Making the world's decentralised data more accessible.

Lab Exercise Guide

Table of Contents

Introduction

In this lab, students will have the opportunity to become familiar with SubQuery with some hands-on experience creating a simple Hello World SubQuery project. This project will use the subql CLI to create an empty project shell, and then code will be provided to query the Polkadot mainnet for the blockheight. A Docker environment will be used to run this example for simplicity.

Pre-requisites

You will require the following:

- NPM package manager
- SubQuery CLI (@subql/cli)
- Docker

Package manager

Run the following command in your terminal to install the latest version of node. Node v12 or higher is required.

```
brew update
brew install node
node -v
v18.2.0
```

SubQuery CLI

Install the latest version of the subql cli:

```
npm install -g @subql/cli
subql -v
@subql/cli/1.0.1 darwin-x64 node-v18.2.0
```

Docker

Please visit https://docs.docker.com/get-docker/ for instructions on how to install Docker for your specific operating system.

Exercise 1: Hello World

High level steps

1. Initialise a project
2. Update your mappings
3. Update your manifest file
4. Update your graphql schema file
5. Generate your code
6. Build your code
7. Deploy your code in Docker

Detailed steps

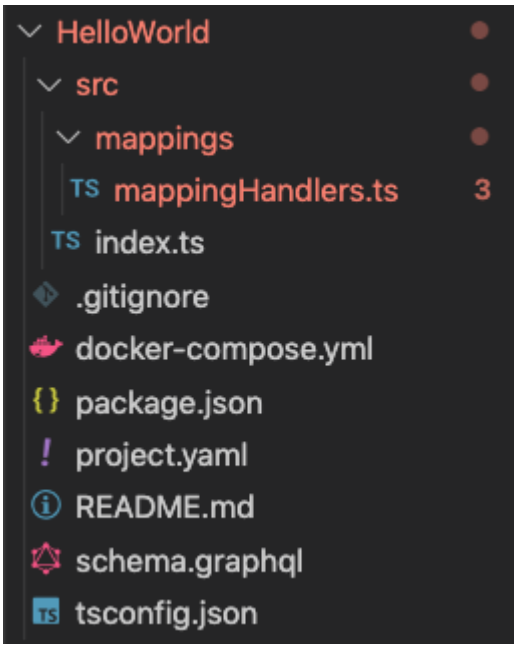
Step 1: Initialise your project

The first step in creating a SubQuery project is to create a project with the following command:

```
$ subql init
Project name [subql-starter]: HelloWorld
? Select a network family Substrate
? Select a network Polkadot
? Select a template project subql-starter     Starter project for
subquery
RPC endpoint: [wss://polkadot.api.onfinality.io/public-ws]:
Git repository [https://github.com/subquery/subql-starter]:
Fetching network genesis hash... done
Author [Ian He & Jay Ji]: Sean
Description [This project can be use as a starting po...]:
Version [1.0.0]:
License [MIT]:
Preparing project... done
HelloWorld is ready
```

Note that any text in the square brackets are the default values that will be used if nothing is provided.

This creates a framework and the following directory structure saving you time.



Step 2: Update the mappings file

The initialisation command pre-creates a sample mappings file with 3 functions, handleBlock, handleEvent and handleCall in src/mappings/mappingHandlers.ts. For this exercise we will focus on the first function called handleBlock so delete the remaining functions. The mappingHandler.ts file should look like this:

```
import {SubstrateExtrinsic,SubstrateEvent,SubstrateBlock} from
"@subql/types";
import {StarterEntity} from "../types";
import {Balance} from "@polkadot/types/interfaces";


export async function handleBlock(block: SubstrateBlock): Promise<void>
{
    //Create a new starterEntity with ID using block hash
    let record = new StarterEntity(block.block.header.hash.toString());
    //Record block number
    record.field1 = block.block.header.number.toNumber();
    await record.save();
}
```

Step 3: Update the manifest file (aka project.yaml)

The initialisation command also pre-creates a sample manifest file and defines 3 handlers. Because we have removed handleEvent and handleCall from the mappings file, we have to remove them from the manifest file as well.

The manifest file should look like this:

```
specVersion: 1.0.0
name: HelloWorld
version: 1.0.0
runner:
  node:
    name: '@subql/node'
    version: '>=1.0.0'
  query:
    name: '@subql/query'
    version: '*'
description: >-
  This project can be use as a starting point for developing your
SubQuery project
repository: 'https://github.com/subquery/subql-starter'
schema:
  file: ./schema.graphql
network:
  chainId:
'0x91b171bb158e2d3848fa23a9f1c25182fb8e20313b2c1eb49219da7a70ce90c3'
  endpoint: 'wss://polkadot.api.onfinality.io/public-ws'
  dictionary:
'https://api.subquery.network/sq/subquery/polkadot-dictionary'
dataSources:
  - kind: substrate/Runtime
    startBlock: 1
    mapping:
      file: ./dist/index.js
      handlers:
        - handler: handleBlock
          kind: substrate/BlockHandler
```

Step 4: Update the graphql schema

The default schema.graphql file contains 5 fields. We can remove fields 2 through to 5 because the handleBlock function in the mappings file only uses “field1”.

> Extra: Rename field1 to something more meaningful. Eg blockHeight. Note that if you do this, don’t forget to update the reference to field1 in the mappings file appropriately.

The schema file should look like this:

```
type StarterEntity @entity {
  id: ID! #id is a required field
  blockHeight: Int!
}
```

Step 5: Install the dependencies

Install the node dependencies by running the following commands:

```
yarn install
```

OR

```
npm install
```

Step 6: Generate the associated typescript

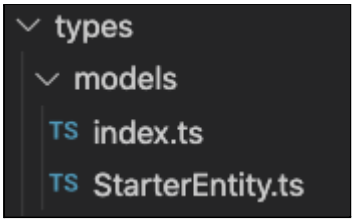
Next, we will generate the associated typescript with the following command:

```
yarn codegen
```

OR

```
npm run-script codegen
```

You should see a new folder appear with 2 new files.



Step 7: Build the project

The next step is to build the project with the following command:

```
yarn build
```

OR

```
npm run-script build
```

This bundles the app into static files for production.

Step 8: Start the Docker container

Run the docker command to pull the images and to start the container.

```
docker-compose pull && docker-compose up
```

> Note: You need to have Docker installed as noted in the prerequisite for this to work.

Step 9: Run a query

Once the docker container is up and running, which could take a few minutes, open up your browser and navigate to www.localhost:3000.

This will open up a “playground” where you can create your query. Copy the example below.

```
{
  query{
    starterEntities(last:10, orderBy: FIELD1_ASC){
      nodes{
        field1
      }
    }
  }
}
```

> Note: If you renamed field1 something else, modify this query appropriately.