MIGRATING DATA BETWEEN CLOUDS

Jozef Cipa, Backend Engineer at STRV

STRV

QUICK INTRO

- **Surge** - 4.5M+ registered users
- **Grizzly** - 1M+ registered users

- Runs on Heroku, Postgres & Firebase
- Data originally stored in Azure (expensive 🤑)
- Let's migrate to AWS
    - Push notifications (moved to Firebase)
    - Schedulers (refactored in code)
    - **Photos & chat files** (migrated to S3)

PROFILE PHOTOS

1. Change presigned URL endpoints to serve new S3 links
2. Prepare a migration script
    1. Fetch profile photos from DB (4 sizes per photo)
    2. Download file from Azure
    3. Upload file to AWS S3
    4. Update file links in DB & Firebase

Surge ~2TB

Grizzly ~1TB


STARZ

LET'S RUN IT!

- Separate Heroku process
- Est. completion time: ~ 8 months (~700 profiles / hour)

LET'S RUN IT!

- Separate Heroku process
- Est. completion time: ~ 8 months (~700 profiles / hour)

We can do better

- Introduce Node.js **cluster** (with 3 workers)
    - Need to orchestrate data access among workers

LET'S RUN IT!

- Separate Heroku process
- Est. completion time: ~ 8 months (~700 profiles / hour)

We can do better

- Introduce Node.js **cluster** (with 3 workers)
    - Need to orchestrate data access among workers
- Est. completion time: ~ 1.5 month (~3,900 profiles / hour)

LET'S RUN IT!

- Separate Heroku process
- Est. completion time: ~ 8 months (~700 profiles / hour)

We can do better

- Introduce Node.js **cluster** (with 3 workers)
    - Need to orchestrate data access among workers
- Est. completion time: ~ 1.5 month (~3,900 profiles / hour)

Can we do even better?

- Move to AWS EC2 (with 6 workers)

LET'S RUN IT!

- Separate Heroku process
- Est. completion time: ~ 5 months (~700 profiles / hour)

We can do better

- Introduce Node.js **cluster** (with 3 workers)
    - Need to orchestrate data access among workers
- Est. completion time: ~ 1.5 month (~3,900 profiles / hour)

Can we do even better?

- Move to AWS EC2 (with 6 workers)
- Est. completion time: ~ 7 days (~24,000 profiles / hour)

LET'S RUN IT!

- Separate Heroku process
- Est. completion time: ~ 5 months (~700 profiles / hour)

We can do better

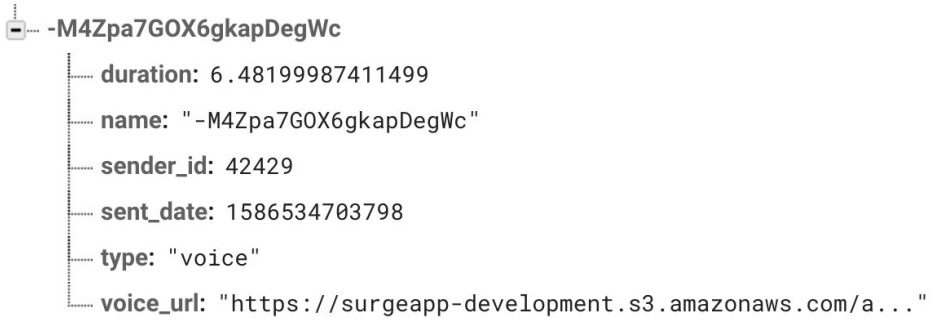
- Introduce Node.js **cluster** (with 3 workers)
    - Need to orchestrate data access among workers
- Est. completion time: ~ 1.5 month (~3,900 profiles / hour)

Can we do even better?

- Move to AWS EC2 (with 6 workers)
- Est. completion time: ~ 7 days (~24,000 profiles / hour)

CHAT FILES

- Firebase (Realtime database, not Firestore)
- Photos, videos, voice messages
- DB tables
  - `snaps`
  - `video_snaps`
  - `voice_messages`



CHAT FILES

- Firebase (Realtime database, not Firestore)
- Photos, videos, voice messages
- DB tables
  - `snaps`
  - `video_snaps`
  - ~~`voice_messages`~~ ❌😒

CHAT FILES

- Firebase (Realtime database, not Firestore)
- Photos, videos, voice messages
- DB tables
  - `snaps`
  - `video_snaps`
  - ~~`voice_messages`~~ ❌😔
- Iterate through the entire collection `msg_messages/293-851`

```
const getNextConversation = async conversationKey => {
  const result = await app.context.firebase
    .ref('msg_messages')
    .orderByKey()
    .startAt(conversationKey)
    // get 2 conversations, first one to process, second one to get nextConversationKey
    .limitToFirst(2)
    .once('value')
  const conversations = Object.entries(result.val())
```


WORKED FINE IN DEV
OPS PROBLEM NOW
memegenerator.net

REALITY

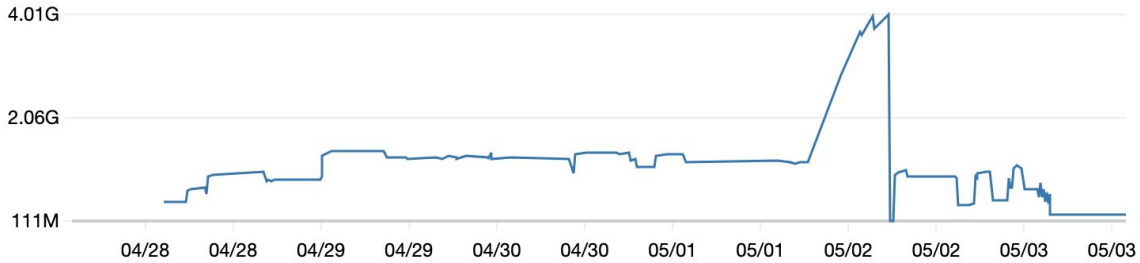
- Pagination doesn’t work well on large datasets
- No lazy loading
- Let’s reconsider it!
  - Fetch DB for users IDs
  - Fetch conversations list from Firebase (`msg_conversations/user-193`)
  - Iterate through that list and fetch conversation messages

ISSUES & TAKE AWAYS

- Memory usage - Heroku isn't very generous (512MB/25$, 1GB/50$)
- Handle worker exits & implement re-spawns
- Implement retries or dead-letter queues

AWS

- **t2.medium 4GB / ~33$**
- Optional VPC endpoint for S3
- *pm2* - stores logs by **default(!)**



ISSUES & TAKE AWAYS

- Use streams

hohy on 17 Mar

It would be nicer and maybe a bit faster to transfer the files using node streams (just pipe read stream from request to upload stream to S3) without storing whole file in memory... But this works too... ;)

- Limit the number of promises created at once

robertrossmann on 18 Mar

This will blow up in a matter of seconds. You need to limit the number of photos being migrated to a relatively small number, like, 10 or 20 at most. Otherwise it will create thousands of pending promises and open up thousands of network requests to start the image download and you will soon run out of memory. 😱

PROCESS LOTS OF PROMISES EFFECTIVELY

```
import chunk from 'lodash.chunk'

export const processInChunks = async (array, handlerFn, { chunkSize = 5 } = {}) => {
  const result = []
  for (const dataChunk of chunk(array, chunkSize)) {
    result.push(...await Promise.all(dataChunk.map(handlerFn)))
  }
  return result
}
```

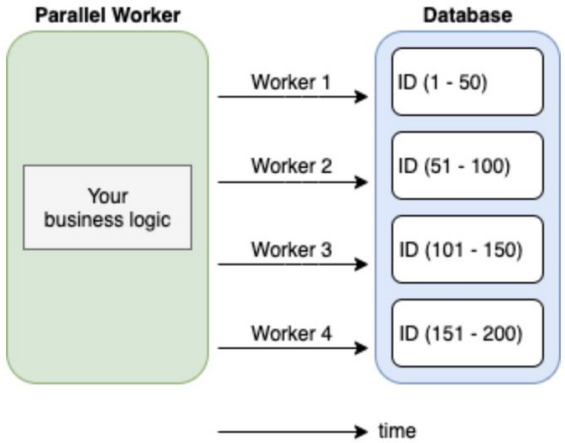
https://gist.github.com/jozefcipa

CHAT TAKE AWAYS

- Keep data that is subject to change easily accessible by either:
  - Keeping DB table as an “index” (store all files)
  - Keeping only IDs in chat (probably not very efficient)
  - Using custom domain URLs so you stay in control (custom CDN domain)
- Use CDN urls (if possible)
- Delete files also from storage, not only from DB

@SURGEAPP/PARALLEL-WORKER

- npm package
- Spawns and manages workers
- Orchestrates workers access to a shared resource
- Features
  - Restart worker after crashing
  - Pick up unprocessed payload
  - Local / distributed lock option



https://github.com/surgeapp/parallel-worker

Fetch data from DB

```
const parallelWorker = new ParallelWorker({
  redis,
})

parallelWorker.setFetchNext(async (lastId: ID | null) => {
  const result = await db('users')
    .where('updated', '=', 0)
    .andWhere('id', '>', lastId ?? 0)
    .orderBy('id')
    .limit(5)

  if (result.length === 0) {
    return null
  }

  return {
    lastId: result[result.length - 1].id,
    idsRange: result.map((row: any) => row.id),
  }
})
```

Process data

```
parallelWorker.setHandler(async ({ idsRange }: Payload) => {
  await db('users')
    .whereIn('id', idsRange)
    .increment('updated', 1)
})
```

THANK YOU!

Jozef Cipa / jozef.cipa@strv.com

STRV