Connecting Ulysses to ATLAS

Contents


```
library(Ulysses)
```

1 Introduction

One of the first steps in putting together a study is adding cohorts. In OHDSI studies we use cohorts defined using `circe`; a software that serializes a series inputs into a standardized sql query. Using `circe` allows us to build consistent definitions that we can leverage across different sets of OMOP data and behind different dbms'. For those familiar with OHDSI, building `circe` cohorts can be done using the web application ATLAS. Within ATLAS, there exists a very nice cohort designer interface. Cohort definitions built in ATLAS are required for a study, so how do you gather them into Ulysses? The purpose of this vignette is to show how users can import atlas cohorts into a Ulysses study and give a pathway as to how to generate them for the construction of a study pipeline.

2 Ulysses Structure

Before getting into pulling ATLAS assets into your study, lets quickly review how cohorts are stored in Ulysses. One of the standard folders in a Ulysses directory is the cohorts folder. In this folder there are three sub-folders: json, sql anc conceptSets/json. The json folder is meant to host `circe` cohort definitions. These a jsons exported out of ATLAS. The sql folder is meant to only host custom sql that falls outside of the `circe` domain. These occur when hacking `circe` sql to deal with cdm domains with lesser coverage or when more extensive sql is required to derive a population that cannot be captured with `circe`. The last section are conceptSets. Concept sets are groupings of OMOP concepts that identify a clinical idea of interest. You can think of them of "beefed-up" code-lists. There are nice heirarchical properties that allow you to cover many terms in a vocabulary in a concept set. These concept sets also have a `circe` class and are stored as json objects. We often use concept sets for characterization or building other cohort definitions. Concept set jsons can be stored in cohorts/conceptSets/json.

Ulysses standardizes the directory structure of a study to make it easier to pull files within tasks. By storing all cohort and concept set assets in a consistent location, it makes it easier to be organized and build automation around this structure.

3 Set up connection with ATLAS

Before you import you need to set up a connection to webApi. We do this using keyring. Use the following code to do this:

```
setAtlasCredentials(keyringName = "atlas", keyringPassword = "ohdsi")

baseUrl <- keyring::key_get(service = "baseUrl", keyring = "atlas")
```

4 Importing from ATLAS

Ulysses offers some helper functions that allow users to import assets into the study directory. These helper functions will evolve over time and offer better services such as searches by tags and regular expressions. Ulysses builds wrappers around a software called `ROhdsiWebApi` which helps interface with the ATLAS api.

4.1 Cohorts

To import a cohort, a user must know the cohort ids they want to pull from ATLAS. Each cohort has a an ATLAS ID attached to it as a reference number. Before importing, collect the cohort ids you want for the study. Once these are collected you can adapt the following block:

```
importAtlasCohorts(cohortIds = c(1, 4, 5))
```

This function will pull the cohort json for each cohort from the web api and save tem to the cohorts/json folder within Ulysses. The files are saved as a json under this schematic {atlasId}_{cohortName}, where the cohort names are coerced into snakecase.

4.2 Concept Sets

To import concept sets, users must know the concept set ids they want to pull from ATLAS. Similar to the cohorts, collect the concept set ids you want to use for the the study. Use this block to import concept sets:

```
importAtlasConceptSets(conceptSettIds = c(2, 10, 14))
```

5 Using ATLAS assets within Ulysses

Once you have imported the desired ATLAS assets into a Ulysses directory, you can now use them for different task. For instance to build `circe` cohorts, users can generate using `CohortGenerator`. With a standard directory structure, it is easy to prep `circe` json for generation. See an example below:

```
# ulysses path
ulysses_path <- here::here()

# path to cohorts folder
path_to_circe <- fs::path(ulysses_path, "cohorts/json")

# list all circe json files in folder
list_of_cohorts <- path_to_circe |>
  fs::dir_ls(type = "file")

# get the cohort names
cohortName <- list_of_cohorts |>
  fs::path_file() |>
  tools::file_path_sans_ext()
```

```r
# set the ids, note this ranks by numeric atlas id
cohortId <- cohortName |>
  stringr::str_rank(numeric = TRUE)

# import json into R
circeJson <- purrr::map_chr(
  list_of_cohorts,
  ~readr::read_file(.x)
)

# serialize json to sql
circeSql <- purrr::map_chr(
  circeJson,
  ~CirceR::buildCohortQuery(
    CirceR::cohortExpressionFromJson(.x),
    CirceR::createGenerateOptions(generateStats = TRUE))
)

# make input table for cohort generator
cohortsToCreate <- tibble::tibble(
  cohortId = cohortId,
  cohortName = cohortName,
  json = circeJson,
  sql = circeSql
)

# Generate the cohorts
# place connection info appropriately
CohortGenerator::generateCohortSet(
  connectionDetails = connectionDetails,
  cdmDatabaseSchema = cdmDatabaseSchema,
  tempEmulationSchema = tempEmulationSchema,
  cohortDatabaseSchema = workDatabaseSchema,
  cohortTableNames = CohortGenerator::getCohortTableNames(cohortTable),
  cohortDefinitionSet = cohortsToCreate,
  incremental = TRUE,
  incrementalFolder = fs::path(here::here("exec/logs"))
)
```