Data-workflow STRAVA-API

Dylan Cluyse

Contents

Data-workflow STRAVA-API

Inleiding

Voor de NPE van het opleidingsonderdeel “Infrastructure Automation” zocht ik naar een vrij beschikbare, REST-driven API. Tijdens het zoeken keek ik naar mijn interesse in de sportwereld en het opkomend fenomeen dat Strava noemt. Strava is een platform gericht op sporters en atleten van alle kwaliteiten en capaciteiten.

Benodigdheden:

Er zijn drie benodigdheden bij het bouwen van een applicatie met de Strava API.

Requests sturen naar Strava vergt een zekere inspanning. Allereerst moet je je account gaan registreren als ‘developer-account’. Dit heeft noch gevolgen op je prestraties als sporter, noch op je verzamelde prestaties. Bij deze registratie moet je een korte uitleg voorzien waarover je applicatie, dat verbonden is met de Strava-API, zal gaan. Dit proces duurt hoogstens twee weken.

Eenmaal geregistreerd zal je een toegangstoken en een ververstoken krijgen. Het toegangstoken vervalt na zes uur en dit kan je enkel ophalen met de ververstoken. De ververstoken verandert niet en blijft statisch. Dit is iets waar je rekening mee moet houden bij het sturen van de GET of POST-requests. Meer info over de authenticatie vindt u hier.

Alle mogelijke requests kan je terugvinden op de documentatie van Strava. Klik hier om de documentatie te bekijken.

Mappenstructuur:

```
------------------------------------------------------------
data-workflow/
   gathered_files
   logs
```

```
        log.txt
    main.sh
    README.md
    reporting_document
    scripts
        DataAnalyser.py
        DataGatherer.sh
        DataReporting.sh
        DataTransformer.sh
    source
    transformed_data
-----------------------------------------------------
```

Het verzamelen van data.

Ik wilde voor deze opdracht statistieken weergeven van vier verschillende groepen. Hiervoor zal ik een manier van verzameling moeten inschakelen. Om deze fase tot een goed eind te brengen werk ik met curl. Alle code voor deze fase kan worden teruggevonden in `DataGathering.sh`.

Eerst moet ik een toegangssleutel ophalen. Dit doe ik met de onderstaande actie. Als resultaat krijg ik een JSON-bestand terug met daarin de nieuwe toegangssleutel. Met een pipe gebruik ik de uitvoer van de curl als invoer voor het Jq-commando. Ik haal enkel de toegangstoken op.

Toegangstoken ophalen:

```
curl -sL -X POST ${url_get_access_token} -d client_id=${client_id} \
                -d client_secret=${client_secret} -d grant_type=${grant_type} \
                -d refresh_token=${refresh_token} | \
                jq -r ".access_token" 2>> "${logfile}"
```

Dit is voorbeeld-uitvoer. Enkel de sleutel 'access_token' wordt opgehaald met Jq:

```
{
  "token_type": "Bearer",
  "access_token": "a9b723...",
  "expires_at":1568775134,
  "expires_in":20566,
  "refresh_token":"b5c569..."
}
```

Ik wil twee soorten gegevens ophalen. Ik moet werken met twee verschillende GET-requests. Beide geven JSON-uitvoer terug. Onderstaande tabel verduidelijkt het proces.

Doel	Actie met curl
Wat zijn de prestaties van mensen in een groep? Hoe staven de activiteiten van groepsleden tegenover de andere groepen? Zijn er groepen waar 'less is more' of zijn de tryhards prominenter dan verwacht?	GET /clubs/{id}/activities
Wat is de evolutie van de groepen? Wat is de wekelijkse toename of afname?	GET /clubs/{id}

De toegangssleutel moeten we in de header injecteren met optie -H. Hieronder een voorbeeld uit de documentatie:

```
curl -G https://www.strava.com/api/v3/example -H "Authorization: Bearer ReplaceWithAccessToken"
```

Het script bestaat uit twee functies:

Functie	Nut
get_access_token	De toegangstoken wordt opgehaald. De functie zal als uitvoer enkel de toegangstoken geven. Er zijn géén parameters nodig.
get_strava_club_data	Op basis van drie parameters wordt er data opgehaald: het unieke clubid, het aantal activiteiten dat moet worden opgehaald en de naam van de club. De data wordt opgeslaan in de vorm van een JSON-bestand.

Met een while-lus wordt er voor iedere groep in het CSV-bestand een curl uitgevoerd. Na iedere curl zorg ik ervoor dat alle bestanden in de verzamelde bestanden op read-only staan. * Indien er géén csv-bestand is, dan wordt er een curl uitgevoerd op vier hardcoded groepen. * Indien er géén subfolder bestaat, dan wordt er een nieuwe subfolder aangemaakt.

De mappenstructuur van `gathered_files` ziet er als volgt uit:

```
gathered_files
   ClubDesKom
       ClubDesKom_activities_20221030-174727.json
         ...
       ClubDesKom_activities_20221211-175952.json
       ClubDesKom_info_20221030-174727.json
         ...
       ClubDesKom_info_20221211-175952.json
   CyclingVlaanderen
       CyclingVlaanderen_activities_20221030-174727.json
         ...
       CyclingVlaanderen_activities_20221211-175952.json
       CyclingVlaanderen_info_20221030-174727.json
         ...
       CyclingVlaanderen_info_20221211-175952.json
   WahooJapan
       WahooJapan_activities_20221030-174727.json
         ...
       WahooJapan_activities_20221211-175952.json
       WahooJapan_info_20221030-174727.json
         ...
       WahooJapan_info_20221211-175952.json
   WahooUS
        WahooUS_activities_20221030-174727.json
         ...
        WahooUS_activities_20221211-175952.json
        WahooUS_info_20221030-174727.json
         ...
        WahooUS_info_20221211-175952.json
```

Data omzetten naar een CSV-bestand.

Iedere subfolder, dus iedere map in `gathered_files`, wordt overlopen. Binnen een subfolder, bijvoorbeeld `CyclingVlaanderen` wordt enkel de laatste 21 activities JSON-bestanden en de laatste 21 info JSON-bestanden opgehaald. Op deze manier zijn we zeker dat we informatie van de laatste zeven dagen hebben, aangezien we werken met twee verschillende tijdstippen én de mogelijkheid om dit manueel uit te voeren. De lijst wordt enkel maar langer, dus het is beter om een deel van de lijst te gebruiken, in plaats van de volledige lijst aan JSON-bestanden.

De teller is een hulpmiddel om bij te houden wanneer een header moet worden toegevoegd. Enkel bij de eerste file dat in de lus wordt opgehaald, activities of info binnen één subfolder, wordt er eerst een header toegevoegd. Hiervoor gebruik ik een enkel '>' teken, zo verwijder ik eerst alle inhoud in het CSV-bestand. Achteraf wordt alles met de meest recente info aangevuld.

Er is één functie:

Functie	Nut
loop_club_data	De functie zal met het Jq-commando de inhoud van een bestand appenden aan een CSV-bestand. De functie vereist vier parameters: de naam van de subfolder, het type (activities/info), de headers en de nodige keys.

De mappenstructuur van `transformed_files` ziet er als volgt uit:

```
transformed_files
   ClubDesKom_activities.csv
   ClubDesKom_info.csv
   CyclingVlaanderen_activities.csv
   CyclingVlaanderen_info.csv
   WahooJapan_activities.csv
   WahooJapan_info.csv
   WahooUS_activities.csv
   WahooUS_info.csv
```

Data Analyse

De data analyse voer ik uit met Python in het script `DataAnalyser.py`. Ik gebruik de volgende libraries:

Naam	Functie
Pandas	CSV-inhoud opslaan in een dataframe. Verwerking van de dataframes.
timedelta	Berekeningen op datetime-objecten uitvoeren.
Seaborn	Grafieken genereren

Ik hou drie dataframes bij: * Een dataframe met alle activiteiten van vandaag. * Een dataframe met alle activiteiten van de voorbije week. * Een dataframe met alle veranderingen per club van de voorbije week.

Voor de dagelijkse verslagen maak ik drie jointplots en vijf histogrammen. De histogrammen bevatten een aanvullend leaderboard. Voor dit leaderboard moest ik het canvas van de figure vergroten. Zo niet werd de leaderboard afgesneden. Voor het design maakte ik gebruik van de ingebouwde kleurenpaletten van Seaborn.

Data Reporting

Voor de laatste stap maak ik gebruik van een Lighttpd & PHP webserver. Uit veiligheidsredenen configureerde ik de Lighttpd-service om enkel op poort 1997 te draaien. Via port-forwarding kan ik de site via poort 80 op mijn lokale machine bereiken.

Op de webserver zijn er vier belangrijke pagina's: * Het meest recente dagelijkse rapport. * Een archief van alle gegenereerde dag- en weekverslagen in PDF-vorm. * Een slides met alle info uit het wekelijkse verslag. * De documentatie voor de applicatie.

Het aanmaken van de pdf's en de slides gebeurt binnen `DataReporting.sh`. Voor zowel de pdf's als de slides maak ik eerst een markdownbestand. De verslagen vereisen een aparte structuur. De inhoud opvullen doe ik met een lus. Dit doe ik door uitvoer van het echo commando te appenden aan een markdownbestand. Meer over de syntax van het markdownbestand vond ik terug op de documentatiesite van Pandoc.

Pandoc wordt gebruikt om de verslagen te genereren op basis van een dagelijks of wekelijks gecreëerd Markdown-bestand. Wekelijkse verslagen worden enkel op zondag gemaakt. Ik gebruik het date commando om dit te checken.

Voor de dynamische webpagina wordt er gebruik gemaakt van PHP. De timestamp wordt opgehaald door middel van het meest recente graph dat op de webserver staat. Hiervan wordt, met PHP, de creationdate opgehaald. De graphs print ik uit met een for-loop. De opmaak is statische en wordt voorzien door een apart, en zelfgemaakt, CSS-bestand.

Om slides te genereren gebruik ik "landslides". Dit is een gratis en open-source Markdown converser, in dezelfde stijl als Pandoc. Zo kan je ook PDFs genereren, maar ik kies ervoor om enkel slides te genereren. De configuratie moet aangepast worden. Bij het aanmaken verwijzen de twee CSS-bestanden en het JS-bestand naar een directory buiten het bereik van de webserver. Daarom pas ik de drie links aan met het sed-commando. Meer over Landslide vindt u hier terug.

Functie	Nut
create_fillertext	Vultekst ophalen van een site. Dit wordt achteraan het markdown-bestand toegevoegd.
create_header	De header genereren om in het markdown-bestand te plaatsen. Dit bevat de titel, auteur, layout, klasse, main- en titelfont en fontgrootte. We geven ook mee dat er een inhoudstafel moet aangemaakt worden. Dit wordt enkel gedaan voor de dagelijkse en wekelijkse verslagen.
create_markdown	Achteraan het markdown-bestand, wat bij start enkel de header bevat, wordt iedere graph en een stuk vultekst toegevoegd. Het stuk vultekst komt van de create_fillertext functie. Er zijn twee verschillende markdown-bestanden (dagelijks of wekelijks), dus het type wordt meegegeven.
create_slides_markdown	Zelfde nut als voor de create_markdown functie, maar de slides vereisen een andere lay-out. Hiervoor werd een aparte functie gemaakt. Er moeten minstens vijf liggende streepjes worden geplaatst om een nieuwe slide te maken.
pandoc_create_report	Er wordt een pandoc-document gegenereerd. De inhoud hangt af van welk bestand: daily_report.md of weekly_report.md

Functie	Nut
create_landslides_html	Er wordt een html-bestand gemaakt op basis van slides.md

De mappenstructuur van de webserver ziet er als volgt uit:

```
reporting_document
   css
      main.css
      print.css
      screen.css
      slides.css
   doc.pdf
   graphs
      daily
         leaderboard_distance.png
         leaderboard_elapsed_time.png
         leaderboard_moving_time.png
         leaderboard_speed.png
         leaderboard_total_elevation_gain.png
         regular_durablegoat.png
         regular_jointplot.png
         regular_speedgoat.png
      weekly
          week_change_members.png
          week_report_count_per_date.png
          week_report_distance_v_number.png
          week_report_speed_hist.png
          week_report_sport_types.png
   index.php
   js
      slides.js
   latestreport.php
   markdown
      daily_report.md
      Montserrat-Black.ttf
      Montserrat-Regular.ttf
      slides.md
      weekly_report.md
   reports
      index.php
      pdf
          report-2022-11-01.pdf
            ...
          report-2022-12-11.pdf
          weekly-report-2022-11-06.pdf
            ...
          weekly-report-2022-12-11.pdf
   slides.html
```

Het overkoepelend script

Ik zocht naar een manier om het `main.sh` bestand worden alle vermelde scripts achtereenvolgend uitgevoerd.

1. Het script opent met een stippellijn. Met een stippellijn wordt er aangeduid welke fase aan de gang is.
2. Eerst wordt er gecontroleerd of alle mappen en packages aanwezig zijn. Dit gebeurt met if-statements.
3. Vervolgens worden de alle script achtereenvolgens afgelopen. Voor de bash-scripts geef ik `/bin/bash` mee. Voor het Python-script geef ik `/bin/python3` mee en ik schrijf alle foutmeldingen uit naar het logbestand. In ieder bash-script wordt dit al gedaan, dus het is niet meer nodig in het overkoepelend script. Daarnaast wordt er eerst gekeken of het script-folder bestaat.
4. Het script wordt afgerond met een volle stippellijn.

Conclusie

Dit was een aangename opdracht om te maken. Ik heb mezelf op de proef gesteld qua scope en omvang van de gemaakte dataworkflow, al voel ik wel dat ik hier iets mooi heb gepresteerd. De API was een gewaagde keuze, vooral omdat dit voor mij ook nieuw terrein is. De zelfgekozen scope had zeker een effect op mijn bash-scripting vaardigheden. Zo voel ik aan dat ik hier een sterke sprong in heb gemaakt vergeleken met de start van dit semester. Hopelijk vinden jullie mijn bijdrage van 'landslide' even aangenaam als ik. Dit is zeker een opdracht dat ik, puur uit interesse, nog eens zal doen met andere vrij beschikbare API's.