**WeatherAPI Data Platform**

*Oscar, Filip och Taymaz*

Vi har skapat en dataplattform som hämtar data från Openweathermap via deras API och lagrar det i Google BigQuery, samt transformerar den för konsumtion av en given ML-modell för att göra predictions på morgondagens maxtemperatur som kan nås av en slutanvändare. Vi har även automatiserat inhämtningen av data och predictions med hjälp av Github Actions och Google Cloud Workflows, med möjlighet till effektiv logging och monitoring med hjälp av deskriptiv error-hantering, samt containerisering av funktioner med Docker eller Cloud Run Functions.

**Processbeskrivning / Verktygsval**

Datan hämtas först från openweathermap genom Python-script som laddar upp raw-datan; formaterad som en json-sträng, till en tabell i bigquery. Om tabellen inte finns skapas den. Användning av Google Cloud Platform var ett givet efter alla föreläsningar om dess olika verktyg och funktioner, samt att ha majoriteten av processen samlad på samma ställe.

Därefter bryter vi ut raw-datan till enskilda kolumner direkt i bigquery genom att skapa en Clean-view av raw-datan. Vi valde att använda oss av views istället för nya tabeller, då det inte sker stora komplexa query:s och datan är relativt simpel, vilket sparar på computation-kostnader samt lagring. Views håller sig dessutom live-uppdaterade, vilket innebär att all ny datainhämtning direkt speglas i vårt clean-lager.

Sedan körs en Cloud Run Function som hämtar datan från vår Clean-view och transformerar den med hjälp av Pandas för att anpassa kolumnerna för konsumtion av ML-modellen. De predictade värdena laddas sedan upp i en ny tabell i bigquery för att kunna hämtas av webbappen via en SQL query.

Webbappen är en python flask app som använder sig av flask.render\_template() och html-kod för att skapa ett grafiskt gränssnitt för UI/UX. Webbappen använder sig även av google clouds’ bigquery library i python för att query:a tabellen som innehåller predictions. Denna webbapp läggs då i en Docker container innan den deploy:as till GCP via github actions.

API:en som hämtar data från datakällan och ligger i en Docker container i Cloud Run samt Cloud Run Functions-funktionen som transformerar datan och använder den för predictions ligger båda i en Cloud Workflow som ser till att modellen endast gör en prediction med nyaste datan efter att API:en har hämtat ny data från openweathermap. Denna Cloud Workflow triggas av en Cloud Scheduler som har Cron-jobbet “0 \* \* \* \*” som triggas varje timme.

Anledningen vi använde Cloud Run Functions för modellen men Cloud Run för webbappen är för att modellen endast triggas 1 gång om timmen, vilket gör Cloud Run Functions mer kostnadseffektiv, medans webbappen skulle kunna få en http-request så mycket som flera gånger i minuten, framförallt om flera personer använder appen samtidigt, så då bedömde vi att Cloud Run är mer kostnadseffektiv. Vi hade även lagt API:en i en Cloud Run Function **om vi visste att det var en giltig container-metod i tid…**

Vi hämtade även in data för syrehalten i hav från SMHI:s API. Denna data hämtas med en funktion som ligger i Cloud Run Functions och som triggas varje timme av Cloud Scheduler. Datan läggs i en bigquery-tabell, som sedan “clean”as av en bigquery view. Därefter hämtas data från clean-view:en men en query av webapp-cloud-run-containern. Query:n i fråga matchar syrehalt-data till temperaturdata men en JOIN på datetime\_utc-columnen i bägge views för att se till att datan som visas representerar samma tidpunkt, för att få en verklighetsbaserad bild av relationen mellan syrehalt och temperatur.