COMPASS Documentation

Release 0.5

Marco Moretto

Sep 27, 2022

Table of Contents

COMPASS (**COM**pendia **P**rogrammatic **A**ccess **S**upport **S**oftware) is a software layer that provides a GraphQL endpoint to query compendia built using COMMAND>_ technology. COMPASS is meant to be the barebone interface on top of a compendium database and it is usually preferable to query the database using the Python package pyCOMPASS or the R package rCOMPASS for data analysis purposes.

Note: VESPUCCI, the *Vitis vinifera* cross-platform expression compendium, is accessible using its COMPASS frontend. Check out the use cases!

CHAPTER 1

Quick start

1.1 About GraphQL

GraphQL is an open-source data query and manipulation language for APIs, and a runtime for fulfilling queries with existing data [...] It provides an efficient, powerful and flexible approach to developing web APIs, and has been compared and contrasted with REST and other web service architectures. It allows clients to define the structure of the data required, and exactly the same structure of the data is returned from the server[1]

1.2 Simple query using the GraphiQL client

GraphiQL is a powerful GraphQL web client we are going to use in this documentation to show how to query a COMPASS GraphQL endpoint. The GraphiQL interface is the default way to query COMPASS and it is accessible at 'http://vespucci.fmach.it/compass'.

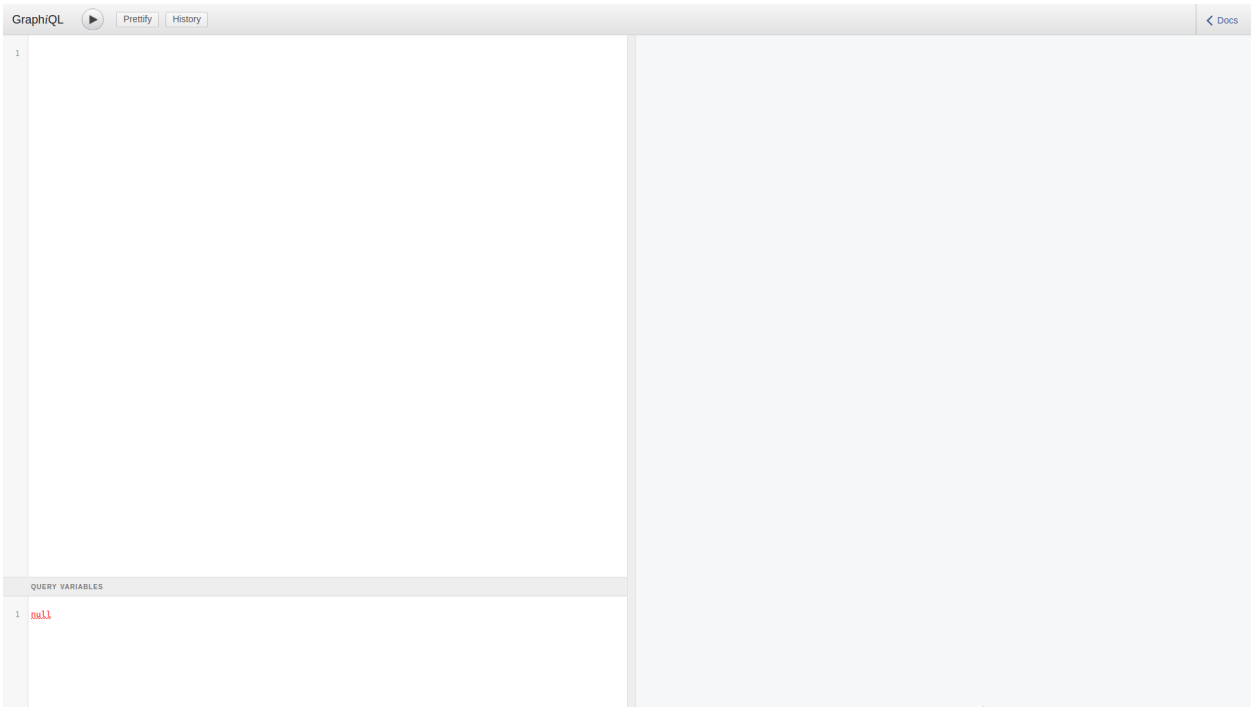
Note: A simple Python script using the requests package would work just fine, but maybe in that case you might want to have a look at the pyCOMPASS package.

```
import requests

url = 'http://vespucci.fmach.it/compass'
query = '''
  {
            compendia {
                  name,
                  fullName,
                  description,
                  versions {
                    versionAlias,
                    versionNumber,
```

(continues on next page)

[1] Wikipedia GraphQL



(continued from previous page)

```
                    databases {
                          name,
                          normalizations
                    }
                }
            }
        }
'''
request = requests.post('http://vespucci.fmach.it/compass', json={'query': query})
print(request.json())
```

CHAPTER 2

How To

This section contains several examples of GraphQL queries in order to perform main operations on a COMPASS GraphQL endpoint. For each query there might be additional parameters. **An easy way to have the list of all available parameters is to use the GraphiQL client** and use the autocompletion (ALT + SPACEBAR).

2.1 GET available compendia

```
{
  compendia {
        name,
        fullName,
        description,
        defaultVersion
        versions {
          versionNumber,
          versionAlias,
          defaultDatabase,
          databases {
                name,
                normalizations
          }
        }
  }
}
```

2.2 GET compendium data sources

```
{
  dataSources(compendium: "vespucci") {
    edges {
```

(continues on next page)

(continued from previous page)

```
      node {
        id,
        sourceName
      }
    }
  }
}
```

2.3 GET platform information

```
{
  platforms(compendium:"vespucci") {
    edges {
      node {
        platformAccessId,
        platformName,
        description,
        dataSource {
          sourceName
        },
        platformType {
          name
        }
      }
    }
  }
}
```

2.4 GET platform types

```
{
  platformTypes(compendium:"vespucci") {
    edges {
      node {
        id,
        name,
        description
      }
    }
  }
}
```

2.5 GET experiments information

```
{
  experiments(compendium:"vespucci") {
    edges {
      node {
```

(continues on next page)

(continued from previous page)

```
        organism,
        experimentAccessId,
        experimentName
      }
    }
  }
}
```

2.6 GET sample annotation

```
{
  sampleAnnotations(compendium: "vespucci", first: 10) {
    edges {
      node {
        sample {
          id,
          sampleName
        },
        annotation
      }
    }
  }
}
```

Note: The returned annotation is the JSON-LD format of RDF annotation.

2.7 GET biological feature annotations

```
{
  biofeatureAnnotations(compendium: "vespucci", bioFeature_Name: "VIT_00s0332g00060")
↪{
        edges {
          node {
                annotation
          }
        }
  }
}
```

Note: The returned annotation is the JSON-LD format of RDF annotation.

2.8 GET ontology structure

```
{
 ontology(compendium:"vespucci", name:"Gene ontology") {
  edges {
    node {
      structure
    }
  }
 }
}
```

Note: The returned structure is a JSON created using the networkx Python package

2.9 GET samples annotation triples

```
{
  annotationPrettyPrint(compendium:"vespucci", ids:"QmlvRmVhdHVyZVR5cGU6MQ==") {
        rdfTriples
  }
}
```

2.10 GET sample via SPARQL query

```
{
  sparql(compendium:"vespucci", target:"sample", query:"SELECT ?s ?p ?o WHERE { ?s
↪<http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://purl.obolibrary.org/obo/
↪NCIT_C19157> . ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://purl.
↪obolibrary.org/obo/PO_0009010>}") {
        rdfTriples
  }
}
```

2.11 GET samples by experiment access id

```
{
 samples(compendium:"vespucci", experiment_ExperimentAccessId:"GSE54347") {
  edges {
    node {
      sampleName,
      description
    }
  }
 }
}
```

2.12 GET sample by access id

```
{
 samples(compendium:"vespucci", sampleName_Icontains:"GSM1313535") {
  edges {
    node {
      sampleName,
      description
    }
  }
 }
}
```

2.13 GET sample set by name

```
{
 sampleSets(compendium:"vespucci", name:"GSE27180_48hours-1-vs-GSE27180_0h-2") {
  edges {
    node {
      id,
      name
    }
  }
 }
}
```

2.14 GET sample set by sample id

```
{
 sampleSets(compendium:"vespucci", samples:["U2FtcGxlVHlwZTox"]) {
  edges {
    node {
      id,
      name
    }
  }
 }
}
```

2.15 GET sample raw data

```
{
 rawData(compendium: "vespucci", sampleId: "U2FtcGxlVHlwZTox") {
  values
  valueTypes
  biofeatureReporters
  biofeatures {
   edges {
```

(continues on next page)

(continued from previous page)

```
    node {
     id
    }
   }
  }
 }
}
```

2.16 GET biological feature by name

```
{
  biofeatures(compendium:"vespucci", name:"VIT_00s0332g00060") {
    edges {
      node {
        name,
        biofeaturevaluesSet(bioFeatureField_Name:"sequence") {
          edges {
            node {
              value
            }
          }
        }
      }
    }
  }
}
```

2.17 GET biological feature annotation triples

```
{
  annotationPrettyPrint(compendium:"vespucci", ids:"QmlvRmVhdHVyZVR5cGU6Mg==") {
        rdfTriples
  }
}
```

2.18 CREATE MODULE with biological features and sample sets

```
{
  modules(compendium: "vespucci", version:"legacy", biofeaturesIds: [
↪"QmlvRmVhdHVyZVR5cGU6MQ==","QmlvRmVhdHVyZVR5cGU6Mg=="], samplesetIds: [
↪"U2FtcGxlU2V0VHlwZToxMjYw", "U2FtcGxlU2V0VHlwZToxMjYx", "U2FtcGxlU2V0VHlwZToxMjYy
↪"]) {
        normalizedValues
        sampleSets {
          edges {
                node {
                  id
                  name
```

(continues on next page)

(continued from previous page)

```
                    normalizationdesignsampleSet {
                          edges {
                            node {
                                  sample {
                                    sampleName
                                  }
                            }
                          }
                    }
                  }
            }
          }
          biofeatures {
            edges {
                  node {
                    id
                    name
                  }
            }
          }
  }
}
```

2.19 GET available PLOT methods

```
{
  plotName(compendium:"vespucci") {
        distribution,
        heatmap,
        network
  }
}
```

2.20 PLOT DISTRIBUTION of coexpressed sample sets GIVEN biological features

```
{
  plotDistribution(compendium:"vespucci", version:"legacy", plotType:"sample_sets_
↪coexpression_distribution", biofeaturesIds: ["QmlvRmVhdHVyZVR5cGU6MQ==",
↪"QmlvRmVhdHVyZVR5cGU6Mg=="]) {
        html
  }
}
```

2.21 PLOT HEATMAP GIVEN a module

```
{
  plotHeatmap(compendium: "vespucci", version: "legacy", plotType: "module_heatmap_
↪expression", biofeaturesIds: ["QmlvRmVhdHVyZVR5cGU6MQ==", "QmlvRmVhdHVyZVR5cGU6Mg==
↪", "QmlvRmVhdHVyZVR5cGU6Mw==", "QmlvRmVhdHVyZVR5cGU6NA==", "QmlvRmVhdHVyZVR5cGU6NQ==
↪"], samplesetIds: ["U2FtcGxlU2V0VHlwZToxNDg=", "U2FtcGxlU2V0VHlwZToxNDk=",
↪"U2FtcGxlU2V0VHlwZToxNTA=", "U2FtcGxlU2V0VHlwZToxNTE=", "U2FtcGxlU2V0VHlwZToxNTI=
↪"]) {
        html
  }
}
```

2.22 PLOT NETWORK of coexpression GIVEN a module

```
{
  plotNetwork(compendium: "vespucci", version: "legacy", plotType: "module_
↪coexpression_network", biofeaturesIds: ["QmlvRmVhdHVyZVR5cGU6MQ==",
↪"QmlvRmVhdHVyZVR5cGU6Mg==", "QmlvRmVhdHVyZVR5cGU6Mw==", "QmlvRmVhdHVyZVR5cGU6NA==",
↪"QmlvRmVhdHVyZVR5cGU6NQ=="], samplesetIds: ["U2FtcGxlU2V0VHlwZToxNDg=",
↪"U2FtcGxlU2V0VHlwZToxNDk=", "U2FtcGxlU2V0VHlwZToxNTA=", "U2FtcGxlU2V0VHlwZToxNTE=",
↪"U2FtcGxlU2V0VHlwZToxNTI="]) {
        html
  }
}
```

2.23 GET THE RANKING methods for sample sets and biological features

```
{
  scoreRankMethods(compendium:"vespucci") {
        sampleSets,
        biologicalFeatures
  }
}
```

2.24 GET THE available NORMALIZATION methods for a compendium version

```
{
  normalizations(compendium:"vespucci", version:"latest") {
        edges {
          node {
                name,
                date
          }
        }
  }
}
```

2.25 RANKS sample sets or biological features GIVEN biological features or sample sets respectively

```
{
  ranking(compendium:"vespucci", version:"legacy", rankTarget:"samplesets", rank:
↪"magnitude", biofeaturesIds: ["QmlvRmVhdHVyZVR5cGU6MQ==", "QmlvRmVhdHVyZVR5cGU6Mg==
↪", "QmlvRmVhdHVyZVR5cGU6Mw==", "QmlvRmVhdHVyZVR5cGU6NA==", "QmlvRmVhdHVyZVR5cGU6NQ==
↪"]) {
        id,
        name,
        type,
        value
  }
}
```

CHAPTER 3

Deploy COMPASS

COMPASS is a complex application and relies on several other software components to work. In order to ease up the deployment process a `docker-compose.yml` file is provided, so assuming you have a working Docker Compose environment, the deployment process will be a matter of running a few commands.

In case you want to manually deploy COMPASS in your environment there will be more steps you will need to take care of such as installing the web-server, the DBMS, etc.

3.1 Requirements

Have a look at the `requirements.txt` file for details. COMMAND>_ main dependencies are:

- Python 3
- Django
- PostgreSQL
- Numpy
- Pandas

3.2 Docker Compose

Assuming that you have Docker Compose correctly installed, you should be able to perform the following steps:

```
# 1. clone the repository
git clone https://github.com/marcomoretto/compass.git

# 2. build
docker-compose build

# 3. start docker
```

(continues on next page)

(continued from previous page)

```
docker-compose up -d

# 4. create database schema
docker-compose exec web python manage.py migrate
```

That's it! You should be able to point your browser to http://localhost/graphql and see the GraphiQL interface.

3.3 Manual Deploy

One easy way to understand what you need to do to manually deploy COMPASS is to have a look at 2 files:

- the Dockerfile
- the docker-compose.yml file

In a nutshell, after having installed and configured Nginx (or another web-server to run Django applications), PostgreSQL

```
pip3 install --upgrade pip
pip3 install -r requirements.txt
```

Now you should be ready configure Django (check the documentation for details), create the database schema and run the application.

```
python manage.py migrate
```

Note: COMPASS id a Django application so refer to the Django docs for database configuration https://docs.djangoproject.com/en/1.11/ref/settings/

CHAPTER 4

Contribute & Support

Use the GitHub Push Request and/or Issue Tracker.

CHAPTER 5

Author

To send me an e-mail about anything else related to COMPASS write to marco.moretto@fmach.it

CHAPTER 6

License

The project is licensed under the GPLv3 license.