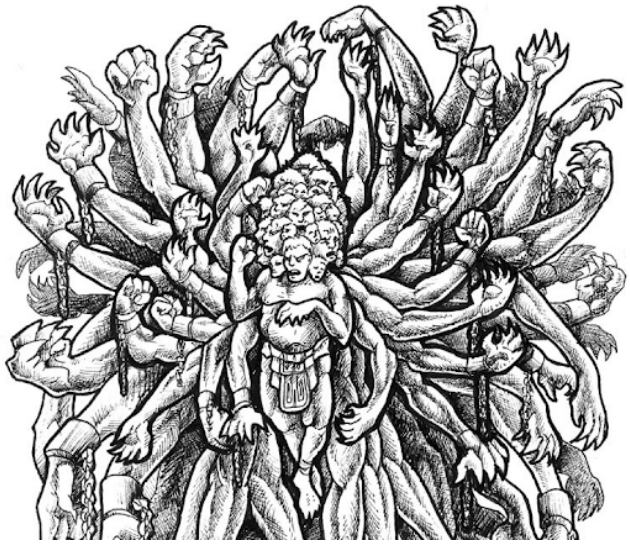
Centimani - User & Developer Guide



What's Centimani?

Centimani is a configurable massive file processor able to split text files in chunks, process them following a strategic pattern and store the results in BigQuery for reporting. It provides configurable options for chunk size, number of retries and takes care of exponential backoff to ensure all requests have enough retries to overcome potential temporary issues or errors.

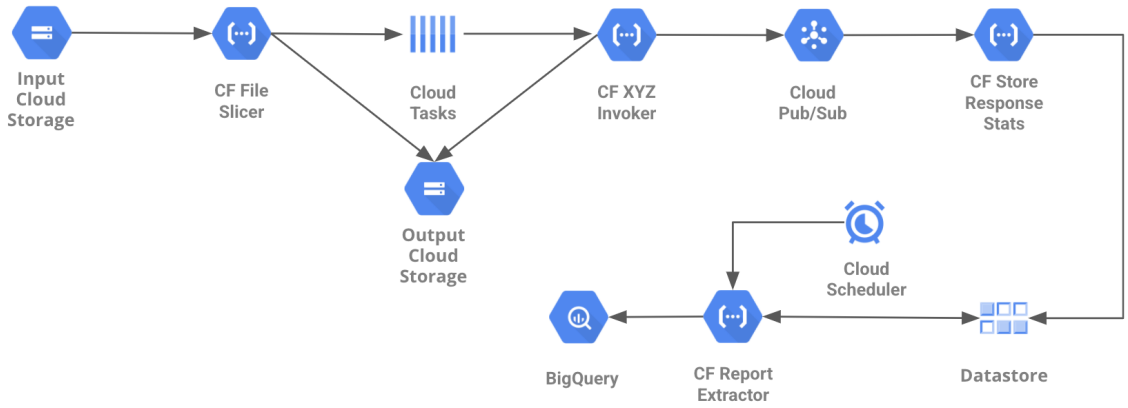
In its vanilla version, Centimani is equipped with a Google Ads offline conversion upload functionality able to handle at least 2.5 million conversion uploads in less than 1 hour, using Google Ads API v6.1.0

What’s inside?

Here’s the list of files included as part of the solution:

```
├── cfs
│   ├── file_slicer
│   │   ├── deploy.sh
│   │   ├── main.py
│   │   ├── requirements.txt
│   │   └── test_main.sh
│   ├── gads_invoker
│   │   ├── deploy.sh
│   │   ├── main copy.py
│   │   ├── main.py
│   │   ├── requirements.txt
│   │   └── test_main.sh
│   ├── reporting_data_extractor
│   │   ├── deploy.sh
│   │   ├── main.py
│   │   ├── requirements.txt
│   │   ├── test_main.sh
│   └── store_response_stats
│       ├── deploy.sh
│       ├── main.py
│       ├── requirements.txt
│       ├── test_main.sh
├── deploy
│   ├── config.yaml
│   ├── deploy.sh
│   ├── env.sh
│   ├── gads_config.json
│   ├── helpers.sh
│   └── utils
│       ├── delete_daily_tables.sh
│       └── delete_sharded_tables.sh
└── README.md
```

Component Diagram



GCS Structure

GCS Input Bucket
|
\----------> input (for file_slicer to read files from)

GCS Output Bucket
|
|----------> <date>/failed (for failed input files: name error, corrupted…)
|
|----------> <date>/processed (for files successfully sliced)
|
|----------> <date>/slices_processing (for slices to be picked by the invoker)
|
|----------> <date>/slices_processed (for slices successfully processed)
|
\----------> <date>/slices_failed (for slices with processing errors)

File Naming Convention

Generic Input File Naming Convention

<Platform uppercase>_<free text no underscore allowed>_<customer/account id>_<free text no underscore allowed>_<free text no underscore allowed>_<date>_<free text>.csv

Part description:

A. <Platform uppercase>: the name of the platform
B. <free text no underscore allowed>: use these fields to describe the purpose of the file or add any additional information. You may include customer name, conversion name, etc. Please do not use underscores, since they are used as separators.
C. <customer/account id>: identifier of the customer or account
D. <date> The date in YYYYMMDD format
E. <free text> any text

Examples:

```
GADS_MY-SOMETHING_000-000-0000_000-000-0000_000-000-0000_20210421_1.csv
CM_MY-SOMETHING_1234_1234_1234_20210421_FREE_TEXT.csv
```

Generic Slice File Naming Convention

The file slicer process will take the input file name and add 3 dashes (---) and the slice number as the suffix for the name for each generated slice:

<Platform uppercase>_<free text no underscore allowed>_<customer/account id>_<free text no underscore allowed>_<free text no underscore allowed>_<date>_<free text>---#

I.e:

```
GADS_MY-SOMETHING_000-000-0000_000-000-0000_000-000-0000_20210421_1.csv---3
```

GAds Invoker Input File Naming Convention

GADS_<free text no underscore allowed>_<CID where conversions are observed>_<MCC where developer token is defined>_<CID where conversions are defined>_<date string in YYYYMMDD format>_<free text>

Part definition:

A. <free text no underscore allowed>: use these fields to describe the purpose of the file or add any additional information. You may include customer name, conversion name, etc. Please do not use underscores, since they are used as separators.
B. <CID where conversions are observed>: the account where the cids are captured.
C. <MCC where developer token is defined>: the MCC account where we could find the developer token to be used.
D. <CID where conversions are defined>: the account where the conversion actions are defined.
E. <date string in YYYYMMDD format>: The date in YYYYMMDD format
F. <free text> any text

I.e:
GADS_MY-SOMETHING_000-000-0000_000-000-0000_000-000-0000_20210421_1.csv

GAds Invoker Slice File Naming Convention

Same as generic

Deploying the vanilla version

- Edit config.yaml inside the *deploy* directory and configure your Solution and Cloud Project settings, using the first part of the file (up to the 'DO NOT MODIFY' line).

  DEPLOYMENT_NAME & SOLUTION_PREFIX values will be used to build the name of every artifact, so no collision with other deployments may happen.

```
DEPLOYMENT_NAME: 'deployment-name' (i.e: production)
SOLUTION_PREFIX: 'descriptive-prefix' (i.e: conversion_uploader)
```

SERVICE_ACCOUNT: the service account to use. It will be created if it does not exist. Add it without domain if you're not sure of the domain name. It must be 6 to 30 char long.

```
SERVICE_ACCOUNT: 'service-account-name-without-domain' (i.e mcu-123)
```

DEFAULT_GCP_PROJECT & DEFAULT_GCP_REGION: are the name of your project and the corresponding region

```
DEFAULT_GCP_PROJECT: 'your-project' (i.e massive-conversion-uploader)
DEFAULT_GCP_REGION: 'your-project-region' (allowed values here)

DEFAULT_GCS_BUCKET: 'the-bucket-name'


SKIP_DATASET_CREATION: 'N'
BQ_REPORTING_DATASET: 'the-dataset-name'
BQ_REPORTING_TABLE: 'the-table-name'
BQ_GCP_BROAD_REGION: 'EU' (allowed values here)
TIMEZONE: 'Europe/Madrid' (allowed values here)

REPORTING_DATA_POLLING_CONFIG: '\*/5 \* \* \* \*' (format explained here)
```

- Add the Google Ads credentials gads_config.json file, inside the *deploy* directory

    - **Replace XXXXX** with the customer id you want to login as. This value is used for looking up the conversion action id later in this file
    - **credentials.XXXXX.developer_token:** the GAds developer token
    - **credentials.XXXXX.client_id:** the OAUTH2 client id
    - **credentials.XXXXX.client_secret:** the OAUTH2 client secret
    - **credentials.XXXXX.refresh_token:** the OAUTH2 client refresh token
    - **credentials.XXXXX.login_customer_id:** the id of the customer to login as. It has to be the same value as XXXXX

- Edit (if required) the performance variables gads_config.json file, inside the *deploy* directory

- **slicer.max_chunk_lines**: the number of lines on each file chunk excluding the header (default 2000)

- **queue_config.name**: the name of the queue for the gads invoker (default gads-conversions-upload)

- **queue_config.max_dispatches_per_second**: number of request to gads invoker per second (default 100)

- **queue_config.max_concurrent_dispatches**: max number of tasks in parallel (default 2000)

- **queue_config.retry_config.max_attempts**: max retry attempts (default 5)

- **queue_config.retry_config.max_retry_duration**: max retry duration (default 1750)

- **queue_config.retry_config.min_backoff**: lower bound in seconds for exponential backoff (default 10)

- **queue_config.retry_config.max_backoff**: upper bound in seconds for exponential backoff (default 300)

- **queue_config.retry_config.max_doublings**: how many times the time between retries will be doubled (default 3)

When configuring the settings, please take these tips into consideration:

- For Google Ads conversions uploading, take into account that the maximum number of conversions in each upload is currently 2,000 but the timeout is currently set to 60 seconds, so we recommend to send a lower number of conversions in each request (around 50), until this discrepancy is fixed.

- Service account names (the part before the @) must be between 6 and 30 chars long. If it's a new service account, please don't include the domain. Otherwise, if you are reusing a service account formerly created, then you should include it.

- Input and output GCS buckets will be automatically created if they don't exist already. Please take into account that bucket names are unique across Google, so you may choose a name already in use in another project. If this is the case, you will need to choose a different name for the deployment to succeed.

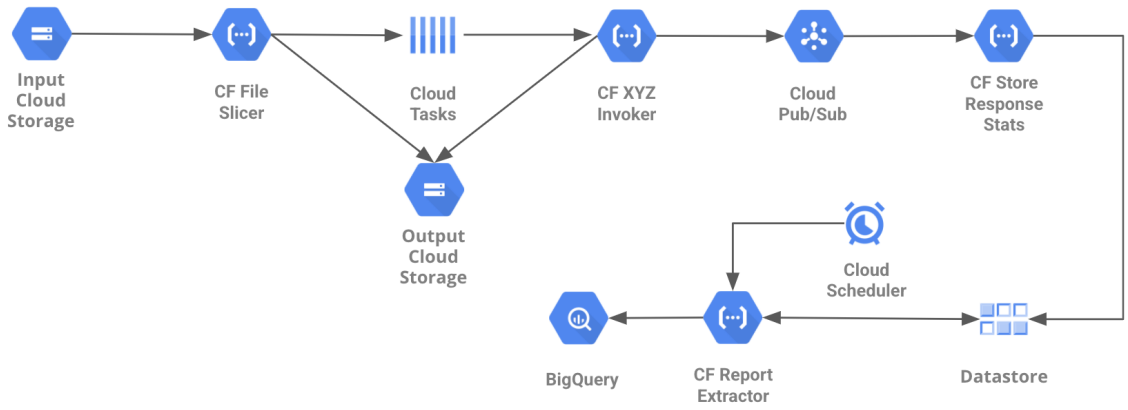
- Cloud Function names are limited to 63 chars max. Please do not choose a long deployment name and solution prefix, or the deployment will fail.

- During the deployment of the schedulers, you may be asked to deploy an App Engine. If that's the case, just say "Y"es and choose the same location that you used in the configuration file. This App Engine will be used to provide the Cloud Task Scheduler functionality.

- Delete the file *_config.json after completing the deployment, in order to avoid making sensitive information available in your deployment system.

How to use it Centimani

Centimani uses Cloud Tasks to parallelize up to thousands of API calls with small payloads, thus reducing the processing time for big input files. It uses Secret Manager to store all sensitive credentials and Datastore to keep an updated report on the progress of the data processing workflow. Finally, reporting status information is sent to BigQuery, so it can be analyzed and/or visualized.



In order to use this system, the first step is to upload one or more "big files" into the input Google Cloud Storage Bucket. Automatically, the File Slicer Cloud Function will be triggered and split each of these big files into smaller chunks (with the configured number of lines). Then, a task will be queued in Cloud Tasks to send the location of each chunk to the invoker component.

The Cloud Tasks will use HTTP to trigger multiple instances of the Invoker Cloud Function in parallel. Each of these triggers will come with the location of a chunk, that will be pre-processed, if required, and then passed as part of the invocation of the target API. At this point, details about responses and errors returned by the API will be sent to the "Store Response Stats" Cloud Function using Pub/Sub, which will store these information in Datastore.

Finally, it is possible to (optionally) set up Google Task Scheduler to periodically trigger a Cloud Function called “Report Extractor”, that will flatten and extract status and error information from Datastore to BigQuery, so it can be properly analyzed or used to provide visual status information about the data processing workflow by using, for example, a dashboard.

GAds file naming convention

The provided Invoker module can upload offline click conversions in Google Ads, however it is possible to add any other functionality as described in “how to extend the solution?”

Input files for Google Ads click conversion uploader must follow the GAds Invoker Input File Naming Convention

```
GADS_<FREE-TEXT-WITH-NO-UNDERSCORES>_<FILE-CID>_<MCC-HOLDING-DEVELOPER-TOKEN>
_<CID-HOLDING-DEFINED-CONVERSIONS>_<FILE-GENERATION-DATE>.csv
```

All these details are required because the gAds Invoker module needs to obtain the conversion action id from the conversion action name, and then upload the conversions at the right level.

Daily Progress & Historical Reports

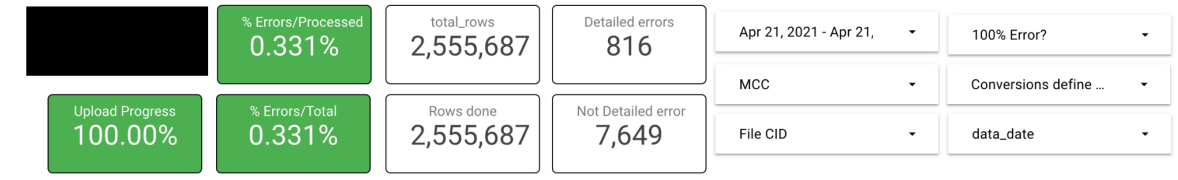
The following query can be used to get a status report of the upload process:

```
SELECT
        _TABLE_SUFFIX AS processing_date,
        parent_file_date AS data_date,
        parent_file_name as file,
        max(parent_total_files) as num_files,
        count(1) as processed_files,
        max(parent_total_rows) total_rows,
        sum(child_num_rows) processed_rows,
        sum(child_num_errors) num_errors
FROM `<project>.<dataset>.daily_results_*`
WHERE
  target_platform = '<PLATFORM>'
GROUP BY
   parent_file_name,
   parent_file_date,
```

A slightly modified version of this query can also be used to power a DataStudio dashboard (or any similar tool) to display the status of the upload process. In the case of DataStudio, you can use Date variables to filter by data range by defining a custom query like the one below as datasource, and enabling the date parameters in the configuration screen:

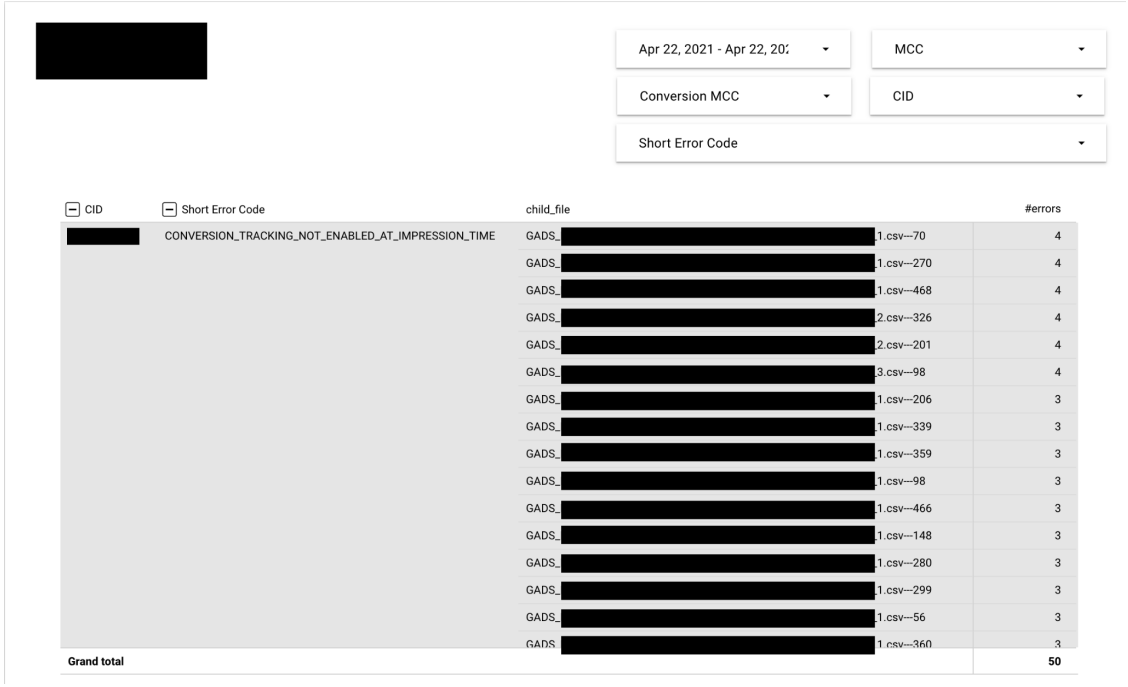
```
SELECT
  _TABLE_SUFFIX AS processing_date,
  parent_file_date AS data_date,
  parent_file_name as file,
  max(parent_total_files) as num_files,
  count(1) as processed_files,
  max(parent_total_rows) total_rows,
  sum(child_num_rows) processed_rows,
  SUM(child_num_errors) num_errors
FROM `<project>.<dataset>.daily_results_*`
WHERE
  target_platform = '<PLATFORM>' AND
  _TABLE_SUFFIX BETWEEN @DS_START_DATE AND @DS_END_DATE
GROUP BY
   parent_file_name,
   parent_file_date,
   _TABLE_SUFFIX
```

Build your own dashboard with Data Studio



Data Date	MCC	File CID	Parts	Parts Done	Rows	Rows Done	Errors	% File	% Error
20210421	[redacted]		153	153	7,649	7,649	7,649	100%	100%
			153	**153**	**7,649**	**7,649**	**7,649**	**100%**	**100%**
			5,038	5,038	251,885	251,885	0	100%	0%
			2,900	2,900	144,992	144,992	30	100%	0.02%
			2,516	2,516	125,775	125,775	62	100%	0.05%
			1,121	1,121	56,001	56,001	674	100%	1.2%
			105	105	5,242	5,242	0	100%	0%
			2	2	65	65	0	100%	0%
			11,682	**11,682**	**583,960**	**583,960**	**766**	**100%**	**0.13%**
			5,115	5,115	255,736	255,736	0	100%	0%
			4,086	4,086	204,251	204,251	0	100%	0%
			3,828	3,828	191,373	191,373	0	100%	0%
			2,754	2,754	137,653	137,653	0	100%	0%
Grand total			**51,130**	**51,130**	**2,555,687**	**2,555,687**	**8,465**	**100%**	**0.33%**

	Data Date	Processed Date	File with Errors	Parts	Parts Done	Rows	Rows Done	Errors	% File	% Error
1.	20210421	20210421	GADS_[redacted]3.csv	500	500	25,000	25,000	9	100%	0.04%
2.	20210421	20210421	GADS_[redacted]2.csv	500	500	25,000	25,000	1	100%	+0%
3.	20210421	20210421	GADS_[redacted]3.csv	500	500	25,000	25,000	16	100%	0.06%
4.	20210421	20210421	GADS_[redacted]4.csv	500	500	25,000	25,000	1	100%	+0%
5.	20210421	20210421	GADS_[redacted]5.csv	500	500	25,000	25,000	4	100%	0.02%



CID	Short Error Code	child_file	#errors
[redacted]	CONVERSION_TRACKING_NOT_ENABLED_AT_IMPRESSION_TIME	GADS_[redacted]1.csv---70	4
		GADS_[redacted]1.csv---270	4
		GADS_[redacted]1.csv---468	4
		GADS_[redacted]2.csv---326	4
		GADS_[redacted]2.csv---201	4
		GADS_[redacted]3.csv---98	4
		GADS_[redacted]1.csv---206	3
		GADS_[redacted]1.csv---339	3
		GADS_[redacted]1.csv---359	3
		GADS_[redacted]1.csv---98	3
		GADS_[redacted]1.csv---466	3
		GADS_[redacted]1.csv---148	3
		GADS_[redacted]1.csv---280	3
		GADS_[redacted]1.csv---299	3
		GADS_[redacted]1.csv---56	3
		GADS_[redacted]1.csv---360	3
Grand total			**50**

Housekeeping

Google Cloud Storage clean up

- A dated directory is created everyday, you may want to clean old ones up at a certain point in time.

Datastore clean up

- In case you need to delete datastore entities, you'll need to create your own script to query entities to be deleted and iterate through them.

BigQuery clean up

- In case you need to delete daily BigQuery sharded tables, you can use the script *delete_daily_tables.sh* included in the utils directory

Data Model

Datastore

Entities Hierarchy

processing_date < child_file

processing_date Data Model

- name/id: string in YYYYMMDD format

child_file Data Model

Key(processing_date)

```
{
date: string (format YYYYMMDD),
target_platform: string enum [“gads” | “cm” |...],
parent: {
        cid: string,
        file_name: string,
        file_path: string,
        file_date: string (YYYYMMDD),
        total_files: int,
        total_rows: int
},
child: {
        file_name: string,
        num_rows: int,
        num_errors: int,
        errors: [{
                code: string,
                message: string,
                count: int
                }]
        }
}
```

BigQuery Data Model

Table daily_report_YYYYMMDD:

-	parent_total_rows	INTEGER	REQUIRED
-	parent_total_files	INTEGER	REQUIRED
-	child_num_errors	INTEGER	REQUIRED
-	child_num_rows	INTEGER	REQUIRED
-	parent_file_path	STRING	REQUIRED
-	parent_file_name	STRING	REQUIRED
-	processing_date	STRING	REQUIRED
-	child_file_name	STRING	REQUIRED
-	target_platform	STRING	REQUIRED
-	last_processed_timestamp	TIMESTAMP	REQUIRED
-	parent_file_date	STRING	REQUIRED
-	cid	STRING	REQUIRED
-	child_errors	RECORD	REPEATED
-	child_errors.code	STRING	REQUIRED
-	child_errors.count	STRING	REQUIRED
-	child_errors.message	STRING	REQUIRED

How to extend the solution?

Think of this solution as a massive generic file processor: it can split big files in chunks and process those chunks in parallel (for upload conversion or any other purpose). The general idea is for the file_slicer cloud function to read the platform name from the first token of the file name, get the configuration for that particular platform and create the file chunks for the processor.

Therefore you will need to:

- Decide on the new name of the platform, from now on: *<platform name>*
- Duplicate gads_invoker directory:
    - Rename it to ***<platform name in lowercase>_invoker*** (in concordance with *deploy/config.yaml*)
    - Change the following variable in ***<platform name in lowercase>_invoker/deploy.sh***:

        CF_NAME=$CF_NAME_<platform name in uppercase>_INVOKER

- Change ***<platform name in lowercase>_invoker/main.py*** and include the desired functionality.

    - Make sure you:
        - Reply with “200 OK” when the process is completed correctly or failed with a non retriable error (wrong data or max retry attempts reached)
        - Reply with “Error 500” when a retriable error happened.
        - Move the file to the right GCS directory after processing it (slices_failed or processed)
        - Send the outcome of the file processing to the ‘store_response_stats’ cloud function (look for _add_errors_to_input_data and _send_pubsub_message functions in main.py)

- Duplicate ***gads_config.json***

    - Rename it to *<platform name in lowercase>_config.json*

    - The file needs a minimum of 2 sections:

        - slicer: containing the configuration for the file_slicer
        - queue_config: containing the parameters for the Cloud Task queue which will manage the requests

    - Add any section you would need (Credentials or any other)

    - The file will be automatically uploaded to a secret in GCP by the deployment script.

  Sample file for GAds:

```
{
 "slicer" : {
    "max_chunk_lines": 2000
 },
 "queue_config": {
   "name": "gads-conversions-upload",
   "rate_limits": {
     "max_dispatches_per_second": 100,
     "max_concurrent_dispatches": 2000
   },
   "retry_config": {
```

```
    "max_attempts": 5,
    "max_retry_duration": 1750,
    "min_backoff": 10,
    "max_backoff": 300,
    "max_doublings": 3
   }
  },
  "credentials": {
   "XXXXXX": {
    "developer_token": "",
    "client_id": "",
    "client_secret": "",
    "refresh_token": "",
    "login_customer_id": "XXXXXX"
   },
   "YYYYYY": {
    "developer_token": "",
    "client_id": "",
    "client_secret": "",
    "refresh_token": "",
    "login_customer_id": "YYYYYY"
   }
  }
 }
```

Frequently Asked Questions

- Why are there separate input and output buckets?
    - GCS sends finalize events to file_slicer for every object created. This means that using a single bucket would trigger the slicer every time a file slice is written by the slicer or when a file is moved to another directory, generating hundreds of thousands of unnecessary (and charged beyond the free quota) calls. Separating the logic into 2 buckets resolves the unnecessary calls.

- There are files in the ‘failed’ GCS directory...
    - Check for errors in the file name pattern

- The number of files I uploaded to GCS do not match with the ones in BigQuery
    - Check the scheduler and make sure the associated Cloud Function is running correctly
    - Check if there are memory exhaustion errors on reporting_data_extractor cloud function. If so, increase the allocated memory for the Cloud Function.
    - If you built your own invoker, make sure that all errors are handled correctly

- In BigQuery I find more slices and with a different num of rows than than configured.
    - Have you changed the slicing parameters and reprocessed the file on the same day? If so, it’s normal. Clean the entries for that file in BigQuery and reprocess it.

- File slicing phase takes too long.
    - Consider uploading smaller files, or creating bigger chunks. (The benchmark we have for GAds conversion upload is 100 input files with 25K records each, split into 50 record files, taking less than 1 hour end to end processing time)
    - Consider increasing the allocated memory for the Cloud Function to 2GB (you can do it in *cfs/file_slicer/deploy.sh*)

- The invoker does not seem to work properly.
    - Check the Cloud Function logs and make sure that you are not running out of memory (due to the size of the data in Datastore, for example) and the API invocations and returns are working properly.

- I can’t see all of the information from Datastore in BigQuery.
    - Make sure that the “Report Extractor” Cloud Function is not running out of memory. Otherwise, increase the allocated memory to prevent this situation.