean cache
- Photo gets dynamically resized on request and cached
- Photo gets cached at public CDN
Sometimes we need to make compromises
Project Architecture - What’s important?

- Existing images get processed for optimization ASAP
- New images can get processed eventually
- Process can move from cloud-based to on-prem
Project Architecture - The Approach

- Collect list of image files on disk
- Load into SQL engine
- Generate list of image files that should exist in cache
- Compare and build queue
- Process queue items
Project Architecture - Why SQL?

- SQL supports locking necessary for concurrency
- SQL is very good at comparisons across data sets
- Integration is easy
Project Architecture - Kubernetes

- Development can be done in a local container
- Scaling is super easy
- Process can be moved on-prem easily
- State can be handled in SQL
- Overview of the project
- Project architecture
- Implementation plan
- What went right?
- What went wrong?
Implementation Plan - How long will it take?

- Over 9,000,000 images
- Each product has multiple known image sizes
- On average, products took ~300 seconds
- Total processing time ~4340 CPU days
Implementation Plan - Timeline

0 1 2 3 4 5 6 7 8 9 10 11 12

PoC Done
Begin Data Processing
Weekend’s Over
Plan & Implement
Time’s Up

#VelocityConf
Implementation Plan - Tasks

- Set up GCP environment
- Generate file list generation processes
- Write queue imports and comparisons
- Write processing applications
Implementation Plan - Set up GCP

- Sign up
- Use CLI to configure GKE
- Ask nicely to get higher quotas
Implementation Plan - Generate file lists

- Perfect task for Jr. SRE
- Generate file spec requirements
- With spec defined, import process can continue
Implementation Plan - Queueing

- Set up Postgres
- Import uploaded source files to a table
- Import optimized file list to another table
- Index, compare, and generate work queue
Implementation Plan - Processing Application

- Go application
- Pull original image from origin
- Resize image via ImageMagick
- Optimize with guetzli
- Publish finalized image
- Overview of the project
- Project architecture
- Implementation plan
- What went right?
- What went wrong?
What went right?

Production workloads behaved similarly to Proof-of-Concept
What went right?

Retry and timeout logic and limits worked as expected
What went right?

Metrics on both processing nodes and origin resources were available and useful
What went right?

Iterative process produced a successful result
- Overview of the project
- Project architecture
- Implementation plan
- What went right?
- What went wrong?
What went wrong?

Origin server was overloaded
What went wrong?

Queue work selection process had contention and was locking
What went wrong?

Download performance to our datacenters was too slow
What went wrong?

Really large images failed to complete
What went wrong?

Some images’ colors were off
Final Results

- Reduced images on page loads by ~75%
- Reduced overall image storage from ~2TB -> ~500GB
- Maintained > 99% CPU across 2,000 cores
- Scaled to 3,600 containers
- Had a maintainable, working pipeline