Data Modelless Microservices to increase Multi-Tenancy in BaaS and SaaS Providers with Application to a Covid-19 Data-Lake

Júlio G. S. F. da Costa
Grad. Prog. in Elec. and Comp. Eng.
Univ. Fed. do Rio Grande do Norte
Natal, Brazil
juliogustavocosta@gmail.com

Francinaldo A. Pereira
Grad. Prog. in Elec. and Comp. Eng.
Univ. Fed. do Rio Grande do Norte
Natal, Brazil
falmeida@dca.ufrn.br

Davi H. dos Santos
Grad. Prog. in Elec. and Comp. Eng.
Univ. Fed. do Rio Grande do Norte
Natal, Brazil
davihenriqueds@gmail.com

Samuel X. de Souza
Grad. Prog. in Elec. and Comp. Eng.
Univ. Fed do Rio Grande do Norte
Natal, Brazil
samuel@dca.ufrn.br

Luiz M. G. Gonçalves
Grad. Prog. in Elec. and Comp. Eng.
Univ. Fed. do Rio Grande do Norte
Natal, Brazil
lmarcos@dca.ufrn.br

***Abstract*—This work is the result of the joint efforts of professionals encouraged to build a solution to predict the contagion and death curves of the Covid-19 pandemic, through the use of data-oriented solutions. This strategy is fundamentally dependent on collection. Regarding this particular aspect, the difficulty is manifested due to the fact that such data exist scattered in different repositories, in different formats, commonly available through files, in addition to being frequently updated. However, in this context, small data scientist teams with few resources suffer in this scenario forced to personally concern themselves with these difficulties. Here, we present a platform that helps these professionals to hide the complexities of having to deal with these issues themselves. This is done by creating and helping to manage, in an automated way, repositories for your users for simplified data consumption and distribution.**

***Index Terms*—MDE, Covid-19, Pandemic, Data Repository, Multi-tenant Service, Model Interpretation**

I. Introduction

During the Covid-19 pandemic scenario, engineers and scientists endeavored to collaborate with society in order to unite efforts to tackle the causes and consequences of the pandemic. One of the main strategies employed was the use of Artificial Intelligence (AI) to give those involved some degree of predictability regarding the behavior of the curves of contagions and deaths. In this context, we became involved in offering a solution based on the use of data-driven models capable of predicting the behavior of both curves. In the effort to build an epidemiological forecasting service for Covid-19 based on Machine Learning (ML), the workflow was divided into tasks carried out by different researchers, with different skills. We structured our process as normally done in the literature [2], [3], despite the fact that there is still no consensus on which is the best flow in terms of defining its stages and the roles of professionals involved. More specifically, the steps defined were data acquisition, model construction, model training, and testing, model evaluation, and deployment. In this work, we deal with issues related to Data Acquisition.

This work is sponsored by CAPES under grant 88887.506889/2020-00.

The most common approaches among professionals involved in this process, related specifically to the Covid-19 pandemic context, especially in the case of small teams, normally resulted in the creation of local repositories. These repositories, as a rule, consist of files of different formats stored in directories of local file systems. These approaches imply at least two difficulties: inefficient searching for content between files and sharing. On the other hand, regarding data and their metadata, two important aspects are the *dynamicity* [1] (the data and its metadata change frequently) and the *spread* of these in the most diverse repositories, which implies difficulties with *availability* and *accessibility*, in addition to their assembly in *consistent datasets* [2].

To face these challenges, we focused on building a service platform for the purpose of data persistence into local repositories (database oriented) capable of completely abstracting the need for its users (data scientists occupied with training and testing models) to deal with the creation and management of data repositories and its schemas and even less with the related infrastructure aspects.

Furthermore, we chose the "as a Service" delivery model for this platform. This choice is a consequence of the need to offer a good level of accessibility and availability. Yet, it serves a diversity of users (data scientists, in particular) with different interests and applications such as exploration, analysis, or prediction, for example. In this sense, each of these users will be able to use the platform at any time, so that they are able to easily create and maintain their own data schema, store data related to these schemas, and easily make them available for consumption.

II. Background

As mentioned, the choice of the “as a Service” (aaS) distribution model means that the software services provided by them are performed through computing clouds. This model can be directly associated with the engineering issues involved in building the Backend as a Service (BaaS) [4] delivery solutions, or the Software as a Service (SaaS) [5]. Such models offer the possibility for companies and their professionals to not need to have their own infrastructure and/or develop all the necessary software components for any of their software applications.

The main challenge for *aaS* platforms is to make them *multi-tenant*, that is, ideally, the resources of such platforms should be used in a shared way into a *single-instance*. However, providers of this type of platform need to take into account that there will always be unforeseen demands by these platforms that must also be met [6]. This means that if the demands of users are prioritized, resources are lost. On the other hand, coverage is lost for user applications. This important tradeoff implies the construction of *aaS* type platforms.

As users of this platform, data scientists externalize different demands as forecasting, exploration, and analysis, among others, necessary to carry out the general scope of this project, which is the construction of an efficient forecast of the contagion curves and deaths of the Covid-19 pandemic, through the use of data-driven methods. Each of these demands is performed as different uses, or applications, for the data persistence service of this platform. Furthermore, each of these uses, or applications, potentially may have to deal with different data schemas and data sets. In this sense, we consider each of these possible applications carried out on top of this platform as a tenant.

In the next, we will present more specifically the challenges related to the construction of an *aaS* platform and we will present the most used technologies for this purpose, as well as how they are used to overcome such challenges.

A. Problem Statement

As already mentioned, the biggest challenge is related to reconciling the demands of *aaS* type software services platforms for sharing and standardizing the use of resources, with the need of their users in circumstances of demands for customization of the use of the same resources. As the platform in question in this work is directly related to persistence services, therefore, responsible for storing and managing data, it must be concerned with the issues that directly imply them, which is the creation and management of data and data schemas. In this sense, the platform offers a mechanism that authorizes different data and data schemas, created and maintained by their respective users, to co-exist without conflicting, despite sharing the same software and hardware resources [11], [12].

Customizations *aaS* services are a central part of the concern for software engineers and architects, whether in the context of *aaS* for software services. This is because customizations are inversely proportional to the costs of carrying them out and, from the perspective of the software life cycle, they occur precisely in the phase in which software expenditures are most concentrated, the maintenance phase and evolution.

III. Related Work

Several technologies in the field of Software Engineering have been developed and used in order to enable the construction of software according to the Multi-Tenant paradigm. Next, we highlight a brief discussion about the technologies commonly used to provide solutions such as SaaS and BaaS platforms.

The use of *Extensible Programming and/or Adaptive Languages* [9], implies a strategy to solve the customization problems in order to make the software components more adaptable to changes in the software business domain (regard to data definitions or processes), with the advantage of these being carried out at runtime. It’s an interesting strategy for allowing changes without having to change the coding. However, for engineers and architects to take advantage of this potential, they need to accept that, if not all, of the microservices or components that perform SaaS, there must be a dependency relationship with those programming languages that implement the resource — this resource. is known as Meta Object Protocol (MOP) [13]. Examples of languages that enable this feature are Ruby and Lua. Furthermore, another technology that plays a relevant role in this context is Aspects Oriented Programming (AOP), in this sense see [16].

Microservices and *Components*, on the other hand, can be seen as a strategy to decompose large systems, more specifically in our case, large data classes hierarchies, into a coordinated set of small systems [10] or classes hierarchies. Commonly, this decomposition derives two subsets. The first core subset is not designed for customizations, whose updates occur only when changes are demanded, due to fault correction, or due to changes in the definitions of concepts and business processes. The second subset is designed with the customization needs of customers in mind, users of providers’ services — in this subset updates can also take place due to the demands of a client’s specific domain processes and concepts, in addition to this, generally, they imply the emergence of new instances of dedicated to the use of a specific client. Notably, to fulfill these commitments, intrusive and non-intrusive microservices are strategic categories that can be evaluated for use for customization purposes [12] e [11].

There is also the possibility of using the *Model Driven Engineering* (MDE) [8] mechanism as a strategy to facilitate the process of software modifications. Here, the paradigm of the software development and evolution process is entirely guided by modeling processes and business models of the application domain. Changes of any kind must occur in the models and these guide all subsequent phases, all of which are automated for automatic code generation. Ideally, all service instances resulting therefrom are obtained from mechanisms that automate the generation of code through the specifications put in the models – they instruct aspects that go from the architecture of the system to the deployment of components or microservices, going through the process business concepts and SaaS concepts, both for what is customizable and for what is not. Such models generally derive from the use of languages such as the Unified Modeling Language, through the use of its most varied diagrams [14] e [15].

IV. Data Modelless Microservices to increase Multi Tenancy

In this work, we use a software development approach [19] from the field of MDE, which guides the development of our persistence service so that it does not require any pre-determined data schema. This is important since data schemas cannot be predicted *a priori*. That is, only at runtime, and only during the duration of the request to the persistence service, will this service know which data schema should be used to guide the persistence task demanded by any of its tenants, for a set of any data. Given the end of the task, and since new data persistence requests are not required to use the same schema, the current schema can be dispensed with. In this way, we achieved the construction of a multi-tenant, single-instance persistence services platform.

As shown in Fig. 1, the service platform basically consists of a modeling service for defining data schemas, and another for the activity of persisting data related to such schemas (two microservices). This is necessary due to the approach described at the beginning of this section and used as a guide for the development and operation of this platform.

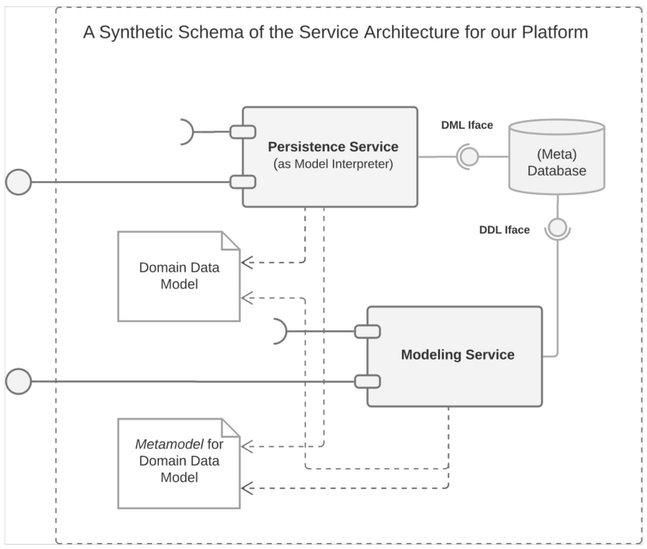


Fig. 1. Synthetic representation of the service platform architecture built from the requirements.

The schema modeling service is anchored in a modeling language and models generated from that language. Such a language can be similar to the Unified Modeling Language (UML), derived from it, or even a DSL built specifically to meet the specifics of this platform. The use of this tool results in the creation and management of the schema in terms of database deployment, and in the description of the data so that it can be persisted in these same databases.

On the other hand, it is the responsibility of the persistence service to recognize which data schema should be used at the time of the persistence request of any set of data, so that those are used in order to guide the requested operation on these and the completion of the request if perform successfully.

A. Implementation: Data Schema Modeling Service

As already highlighted, due to the software development approach used in this platform, a tool to assist in the modeling of data schemas for its users is mandatory. In general, however, this demand, within the scope of this platform, does not imply a significantly different use compared to current tools already known, in industry or in academia. Furthermore, the models generated by such tools, as long as they are exported in the XML Metadata Interchange (XMI) format [18] — something common among these tools —- can be consumed to fulfill the three purposes of using models for this service, namely: specification of data schemas, creation and maintenance of databases of these schemas. In the same way, it is common that the modeling tools already known are able to fulfill the same purpose.

Having highlighted these observations, we will refrain from delving into the issue related to the modeling activity of user data schemas. Furthermore, as the metamodels that define data schemas for the most varied databases (relational, non-relational, column-oriented, document-oriented, etc.) are widely known, we will only focus on leaving here the metamodel that determines for our platform the possibilities of modeling data schemas and, also, an example of a data schema model — these are really useful, in the sense that they can be used by any user of this modeling service to create a particular repository of data (database).

As shown in Fig. 2, at least three distinct data types can be modeled in a specific data schema and recognized by our services platform: the *Dataset* type, the *t*ype Class and a file of type *Text*. Here, the distinction of types occurs in order to deliver three possibilities that imply different ways of using data. *Dataset* is an appropriate data type for the data context in non-relational databases, more specifically in our case, a column-oriented database. *Class*, in turn, better serves data models with relational or non-relational characteristics oriented to documents, finally, *Text* file in case it is necessary to store and manipulate files of this format.

In the case of Fig. 3, it derives from the use of the metamodel presented in Fig. 2 in order to model, for any user of the platform, a data schema called *Natalnet*. This *schema* contains a data type *Dataset*, called *Covid19Infos*. This *Dataset*, in turn, is defined as a set of *Feature*s listed in the figure, in the symbol body of the *Dataset*.

Once the *Dataset Covid19Infos* modeling is completed, under the *schema Natalnet*, your user will be able to deploy this specification on the platform, whose unseen consequence is the creation of a repository capable of storing the data relating to this *schema*, which is carried out in a database.

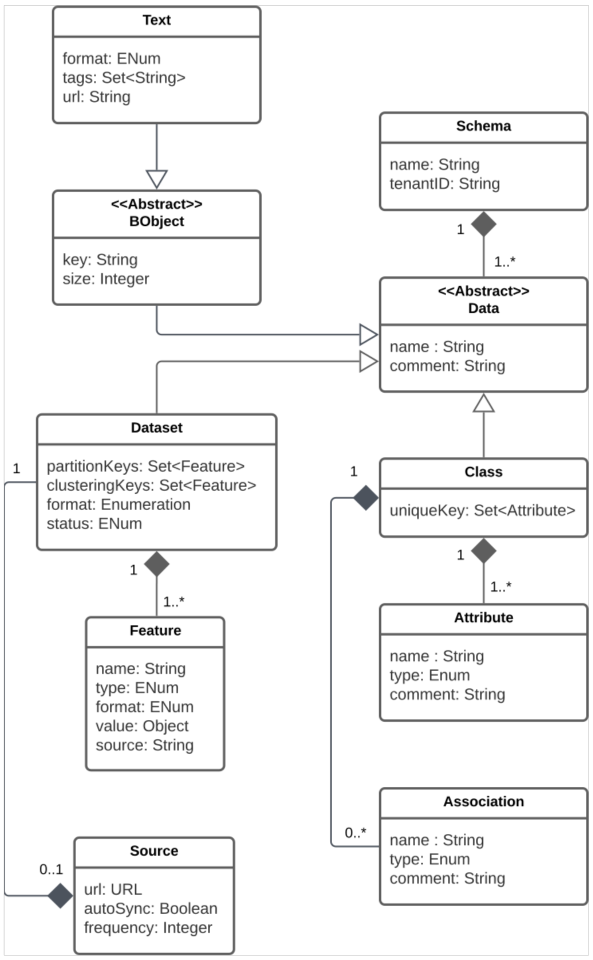


Fig. 2. Our platform metamodel for modeling and persistence services.

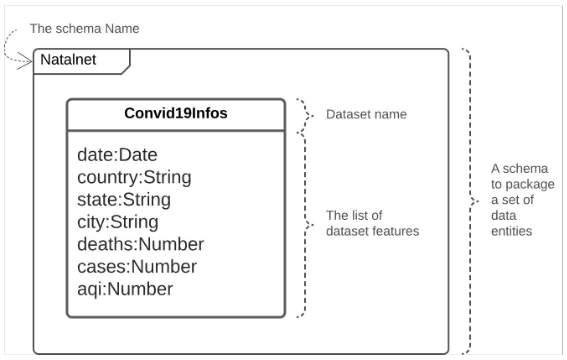


Fig. 3. An example of specifying a *Dataset* type, according to the metamodel shown in Fig. 2.

B. Implementation: Data Persistence Service

Like the modeling service, the persistence services are accessible through a single interface (in Fig. 4, the *Service Interface* element). The mechanism that realizes this interface is the *Object Persistence* microservice. Such a microservice must recognize which persistence action is being requested by the user, which type of object in the context of the model in which it is defined (in YCL) and then invoke the appropriate transform operation to build the persistence command, now, in the format of the underlying database's data manipulation language.

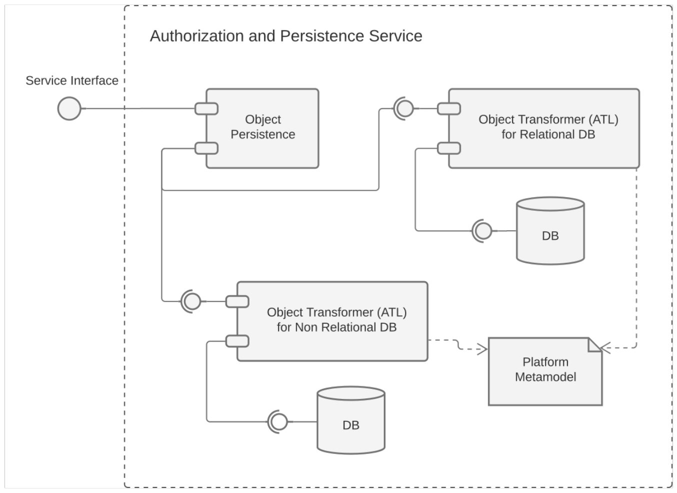

Authorization and Persistence Service
Service Interface
Object Persistence
Object Transformer (ATL) for Relational DB
DB
Object Transformer (ATL) for Non Relational DB
Platform Metamodel
DB

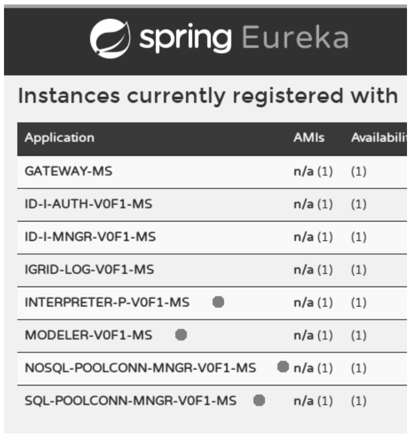
Fig. 4. Representation of the persistence and authorization service architecture.

The transformation operations necessary for the mapping referred to in the previous paragraph are the responsibility of other microservices. In this case, there are, in this architecture, to perform this type of activity, one for each database target of transformation. For example, for relational databases, the microservice is described by the name *Object Transformer (ATL) for Relational DB*, in the case of columnar non-relational databases we call it *Object Transformer (ATL) for Relational DB* . As much as the transformation microservices of the modeling service, here the transformation microservices need to know the transformation rules and, therefore, the metamodels of the languages involved in the transformations.

The verification of authorizations related to the operations of creating, reading, updating, or removing objects from the underlying database is done concurrently with the transformation process.

V. Experiments and Results

Fig. 5 reveals instances of microservices related to modeling (MODELER) and persistence services — our platform uses the framework *JEE/Spring Cloud*, which has the service *Eureka* that, together with others, implements the framework's microservices orchestration. The INTERPRETER-P instance is equivalent to the specified *Object Persistence* component in Fig. 4 and the SQL-POOLCONN and NOSQL-POOLCONN instances, respectively, to *Object Transform for Relational DB* and a *Object Transform for Non Relational DB*.

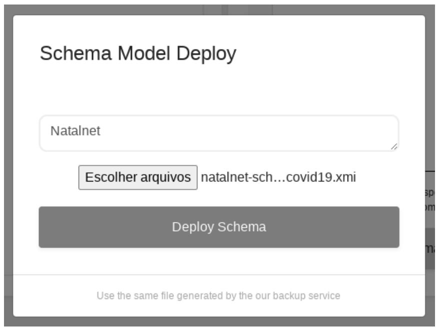


Instances currently registered with

Application	AMIs	Availabili
GATEWAY-MS	n/a (1)	(1)
ID-I-AUTH-V0F1-MS	n/a (1)	(1)
ID-I-MNGR-V0F1-MS	n/a (1)	(1)
IGRID-LOG-V0F1-MS	n/a (1)	(1)
INTERPRETER-P-V0F1-MS ●	n/a (1)	(1)
MODELER-V0F1-MS ●	n/a (1)	(1)
NOSQL-POOLCONN-MNGR-V0F1-MS ●	n/a (1)	(1)
SQL-POOLCONN-MNGR-V0F1-MS ●	n/a (1)	(1)

Fig. 5. List of active microservices reported by the Eureka/Spring Cloud service.

Regarding the process of using the platform, the first interaction involves user registration — used as a key to help address the schemas and achieve logical em data isolations. Fig. 6 would be the next step, the deployment of the data schema model — performed by importing the XMI with the specification. The immediate result of this deployment is in Fig. 7: the database — data repository for the user.


Schema Model Deploy

Natalnet

Escolher arquivos natalnet-sch...covid19.xmi

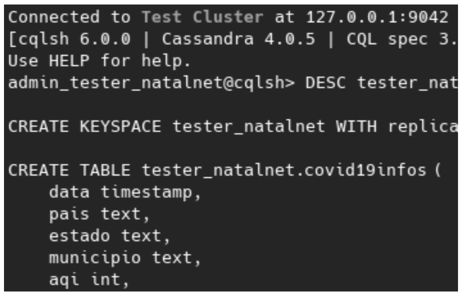
Deploy Schema

Use the same file generated by the our backup service

Fig. 6. Data schema import and deployment screen.

Once deployed, the platform user will populate the repository (database) as described in Fig. 8. The user will be able to select files in CSV format, for example, and describe some rules for this operation. As an example, for the file *gov_ms_brazil_covid19*, the platform must take all the columns from the CSV and the name of its columns must match the names of the columns in the table. In the case of the second file, with *exclude* mode flagged, the platform must use only the columns from this file that are placed under the *features* node, renaming the CSV columns to match the columns in the table.

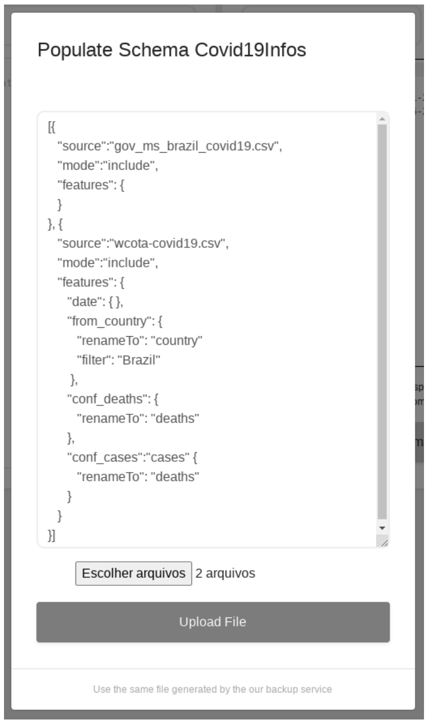
Moreover, the user can specify automatic updates for entire tables, or even just one or some of its features, see Fig. 9. This capability is relevant given the dynamic nature of the data consumed by this project: Time Series.


```
Connected to Test Cluster at 127.0.0.1:9042
[cqlsh 6.0.0 | Cassandra 4.0.5 | CQL spec 3.
Use HELP for help.
admin_tester_natalnet@cqlsh> DESC tester_nat

CREATE KEYSPACE tester_natalnet WITH replica

CREATE TABLE tester_natalnet.covid19infos (
    data timestamp,
    pais text,
    estado text,
    municipio text,
    aqi int,
```

Fig. 7. Convid19Infos schema database console prompt.


Populate Schema Covid19Infos

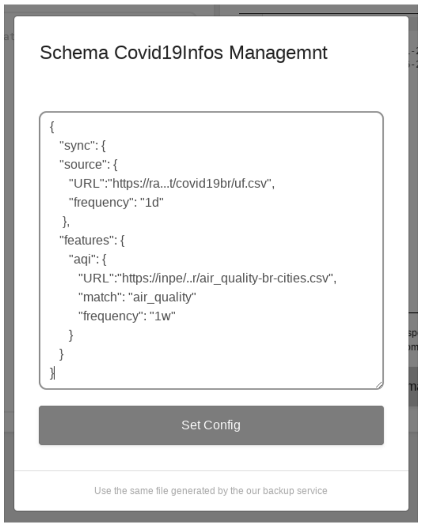
```
[{
  "source":"gov_ms_brazil_covid19.csv",
  "mode":"include",
  "features": {
  }
}, {
  "source":"wcota-covid19.csv",
  "mode":"include",
  "features": {
    "date": { },
    "from_country": {
      "renameTo": "country"
      "filter": "Brazil"
    },
    "conf_deaths": {
      "renameTo": "deaths"
    },
    "conf_cases":"cases" {
      "renameTo": "deaths"
    }
  }
}]
```

Escolher arquivos 2 arquivos

Upload File

Use the same file generated by the our backup service

Fig. 8. Covid19Info repository population operation configuration screen.


Schema Covid19Infos Managemnt

```
{
  "sync": {
  "source": {
    "URL":"https://ra...t/covid19br/uf.csv",
    "frequency": "1d"
  },
  "features": {
    "aqi": {
      "URL":"https://inpe/..r/air_quality-br-cities.csv",
      "match": "air_quality"
      "frequency": "1w"
    }
  }
}|
```

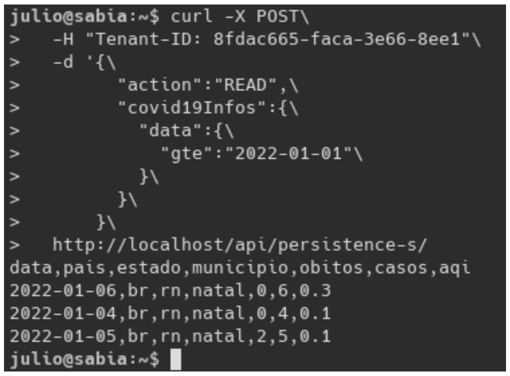
Set Config

Use the same file generated by the our backup service

Fig. 9. Covid19Info repository auto-update configuration screen.

Fig. 10, informs how those who know the endpoint */api/persistence-s* can retrieve/consult the data persisted in the repositories created by this platform.

Finally, the usage experiments involving 3 data scientists confirmed the alignment between the platform's functionalities and the interest related to the form of use desired by such professionals. All the complexity of creating and managing data repositories performed by the database was hidden. The platform allows its users and tenants to build, deploy, and operate different data and data schemas, which are best suited to the uses they intend in the context of this project. And the accessibility and availability to consume data from the repository are increased due to the realization of the "as a service" distribution model. On the other hand, any of the operations on the data schemas or their data were performed without the platform having its code changed (nor need to establish a relationship of dependence on specific languages, such as adaptive languages), or instances of new microservices added, concerning or not the specifics of the users, confirming the accomplishment of the best service offering format for such platform types, *multi-tenant single-instance*.



```
julio@sabia:~$ curl -X POST\
>   -H "Tenant-ID: 8fdac665-faca-3e66-8ee1"\
>   -d '{\
>         "action":"READ",\
>         "covid19Infos":{\
>           "data":{\
>             "gte":"2022-01-01"\
>           }\
>         }\
>       }\
>   http://localhost/api/persistence-s/
data,pais,estado,municipio,obitos,casos,aqi
2022-01-06,br,rn,natal,0,6,0.3
2022-01-04,br,rn,natal,0,4,0.1
2022-01-05,br,rn,natal,2,5,0.1
julio@sabia:~$
```

Fig. 10. Command for accessing data from repositories held by the platform.

VI. Conclusion

The platform as it was built gave us the opportunity to experiment with a software development approach that, although very close to MDE and Adaptive Languages approaches, presents itself as an alternative as, from a dispensation of code generators, on the other, the dependence on specific technologies. Such an approach authorizes the construction of "as a service" platforms, like ours, according to the most desired software service delivery format, *multi-tenant single-instance*. Its application resulted in a data management service and data schemas for the context of Covid19 pandemic applications, capable of facilitating, better organizing the aggregation and distribution of data — a relevant task for data scientists in the ML process.

Our effort from now on is to evolve the platform, adding new services that follow the same development approach adopted in this work. However, we must do this, from now on, in a perspective that allows us to investigate the efficiency of this solution, compared to solutions that are applied to the same context. Such comparisons must be guided by metrics that, for example, reveal how much more or less memory and processing that platform consumes from the host where it is deployed — given that it needs to load, outside the code, the schemas of Dice.

References

[1] Mohammadi, Mehdi, and Ala Al-Fuqaha. "Enabling cognitive smart cities using big data and machine learning: Approaches and challenges." IEEE Communications Magazine 56.2 (2018): 94-101.

[2] Roh, Yuji, Geon Heo, and Steven Euijong Whang. "A survey on data collection for machine learning: a big data-ai integration perspective." IEEE Transactions on Knowledge and Data Engineering 33.4 (2019): 1328-1347.

[3] Miao, Hui, et al. "Modelhub: Deep learning lifecycle management." 2017 IEEE 33rd International Conference on Data Engineering (ICDE). IEEE, 2017.

[4] Dudjak, Mario, and Goran Martinović. "An API-first methodology for designing a microservice-based Backend as a Service platform." Information Technology and Control 49.2 (2020): 206-223.

[5] Neto, Josino Rodrigues, et al. "Software as a Service: Desenvolvendo Aplicações Multi-tenancy com Alto Grau de Reuso." Sociedade Brasileira de Computação (2012).

[6] Kalra, Sumit and Prabhakar, T. V. "Towards Dynamic Tenant Management for Microservice based Multi-Tenant SaaS Applications". In Proceedings of the 11th Innovations in Software Engineering Conference (ISEC '18). Association for Computing Machinery, New York, NY, USA, 2018, Article 12, 1–5. https://doi.org/10.1145/3172871.3172882.

[7] Bezemer, C., and Andy Zaidman. "Challenges of reengineering into multi-tenant SaaS applications." Technical Report Series TUD-SERG-2010-012 (2010).

[8] Bucchiarone, Antonio, et al. "Grand challenges in model-driven engineering: an analysis of the state of the research." Software and Systems Modeling 19.1 (2020): 5-13.

[9] Kiczales, Gregor, et al. "Metaobject protocols: Why we want them and what else they can do." Object-Oriented Programming: The CLOS Perspective (1993): 101-118.

[10] Thönes, Johannes. "Microservices." IEEE Software 32.1 (2015): 116-116.

[11] Nguyen, Phu H., et al. "Using microservices for non-intrusive customization of multi-tenant SaaS." Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2019.

[12] Song, Hui, Phu H. Nguyen, and Franck Chauvel. "Using microservices to customize multi-tenant saas: From intrusive to non-intrusive." Joint Post-proceedings of the First and Second International Conference on Microservices (Microservices 2017/2019). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[13] Kiczales, Gregor, Jim des Rivieres, and Daniel G. Bobrow. "A Review of The Art of the Metaobject Protocol." (2010).

[14] Mane, Babacar, et al. "A Domain Specific Language to Provide Middleware for Interoperability among SaaS and DaaS/DBaaS through a Metamodel Approach." ICEIS (1). 2021.

[15] Moradi, Hossein, Bahman Zamani, and Kamran Zamanifar. "Caasset: A framework for model-driven development of context as a service." Future Generation Computer Systems 105 (2020): 61-95.

[16] Mazeiar Salehie and Ladan Tahvildari. 2009. Self-adaptive software: Landscape and research challenges. ACM Trans. Auton. Adapt. Syst. 4, 2, Article 14 (May 2009), 42 pages. https://doi.org/10.1145/1516533.1516538

[17] Díaz, Oscar & Iturrioz, Jon & Piattini, Mario. (1998). Promoting business policies in object-oriented methods. Journal of Systems and Software. 41. 105-115. 10.1016/S0164-1212(97)10011-5.

[18] Skogan, David. "UML as a schema language for XML based data interchange." Proceedings of the 2nd International Conference on The Unified Modeling Language (UML'99). 1999.

[19] da Costa, Júlio G. S. F., Reinaldo A. Petta & Samuel Xavier de Souza (2021), Metadata Interpretation Driven Development.