**Mac set up :** [Introduction · macOS Setup Guide (sourabhbajaj.com)](https://sourabhbajaj.com/mac-setup/)

**Introduction**

*Ocean Breeze* is the evolution of *Frontiers Ocean*. *Frontiers Ocean* was born as a common point to share datasets, and to provide good response times when dealing with multiple datasets that in origin can come from various sources.

*Frontiers Ocean* started from scratch, adding different datasets and users, allowing quick exploration of the data. Based on the success of the platform, and the lessons learnt from the first year of use, *Ocean Breeze* came to improve the user experience, and as the company and the project grew so much during last year, also new organizational changes needed to be made.

*Ocean Breeze* has a lot of things in common with *Frontiers Ocean*, but it provides new functionalities and organization possibilities that *Frontiers Ocean* didn't have.

**What is Ocean Breeze?**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/introduction/#what-is-ocean-breeze)

*Ocean Breeze* is a Serverless Data Lakehouse, where you can explore, combine, model and visualize information coming from multiple sources inside and outside Frontiers universe.

*Ocean Breeze* is a platform meant to facilitate the access to different datasets, work with them, and share if necessary, having user and group profiles in order to adapt the experience of the user depending on its needs.

**What can I find inside Ocean Breeze?**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/introduction/#what-can-i-find-inside-ocean-breeze)

There is a variety of official datasets, having the possibility of blending them and generating insights easily with a basic knowledge of SQL Language.

Datasets inside *Ocean Breeze* can come from internal or external sources, and they might have several stages, so we can find the information in different shapes and flavours, depending on our needs.

**Use cases**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/introduction/#use-cases)

There are a lot of use cases where *Ocean Breeze* can help, and we are still discovering new ways of helping Frontiers with the information within *Ocean Breeze*. Some examples are:

* Data Warehousing & Lakehousing
* Analytics
* Streaming data
* Business Intelligence
* Data Engineering
* Data Science
* Machine Learning

# BigQuery

BigQuery is the technology selected to provide access to the datasets, and perform the queries on them. It is a fully managed, serverless data warehouse from Google, available in Google Cloud Platform, that enables scalable analysis over big datasets without the need of any effort in managing the platform or the jobs.

Queries on BigQuery are done using ANSI SQL, and it has some machine learning capabilities. The easy access to cross dataset queries and the scalability are key points in order to select it as an enterprise level cloud data warehouse.

*Frontiers Ocean* was based in BigQuery, as it allowed quick growth and availability of the datasets for use, and *Ocean Breeze* is still using it as the main technology, adding some governance and organization over the datasets.

In this documentation you can find the [Learning Path](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/first-steps/) in order to help in the onboarding of newcomers in Google Cloud Platform, or BigQuery itself.

# AnalyticsHub

AnalyticsHub is a product within Google Cloud Platform. AnalyticsHub is a way to provide access to datasets, and distribute them, either privately or publicly. The main advantage is that the original dataset doesn't need to be copied, but it becomes linked in the destination project.

Users can browse the AnalyticsHub just like a maketplace of data, and then decide which datasets they want to add to their projects. This is a huge step forward in the way *Ocean Breeze* will share information, as in *Frontiers Ocean* information had to be shared in general projects, having to deal all the time with the full list of shared datasets.

Now with AnalyticsHub, and some project restructuring, they can have a personal space in which to add or remove the datasets they are interested in, and go to the marketplace to see the list of datasets provided by the company.

This helps in reducing the cost, allowing new organization structures and also better control on the distribution, as well as a very much clear user experience.

*Ocean Breeze* relies on the AnalyticsHub as the main point for sharing datasets within BigQuery, and changes project structure so that every team has its own space, adding also the possibility of having private datasets, and many more advantages that we will cover in this documentation.

**Why Ocean Breeze?**

INFO

The aim of this section is to provide insight into the evolution of Frontiers Ocean and Ocean Breeze, as well as to highlight their differences. If you are not familiar with Frontiers Ocean, you can skip this page.

Ocean Breeze was initially created to offer users within the Frontiers universe a means of accessing information. Over time, as the platform grew in size and popularity, new datasets and users emerged, prompting the identification of more effective ways to provide and access information, with a focus on enhancing user experiences.

**What is different from Frontiers Ocean?**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/why-breeze/#what-is-different-from-frontiers-ocean)

In practice, Frontiers Ocean and Ocean Breeze share the same underlying concept. Anyone who has used Frontiers Ocean in the past is already familiar with how to use Ocean Breeze. The main difference lies in how information is presented and accessed.

Ocean Breeze is a new interface for interacting with BigQuery that builds upon the previous experience with Frontiers Ocean. It introduces greater separation between groups of users, along with a new set of roles and governance possibilities. The data exposure has been modified, such that each group can have better access to the data that is relevant to their specific use cases. Users can subscribe to the datasets they find useful, streamlining the interface and enabling a more focused and user-oriented experience.

**Key changes in Ocean Breeze**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/why-breeze/#key-changes-in-ocean-breeze)

Groups of users (teams) have their own space, where they can explore, add data, create self service datasets and have a clear view of the datasets they are using, cleaning the interface.

Datasets are now populated through the analytics hub, instead of being shared in a common proyject. This allows to have a marketplace of data to explore, with quick access documentation for each dataset, improving user experience.

There is a new set of dataset stages, defined in this documentation, and clearly exposed in the analytics hub, so that every time a user is exploring the dataset, he can have a clear view of the stage the data he is looking at is, and what use cases it would fit better.

* Datasets are now versioned, having a new analytics hub dataset when the schema changes, allowing the users to identify the changes and avoid errors when connecting automated tools to the data.
* New documentation including contextual explanation for each dataset, as well as detailed information for all the versions of a dataset, schemas and potential relationships with other datasets.
* Dataset freshness and sanity is now populated in the new documentation, and in the direct access from the dataset description on BigQuery, helping the user to identify when the data is safe to be used.

**How to start**

First of all, you will need to request access to Ocean Breeze. Team Braavos is the team in charge of managing Ocean Breeze, so just open a ticket using this form and they will manage your access.

**Projects in GCP**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/how-to-start/#projects-in-gcp)

Everything within Ocean Breeze takes place inside a project. Your group or team should have a project (or a list of them), and if you did the first step you will probably already have it.

The project is the place where your team or group share the experience. Your project will have a clear view on the list of datasets your group or team decide to import, and if you can anyway search and import any of the official datasets into your project, as well as creating private datasets for your own use.

FOR OCEAN USERS

In Frontiers Ocean, there was only an official project that held all the datasets, and all users had the vision of all datasets all the time, no matter if they were going to use them or not.

In Ocean Breeze you are able to select what datasets you use for your activity and what you don't, and also to create private datasets within your project, having a playground space that won't be visible or accessible for other teams.

When you feel that your data is ready to be shared with the rest of the company, there is a way to share it and make the dataset become officially provided by Ocean Breeze in the market place (Analytics Hub).

## Learning path[​](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/how-to-start/#learning-path) (https://scribehow.com/)

We have created a [learning path](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/first-steps/) for new users that will help you in your first steps with *Ocean Breeze*. Take your time to visit it and getting familiar with the interface and project organization.

Once you have finished your learning path feel free to start searching for data and explore all the possibilities you have to use the information present in *Ocean Breeze*.

Access

**API**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/first-steps/#api)

Sometimes, it would be necessary access to GCP through different tools. In that case, you can always use the API.

You have to install the client library through a terminal, with the command:

pip install --upgrade google-cloud-bigquery

Install the gcloud CLI and allow it to run the command:

 gcloud init

In the CLI, it is possible to create the Application Default Credentials:

gcloud auth application-default login

Now, you should be able to connect to the API in Jupyter (or any python, for example, interface):

from google.cloud import bigquery  
client = bigquery.Client()  
query = """  
SELECT \* FROM …  
"""  
query\_job = client.query(query)  
results = query\_job.result()  
results\_df = results.to\_dataframe()

**Basic notions about costs and usage**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/querying/#basic-notions-about-costs-and-usage)

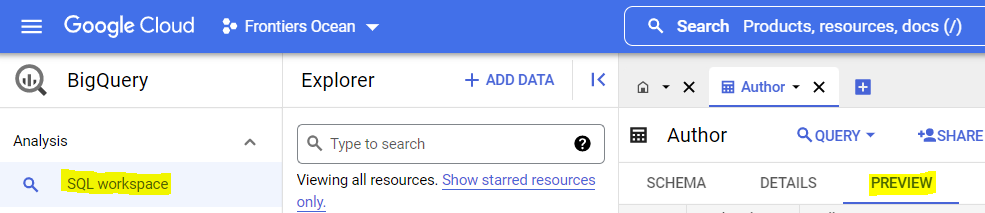
BigQuery pricing has two main components:

* **Storage** pricing: the cost to store data in BigQuery.
* **Analysis** pricing: the cost to process queries.

We use an on-demand pricing model so we are charged for the number of bytes processed by each query, so being aware of what we are doing when we query data is fundamental.

**Avoid SELECT ALL**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/querying/#avoid-select-all)

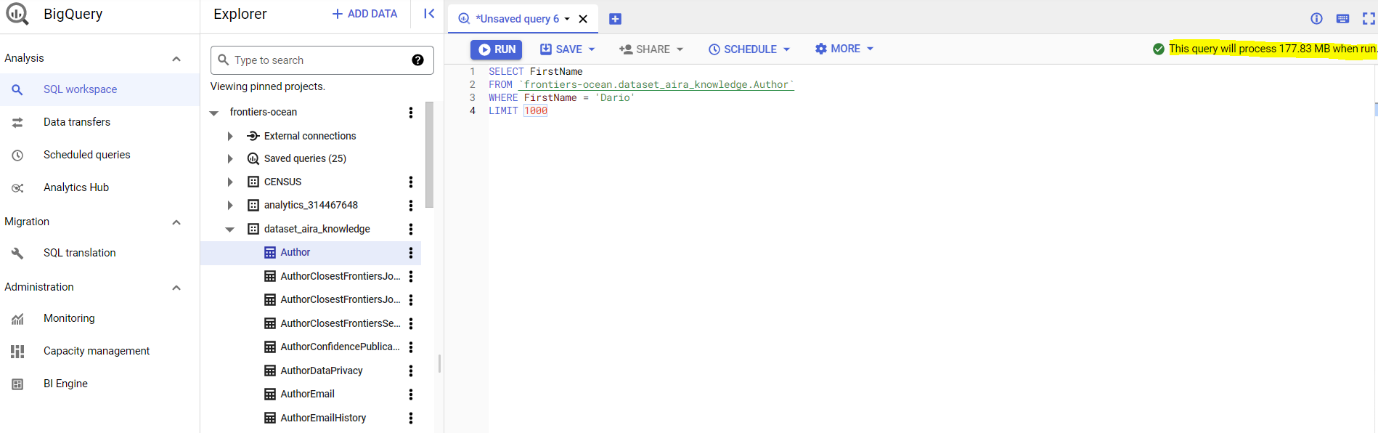
BigQuery uses a columnar data structure, it analyzes tables by column, for this reason, it is very important put in the SELECT section of your query only the field that you need. It is NOT recommended to use SELECT \* ( or SELECT ALL ) because the full table is scanned in this way. If you need to explore data, don't run queries but the best option is to use the "Preview" function available when you click on a table.



Otherwise, you could use the TABLESAMPLE to obtain only a random subset of the dataset. This result are not cached and if you re run the same query, you might get a different result subset each time.

**Using Query Validator**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/querying/#using-query-validator)

You can check how much data are you retrieving through the query validator in SQL Workbench. When you compose your query, if the query is valid, the dimension of data processed is shown in the upper-right corner.



**Using LIMIT**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/querying/#using-limit)

The LIMIT clause affects only the final query output and not about the amount of data read by BigQuery, so there is always the risk of scanning and billing the entire table.

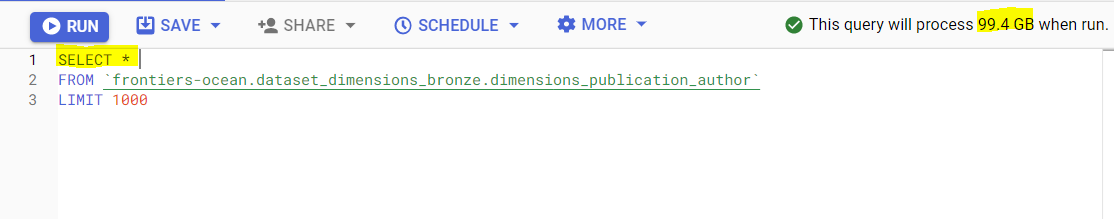
However, a LIMIT clause could be useful to reduce the number of bytes scanned with a clustered table, due to how data are organized in this type of table. So it is recommended to use the LIMIT clause but not to control costs.

**Materialized Views**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/querying/#materialized-views)

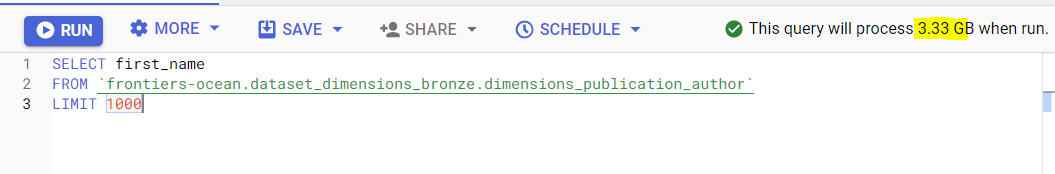
In some cases, it might be helpful to use Materialized Views, you can check the last section of LP4 to see how to create them.

**Cost optimization examples**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/querying/#cost-optimization-examples)

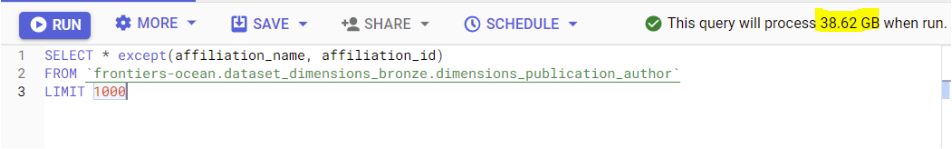
If you use "SELECT \*", the query will process 99.4 GB how it is shown in the query validator.



But if you select only the columns of your interest, you can reduce the result size.



Or also if you exclude some columns by the result set, thanks to the operator, you can obtain some advantages.



**Writing data and creating new datasets**

**Where I can create my data?**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/writing/#where-i-can-create-my-data)

CAUTION

Only some users can edit and create new resources according to their responsibility and necessity. In Frontiers-Ocean most of the content is read-only, if you need specific requirements, please contact the data owner for more details.

If you need an your own project contact the Bravoos Team.

BigQuery organizes data into a logical hierarchical structure. The data is collected in Tables inside Datasets that belong to different Projects.

This structure helps us to organize information logically. You can use various datasets to separate tables, views, and processes according to their analytical domain and with a different scope thanks to different projects.

A Dataset must have a unique name and the geographic location cannot be changed after creation. All tables used in a query must be stored in the same location.

A Table or a View must belong to a dataset, so you need to create at least one dataset. Each Table name must be unique per dataset.

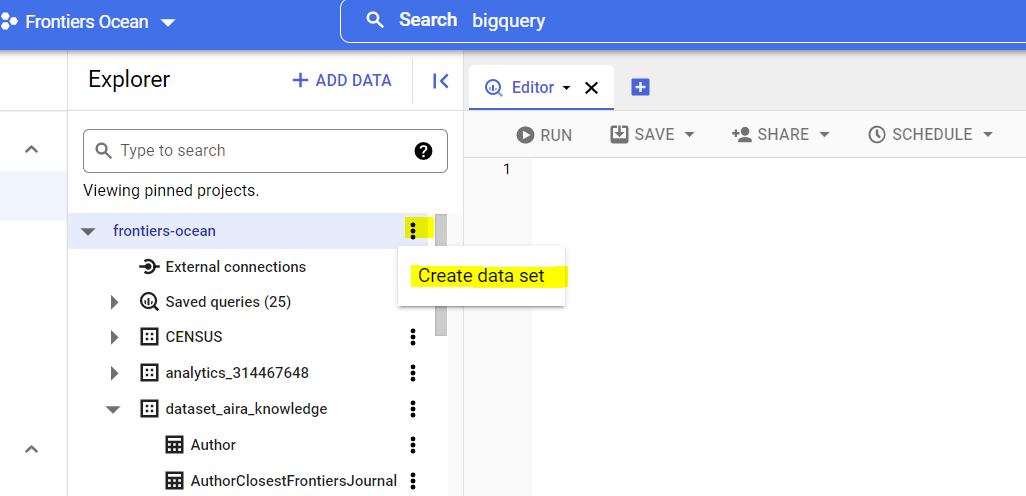
You can create a dataset or a table in several ways:

* By using the Google Cloud Console UI
* By using the *bq* command line tool
* By using the API method
* By using SQL statement
* Other

**Dataset**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/writing/#dataset)

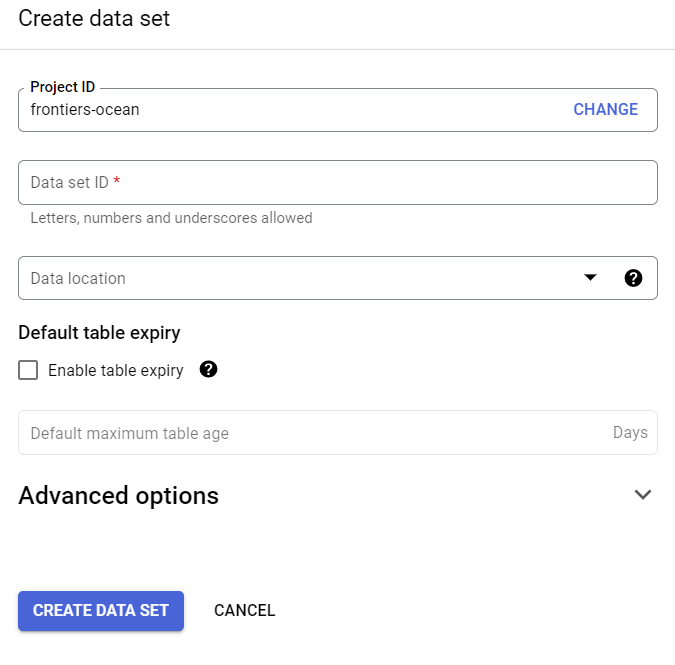
**Console**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/writing/#console)

From the Bigquery page, it is possible to navigate to the *Explorer panel* and select the project where you want to create the new dataset. Then click on the three vertical dots near the name of the project and *Create dataset*.



In the next pane, fill all the fields according to your needs except for *Data location*. All our datasets must be in the data center within European Union, so it is necessary to set the location to the multizone *eu*.

Dataset Id must be unique for the entire project.



**bq command line**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/writing/#bq-command-line)

From your CLI, you can create a new dataset using *bq mk* with many flags and the option --dataset or the shortcut -d.

bq --location=LOCATION mk --dataset \  
 --default\_partition\_expiration=PARTITION\_EXPIRATION \  
 --default\_table\_expiration=TABLE\_EXPIRATION \  
 --description="DESCRIPTION" \  
 PROJECT\_ID:DATASET\_ID

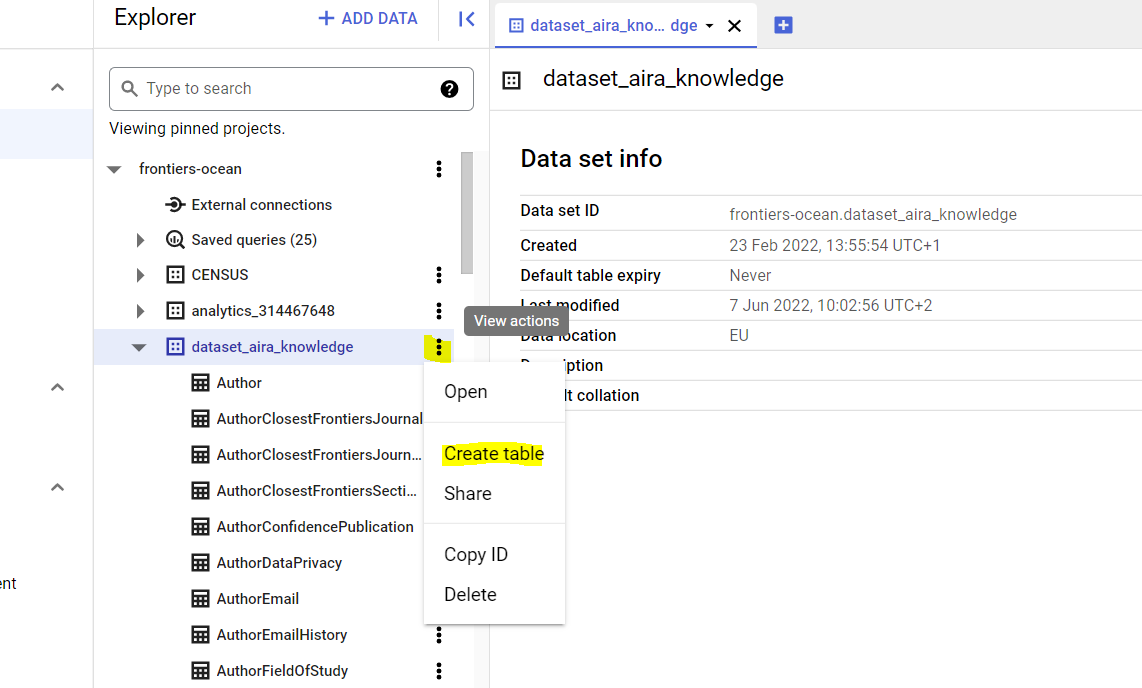
You have to replace the placeholder with your information:

* LOCATION: define the dataset's location which cannot be changed after the creation.
* PARTITION\_EXPIRATION: the default lifetime in seconds for partitions in newly created partitioned tables. The default value has no minimum value.
* TABLE\_EXPIRATION: the default lifetime in seconds for newly created tables. The default value is the minimum which is 3600 seconds (one hour).
* DESCRIPTION: a text about the dataset.
* PROJECT\_ID: where you want to create your dataset.
* DATASET\_ID: the id you want to set on the new dataset. For more options and flags, you can check out this [link](https://cloud.google.com/bigquery/docs/datasets#bq).

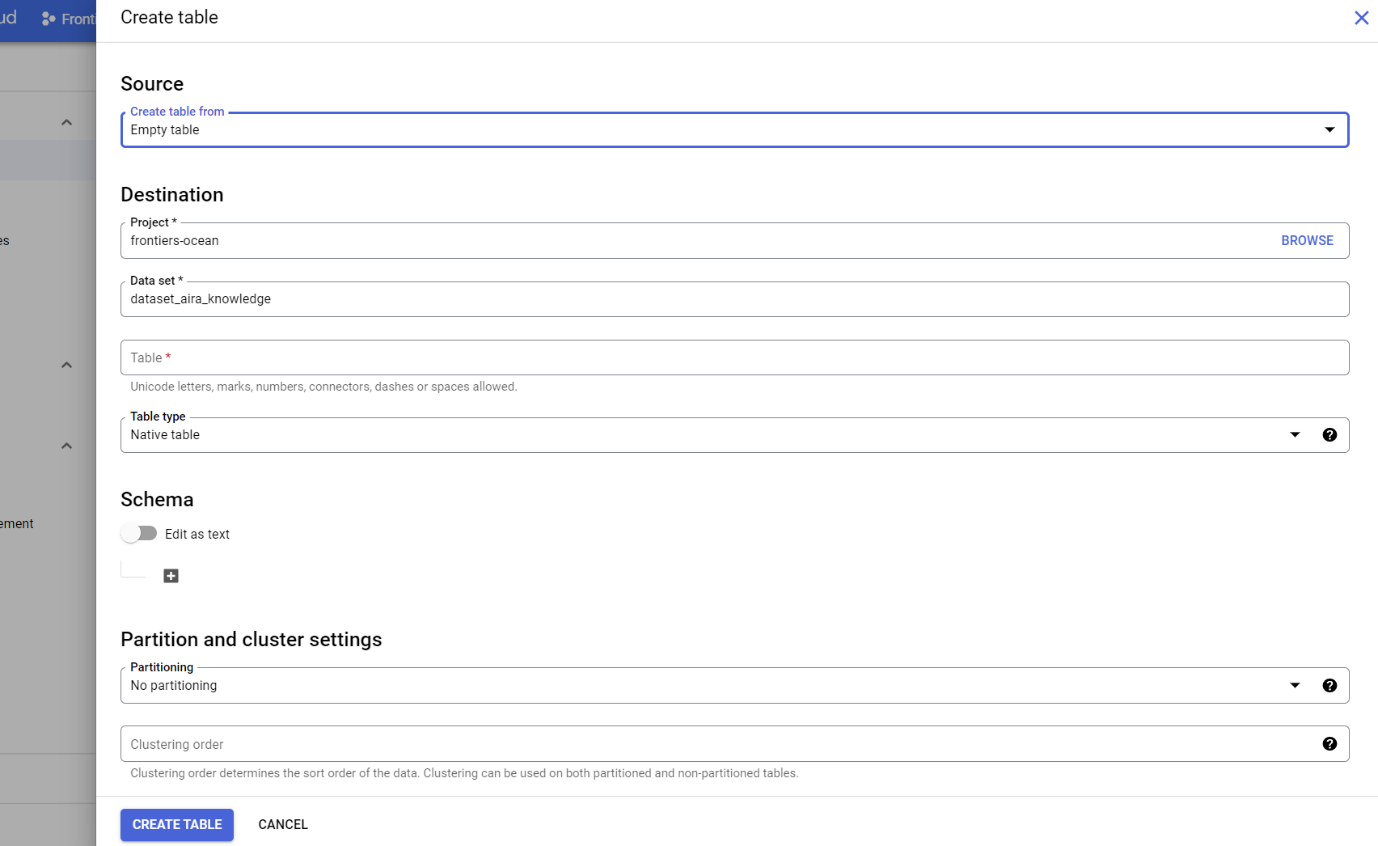
**Table**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/writing/#table)

**Console**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/writing/#console-1)

From the *Explorer pane* on the BigQuery page, after the selection of a dataset, it is possible to create a new table using the three vertical dots option.



In the next pane, fill all the fields according to your needs. The new table, with a unique name, can be empty or you can choose various options as a source of data. According to the Source, the Schema can be automatically detected or inserted manually. When it is possible, creating a partitioned and clustered table can be useful to improve query performance.



**bq command line**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/writing/#bq-command-line-1)

A table can be created also with the *bq mk* command with the option --table or the shortcut --t.

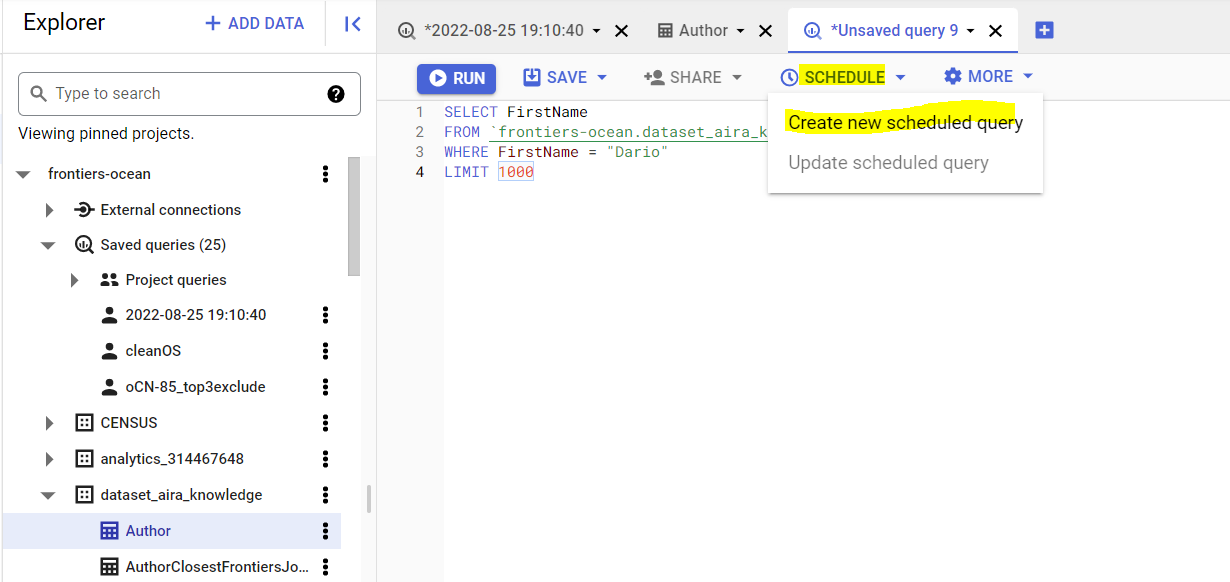
bq mk --table \  
--expiration EXPIRATION\  
--description "DESCRIPTION"\  
PROJECT\_ID:DATASET\_ID.TABLE\  
SCHEMA  
EXPIRATION: the default lifetime in seconds for the table. If you set this value, the default value which is the dataset's value is ignored. The minimum is 3600 seconds (one hour).

* DESCRIPTION: a text about the dataset.
* PROJECT\_ID: where you want to create your table.
* DATASET\_ID: a dataset in your project.
* TABLE: the name of the new table.
* SCHEMA: is an inline schema definition in the format *field:data\_type, field:datatype* or the path to the JSON schema file.

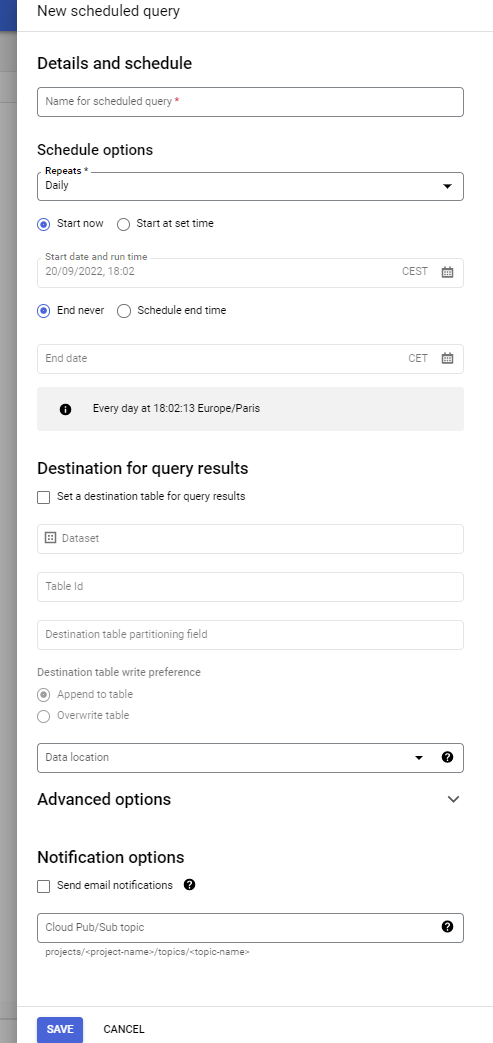
**Scheduled queries**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/writing/#scheduled-queries)

Big Query allows running queries automatically based on a recurring basis. The query results can be organized by date and time by parameterizing both the query string and the destination table.

In the Editor of the *Explorer pane,* you can compose your query and then on the upper menu, there is the *Schedule* button.

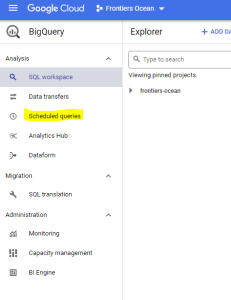


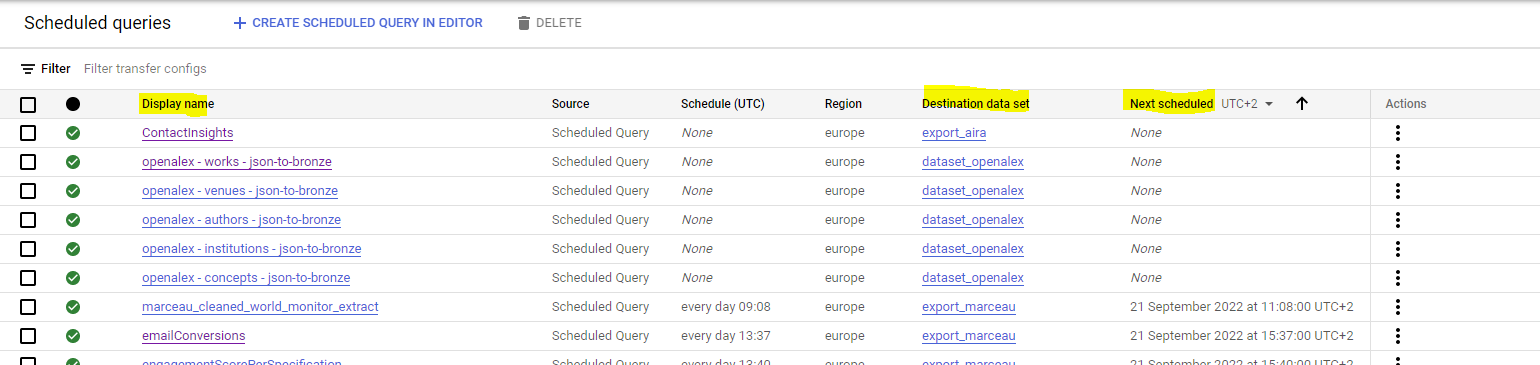
The scheduled query options open in the New scheduled query pane. You must insert a name and you can set the schedule or select the *On-demand* option in the *Repeats* menu.



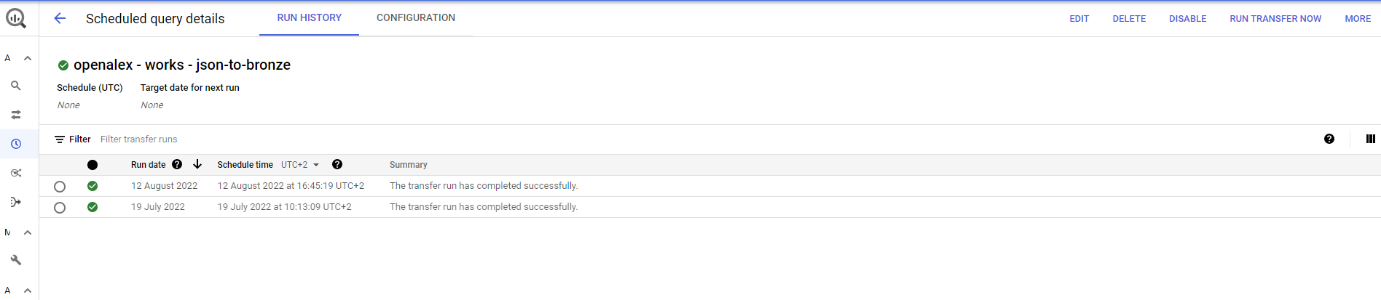
The results can be saved into a destination table that can be created with this specific purpose or also an already existing one. It is important to set the write preference to avoid unwanted behavior.

After that, using the *Scheduled Queries* button in the navigation left menu, it is possible to check important information like the destination dataset and general and next schedule.

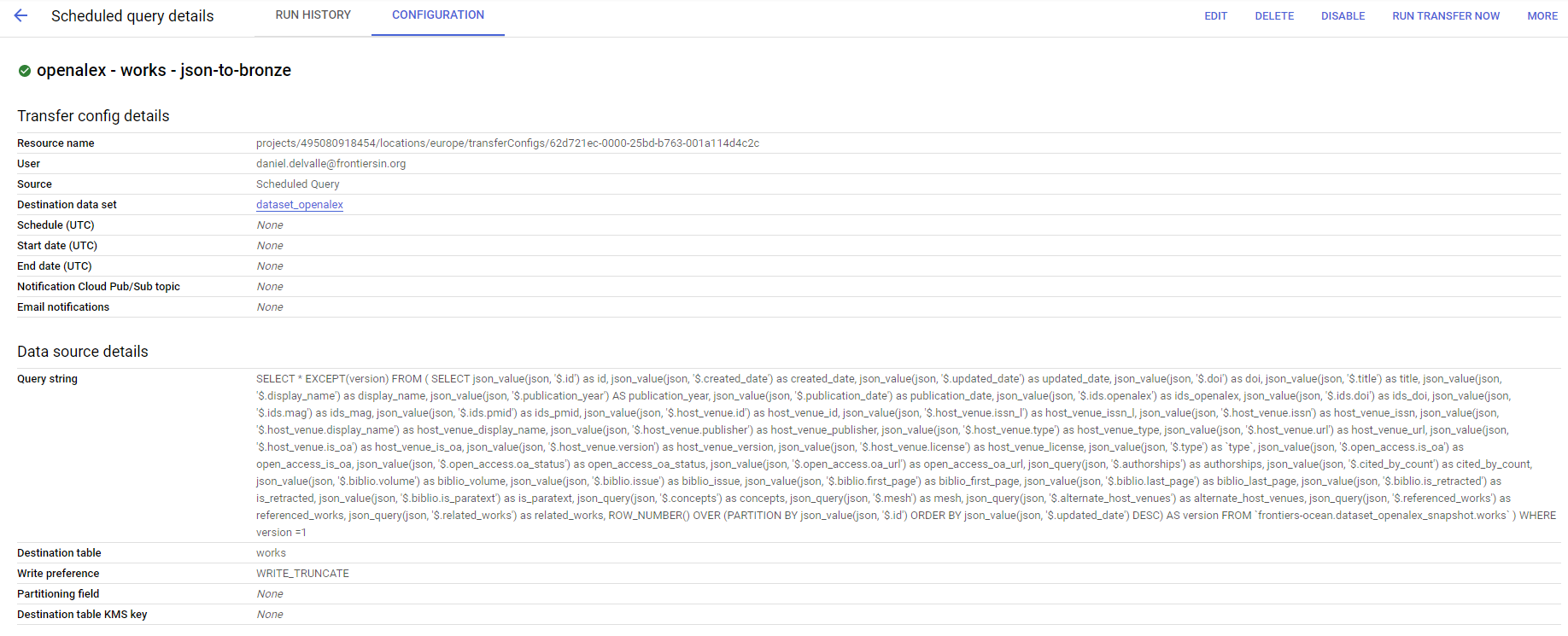




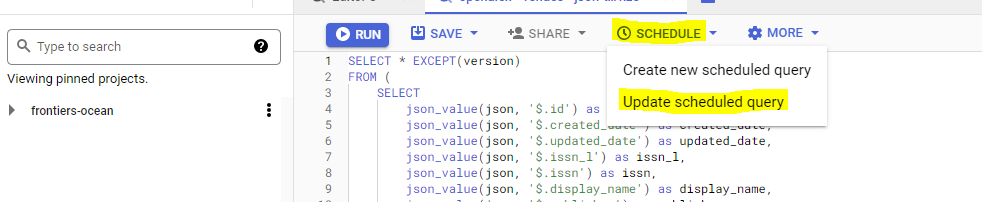
More information is available if you click on the *Display name*. You can see the *Run History* of one specific scheduled query



Or also his Configuration. In the upper right corner, there are some useful buttons to use to modify/delete/disable the configuration, or also *Run Transfer Now* can be used to execute the query immediately.

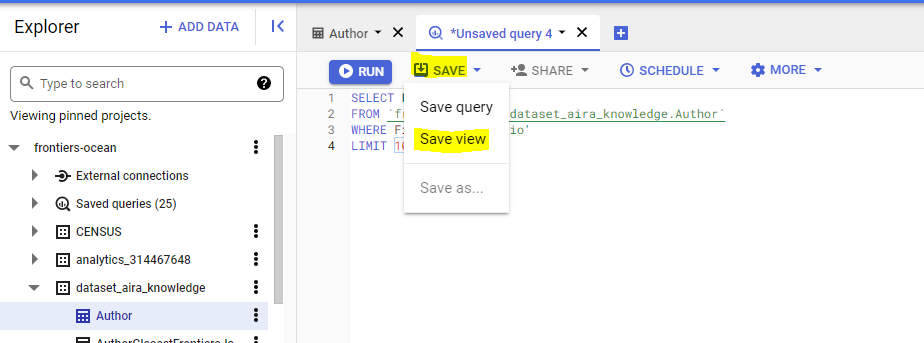


The *Edit* button can be used to update the Query string in the BigQuery Editor. You can save your change using also *Schedule* button and then the *Update scheduled query*

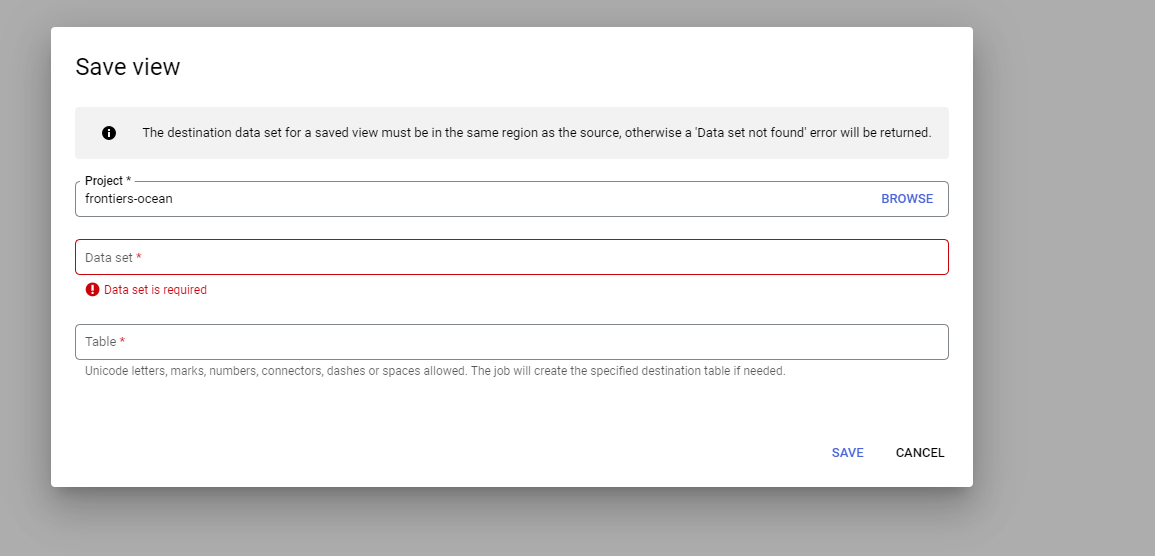


Creating views A View is a virtual table that contains data from one or multiple tables as the result set of your SQL statement. You can query a View in the same way you query a Table, but the first one is read-only and contains only the columns specified in the query.

A View can be created by the Editor pane in the SQL Workbench. You have to compose your query and then on the upper menu, there is the *Save* button that shows you the *Save view* option.



In the *Save view* dialog you must set the Project and the Dataset, in which store the view, and also the Name which must be unique per dataset.



**Creating materialized views**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/writing/#creating-materialized-views)

Materialized Views are precomputed views used to increase performance and efficiency in workloads that have the characteristic of common and repeated queries. It is not possible to load data into a materialized view or manipulate directly its data.

If your query has several steps and retrieves a large amount of data, it could be better if you break your query results into stages. You can use the materialized view for your intermediate query results to reduce the retrieved information each time the query is run.

You must have permission to create a table to create materialized views and once created, the cost of storage will be on us.

A materialized View cannot be created from query results or by the Console but you can use a SQL statement in the BigQuery Editor or *bq command* from your terminal.

**SQL**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/writing/#sql)

In the Query editor you can define the following SQL statement to create a Materialized View:

CREATE MATERIALIZED VIEW <myproject.mydataset.my\_materialized\_table> AS (  
 SELECT <column1>, <column2>  
 FROM <myproject.mydataset.my\_base\_table>  
);

and then *Run*.

**bq command**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/writing/#bq-command)

From your Terminal, with gcloud CLI, you can execute:

bq query --use\_legacy\_sql=false '  
CREATE MATERIALIZED VIEW <myproject.mydataset.my\_materialized\_table>  
AS SELECT <column1>,<column2>  
FROM <project-id.my\_dataset.my\_base\_table>'

**How to connect**

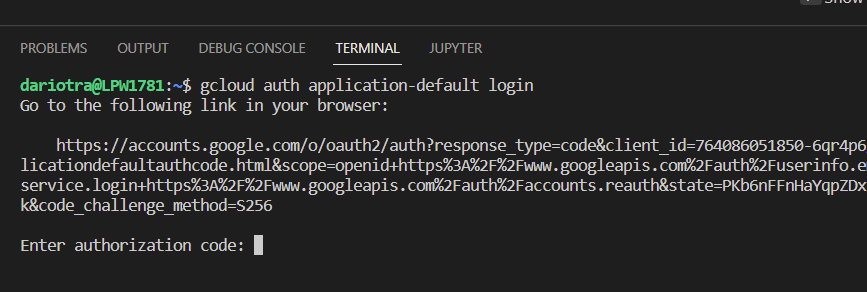
**Connect to BigQuery via terminal**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/connect/#connect-to-bigquery-via-terminal)

You can use a *Terminal* and set up a local environment to connect with the Google Cloud Platform.

You have to install gcloud CLI. According to your OS, you can follow [this guide](https://cloud.google.com/sdk/docs/install)

Then use the [Application Default Credentials (ADC)](https://cloud.google.com/docs/authentication/provide-credentials-adc) to authenticate with your account, without using Service Account.

$ gcloud auth application-default login



The command gcloud auth application-default login flow forces to use a web browser. If we run this on the WSL, there is no browser installed, neither that can be launched with ease (no X server for launching and managing windows) and so gcloud prompts the user with a command to be launched on a machine with gcloud already configured that also has access to a web browser.

The official way to address this is to install gcloud both on the WSL and the W10 environment, using PowerShell, and entering the command on PowerShell to launch the web browser and continue with the login process. However, this is a little bit annoying, and a little unofficial workaround can be used by entering the following command on the WSL:

$ DISPLAY=":0" gcloud auth login

This forces the system to launch the default browser on Linux and seems to work for this specific scenario.

This problem appears in W10. I don't have access to a W11 machine, but theoretically, W11 adds support for GUI applications on WSL, so there should be no problem. ([For more](https://docs.microsoft.com/en-us/windows/wsl/tutorials/gui-apps)) Otherwise, it is possible also to use a command line by the web browser.

The default project can be set with the command:

gcloud config set project your-project

Otherwise, if you want to override the default option you can use the flag *–project\_id=your-project* inside your command.

If you want to create a configuration file, you can create a "bigqueryrc" file and specify it in the command with the global flag

--bigqueryrc=<path-of-your-bigqueryrc>

If that is not specified, then the command uses an environment variable, otherwise *$HOME/.bigqueryrc* or *~/.bigqueryrc* are used.

The *bq* command line enables you to manipulate all the data with many commands. You can check all of them and all the available options here.

Here there some useful commands:

to list resource

```sh  
bq ls <FLAG> <RESOURCE>

to create a query job:

bq query <FLAG> 'YOUR-QUERY'

to display all columns of the first 100 elements, but with *-–selected\_field* and *--max\_rows* options you can manipulate your results.

bq head <FLAG>

**BigQuery APIs**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/connect/#bigquery-apis)

Google provides multiple sets of APIs and libraries to manage BigQuery resources. There are many distinct API services, each one with distinct functionality.

The main APIs are

* **BigQuery API**: you can find all you need to interact with core resources.
* **BigQuery Data Transfer API**: it is used to manage ingestion workflows.
* **BigQuery Connection API**: it enables you to interact with remote data sources. You can check all the available APIs [here](https://cloud.google.com/bigquery/docs/reference/libraries-overview).

**Explain main consumers**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/connect/#explain-main-consumers)

**Python**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/connect/#python)

Google Cloud Platform provides some [**Python** libraries](https://cloud.google.com/python/docs/reference) to use in your favorite environment, otherwise you can also set a new python env. From your editor you can install a package for an individual API that you need using thanks to command *pip install*, for example:

pip install --upgrade google-cloud-bigquery

Then you have all you need to create your query and interact with the Ocean like in this example:

*#Import a specific library*  
from google.cloud import bigquery  
  
*#define the project\_id where the query have to be executed*  
project\_id = 'frontiers-ocean'  
  
*#Initialize a client to authenticate and connect to the BigQuery API*  
client = bigquery.Client(project\_id)  
  
*#Write you query*  
query\_job = client.query(  
 """  
 SELECT  
 SourceId,  
 FullName  
 FROM `frontiers-ocean.dataset\_aira\_knowledge.Source`  
 WHERE FullName like '%Frontiers%'  
 ORDER BY Name DESC  
 """  
)  
  
*#Execute your query*  
results = query\_job.result() *# Waits for job to complete.*  
  
*#Display your result*  
for row in results:  
 print("{} : {} views".format(row.SourceId, row.FullName))

**Java**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/connect/#java)

Google Cloud Platform provides some [**Java bom**](https://cloud.google.com/java/docs/bom) to use in your favorite environment, otherwise you can also set a [new Java env](https://cloud.google.com/java/docs/setup). If you are using Visual Studio Code or Intellij etc you can add client libraries to your project directly from the IDE plugins.

#Import a specific library import com.google.cloud.bigquery.\*;  
import java.util.UUID;  
  
QueryJobConfiguration queryConfig =  
 QueryJobConfiguration.newBuilder(  
 "SELECT commit, author, repo\_name "  
 + "FROM `bigquery-public-data.github\_repos.commits` "  
 + "WHERE subject like '%bigquery%' "  
 + "ORDER BY subject DESC LIMIT 10")  
 // Use standard SQL syntax for queries.  
 // See: https://cloud.google.com/bigquery/sql-reference/  
 .setUseLegacySql(false)  
 .build();  
  
// Create a job ID so that we can safely retry.  
JobId jobId = JobId.of(UUID.randomUUID().toString());  
Job queryJob = bigquery.create(JobInfo.newBuilder(queryConfig).setJobId(jobId).build());  
  
// Wait for the query to complete.  
queryJob = queryJob.waitFor();  
  
// Check for errors  
if (queryJob == null) {  
 throw new RuntimeException("Job no longer exists");  
} else if (queryJob.getStatus().getError() != null) {  
 // You can also look at queryJob.getStatus().getExecutionErrors() for all  
 // errors, not just the latest one.  
 throw new RuntimeException(queryJob.getStatus().getError().toString());  
}  
  
// Get the results.  
TableResult result = queryJob.getQueryResults();  
  
// Print all pages of the results.  
for (FieldValueList row : result.iterateAll()) {  
 String commit = row.get("commit").getStringValue();  
 FieldValueList author = row.get("author").getRecordValue();  
 String name = author.get("name").getStringValue();  
 String email = author.get("email").getStringValue();  
  
 String repoName = row.get("repo\_name").getRecordValue().get(0).getStringValue();  
 System.out.printf(  
 "Repo name: %s Author name: %s email: %s commit: %s\n", repoName, name, email, commit);  
}

**.NET**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/connect/#net)

Google provides Big Query client for **.NET** platform, to install it into your project simply add it with dotnet command

dotnet add package Google.Cloud.BigQuery.V2

When you create client without specifying credentials details like BigQueryClient.Create(projectId), private key path will be loaded from env. variable **GOOGLE\_APPLICATION\_CREDENTIALS**.

It's not always feasible to use key path for authentication, fortunately client supports passing credentials via JsonCredentialParameters

var parameters = new JsonCredentialParameters  
{  
 Type = "service\_account",  
 ClientEmail = clientEmail,  
 PrivateKey = options.PrivateKey  
};  
var googleCredential = GoogleCredential.FromJsonParameters(credentialParameters);  
  
BigQueryClient.Create(projectId, googleCredential);

**Third party**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/connect/#third-party)

There are also some third-party libraries available:

* [Pandas (for python)](https://cloud.google.com/bigquery/docs/pandas-gbq-migration)
* [R bigrquery or BigQueryR](https://cloud.google.com/bigquery/docs/pandas-gbq-migration)
* [Scala](https://github.com/GoogleCloudDataproc/spark-bigquery-connector)

# Improving your queries

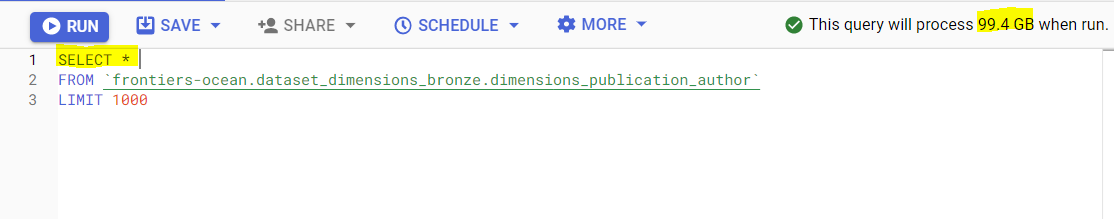
## Why to improve queries[​](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/improve/#why-to-improve-queries)

It is often insufficient to merely know the location of data, as the method of data retrieval can significantly impact results.

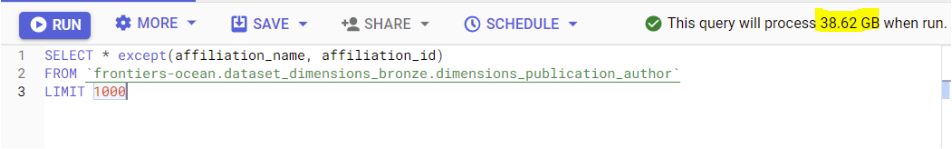
Query composition efficiency plays a crucial role in facilitating fast query execution and cost savings. Numerous optimization techniques are available, and the following suggestions serve as a starting point.

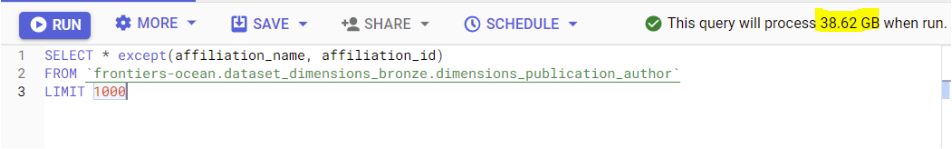
## Necessary column[​](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/improve/#necessary-column)

To optimize queries, it is essential to limit the number of bytes scanned. This can be achieved through one of the most straightforward methods: ensuring that only the necessary columns are included in the query.



The first thing you can do, ensures you are only selecting the column that you need. The data scanned is proportional to the number of columns used in the query.





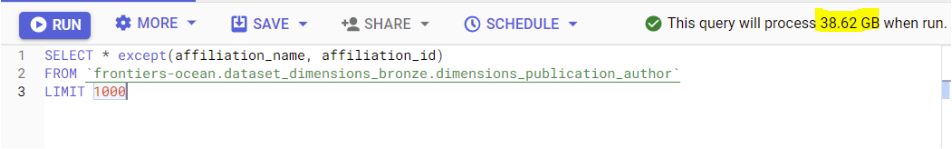
If you want to take a peek at the first few rows of the data to get a feel for what you are working with, use the Preview feature instead.

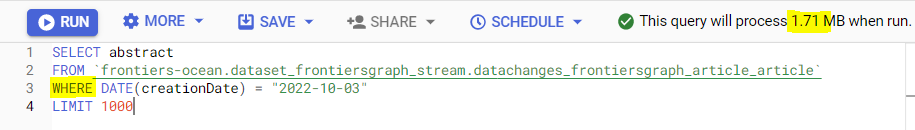
## Partitioning and Clustering[​](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/improve/#partitioning-and-clustering)

How data is stored inside the table can also impact how you compose your query. In Big Query, a table can be partitioned and clustered to optimize how the data is divided into storage blocks. All this information is gathered on the Details page of the table.



In this case, you can leverage partitioning and clustering to automatically prune out data you don't need in your query. You can use the WHERE clauses on columns used to partition or cluster the table itself to make your query faster.

Before: 

After: 

To limit the partitions that are scanned in a query, use a constant expression in your filter, if you use table value, which is dynamic, you could have worse performance.

## Aggregate Function[​](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/improve/#aggregate-function)

It is possible to optimize the performance of aggregate function using an equivalent approximate version. This approximation yields faster query performance but introduces some uncertain in the results. If your use case enables you to do it, you can use the function APPROX\_COUNT\_DISTINCT instead of using COUNT (DISTINCT).

For more information about approximate aggregate function, you can check here.

## Group by[​](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/improve/#group-by)

In your query you could use the GROUP BY clauses. It has an impact on how Big Data executes the query, it requires a few different aggregation steps and could slow down your execution.

### Late aggregation[​](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/improve/#late-aggregation)

In case of more complex queries, that have sub-SELECTs, move your aggregation function up to the top to avoid multiple useless GROUP BY executions.

For example, instead of having multiple steps for aggregate function and multiple GROUP BY like this one:

SELECT t1.dim1, sum(t1.m1), sum(t2.m2)  
FROM (  
 SELECT dim1, sum(metric1) m1  
 FROM "dataset.table1"  
 GROUP BY 1) t1  
JOIN (  
 SELECT dim1, SUM(metric2) m2  
 FROM "dataset.table2"  
 GROUP BY 1) t2  
ON t1.dim1 = t2.dim2  
GROUP BY 1

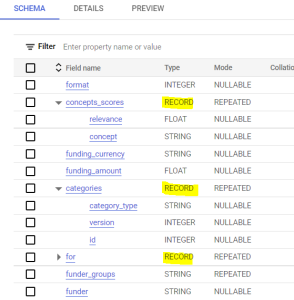
The optimized version is:

SELECT t1.dim1, sum(t1.m1), sum(t2.m2)  
FROM (  
 SELECT dim1, metric1 m1  
 FROM "dataset.table1" ) t1  
JOIN (  
 SELECT dim1, metric2 m2  
 FROM "dataset.table2") t2  
ON t1.dim1 = t2.dim2  
GROUP BY 1

In this way, we have eliminated some intermediate-aggregation steps and there is only **one** GROUP BY at the end.

Only in the case that your table can be drastically reduced by aggregating in preparation for being joined, then aggregate it early! The fewer data to join the more efficient will be your execution.

### Nested reported data[​](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/improve/#nested-reported-data)

Another tip that can be used to avoid using GROUP BY is to create a table with nest repeated data. 

Instead have a flat normalized table, with one row for each element, you can create a repeated column that contains a list of all the items. Then, you can use some functions like ARRAY\_LENGTH to obtain aggregate information.

## Join[​](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/improve/#join)

BigQuery can automatically detect which type of join would be more convenient due to the tables' size.

It could be a shuffle join in case of two large tables, or a broadcast join with a table small enough to fit in memory. For this reason, the order in JOIN clauses could make a difference.

### Table order[​](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/improve/#table-order)

First, make sure you place the largest table first in the JOIN query followed by the smallest, and then by decreasing size.

The details panel of the table and also the Query Validator (in the upper-right corner inside the SQL Editor), could be helpful to estimate each table size.

### Where order[​](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/improve/#where-order)

Next, WHERE clauses should be executed as soon as possible so that slots for forming the JOIN are working with the least amount of data.

Make sure that we are using filters on both tables, so to eliminate data from both sides of the JOIN.

When you use the WHERE clauses, a big query assumes that you have chosen the best order for your filters. The first part of your WHERE clauses should contain a filter that will eliminate the most data.

## Order by with a limit[​](https://ocean-docs.frontiers-ss-dev.info/next/getting-started/learning-path/improve/#order-by-with-a-limit)

If you need to sort your result set you could use the ORDER BY clauses.

Sometimes if you are attempting to order a very large result set, you may run into a resource exceeded error. It is preferable to sort on only a subset of data than sort all the dataset and apply the LIMIT clause

# New project organization

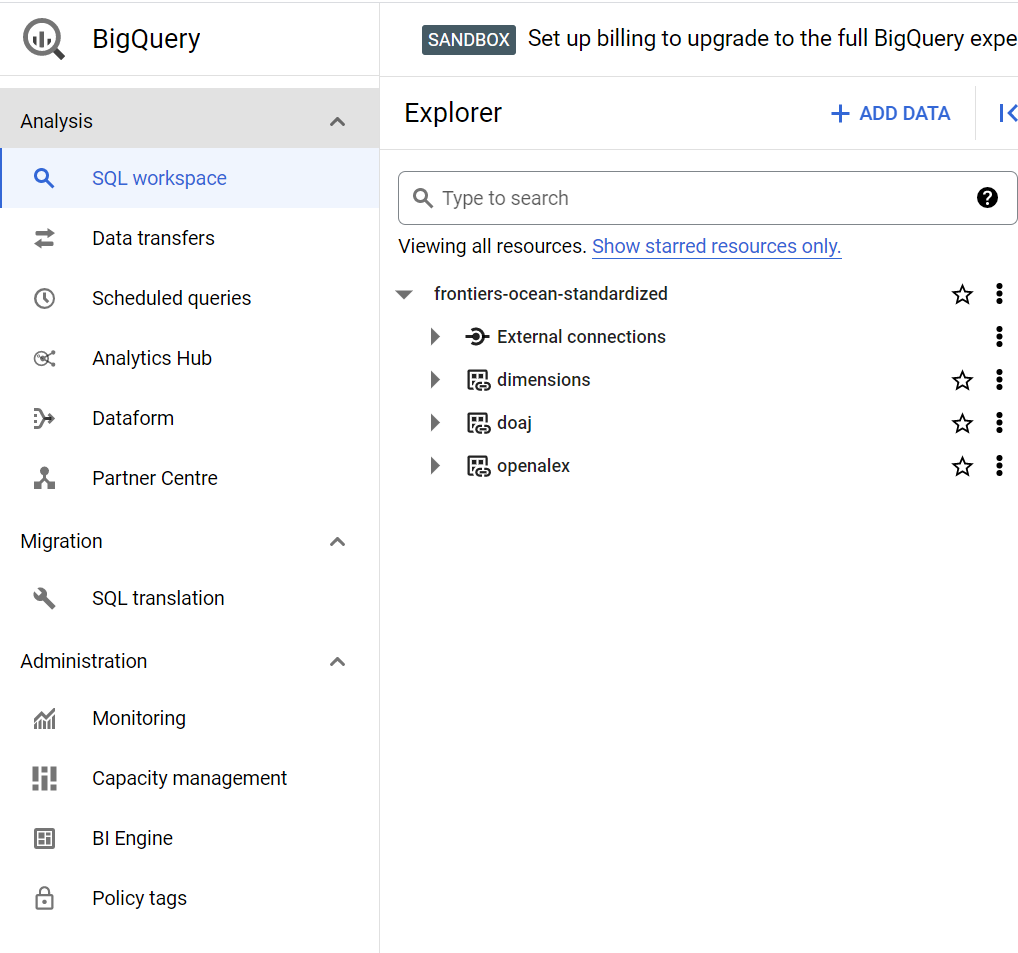
At Ocean Breeze, our primary focus is on providing the best user experience to our users and groups. Our review of Frontiers Ocean usage revealed that although the **platform offers numerous datasets** and possibilities, in reality, **each group** only works with a **very limited number** of datasets. Having a global project with all the datasets may negatively impact their daily performance, as they have to constantly face a growing amount of datasets that can in some cases overwhelm the user.

In Frontiers Ocean, datasets are placed in global projects, that will hold all the datasets with a specific state or goal. The datasets are visible for all groups of users, no matter if they are going to use them or not. As a consequence, the same permissions will be shared among all the users, without the need to tailor our policies individually:



To address this issue, we have changed the structure in Ocean Breeze in a way in which **each group has its own project**. By default, the project is empty, except for some sample datasets for tutorial purposes, and each group can **choose to add the datasets they need**. This approach prevents users from having to work with the whole list of datasets in the platform, which can negatively impact their productivity.

Each group has a set of permissions for their projects, which hold only the datasets the users choose. This is achieved using the linked datasets from the AnalyticsHub. The users still have the possibility of adding private datasets in their projects:



Furthermore, new users will find it more comfortable to access Ocean Breeze as they will be presented with an empty workspace or a reduced workspace based on their group's work. They will have access to a full set of datasets that they can add to their workspace but will not be required to navigate through all of them all the time.

**Project management**

*Ocean Breeze* is organized just as the company structure is, and it allows access to each project through Google Groups. Each project has at least a Google Group that has permissions on it, and for the creation of a project it is required to define which Google Group will be the owner of the project.

The folder structure in *Ocean Breeze* is based in the company structure, in 2 levels of depth, having the area as the first level, and the team name as the second, and looks like:

Ocean Breeze folder structure

📂 Ocean-Breeze  
 ├── 📂 Technology  
 ├── 📂 SE-ADMAU-Core-Data-Braavos  
 | ├── ocean-tech-admau-asgard-p-usr  
 ├── 📂 AI Products-AppliedAI-DataScience  
 ├── ocean-tech-ai-data-sc-p-usr  
 ├── ocean-tech-ai-data-sc-e-tableau  
 ├── 📂 Publishing  
 ├── 📂Journal-Market-Intelligence  
 ├── ocean-tech-market-intelligence-p-users  
 ├── 📂Advanced-Analytics  
 ├── ocean-tech-adv-analytics-p-usr  
 ├── ocean-tech-adv-analytics-e-dev  
 ├── ocean-tech-adv-analytics-e-pro

The folder structure in Google is only visual and for organization purposes, but the project names are all shared in a common root. That means we have to name it already with an structure that makes them identifiable and able to be ordered no matter if the folders exist or not.

Google allows 30 characters for the naming of the projects, and in *Ocean Breeze* the naming structure is:

Ocean Breeze project name structure

ocean-<area>-<team>-<project\_type>-<project\_name>

Where:

* *ocean* prefix is fixed, and a way to separate all the projects for Ocean from the rest of the projects in GCP
* *area* is the abbreviation of the area of the company the team belongs to.
* *team* is the team name. As long as the total amount of chars (30) is not passed, it can be abbreviated or not.
* *project\_type* is the project type. It is only one letter and it can be one of the following: p, e or c. Later in the docs we will explain the different options.
* *project\_name* as the project name for the team. We have several predefined names but again, as long as the total amount of chars is smaller or equal to 30, it can be custom decided by the team.

Although there is no limit in the amount of projects a team can hold, we recommend using the smallest amount of projects needed for the team use case.

# Project types

Depending on the needs and usage of the projects in Ocean, they will be created based on a project type. The project type determines the set of tools and actions that can be used within the project. Each project type is intended to solve some user needs.

## Playground projects (p)[​](https://ocean-docs.frontiers-ss-dev.info/next/project-management/project-types/#playground-projects-p)

Playground projects are named type "p" in the project name and they are created for manual use of BigQuery. These projects are for users manual interaction with datasets, data wrangling and exploration. They allow private datasets, and all functionalities within BigQuery, as long as they can be done with a user account.

By default every team will have a playground project, as it is the project in which to access ocean, explore, find the datasets your team may use, and use them to create private datasets for your own consumption.

## Environment projects (e)[​](https://ocean-docs.frontiers-ss-dev.info/next/project-management/project-types/#environment-projects-e)

These projects extend the playground projects so that service accounts can be linked to them. There can be several of them by team, and there will be teams that don't ever need an environment project. The idea is to have these projects as long as you have a third party service needing to access Ocean Breeze programmatically.

Connect BigQuery to a visualization tool, or to any other system that gets or puts information into BigQuery programmatically is the typical use case for these projects.

These projects don't grant permissions to external services on GCP, like virtual machines or other tools.

## Custom projects (c)[​](https://ocean-docs.frontiers-ss-dev.info/next/project-management/project-types/#custom-projects-c)

Custom projects allow all permissions on GCP platform. They are meant for teams with high technical skill and custom needs allowing them to use other services besides BigQuery. Custom projects will still place the interaction with Ocean Breeze data sources at the core.

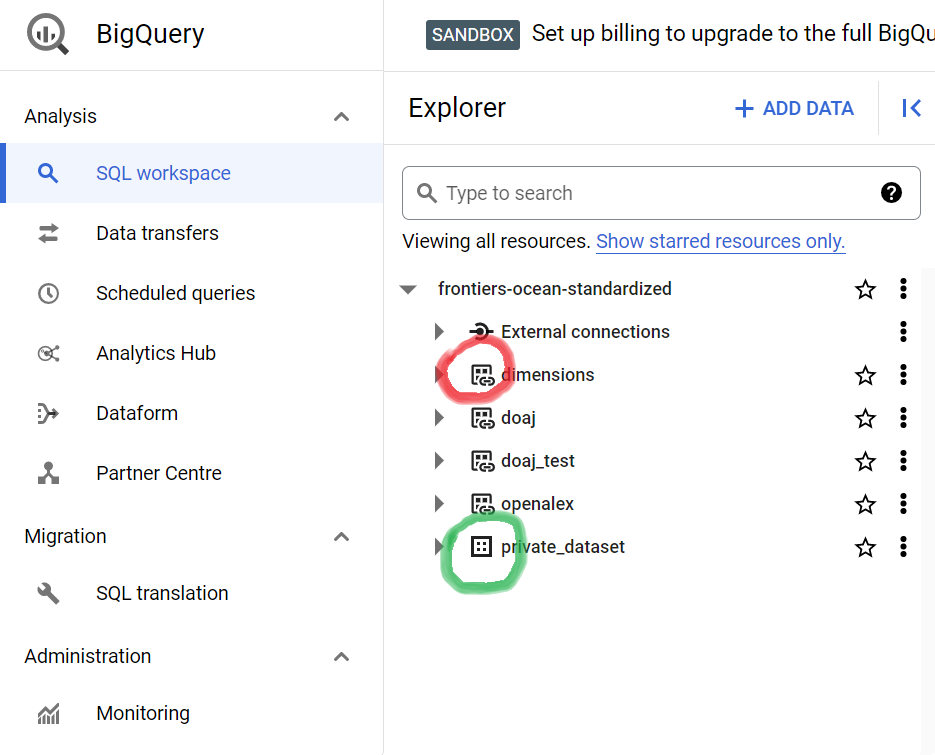
In this case, Team Braavos only provides access to the projects, and manages them, but the owners are directly the teams involved, being able to perform any action with any technology within the Google Cloud Platform.

# How to find data

In the [previous section](https://ocean-docs.frontiers-ss-dev.info/next/project-management/project-organization/), we described the new project organization in Ocean. With this new organization, each team will have its own project in BigQuery, allowing to have private datasets, as well as adding and searching for official datasets.

**Analytics Hub** is a tool within Google Cloud Platform that allows sharing datasets across the organization (and with other organizations if needed). When a dataset is shared, a virtual copy is moved to the project that required the data, and it appears as a **linked dataset**. In the next image you can identify a linked dataset, surrounded in red, and a private dataset, surrounded in green.

A liked dataset cannot be edited, it can only be added or removed (that won't cause a delete action) as it works just as a subscription. You are granted the use of this dataset while you have it linked to your project, and all queries done to this dataset will be automatically done in the origin dataset, just as if you were querying the original dataset.

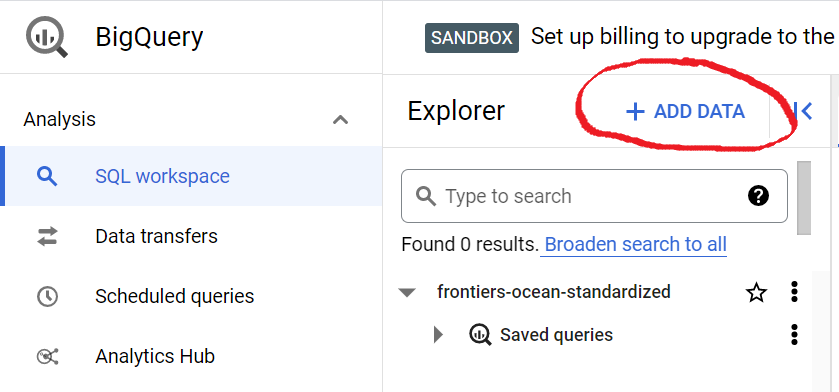


If users within a group only have their custom selection of datasets, how can they know **where are the rest of the datasets?**

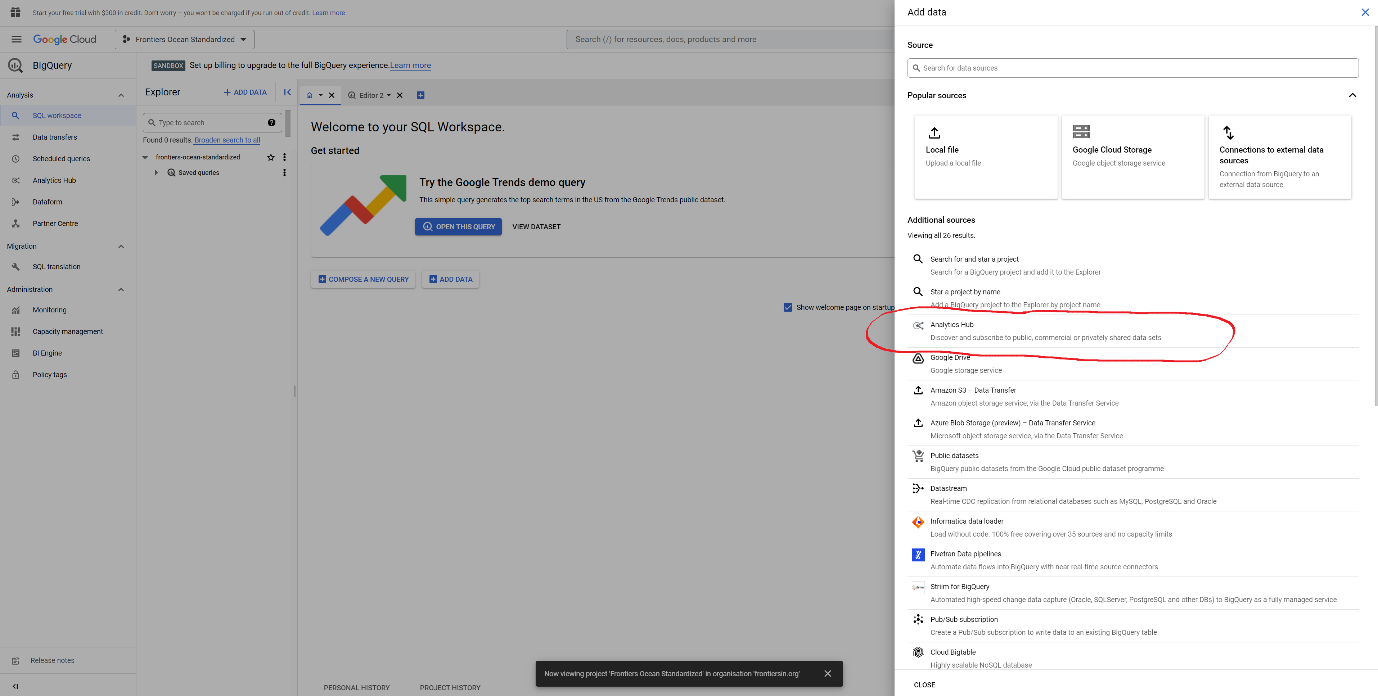
To find datasets that you can link to your project you can use the **Analytics Hub**. Ocean Breeze relies on the Analytics Hub to provide access to the different datasets. Analytics Hub is a product within Google Cloud Platform that provides a way to share data across projects, enabling the possibility of accessing the same dataset from multiple projects without the need of having a copy of them.

In the new Ocean Breeze, we are sharing all datasets through the **Analytics Hub**, allowing every group to **bring the datasets** they find more interesting to their projects, having a clean view of the list of datasets they are working on.

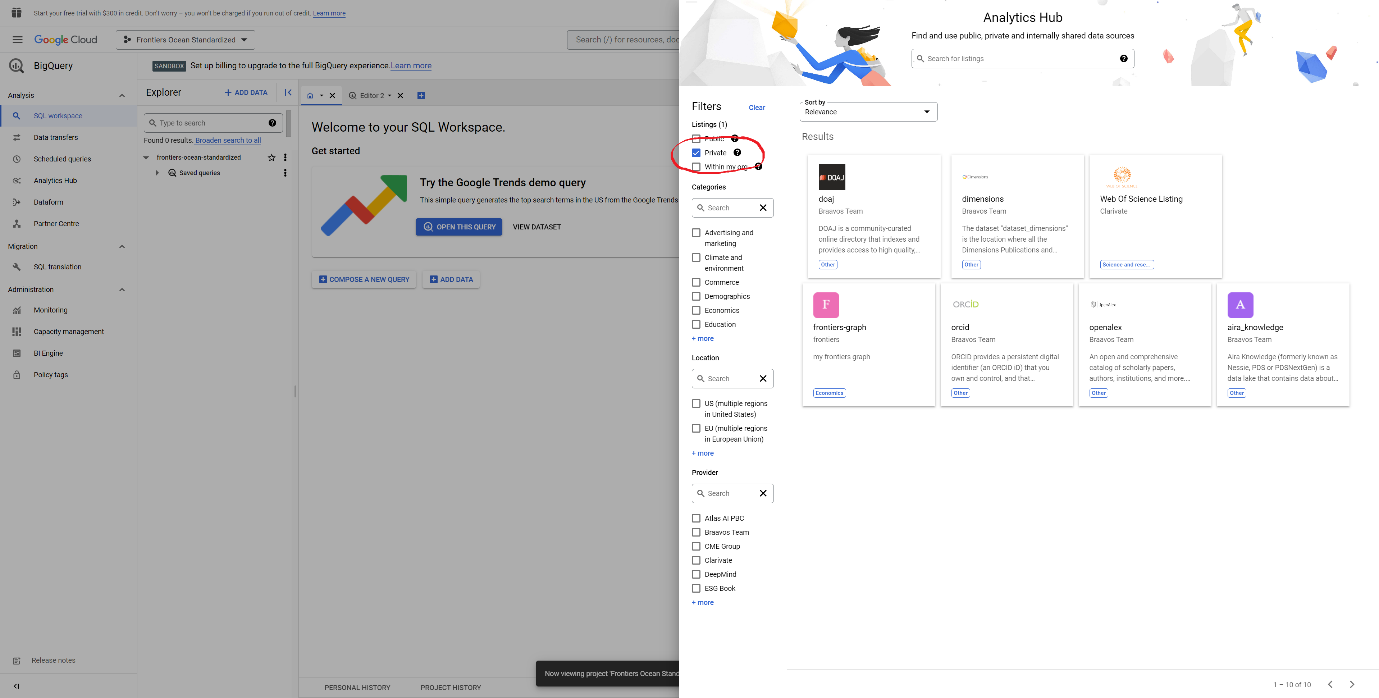
To explore the whole set of private datasets from Ocean, just click on the button "ADD DATA" in the BigQuery interface.



Then you will be presented a set of options, select the **Analytics Hub** as the source, and you will see the whole list of datasets available.



Within the **Analytics Hub**, check the Private listings in order to see all the listings of datasets that our company is providing.

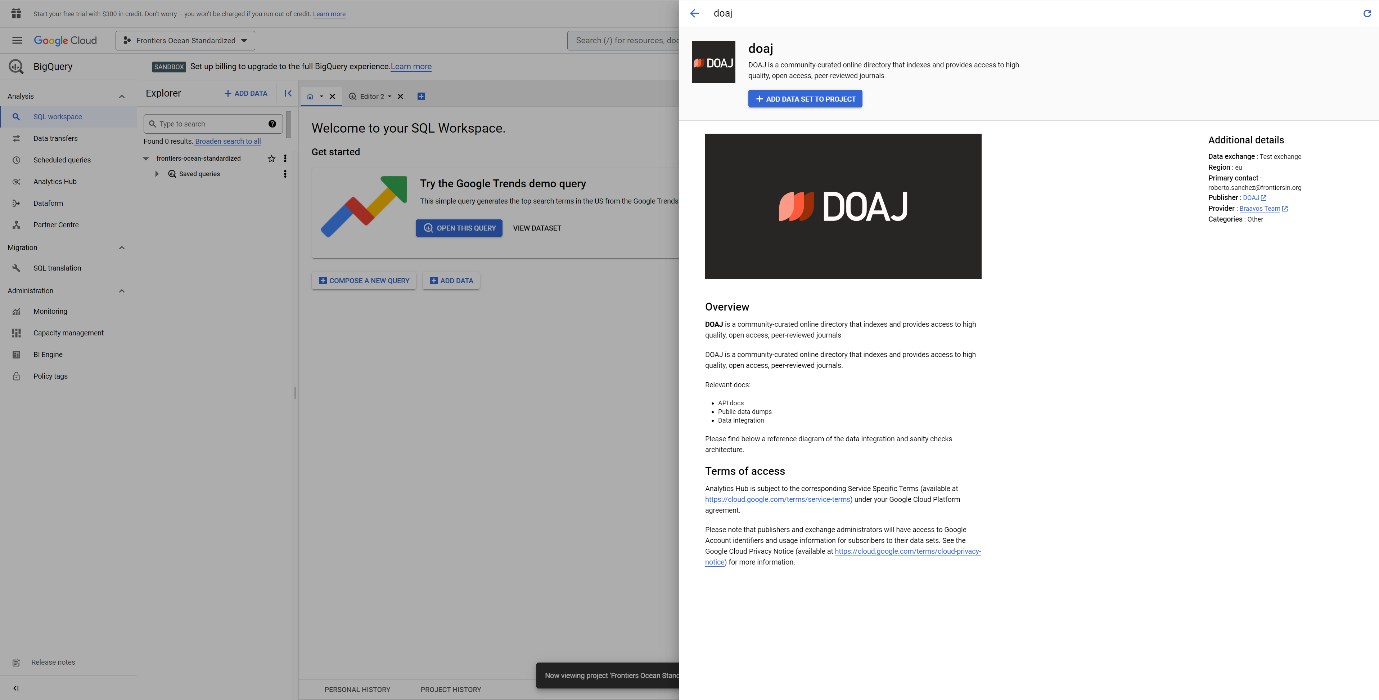


Each listing has **icon, name, owner and a description,** so you can identify if the data inside this listing looks promising for your daily work or not, and **decide if you want to add it** to your project or not.

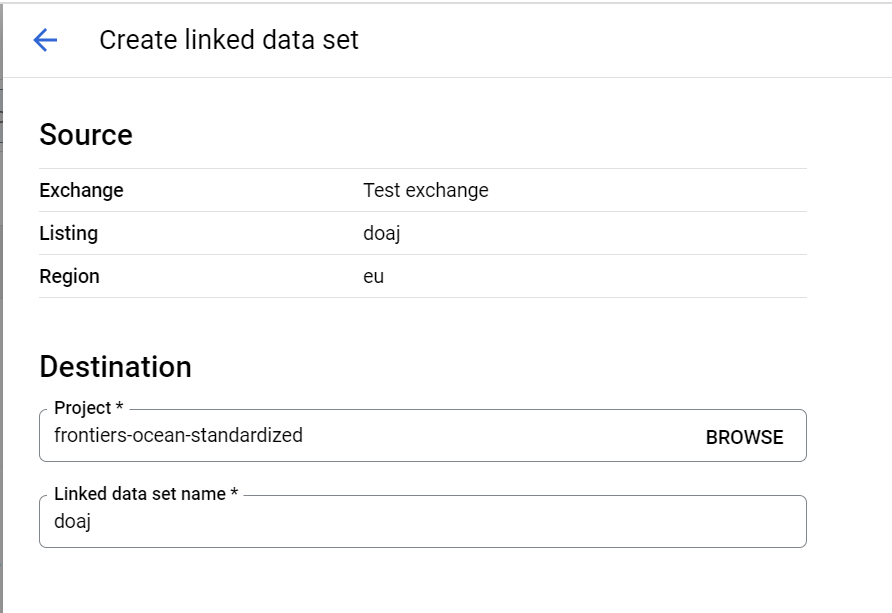
# Adding datasets to your project

Once you click in a listing, you will access a **brief documentation** for the listing, that should provide you an overview of the dataset, to understand if it can be interesting for you.

Now you have the possibility to add it to your project in just one click.

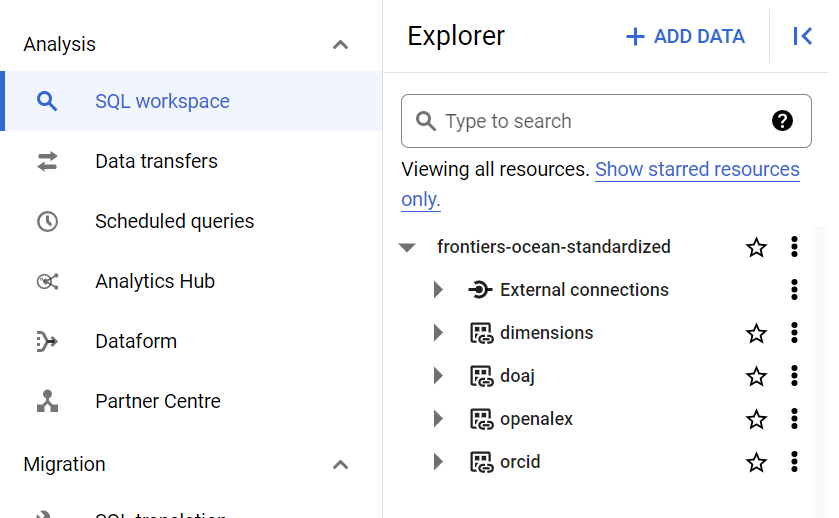


Click on **Add Data to my project** button, and you will be prompted to select a name for the dataset within your project. It is possible to add several times the same dataset to a project, as long as they have different names, and the naming doesn't have to be compliant with naming conventions, as each team can have their own convention.



The dataset within your project will be then linked to the official dataset, and you can use it just how you used the rest of the datasets in the past.

Linked datasets have a special icon, that will let you know that the dataset is coming from a published official dataset within the **Analytics Hub**. The linked dataset works as if it was a view of the original dataset, linked to the changes and actually querying remotely to the original one when it's queried, but without the disadvantages of other options like views had in the past.



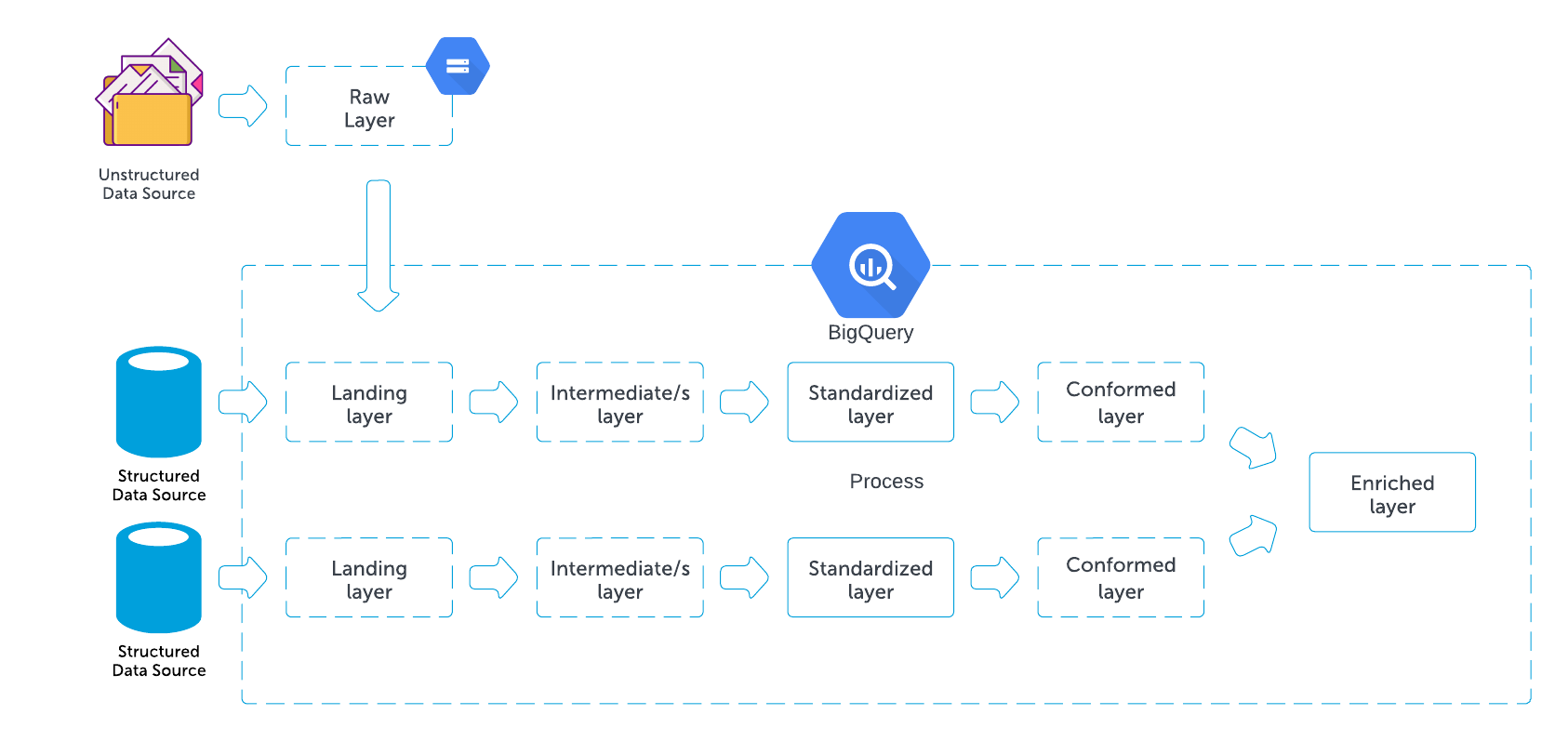
Feel free to use as many linked datasets as you want, as they are a tool for your team to customize BigQuery to improve the workflow of your team.

**How data is layered**

*Ocean Breeze* ingests and transform data sources in order to be ready to be consumed by users.

Ocean Breeze offers access to **multiple stages of the information** processing pipeline, catering to various user profiles and needs. While **some users** may only require access to the **final stage** of information, we prioritize providing access to **different stages** of the pipeline. This allows each user or service to **leverage the entire pipeline** and determine the **stage, format, and structure** they wish to access.

**Dataset Stages**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/project-management/how-data-is-layered/#dataset-stages)



Stages in *Ocean Breeze* are not always present for every source. A source can have many stages or it can have only one.

The only stage that every source needs to have is the **standardized layer**. The rest is depending on the dataset. Datasets can be provided in different stages or not, depending on the use case and the needs.

When accessing new datasets it is important to **check the stage of the dataset** to understand how it should be regarding **naming conventions, curation and structure status**.

In order to make it easier to identify the **stage of a dataset**, we created a set of **badges** that will be added to every dataset provided in the hub. These are the badges that will be shown for the different stages and layers when accessing information:

Logo

Description automatically generated

**Raw layer**

This layer is still not directly accessible from BigQuery, and it is just a place in which we can have original unstructured information in **Google Cloud Storage**, that will be the first step for getting the data into BigQuery when dealing with unstructured information or files.

Logo

Description automatically generated

**Landing layer**

This is the **first stage** of incorporating external data into **BigQuery**. Data arriving this layer is **already in a somehow tabular format**. This stage retains all the information in the source, without any structure, naming changes or curation process.

Logo

Description automatically generated

**Intermediate layer**

At this stage, the data has **not yet been transformed into the standardized format** but needs some intermediate steps in order to do the process. The datasets can pass through a series steps before being in standardized format, so there can be **zero or more** intermediate datasets for each source, depending on the required transformations.

Logo, company name

Description automatically generated

**Standardized layer**

This layer contains **clean and tabular data** in its **original naming conventions**. **All sources must have a standardized dataset published**. This layer serves as the primary starting point for using data in Ocean Breeze as a normal consumer. The standardization is applied on field formats

Logo

Description automatically generated

**Conformed layer**

The data in this layer is clean, has **uniform naming conventions across the company**, and can be **denormalized**. This layer serves to simplify the use of data when dealing with multiple sources or **working in analytics**. There can be **zero or more conformed layers** for each data source, depending on user needs.



**Enriched layer**

When data is **joint from different sources** to create a new dataset, it is known as an **enriched dataset**. Enriched datasets adhere to the general naming conventions for Ocean and may utilize **two or more data sources in any stage**. Enriched datasets are the result of common shared needs between different data sources, that become a dataset themselves in order to help \_Ocean Breeze\_ users.

**Reference**



This section will be a table of all the sources, and context information on who created the sources, who is integrating the data, and a briefing. Then, each source will have its own section, having contextual information to understand the source and the possibilities.

By now, Ringgold will be the example of the kind of things you will be able to see here.

| **Directory** | **Source** | **Producer** | **Integrator** | **Description** |
| --- | --- | --- | --- | --- |
| Ringgold | Ringgold | Ringgold | Braavos Team | Organization information from an external provider. It provides organization data and hierarchy |

**Entities**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/data-directory/reference/#entities)



The entities table will describe all the entities that are commonly found in the datasets, the sources that contain information about those entities, and the field that corresponds to that entity in each source. The purpose of this table will be to help understand all the entities and serve as a guide to know which ones can be used to enrich each dataset.

This is still a WIP. The structure of the table may vary over time, and the format will change to make the table more readable.

| **Entity** | **Description** | **Source** | **Field** | **Comment** |
| --- | --- | --- | --- | --- |
| Author name | *Author description* | Aira Knowledge | AuthorFullList.NormalizedName |  |
| Author name | *Author description* | Frontiers Graph | article\_article\_author.firstName + article\_article\_author.middleName + article\_article\_author.lastName |  |
| Author name | *Author description* | PubPeer | pubpeer.Authors | Array of author names |
| DOI | *DOI description* | Aira Knowledge | Publication.Doi |  |
| DOI | *DOI description* | Frontiers Graph | article\_article.doi |  |
| DOI | *DOI description* | PubPeer | pubpeer.Doi | No hyperlink, lowercase |
| Journal name | *Journal description* | Aira Knowledge | Journal.DisplayName |  |
| Journal name | *Journal description* | Frontiers Graph | journal\_journal.name |  |
| Journal name | *Journal description* | PubPeer | pubpeer.Journal |  |
| Article title | *Article title description* | Aira Knowledge | Publication.Title |  |
| Article title | *Article title description* | Frontiers Graph | article\_article.title |  |
| Article title | *Article title description* | PubPeer | pubpeer.Article\_Title |  |
| Organization name | *Organization description* | Aira Knowledge | Organization.normalized\_name |  |
| Organization name | *Organization description* | Frontiers Graph | organization\_organization.name |  |
| Organization name | *Organization description* | Ringgold | Organizations.name OrgAltNames.name |  |
| Places | *Places description* | Ringgold | Places.name |  |

**Cost optimization**

BigQuery billing has two main components:

* **Storage costs** - to store the data loaded. Based on the number of bytes stored.
* **Analysis costs** - to process queries. Based on the number of bytes scanned to serve a query.

In the upcoming sections, we will cover some recommendations to achieve a cost-effective and performant usage of BigQuery.

<https://cloud.google.com/blog/products/data-analytics/cost-optimization-best-practices-for-bigquery>

**Storage optimization**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/conventions-and-data-policy/costs/#storage-optimization)

**Take advantage of long-term storage**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/conventions-and-data-policy/costs/#take-advantage-of-long-term-storage)

All data loaded to BigQuery is transitioned to long-term storage after 90 days. This storage class has no performance changes associated, but a 50% price drop. The key to benefiting from long-term storage is to not alter the data, something that might be tricky or impossible in some scenarios. However, it can be handled at partition-level, so untouched partitions for 90 days will be eligible.

TIP

So, try to use time based partitions (by ingestion time or by some data field) to get into long-term storage.

**Use table expiration for temporary datasets**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/conventions-and-data-policy/costs/#use-table-expiration-for-temporary-datasets)

Sometimes a particular table or dataset is only needed for a short period of time. In these situations, is recommended to enable a default expiration policy to make BigQuery to automatically remove the data when it is no longer necessary. This setting can be set at table creation or by alter the table with a DDL statement.

**Analysis optimization**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/conventions-and-data-policy/costs/#analysis-optimization)

**Select only the columns needed**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/conventions-and-data-policy/costs/#select-only-the-columns-needed)

BigQuery stores the data in a columnar fashion. For a "SELECT \*" query, BigQuery would perform a full data scan across of data columns, producing high costs as queries are billed depending on the scanned bytes. So, the most important cost-optimization technique, is to just select the columns needed.

**Take advantage of partitioned and clustered tables**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/conventions-and-data-policy/costs/#take-advantage-of-partitioned-and-clustered-tables)

Partitioned tables allow to reduce costs and boost performance if a query filters the data by the partition key.

While creating or updating a partitioned table, it's possible to enable “Require partition filter” which will force users to include a WHERE clause that specifies the partition column, or else the query will result in an error.

Inside a partition, BigQuery can sort the data into smaller blocks by using the clustering columns. Thus, when using query filters by clustering columns, BigQuery intelligently only scans the relevant blocks producing again lower costs and better performance.

**Prevent accidental human errors**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/conventions-and-data-policy/costs/#prevent-accidental-human-errors)

In this case, use the maximum bytes billed setting to limit query cost. Going above the limit will cause the query to fail without incurring the cost of the query.

Along with enabling cost control on a query level, you can apply similar logic to the user level and project level as well.

**Caching**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/conventions-and-data-policy/costs/#caching)

Caching can boost query performances, and results retrieved from the cached tables are free. By default, cache preference is turned on. The only con is that caching is per user, so a user could not benefit from other user's cached tables.

To significantly increase the cache hit across multiple users, use a single service account to query BigQuery

**Check BigQuery audit logs**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/conventions-and-data-policy/costs/#check-bigquery-audit-logs)

In order to track and monitor the queries, you can check the BigQuery audit logs. With the help of logs, you can troubleshoot issues, optimize your queries and understand the data usage patterns.

The logs are located within the project called ocean-tech-braavos-aud. Each dataset name in this project corresponds to the name of the project from which the logs are being collected. So for instance, the logs for the ocean-mc-ml-seo-p-usr project are located in the ocean-tech-braavos-aud.bq\_audit\_mc\_ml\_seo\_p\_usr dataset.

Here are some example queries that can be useful to understand the data usage (you can modify them accordingly, by changing the dataset, principal email, and other conditions to match your specific use case):

* Who is responsible for heavy queries?

SELECT  
 principalEmail,  
 totalbilledgigabytes,  
 jobChange.jobConfig.queryConfig.query,  
 eventTimestamp  
FROM  
 `ocean-tech-braavos-aud.{dataset\_name}.bigquery\_audit\_logs\_v2`  
WHERE  
 eventTimestamp > TIMESTAMP\_SUB(CURRENT\_TIMESTAMP(), INTERVAL 60 DAY)  
 AND principalemail NOT LIKE "%gserviceaccount.com"  
ORDER BY  
 totalbilledgigabytes DESC  
LIMIT  
 100;

* Which are the most repeated and expensive queries?

SELECT  
  count(0) as queries,  
  sum(totalbilledgigabytes) as sumtotalbilledgigabytes,  
  sum(estimatedcostusd) as sumestimatedcostusd,  
  ANY\_VALUE(jobChange.jobConfig.queryConfig.query) as query,  
  MD5(jobChange.jobConfig.queryConfig.query) AS query\_md5  
FROM  
  `ocean-tech-braavos-aud.{dataset\_name}.bigquery\_audit\_logs\_v2`  
WHERE  
  eventTimestamp > TIMESTAMP\_SUB(CURRENT\_TIMESTAMP(), INTERVAL 60 DAY)  
  AND principalemail LIKE "%gserviceaccount.com"  
  AND estimatedcostusd > 0.5  
group by query\_md5  
ORDER BY  
  queries desc

**SLA and Disaster Recovery**

This chapter addresses service availability and fail recovery considerations regarding BigQuery. BigQuery [offers a 99.99% uptime SLA](https://cloud.google.com/bigquery/sla), based on two components:

* In terms of **storage**, BigQuery always keeps two copies of the data in two different zones of the selected location. With multi-region locations, such as US or EU, it's guaranteed that both zones are at least 160 km distant.
* Regarding **computing**, BigQuery has redundant compute capacity across multiple zones.

The following table summarizes the involved GCP and BigQuery capabilities and how the policy manages them.

| **Feature** | **Category** | **Applies to all datasets** | **Description** |
| --- | --- | --- | --- |
| Multi-region (EU) | availability | Yes | Multi-regions increase the availability by being able to function following the loss of a single region. In addition, with the EU multi-region, loading or exporting data to Cloud Storage is more flexible, as it supports using regional buckets (included in the multi-region). Regarding pricing, multi-region is equivalent. <https://cloud.google.com/docs/geography-and-regions#multiregional_resources> <https://cloud.google.com/bigquery/docs/reliability-disaster> |
| Time-travel | accidental-error-recovery (1 week) business logic | Yes | Time-travel feature allows querying data from any point in time over the last seven days. Enabled by default. Allows to recover from accidental human errors. Useful in some business-logic scenarios that require access to old versions of the datasets. <https://cloud.google.com/bigquery/docs/time-travel> |
| Cross-region backups | disaster recovery | No | Under the failure of a whole location (in our case, the EU multi-region), the most reliable approach for disaster recovery would be to back up to other regions that do not share failure domains. For example, automated copies to Cloud Storage or BQ transfer jobs. In the case of EU multi-region, all included regional locations such as Belgium or Frankfurt would not be eligible, as they have shared failure domains with the multi-region. For mission-critical scenarios, the most reliable architecture would be running active-active. This means duplicating our pipelines (and data) to another region (e.g.: "US"). <https://cloud.google.com/bigquery/docs/reliability-disaster> |
| Snapshots | accidental-error-recovery (unlimited) business logic | No | Snapshots allow to backup data within the same region for longer than the 7 day time travel window. As time-travel, adds protection against accidental errors, but it does not increase the durability of the data. It requires the base table in order to recover a snapshot version. <https://cloud.google.com/bigquery/docs/table-snapshots-intro> <https://cloud.google.com/bigquery/docs/table-snapshots-scheduled> |

**Onboarding**

**Tools that you might want to install**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/onboarding/#tools-that-you-might-want-to-install)

Of course it depends on whether you have a Unix or a Windows machine, but let's try to be as general as possible.

* An IDE (Pycharm, Intellij) or a TextEditor (VS Studio) or the one you prefer
  + Currently, Frontiers doesn't provide and licence for these so you are bounded to the community editions
* A Terminal app (and if you're on windows [WSL](https://learn.microsoft.com/en-us/windows/wsl/install))
* git
* [Docker Hub](https://docs.docker.com/get-docker/) (that includes Docker CLI) or just Docker CLI
* [Kubernetes CLI](https://kubernetes.io/docs/tasks/tools/)
* A kubernetes context switcher: [kubectx](https://github.com/ahmetb/kubectx" \t "_blank)
* A Kubernetes UI to interact with the cluster. Pick one:
  + [OpenLens](https://github.com/MuhammedKalkan/OpenLens)
  + [K9s](https://k9scli.io/)
* Python 3.8+
  + It's nice if you install python via a python version manager. I recommend using [pyenv](https://github.com/pyenv/pyenv" \t "_blank)
* [Terraform](https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli)
* [dbt](https://docs.getdbt.com/docs/get-started/installation) with bigquery as adapter
* [Google Cloud CLI](https://cloud.google.com/sdk/gcloud)
* A Database management tool (we don't have a licence for these too). Pick one:
  + DBeaver
  + TablePlus
  + DataGrip
* [Frontiers GCP Tools](https://devops-server.frontiersin.net/Madrid/Architecture/_git/Frontiers.GCP.Tools) is a command-line project managed by Braavos that provides some shared functionally to manage resources on Google Cloud Platform

If you're on a Macbook and you're not familiar with it (or you want some nice tips to set your dev environment up), consider following [this guide](https://sourabhbajaj.com/mac-setup/) by Sourabh Bajaj et al.

**Our tools**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/onboarding/#our-tools)

Some of these tools are in the Frontiers private network and cannot be reached if you're not signed in with the VPN.

* Our repositories are in [Azure DevOps hosted, within the Architecture project](https://devops-server.frontiersin.net/Madrid/Architecture)
  + Once you're signed-in into the VPN you can clone via https
* At Frontiers the main cloud provider is [Azure](https://portal.azure.com/#home), it is managed by Operations, however at Braavos we mainly use [GCP](https://console.cloud.google.com/welcome?project=frontiers-ocean&pli=1), which is managed by us.
  + Operations will provide you access to Azure as part of your onboarding, while you can ask the team to grant you access to GCP
* [Jira](https://jira.frontiersin.net/secure/RapidBoard.jspa?rapidView=799&view=detail&selectedIssue=OCN-248)
* [Confluence](https://confluence.frontiersin.net/pages/viewpage.action?pageId=108796773) (VPN on)
* Main GCP services used at Braavos:
  + BigQuery
  + IAM
  + Cloud Compute
  + Cloud Build
  + Cloud Composer (managed Apache Airflow)
  + Cloud Logging
  + Cloud Storage
* At frontiers we use a [Y42](https://www.y42.com/), this is the [link to access.](https://app.y42.com/organization/frontiers/integrations/list) You can request an account to the team.
* At Frontiers, data is shared across departments, teams using Kafka topics and Kafka Connect to sink those topics. [Some information about the Kafka infrastructure and links to the Kafka environments.](https://confluence.frontiersin.net/pages/viewpage.action?spaceKey=TECH&title=Kafka+-+Infrastructure)
* The [ConfigurationHub](https://configurationhub.frontiersin.net/components?environmentId=172&applicationId=1340" \t "_blank) for Kafka Connect connectors configurations
* For logs of applications running in K8s:
  + Non-production: you can access directly the pods and consult the logs
  + Production: you can check [NewRelic](https://confluence.frontiersin.net/pages/viewpage.action?pageId=56527469" \t "_blank)
* For sketching we use both [Lucida](https://lucid.app/documents#/dashboard?folder_id=home) (licensed, sign-in with the .org account) and [Miro](https://miro.com/app/dashboard/) (free account)

**Useful documents**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/onboarding/#useful-documents)

Note that some of these documents are not in a final state as of now and everything will be restructured as part of the standardisation process. If you have doubts it is always good to check with the team.

* [Braavos Landscape](https://lucid.app/lucidchart/44731d82-29c3-47f8-8f55-7f38f2ece03c/edit?invitationId=inv_3275f3f7-e234-43d4-9673-816334acc07c&page=0_0)
* [Braavos members skills and attitude](https://miro.com/app/board/uXjVPcBQXzI=/)
* [Braavos & Ocean vs Frontiers relationship](https://miro.com/app/board/uXjVP66HjF0=/?userEmail=davide.dimatteo@frontiersin.org&track=true&utm_source=notification&utm_medium=email&utm_campaign=add-to-board&utm_content=go-to-board)
* [Frontiers Technology landscape](https://lucid.app/lucidchart/5602f70b-d03d-4fe9-8b9a-73bb0e35d348/edit?page=0_0)
* [GCP IAM Permission flowchart](https://lucid.app/lucidchart/9460da97-488a-4e14-a603-d68fddb577fe/edit?invitationId=inv_390011c1-3715-49d9-9c5e-ddf57b100d3c&page=0_0)
* [GCP projects organisation](https://confluence.frontiersin.net/pages/viewpage.action?spaceKey=TECH&title=%5BFrontiers+Ocean%5D+-+1.+Projects+and+Organization)
* [Ocean home](https://confluence.frontiersin.net/pages/viewpage.action?pageId=87163466)
* [Ocean learning path](https://confluence.frontiersin.net/display/TECH/%5BFrontiers+Ocean%5D+-+Learning+Path)

**Way of working**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/onboarding/#way-of-working)

At the moment, the team works on two different areas with two different ways of working:

* [Roadmap items, arranged in sprints following SCRUM](https://jira.frontiersin.net/secure/RapidBoard.jspa?rapidView=799&view=detail&selectedIssue=OCN-248)
* [Support items, following KANBAN](https://jira.frontiersin.net/secure/RapidBoard.jspa?rapidView=800&view=detail&selectedIssue=OCN-116)

By now we will share 50/50 the time between support and roadmap, adding more time to sprint gradually until we reach a 70/30 or 80/20 when we have the tools and we have educated the users.

* 3 weeks sprints
* Story points as a base for estimations, being 1, 3, 5, 8 the equivalent to 1 or less days, 1-2 days, a week, and two weeks
* Team capacity will be the 50% of the story points by sprint multiplied by the team members in the sprint. This can change with the decision of adding the support to the sprint or not, as if added then the capacity should be adapted
* Ceremonies: Internal sprint review, Thursday afternoon before the end of the 3 week period. Sprint planning, Friday morning. Daily standup, everyday at 9:30 except Thursday in which we meet with the affiliated team Damas.

**Technology**

**Stack**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/tech-stack/#stack)

* 📦 EL/rEL
* 💾 Storage
* 🧰 Transform
* 🕹️ Data Ops
* Files, DBs, APIs: [Airbyte Open Source](https://docs.airbyte.com/" \t "_blank).
* Kafka: [Kafka Connect](https://docs.confluent.io/platform/current/connect/concepts.html).
* Custom: [Google Functions](https://cloud.google.com/functions).
  + Python 3.9+

**Stack**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/tech-stack/#stack)

* 📦 EL/rEL
* 💾 Storage
* 🧰 Transform
* 🕹️ Data Ops
* Data Warehouse: [Google BigQuery](https://cloud.google.com/bigquery)
* Object Store: [Google Cloud Storage](https://cloud.google.com/storage) (intake zone for raw / unstructured data)
* Orchestration: [Cloud Composer](https://cloud.google.com/composer)
* Secret Manager: [Google Secret Manager](https://cloud.google.com/secret-manager)
* CI/CD:
  + [Azure DevOps](https://devops-server.frontiersin.net/Madrid/Architecture/_git/ocean) as git platform, CI checks and CD for Azure-hosted services (e.g.: kafka connectors).
  + [Google Cloud Build](https://cloud.google.com/build) for CD for GCP-hosted services.
* Sanity/Quality checks runtime: [Google Functions](https://cloud.google.com/functions).

**Google Cloud Projects**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/tech-stack/#google-cloud-projects)

INFO

In Ocean Breeze there is no need for a separate infrastructure project.

| **#** | **GCP Project** | | |
| --- | --- | --- | --- |
| **ocean-tech-braavos** | **ocean-tech-braavos-dev** | **ocean-tech-braavos-pro** |
| Scope | Consume | Produce | Produce |
| Environment | Playground | dev | pro |
| Use cases | Playground, PoC, research | Dev env for new data assets | Production data warehouse, to feed Analytics Hub |
| Airflow | No | Yes (Cloud Composer) | Yes (Cloud Composer) |
| Airbyte | No | Yes (Compute Engine) | Yes (Compute Engine) |
| GCP Secret Manager | No | Yes | Yes |
| Alerts | No | No | Yes |
| Raw bucket | No | [Yes](https://console.cloud.google.com/storage/browser/ocean-tech-braavos-dev-raw) | [Yes](https://console.cloud.google.com/storage/browser/ocean-tech-braavos-pro-raw) |

**Azure Environments**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/tech-stack/#azure-environments)

INFO

Only for internal data integrations, such as **Kafka Connect** deployments.

Frontiers primarily uses Azure as its cloud provider, leading to the deployment of certain internal integrations on the Azure platform. In these instances, we exclusively utilize 2 designated Azure environments, referred to as "Frontiers Shared Services," where data sources are typically hosted.

| **Azure Subscription** | **Azure AKS Cluster** | **Corresponding GCP Project** | **Scope** |
| --- | --- | --- | --- |
| Shared Services Staging | [SharedServicesStagingAKS](https://portal.azure.com/#@frontiersin.net/resource/subscriptions/a3f3ca84-978f-4446-95e3-c91cfebd919d/resourceGroups/shared-services-staging-aks/) | ocean-tech-braavos-dev | Develop new integratiopns between Frontiers Shared Services and Ocean. |
| Shared Services Production | [SharedServicesProductionAKS](https://portal.azure.com/#@frontiersin.net/resource/subscriptions/17497965-f3cd-402e-be8d-27f03c6759bf/resourceGroups/shared-services-production-aks/) | ocean-tech-braavos-pro | Production-ready integrations between Frontiers Shared Services and Ocean. |

# Standard pipeline

We will go step by step now providing more information about each one.

## Extract and Load[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/standard-pipeline/#extract-and-load)

### Create dataset step[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/standard-pipeline/#create-dataset-step)

First, we can add an additional step to verify that the landing dataset exist, and if not, create it.

create\_landing\_dataset = BigQueryCreateEmptyDatasetOperator(  
 task\_id="create\_landing\_dataset",  
 dataset\_id=f"{SOURCE\_NAME}\_{SOURCE\_VERSION}\_0lan",  
 location="EU",  
 project\_id=PROJECT\_ID,  
)

### Extract and load[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/standard-pipeline/#extract-and-load-1)

* Airbyte
* Load Job
* Kafka

Using airbyte we need to do so in two steps: first we create the airbyte trigger for starting the sync process and then we have to use a sensor to continue when the process has finished. You can see an example here:

async\_airbyte\_trigger = AirbyteTriggerSyncOperator(  
 task\_id="airbyte\_sync\_" + SOURCE\_NAME,  
 airbyte\_conn\_id="airbyte\_default",  
 connection\_id=AIRBYTE\_CONNECTION\_IDS[ENV],  
 asynchronous=True,  
 do\_xcom\_push=True,  
)  
  
airbyte\_sensor = AirbyteJobSensor(  
 task\_id="airbyte\_sensor\_" + SOURCE\_NAME,  
 airbyte\_conn\_id="airbyte\_default",  
 airbyte\_job\_id=async\_airbyte\_trigger.output,  
)

## Prechecks[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/standard-pipeline/#prechecks)

We do our prechecks with DBT, having the following tests as common for all sources. To do so, we have created a custom DBTOperator that can be found in common.operators.DbtOperator

Code Example:

prechecks = DbtOperator(  
 task\_id="dbt\_prechecks",  
 dbt\_operation="build",  
 model="source:your\_precheck\_model",  
 target="prechecks",  
 git\_branch=GIT\_BRANCH,  
 git\_repo="ocean",  
 *# with this operation we load the csv files in a gcp folder locally to use with duckdb*  
 gcloud\_operation=f"mkdir data && gsutil -m cp -r {FOLDER}\*.csv data/",  
 install\_duckdb=True,  
)

## Transform[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/standard-pipeline/#transform)

In this step we use DBT to do the needed transformations in our data. We will use again the same DbtOperator, changing the model for the transform model and using a **run** operation

dbt\_transform = DbtOperator(  
 task\_id="dbt\_transform",  
 dbt\_operation="run",  
 model="source:your\_transform\_model",  
 target=ENV,  
 git\_branch=GIT\_BRANCH,  
 git\_repo="ocean",  
)

## Postchecks[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/standard-pipeline/" \l "postchecks" \o "Direct link to Postchecks)

After the transformations, we run our last test checks to ensure that we have the quality needed in the dataset to publish in the Analytics Hub. We use again DBT with the DBTOperator, but we will use the test command again.

dbt\_postchecks = DbtOperator(  
 task\_id="dbt\_postchecks",  
 dbt\_operation="test",  
 model="source:your\_postcheck\_model",  
 target=ENV,  
 git\_branch=GIT\_BRANCH,  
 git\_repo="ocean",  
)

## Publish[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/standard-pipeline/#publish)

After all of this steps, we are ready to publish the final dataset. We will need to create the final dataset with the current version and the copy the tables into that version.

### Create final dataset[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/standard-pipeline/#create-final-dataset)

create\_final\_dataset = BigQueryCreateEmptyDatasetOperator(  
 task\_id="create\_final\_dataset", dataset\_id=SOURCE\_FINAL\_DATASET, location="EU", project\_id=PROJECT\_ID  
)

### Copy tables into final dataset[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/standard-pipeline/#copy-tables-into-final-dataset)

We have create also a custom operator here to make our process more agile since there are some defaults that are common on all the pipelines. The operator can be found in common.tasks.bq\_insert\_job\_copy\_table.

*# Copy the tables to the final dataset*  
with TaskGroup("bq\_copy\_tables", tooltip="Copy tables to the final dataset") as bq\_copy\_tables:  
 for table\_name in TABLE\_LIST:  
 bq\_insert\_job\_copy\_table(  
 table\_name=table\_name,  
 project=PROJECT\_ID,  
 source\_dataset=SOURCE\_FINAL\_DATASET + "\_qa",  
 dest\_dataset=SOURCE\_FINAL\_DATASET,  
 )

## Notifications, State Update, Trigger custom DAGS[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/standard-pipeline/#notifications-state-update-trigger-custom-dags)

In the final step, we need to update our metadata so we can have updated documentation and, in case that we need to, we will trigger other DAGs from other teams and notify other teams.

The metadata update task is defined also in common.tasks.bq\_stats\_task

bq\_stats = bq\_stats\_task(project=PROJECT\_ID, dataset=SOURCE\_FINAL\_DATASET)

**Airbyte**

**What is Airbyte?**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airbyte/#what-is-airbyte)

[**Airbyte**](https://airbyte.com/) is our general-purpose **el** (*extract/load*) or **ingestion** tool. We'll use Airbyte for integrations such as SQL systems, files (e.g.: "S3") or APIs.

There are two versions of Airbyte: Airbyte Cloud, which is a fully-managed PaaS, and Airbyte Open Source, which can be self-hosted. In our case we have the open-source version, self-hosted in GCP.

**User guide**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airbyte/#user-guide)

**Web UI**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airbyte/#web-ui)

We are utilizing the open source version on Compute Engine. To securely access Airbyte, we recommend using the gcloud tool as demonstrated below to create a tunnel from your laptop:

* ✏️ Dev
* 🔒 Pro

gcloud

*# Using gcloud configurations is recommended*  
gcloud config configurations activate pro  
  
*# Open the tunnel*  
gcloud compute ssh ocean-airbyte-pro -- -L 8000:localhost:8000 -N -f

TIP

Open [**http://localhost:8000/**](http://localhost:8000/) to access the Airbyte UI.

* Username: airbyte
* Password: check ocean-airbyte-{env}-password in Google Secret Manager

CAUTION

Google creates by default two firewall rules that allow ssh and rdp traffic from any source. With terraform, we disable these rules. <https://console.cloud.google.com/networking/firewalls/>

* default-allow-ssh disabled
* default-allow-rdp disabled

To gather external connectivity from your laptops, you'll need to add a rule to authorize our public IP.

* allow-{your-name}
  + ports 22
  + targets: [airbyte, airflow]
  + source: ip\_ranges = YOUR\_IP

**Command line**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airbyte/#command-line)

Airbyte is equipped with a command-line interface called octavia-cli, which can be installed from the [official Airbyte GitHub repository](https://github.com/airbytehq/airbyte/blob/master/octavia-cli/README.md#install). This user-friendly tool allows users to define Airbyte sources, destinations, and connections using YAML templates, and provides a convenient method to apply these configurations.

Additionally, octavia-cli supports the exporting of resources created through the UI. This opens up a **streamlined workflow**: users can utilize the UI to set up and test their configurations, and once they are confident that everything is working correctly, they can easily export the templates and add them to their repository.

The Airbyte templates will be hosted in the airbyte folder in our monorepo. The below caption show how to initialize octavia:

octavia-cli init

$ cd airbyte  
$ ./octavia.sh init  
🐙 - Octavia is targeting your Airbyte instance running at localhost:8000  
on workspace xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.  
🐙 - Project is not yet initialized.  
🔨 - Initializing the project.  
✅ - Created the following directories: connections, destinations, sources.  
✅ - Created API HTTP headers file in api\_http\_headers.yaml

TIP

The **octavia.sh** script is a helper to run octavia-cli in a docker container.

Let's say we have created a source named "Postgres" through the UI and we want to start managing it (or back it up) as a octave-cli template. We could easily do it with the import command:

$ ./octavia.sh import source "Postgres"  
🐙 - Octavia is targeting your Airbyte instance running at localhost:8000  
on workspace xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.  
✅ - Imported source Postgres in sources/postgres/configuration.yaml.   
State stored in sources/postgres/state\_e4afa25d-3189-4a63-be83-50082d467c48.yaml  
⚠️ - Please update any secrets stored in sources/postgres/configuration.yaml

Then, we could modify the template and update Airbyte with the apply command:

$ ./octavia.sh apply -f sources/postgres/configuration.yaml   
🐙 - Octavia is targeting your Airbyte instance running at localhost:8000  
on workspace xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx.  
🐙 - Postgres exists on your Airbyte instance according to your state file,   
let's check if we need to update it!  
😴 - Did not update because no change detected.

**Admin guide**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airbyte/#admin-guide)

**Terraform**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airbyte/#terraform)

Àirbyte is deployed in Google Cloud Compute VMs using terraform. As this is an infrastructure-component, the corresponding terraform project is located in:

\_shared/terraform

The relevevant template in this case is the file airbyte.tf, which uses a side-script airbyte-init-script.sh to install and configure airbyte.

To deploy, just modify the templates and run:

$ terraform workspace select [default|pro]  
$ terraform apply

**Secrets**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airbyte/#secrets)

Airbyte offers the option to utilize [Google Secret Manager](https://console.cloud.google.com/security/secret-manager) to securely store all sensitive information related to sources and destinations. This includes items such as database passwords, which will not be saved in the Airflow state DB but instead, will be securely stored in Google Secret Manager.

To enable this feature, we set specific environment variables:

SECRET\_PERSISTENCE=GOOGLE\_SECRET\_MANAGER  
SECRET\_STORE\_GCP\_PROJECT\_ID=ocean-tech-braavos-{dev|pro}  
SECRET\_STORE\_GCP\_CREDENTIALS={SA\_JSON\_KEY}

**Monitoring**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airbyte/#monitoring)

While most monitoring information is automatically collected, memory and disk monitoring require the installation of the [OpsAgent](https://cloud.google.com/stackdriver/docs/solutions/agents/ops-agent" \t "_blank).

TODO

As per documentation, it looks possible to use OpsAgent OpenTelemetry collector to handle Airbyte metrics:

* <https://docs.airbyte.com/operator-guides/collecting-metrics/>
* <https://cloud.google.com/monitoring/agent/ops-agent>

Example of using the portal to access monitoring:

**Snapshots**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airbyte/#snapshots)

ONLY IN PRO

**Only available for the pro instance.**

The backup mechanism is primarily intended for disaster recovery purposes. In the event that a snapshot needs to be restored, it is important to carefully follow Google's guidelines.

<https://cloud.google.com/compute/docs/disks/restore-snapshot>

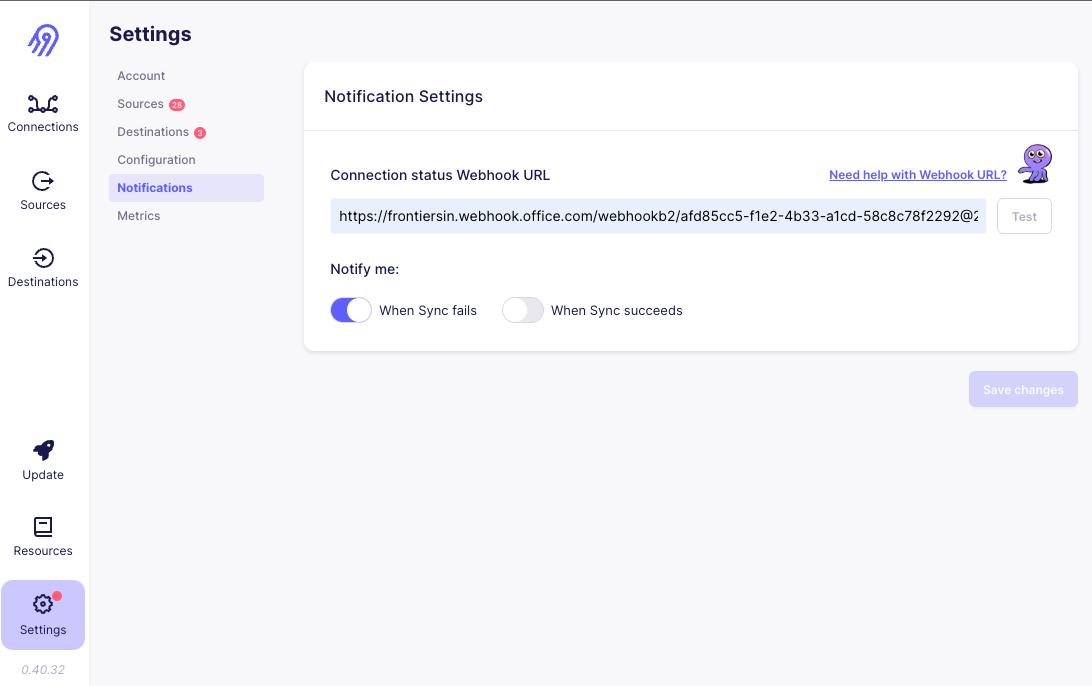
Please also remember to update the documentation or Terraform template as needed.

We use Terraform to configure a weekly snapshot schedule, which backs up the VM disk every Friday at 10:00 PM GMT.

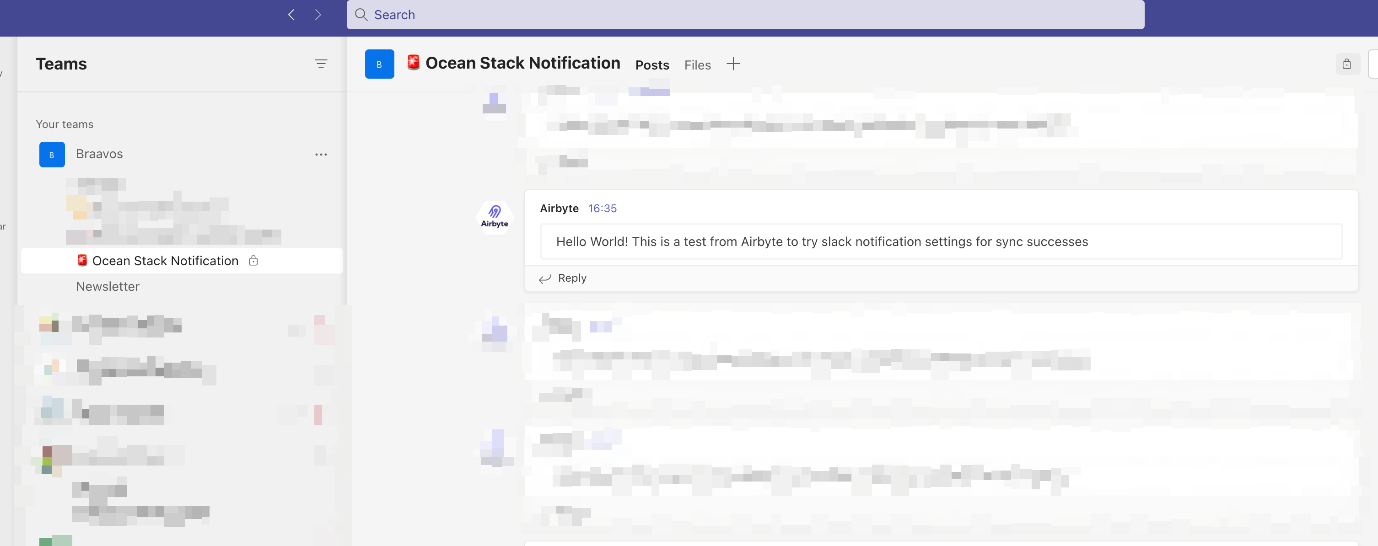
How to access the snapshot schedule through the portal:

**Notifications**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airbyte/#notifications)

Airbyte supports sending notifications via webhook. We have created a dedicated channel for receiving these kind of notifications in the Braavos MS Teams.



Example in MS Teams:



**Change UI sidebar color[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airbyte/" \l "change-ui-sidebar-color" \o "Direct link to Change UI sidebar color)**

To better distinguish between the different Airbyte instances, we have changed the sidebar color of Airbyte to match the colors of our 2 Airflow instances:

* dev: 🟢 #8CC084
* pro: 🔴 #E94F37

In order to color the UI, we edited the .css file related to the Sidebar element of the Airbyte UI. This is needed every time we do a fresh-install of Airbyte on the VM. You can reproduce the change by following the steps below:

* ✏️ Dev
* 🔒 Pro

1. SSH into the Airbyte Dev VM: gcloud compute ssh ocean-airbyte-dev
2. SSH into the Airbyte-webapp container: docker exec -it airbyte-webapp /bin/sh
3. locate the .css file: find / -iname 'SideBar\*' (which will result in something like */usr/share/nginx/html/assets/SideBar-3d457535.css*)
4. edit the .css file: vi /usr/share/nginx/html/assets/SideBar-3d457535.css
5. locate the background-color property of the *.SideBar-module\_\_nav* element and change the value to #8CC084. Note that if you are not sure about the name of the element, you can *inspect* our Airbyte UI from a browser and find it out.

**Pending stuff**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airbyte/#pending-stuff)

TODO

* Alarms
* Airbyte syncs logs
* Airbyte Open telemetry
* Different ports for dev and pro

**Airflow**

**What is Airflow?**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airflow/#what-is-airflow)

[**Apache Airflow**](https://airflow.apache.org/docs/apache-airflow/stable/) is our **orchestation** tool. Airflow is an open-source platform for developing, scheduling, and monitoring batch-oriented workflows.

There are several fully-managed PaaS solutions to use Airflow, and also it can be self-deployable in multiple ways. In our case, we will rely mostly on a fully-managed service:

* For pro and dev we'll use [Google Cloud Composer](https://cloud.google.com/composer), the Google's fully-managed Airflow solution.
* For local, we'll use docker-compose.

**User guide**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airflow/#user-guide)

**Web UI**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airflow/#web-ui)

* 🏡 Local
* ✏️ Dev
* 🔒 Pro

Inside the monorepo there is a convenient docker-compose file that allows you to run Airflow locally.

local

*# See https://cloud.google.com/sdk/gcloud/reference/auth/application-default/login*  
gcloud auth application-default login  
  
*# See https://airflow.apache.org/docs/apache-airflow/stable/howto/docker-compose/index.html*  
*# from our monorepo*  
cd airflow  
mkdir logs  
docker compose up airflow-init *# first time only*  
docker compose up

TIP

Open [**http://localhost:8080/**](http://localhost:8080/) to access the Airflow UI.

* Username: airflow
* Password: airflow

**DAG Deployment**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airflow/#dag-deployment)

Airflow DAGs are Python source files that are stored in our monorepo under the airflow/dags folder. In order to deploy the DAGs, it is necessary to copy the source files to a bucket. The Airflow environment is then synced to the bucket. Typically, it takes up to 5 minutes for Airflow to discover any changes that were made in the bucket.

To automate the upload of the source files to the bucket, we have a Terraform template that streamlines the process. This template is aware of the environment and ensures that the source files are synced to the correct bucket. With this approach, the upload process becomes automated and eliminates the need for manual intervention.

DAG Deployment example with Terraform

$ cd terraform/\_airflow\_dags  
$ terraform workspace select default # dev environment  
$ terraform apply

CAUTION

When multiple people are working (and deploying) on Airflow Dev at the same time, it is important to consider that it is possible to overwrite each other's changes if the deployments occur in the same ~5 minutes window. This is because the DAGs are deployed to the same bucket.

**Dev guide**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airflow/#dev-guide)

**Example DAGs**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airflow/#example-dags)

Check airflow/dags/examples for example DAGs for common tasks.

| **Source** | **Description** |
| --- | --- |
| airbyte\_async.py | How to trigger an Airbyte connection asynchronously |
| email.py | How to send emails with SendGrid connection |

**Custom operators**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/airflow/#custom-operators)

To find custom operators designed for reusability and standardization, check out the airflow/dags/frontiers directory.

| **Operator** | **Description** |
| --- | --- |
| CloudBuildOperator | Custom Cloud Build operator that gathers and shows the build output logs directly in Airflow |
| DbtOperator | Run dbt workloads in GCP using Cloud Build as the runtime |
| GoogleCloudFunctionOperator | Custom Cloud Function operator with built-in auth |
| KafkaCheckConsumerGroupOperator | Checks a Kafka Consumer Group health |

**BigQuery**

**What is BigQuery?**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/bigquery/#what-is-bigquery)

[**BigQuery**](https://cloud.google.com/bigquery/docs/introduction) is our **data warehouse** for **analytical** use cases. BigQuery is fully-managed and serverless, so zero infrastructure management.

**Conventions**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/bigquery/#conventions)

In our ocean-tech-braavos-\* environments, we'll use the below described conventions.

**Dataset versioning**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/bigquery/#dataset-versioning)

All datasets will be versioned. We will support a majorVersion and a minorVersion:

* majorVersion = n, being n a **natural number** (starting from 1). We'll increase this version any time there is a breaking change in a dataset (e.g.: changing the schema in a backwards-incompatible fashion).
* minorVersion = m, being m a **whole number** (starting from 0). We'll increase this version any time there is a no-breaking change in a dataset (e.g.: adding a new nullable column to a table).

Example: versioning changelog

[1.0] 🎉 New datasource  
[1.1] ✨ Added 1 new optional column  
[2.0] 🔥 Removed 1 column (potential impact on consumers)  
[2.1] ✨ Added 1 new table (no impact on the others)  
[3.0] 🔥 Remove 1 table (potential impact on consumers)

**Dataset naming**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/bigquery/#dataset-naming)

We'll use the following naming convention for datasets:

{source}\_{layerId}\_{versionMajor}\_{versionMinor}

where:

* **{source}**: the data source main id, gathered from the Data Directory.
* **{layerid}**: according to our [layers](https://ocean-docs.frontiers-ss-dev.info/project-management/how-data-is-layered/), must be one of: 0lan, 1int, 2std, 3con or 4enr.
* **{versionMajor}** and **{versionMinor}**: as described above.

Example: dataset naming

*# ringgold*  
ringgold\_2std\_1\_0  
  
*# openalex*  
openalex\_0lan\_1\_0  
openalex\_2std\_1\_0  
  
*# researcher profiles*  
researcherprofiles\_4enr\_1\_0

DANGER

TODO

**Check BigQuery audit logs**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/bigquery/#check-bigquery-audit-logs)

FOR FRONTIERS OCEAN USERS

In order to track and monitor the queries, you can check the BigQuery audit logs. With the help of logs, you can troubleshoot issues, optimize your queries and understand the data usage patterns. The audit logs are stored in different datasets depending on the project:

| **Project** | **Dataset with the logs** |
| --- | --- |
| frontiers-ocean | frontiers-infraestructure.bq\_audit\_logs |
| frontiers-ocean-exports | frontiers-infraestructure.bq\_audit\_logs\_exports |
| frontiers-ocean-sandbox | frontiers-infraestructure.bq\_audit\_logs\_sandbox |
| frontiers-exports-sandbox | frontiers-infraestructure.bq\_audit\_logs\_sandbox\_exports |

Inside the frontiers-infraestructure project there are some saved queries that can be useful to understand data usage patterns. You can copy and modify them accordingly, by changing the dataset, principal email, and other conditions to match your specific use case.

Example queries:

* Who is responsible for heavy queries?

SELECT  
 principalEmail,  
 totalbilledgigabytes,  
 jobChange.jobConfig.queryConfig.query,  
 eventTimestamp  
FROM  
 `frontiers-infraestructure.{dataset\_name}.bigquery\_audit\_logs\_v2`  
WHERE  
 eventTimestamp > TIMESTAMP\_SUB(CURRENT\_TIMESTAMP(), INTERVAL 60 DAY)  
 AND principalemail NOT LIKE "%gserviceaccount.com"  
ORDER BY  
 totalbilledgigabytes DESC  
LIMIT  
 100;

* Which are the most repeated and expensive queries?

SELECT  
  count(0) as queries,  
  sum(totalbilledgigabytes) as sumtotalbilledgigabytes,  
  sum(estimatedcostusd) as sumestimatedcostusd,  
  ANY\_VALUE(jobChange.jobConfig.queryConfig.query) as query,  
  MD5(jobChange.jobConfig.queryConfig.query) AS query\_md5  
FROM  
  `frontiers-infraestructure.{dataset\_name}.bigquery\_audit\_logs\_v2`  
WHERE  
  eventTimestamp > TIMESTAMP\_SUB(CURRENT\_TIMESTAMP(), INTERVAL 60 DAY)  
  AND principalemail LIKE "%gserviceaccount.com"  
  AND estimatedcostusd > 0.5  
group by query\_md5  
ORDER BY  
  queries desc

**dbt**

**What is dbt?**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/dbt/#what-is-dbt)

[**dbt**](https://docs.getdbt.com/docs/introduction) is our **transformation** tool. It allows to collaborate on data models, version them, and test and document your queries before safely deploying them to production, with monitoring and visibility.

dbt it not a platform-tool, it is a client-side tool. It just compiles and runs your transformation code against BigQuery.

**Dev guidelines**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/dbt/#dev-guidelines)

**Git**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/dbt/#git)

Our *dbt* project lives within our monorepo, in the dbt folder. As any standard dbt project, it has the following layout:

dbt project layout

📂 ./dbt  
├── 📂 analyses *# sql files with analytical queries - not part of the pipelines (optional)*  
├── 📂 dbt\_project.yml *# dbt main config file*  
├── 📂 macros *# custom macros*   
├── 📂 models *# !! OUR TRANSFORMATION LOGIC GOES HERE!*  
├── packages.yml *# dependencies*  
├── pyproject.toml *# side tools settings, as sqlfmt*  
├── 📂 seeds *# static csv files that needs to be loaded and managed (optional)*  
└── 📂 tests *# custom macros*

**Where do I save my code?**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/dbt/#where-do-i-save-my-code)

Let's say you are working on building the transformation from landing to standardized for a new source, named foo. You would need to create, at least:

* A new directory under dbt/models: dbt/models/foo
* A YAML configuration: dbt/models/foo/foo.yaml. This file would follow the dbt standards and thus describe:
  + **Sources** that your transformation need
  + **Models** (*tables*) you will create, with the corresponding descriptions and metadata.
  + **Tests**
  + Seeds (optional)
  + Exposures (optional)
* One or more SQL files in that directory to define your models. For example:
  + dbt/models/foo/bar1.sql
  + dbt/models/foo/bar2.sql
* (Optional) Custom macros or test would be set in:
  + dbt/macros/foo.sql
  + dbt/tests/foo.sql

**Code formatter and linter**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/dbt/#code-formatter-and-linter)

* Linter: **[sqlfluff](https://www.sqlfluff.com/" \t "_blank)**
* Formatter: **[sqlfmt](http://sqlfmt.com/" \t "_blank)**

Formatting, linting

$ sqlfluff lint dbt/models  
All Finished 📜 🎉!  
  
$ sqlfmt dbt/models/aira\_recommender/  
10 files left unchanged.

**GCP**

**Extra topics**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/gcp/#extra-topics)

**Access management**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/gcp/#access-management)

[Check the Lucid flowchart before starting](https://lucid.app/lucidchart/9460da97-488a-4e14-a603-d68fddb577fe/edit?page=0_0&invitationId=inv_390011c1-3715-49d9-9c5e-ddf57b100d3c)

* Accesses are usually managed at the Google Group level and we should try to enforce this type of hierarchy. If you need to request a new Google Group, ping *John Tecu* from IT.
* Then assign the requesting user to that specific Google Group.
* Each Google Group can have 1:N IAM Roles.
* Each Role has 1:N policies (**permissions** in GCP terms)

**Access management Ocean Breeze**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/gcp/#access-management-ocean-breeze)

Our aim is to establish consistency in the process of creating new GCP projects. In our project, Ocean Breeze, we have decided to create multiple projects for each team. However, the setup process entails various configurations that must be performed prior to release the project to the final users. While some of these steps can be automated, a few others require manual intervention.

* Create the folder and the project manually by the UI.
* Run the terraform template with the new project (/terraform/foundation/)
* Set the audit log for the new project (/src/foundation/)
* Ask Enrique De Prada to attach the billing to the new project.

**Manual Steps**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/gcp/#manual-steps)

**Project Creation**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/gcp/#project-creation)

The manual creation of the projects and folders is mandatory. To create the folder:

To create the project:

Although Terraform can be used to automate this activity, it is a critical activity since the accidental deletion of projects could always happen. Moreover, Terraform would overwrite all the project information, including that of billing which the Braavos team is not granted to modify.

**Audit log**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/gcp/#audit-log)

In the project *ocean-tech-braavos-aud*, there will be an audit dataset for each project. It is possible to follow [this guide](https://github.com/GoogleCloudPlatform/bigquery-utils/blob/master/views/audit/README.md#getting-started-with-bigquery_audit_logs_v2sql).

The creation of the audit sink and of the audit dataset in the our audit project are done with Terraform. This sink create automatically two tables inside the audit dataset of the project. We need to wait for it or we could also make some actions on the new project to generate logs.

Once completed, you can generate the resultant View - bigquery\_audit\_logs\_v2, with this command:

bash create\_view.sh bigquery\_audit\_logs\_v2.sql ocean-tech-braavos-aud <audit-dataset-name>

This View is used for analyzing audit logs. Please note that both tables, *cloudaudit\_googleapis\_com\_activity* and *cloudaudit\_googleapis\_com\_data\_access*, must be created prior to generating the final View.

**Automated Steps**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/gcp/#automated-steps)

**Terraform**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/gcp/#terraform)

Terraform can automate several steps involved in the process. Each step has its corresponding .tf file. If an explicit order is not placed, adding the name of the new project in the var *projects\_map* in var.tf file and applying it suffices.

# Pipelines

**Extraction and load**

Depending on the source type, we use different tools for the extraction and load step:

| **Source type** | **Tool** | **Reason** | **Example** |
| --- | --- | --- | --- |
| CSV files | BQ load | Simplicity and efficiency, as the files are stored in GCP buckets. | [Ringgold](https://ocean-docs.frontiers-ss-dev.info/next/data-directory/data-sources/ringgold/general-info/) |
| PostgreSQL | Airbyte | Simplicity, easy to configure data sources and connectors without having to write any code. | [Quality Funnel](https://ocean-docs.frontiers-ss-dev.info/next/data-directory/data-sources/quality-funnel/general-info/) |
| API to download files | Custom script to download the files to GCP bucket + BQ load | Custom script is needed for each source in order to download the files. Once the files are in the GCP bucket, it's simple and efficient to load them using BQ load. | [Qualtrics](https://ocean-docs.frontiers-ss-dev.info/next/data-directory/data-sources/qualtrics/general-info/) |
| Streaming | Kafka | Simplicity, all the configuration is done on the [Kafka Connector](https://confluence.frontiersin.net/display/TECH/Kafka+-+Infrastructure) which pushes data on our BQ, it will allow getting real-time information | FrontiersGraph |

**Pre-Checks**

The pre-checks are defined as the checks performed on the original data before transferring it anywhere or undergoing any kind of transformation. Those are the pre-checks that we apply depending on the source type:

**CSV files**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/pipelines/pre-checks/#csv-files)

* GCS object updated: check if the input file has been updated in the GCS bucket.
* DBT expectations tests on every table (applied on the CSV files using DuckDB):
  + expect\_table\_row\_count\_to\_be\_between: verify that the number of rows is equal or greater than a minimum value, which is a rough estimation of the expected number of rows.
  + expect\_table\_columns\_to\_match\_ordered\_list: verify that the columns match the specified list. (TODO: apply this check also in Qualtrics)
  + expect\_column\_values\_to\_be\_of\_type: verify that fields are of the expected data types. (TODO: decide if we remove the data type checks from the pre-checks and in that case remove them from PubPeer and Ringgold)

Example: [Ringgold](https://ocean-docs.frontiers-ss-dev.info/next/data-directory/data-sources/ringgold/general-info/)

**PostgreSQL**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/pipelines/pre-checks/#postgresql)

TBD

Example: [Quality Funnel](https://ocean-docs.frontiers-ss-dev.info/next/data-directory/data-sources/quality-funnel/general-info/)

**Kafka**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/pipelines/pre-checks/#kafka)

* KafkaCheckConsumerGroupOperator: custom operator to verify the status of the Kafka topics and check the lag between what we have on BigQuery and Kafka.
* BigQueryTableExistenceSensor: check that all the expected tables exist.
* DBT expectations tests on every table (applied on landing dataset using DuckDB):
  + expect\_table\_columns\_to\_match\_ordered\_list: verify that the columns match the specified list.

Example: FrontiersGraph

**Custom checks**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/pipelines/pre-checks/#custom-checks)

Depending on the source, some custom checks might be needed. Examples:

* Example: [Qualtrics](https://ocean-docs.frontiers-ss-dev.info/next/data-directory/data-sources/qualtrics/general-info/): A check is needed in order to ensure that the definition table has same number of rows as all other tables (1 survey > 1 row in definition table > 1 CSV file).

**Transformations**

The transformations are defined as the operations performed on the data in order to make it ready before publishing.

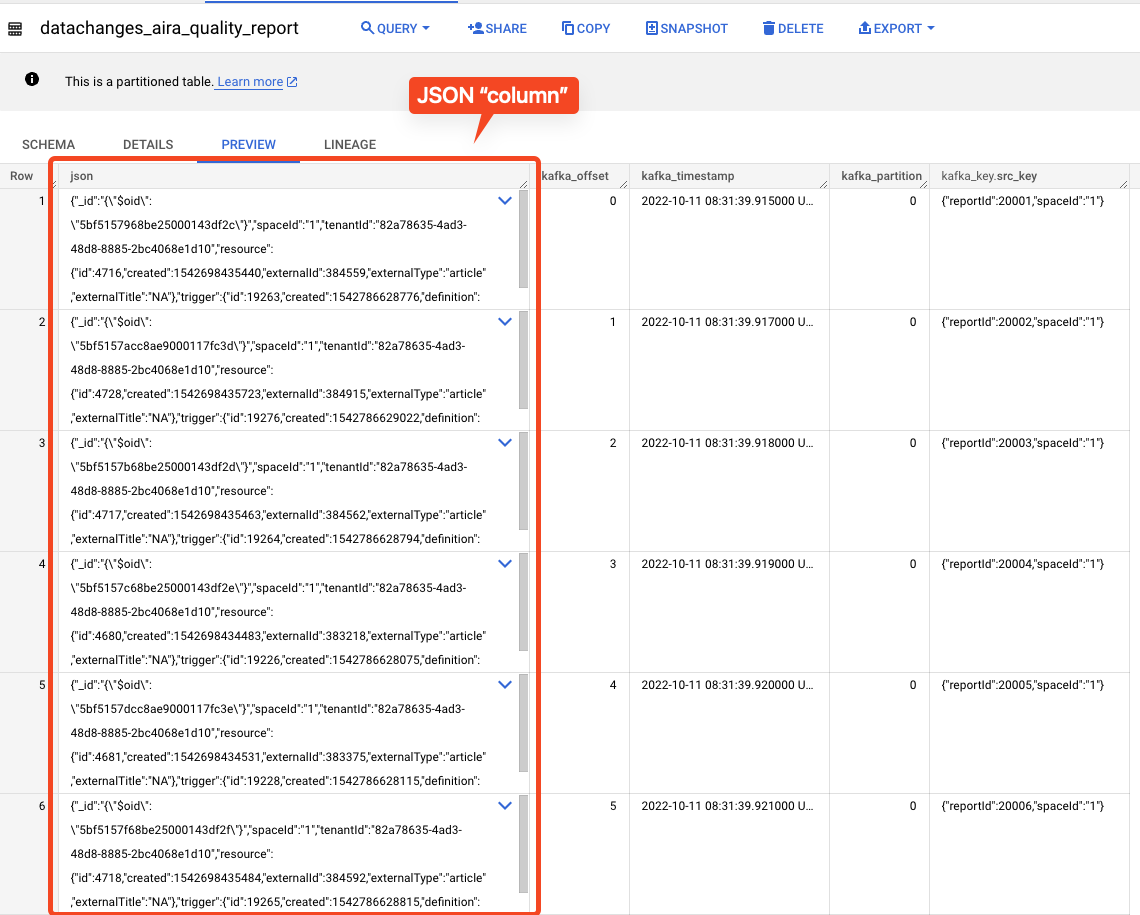
We use *dbt* models defined in dbt/models/ to perform the transformations. All of our transformation steps are applied to individual fields and involve mostly casting *some* fields to a better type. More specifically:

* casting numerical IDs to Integer;
* casting Yes/No fields to Boolean;
* (parse and) casting dates or timestamp fields to Date/Timestamp;
* casting numerical fields (metrics usually) to each corresponding type (Float/Integer usually);
* in a few rare occasions we also: split some fields.

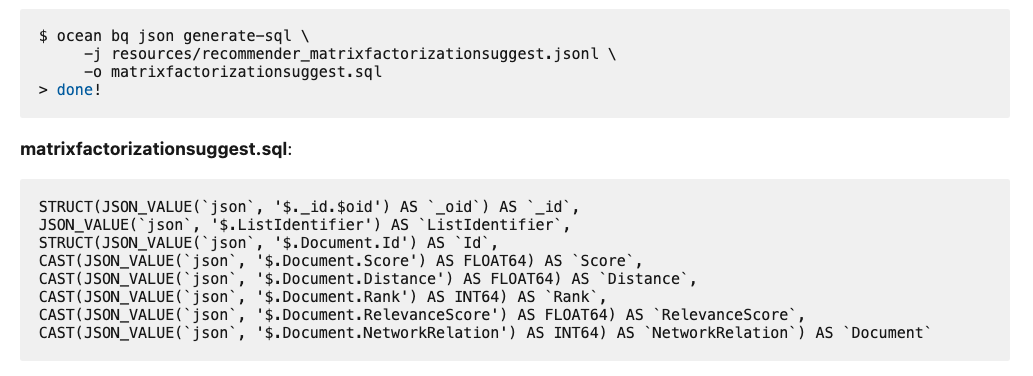
**JSON**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/pipelines/transformations/#json)

The above assumes that the data-to-be-transformed is already in a BigQuery native tabular shape. If not, for example when dealing with semi-structured data such as JSON, we follow the below high-level steps:

1. After the Extract and Load stage, the source JSON documents should be loaded as a STRING column in a landing BigQuery table.



1. Our transformation step should then parse the JSON documents into a proper tabular shape. Depending on the structure of the JSON documents, this step might be as simple as using a few JSON\_EXTRACT functions or as complex as be almost impossible to be done manually. In the latter case, we use our ocean cli tool to parse a sample of the JSON documents and to automatically generate the required SQL code. (Reference: [BigQuery JSON Functions](https://cloud.google.com/bigquery/docs/reference/standard-sql/json_functions" \t "_blank)).



1. Using the auto-generated SQL code , we define a dbt model as normal. Please notice that the generated code is only the SQL SELECT CLAUSE and not the full SQL code. We still need to define the FROM and the WHERE clauses manually.

**Post-Checks**

The post-checks are defined as the checks performed on the data once it has been transferred and transformed. Those are the post-checks that are applied right now:

* DBT tests:
  + not null: verify that fields that are not expected to have null values do not contain any.
  + unique: verify that fields that are not expected to have duplicates only contain unique values.
  + dbt\_expectations.expect\_column\_values\_to\_be\_of\_type: verify that fields are of the expected data types (TODO: only applied to some sources and to some fields, decide if we should apply it to all of them).

# Publishing

After all the postchecks have been successful, we have to deploy the final version in BigQuery. To do so, what we need to do is to copy the tables from the QA dataset that we were using into the final dataset, the one that then will be pushed into Analitycs Hub. This is done with the Operator that we built, to copy all the tables.

Publishing into the Analytics Hub, however, is not done in the pipeline. We will do this manually after we verify that everything is correct.

**Last steps**

**Update metadata**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/pipelines/last-steps/#update-metadata)

When we finish with the pipelines, we need to update some metadata in the Airflow DAGs from BigQuery so we can then use this with other parts of our stack, like for example, when using the badges in this documentation.

We are currently gathering the following data and adding it as XCOMs data on the DAG run:

* Table count
* Row count
* Size in bytes
* Last modified time

More information can be found about the badges in the following [page](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/toolbox/badges/).

**Notifications**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/pipelines/last-steps/#notifications)

If we need to notify other teams or anyone external to or team we can add a notifications step here.

**DAG Triggering**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/pipelines/last-steps/#dag-triggering)

After the updated/final dataset is ready, we may have some other datasets that depend on this one, so we can use the [Airflow Trigger DAG operator](https://airflow.apache.org/docs/apache-airflow/stable/_api/airflow/operators/trigger_dagrun/index.html#module-airflow.operators.trigger_dagrun) to do so.

**Documentation**

**Organization**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/pipelines/documentation/#organization)

Each pipeline has to be *well documentated* before being published. All the information about it has to be stored under the folder *2-data-sources* with the following structure.

Organization of Pipeline Documentation

📂 frontiers-ocean-breeze  
├── 📁 docusaurus  
│ └── 📁 3-data-directory  
│ └── 📁 2-data-sources  
│ └── 📁 <source-name>  
│ ├── \_category\_.json *#label about the source name*  
│ ├── 1-general-info.md *#info about the sources*  
│ └── 📁 datasets  
│ ├── \_category\_.json *#label Datasets*  
│ └── 📁 <version>  
│ ├── \_category\_.json *#label version*  
│ ├── bqmetadata.yaml *#retrieve info from*  
│ └── <source-name>\_<version>.md *#created with the ocean tool*

The files to create are:

* ***category*.json**: in each folder there is one of this file providing more details about the folder structure. It is also used to propagate the section concept to Docusaurus and to the Confluence documentation sync tool.
* **1-general-info.md**: this file has to manually be written with the specific information on the sources with at least two mandatory sections *General information* and *What can I find in <Source-name>*
* **bqmetadata.yaml**: it is possible to use this file to load the description in the BigQuery dataset with the [*ocean bq metadata* command](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/toolbox/ocean-cli/commands/bigquery/#metadata-management)
* **<source-name>\_<version>.md**: it can be generated automatically with the [command *ocean docs*](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/toolbox/ocean-cli/commands/docs/#docs-generator)

**Future Improvements**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/pipelines/documentation/#future-improvements)

In the next weeks these tools will be improved with some new features. Some improvements are:

* update the *ocean docs generate* command to get information also about the nested fields
* update the *ocean docs generate* command to add relationship between tables
* improve ER diagram (zoom,details)
* create automatically all the folder structure

**Python**

We currently use Python **3.10**. It is recommended to use [pyenv](https://github.com/pyenv/pyenv" \t "_blank) to install and manage multiple Python runtimes on your machine.

It is recommended to create a Virtual Environment inside the project directory to isolate the project dependencies. Pyenv works really well with [pyenv-virtualenv](https://github.com/pyenv/pyenv-virtualenv" \t "_blank). You can create a virtual environment named ocean-3.10.10 running Python 3.10.10 as follows:

pyenv virtualenv 3.10.10 ocean-3.10.10

DANGER

For this to work, you need to have a Python 3.10.10 runtime installed via pyenv on your machine.

Both tools interact really well; in fact, you can seamlessly switch between virtual environments when navigating your file-system by creating .python-version files. For instance, you can simply add to the root of the ocean repo a .python-version file containing the name of the pyenv virtualenv created in the previous step. It should look like the following:

ocean-3.10.10

You can finally install the required packages using the requirements-dev.txt file:

$ pip install -r requirements-dev.txt

We use black for code formatting and flake8 for linting.

**VSCode recommended setup**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/python/#vscode-recommended-setup)

Additionally, the below settings will make vscode to automatically format your source files on save.

{  
 "editor.rulers": [  
 120  
 ],  
 "python.linting.flake8Args": [  
 "--config=pyproject.toml"  
 ],  
 "python.linting.flake8Enabled": true,  
 "python.formatting.provider": "black",  
 "editor.formatOnSave": true,  
 "[python]": {  
 "editor.codeActionsOnSave": {  
 "source.organizeImports": true  
 }  
 }  
}

**JetBrains IDEs (Pycharm, Intellij + Python plugin) recommended setup**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/python/#jetbrains-ides-pycharm-intellij--python-plugin-recommended-setup)

Unfortunately, JetBrains IDEs do not support black and flake8 out-of-the-box. But this section will get you covered.

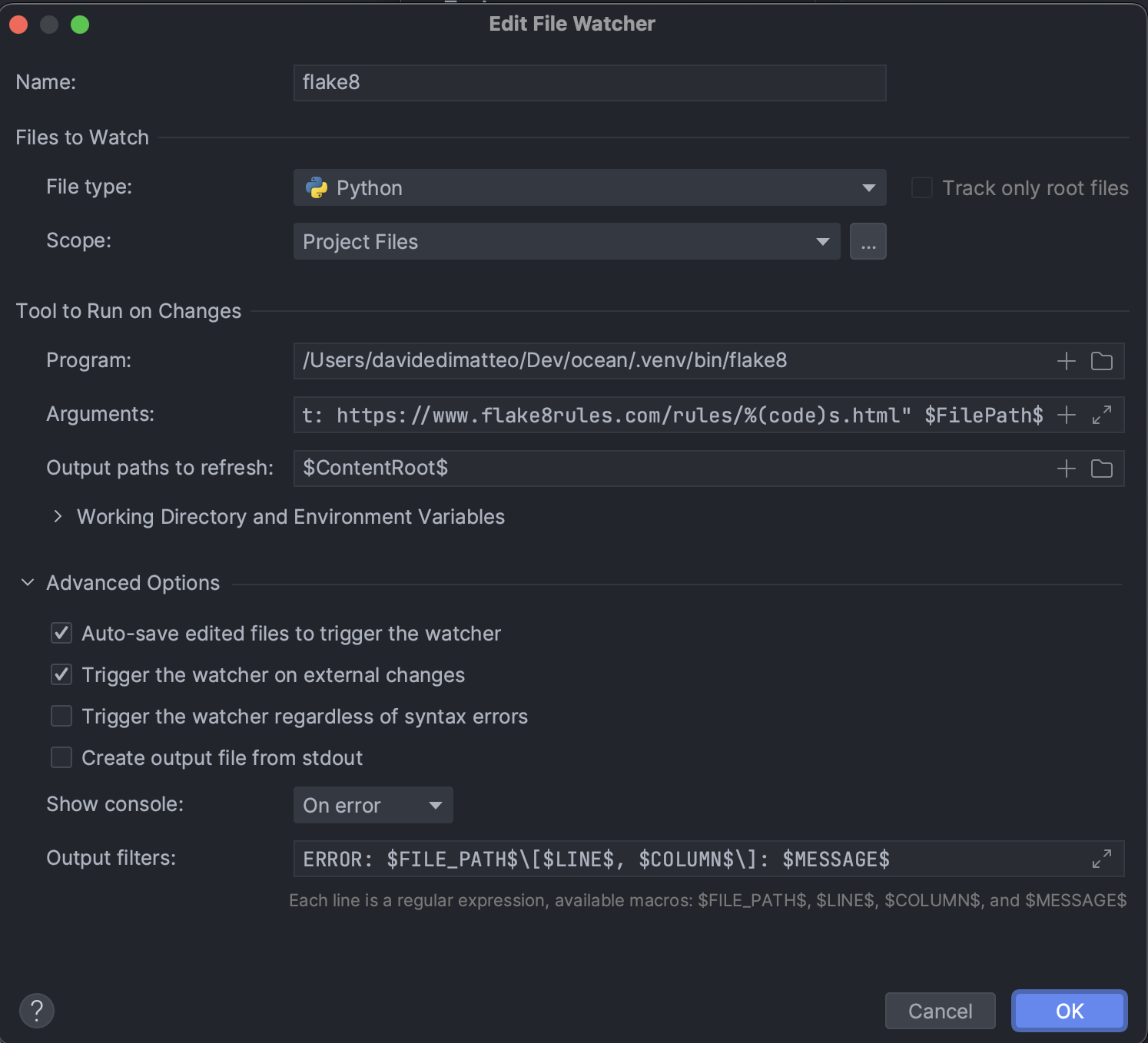
**black**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/python/#black)

Please refer to the [black docs](https://black.readthedocs.io/en/stable/editor_integration.html#pycharm-intellij-idea) to choose your preferred method.

**flake8**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/python/#flake8)

1. Install the *File Watchers* plugin from the Plugins Marketplace of your JetBrains IDE
2. Go to File > Settings > Tools > File Watchers to access the File Watchers settings
3. Click on the + button to add a new File Watcher
4. Add the following configuration:
   * Name: flake8
   * File type: Python
   * Scope: Project Files
   * Program: path/to/your/flake8/bin/inside/your/virtualenv (e.g. /Users/davidedimatteo/Dev/ocean/.venv/bin/flake8)
   * Arguments: --format="ERROR: %(path)s[%(row)d, %(col)d]: %(text)s (%(code)s) %(code)s. More info at: https://www.flake8rules.com/rules/%(code)s.html" $FilePath$
   * Output paths to refresh: $ContentRoot$
   * Working directory: $ProjectFileDir$
   * Tick *Auto-save edited files to trigger the watcher*
   * Tick *Trigger the watcher on external changes*
   * *Show console: On error*
   * Error output: ERROR: $FILE\_PATH$\[$LINE$, $COLUMN$\]: $MESSAGE$ (this setting will show the error in the editor)
5. Apply the changes
6. Install the *Awesome Console* plugin from the Plugins Marketplace of your JetBrains IDE to render links properly

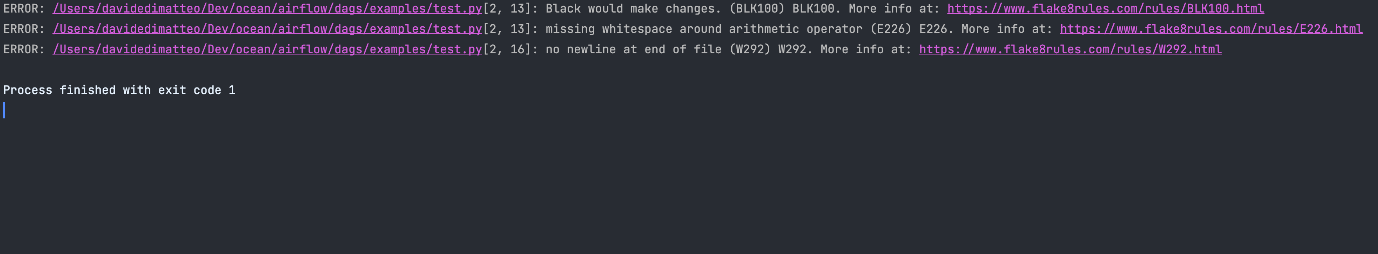
Your File Watchers settings should look like this:



To test that everything works, just create a new python file in the project and try to paste the following code:

def f(a=10):  
 return a\*10

You should see some flake8 errors in the console. It should look like this:



Note that if you don't like the console to be opened automatically, you can disable it by going to File > Settings > Tools > File Watchers and un-tick *Show console: On error*. You can then assign a shortcut to the File Watchers > flake8 action to run it manually.

# Terraform

## What is Terraform?[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/terraform/#what-is-terraform)

[**Terraform**](https://cloud.google.com/bigquery/docs/introduction) is our tool for **infrastructure as code (IaC)**. It allows users to define, manage, and provision infrastructure resources such as virtual machines, networks, storage, and more, in a declarative way, using a high-level configuration language.

## Dev guidelines[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/terraform/#dev-guidelines)

### Git[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/terraform/#git)

Our terraform project lives within our monorepo.

#### Where do I save my code[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/terraform/#where-do-i-save-my-code)

There is a specific terraform folder nested in each integration or resource folder which need to use this tool. It is possible to use the \_shared/terraform/common folder to save "common" templates to define shared or repeated resources (IAM bindings, shared buckets, variables, etc.). For any initiatives that do not pertain to a specific integration or tool, it is possible to create a new folder.

### Backend configuration[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/terraform/#backend-configuration)

A backend defines where Terraform stores its states data files

### Code formatter[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/terraform/#code-formatter)

* Formatter: [**terraform fmt**](https://developer.hashicorp.com/terraform/cli/commands/fmt)

"Formatting

$ terraform fmt

### Terraform command line[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/terraform/#terraform-command-line)

It is possible to use the Terraform command line to execute various operations such as creating, updating, and deleting infrastructure resources. The Terraform command line is used to interact with the Terraform configuration files and the target infrastructure provider's APIs Here are some common Terraform command:

* init: Initializes a Terraform working directory, downloading the necessary providers and modules specified in the configuration file

init

$ terraform init  
Initializing modules...  
 - common in ../terraform-common  
Initializing the backend...  
Successfully configured the backend "gcs"! Terraform will automatically  
use this backend unless the backend configuration changes.  
[..]  
'''  
- plan: Generates an execution plan for applying changes to the infrastructure. This shows you what Terraform plans to do before making any actual changes  
```bash title="plan"  
$ terraform plan

* apply: Applies the changes to the infrastructure as described in the Terraform configuration file

apply

$ terraform apply

* workspace: used to manage multiple named workspaces within a single Terraform configuration

Workspace

$ terraform workspace list  
\* default  
 pro  
  
$terraform workspace select pro  
Switched to workspace "pro".

* state: Provides information about the current state of the infrastructure

state

$ terraform state list  
data.archive\_file.save\_metrics\_function\_src  
[...]  
google\_service\_account.ocean\_service\_account]  
  
$ terraform state show google\_service\_account.ocean\_service\_account  
*# google\_service\_account.ocean\_service\_account:{*  
 [...]  
}

## Best Practices[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/terraform/#best-practices)

Please find below some recommend best practices regarding Terraform:

* Use a remote backend to store the terraform state. We use a GCS bucket only for this purpose, and it belongs to the frontiers-infraestructure project.
  + <https://www.terraform.io/language/settings/backends>
  + <https://www.terraform.io/language/state/sensitive-data>
* Do NOT include secrets or sensitive information in the terraform templates. Instead, define the secret in Secret Manager, and reference it from the template to use it in a secure manner:
  + <https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/secret_manager_secret_version>
* Keep the terraform templates DRY - Don't repeat yourself
  + Use "common" templates to define shared or repeated resources (IAM bindings, shared buckets, variables, etc.)
* Use terraform workspaces to map to different envs or projects. E.g:
  + default → frontiers-ocean-sandbox
  + pro → frontiers-ocean
* Use terraform fmt for automatic code formatting.
* Avoid unmanaged resources
  + e.g.: "Databricks" - Clusters, Notebooks and Jobs can also be managed by terraform.

# SQL Validation Pipeline

This pipeline checks SQL code style and quality using SQLFluff when a PR is created. It can be found in our ocean monorepo at .cicd/azure-prs-sql.yaml It includes the following steps:

1. Check the Python version and pip version to ensure compatibility with SQLFluff.
2. Install SQLFluff using pip install sqlfluff.
3. Check the SQLFluff version to ensure the correct version is being used.
4. Run SQLFluff linting on the models folder inside the dbt directory. This step checks the SQL code for compliance with established style rules and reports any discrepancies found.

# Pull Request Merged Notification Pipeline

This pipeline is responsible for sending a notification to a Microsoft Teams channel when a Pull Request (PR) is merged in the Ocean repository. The pipeline is triggered when a commit is pushed to any branch, but it excludes all paths, meaning it will only run on PR merges.

## Job: NotifyTeamsOnPRMerged[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/toolbox/azure-devops/notifications-pipelines/pr-merge/" \l "job-notifyteamsonprmerged" \o "Direct link to Job: NotifyTeamsOnPRMerged)

**DisplayName**: Notify Teams on PR Merged

1. **Set Variables**: This step sets the variable for the commit message.
2. **PowerShell**: This step uses the PowerShell@2 task to send a webhook notification to the Braavos PRs Microsoft Teams channel with details about the merged PR, including the commit message.

**Pull Request Created Notification Pipeline**

This pipeline is responsible for sending a notification to a Microsoft Teams channel when a new Pull Request (PR) is created in the Ocean repository. The pipeline is triggered when a PR is created or updated, targeting any branch.

**Job: NotifyTeamsOnPR[​](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/toolbox/azure-devops/notifications-pipelines/pr-create/" \l "job-notifyteamsonpr" \o "Direct link to Job: NotifyTeamsOnPR)**

**DisplayName**: Notify Teams on PR

1. **checkout**: This step checks out the repository and persists credentials for subsequent steps.
2. **PowerShell**: This step uses the PowerShell@2 task to perform the following actions:
   * Set variables for the PR ID, organization, project, repo, and personal access token.
   * Retrieve PR information using the Azure DevOps REST API.
   * Send a webhook notification to the designated Microsoft Teams channel with details about the created PR, including PR ID, title, author, description, and URL.

**Storing the Personal Access Token in Azure DevOps Pipeline Variable**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/toolbox/azure-devops/notifications-pipelines/pr-create/#storing-the-personal-access-token-in-azure-devops-pipeline-variable)

To store the personal access token (PAT) securely in Azure DevOps, follow these steps:

1. Open your pipeline in Azure DevOps.
2. Click on the "Edit" button to edit the pipeline.
3. Click on the "Variables" tab.
4. Click on the "+ Add" button to add a new variable.
5. Name the variable pat and enter the value of your Personal Access Token.
6. Click on the "Keep this value secret" checkbox to make sure the token is stored securely and not exposed in logs.
7. Click on "Add" to save the variable.
8. Use the variable in your pipeline script by using the syntax $(pat).

Make sure to replace the personalAccessToken variable assignment in your script with the following line:

# Python Validation Pipeline

This pipeline checks Python code style and quality using Flake8 when a PR is created. It can be found in our ocean monorepo at .cicd/azure-prs-python.yaml It consists of the following steps:

1. Check the Python version and pip version to ensure compatibility with Flake8 and related plugins.
2. Install Flake8 and related plugins using pip install. The installed plugins include flake8-black, flake8-isort, flake8-bugbear, flake8-bandit, and flake8-pyproject.
3. Run Flake8 on the codebase. This step checks the Python code for compliance with the defined style rules, as well as potential bugs, security issues, and adherence to pyproject.toml configurations.

**Git**

**Monorepo structure**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/toolbox/git/#monorepo-structure)

We have decided to use a monorepo strategy for Ocean Breeze. A **Git monorepo** is a version control strategy where all code and assets are stored in a single repository. There are several benefits to using a monorepo:

* Easier to **track changes** and ensure consistency across the entire codebase.
* Improved **organization**: A monorepo makes it easier to structure and organize code, as all code and assets are stored in a single place.
* **Better visibility**: With a monorepo, it is easier to see the big picture and understand how different parts of the codebase are connected and interdependent.
* Simplified **dependency management**, it's easier to resolve dependency conflicts.
* **Better testing**: This makes it easier to ensure that changes in one part of the codebase do not negatively impact other parts of the codebase.
* Ocean repository available at: [open in azure devops ⧉](https://devops-server.frontiersin.net/Madrid/Architecture/_git/ocean)

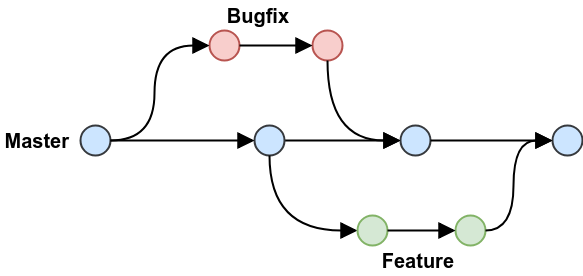
Ocean Breeze Repository Organization

📂 frontiers-ocean-breeze  
├── README.md  
├── CHANGELOG.md *# keep track of added/fixed/changed stuff*  
├── pyproject.toml *# python settings*  
├── 📂 .cicd *# azure devops pipelines*  
├── 📁 airbyte  
├── 📁 airflow  
├── 📁 dbt  
├── 📁 docusaurus  
├── 📁 kafka  
├── 📂 src  
│ ├── 📂 \_shared *# sources of shared tools such as "KafkaConsumerCheck" or GcpTools*  
│ └── 📂 frontiers-graph  
│ ├── bqmetadata.yaml  
│ ├── README.md  
│ ├── 📁 config  
│ └── 📁 src  
└── 📂 terraform  
 ├── 📁 \_airflow\_dags *# dags deploy to Composer/Airflow envs*  
 ├── 📁 \_common *# common settings to use in all terraform subprojects*  
 └── 📁 \_infra *# IaC to deploy our stack: airflow, buckets, airbyte, functions, etc.*

**Workflow**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/toolbox/git/#workflow)

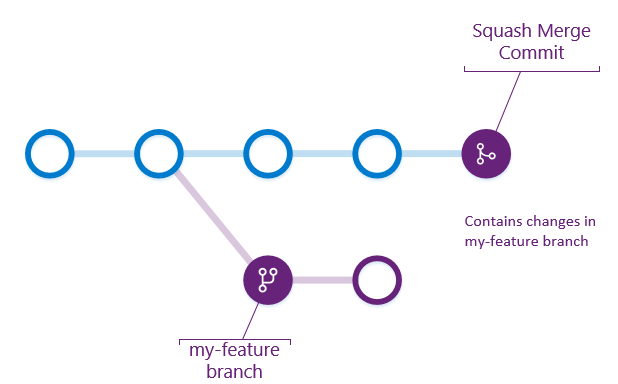
We have chosen the **GitHub Flow** workflow for its simplicity and frequency of releases. In this workflow, the following steps are followed:

1. A main branch is maintained, which is always deployable.
2. When working on a new feature or bugfix, a new branch is created with a descriptive name, such as "feature/pubpeer-integration".
3. The developers make changes on the new branch and when it is ready, they submit a Pull Request with the main branch as the target.
4. During code review, further changes may be made to the branch. Once the code review is completed, the branch is merged into main.
5. Upon updating main, the code should be deployed immediately.



**Merge strategy**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/toolbox/git/#merge-strategy)

**Squash merging** condenses multiple commits into a single commit, resulting in a cleaner and more streamlined Git history. This can make it easier to understand the evolution of your codebase over time.



**Branch names**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/toolbox/git/#branch-names)

* feature/ for a new addition that would produce a new deployment.
* bugfix/ for a fix that, once merged, would be deployed.

**Pull requests titles and descriptions**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/toolbox/git/#pull-requests-titles-and-descriptions)

When writing issue or pull request (PR) titles, focus on describing the desired outcome rather than the implementation details. This applies to most issues and PRs, except for sub-issues of an epic. It's also important to specify the relevant domain, as Airbyte operates within a monorepo environment.

To illustrate this, consider the following examples:

* 🔔 Subpar Issue Title: "Remove airbyteCdk.dependsOn("unrelatedPackage")." This title focuses on the solution, but doesn't explain the problem being solved.
* ✅ Improved Issue Title: "Prevent the building of unrelated packages when building the Airbyte Python CDK." This title clearly states the desired outcome and provides enough information to understand the issue without reading further.

Or:

* 🔔 Subpar PR Title: "Update tests." This title doesn't specify which tests were updated or what the changes were.
* ✅ Improved PR Title: "Update acceptance tests for the Source MySQL connector to support connecting to an SSL-enabled database." This title specifies the domain and the specific change that was made, providing a clearer understanding of the PR's purpose.

When providing descriptions, include only the necessary information relevant to the review. Any additional details or tips can be included in the new feature's README.md file. This helps to avoid duplicating information and ensures that all relevant information is easily accessible in one place.

**Pull requests examples**[**​**](https://ocean-docs.frontiers-ss-dev.info/next/team-braavos/toolbox/git/#pull-requests-examples)

* 💨 Airbyte
* ⚡ Kafka Connect
* ✏️ Custom code
* 🐛 Bugfix

Given a new source named **pubpeer**:

* *Title*: **New data source: PubPeer**
* feature/pubpeer into main

PR Files

*# always present*  
+ src/pubpeer/README.md  
+ src/pubpeer/bqmetadata.yaml  
M CHANGELOG.md  
  
*# airbyte settings to set up the source and the connection, which would*   
*# produce the `Initial` dataset*  
+ airbyte/sources/pubpeer/configuration.yaml  
+ airbyte/connections/pubpeer/configuration.yaml  
*# + airbyte/destinations/pubpeer/configuration.yaml*  
*# └── optional, if it is BigQuery it should be already created*  
  
*# dbt would be used to create the Standardized and Conformed layers*  
+ dbt/models/pubpeer/pubpeer.yaml  
+ dbt/models/pubpeer/{1-n}.sql  
  
*# a new Airflow DAG*  
+ airflow/dags/pubpeer.py  
  
*# update the docs*  
M docusaurus/doc/data-directory/directory.md  
+ docusaurus/doc/data-directory/pubpeer.md  
M docusaurus/doc/data-sets/sets.md