



**Making the world's
decentralised data more
accessible.**



Lab Exercise Guide

Table of Contents

Introduction	2
Pre-requisites	2
Package manager	2
SubQuery CLI	2
Docker	2
Exercise 1: Hello World	3
High level steps	3
Detailed steps	3
Step 1: Initialize your project	3
Step 2: Update the mappings file	4
Step 3: Update the manifest file (aka project.yaml)	4
Step 4: Update the graphql schema	5
Step 5: Install the dependencies	5
Step 6: Generate the associated typescript	5
Step 7: Build the project	6
Step 8: Start the Docker container	6
Step 9: Run a query	6
Exercise 2: Publishing to SubQuery Projects	7
Prerequisites	7
High level steps	7
Detailed steps	7
Step 1: Log into SubQuery Projects	7
Step 2: Create a new project	9
Step 3: Deploy to the Staging Slot	11
Step 4: View on SubQuery Explorer	13

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:

```
brew update  
brew install node
```

SubQuery CLI

```
npm install -g @subql/cli
```

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

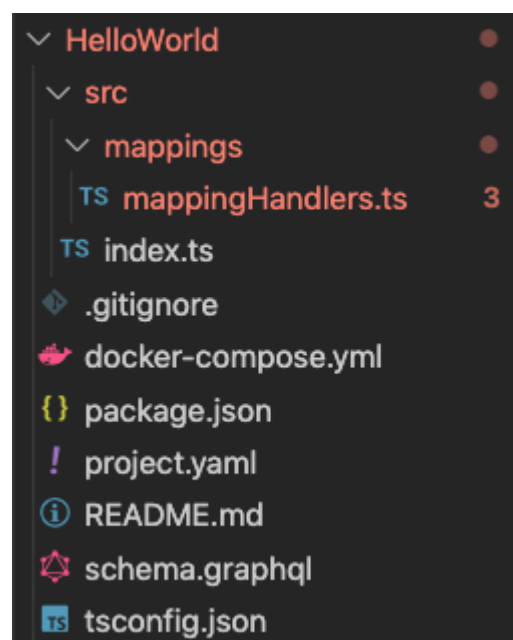
Step 1: Initialize your project

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

```
~/Code/subQuery$ subql init
Project name [subql-starter]: HelloWorld
Git repository:
RPC endpoint [wss://polkadot.api.onfinality.io/public-ws]:
Authors: sa
Description:
Version: [1.0.0]:
License: [Apache-2.0]:
Init the starter package... HelloWorld is ready
```

All the fields are optional except “author”. Also, 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. 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: 0.0.1
description: ''
repository: ''
schema: ./schema.graphql
network:
  endpoint: wss://polkadot.api.onfinality.io/public-ws
  dictionary:
https://api.subquery.network/sq/subquery/dictionary-polkadot
dataSources:
  - name: main
    kind: substrate/Runtime
    startBlock: 1
    mapping:
      handlers:
        - handler: handleBlock
          kind: substrate/BlockHandler
```

Step 4: Update the graphql schema

The default schema.graphql file will contain 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

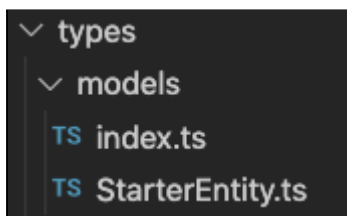
Next, we will generate the associated typescript with the following command:

```
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

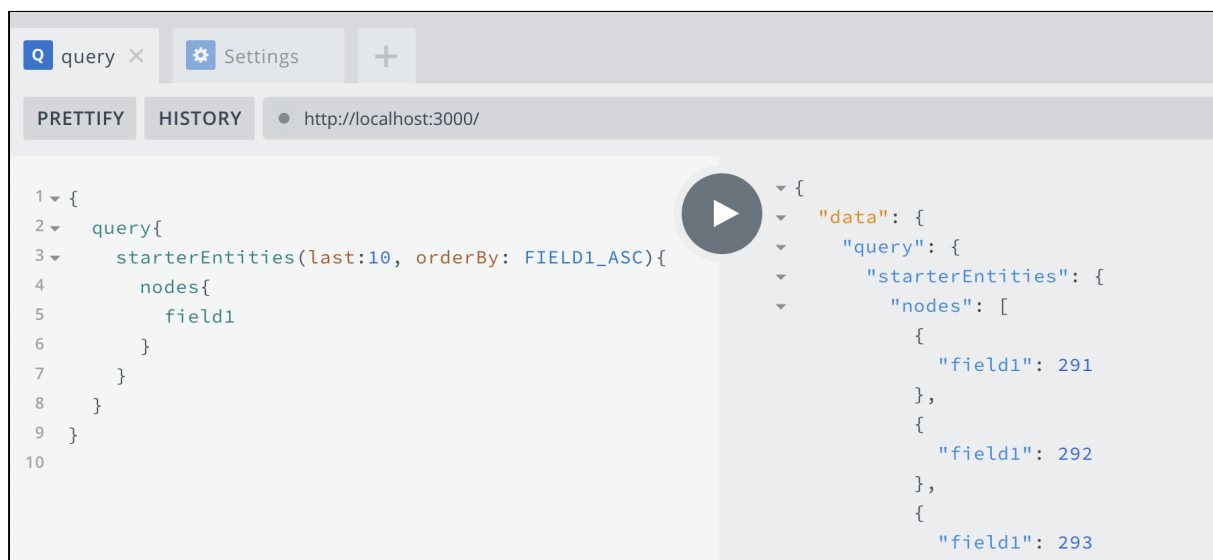
Run the docker command to pull the images and to start the container.

```
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.

Exercise 2: Publishing to SubQuery Projects

In the previous exercise, we ran our SubQuery project locally in a docker container. SubQuery also provides hosted infrastructure where users can publish their projects free of charge. In this exercise, we will demonstrate how to do this.

Prerequisites

You will need to have a GitHub account and have already created a repository and pushed your Hello World code in exercise 1 to this repository. This is because SubQuery Projects will read a designated repository and then automatically build and run the code.

This guide assumes you have a Github account and are familiar with creating repositories and committing code to that repository.

Note: If you did not manage to complete exercise 1 from above, you can clone the completed Hello World repository at <https://github.com/subquery/subql-helloworld> and then push this to your GitHub repository to continue this exercise.

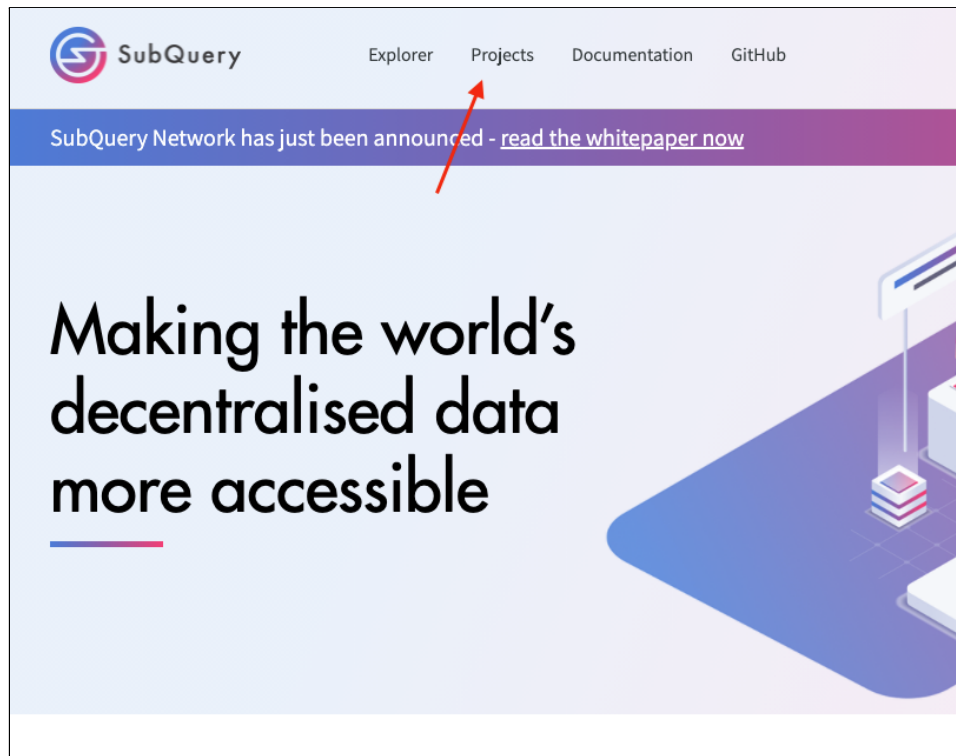
High level steps

1. Log into SubQuery Projects via GitHub
2. Create a new project
3. Deploy to the staging slot
4. View on SubQuery Explorer
5. Run query in the playground

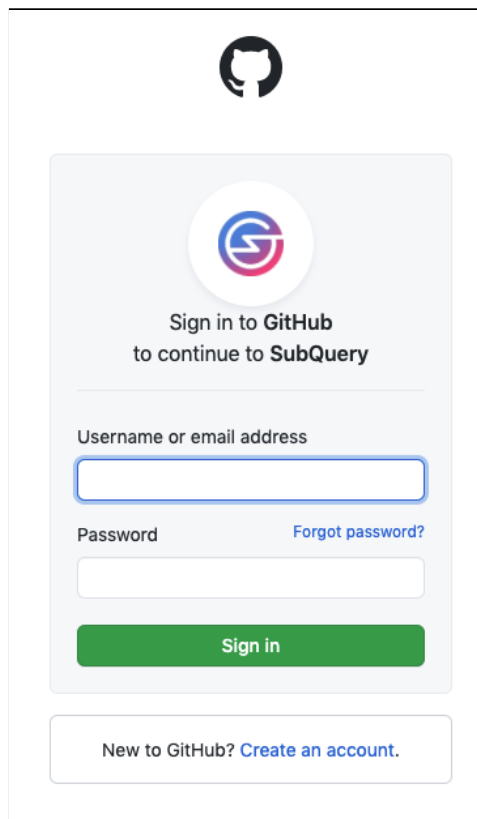
Detailed steps


Step 1: Log into SubQuery Projects


Open the SubQuery home page at <https://www.subquery.network/> and click on the Projects link at the top navigation bar. Alternatively, visit: <https://project.subquery.network/>



You will need your GitHub credentials to Single Sign On (SSO).







Sign in to **GitHub**
to continue to **SubQuery**

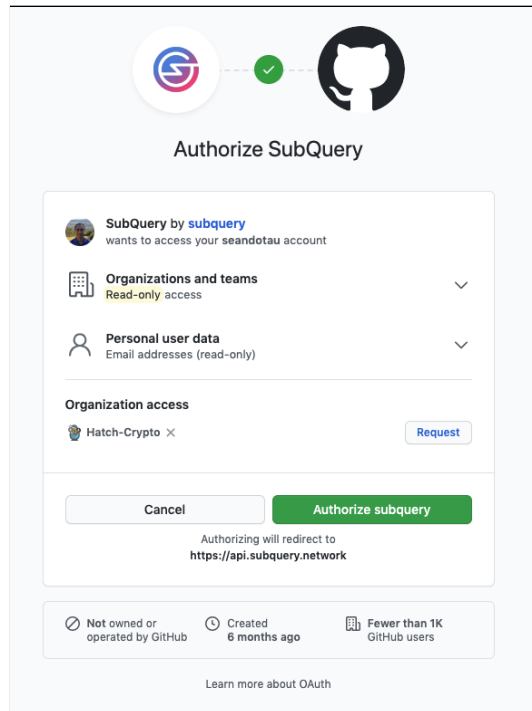
Username or email address

Password [Forgot password?](#)

Sign in

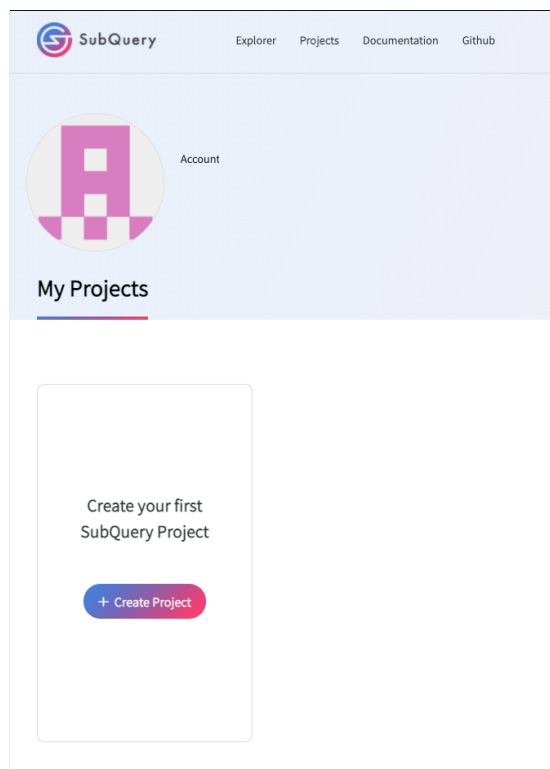
New to GitHub? [Create an account.](#)

You will then need to authorise SubQuery to access your GitHub account.

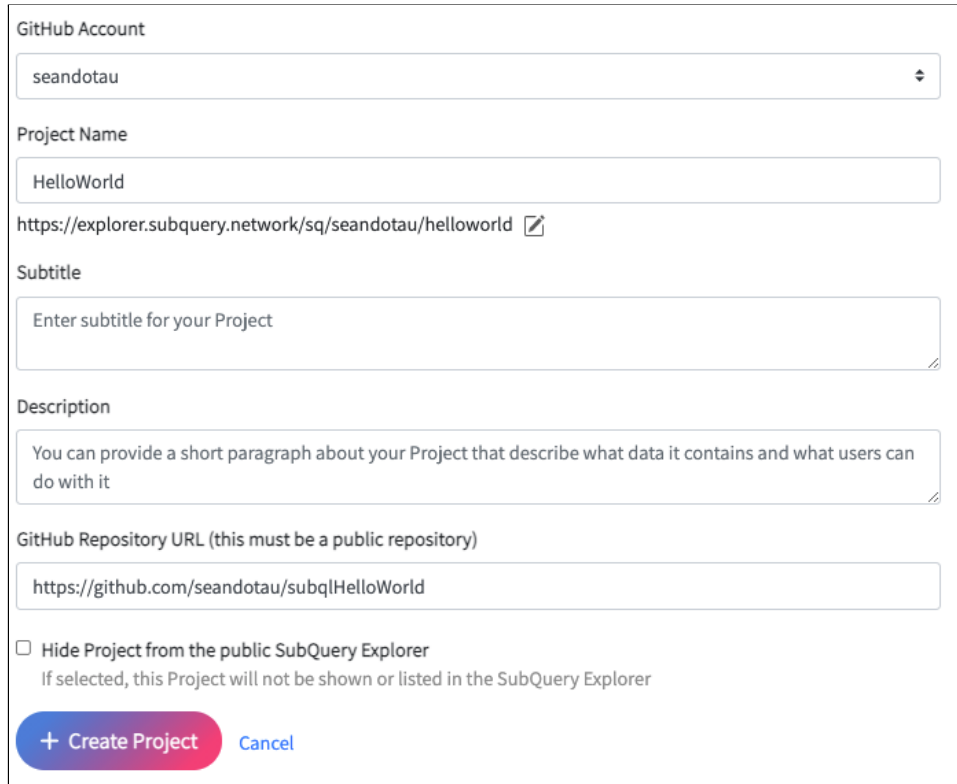


Step 2: Create a new project

Select "Create Project" and fill in all the required fields. Note: The mandatory fields will be the Project Name and GitHub Repository URL.



Below is an example of the completed form.

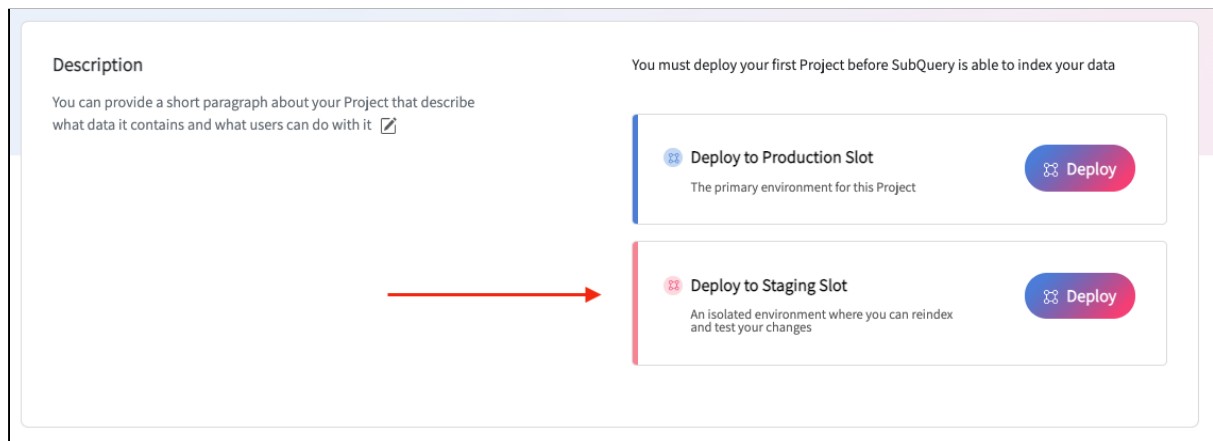


The screenshot shows a form for creating a new SubQuery project. It includes the following fields and options:

- GitHub Account:** A dropdown menu with "seandotau" selected.
- Project Name:** A text input field containing "HelloWorld".
- URL:** A text input field showing the generated URL: `https://explorer.subquery.network/sq/seandotau/helloworld` with a copy icon.
- Subtitle:** A text input field with the placeholder "Enter subtitle for your Project".
- Description:** A text area with the placeholder "You can provide a short paragraph about your Project that describe what data it contains and what users can do with it".
- GitHub Repository URL (this must be a public repository):** A text input field containing `https://github.com/seandotau/subqlHelloWorld`.
- Hide Project from the public SubQuery Explorer:** An unchecked checkbox with the text "If selected, this Project will not be shown or listed in the SubQuery Explorer".
- Buttons:** A blue "+ Create Project" button and a grey "Cancel" button.

Step 3: Deploy to the Staging Slot

Next, we have two deployment options. We can either deploy our SubQuery project to a production environment or a staging environment. We will first deploy to the staging slot by clicking deploy.




The screenshot shows the deployment options for a SubQuery project. It includes the following elements:

- Description:** A text area with the placeholder "You can provide a short paragraph about your Project that describe what data it contains and what users can do with it".
- Message:** "You must deploy your first Project before SubQuery is able to index your data".
- Deployment Options:** Two buttons with "Deploy" labels and a gear icon:
 - Deploy to Production Slot:** "The primary environment for this Project".
 - Deploy to Staging Slot:** "An isolated environment where you can reindex and test your changes".
- Red Arrow:** A red arrow points from the "Deploy to Staging Slot" button to the "Deploy" button.

Several fields that can be overridden but we'll leave these empty for now. We can also choose a particular GitHub commit but again, we'll accept the default for now. Select "Deploy Update"

Deploy to Staging Slot

Commit Hash of new Version



First commit

390a2af85755e986018ada0f93022cdf2b070c06

seandotau authored a month ago

Override Network endpoint (optional)

This defaults to the definition in project.yaml

Override Dictionary endpoint (optional)

This defaults to the definition in project.yaml

Indexer Version (@subql/node)

v0.19.1


Query Version (@subql/query)

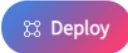
v0.6.0

Deploy Update

This will then deploy the code and a status of processing will appear. It will take a few minutes before the status changes to “running”.

Deployment Details

 **Deploy to Production Slot**
The primary environment for this Project



Staging Slot


VER: 390a2af85755e986018ada0f93022cdf2b070c06


PROCESSING ...

0%

Indexer Version (@subql/node)	onfinality/subql-node:v0.19.1	
Query Version (@subql/query)	onfinality/subql-query:v0.6.0	
Query Endpoint	https://api.subquery.network/sq/seandotau/helloworld__c2Vhb	
Network Endpoint	<default>	
Dict Endpoint	<default>	

Deployment Details

 **Deploy to Production Slot**
The primary environment for this Project





Staging Slot
VER: 390a2af85755e986018ada0f93022cdf2b070c06

RUNNING

...


0.02%


Indexer Version (@subql/node)	onfinality/subql-node:v0.19.1	
Query Version (@subql/query)	onfinality/subql-query:v0.6.0	
Query Endpoint	https://api.subquery.network/sq/seandota	
Network Endpoint	<default>	
Dict Endpoint	<default>	

Step 4: View on SubQuery Explorer

Once your project is running, select the 3 ellipsis and click on “View on SubQuery Explorer”. Note the other options available as well such as deleting the project, deploying a new version, and promoting to production.

Deployment Details

 **Deploy to Production Slot**
The primary environment for this Project



Staging Slot
VER: 390a2af85755e986018ada0f93022cdf2b070c06

RUNNING

...

0.04%

Indexer Version (@subql/node)	onfinality,	
Query Version (@subql/query)	onfinality,	
Query Endpoint	https://ap	
Network Endpoint	<default>	
Dict Endpoint	<default>	

Promote to Production

Deploy New Version

View on SubQuery Explorer

Delete

Step 5: Run a query

In the playground, run the following GraphQL query to test that your project works.

```
{
  query{
    starterEntities(last:10, orderBy: BLOCK_HEIGHT_ASC){
      nodes{
        blockHeight
      }
    }
  }
}
```

Playground

Log

▶ History

1 {

2 query{

3 starterEntities(last:10, orderBy: BLOCK_HEIGHT_ASC){

4 nodes{

5 blockHeight

6 }

7 }

8 }

9 }

10 }

▼ {

▼ "data":{

▼ "query":{

▼ "starterEntities":{

▼ "nodes":[

{

"blockHeight": 4969

},

{

"blockHeight": 4970

},

{

"blockHeight": 4971

},

}

Note: The model answer uses the field “blockHeight” instead of field1.