How we migrated our JSON DB to a Relational DB using Apache Beam / Dataflow

Lakshmanan Arumugam

Senior Software Engineer @ Recursion

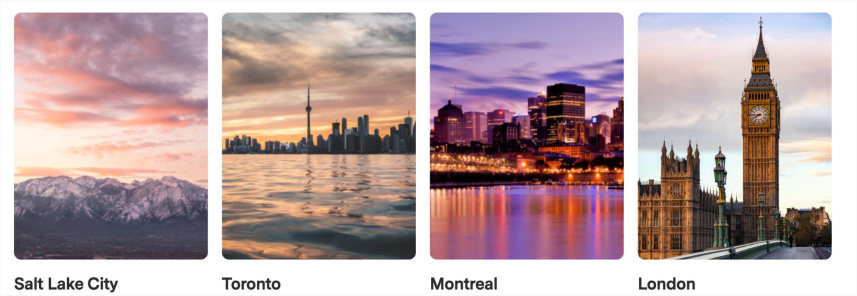
BEAM SUMMIT

September 4-5, 2024

Sunnyvale, CA. USA

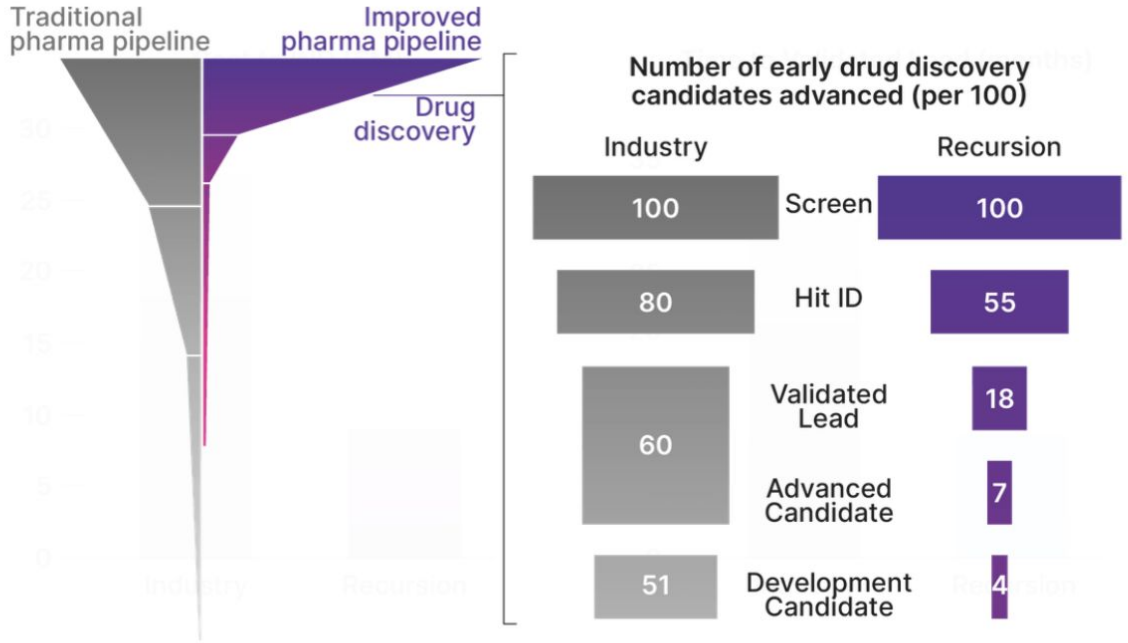
Recursion - overview

- TechBio company
  - leveraging tech to industrialize drug discovery
- Petabytes of data
  - Our robots run millions of unbiased experiments and generate data
- Offices in North America and Europe



BEAM SUMMIT

Recursion - contd.,



Beam / Dataflow @ Recursion

- Image processing (ex: TIFFs -> PNGs)
- Dataset packaging & publishing
  - Group different sets of images based on metadata, process and upload
- Processing our ML artifacts (ex: embeddings)
  - Aggregation

BEAM SUMMIT

How many of you have performed some sort of migration in the past? (DB/API/Service/Tech)

BEAM SUMMIT

Context - DB migration

- We were using *Cloud Datastore* as our DB
- Why the migration?
  - Moving away from deprecated datastore (now: firestore)
  - Decisions made to choose datastore years ago are not valid anymore
    - Ex: we are not using all the columns for searching but datastore indexes all the columns (~10 TB indexes, $$$$)
  - Postgres Relational DB (well-established “boring” way in tech)
  - We were also migrating the service that was using this DB

BEAM SUMMIT

Migration approach - contd.,

1. Migrate data until a certain timestamp (ex: ***timestamp_1*** *or now*)
    a. To an “offline” postgres instance
2. Start writing (from ***timestamp_2***) new data to both postgres and datastore
3. Now migrate the **delta** from ***timestamp_1*** to ***timestamp_2***
4. Switch the services to read data from postgres (instead of datastore)

BEAM SUMMIT

Migration approach - visualized

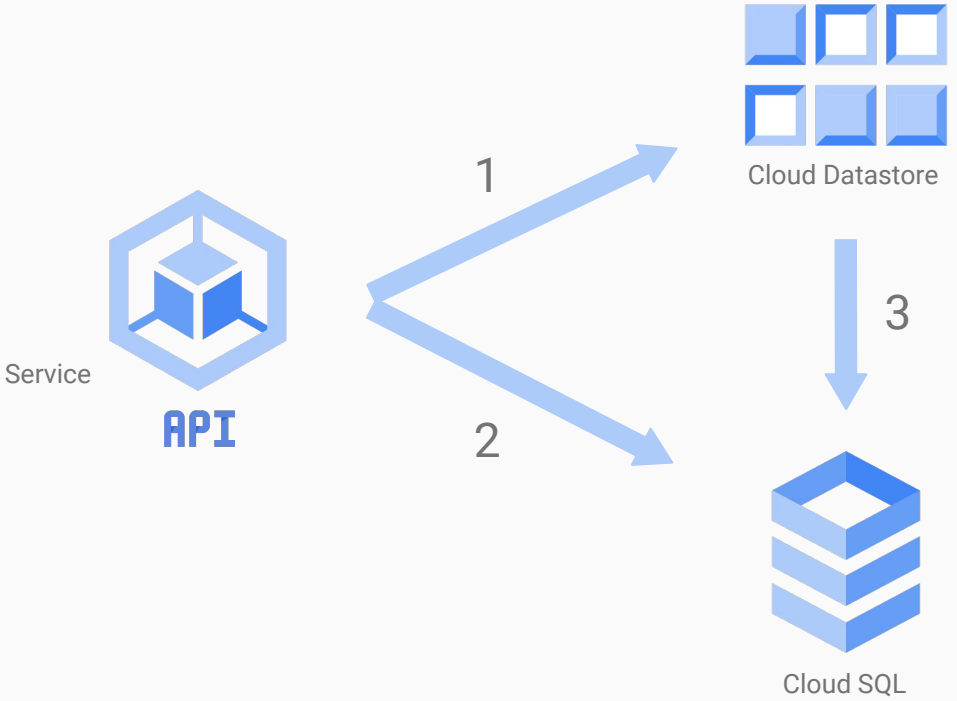


Image credits

Parts of the migration

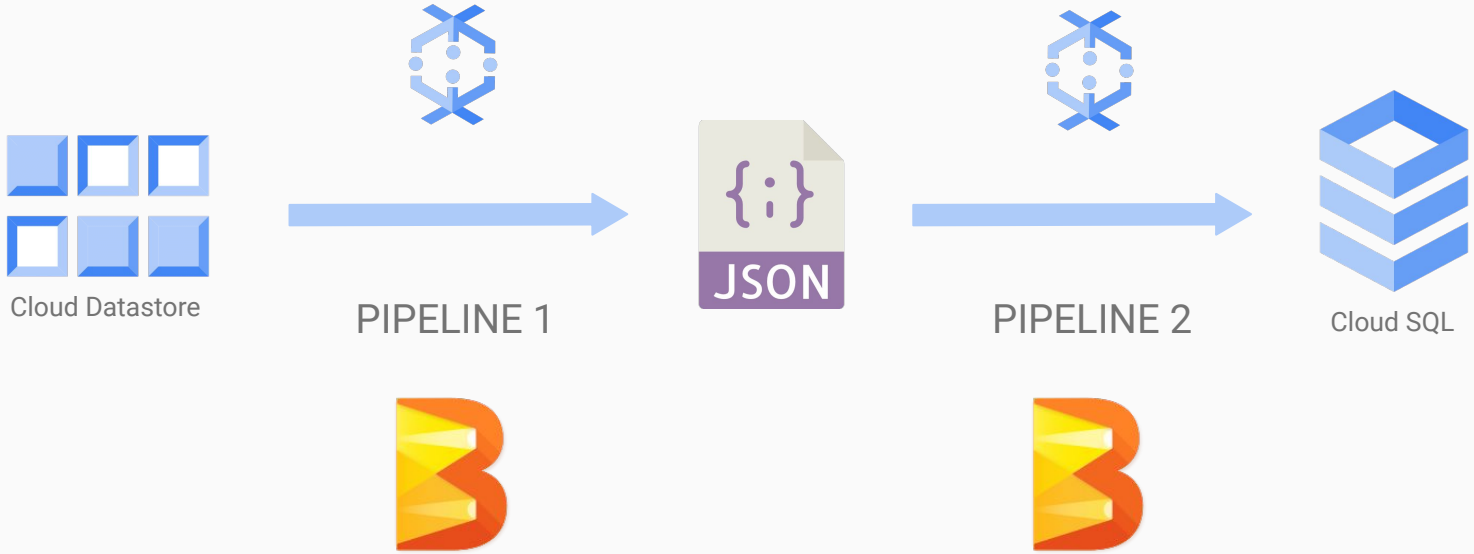


Image credits

Image credits

Part 1 of the migration

Pipeline 1: Export existing data (as files)

BEAM SUMMIT

Export existing data (*until timestamp_1*)

- Off the shelf pipeline (to export as files)
  - Export from Datastore
  - Pipeline 1: firestore-to-cloud-storage template
  - Code: DatastoreToText.java
- If we read from live DB, we are putting extra load on prod
- Dataflow template,
  - Inequality filters will not export data in parallel
  - But you can use JS filters (via the UDF) to filter data

BEAM SUMMIT

Pipeline options

appName	DatastoreToText
filesToStage	[/export/hda3/borglet/remote_hdd_fs_dirs/0.rapid.runner-kkiazvmw-euuj-rpb6-fqoq-iw3piuo2yuu4.dataflow-releaser.2335046332647.14b334fb3717c109/mou ... SEE ALL
firestoreReadGqlQuery	SELECT * FROM address where [redacted] >= 1690269709000
firestoreReadProjectId	[redacted]
gcpTempLocation	gs://[redacted]/tmp
jobName	delta-export-from-1690269709000
labels	{goog-dataflow-provided-template-name=firestore_to_gcs_text, goog-dataflow-provided-template-version=2023-09-12-00_rc00, goog-dataflow-provided-templa
maxNumWorkers	1,000
pipelineUrl	gs://dataflow-templates-libraries/2023-09-12-00_RC00/pipeline-cfyJcRLP7eN3hgezM68VZW6JltJUFKxG47Ju4KvpAKY.pb
project	[redacted]
region	us-east1
runner	org.apache.beam.runners.dataflow.DataflowRunner
sdkContainerImage	–
serviceAccountEmail	[redacted]
stagingLocation	gs://dataflow-templates-libraries/2023-09-12-00_RC00
subnetwork	https://www.googleapis.com/compute/[redacted] View details
templateLocation	gs://[redacted]
tempLocation	gs://[redacted]
textWritePrefix	gs://[redacted]
userAgent	Apache_Beam_SDK_for_Java/2.50.0(JRE_11_environment)

Equivalent REST

Logs HIDE

Click to clear the selected step

STEP LOGS DATA SAMPLING

Worker logs for step "DatastoreConverters.ReadJsonEntities"

Severity: Default

Filter Search all fields and values

SEVERITY	TIMESTAMP	SUMMARY
> i	2023-09-26 13:47:25.504 EDT	User gql query translated to Query(kind { name: "address" } filter { property_filter { property { n…
> i	2023-09-26 13:47:25.622 EDT	Latest stats timestamp for kind address is 1695626593000000
> i	2023-09-26 13:47:25.688 EDT	Estimated size bytes for the query is: [redacted]
> i	2023-09-26 13:47:25.690 EDT	Splitting the query into 19008 splits
v !		Unable to parallelize the given query: kind { name: "address" } filter { property_filter { property { name: "[redacted]created-at" } op: GREATER_THAN_OR_EQUAL value { integer_value: 1690269709000 } } }

Open in Logs Explorer


```
{
  insertId: "5345541829037041209:164762:0:11778"
  jsonPayload: {10}
  labels: {7}
  logName: "projects/[redacted]logs/dataflow.googleapis.com%2Fworker"
  receiveTimestamp: "2023-09-26T17:47:35.965549715Z"
  resource: {2}
  severity: "WARNING"
  timestamp: "2023-09-26T17:47:25.693Z"
}
```

Filtering Data during export

Users > Laksh.Arumugam > Desktop > Beam : Dataflow Migration > JS filterData.js > ...

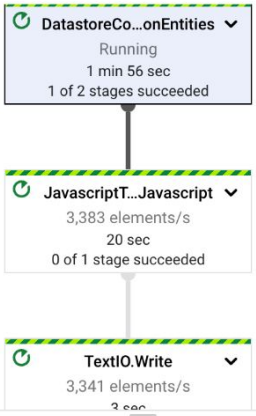
```
/**
 * A transform function which only accepts objects that are created before: Oct 20, 2023 09:00:00 EST.
 * @param {string} inJson
 * @return {string} outJson
 */
function transform(inJson) {
  var obj = JSON.parse(inJson);
  // only output objects which are created before: Oct 20, 2023 09:00:00 EST
  if (obj.hasOwnProperty('properties')) {
    if (obj['properties']['created-at'] === undefined) {
      return JSON.stringify(obj);
    }
    created_at = parseInt(obj['properties']['created-at']['integerValue']);
    if (created_at <= 1697806800000) {
      return JSON.stringify(obj);
    }
  }
}✨
```

datastore-arc... CLONE STOP IMPORT AS PIPELINE SHARE TRIGGER SEND FEEDBACK

JOB GRAPH | EXECUTION DETAILS | JOB METRICS | COST | RECOMMENDATIONS

Job steps view: Graph view

CLEAR SELECTION



Screen Shot 2023-08-23 at 5.06.12 PM

Logs HIDE

STEP LOGS | DATA SAMPLING

Worker logs for step "DatastoreConverters.ReadJsonEntities" Severity: Default Filter Search all fields and values

SEVERITY	TIMESTAMP	SUMMARY
i	2023-11-03 15:06:47.017 EDT	User query: 'SELECT * FROM address'
i	2023-11-03 15:06:48.243 EDT	User gql query translated to Query(kind { name: "address" })
i	2023-11-03 15:06:48.476 EDT	Latest stats timestamp for kind address is 1698909706000000
i	2023-11-03 15:06:48.590 EDT	Estimated size bytes for the query is: [redacted]
i	2023-11-03 15:06:48.594 EDT	Splitting the query into 19809 splits

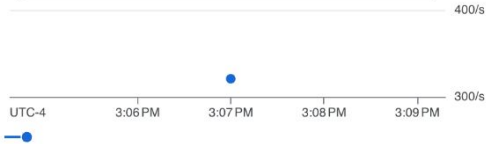
Loading... Scanned 16.3 KB.

Step info

Step name	DatastoreConverters.ReadJsonEntities
Wall time	1 min 56 sec

Output collections

Chart: Throughput (elements/sec)



DatastoreConverters.ReadJsonEntities/EntityToJson/ParMultiDo(EntityTo.
:
321.33/s

DatastoreConverters.ReadJsonEntities/EntityToJson/ParMultiDo(EntityToJson).out0

Elements added	247,089
Estimated size	268.4 MB

Optimized stages

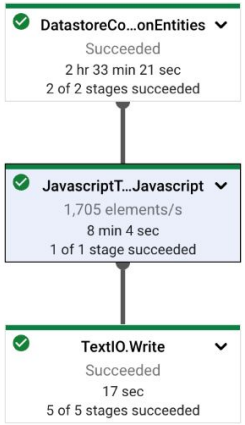
Stage name	Progress
F49	Running - 0%
F50	Succeeded

dataflow-expo... CLONE STOP SEND FEEDBACK

JOB GRAPH | EXECUTION DETAILS | JOB METRICS | COST | RECOMMENDATION

Job steps view: Graph view

CLEAR SELECTION



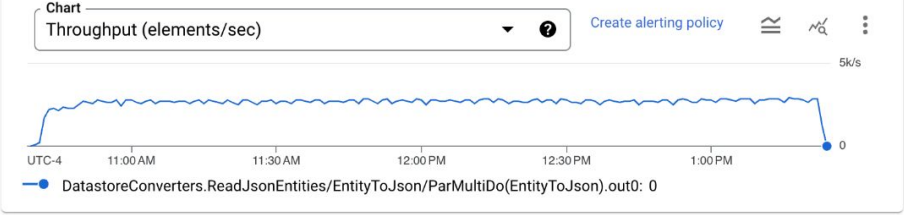
Logs SHOW

Click to clear the selected step

Step info

Step name	JavascriptTextTransformer.TransformTextViaJavascript
Wall time	8 min 4 sec

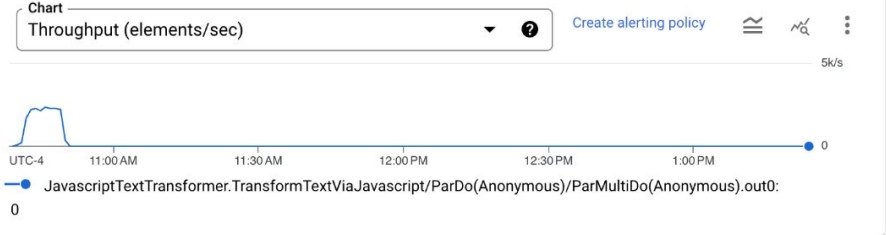
Input collections



DatastoreConverters.ReadJsonEntities/EntityToJson/ParMultiDo(EntityToJson).out0

Elements added	25,772,647
Estimated size	27.53 GB

Output collections



JavascriptTextTransformer.TransformTextViaJavascript/ParDo(Anonymous)/ParMultiDo(Anonymous).out0

Elements added	1,066,366
Estimated size	1.14 GB

Optimized stages

Stage name	Progress
F49	Succeeded

Exported files

UPLOAD FILES UPLOAD FOLDER CREATE FOLDER TRANSFER DA

Filter by name prefix only | Filter | Filter objects and folders

Name	Size	Type
-04200-of-04443.json	293.8 MB	text/plain
-04201-of-04443.json	125.9 MB	text/plain
-04202-of-04443.json	124 MB	text/plain
-04203-of-04443.json	74.7 MB	text/plain
-04204-of-04443.json	722.7 MB	text/plain
-04205-of-04443.json	1.8 GB	text/plain
-04206-of-04443.json	207.1 MB	text/plain
-04207-of-04443.json	921.5 MB	text/plain
-04208-of-04443.json	93.9 MB	text/plain
-04209-of-04443.json	87.1 MB	text/plain
-04210-of-04443.json	117.6 MB	text/plain
-04211-of-04443.json	318.7 MB	text/plain
-04212-of-04443.json	113.6 MB	text/plain
-04213-of-04443.json	102.9 MB	text/plain
-04214-of-04443.json	1.1 GB	text/plain
-04215-of-04443.json	1.8 GB	text/plain
-04216-of-04443.json	1.2 GB	text/plain
-04217-of-04443.json	106.4 MB	text/plain
-04218-of-04443.json	1.8 GB	text/plain

Filter by name prefix only | Filter | Filter objects and folders

Name	Size	Type
-03700-of-04443.json	247.9 MB	text/plain
-03701-of-04443.json	80.8 MB	text/plain
-03702-of-04443.json	127.6 MB	text/plain
-03703-of-04443.json	101 MB	text/plain
-03704-of-04443.json	106.5 MB	text/plain
-03705-of-04443.json	419.3 MB	text/plain
-03706-of-04443.json	106.1 MB	text/plain
-03707-of-04443.json	947.3 MB	text/plain
-03708-of-04443.json	996.7 MB	text/plain
-03709-of-04443.json	114.4 MB	text/plain
-03710-of-04443.json	113.5 MB	text/plain
-03711-of-04443.json	77 MB	text/plain
-03712-of-04443.json	1.6 GB	text/plain
-03713-of-04443.json	131.4 MB	text/plain
-03714-of-04443.json	841 MB	text/plain
-03715-of-04443.json	1.3 GB	text/plain
-03716-of-04443.json	247.1 MB	text/plain

BEAM SUMMIT

Part 2 of the migration

Pipeline 2: Normalize the exported json data to relational database

BEAM SUMMIT

Normalize JSON to Postgres schema

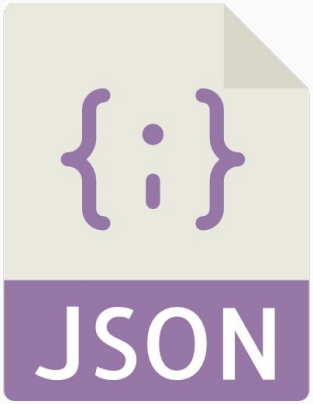


Image credit

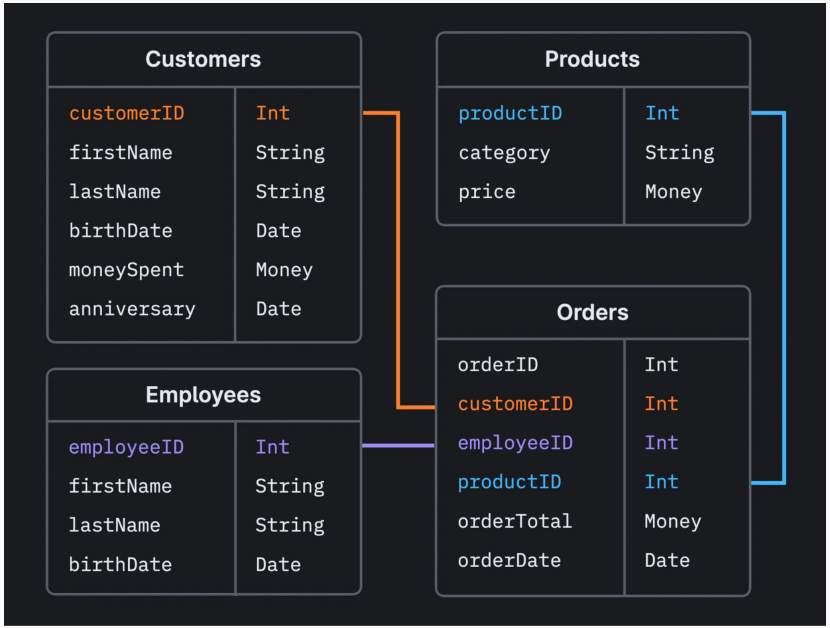


Image credit

BEAM SUMMIT

Pipeline

- ReadFromText(*bucket*)
- BatchingFn(*batch_size*)
- WriteJSONToPostgres()
  - Output errors to files

```
postgres_pipeline = beam.Pipeline(options=PipelineOptions(pipeline_args))

json_lines_pcollection = (
    postgres_pipeline
    | "Read lines from *.json files"
    >> beam.io.ReadFromText(json_files_bucket_pattern)
)
batched_json_lines = (
    json_lines_pcollection
    | "Group records to batches" >> beam.ParDo(BatchingFn(batch_size))
)
write_postgres_results = batched_json_lines | "Write to postgres" >> beam.ParDo(
    WriteJSONToPostgres(), postgres_url
).with_outputs(
    "write_postgres_success", "write_postgres_error", "write_postgres_skipped"
)
# write postgres errors to files
errors = (
    write_postgres_results.write_postgres_error
    | "flatten failed records" >> beam.FlatMap(lambda elements: elements)
    | "write to failed bucket"
    >> beam.io.WriteToText(postgres_error_bucket, file_name_suffix=".json")
)

return postgres_pipeline
```

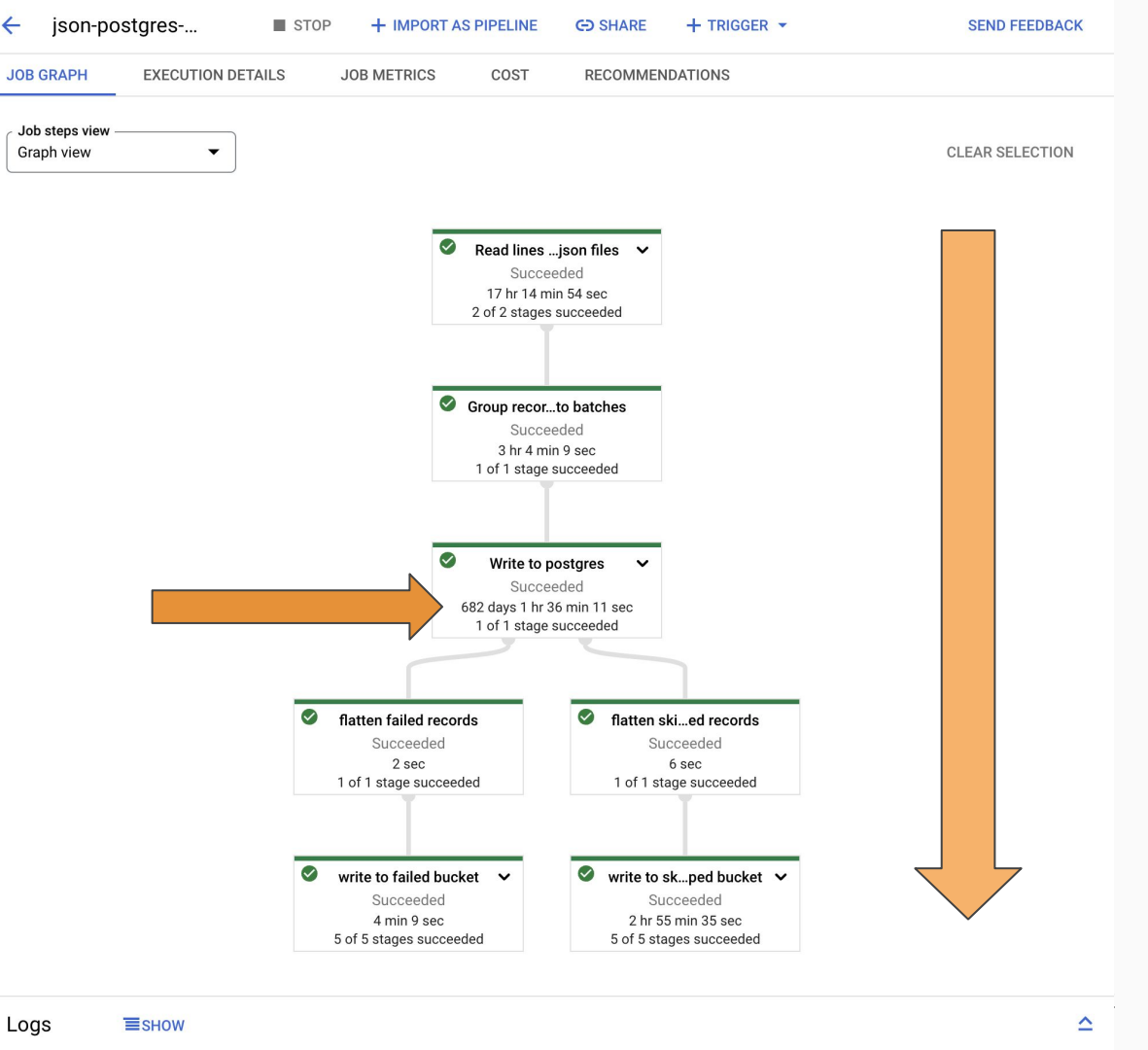
BatchingFn

- start_bundle
- process
- finish_bundle

```python
class BatchingFn(beam.DoFn):
    def __init__(self, batch_size=1000):
        self._batch_size = batch_size
        self.window = beam.transforms.window.GlobalWindow()

    def start_bundle(self):
        # buffer for string of lines
        self._lines = []

    def process(self, element, window=beam.DoFn.WindowParam):
        self.window = window

        # Input element is a string (representing a JSON line)
        self._lines.append(element)
        if len(self._lines) >= self._batch_size:
            # print(f"flushing batched_json_lines... {len(self._lines)}")
            yield self._lines
            self._lines = []

    def finish_bundle(self):
        # takes care of the unflushed buffer before finishing
        if self._lines:
            # print(f"flushing last batch... {len(self._lines)}")
            yield beam.utils.windowed_value.WindowedValue(
                value=self._lines,
                timestamp=0,
                windows=[self.window],
            )
            self._lines = []
```

Steps

- **Read from files** line by line
- **Group** the lines (records) into batches
- **Normalize** and write to postgres
- If success, proceed
- If failed, **flatten** the entire batch and write as .jsonl files again



Job metrics

Job info

Job name	json-postgres-all-20k-batched-no-constraint
Job ID	2023-08-16_09_22_35-6925631302841689133
Job type	Batch
Job status	Succeeded
SDK version	Apache Beam Python 3.9 SDK 2.42.0
Job region	us-east1
Worker location	us-east1
Current workers	0
Latest worker status	Worker pool stopped.
Start time	August 16, 2023 at 12:22:36 PM GMT-4
Elapsed time	16 hr 46 min
Encryption type	Google-managed
Dataflow Prime	Disabled
Runner v2	Enabled
Dataflow Shuffle	Enabled

Resource metrics

Current vCPUs	1,000
Total vCPU time	16,608.98 vCPU hr
Current memory	3.66 TB
Total memory time	62,283.675 GB hr
Current HDD PD	24.41 TB
Total HDD PD time	415,224.497 GB hr
Current SSD PD	0 B
Total SSD PD time	0 GB hr
Total Shuffle data processed	2.06 MB
Billable Shuffle data processed	526.78 KB

BEAM SUMMIT

Cost

- ~ 1175$
- vCPU $$$$$$$$

JOB GRAPH | EXECUTION DETAILS | JOB METRICS | COST | RECOMMENDATIONS

The Cost tab shows the estimated cost of your current Dataflow job. Estimated costs are calculated by multiplying your resource usage (as shown in Cloud Monitoring) by the list price of those resources in the job region. The estimated cost might not reflect your actual job cost for a variety of reasons, such as contractual discounts, temporary billing adjustments, and so on. You can also view the Cloud Billing reports for your Cloud Billing account in the console.

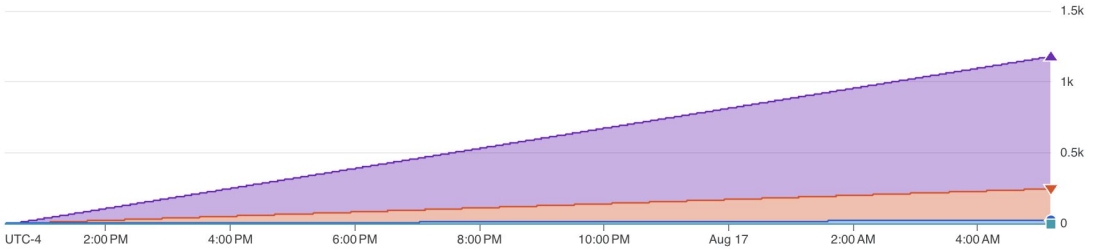
Estimated Cost

CREATE ALERT Auto Refresh MAX TIME

OPTIONS

Total cost	Cost for last week	Cost for last 24h	Cost for last hour	Cost for selected range
$1,174.07	$0.00	$0.00	$0.00	$1,174.07

Job Cost Estimation



	Cost	Adjustments	Net cost
vCPU	$930.10	$0.00	$930.10
Memory	$221.54	$0.00	$221.54
Processed data	$0.00	$0.00	$0.00
HDD	$22.42	$0.00	$22.42
SSD	$0.00	$0.00	$0.00
Total	**$1,174.07**	**$0.00**	**$1,174.07**

Logs SHOW

Hypothetical cost - batching script

- Simple batching script + concurrency
  - Track progress of migration in a table (lil bit more dev effort = more time = more $$$$$)
- Rent VMs in the cloud
- 1000 vCPUs * 0.03465 * 20 hours = ~ 700$ + mem costs + extra dev time
  - Pricing: General purpose VM

BEAM SUMMIT

Takeaways - 1

- Batched commits were very important
  - Extremely slow otherwise
  - Without batching: 1000 commits/sec (1 commit per worker)
  - 3.6M records per hour => 472 hours for 1.7B records (~19.5 days)
- You can upgrade/downgrade the postgres machine in the cloud
  - useful for pre-/post- migration

BEAM SUMMIT

Takeaways - 2

- Cross platform powers of Beam (Pipeline 1 is Java, Pipeline 2 is Python)
  - Leveraging pipelines built by the community
- Error handling, repeatability
  - Basically re-executed the same pipeline with failed batches as input
  - Batch_size as 1
- Relatively no downtime
  - Offline postgres 🤦‍♂️

BEAM SUMMIT

Takeaways - 3

- Not a “boring” approach to migration
  - Throwing compute at a migration problem is not conventional
  - But: it worked for us and all we had to write was two functions
    - **BatchingFn**, **WriteToPostgres** in Beam
    - Dataflow did the rest

BEAM SUMMIT

Thank you!

Questions?

Feel free to reach out to me on,
Laksh47 - LinkedIn

https://linkedin.com/in/laksh47
Handle: **laksh47**
Lakshmanan Arumugam

BEAM SUMMIT