# Wikipedia Task

# Technical Documentation

## Objective

The task at hand is to set up a scheduled ingestion process to extract the top 100 most viewed Wikipedia pages for a specified date, along with their associated categories, and save this information in BigQuery. The process is configured to run once a day and backfill data from April 1st, 2023, to the present.

## Implementation

### Code Files

The implementation consists of two main Python files: `main.py` and `functions.py`.

### main.py

This file contains the main function that triggers the data extraction and upload process. It calls the `main\_process` function with the start and end dates, and then uploads the resulting data to a BigQuery table named 'dataEngineeringTask.wikiDataNormal'. If the table already exists, the data will be appended to it.

### functions.py

This file contains several helper functions that are used to extract and process the data. These include:

- `get\_page\_views(date,access,app)`: This function queries the Wikimedia REST API for the top viewed pages on a particular date.

- `get\_dates(from\_date, to\_date)`: This function generates a list of dates from the start date to the end date.

- `process\_dates(from\_date=None, to\_date=None)`: This function processes a range of dates and calls the `get\_page\_views` function for each date to get the page view data for that date.

- `get\_categories(page\_title)`: This function gets the categories for a particular Wikipedia page by querying the Wikipedia API.

- `process\_dataframe(data)`: This function processes a dataframe containing page view data and appends a new column with the categories for each article.

- `main\_process(from\_date, to\_date)`: This is the main processing function that gets the page view data for a range of dates, filters out rows with rank greater than 100, and processes the dataframe to get categories for each article.

## Configuration

The Google Cloud Function (GCF) is configured with the following settings:

- Region: us-central1

- Memory allocated: 8 GiB

- CPU: 2

- Timeout: 3,600 seconds

- Service account: 21784382617-compute@developer.gserviceaccount.com

### Secrets

The API secret key is stored in the Google Cloud Secret Manager and exposed as an environment variable `SECRETKEY`. Also it has been ensured that no keys are committed to the GitHub repository.

### Scheduler

The Cloud Scheduler job `wikiRunSchedulerv1` is configured to trigger the Cloud Function once a day at 7 am BST. The scheduler uses the URL of the Cloud Run service as the trigger.

BigQuery Table

The BigQuery table `advancedml-saahil.dataEngineeringTaskv1.wikipediaTopPages` is used to store the extracted data. As of the latest run, the table contains 104,643 rows of data, covering the top 100 pageview pages from April 1, 2023, to the present.

### Terraform Configuration Files

The infrastructure is defined and managed using Terraform. The main configuration file is `main.tf`, and the variables used in the main configuration are defined in `terraform.tfvars`. The Terraform plan is saved as `tfPlanv1`.

### GitHub Repository

The code is hosted in a private GitHub repository named `interview-de-task`. Both principals have been provided with access to the repository and the Cloud platform.

Workflow

The workflow is built using Terraform Infrastructure as Code (IaC) and is tested for robust functionality. The code is written in Python, adhering to high standards of efficiency, scalability, and reproducibility. The code is deployed on a Google Cloud Function, which is triggered by a Cloud Scheduler job to extract data and upload it to BigQuery daily.

Future Improvements

While the current architecture allows for a good amount of scalability due to the use of Terraform, further development could include the use of Google Cloud Build and Cloud Run. However, for a singular routine task such as this, it might be an overkill. A simpler alternative could be a Compute Engine VM with only the required scripts and API key as an environment variable. However, it has not been used keeping in mind that the key could very well be an even more sensitive one.

## Failsafe or Backup

In addition to the main implementation, a local backfill notebook has been created as a failsafe measure. This notebook can be run locally and serves two main purposes:

1. Backup: In case of any failure or disruption in the main pipeline (Cloud Function, Scheduler, or BigQuery), this notebook can be used to manually extract the data and upload it to BigQuery. This ensures that the data ingestion process can continue uninterrupted even in the face of unforeseen issues with the main pipeline.

2. Local Run: The notebook can also be used for local testing and development. For example, if there are any changes or updates to the data extraction and processing logic, these can be first tested locally using the notebook before updating the main pipeline.

The notebook contains the same Python code for data extraction and processing as the main pipeline. It requires the same environment variables (such as the API secret key) and Python libraries as specified in the `requirements.txt` file, however there are some minor tweaks.

Please note that while the notebook provides a valuable backup, it requires manual intervention to run. Therefore, it's crucial to monitor the main pipeline regularly to detect any issues early and either fix them or switch to the backup notebook as needed.