Chess.com API - Data Engineering Pipeline

I've designed an end-to-end ETL pipeline for ingesting large volumes of data from the chess.com API into GCP to improve data engineering skills.

Latest revision of chess data application deployed to Cloud Run: > https://chess-app-dash-810099024571.europe-west1.run.app/

Repositories

Update 2025-07-27: > Migrated logic to seperate repo's to add modularity and simplify deployment processes

- Data Ingestion/Loading + Data Visualisation: https://github.com/Filpill/chess_analysis
- Data Transformation: https://github.com/Filpill/dbt_chess
- Data Orchestration: https://github.com/Filpill/airflow_chess

Data Ingestion Into GCS

All the historically archived game data in the API is segmented by the player endpoint. The approach is to harvest a list of usernames by making a request to the leaderboards endpoint of the chess.com API to get the top players.

For each player in the games archive, we request for each monthly date period.

Data landing into the bucket, mirrors the API endpoints paths for clarity, consistency and idempotency.

On each pipeline run, a check is performed to see if the data already exists inside our GCS bucket which yields the following benefits: - Efficiency: | Not wasting our time re-requesting data we've already ingested. - Restartability | We can easily restart our ingestion scripts if the pipeline fails.

After the GCS ingestion is completed, a loading script will follow up to land the data from GCS into BigQuery.

CI/CD Process

This is a guideline of the CI/CD process that gets executed when the main branch is updated.

> This does not include updates made to docker images, all image deployments into artifact registry are conducted manually via the CLI

Project Directories

The project is segmented into the following directories which contain the following:

Directory	Description
scripts/	Core python scripts/notebooks to process chess data
scripts/cloud_run_services/	Cloud Run Services for controlling GCP resources
scripts/functions/	Functions which are imported into the core data processing scripts
scripts/inputs/	Input parameter files for passing into core data processing scripts
scripts/gcloud/	Miscellaneous gcloud commands for interacting with GCP

Directory	Description
docker/	Contains dockerfiles to define toolng environment for running data deployment pipeline
dash/	Contains files for developing “Dash” python applications
diagrams/	Illustrations for architectural design

Terraform Configuration

The following is an overview of the terraform configuration:

Terraform File	Description
.github/workflows/terraform.yml	Terraform config for GCP authentication and terraform deployment steps
providers.tf	Declaring GCP project provider
main.tf	GCP resource configuration to be created/updated/deleted
backend.tf	Terraform state file location on GCS
variables.tf	Variable declaration for terraform configuration
terraform.tfvars*	Values for declared variables

Python Environment

Python environment is created using the uv package manager and the following files specify dependencies:

File	Description
pyproject.toml	Keep track of key library dependencies
uv.lock	Precisely captures all python package versions
requirements.txt	Default python requirements file compiled from pyproject.toml

To create a virtual environment:

```
uv venv --python 3.11
```

To add packages to pyproject.toml:

```
uv add <package-name>
```

To compile package to requirements.txt:

```
uv pip compile pyproject.toml > scripts/requirements.txt
```

To activate virtual environment locally:

```
source .venv/bin/activate
```

To install python library requirements:

```
uv sync
```

Docker Image Deployment

Docker images are the primary method of packaging together the necessary tooling for running workloads on Virtual Machines.

Takes a configuration style approach in building images, but supplying input parameters for determining the image name and Artifact Registry Repo.

Images are stored in the GCP Artifact Registry.

Docker File	Description
Dockerfile	Configuration creating tool environment ready for executing
variables	List of variables for plugging into local command-line docker scripts

Docker Scripts For Building and Pushing Images to Artifact Registry

```bash
#!/bin/bash
BASE_DIRECTORY="$(pwd)"
source $BASE_DIRECTORY/variables

read -p "Do you want to enable caching? (y/n): " CACHE_CHOICE

if [[ "$CACHE_CHOICE" == "y" || "$CACHE_CHOICE" == "Y" ]]; then
  docker build -t $AR_TYPE/$GCP_PROJECT/$AR_REPO/$IMAGE_NAME:$IMAGE_TAG .
else
  docker build --no-cache -t $AR_TYPE/$GCP_PROJECT/$AR_REPO/$IMAGE_NAME:$IMAGE_TAG .
fi

docker push $AR_TYPE/$GCP_PROJECT/$AR_REPO/$IMAGE_NAME:$IMAGE_TAG
```

Pipeline Architecture

This is a high-level view of the key components in the pipeline architecture for executing data pipeline resources to both ingest, transform and load our chess data:

Ingestion Dataflow into GCS

The following flow chart illustrates how data is processed when ingesting data into GCS:

Transform/Load Dataflow into BigQuery

The following flow chart illustrates how data is processed when transforming and loading data from GCS to BigQuery:

Chess Analysis

Here are a couple of sample matplotlib charts which are analysing some player data: