

What is google cloud dataflow?

1. What is data flow?

- **Unified stream and batch data processing service**

Google Cloud Dataflow is a managed service for building data processing pipelines. It allows you to write code that defines how data should be processed, and then Dataflow takes care of running that code on a scalable and reliable infrastructure.

Here's a practical example of how Dataflow can be used:

Use Case: Real-time analytics for website traffic

Imagine you run an e-commerce website and you want to analyze website traffic in real-time. You might collect data about user clicks, page views, and purchases. This data can be streamed into Google Cloud Pub/Sub, a real-time messaging service.

Dataflow can then be used to create a pipeline that processes this data stream. The pipeline could:

1. **Extract** data from Pub/Sub.
2. **Transform** the data by filtering out unwanted information and formatting it for analysis.
3. **Load** the transformed data into BigQuery, a data warehouse service for large datasets.

Once the data is in BigQuery, you can use business intelligence (BI) tools to create dashboards that show real-time insights into your website traffic. This can help you understand what products are popular, how users are navigating your site, and identify any potential issues.

2. It is Serverless too
3. ANd it is based on

- **Based on Apache Beam open source programming model**
- **Support both Batch and Stream processing**

beam is nothing but batch and stream

4. Next we have templates

In Google Cloud Dataflow, templates are pre-built or user-defined pipelines that offer a convenient way to run data processing jobs without writing code from scratch. They act as blueprints for data processing tasks.

There are two main types of Dataflow templates:

- **Flex templates:** These are the recommended approach for new templates. They package the pipeline code as a Docker image and utilize a separate template specification file. This allows for greater flexibility and isolation of dependencies.
- **Classic templates:** These are the older type of template and contain the entire pipeline code within the template file itself. They are simpler to set up but less flexible compared to Flex templates.

Example: Using the WordCount Template

Dataflow offers a pre-built Flex template for a simple word count job. This template takes text data from a Cloud Storage bucket, counts the occurrences of each word, and writes the results to another Cloud Storage bucket.

Here's how you can use it:

1. Go to the Dataflow [Create job from template](#) page in the Google Cloud Console.
2. Enter a unique job name.
3. Select the **WordCount** template from the dropdown menu.
4. Configure the template parameters:
 - **Input GCS bucket:** Specify the Cloud Storage bucket containing the text data.
 - **Output GCS bucket:** Specify the Cloud Storage bucket where the word count results will be written.
5. Click **Run job.**

5. Practical

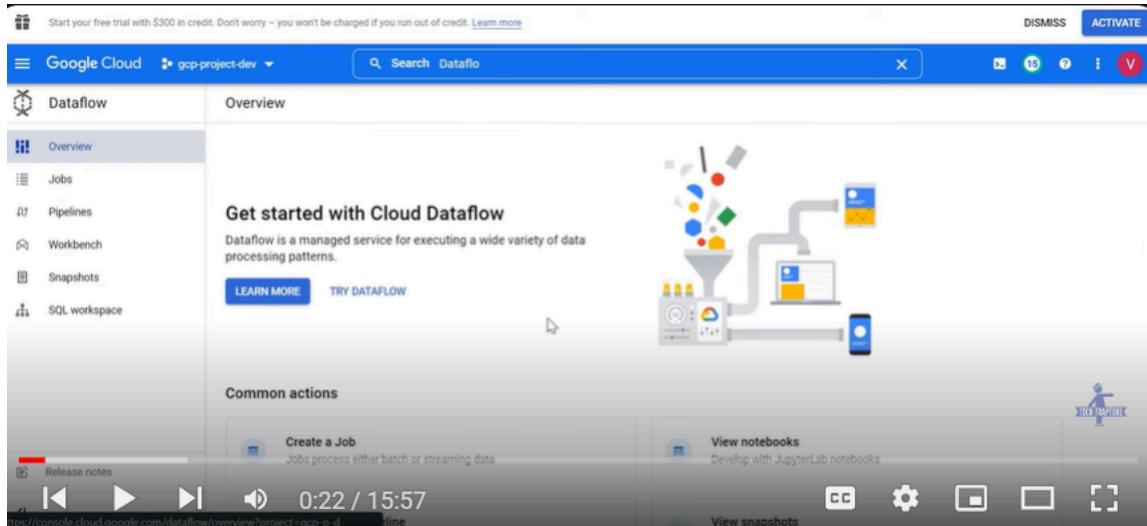
6. Next we will create batch job and stream job

Dataflow jobs

- Batch Job
 - Spanner to GCS
 - GCS to Bigquery
- Stream Job
 - Pub/Sub messages to GCS bucket
 - Spanner Change stream to bigQuery
- Utilities
 - Bulk Compress

Load data from GCS to BigQuery using DataFlow

1. Here this is the dataflow UI



2. Next we go to jobs and create job from template
3. Here first we focus on batch templates..

4. First we used text files on cloud storage to bigquery template

The screenshot shows the Google Cloud Dataflow interface for creating a new job from a template. The left sidebar lists options like Overview, Jobs, Pipelines, Workbench, Snapshots, and SQL workspace. The main area has a title 'Create job from template'. It includes fields for 'Job name' (with a note: 'Must be unique among running jobs'), 'Regional endpoint' (set to 'us-central1 (Iowa)'), and 'Dataflow template' (set to 'Text Files on Cloud Storage to BigQuery'). Below these are 'OPEN TUTORIAL' and 'RUN JOB' buttons. A note states: 'The costs of this batch pipeline will depend on the amount of data that you process.' A 'SHOW MORE' link is also present. On the right, a pipeline diagram shows a flow from 'Read from source' through a 'JavascriptT...Javascript' step to a 'BigQueryCon...ToTableRow' step.

5. We have this data...and we load few cols to bigquery
6. Next we have to give required parameters
7. And we create a new cloud storage bucket..and later we upload the js,csv and json files
8. Next we pass this values in parameters

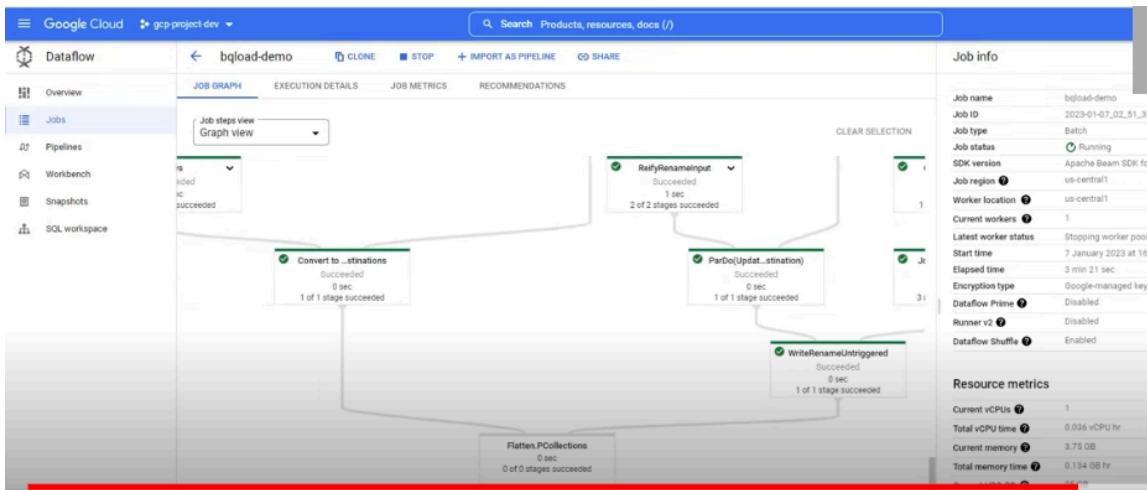
This screenshot shows the configuration details for the 'Text Files on Cloud Storage to BigQuery' template. It includes:

- JavaScript UDF path in Cloud Storage:** gs://bkt-dfdemo-00/udf.js
- JSON path:** gs://bkt-dfdemo-00/bq.json
- JavaScript UDF name:** transform
- BigQuery output table:** gcp-p-d:dfload

Below each input field is a brief description of its purpose.

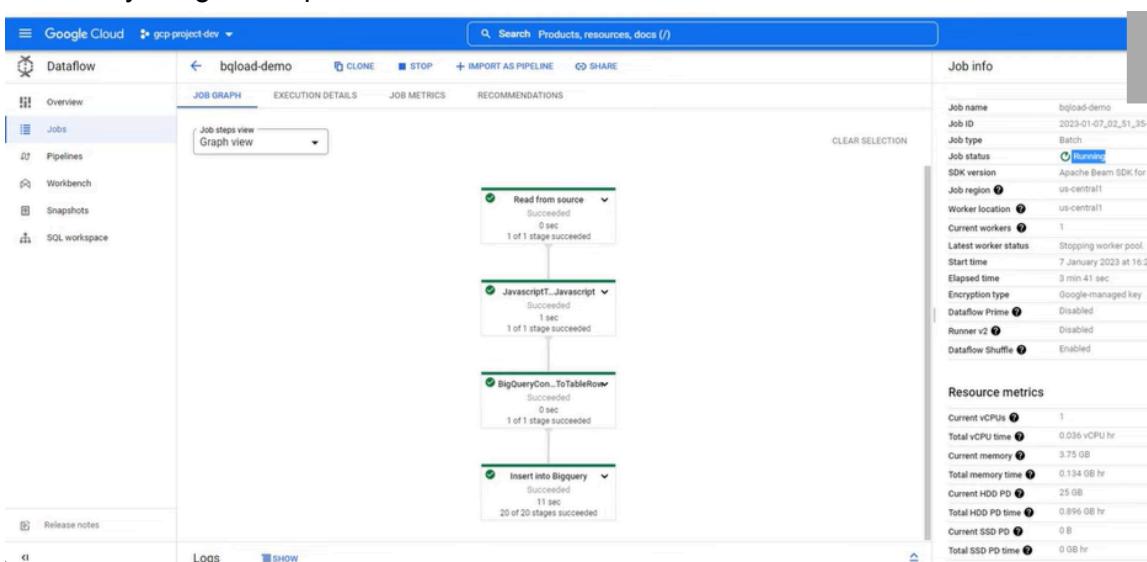
9. And also we have to create a dataset in bigquery
10. Later we run the job
11. Now in the job section...we can see the details for our job

12. This is our job graph

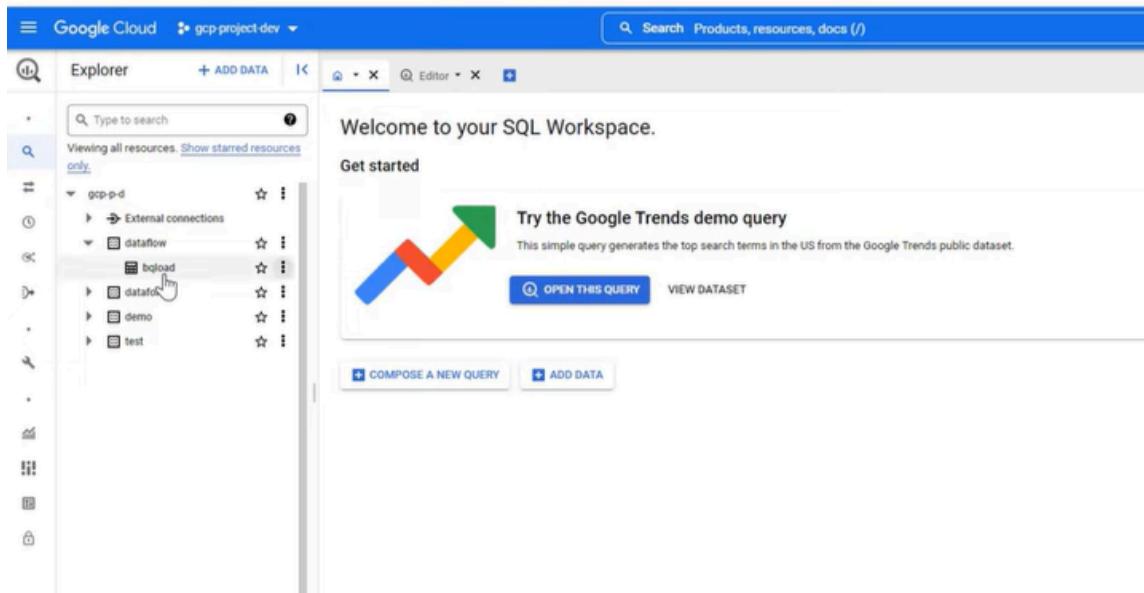


if there are any errors..then we can rectify at which stage our job failed

13. Now every thing is completed



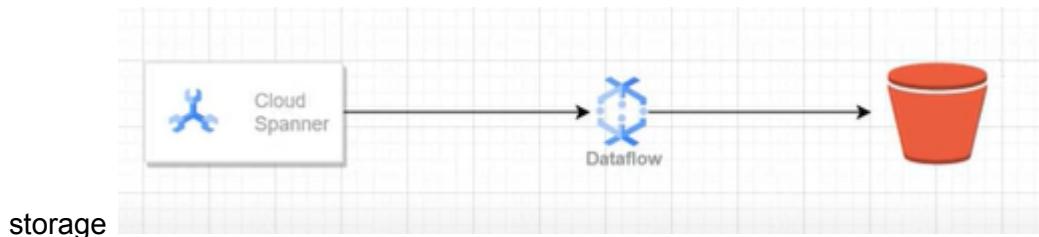
we can our data in bigquery



The screenshot shows the Google Cloud SQL Workspace interface. On the left, there's a sidebar titled 'Explorer' with a search bar and a list of resources under 'gcp-p-d'. One item, 'dataflow/bload', has a cursor icon over it, indicating it's selected. The main area is titled 'Welcome to your SQL Workspace.' and features a 'Get started' section with a 'Try the Google Trends demo query' button. At the bottom, there are buttons for 'COMPOSE A NEW QUERY' and 'ADD DATA'.

Extract Data from cloud spanner to GCS using DataFlow

1. Here dataflow will extract the data from cloud spanner and store the data in cloud



storage

Basically dataflow retrieves the data from source and loads in destination

2. What is cloud spanner?

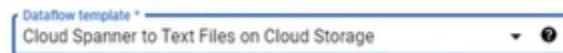
Google Cloud Spanner is a relational database service built specifically for the cloud. It combines the familiar features of relational databases, like SQL queries and strong consistency, with the scalability and flexibility of non-relational databases. Here's a breakdown of what Cloud Spanner

Imagine a large retail chain with stores across different countries. They need a database to track their inventory levels in real-time across all locations. A traditional relational database might struggle with this due to scalability limitations and potential consistency issues if data is spread across multiple servers.

Cloud Spanner can be a perfect solution for this scenario. Here's how:

1. **Data Model:** The retail chain can design a schema in Spanner to store information like product IDs, quantities, and store locations.
2. **Global Distribution:** Spanner can store the inventory data across multiple regions where the stores are located. This ensures low latency for queries and high availability in case of regional outages.
3. **Real-time Updates:** Whenever a sale occurs in a store, the inventory level for that product in that location can be updated in Spanner.
4. **Strong Consistency:** Due to strong consistency, the updated inventory level is immediately reflected across all replicas, ensuring all stores have the latest information.
5. **Scalability:** As the retail chain grows and their inventory volume increases, Spanner can automatically scale to accommodate the additional data.

3. Now lets create a job template ...and select dataflow template



4. Next we need this parameters from cloud spanner

Required parameters

Read data from Cloud Spanner Project Id of *

The Google Cloud Project id of the Cloud Spanner database that you want to read data from

Read data from Cloud Spanner Database of *

! Error: value is required

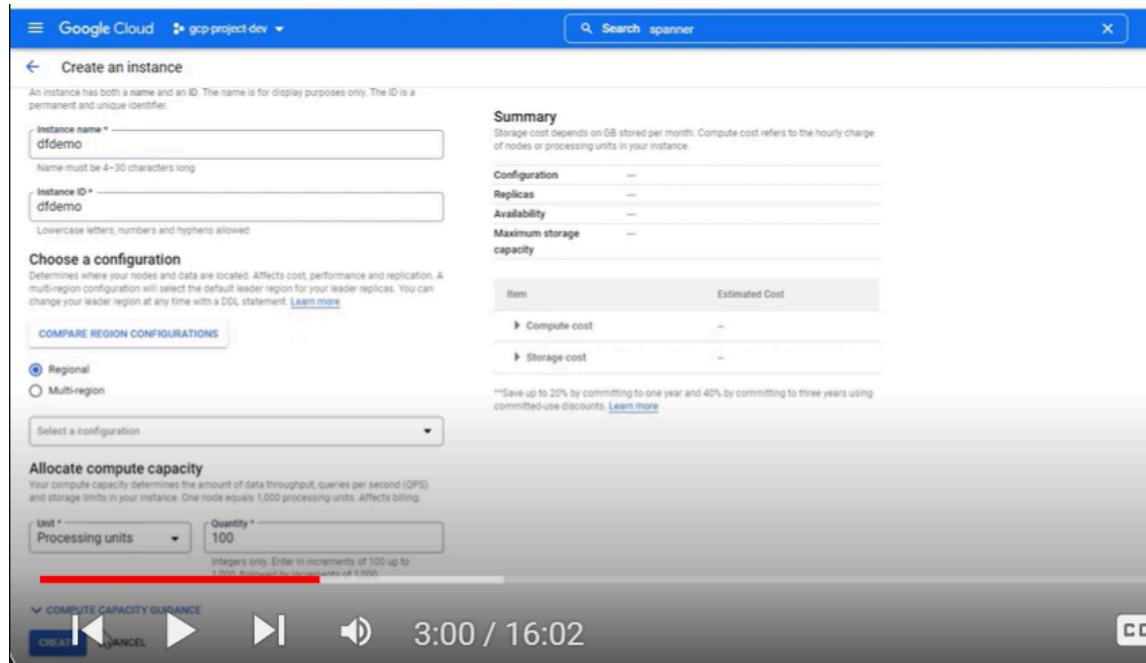
Read data from Cloud Spanner Instance of *

Instance of requested table.

Table *

Table to export

5. Next we will create cloud spanner instance to provide the required parameters



6. Next we create a db and a table and insert sample data
 7. Now we will export these 3 rows into csv file and load in GCS

SingerId	FirstName	LastName	BirthDate	LastUpdated
1	Marc	Richards		
2	Catalina	Smith		
3	Alice	Trentor		

8. Here we gave the required parameters

Required parameters

Read data from Cloud Spanner Project Id of *

The Google Cloud Project Id of the Cloud Spanner database that you want to read data from

Read data from Cloud Spanner Database of *

Database of requested table.

Read data from Cloud Spanner Instance of *

Instance of requested table.

Table *

Table to export

Cloud Storage path of where to write data * BROWSE

Cloud Storage Path Prefix as to where data should be written. Ex: gs://your-bucket/your-path

Temporary location * BROWSE

Path and filename prefix for writing temporary files. E.g.: gs://your-bucket/temp

Encryption

Google-managed encryption key
No configuration required

Customer-managed encryption key (CMEK)
Manage via Google Cloud key management service

Dataflow Prime

9. Here we got an error...which can be identified using job logs

← spannertpgcs CLONE STOP + IMPORT AS PIPELINE SHARE

JOB GRAPH EXECUTION DETAILS JOB METRICS RECOMMENDATIONS

Job steps view Graph view CLEAR SELECTION

Logs HIDE 1 5

JOB LOGS WORKER LOGS DIAGNOSTICS

Severity Error Filter Search all fields and values

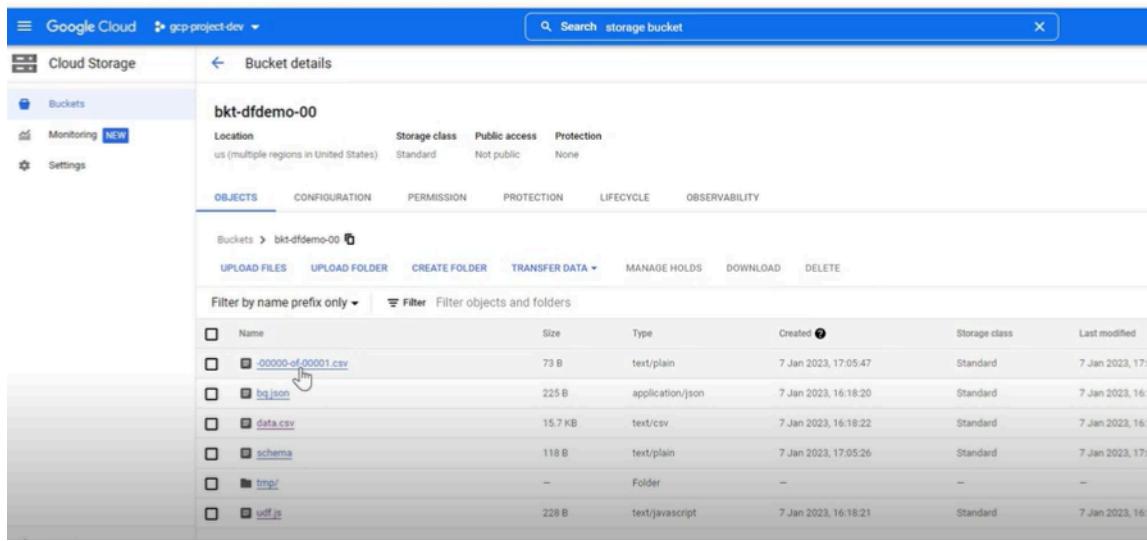
SEVERITY TIMESTAMP SUMMARY

No older entries found matching current filter.

2023-01-07 16:55:46.981 IST Error message from worker: java.lang.IllegalArgumentException: objectName must not be null or empty
com.google.common.base.Preconditions.checkNotNull(Preconditions.java:145) com.google.cloud.hadoop.io.gcsio.StorageResourceId.<init>(StorageResourceId.java:127) org.apache.beam.sdk.extensions.gcp.util.GcsUtil.create(GcsUtil.java:587)
org.apache.beam.sdk.extensions.gcp.storage.GcsFileSystem.create(GcsFileSystem.java:154)
org.apache.beam.sdk.extensions.gcp.storage.GcsFileSystem.create(GcsFileSystem.java:71)
org.apache.beam.sdk.io.FileSystems.create(FileSystems.java:243) org.apache.beam.sdk.io.FileSystems.create(FileSystems.java:2
com.google.cloud.teleport.templates.common.SpannerConverters\$ExportTransform\$ExportFn.saveSchema(SpannerConverters.java:335)
com.google.cloud.teleport.templates.common.SpannerConverters\$ExportTransform\$ExportFn.processElement(SpannerConverters.java:

10. The issue here was...we gave table name as singers..it has to be 'Singers'

11. After the job completion..we can see our data in cloud storage



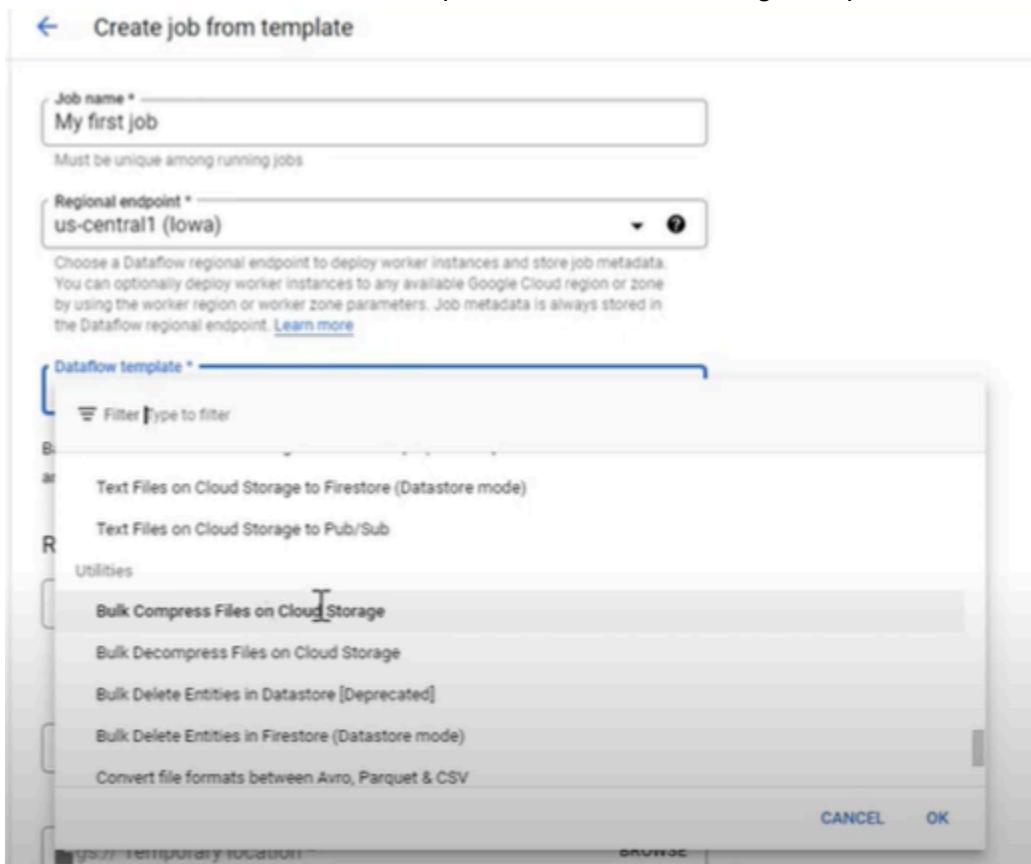
The screenshot shows the Google Cloud Storage interface for a bucket named 'bkt-dfdemo-00'. The bucket details page includes sections for Location (us), Storage class (Standard), Public access (Not public), and Protection (None). Below this, there are tabs for OBJECTS, CONFIGURATION, PERMISSION, PROTECTION, LIFECYCLE, and OBSERVABILITY. The OBJECTS tab is selected, displaying a list of objects. The list includes:

Name	Type	Created	Storage class	Last modified
_00000-of-00001.csv	text/plain	7 Jan 2023, 17:05:47	Standard	7 Jan 2023, 17:05:47
bq.json	application/json	7 Jan 2023, 16:18:20	Standard	7 Jan 2023, 16:18:20
data.csv	text/csv	7 Jan 2023, 16:18:22	Standard	7 Jan 2023, 16:18:22
schema	text/plain	7 Jan 2023, 17:05:26	Standard	7 Jan 2023, 17:05:26
tmp/	Folder	-	-	-
udf.js	text/javascript	7 Jan 2023, 16:18:21	Standard	7 Jan 2023, 16:18:21

Compress large files in bulk using dataflow

1. We create a job ...which compress the files from GCS

2. Here we have to choose bulk compress files in cloud storage template



3. Here we have multiple CSV files...and we want to compress the csv files

The screenshot shows the 'Bucket details' page in the Google Cloud Storage interface. On the left, there's a sidebar with 'Cloud Storage' selected, showing 'Buckets' and 'Monitoring' (marked as NEW). The main area displays a table of objects in the 'bkt-dfdemo-00' bucket. The columns include Name, Size, Type, Created, and Storage class. Several CSV files are listed, such as 'data_2022_06_15_03_28_30_AM.csv', 'data_2022_06_30_06_04_39_PM.csv', and 'data_2022_07_01_03_27_45_AM.csv'. Other files like 'bq.json' and 'schema' are also visible. A hand cursor is hovering over the 'data_2022_06_15_03_28_30_AM.csv' file.

4. Next we give the required parameters

Required parameters

Input file(s) in Cloud Storage *
gs://bkt-dfdemo-00/*.csv

The input file pattern Dataflow reads from. Ex: gs://your-bucket/uncompressed/*.txt

Output directory *
gs:// bkt-dfdemo-00/zipped/ BROWSE

The output location Dataflow writes to. Ex: gs://your-bucket/compressed/

Output failure file *
 gs:// bkt-dfdemo-00/failed/ BROWSE

The error log output file to use for write failures that occur during compression. Ex:
gs://your-bucket/compressed/failed.csv

Compression *
GZIP

The compression algorithm used to compress the matched files. Valid algorithms: BZIP2,
DEFLATE, GZIP

Temporary location *
gs:// bkt-dfdemo-00/tmp BROWSE

Path and filename prefix for writing temporary files. E.g.: gs://your-bucket/temp

Encryption

Google-managed encryption key
No configuration required

5. Here this will only compresses the csv files

Required parameters

Input file(s) in Cloud Storage *
gs://bkt-dfdemo-00/*.csv

The input file pattern Dataflow reads from. Ex: gs://your-bucket/uncompressed/*.txt

6. When we run the job...it will automatically create the resources that are required for the jobs
7. Here we have zip file..which contains all the compressed csv files



Create DataFlow Job using Dataflow SQL

1. We will writing queries on bigquery to create dataflow job

- Inside the bigquery...we have project->dataset->table

The screenshot shows the Google Cloud Platform BigQuery interface. At the top, there's a navigation bar with 'Google Cloud Platform' and 'My-service-project-dev'. Below it is a search bar with 'Search big'. The main area has tabs for 'FEATURES & INFO', 'SHORTCUT', and 'DISABLE EDITOR TABS'. A sidebar on the left shows 'my-service-project-dev' and 'Dev' with a 'sales' dataset selected. The main content area displays a 'Query results' table with columns: Row, Region, Country, Item_Type, Sales_Channel, Order_Priority, Order_Date, Order_ID, Ship_Date, Units_Sold, and Unit_Price. The table contains 8 rows of data. At the bottom, there are buttons for 'PERSONAL HISTORY', 'PROJECT HISTORY', and 'SAVED QUERIES', along with pagination controls.

here we can see the table data

- Now if want to see data where region is 'Europe'

Create Change Stream in Spanner and stream into BigQuery

- Change Stream helps us in CDC

What is change stream ?

In Cloud Spanner , A change stream watches and streams out a Cloud Spanner database's data changes—inserts, updates, and deletes—in near real-time.

Change stream is way to enable change data capture for cloud spanner.

- What is change data capture?

Change data capture (CDC) is a technique for identifying and tracking changes made to data in a database. It essentially captures the modifications (inserts, updates, deletes) happening to specific tables over time. This captured data can then be used for various purposes, such as:

- **Real-time analytics:** CDC allows you to analyze changes as they occur, enabling applications to react to updates promptly.
- **Data warehousing and synchronization:** Captured changes can be efficiently transferred to data warehouses or other systems for further analysis or to keep them synchronized.
- **Audit trails and compliance:** CDC provides a record of data modifications, which can be helpful for auditing purposes and ensuring regulatory compliance.

3. Practical
4. Here we have a singers table in spanner instance

The screenshot shows the Google Cloud Spanner Overview page for the 'gcp-project-dev' project. The left sidebar has sections for DATABASE (Overview, Import/Export, Backup/restore, Query, Change streams), OBSERVABILITY (Monitoring, Query Insights, Lock insights, Transaction insights, Key Visualizer). The main area shows the 'dfdemo: Overview' instance under 'INSTANCE'. It displays 'GOOGLE STANDARD SQL DATABASE dfdb: Overview'. Below this, there's a 'Schema updates' section (No recent updates) and a 'TABLES' section. The 'Singers' table is listed in the 'TABLES' section with columns: Name, Indexes, Interleaved in, and Watched by. The table has 0 rows. The right side of the screen shows performance metrics: CPU utilisation (mean) at 5.82%, Operations (Read: ~/s, Write: ~/s), Throughput (Read: ~/s, Write: ~/s), and Total database storage at 0 B.

5. On left side we have change stream click on it...and create change stream for All

This screenshot is identical to the one above, showing the Google Cloud Spanner Overview page for the 'gcp-project-dev' project. The left sidebar and main interface are the same, displaying the 'dfdemo: Overview' instance and the 'Singers' table. The table has 0 rows and is part of the 'dfdb: Overview' database.

6. Next we need to create a dataflow job

7. And we give a new job name and dataflow template

Job name * spanner-cdc
Must be unique among running jobs.

Regional endpoint * us-central1 (Iowa)
Choose a Dataflow regional endpoint to deploy worker instances and store job metadata. You can optionally deploy worker instances to any available Google Cloud region or zone by using the worker region or worker zone parameters. Job metadata is always stored in the Dataflow regional endpoint. [Learn more](#)

Dataflow template * Cloud Spanner change streams to BigQuery
Batch pipeline. Reads text from Cloud Storage, tokenizes text lines into individual words, and performs frequency count on each of the words.

Required parameters

gs:// input file(s) in Cloud Storage * BROWSE

8. Here we need theses parameters

Cloud Spanner instance ID *
The Cloud Spanner instance to read change streams from.

Cloud Spanner database *
The Cloud Spanner database to read change streams from.

Cloud Spanner metadata instance ID *
The Cloud Spanner instance to use for the change streams connector metadata table.

Cloud Spanner metadata database *
The Cloud Spanner database to use for the change streams connector metadata table.

Cloud Spanner change stream *
The Cloud Spanner change stream to read from.

BigQuery dataset *

9. We create new spanner instance and mt DB..and give them as parameters

[Create job from template](#)

Required parameters

Cloud Spanner instance ID * The Cloud Spanner instance to read change streams from.

Cloud Spanner database * The Cloud Spanner database to read change streams from.

Cloud Spanner metadata instance ID * The Cloud Spanner instance to use for the change streams connector metadata table.

Cloud Spanner metadata database * The Cloud Spanner database to use for the change streams connector metadata table.

Cloud Spanner change stream * The Cloud Spanner change stream to read from.

BigQuery dataset * The BigQuery dataset for change streams output.

Encryption

Google-managed encryption key
No configuration required

Customer-managed encryption key (CMEK)
Manage via Google Cloud key management service

[SHOW OPTIONAL PARAMETERS](#)

10. [RUN JOB](#) next we give CDC that we created and a bigquery dataset
11. And run this job

12. Now lets change any row in the singers table

The screenshot shows the Google Cloud Spanner interface. On the left, there's a sidebar with 'Spanner' selected under 'DATABASE'. Below it are 'Overview', 'Import/Export', 'Backup/restore', and 'Query' (which is currently selected). Under 'OBSERVABILITY', there are 'Monitoring', 'Query insights', 'Lock insights', and 'Release notes'. The main area is titled 'QUERY 1' and contains the following SQL code:

```
1 select * from Singers
2 update Singers set FirstName = 'Vishal' WHERE SingerId = 1
```

Below the code, there are tabs for 'SCHEMA', 'RESULTS', and 'EXPLANATION'. The 'RESULTS' tab is selected, showing the output of the 'select * from Singers' query:

SingerId	FirstName	LastName	BirthDate	LastUpdated
1	Marc	Richards		
2	Catalina	Smith		
3	Alice	Trentor		

13. Now here in the bigquery...we can see all the changes made in our singers table

The screenshot shows the Google BigQuery interface. On the left, the 'Explorer' sidebar shows a project named 'gcp-p-d' containing 'External connections', 'dataflow' (with 'Singers_changelog' selected), 'bqload', 'data', 'dataform', 'demo', and 'test'. The main area is titled 'Query results' and displays the following SQL query and its results:

```
1 SELECT * FROM `gcp-p-d.dataflow.Singers_changelog` LIMIT 1000
```

The results table has columns: Row, SingerId, FirstName, LastName, BirthDate, LastUpdated, and _metadata_spanner_mod_type. The data is as follows:

Row	SingerId	FirstName	LastName	BirthDate	LastUpdated	_metadata_spanner_mod_type
1	1	Vishal	Richards	null	null	UPDATE
2	1	null	null	null	null	DELETE
3	4	abc	null	null	null	INSERT
4	1	Vishal	Richards	null	null	UPDATE

Data Loss Prevention (DLP) API in GCP

1. What is DLP API?

What is DLP API?

Cloud Data Loss Prevention (DLP) API: Provides methods for detection, risk analysis, and de-identification of privacy-sensitive fragments in text, images, and Google Cloud Platform storage repositories

Personally identifiable information (PII) is information that, when used alone or with other relevant data, can identify an individual.

2. It identifies the PII data ..such as SSN Number, Phone number, Aadhar card number etc...and de-identifies them using

What DLP API Do ?

- Detect/identify sensitive data such as PII data
- De-identify data using
 - Redaction
 - Replacement
 - Masking
 - Encryption
- Re-identify data

3. Demo

4. GCP has provided a demo..where can type something and it identifies whether it is PII data or not

5. Practical

6. First we have to enable this API

7. First we'll identify the sensitive data in table or file..so we create a template for inspect

8. Then we can configure detection of sensitive data(what to consider as sensitive data)
9. We can also add exclusion rules as well..(to exclude particular names or entity as sensitive data)
10. And we create this template
11. Later we create new bucket in GCS and add a csv file...after adding the file...we retrieve the full path of file
12. Next we go to DLP and we create inspect job...inside that we give location of our file(path)

The screenshot shows the 'Create job or job trigger' page in the Google Cloud DLP interface. The left sidebar shows various security services like Certificate manager, Protect, Security Command Centre, Asset inventory, reCAPTCHA Enterprise, BeyondCorp Enterprise, Data Loss Prevention (selected), Web Security Scanner, and Certificate Authority Service. The main form has a 'Resource location' dropdown set to 'Global (any region)'. A 'Resource preview' box shows 'projects/gcp-p-d/locations/global/dlpJobs/detect-pan'. The 'Location' section specifies a 'Storage type' of 'Google Cloud Storage' and a 'Location type' of 'Scan a single file or folder path'. The 'URL' field is filled with 'gs://bkt-dlpdemo/sampleuserdata.csv'.

13. Then we can also give sampling
14. Now in the add actions..we'll save this results to bigquery by creating a dataset in bigquery
15. Here are the inspection results

Name	Description	Categories	Total
PERSON_NAME	A full person name, which can include first...	Location: GLOBAL, Data type: PII	25
DATE_OF_BIRTH	A date that is identified by context as a dat...	Location: GLOBAL, Data type: PII	11
INDIA_PAN_INDIVIDUAL	The Indian Personal Permanent Account N...	Location: INDIA, Data type: SPii	11
PASSWORD	Clear text passwords in configs, code and...	Location: GLOBAL, Data type: SPii	11
EMAIL_ADDRESS	An email address identifies the mailbox th...	Location: GLOBAL, Data type: PII	10
STREET_ADDRESS	A street address. Note: Not recommended...	Location: GLOBAL, Data type: PII	3
ADVERTISING_ID	Identifiers used by developers to track use...	Location: GLOBAL, Data type: PII	0
AGE	An age measured in months or years.	Location: GLOBAL, Data type: PII	0
ARGENTINA_DNI_NUMBER	An Argentine Documento Nacional de Iden...	Location: ARGENTINA, Data type: GOVERNMENT_ID	0

16. And we can see this results inside a table in Bigquery

17. Next we'll create a template to deidentify

The screenshot shows the 'Create template' interface in Google Cloud DLP. On the left, a sidebar lists various security tools under 'Protect': Certificate manager, Security Command Centre, Asset inventory, reCAPTCHA Enterprise, BeyondCorp Enterprise, Data Loss Prevention (selected), and Web Security Scanner. The main panel is titled 'Define template'. Under 'Template type *', 'De-identify (remove sensitive data)' is selected. Below it, 'Data transformation type' is set to 'InfoType'. A detailed description follows: 'Treat the data set as free-form text and apply the same free text transformation everywhere.' Other options like 'Record' and 'Image' are also listed with their descriptions.

18. Here for PII data it masks with '#'

The screenshot shows the 'Create template' interface in Google Cloud DLP. The 'Define template' step is completed with a checkmark. The 'Configure de-identification' step is currently active. Under 'Transformation rule', a 'Transformation method' is selected: 'Mask with character'. The 'Masking character' field contains the character '#'. There is also a link to 'Learn more' about transformation techniques.

Tokenize/Redact sensitive data using DLP API and DataFlow

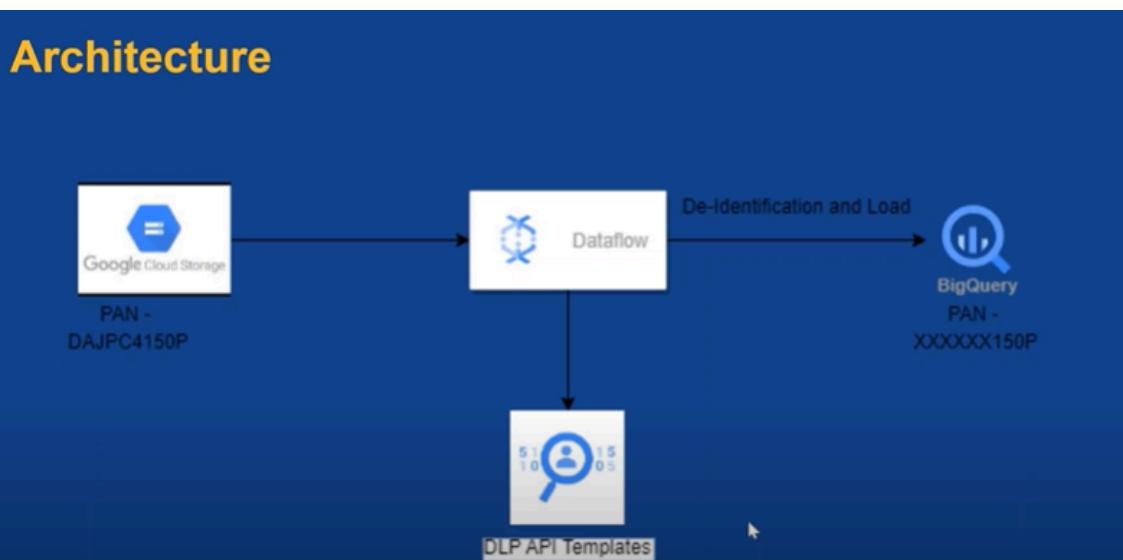
1. Here first step would be to create Inspect template

Steps

- Create Inspect template
- Create De-identify template
- Create storage bucket
- Create dataflow Job
- Upload file with sensitive data to bucket

2.

3. This is our architecture



4. Practical

5. Now inside the dataflow we create a job

The screenshot shows the 'Create job from template' page in the Google Cloud Dataflow interface. On the left, there's a sidebar with links for Overview, Jobs, Pipelines, Workbench, Snapshots, and SQL workspace. The main area has a search bar at the top right. Below it, the 'Create job from template' form is displayed. It includes fields for 'Job name' (set to 'dipapi-test'), 'Regional endpoint' (set to 'us-central1 (Iowa)'), 'Dataflow template' (set to 'Word Count'), and 'Input file(s) in Cloud Storage' (with a browse button). To the right, a flow diagram shows the pipeline stages: ReadLines, WordCount.CountWords, MapElements, and WriteCounts. A red box highlights the 'Input file(s)' field.

6. Here we have to choose this template

The screenshot shows the 'Create job from template' form again. The 'Dataflow template' dropdown is open, displaying a list of available templates. The 'Word Count' template is selected and highlighted in blue. Other visible templates include 'Get Started', 'Process Data Continuously (stream)', 'Cloud Spanner change streams to BigQuery', 'Cloud Spanner change streams to Cloud Storage', 'Cloud Spanner change streams to Pub/Sub', 'Data Masking/Tokenization from Cloud Storage to BigQuery (using Cloud DLP)', and 'Data Masking/Tokenization from Cloud Storage to BigQuery (using Cloud DLP) with BQ Storage Write API support'. A red box highlights the 'Word Count' template in the list.

7. Next we have to give the identification template too

8. So in our DLP we have inspect and de-identify templates..so we copy url of deidentify template and paste it above
9. After that we will give our destination as bigquery
10. Later we upload a csv file to GCS and run our job
11. Here in our big query...we can see our data is masked

Row	state	country	birthdate	password	last_login	PAN
1	Tennessee	Malawi	08-09-2013	sensitive_password	02-09-1983 13:55	*****4444T
2	WA	United Arab Emirates	14-08-1995	sensitive_password	09-04-2004 21:08	*****509Y
3	Idaho	Brazil	02-01-1994	sensitive_password	14-04-1972 07:28	*****225A
4	Wyoming	Trinidad and Tobago	06-11-1984	sensitive_password	06-02-1981 12:26	*****150P
5	North Dakota	Anguilla	20-02-2002	sensitive_password	15-03-1985 20:11	*****616I
6	H	India	12-10-1993	sensitive_password	09-04-2004 21:08	*****014J
7	Wisconsin	Guinea	07-10-1990	sensitive_password	20-12-1992 10:09	*****856B
8	Michigan	Morocco	07-12-2015	sensitive_password	21-11-1989 13:24	*****905E
9	Pennsylvania	Bruni Darussalam	15-08-1981	sensitive_password	30-12-1986 04:35	*****049V
10	Alabama	Sao Tome and Principe	27-06-2011	sensitive_password	25-05-2001 17:29	*****008X
11	Ashington	Antigua and Barbuda	30-09-1989	sensitive_password	10-07-2004 08:13	*****789D

"/>

12.