Contents

[Cloud-Based Data Cleaning — Concepts & Tools 1](#_Toc204116061)

[Google Cloud Storage (GCS) 1](#_Toc204116062)

[Cloud Functions 1](#_Toc204116063)

[Vertex AI Workbench 1](#_Toc204116064)

[Pandas 1](#_Toc204116065)

[gcsfs & fsspec 1](#_Toc204116066)

[Runtime Logging Strategy 1](#_Toc204116067)

[CloudEvents Format 2](#_Toc204116068)

[GCP Console (UI) Steps — Project & Resource Setup 2](#_Toc204116069)

[Login & Project Creation 2](#_Toc204116070)

[Create a Storage Bucket 2](#_Toc204116071)

[Organize Folders Inside the Bucket 2](#_Toc204116072)

[Create a Service Account 2](#_Toc204116073)

[IAM Access Controls 3](#_Toc204116074)

[Create a Vertex AI Workbench Instance 3](#_Toc204116075)

[Use JupyterLab for Coding & Deployment 3](#_Toc204116076)

[Cloud-Based Data Cleaning — Step-by-Step Guide 3](#_Toc204116077)

[Step 1: Prepare GCS Buckets & Folders 3](#_Toc204116078)

[Step 2: Write the Data Cleaning Pipeline 3](#_Toc204116079)

[Step 3: Build Cloud Function Logic 4](#_Toc204116080)

[Step 4: Package & Deploy the Cloud Function 4](#_Toc204116081)

[Step 5: Trigger the Pipeline 4](#_Toc204116082)

[Bonus: Safe Shutdown & Cleanup 5](#_Toc204116083)

[Code Snippets 5](#_Toc204116084)

[Io\_gcs.py 5](#_Toc204116085)

[Cleaning.py 5](#_Toc204116086)

[Pipeline.py 6](#_Toc204116087)

[Main.py 7](#_Toc204116088)

[Requirements.txt 7](#_Toc204116089)

# Cloud-Based Data Cleaning — Concepts & Tools

This workflow automates data ingestion, transformation, and output using cloud-native components. Below are the key tools and concepts involved:

## Google Cloud Storage (GCS)

* **Role:** Stores raw CSV files, cleaned outputs, and per-run logs.
* **Folders Used:**
  + raw/ → source files (trigger)
  + cleaned/ → cleaned CSV outputs
  + logs/ → JSON logs for audit trail
* **Concepts Introduced:**
* URI format: gs://bucket-name/path
* Avoiding trigger loops by filtering object paths

## Cloud Functions

* **Role:** Auto-triggers cleaning on file upload in raw/
* **Language:** Python 3.10 (deployed via zipped bundle)
* **Highlights:**
  + Trigger filtered to only .csv in raw/
  + Function logic moved from Prefect-style to vanilla Python
  + Uses @functions\_framework.cloud\_event for event handling
* **Dependencies:** functions-framework, google-cloud-storage

## Vertex AI Workbench

* **Role:** Local development environment for cleaning code
* **Used For:**
  + Notebook-based testing
  + Packaging and pushing Cloud Function zip bundles
  + Inspecting main.py, pipeline.py, and logs
* **Note:** Kernel and working directory behavior impacted ZIP contents

## Pandas

* **Role:** Data cleaning logic inside pipeline
* **Common Ops:** .read\_csv(), .dropna(), .to\_csv(index=False)
* **Used In:** gcs\_cleaning\_pipeline() to transform raw files

## gcsfs & fsspec

* **Role:** Read/write GCS files via Pandas natively
* **Limitation Discovered:**
  + gcsfs.open(..., mode='a') is not safe or reliable in GCP
  + Fork-safety error due to Cloud Functions runtime constraints
* **Resolution:**
* Replaced with local file write + GCS upload\_from\_filename()

## Runtime Logging Strategy

* **Problem:** GCS doesn’t support append; fork-safe limitations
* **Solution:**
* Log saved locally (/tmp/) → uploaded to GCS
* Timestamped log filenames: cleaning\_log\_.json

## CloudEvents Format

* **Used In:** Cloud Function trigger
* **Handled With:** @functions\_framework.cloud\_event
* **Concepts:**
* cloud\_event.data["name"] used instead of event["name"]
* Ensures compatibility with modern event schema

# GCP Console (UI) Steps — Project & Resource Setup

## Login & Project Creation

* Visit <https://console.cloud.google.com>
* Sign in using your Google account
* In the header bar, click on the **project selector**
* Choose **“New Project”**
  + Give it a meaningful name (e.g., auto-cleaning-project)
  + Select billing account (if applicable)
  + Click **“Create”**

## Create a Storage Bucket

* Go to **Navigation Menu → Storage → Buckets**
* Click **“Create”**
* Configure:
  + **Name:** e.g., sandeep-data-bucket2
  + **Location:** Multi-region or region as preferred
  + Keep other defaults unless needed
* Click **“Create”**

Bucket appears in your list. Click into it to view objects.

## Organize Folders Inside the Bucket

Inside the bucket UI:

* Click **“Create Folder”** and set up:
  + raw/
  + cleaned/
  + logs/
  + cloud-functions/

These give structure to your file triggers, outputs, and logs.

## Create a Service Account

* Navigate to **IAM & Admin → Service Accounts**
* Click **“Create Service Account”**
  + Name: auto-cleaning-sa
  + Description: For Cloud Function execution
* Grant role:
  + Storage Object Admin (for read/write access)
* Finish creation and note the email (e.g., auto-cleaning-sa@yourproject.iam.gserviceaccount.com)

## IAM Access Controls

* Go to **IAM & Admin → IAM**
* Locate your service account in the list
* Use **Edit** to:
  + Confirm it has Storage Object Admin
  + Optionally add Cloud Functions Invoker if triggered externally
* This controls execution rights over GCS files

## Create a Vertex AI Workbench Instance

* Go to **Vertex AI → Workbench**
* Click **“New Instance”**
  + Choose **JupyterLab (Managed)** or custom environment
  + Machine type: Start with basic (e.g., n1-standard-1)
* Deploy and open **JupyterLab**

## Use JupyterLab for Coding & Deployment

Inside your Vertex Jupyter workspace:

* Create files: main.py, requirements.txt, and gcs\_etl/pipeline.py
* Run Python cells to test cleaning functions
* Use terminal (lower pane):
  + Run zip commands
  + Execute gsutil and gcloud CLI for deployment
* Debug kernel issues by restarting notebook session if needed

# Cloud-Based Data Cleaning — Step-by-Step Guide

This outlines the full workflow, from ingesting raw files to producing cleaned outputs and run-level logs.

## Step 1: Prepare GCS Buckets & Folders

**Create Bucket:**  
Example: sandeep-data-bucket2

**Folder Structure:**

* raw/ → incoming source CSVs
* cleaned/ → cleaned output files (CSV)
* logs/ → per-run log files (.json)

## Step 2: Write the Data Cleaning Pipeline

pipeline.py — inside gcs\_etl/ folder

**Function:** gcs\_cleaning\_pipeline(raw\_path: str)

* Reads raw CSV from GCS
* Cleans data (e.g., removes empty rows)
* Saves cleaned CSV to cleaned/ path
* Generates a log dictionary with:
  + Status
  + Input/output file URIs
  + Timestamp

log\_run(log\_dict, log\_path)

* Writes log entry locally to /tmp/
* Uploads as cleaning\_log\_<timestamp>.json to logs/

## Step 3: Build Cloud Function Logic

main.py

**Trigger Format:** CloudEvents with @functions\_framework.cloud\_event

def auto\_cleaning(cloud\_event: CloudEvent):

data = cloud\_event.data

object\_name = data["name"]

# Filter: Only CSVs inside raw/

if not object\_name.startswith("raw/") or not object\_name.endswith(".csv"):

print(f"Ignoring file outside raw/: {object\_name}")

return

raw\_path = f"gs://{data['bucket']}/{object\_name}"

gcs\_cleaning\_pipeline(raw\_path)

**Prevents recursive triggers from cleaned/ or logs/**

## Step 4: Package & Deploy the Cloud Function

zip -r auto\_cleaning.zip main.py requirements.txt gcs\_etl/

gsutil cp auto\_cleaning.zip gs://sandeep-data-cleaning/cloud-functions/

gcloud functions deploy auto-cleaning \

--runtime=python310 \

--trigger-event=google.storage.object.finalize \

--trigger-resource=sandeep-data-bucket2 \

--entry-point=auto\_cleaning \

--source=gs://sandeep-data-cleaning/cloud-functions/auto\_cleaning.zip \

--region=us-east1

**Uses Python ZIP bundle and functions-framework**

## Step 5: Trigger the Pipeline

**Upload a file to raw/:**

gsutil cp test.csv gs://sandeep-data-bucket2/raw/

Function auto-triggers and:

* Cleans the file
* Saves to cleaned/
* Creates log:  
  logs/cleaning\_log\_<timestamp>.json

## Bonus: Safe Shutdown & Cleanup

gcloud functions delete auto-cleaning --region=us-east1

gsutil rm -r gs://sandeep-data-bucket2/{cleaned/,logs/,raw/}

gsutil rm gs://sandeep-data-cleaning/cloud-functions/auto\_cleaning.zip

# Code Snippets

### Io\_gcs.py

import pandas as pd

import gcsfs

from google.auth import default

from google.cloud import storage

from urllib.parse import urlparse

creds, \_ = default(scopes=["https://www.googleapis.com/auth/cloud-platform"])

fs = gcsfs.GCSFileSystem(token=creds)

def read\_csv(bucket\_path):

return pd.read\_csv(f"gs://{bucket\_path}", storage\_options={"token": creds})

def save\_csv(df, gcs\_uri):

local\_path = "/tmp/cleaned.csv"

df.to\_csv(local\_path, index=False)

# Properly parse the GCS URI

parsed = urlparse(gcs\_uri)

bucket\_name = parsed.netloc # e.g. 'sandeep-data-cleaning'

object\_path = parsed.path.lstrip("/") # e.g. 'cleaned/file.csv'

if not bucket\_name or not object\_path:

raise ValueError(f"Invalid GCS URI: {gcs\_uri}")

client = storage.Client()

bucket = client.bucket(bucket\_name)

blob = bucket.blob(object\_path)

blob.upload\_from\_filename(local\_path)

### Cleaning.py

import pandas as pd

EXPECTED\_SCHEMA = {

"transaction\_id": "int64",

"customer\_name": "object",

"amount": "float64",

"date": "datetime64[ns]"

}

def clean\_sales\_data(df):

df.drop\_duplicates(inplace=True)

df['amount'] = df['amount'].fillna(0)

df['customer\_name'] = df['customer\_name'].str.strip().str.title()

df['date'] = pd.to\_datetime(df['date'], format="%d-%m-%Y", errors="coerce")

return df

def validate\_schema(df):

for col, dtype in EXPECTED\_SCHEMA.items():

if col not in df.columns:

raise ValueError(f"Missing column: {col}")

if str(df[col].dtype) != str(dtype):

print(f"⚠️ Column '{col}' has type {df[col].dtype}, expected {dtype}")

### Pipeline.py

from gcs\_etl.cleaning import clean\_sales\_data, validate\_schema

from gcs\_etl.io\_gcs import read\_csv, save\_csv

from datetime import datetime

import gcsfs

import json

from google.auth import default

from google.cloud import storage

creds, \_ = default(scopes=["https://www.googleapis.com/auth/cloud-platform"])

fs = gcsfs.GCSFileSystem(token=creds)

def log\_run(log\_dict, log\_gcs\_path):

local\_log\_path = "/tmp/cleaning\_log.jsonl"

# Write single line to local file

with open(local\_log\_path, "w") as f:

f.write(json.dumps(log\_dict) + "\n")

# Upload to GCS — overwriting log

parsed = urlparse(log\_gcs\_path)

bucket\_name = parsed.netloc

object\_path = parsed.path.lstrip("/")

client = storage.Client()

bucket = client.bucket(bucket\_name)

blob = bucket.blob(object\_path)

blob.upload\_from\_filename(local\_log\_path)

def gcs\_cleaning\_pipeline(raw\_path):

#raw\_path = "sandeep-data-bucket2/raw/D16\_sales\_data\_gcp.csv"

timestamp = datetime.now().strftime('%Y%m%d\_%H%M%S')

cleaned\_path = f"gs://sandeep-data-bucket2/cleaned/D16\_cleaned\_{timestamp}.csv"

log\_path = "gs://sandeep-data-bucket2/logs/cleaning\_log.jsonl"

df = read\_csv(raw\_path)

df\_cleaned = clean\_sales\_data(df)

validate\_schema(df\_cleaned)

save\_csv(df\_cleaned, cleaned\_path)

log\_run({

"status": "success",

"input\_file": raw\_path,

"output\_file": cleaned\_path,

"timestamp": timestamp,

}, log\_path)

### Main.py

import functions\_framework

from gcs\_etl.cleaning import clean\_sales\_data, validate\_schema

from gcs\_etl.io\_gcs import read\_csv, save\_csv

from datetime import datetime

from gcs\_etl.pipeline import gcs\_cleaning\_pipeline

from cloudevents.http import CloudEvent

@functions\_framework.cloud\_event

def auto\_cleaning(cloud\_event: CloudEvent):

data = cloud\_event.data

object\_name = data["name"]

if not object\_name.startswith("raw/") or not object\_name.endswith(".csv"):

print(f"Ignoring file outside raw/: {object\_name}")

return

raw\_path = f"gs://{data['bucket']}/{object\_name}"

gcs\_cleaning\_pipeline(raw\_path)

### Requirements.txt

functions-framework

pandas

gcsfs

google-auth