Federal University of Rio Grande do Sul - UFRGS
Institute of Informatics

OSDU Tutorial

PETWIN – Digital Twins for Optimization and Production Management

Data Ingestion into the OSDU Data Platform

Author: **Jaqueline Bitencourt**
Technical collaboration: **Marcos Leipnitz**

July 2023

Contents

1 Introduction

The OSDU Data Platform (Open Subsurface Data Universe)[1] focuses on the upstream subsurface scope of O&G. It aims to break down data silos, avoid data transfer between systems and drive innovation (e.g., predictive models). This solution will allow building data-centric systems that are authoritative data sources (System of Record - SoR). For this, it provides an architecture for cloud-native data management based on standardized Application Programming Interfaces (APIs) and micro-service architecture, scalable and technology-agnostic. This document details each platform service and how to use them.

1.1 OSDU Services Endpoints

The current Mercury R3 release offers services in the form of APIs aimed at ingesting and storing data, defining metadata schemata, indexing, searching, access control and security, compliance, event notifications, and utility services (spatial references and unit conversions). The services of the OSDU Data Platform are:

1. **Dataset:** provides internal and external APIs to allow fetching storage/retrieval instructions for various types of datasets. In summary, this service allows uploading our datasets (e.g., csv, pdf, txt, doc...) into OSDU. More details on dataset APIs are available here in GitLab.
   The endpoint address is *"https://your-domain/api/dataset/v1/info"*.

2. **File:** provides internal and external APIs to request for file location data, such as a signed URL per file upload. Using the signed URL, OSDU users will be able to upload their files for ingestion into the system. More details on file APIs are available here in GitLab. The endpoint address is *"https://your-domain/api/file/v2/info"*.

3. **Indexer:** indexes the metadata store to support the Search Service. More details on indexer APIs are available here in GitLab.
   The endpoint address is *"https://your-domain/api/indexer/v2/info"*.

4. **Notification:** allows interested consumers to subscribe to data and metadata changes using a publish/subscriber pattern. More details on notification APIs are available here in GitLab.
   The endpoint address is *"https://your-domain/api/notification/v1/info"*.

5. **Partition:** dynamically pulls the correct connection information at runtime to connect to the correct partition. More details on partition APIs are available here in GitLab. The endpoint address is *"https://your-domain/api/partition/v1/info"*.

6. **Register:** manages subscriptions, action, and DDMS registrations in the OSDU. More details on register APIs are available here in GitLab. The endpoint address is *"https://your-domain/api/register/v1/info"*.

7. **Schema:** enables schema management in the OSDU and offers an implementation of the schema standard. In summary, the OSDU schemata are the standard to represent the raw data (e.g., csv, pdf...), and they can be customized as needed. More details on schema APIs are available here in GitLab. The endpoint address is *"https://your-domain/api/schema-service/v1/info"*.

[1] https://osduforum.org

8. **Search:** supports full-text search on string fields, range queries on date, numeric or string fields, along with geospatial search. This service works at the schema level only. If you want access to the raw data, you need to use the file service. More details on search APIs are available here in GitLab. The endpoint address is *"https://your-domain/api/search/v2/info"*.

9. **Storage:** handles metadata ingestion in OSDU. This means that this service works at the schema level once file loading and storage is managed by the file service. More details on storage APIs are available here in GitLab. The endpoint address is *"https://your-domain/api/storage/v2/info"*.

10. **CRS Catalog:** enables end-users to make CRS selections, search for CRSs given a number of constraints, download of the entire catalog for local caching, and access to various sub-sets of the catalog. More details on CRS catalog APIs are available here in GitLab.
    The endpoint address is *"https://your-domain/api/crs/catalog/v2/info"*.

11. **CRS Conversion:** provides spatial reference conversions for coordinates. Coordinates are represented by an array of 3D points. More details on CRS conversion APIs are available here in GitLab.
    The endpoint address is *"https://your-domain/api/crs/converter/v2/info"*.

12. **Unit:** provides dimension/measurement and unit definitions. Given two unit definitions, the service also offers conversion parameters in two different parameterizations. More details on unit APIs are available here in GitLab. The endpoint address is *"https://your-domain/api/unit/v3/info"*.

13. **Entitlements:** is used to enable authorization in OSDU Data Ecosystem. The service allows for the creation of groups. A group name defines a permission. Users who are added to that group obtain that permission. The main motivation for entitlements service is data authorization, but the functionality enables three use cases: data groups, service groups, and user groups. More details on entitlements APIs are available here in GitLab.
    The endpoint address is *"https://your-domain/api/entitlements/v2/info"*.

14. **Legal:** provides APIs to help with legal data governance in the ecosystem. More details on legal/compliance APIs are available here in GitLab. The endpoint address is *"https://your-domain/api/legal/v1/info"*.

15. **Workflow:** provides a wrapper functionality around the Apache Airflow functions and is designed to carry out preliminary work with files before running the Airflow Directed Acyclic Graphs (DAGs) that will perform actual ingestion of OSDU data. More details on workflow APIs are available here in GitLab. The endpoint address is *"https://your-domain/api/workflow/v1/info"*.

16. **Well Delivery:** Domain Data Management Service focused on Well Planning and Well Execution. More details on well delivery APIs are available here in GitLab. The endpoint address is *"https://your-domain/api/well-delivery/info"*.

17. **Wellbore DDMS:** Wellbore Domain Data Management Services include type-safe entity access and optimized accessors for bulk data such as logs, trajectories,

checkshots. More details on wellbore DDMS APIs are available here in GitLab. The endpoint address is *"https://your-domain/api/os-wellbore-ddms/ddms/v2/about"*.

18. **Seismic Store:** Seismic Store is a cloud-based solution composed by restful microservices, client APIs, and tools designed to implement a multi-object storage approach. The system saves objects that compose a dataset as a hierarchical data structure in cloud storage and the dataset properties as a metadata entry in a no-relational catalog. Having the datasets stored as multiple independent objects improve the overall performance, as generic I/O operations, for example, read or write objects, can be easily parallelized. More details on wellbore DDMS APIs are available here in GitLab. The endpoint address is *"https://your-domain/api/seismic-store/v3/svcstatu"*.

1.2 Purpose of the tutorial

The main goal of this tutorial is to provide a step-by-step about how to use the OSDU services and do basic operations, such as creating legal tags, adding new users, creating new schema and records, uploading files, and downloading them.

1.3 References

Please note that the links below are good at the time of writing but cannot be guaranteed for the future.

- OSDU Portal (https://osduforum.org)
- OSDU Community GitLab (https://community.opengroup.org/osdu)
- OSDU Member GitLab (https://gitlab.opengroup.org/osdu)

2 Used tools

Before we get our hands dirty, let's know the needed tools and how to configure them. Your infrastructure team should provide all necessary usernames and passwords for you to access the tools.

2.1 MinIO Object Storage

MinIO is an object storage solution that provides an Amazon Web Services S3-compatible API and supports all core S3 features. MinIO is built to deploy anywhere - public or private cloud, baremetal infrastructure, orchestrated environments, and edge infrastructure.

Figure 1 shows the home page with buckets of the OSDU instance. Note that these buckets store the information related to Airflow logs, system configuration, schemata, records, and files uploaded into the OSDU. Therefore, when you create new records or upload some files (csv, las, pdf, doc) into the OSDU, you can visualize them quickly using the MinIO interface or to use the APIs.

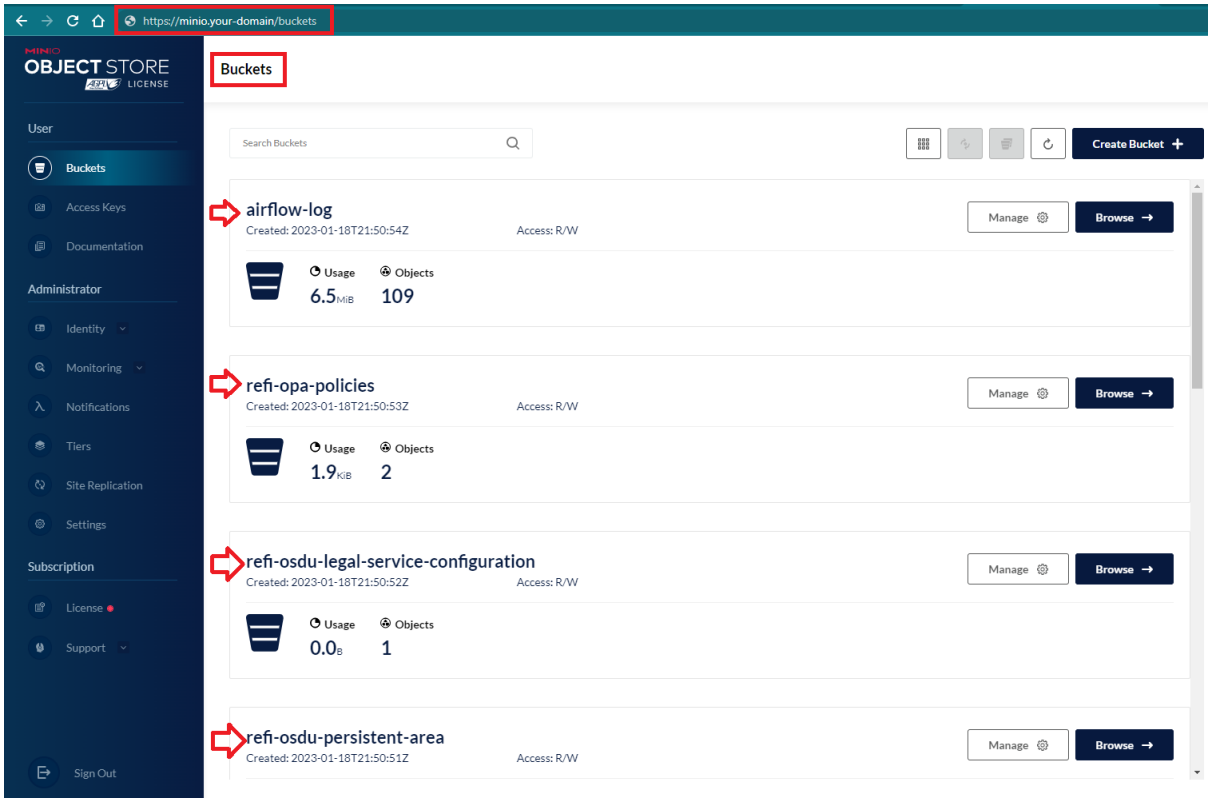

Figure 1: MinIO buckets

2.2 Keycloak

The OSDU has a sophisticated protocol to manage the authentication and authorization operations. OpenID Connect is used for signing in operation, and OAuth2 is the basis of OpenID Connect, and it is a delegation protocol designed to help you access third-party APIs. Keycloak tool provides a graphic interface to manage all related to authentication and authorization. Besides, it is in Keycloak where get the vital information to use and make API requests.

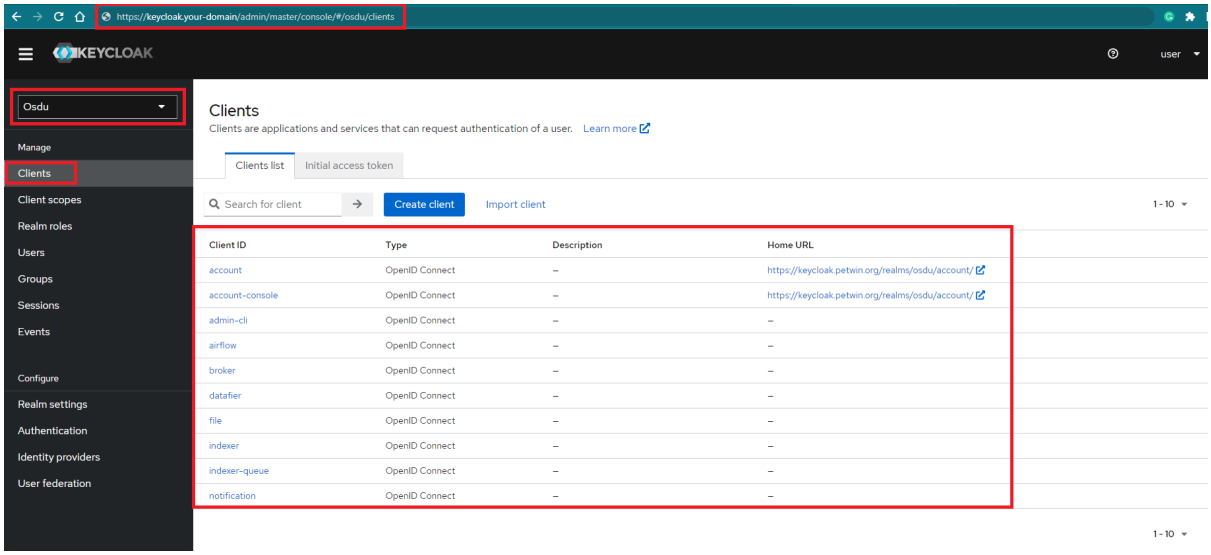

Figure 2: Keycloak home page

Note: In the two following paragraphs is explained how to get the main information that will be used in the Insomnia tool.

The three vital information are ACCESS TOKEN URL, CLIENT ID, and CLIENT SECRET. To get them, you need to access the account at the link `https://keycloak.your-domain/admin/master/console/#/master/clients`, select the OSDU namespace, and *Clients* in the side menu, and then click on *datafier* client id as show Figure 2.

Figure 3 presents how to get the CLIENT SECRET value. As mentioned before, the client id value is *datafier*. Finally, you can get the ACCESS TOKEN URL at the link `https://keycloak.your-domain/realms/osdu/.well-known/openid-configuration`, or click on the *Realm Settings* option in the sidebar menu, then in the link OpenID Endpoint Configuration, and look for *token_endpoint* to get the access token URL, as shown in the Figure 4.

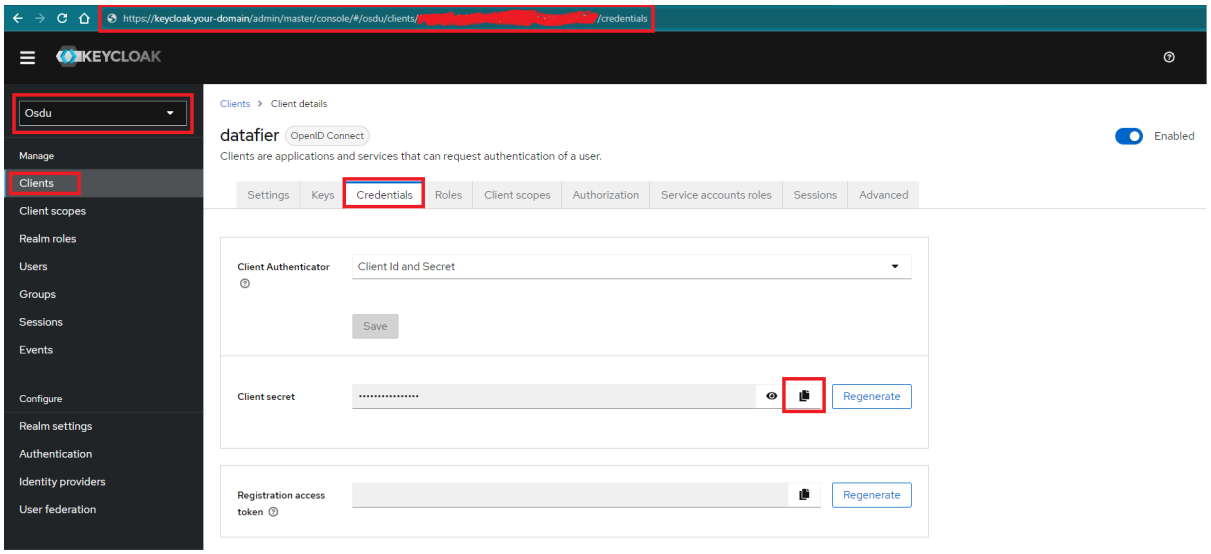

Figure 3: Keycloak: Getting the Client Secret

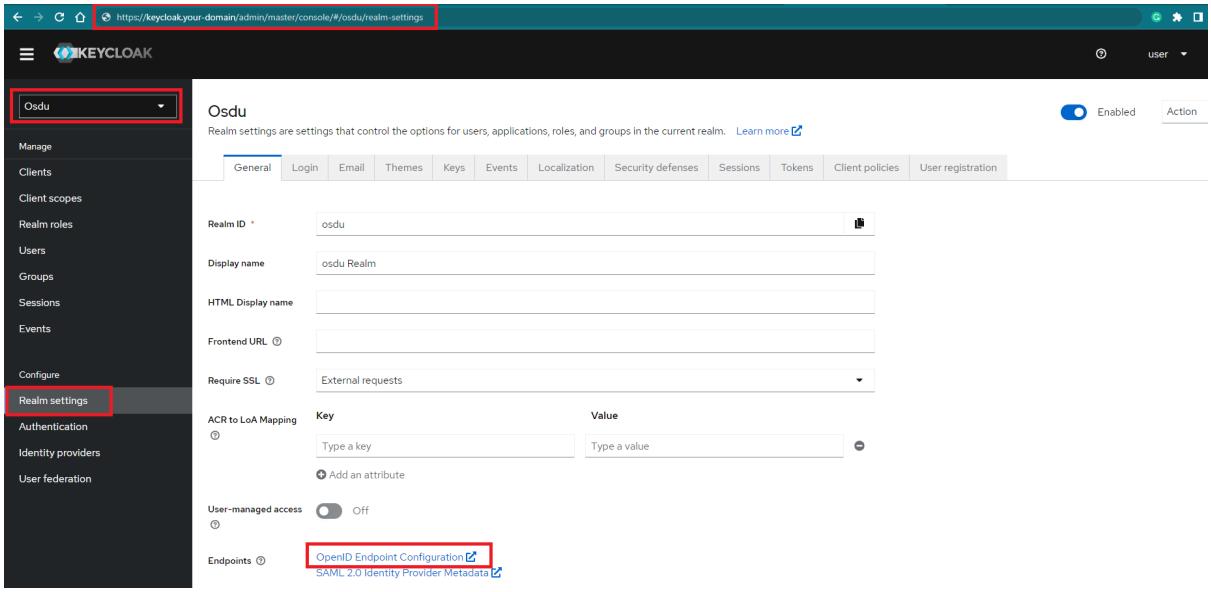

Figure 4: Keycloak: Getting the Access Token URL

2.3 Insomnia Rest - configuration

Insomnia is an open-source desktop application that takes the pain out of interacting with and designing, debugging, and testing APIs. Insomnia combines an easy-to-use interface with advanced functionality like authentication helpers, code generation, and environment variables. In summary, Insomnia is similar to the Postman tool.

To start your tests, you need to download the Insomnia tool at the link `https://insomnia.rest/download`. Open the tool as shown in Figure 5.

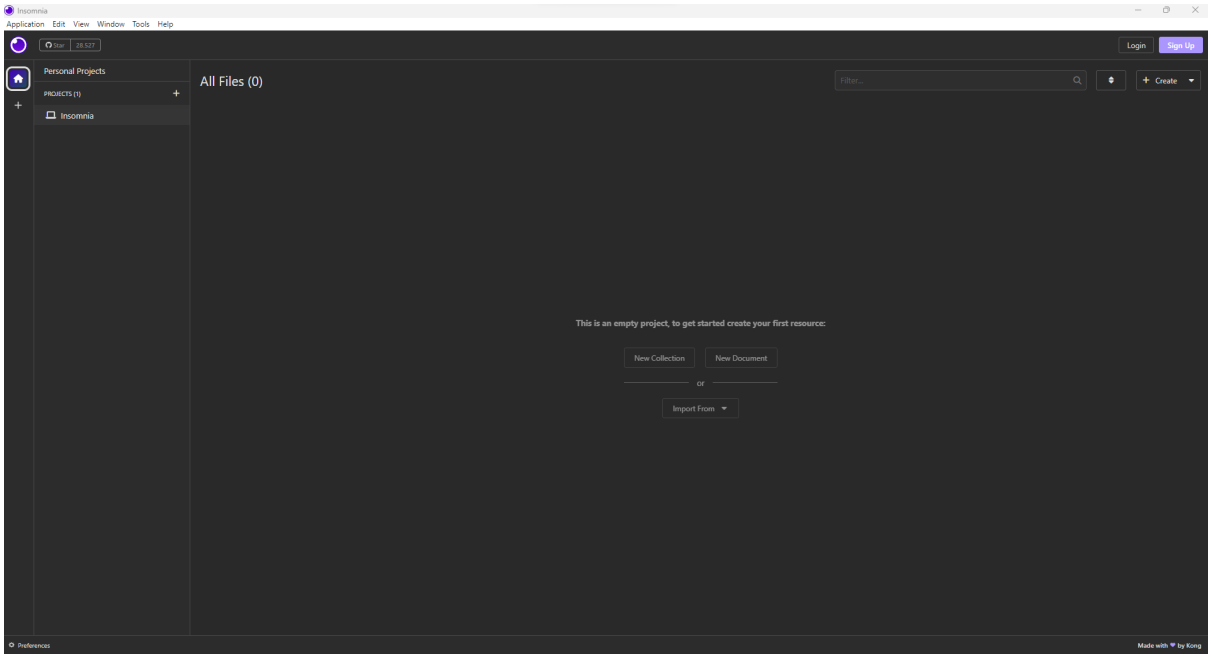

Figure 5: Insomnia default page

Now, your next steps are to create a new project, a new collection, and finally, a new HTTP request, as shown in Figures 6, 7, and 8

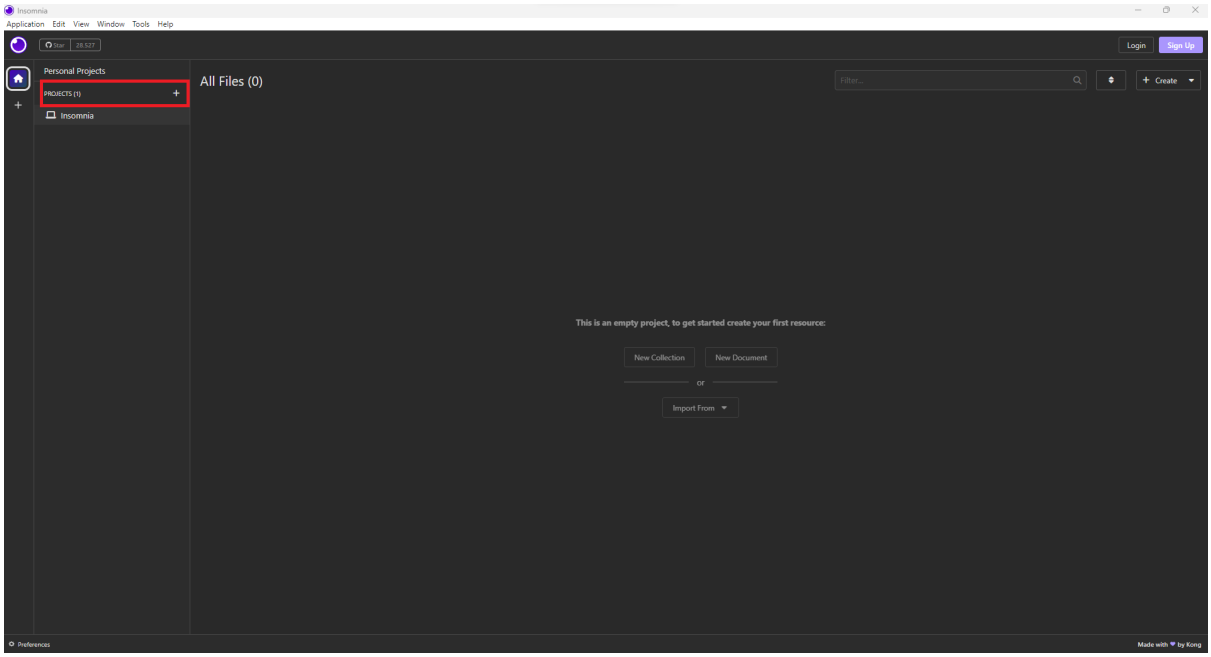

Figure 6: Insomnia: New project

2.3.1 First API request

With project, collection, and request created, you need to configure the authorization of your first request. Below the address field are five tabs named body, auth, query, headers, and docs. You will select the OAuth 2.0 option in the auth tab, as shown in Figure 9. Then, in the grant type field, you will select the *Client Credentials* option and fill in the access token URL, client id, and client secret fields, as shown in Figure 10.

Finally, you just need to inform the API address and select the corresponding method type and click on *send*. Let's make a GET type request for the schema service (https://your-

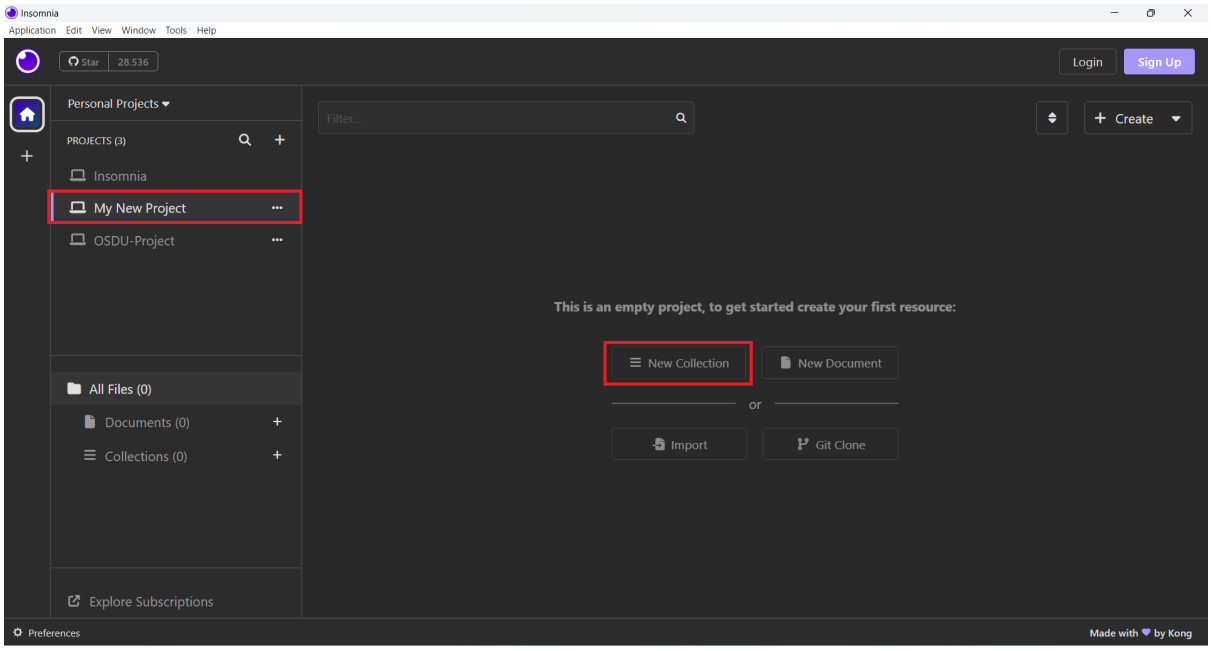

Figure 7: Insomnia: New collection

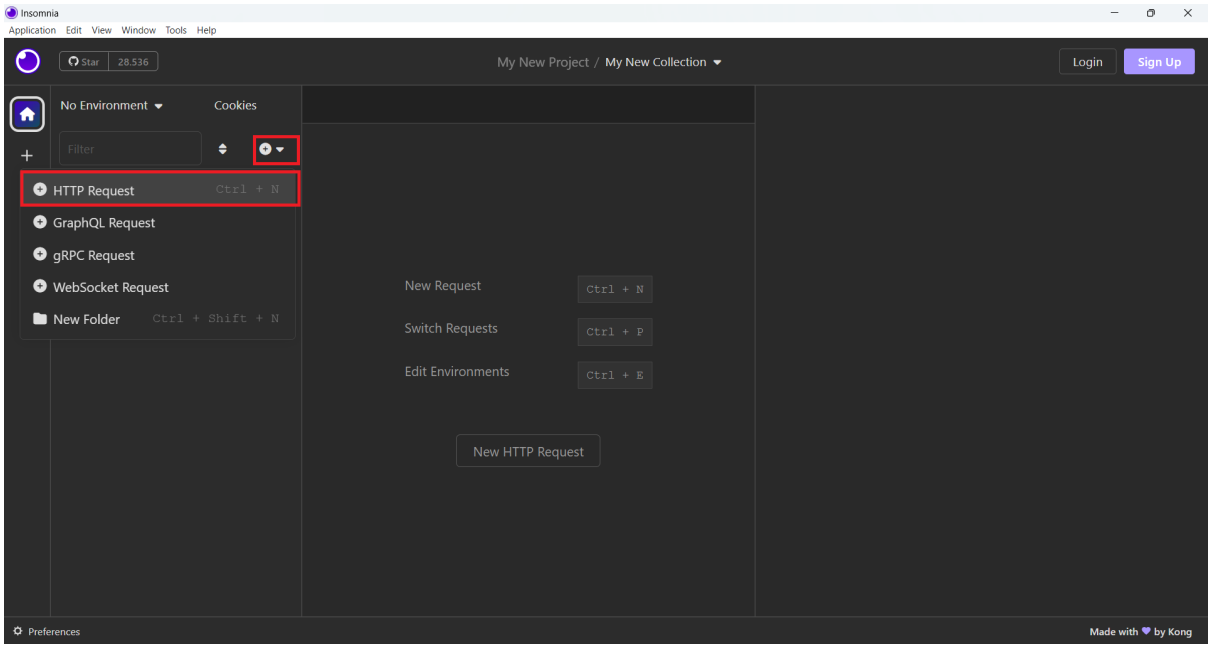

Figure 8: Insomnia: New HTTP request

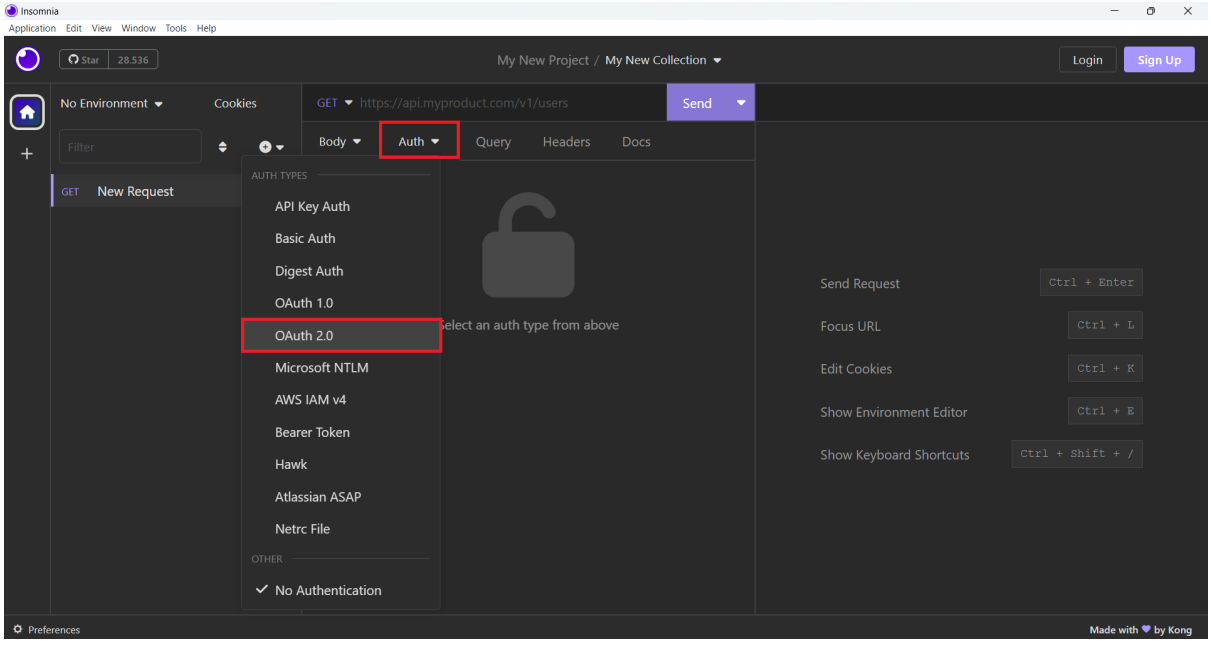

Figure 9: Insomnia: Select OAuth2

domain/api/schema-service/v1/info). Figure 11 shows an example of a request. Besides, you can change the request name. For that, click on the request name with the right button and select the *rename* option. It helps keep the collection organized.

2.3.2 Defining environment variables

In this subsection, we demonstrate how to create environment variables. This is a useful resource that will be used in the next section, and you need to know to use them. To define your environment variables, first, you need to click on the arrow of *No environment* and select *Manage environments*, as shown in Figure 12. Let's define your first variable.

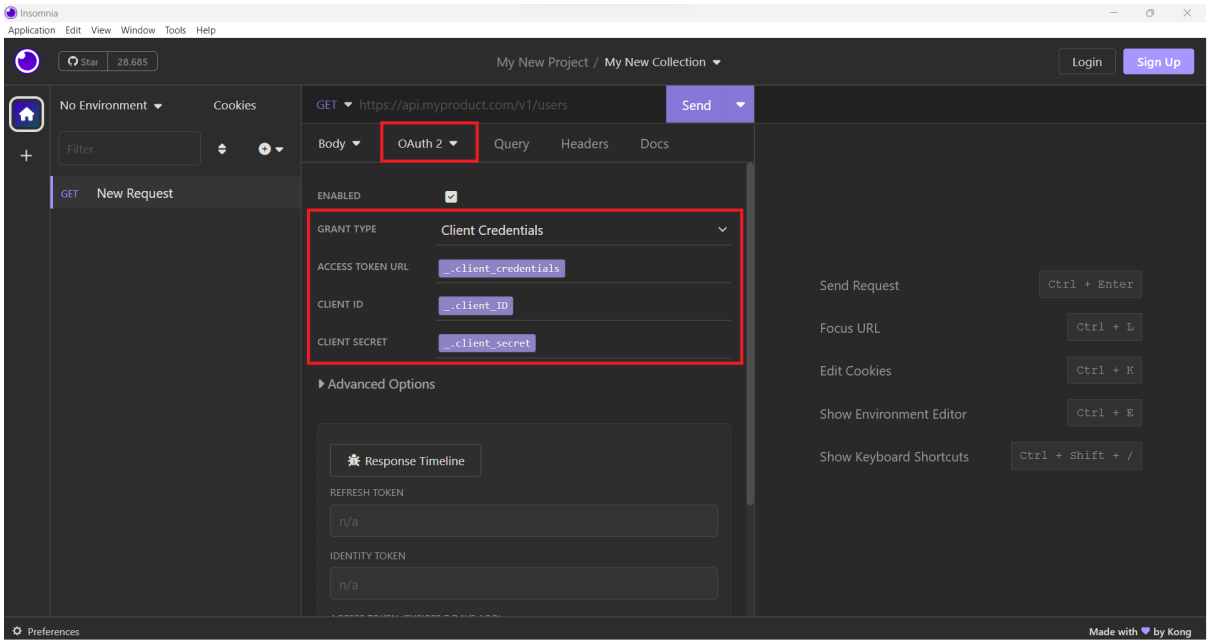

Figure 10: Insomnia: OAuth2 configuration

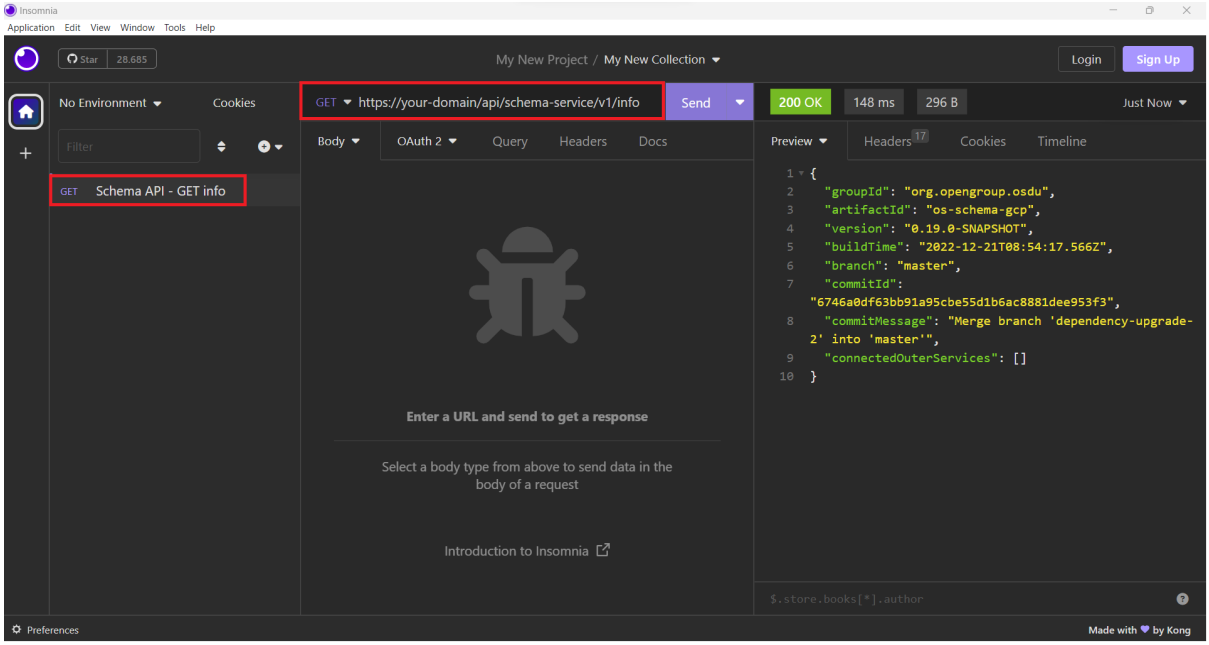

Figure 11: Insomnia: First API request

For that, you need to declare the following text: "SCHEMA_HOST": "https://your-domain/api/schema-service/v1" inside the keys of *Base environment* as shown in the Figure 13. You can add more variables just by separating them with commas.

Now you can replace the initial part of the API address (https://your-domain/api/schema-service/v1) with the variable recently created (SCHEMA_HOST), exactly as shown in Figure 14.

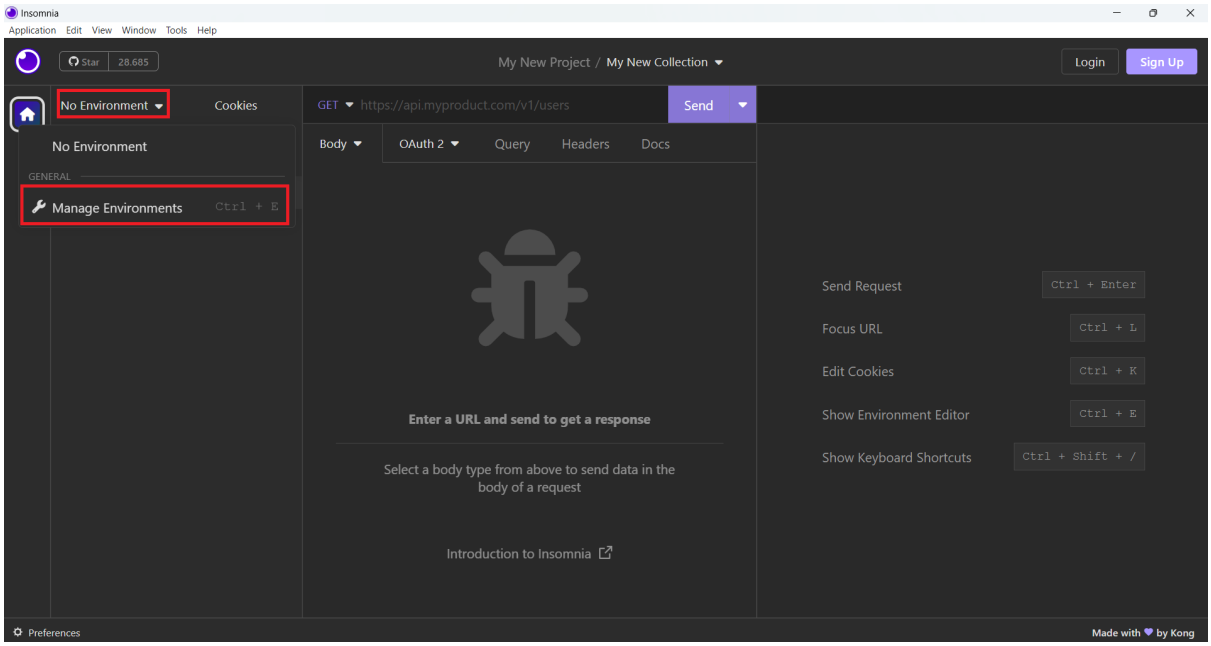

Figure 12: Insomnia: Manage Environments

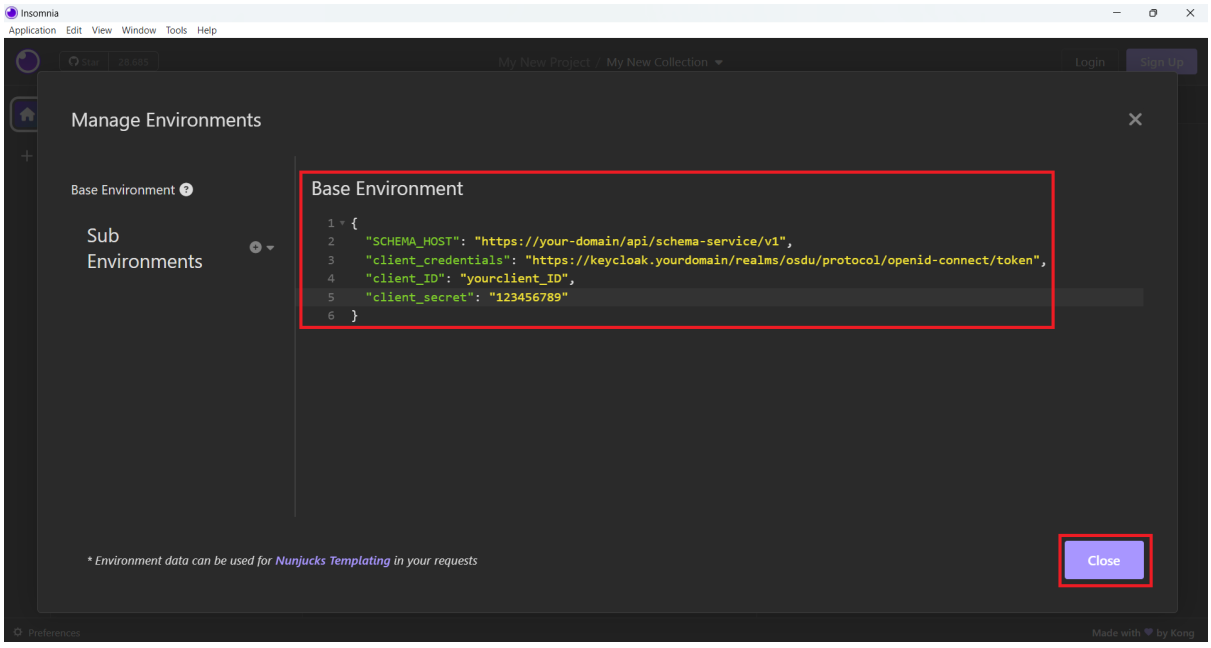

Figure 13: Insomnia: Defining a variable

2.4 Airflow

Apache Airflow provides a useful UI to monitor, schedule and manage workflows via a robust and modern web application. No need to learn old, cron-like interfaces. You always have full insight into the status and logs of completed and ongoing tasks. In the OSDU scope, this tool is used to monitor the workflows related to ingestion based on the manifest, CSV ingestion, and Energistics XML ingestion. You can access the Airflow account at the link `https://airflow.your-domain/home`.

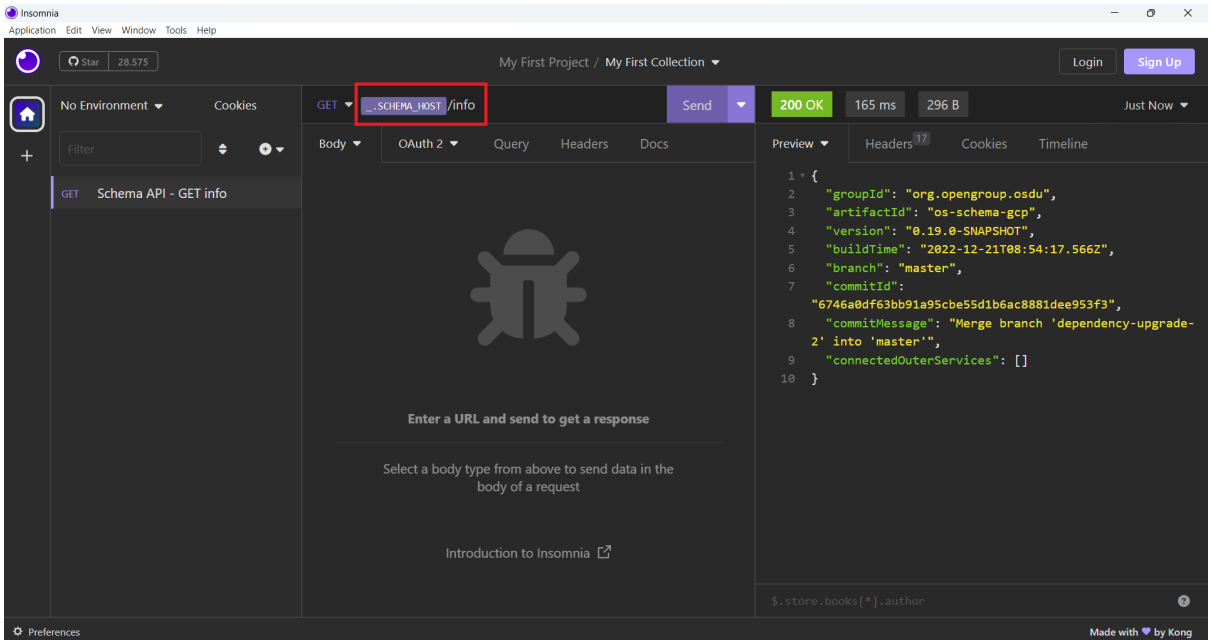

Figure 14: Insomnia: Using a variable in API address

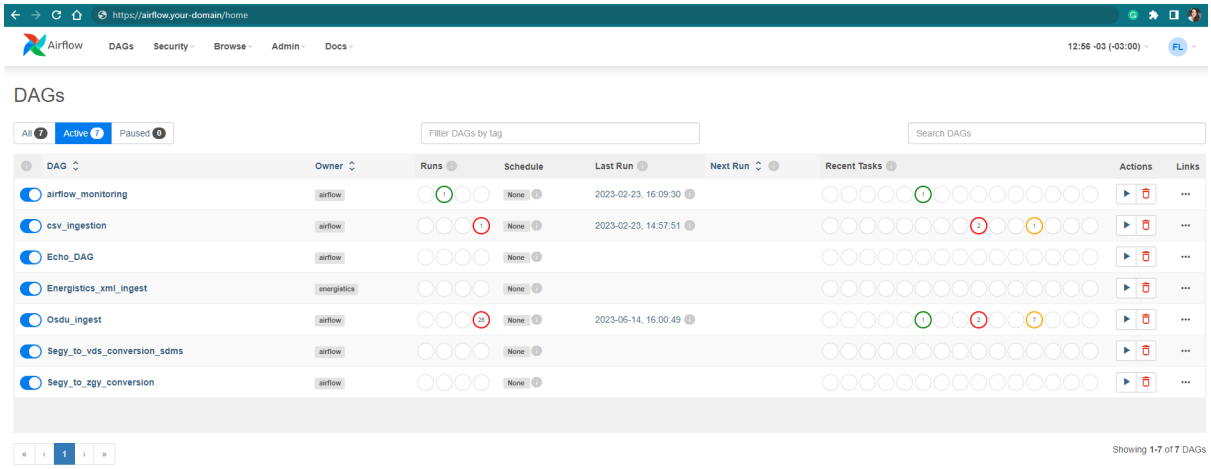

Figure 15: Airflow home

3 Working with OSDU services

For your practicality, we make all example code requests and environment variables available in GitHub. You can download it at the link `https://github.com/JaquelineBitencourt/OSDU-Tutorial-Code.git`.

To import into your Insomnia project, click on *Create* and select the *Import* option, as shown in Figure 16. Then, you choose the recently downloaded file, select your project, and provide the name of the new document. Finally, you need to choose the *Debug* option, and on the left side, you will see all HTTP requests, as shown in Figures 17, 18, and 19.

Please don't forget to copy and paste the environment variables as demonstrated in 2.3.2.

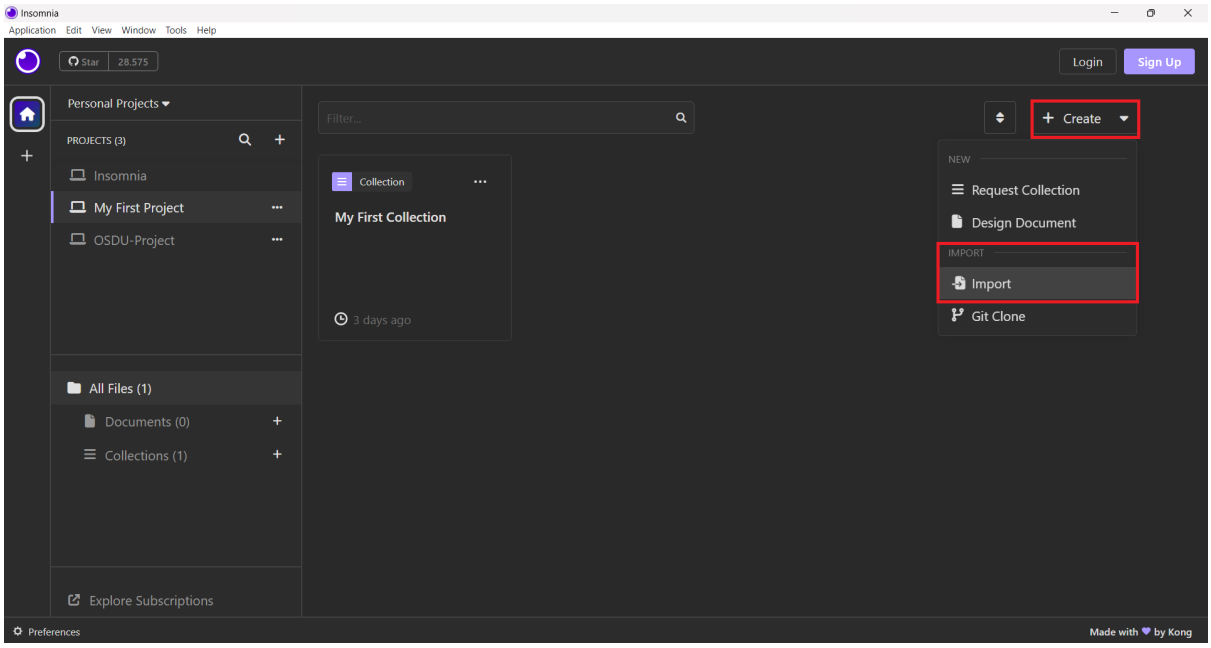

Figure 16: Insomnia: Select import option

In this tutorial, we will consider that you are using the provided code, which is why we do not detail the code from here.

3.1 Scenario 1: Data ingestion using pre-existing schemata

Let's start with the most basic scenario, which is to leverage users, legal tags, and schemata pre-existing for uploading files into the OSDU data platform. Ingestion into the OSDU can be made in two different ways:

- **Data (files) ingestion:** heterogeneous files are uploaded into OSDU, such as CVS, PDF, DOC, LAS, XML, OpenVDS, SEGY, TXT, and so on.
- **Metadata ingestion:** data that describes the ingested file to make it searchable, that is, JSON records that are based on the schemata standard.

3.1.1 Data ingestion using File Service API

The file service allows us to upload heterogeneous files into the OSDU data platform.

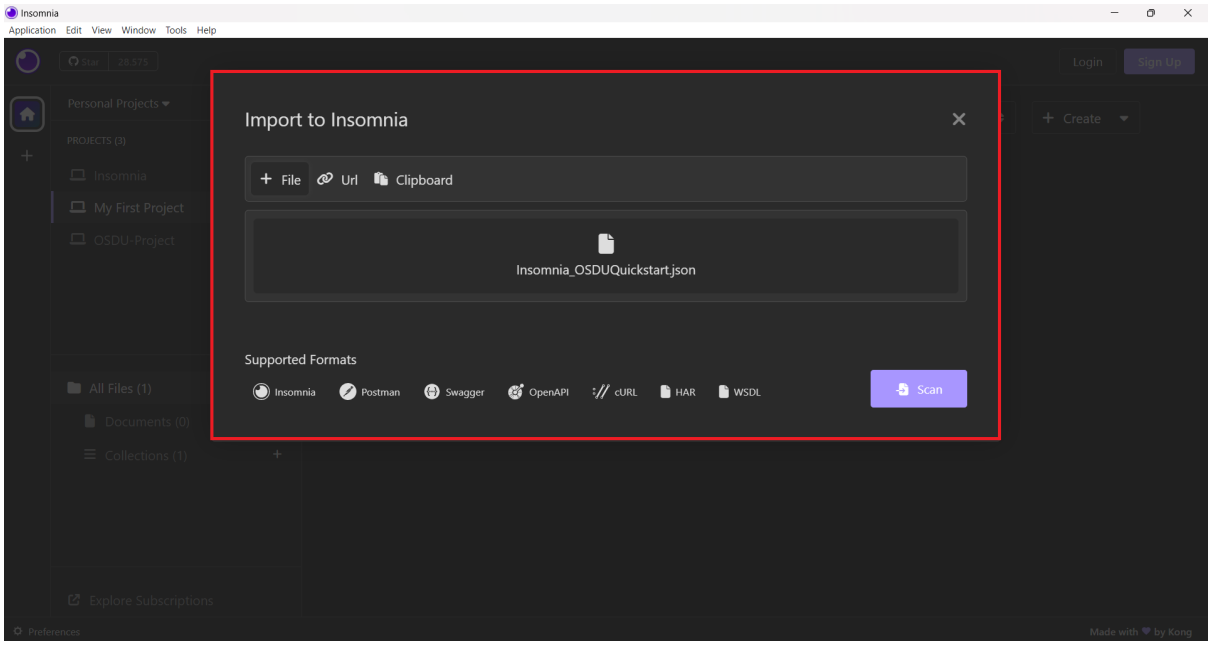

Figure 17: Insomnia: Select the file

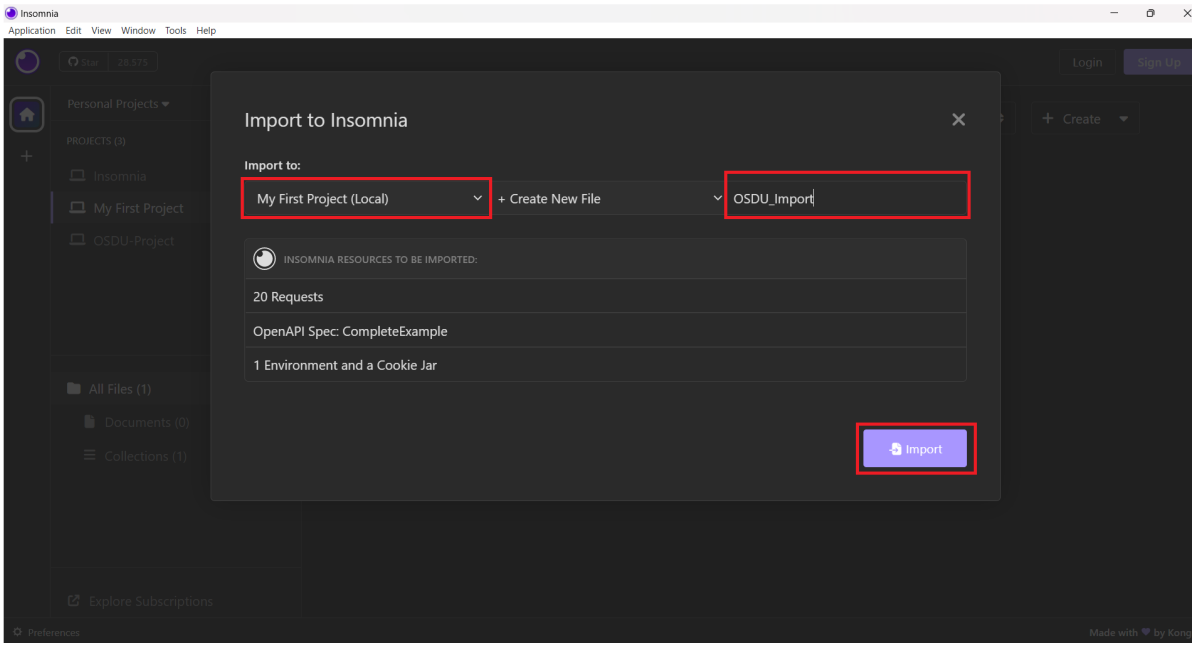

Figure 18: Insomnia: Importing the code

Step 1: Get File upload SignedURL. The value of *SignedURL* should be stored as **upload_signedURL** variable, and *FileSource* should be stored as **file_source** variable. Figure 20 shows an example of the request and the response obtained.

Step 2: Upload your file. Figure 21 shows an example of the PUT method using the *SignedURL* variable as a parameter. Remember to change the request body type to a *Binary file*. Set the header as (Content-type = text/csv) or (X-ms-blob-type= BlockBlob). Status 200 indicates that the upload is a success. As a result of this step, the file is moved into the OSDU Staging zone (Compute area).

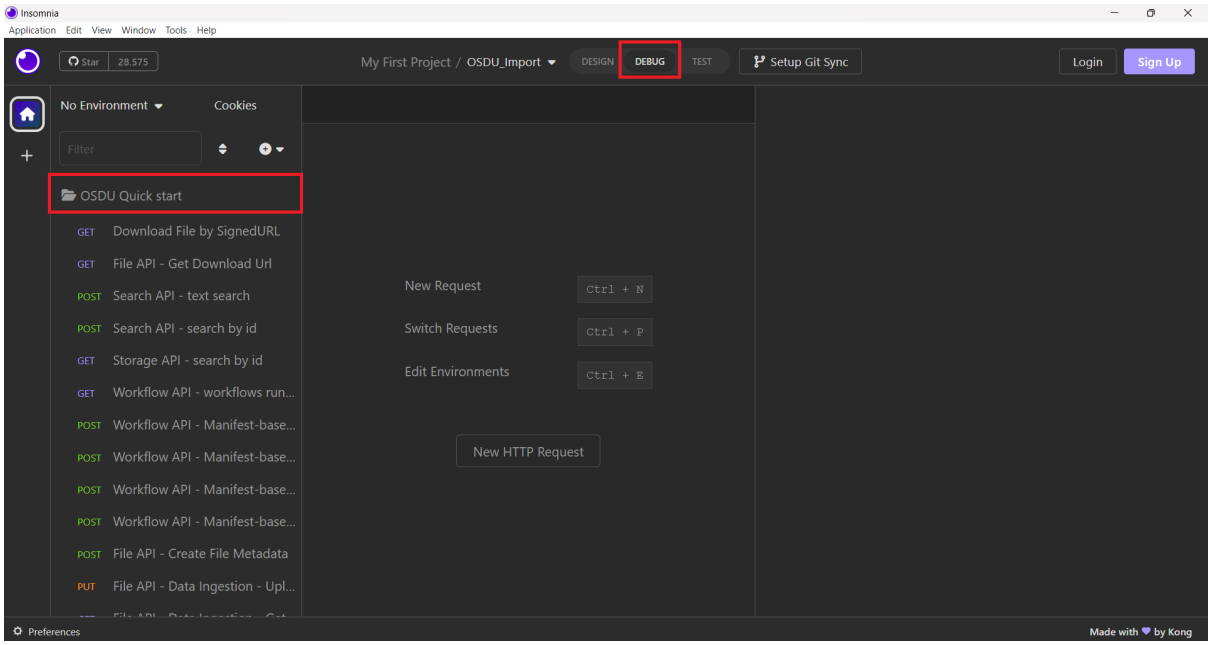

Figure 19: Insomnia: Select debug view

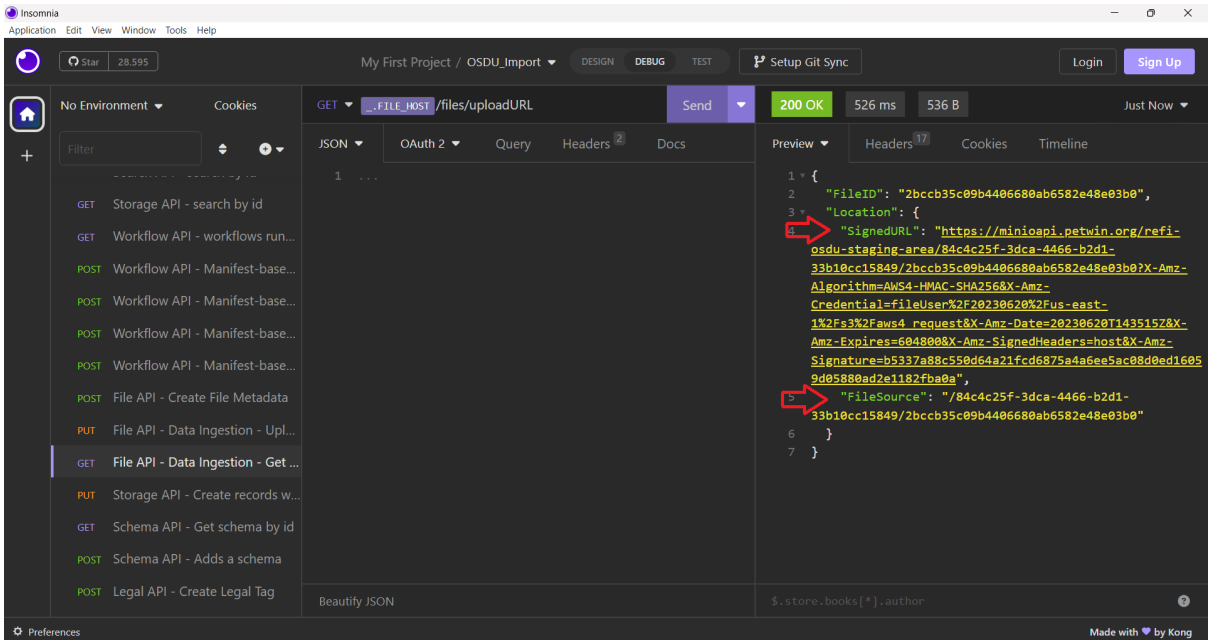

Figure 20: Get File upload SignedURL

3.2 Metadata data ingestion

To make data (ingested files) discoverable (searchable by OSDU Search service), metadata should be stored in OSDU. All metadata that can be stored in OSDU is described by `https://gitlab.opengroup.org/osdu/subcommittees/data-def/work-products/schema/-/tree/master/Generated`.

The OSDU metadata can be one of several types:

- **Master data:** Slow-changing O&G entities, this data does not have an association with the specific file. A full list of OSDU-supported Master data can be found here.
- **Work Product Component (WPC):** Metadata associated with the specific file

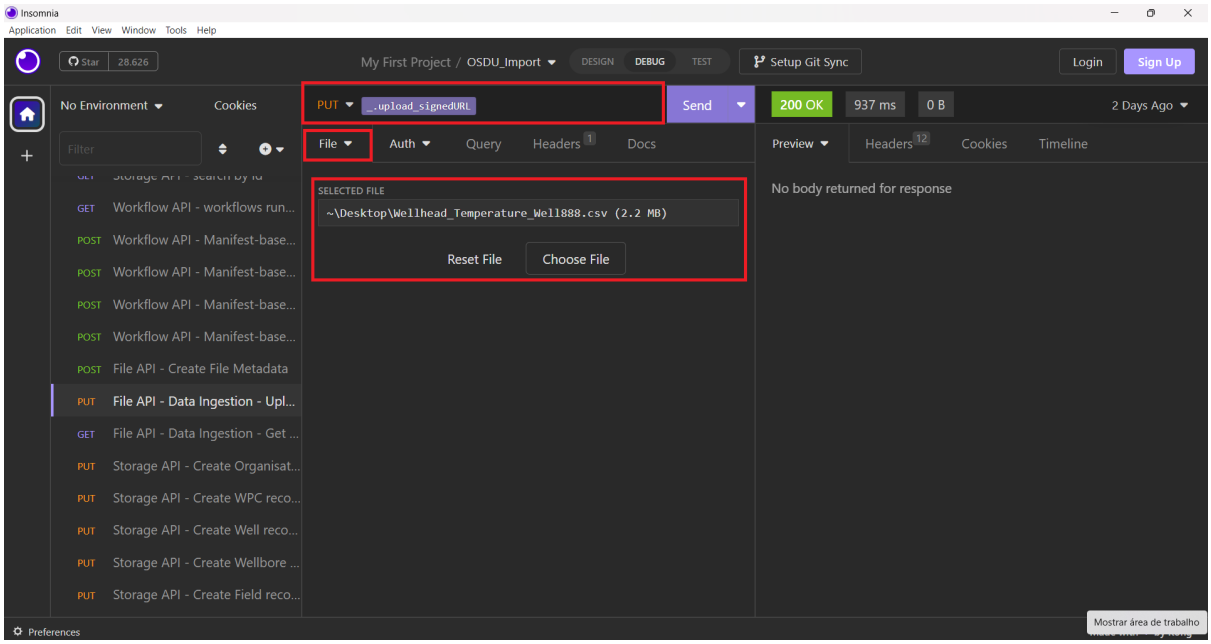

Figure 21: PUT SignedURL

or file collection. It is changing in time data, thus, WPC may have different versions, each version describing the snapshot of the raw data at a certain point in time. WPC must have a reference to one or more Dataset records that describe the file or file collection. A full list of supported WPCs can be found here.

- **Reference data:** Think about it as drop-down values to describe Master or WPC attribute values (e.g., units of measures, list of countries, etc.). By governance type, reference data can be: (i) fixed (values are provided by OSDU; (ii) open (values are provided by OSDU, but an operating company can add its own values); (iii) local (schema is provided by OSDU, but values should be provided by an operating company). See the full list of reference data schemas with their governance type here.
- **Dataset** Schemata that describe properties of a physical file (this file does not describe the logical entity!). They can describe files of a specific type (e.g., .tiff, .segy) or any type (File.Generic kind). Each Dataset record must have a link to the physical file location that can be used to download the file from the OSDU Storage service. Schemata that describe file properties can be found here.
- **Work Product (WP):** Think of it as a project or collection of related files and datasets. Work Product is an abstract entity that allows the organization of different WPCs and datasets. You can have any number of WPCs and Datasets in your WP.

There are several ways of storing metadata in the OSDU; however, in this tutorial, you will learn how to ingest metadata using the File service.

Figure 22 shows an example of how to create metadata ingestion related to the recently uploaded file. Please note that the id parameter is not specified in the request, and it was assigned automatically by the Storage Service. You may choose to specify the *id* parameter in case you want to assign a specific id. As a result of this step, the file is moved from the Staging area to the Persistent area, and the metadata record for the file is created.

So far, all you have done has been upload a file and creating a related metadata

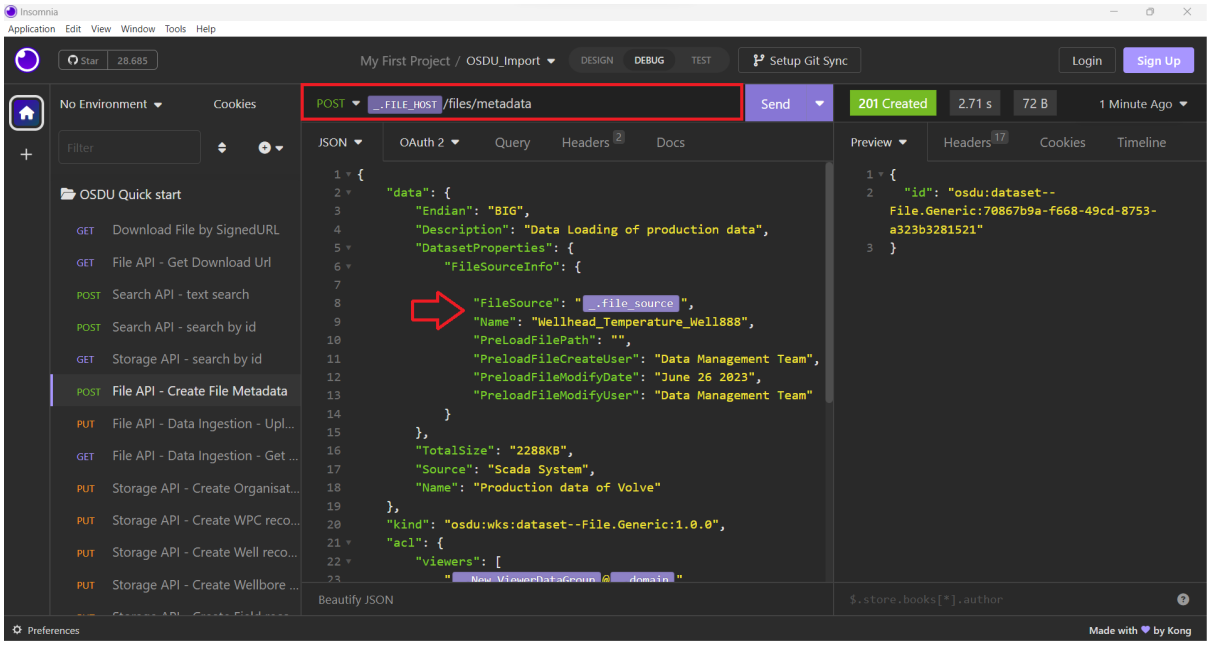

Figure 22: File API: Create File Metadata

record. Therefore your next step is to create metadata records related to this file. As mentioned before, the OSDU data platform has different types of metadata to represent other things or entities. For a suited contextualization, you must create at least WPC and Master Data records to describe details about the business content of the temporal series uploaded and its related entities, such as field, basin, well, wellbore, and organization, time-series.

3.2.1 Creating Field Master Data record

Figure 23 presents the creation of field type master data record. Note that is used the storage service endpoint to create new metadata records. You must define the *id*, and its value must be unique. If your request has been correct, your response will be (201 created).

3.2.2 Creating Basin Master Data record

Figure 24 presents the creation of a Basin type master data record. You must define the *id*, and its value must be unique. Besides, in *GeoContext* block highlighted in red, is where you will put the id of the field record recently created. This reference declares the relationship between the field and basin entity. If your request has been correct, your response will be (201 created).

3.2.3 Creating Organisation Master Data record

Figure 25 presents the creation of an Organisation type master data record. This record is created to represent the company or operator involved in the oil production process. If your request has been correct, your response will be (201 created).

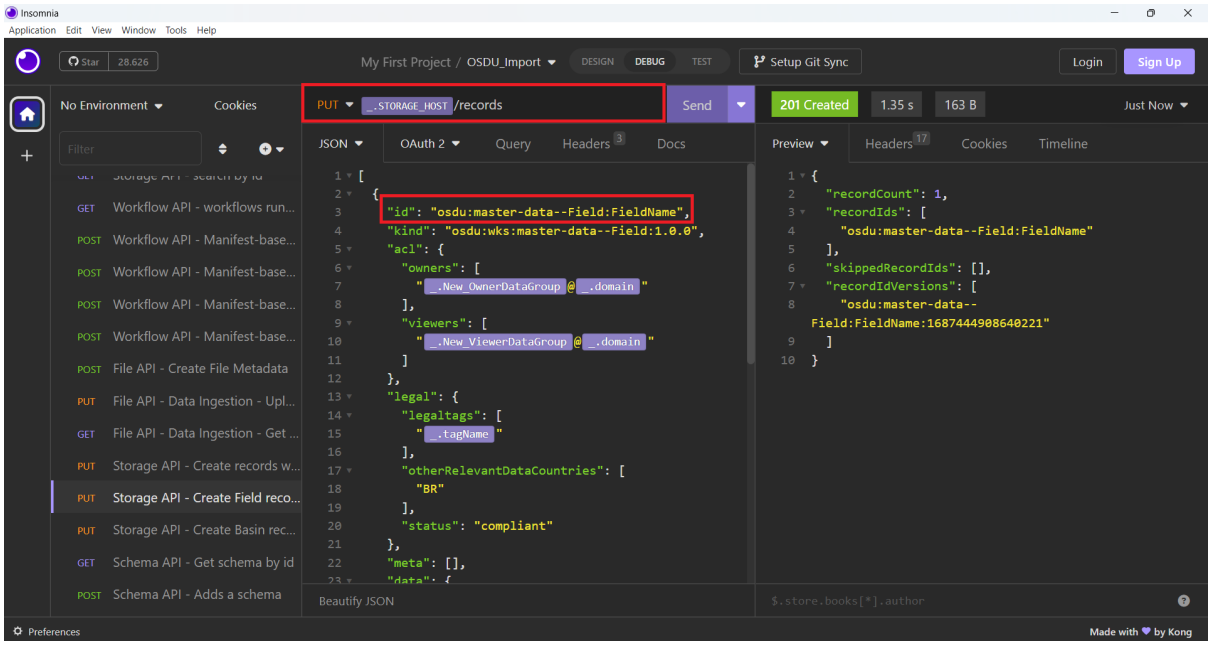

Figure 23: Storage API: Create Field Master Data record

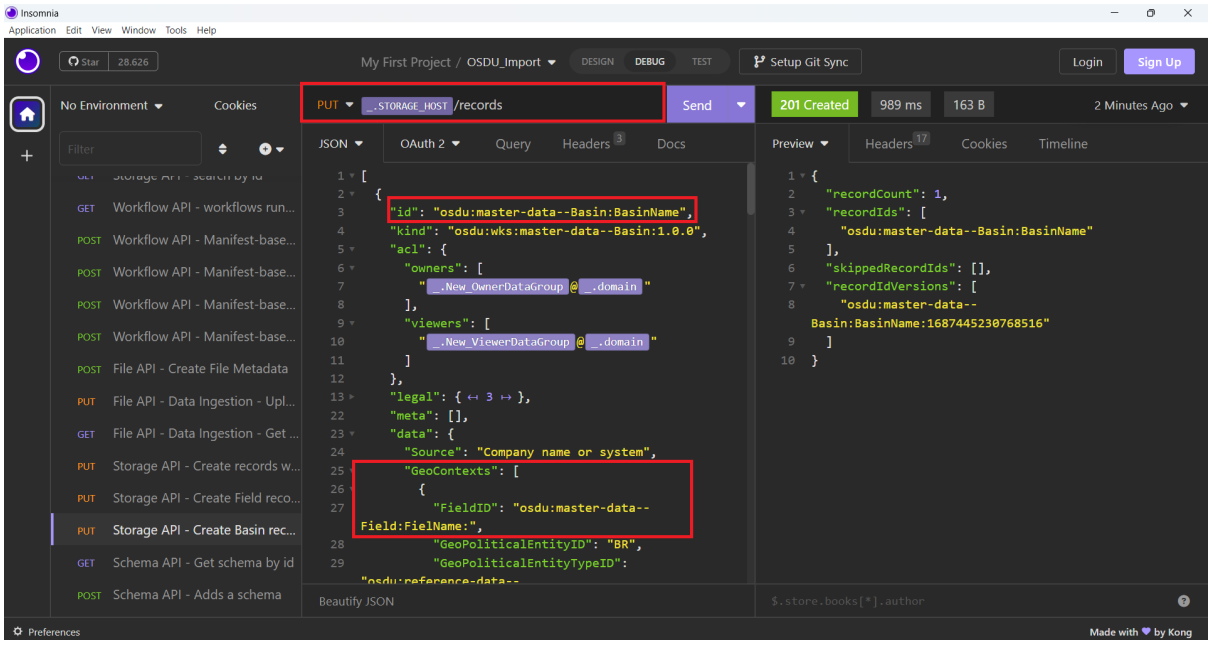

Figure 24: Storage API: Create Basin Master Data record

3.2.4 Creating Well Master Data record

Figure 26 presents the creation of a Well type master data record. You must define the *id*, and its value must be unique. Besides, in *GeoContext* block highlighted in red, is where you will put the id of the basin record recently created. This reference denotes the relationship between the basin entity. If your request has been correct, your response will be (201 created).

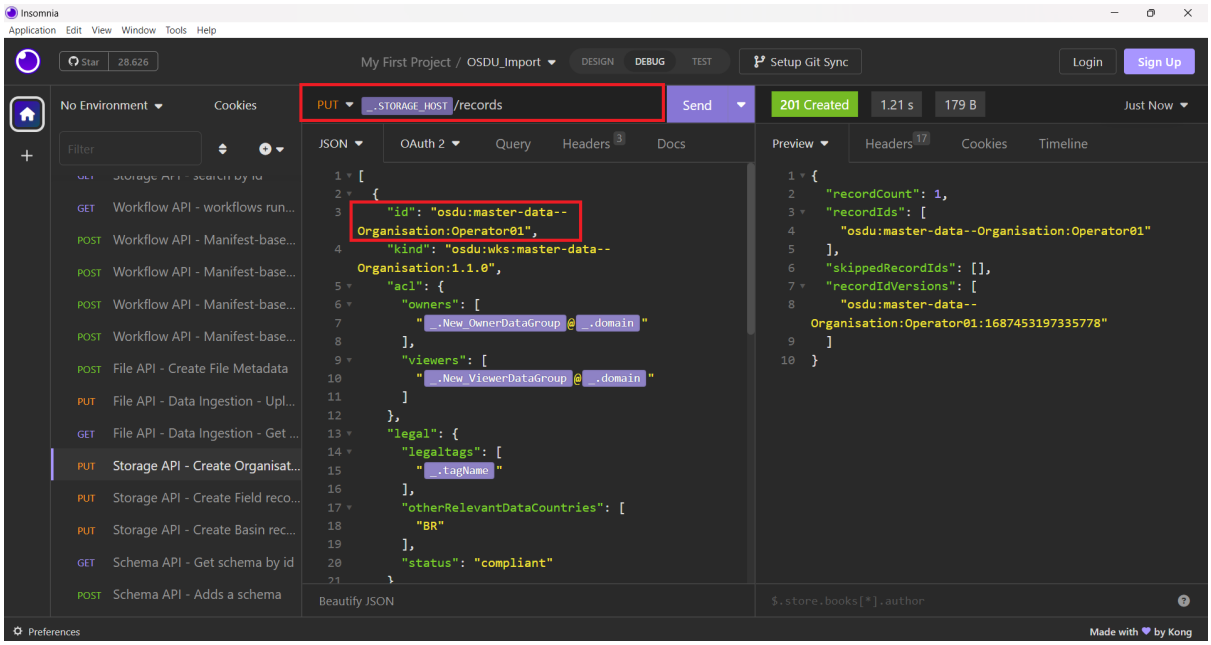

Figure 25: Storage API: Create Organisation Master Data record

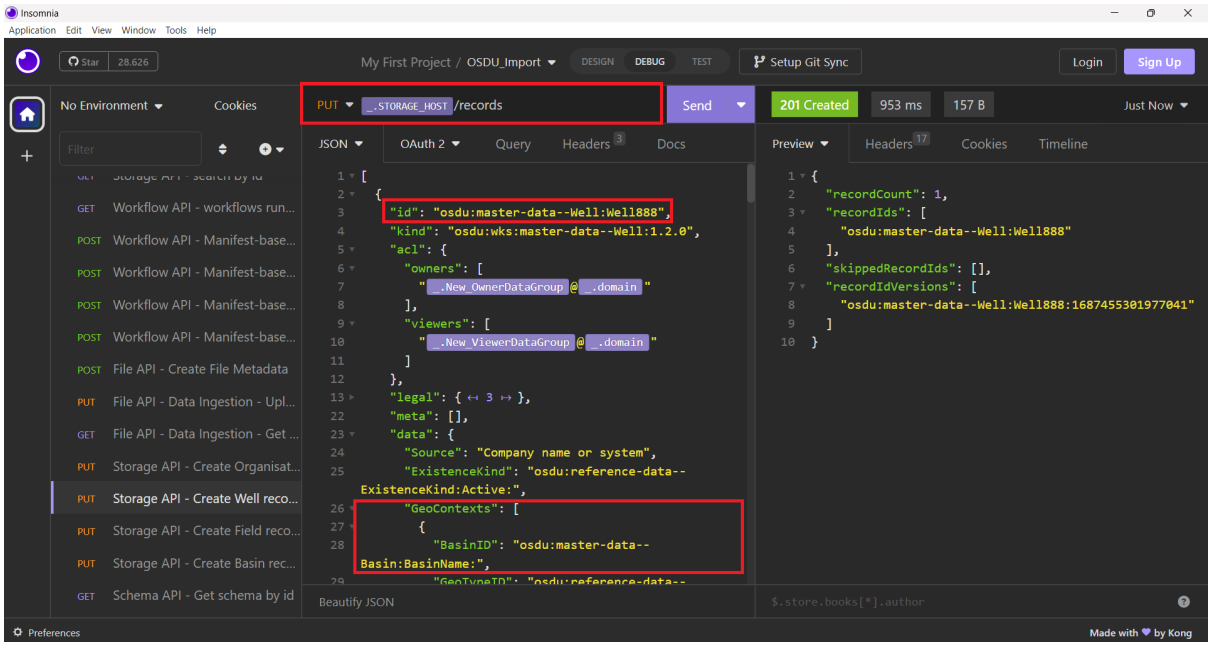

Figure 26: Storage API: Create Well Master Data record

3.2.5 Creating Wellbore Master Data record

Figure 27 presents the creation of a Wellbore type master data record. You must define the *id*, and its value must be unique. This record declares a relationship with basin, field, organization, and well entities. Besides, it is here where we define the functionality of the wellbore if it is for production or injection, drilling start, drilling finish, and other inherent information. If your request has been correct, your response will be (201 created).

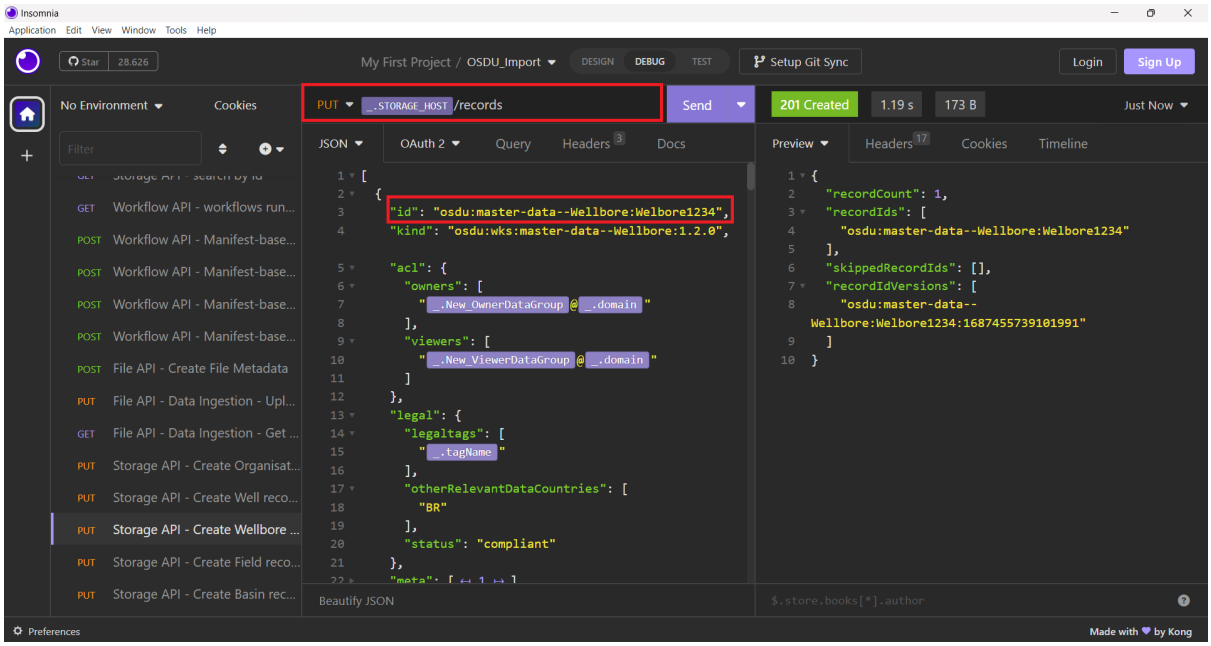

Figure 27: Storage API: Create Wellbore Master Data record

3.2.6 Creating Time series WPC record

Figure 28 presents the creation of a temporal series WPC. Again, don't forget to define *id* with an exclusive value. A temporal series WPC allows us to describe the commercial content of the CSV (temporal series) file and declare the relationship between it and the record of the *dataset* type. It is essential to highlight that only WPC has artifacts or relationships with raw files. In addition, it is through WPC that we describe the entities that generated the time series (raw data), in this case, *well* and *wellbore*. If your request is correct, your response will be (201 created).

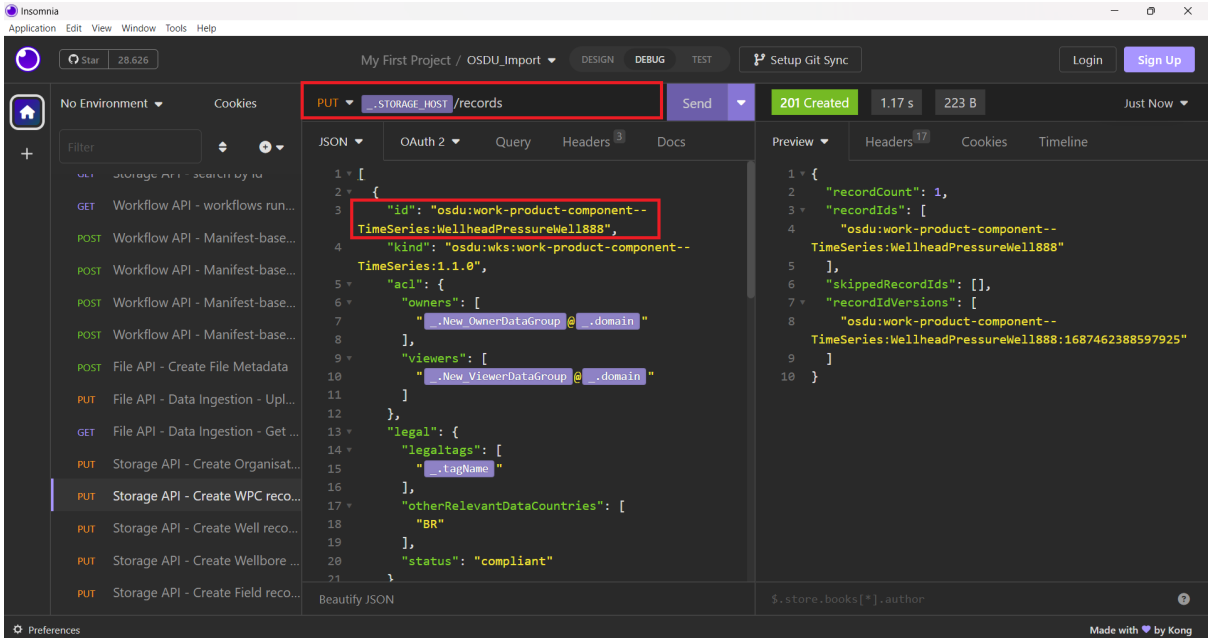

Figure 28: Storage API: Create Time series WPC

Finally, you finished the necessary metadata ingestion to a suited contextualization of a temporal series using the OSDU services endpoints. In this scenario, it was considered the pre-existing schemata to create the metadata records, legal tags, users, and groups. However, if you want to create a schema or a new user, these operations are demonstrated in the section 3.5.

3.3 Searching metadata

You can search the metadata through storage and search API. Figure 29 shows an example of searching using the storage API, and Figure 30 shows an example of querying using the search API. Note that the search using the storage API uses the GET method. However, the search API uses the POST method.

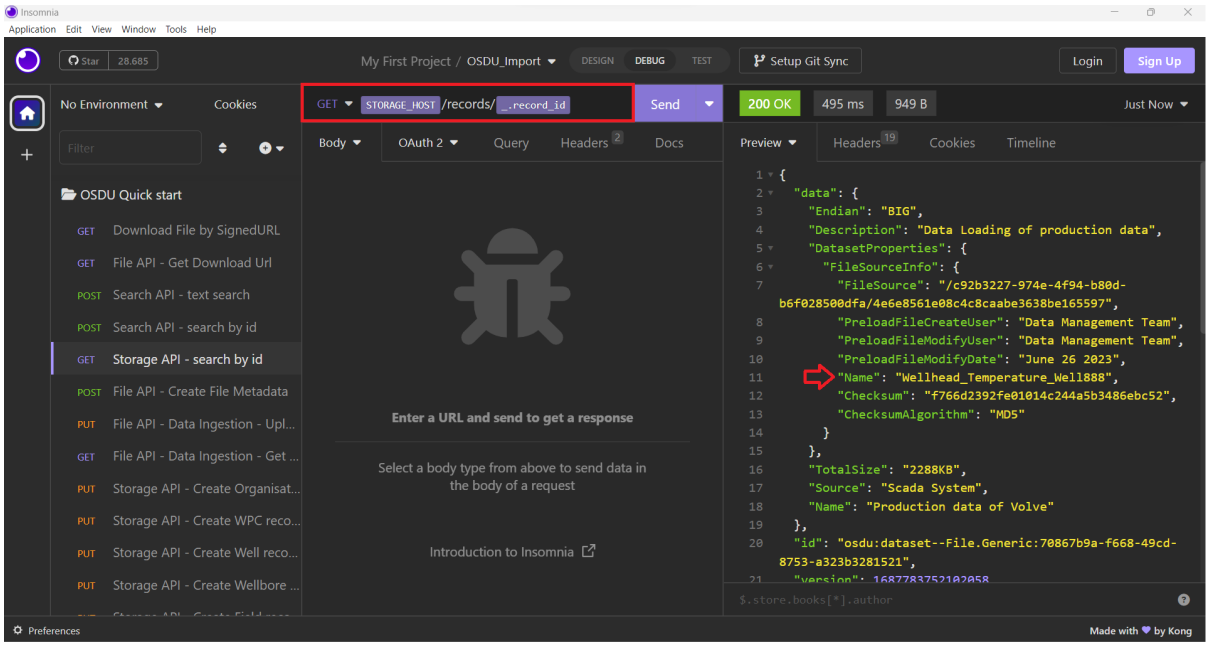

Figure 29: Storage API: Searching dataset metadata by id

3.4 Downloading the raw data

You may use similar services used for file ingestion to get an original file. First, you need to search for the metadata record of the dataset (see above). Then using the dataset record id, you may need to request file DownloadUrl using File service, Dataset service, or DDMSes. After executing this request, download the file using the provided link. Note that the *signedUrl* value was stored as an environment variable called *download_signed_url*. Besides, for the download request, you don't need to provide the authorization data. Figures 31 and 32 show the download example.

3.5 Scenario 2: Creating new legal tags, users, and schemata

In this section, we show you how to create a new legal tag, a user, and a schema.

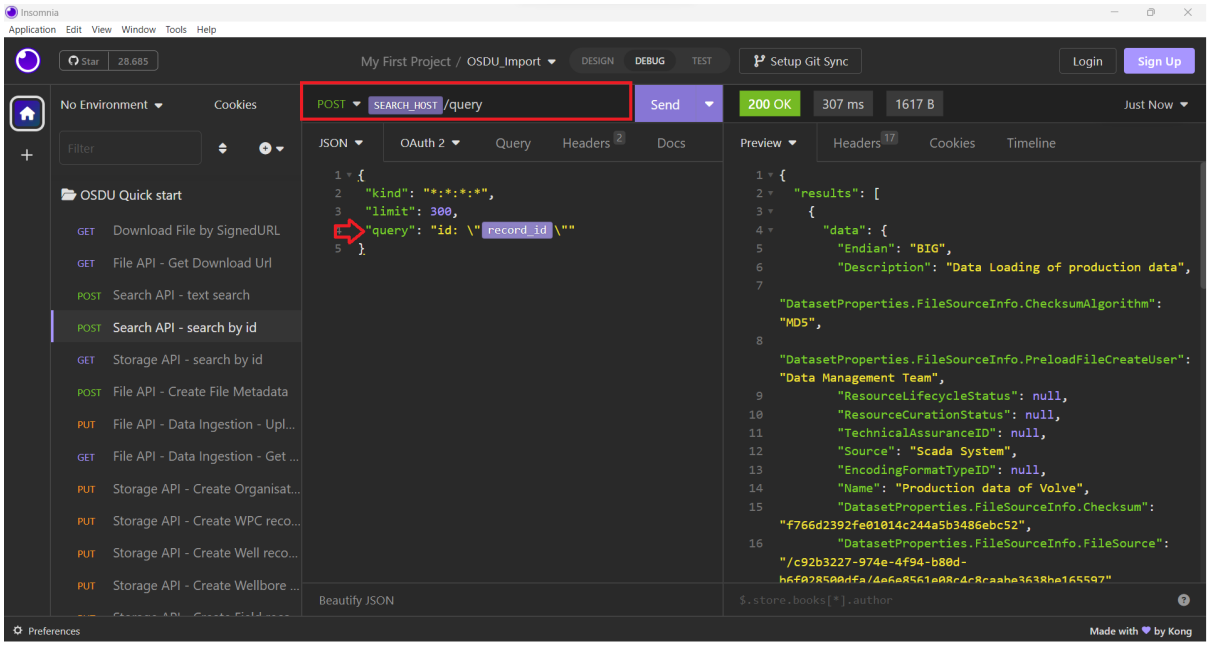

Figure 30: Search API: Searching dataset metadata by id

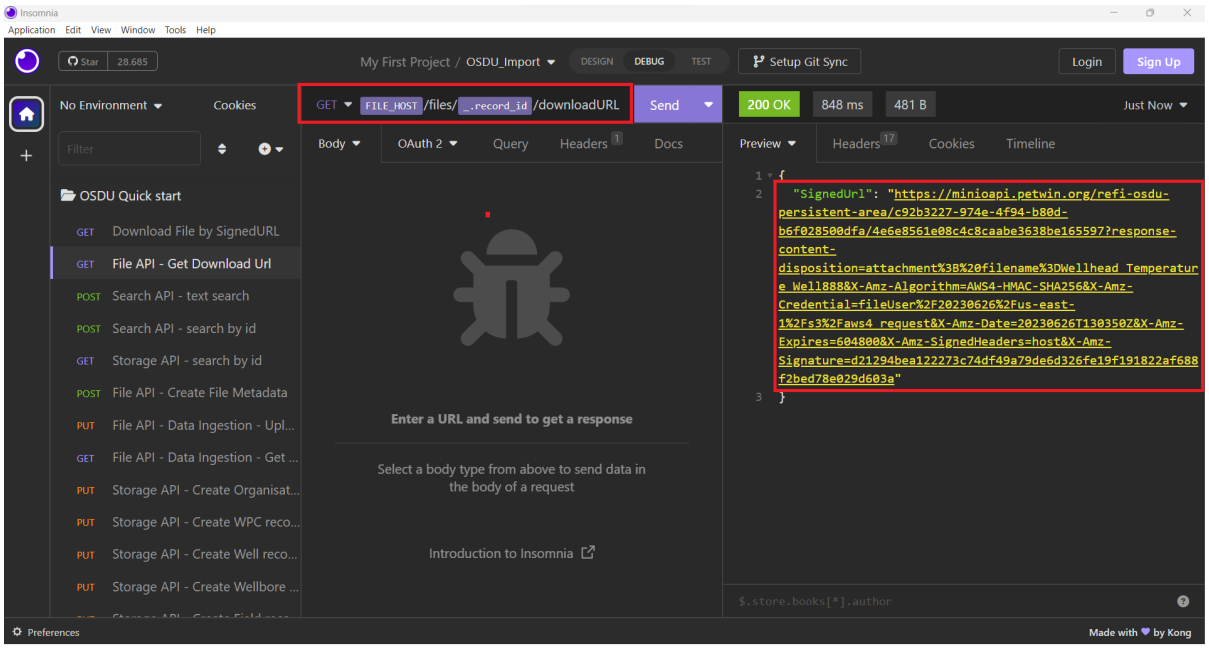

Figure 31: File API: GET download URL

3.5.1 Creating a Legal Tag

A LegalTag is an entity that represents the legal status of data in the Data Ecosystem. It is a collection of properties that governs how the data can be consumed and ingested. A legal tag is required for data ingestion. Therefore, the creation of a legal tag is a necessary first step if there isn't a legal tag already exists for use with the ingested data. The LegalTag name needs to be assigned to the LegalTag during creation and is used for reference. The name is the unique identifier for the LegalTag that is used to access it.

When data is ingested, it is assigned the LegalTag name. This name is checked for a corresponding valid LegalTag in the system. A valid LegalTag means it exists and has

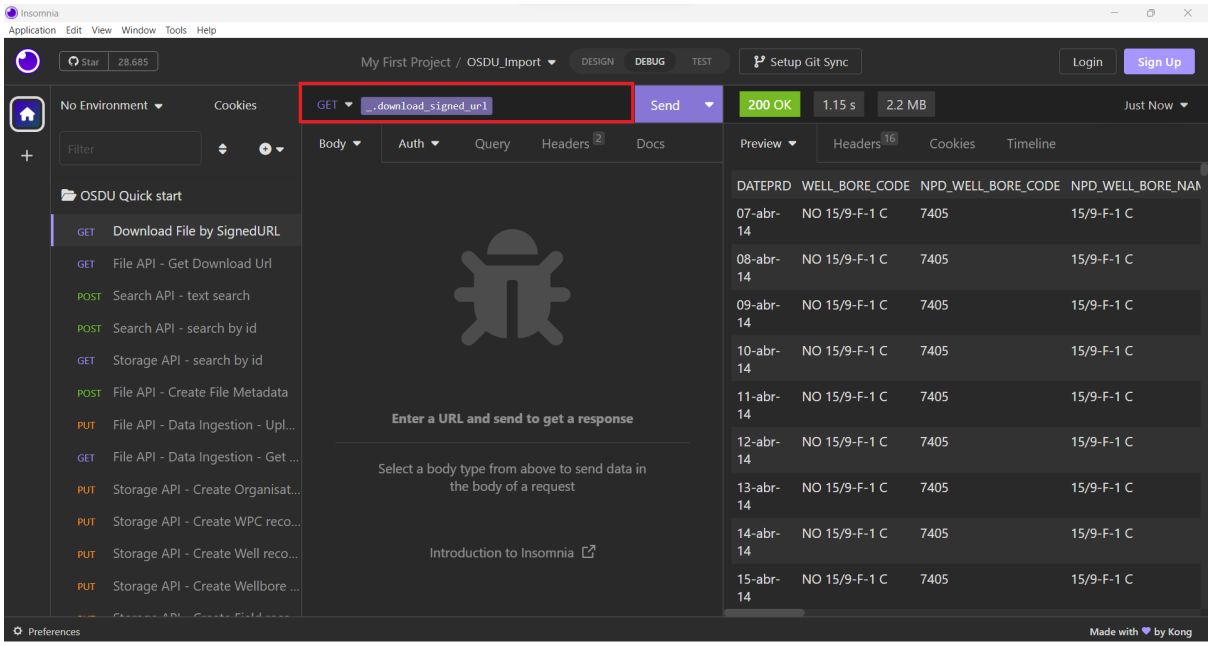

Figure 32: Download File by SignedURL

not expired. If a LegalTag is invalid, the data is rejected. For instance, we may not allow the ingestion of data from a certain country, or we may not allow the consumption of data that has an expired contract. In the same manner, the ingested data will be invalidated (soft-deleted) when the legal tag expires, as it would no longer be compliant.

Any data being ingested needs a LegalTag associated with it. You can create a LegalTag by using the POST LegalTag API (e.g., POST /api/legal/v1/legaltags). Figure 33 shows an example of LegalTag creation. On the right side is the (201 created) response.

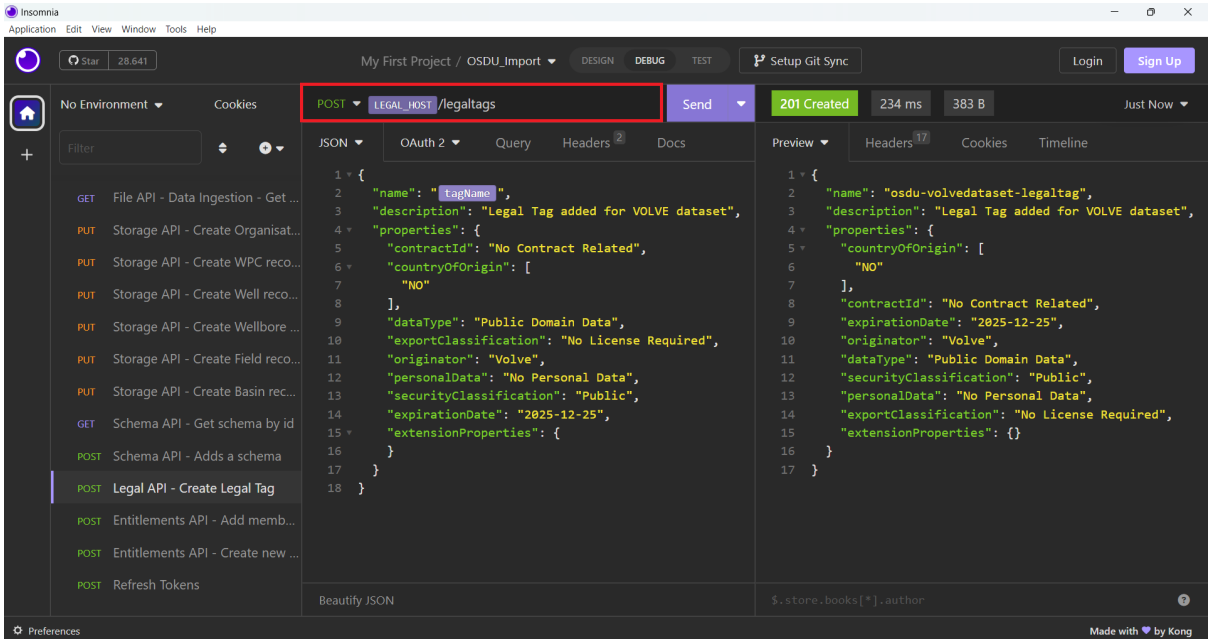

Figure 33: Legal API: Creating a Legal Tag

It is good practice for LegalTag names to be clear and descriptive of the properties it represents, so it would be easy to discover and associate with the correct data. Also,

the description field is a free-form optional, allowing you to add context to the LegalTag, making it easier to understand and retrieve.

The "extensionProperties" field is an optional JSON object field, and you may add any company-specific attributes inside this field. When creating LegalTags, the name is automatically prefixed with the data-partition-name that is assigned to the partition. So in the example above, if the given data-partition-name is my-partition, then the actual name of the LegalTag would be mypartition-demo-legaltag.

Valid values: The LegalTag name needs to be between 3 and 100 characters, and only alphanumeric characters and hyphens are allowed.

- **Expiration date:** Any date in the future in the format YYYY-MM-DD (e.g., 2099-12-25) or empty.
- **Originator:** Should be the name of the client or supplier. This is always required. This property is case-sensitive.
- **Data type:** See Table 1 for valid data types values. Different data types are allowed dependent on the data partitions e.g., vendor partitions have different governing rules as opposed to standard partitions.
- **Security classification:** 'Public', 'Private', 'Confidential'.
- **Export classification:** '0A998'(0 as Zero), 'EAR99', 'Not - Technical Data', 'No License Required'. We currently only allow data with the ECCN classification 'EAR99' and '0A998'(0 as Zero). This property is NOT case-sensitive.
- **Personal data:** 'Personally Identifiable', 'No Personal Data'. We do not currently allow data that is 'Sensitive Personal Information' and this should not be ingested. This property is NOT case-sensitive.

Table 1: Valid data types values

Data type	Data residency restriction
"Public Domain Data"	"public data, no contract required"
"First Party Data"	"partition owner's data, no contract required"
"Second Party Data"	"client data, contract is required"
"Third Party Data"	"contract required"
"Transferred Data"	"EHC/Index data, no contract required"

To help with LegalTag creation, it is advised to use the (GET /api/legal/v1/legaltags:properties) API to obtain the allowed properties before creating a legal tag. This returns the allowed values for many of the LegalTag properties.

You can get the list of all valid LegalTags using the (GET /api/legal/v1/legaltags?valid=true) API method. You can use this to help assign only valid LegalTags to data when ingesting. Figure 34 shows the valid LegalTags.

3.5.2 Creating a new group and user

Entitlements service is used to enable authorization in OSDU Data Ecosystem. The service allows for the creation of groups. A group name defines the permissions. Users who are added to that group obtain that permission. The main motivation for entitlements service is data authorization, but the functionality enables three use cases:

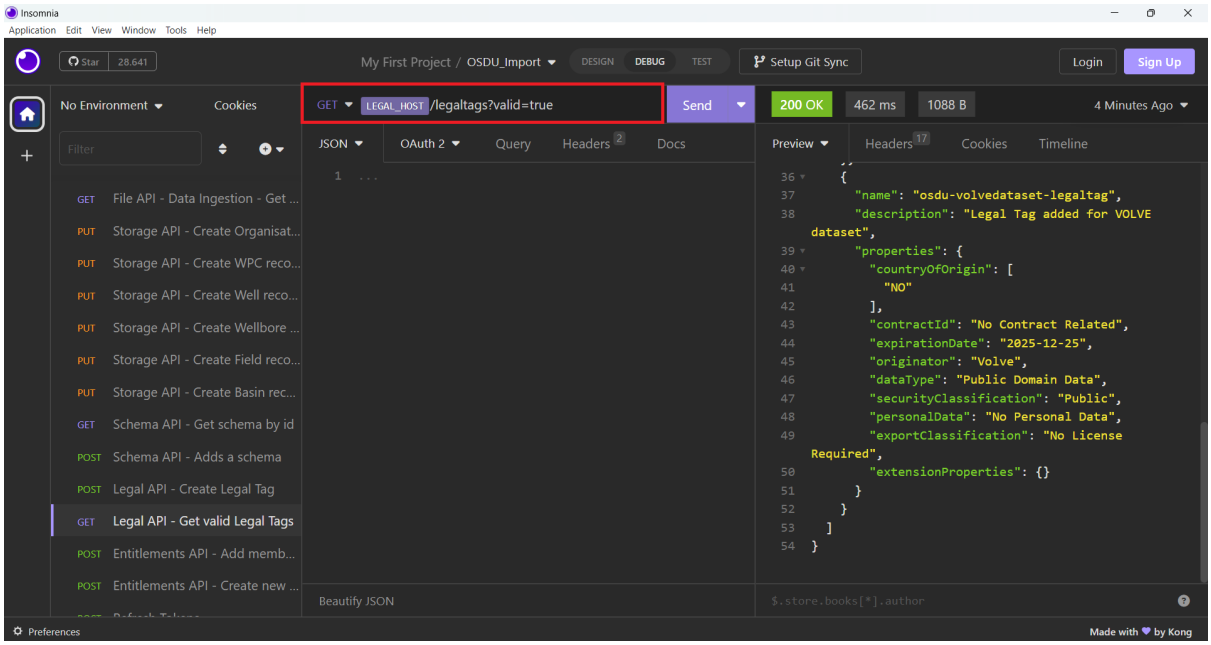

Figure 34: Legal API: Getting valid Legal Tags

- Data groups used for data authorization, e.g., *data.welldb.viewers, data.welldb.owners.*
- Service groups used for service authorization, e.g., *service.storage.user, service.storage.admin.*
- User groups used for hierarchical grouping of user and service identities, e.g., *users.datalake.viewers, users.datalake.editors.*

For each group, you can either be added as an OWNER or a MEMBER. The only difference is if you are an OWNER of a group, then you can manage the members of that group.

POST /entitlements/v1/groups - Creates the group within the data partition provided in data-partition-id header. This API will create a group with the following email name@data-partition-id.domain.com, where *data-partition-id* is received from the *data-partition_id* header. The user or service extracted from JWT in the Authorization header made an OWNER of the group. The user or service must belong to *service.entitlements.ad-min@data-partition-id.domain.com* group. This API will be mainly used to create service and data groups. Group creation guidelines:

- **Data groups** used for data authorization, e.g. of group name is: *data.resourceName.-permission@data-partition-id.domain.com*
- **Service groups** used for service authorization, e.g. of group name is: *service.service-Name.permission@data-partition-id.domain.com*
- **User groups** used for hierarchical grouping of user and service identities, e.g. of group name is: *users.serviceName.permission@data-partition-id.domain.com*

Figure 35 shows an example of how to create a new group using the Entitlement API.

POST /entitlements/v1/groups/group_email/members - Adds members to a group with group_email within the data partition provided in *data-partition-id* header. The member added can either be a user or a group. E.g. *group_email* value is *name@data-partition-id.domain.com.* Member body needs to have an email and role for a member. Member role can be OWNER or MEMBER. The user or service extracted from JWT in the Authorization header checked for OWNER role membership within group_email or within *users.datalake.ops* (ops user) group. In case the user is not ops user, it should be

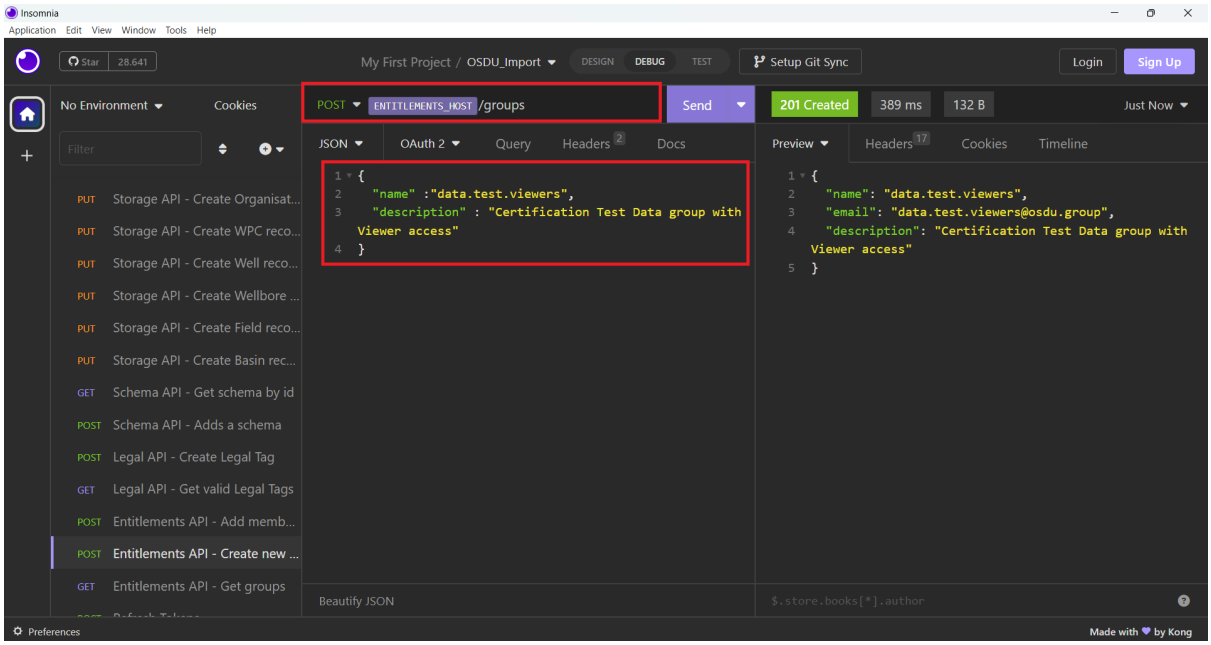

Figure 35: Entitlement API: Creating a new group

within service.entitlements.user or service.entitlements.admin group.

Figure 36 shows how to create a new user and add to a group recently created.

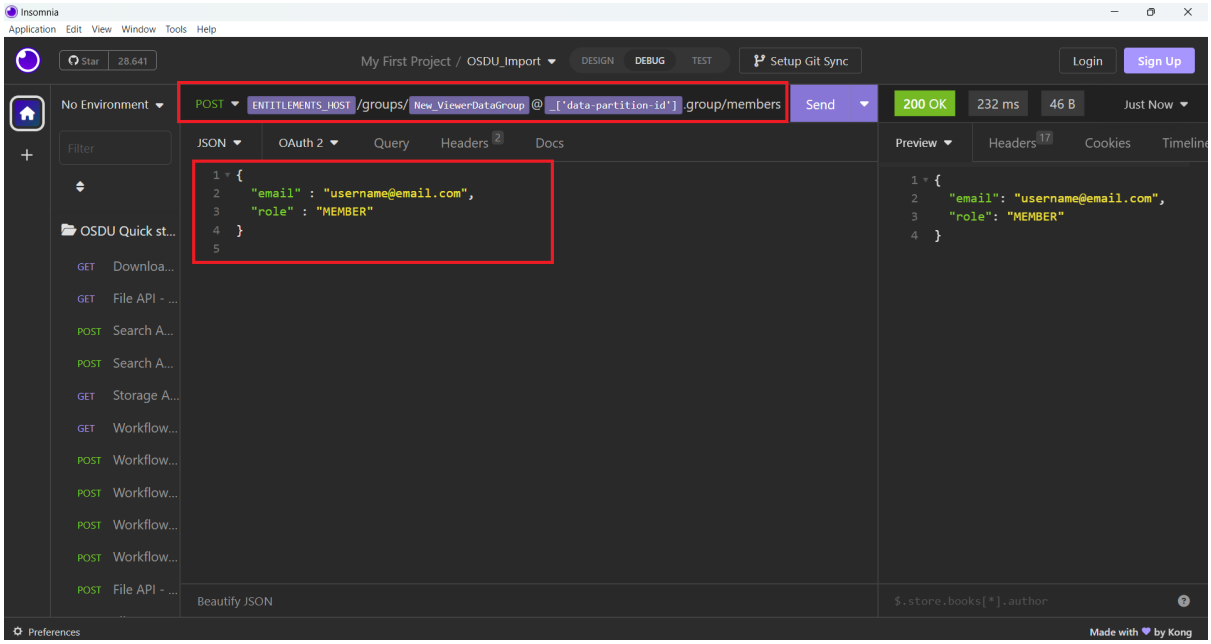

Figure 36: Entitlement API: Creating a new user and adding to a group

3.5.3 Creating a Schema

Note: The schema service only creates or updates schemata. It is impossible to delete a schema.

The schema service is used for data model definition. In other words, the schema defines whether a given field in the record is a string, an integer, a float, a geopoint,

etc. Schema Service allows data models to be defined in rich JSON objects. Although in NoSQL structures, we don't need that all records have the exact attributes, the schema service works as a way to create and manage a standard data model for all data. You can use the "Well" schema attributes as an example to create the records of the kind Well, for instance.

Different from the records, creating a schema implies declaring the *SchemaInfo* block, *createdBy*, *scope*, and *status*. Figure 37 shows the information related to schema identity, authority, source, entity type, and version. All of these are required for schema creation. More details are available at the link `https://community.opengroup.org/osdu/platform/system/schema-service/-/blob/master/docs/SchemaService-OSDU.md`.

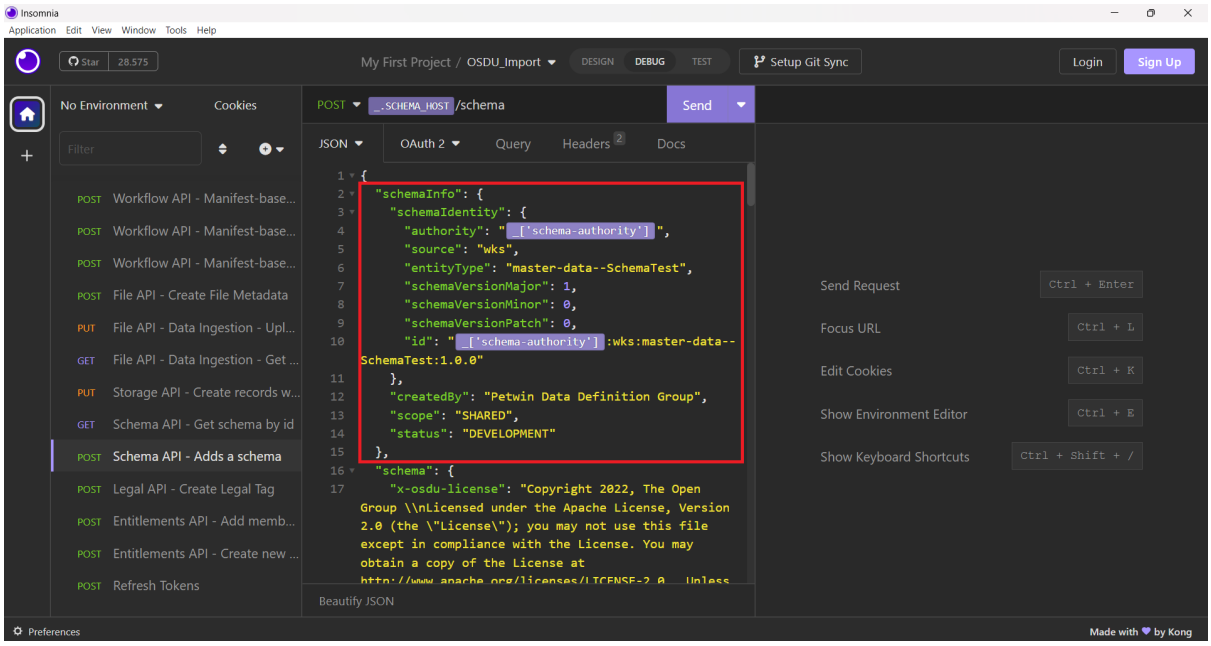

Figure 37: POST: Schema creation