An open source data science platform to build production-grade machine learning projects

Romain Avouac, Frédéric Comte and Thomas Faria

Abstract In recent years, statisticians trained in data science methods have been joining National Statistical Offices (NSOs), aiming to harness non-traditional data sources and machine learning methods to enhance the production of official statistics. Despite their expertise, these professionals encounter significant barriers, including limited computational resources, inflexible development environments that don't foster collaborative work, and limited tools to transition from innovative experiments to production-ready solutions. This paper presents Onyxia, an open-source project developed to address these challenges by enabling organizations to build modern and flexible data science environments that enhance the autonomy of statisticians. With Onyxia and its showcase instance, the SSP Cloud, we demonstrate how cloud technologies can be made readily accessible, fostering innovation, collaboration, and reproducibility within the realm of official statistics. Through a case study of the classification of the activity domain of French companies, we illustrate how these tools have been instrumental in operationalizing machine learning models in accordance with MLOps principles, marking a significant step forward in the valorisation of data science projects at Insee.

Key words: official statistics, big data, cloud technologies, data science, machine learning, MLOps, reproducibility, open source

Romain Avouac
Insee, 88 avenue François Verdier, Montrouge, e-mail: `romain.avouac@insee.fr`

Frédéric Comte
Insee, 88 avenue François Verdier, Montrouge, e-mail: `frederic.comte@insee.fr`

Thomas Faria
Insee, 88 avenue François Verdier, Montrouge, e-mail: `thomas.faria@insee.fr`

1 Introduction

In recent years, the European Statistical System (ESS) has committed to leverage non-traditional data sources in order to improve the process of statistical production, an evolution that is encapsulated by the concept of Trusted Smart Statistics [34]. This dynamic is accompanied by innovations in the statistical processes, so as to be able to take advantage of the great potential of these new sources (greater timeliness, increased spatio-temporal resolution, etc.), but also to cope with their complexity or imperfections. At the forefront of these innovations are machine-learning methods and their promising uses in the coding and classification fields, data editing and imputation [17]. The multiple challenges faced by statistical institutes because of this evolution are addressed in the Bucharest Memorandum on Official Statistics in a Datafied Society (Trusted Smart Statistics), which predicts that "the variety of new data sources, computational paradigms and tools will require amendments to the statistical business architecture, processes, production models, IT infrastructures, methodological and quality frameworks, and the corresponding governance structures", and consequently invites the ESS to assess the required adaptations and prioritize them [10].

In line with these recommendations, much work has been done in the context of successive projects at the European level in order to operationalize the use of non-traditional data sources in the production of official statistics. Within the scope of the ESSnet Big Data II project (2018-2020), National Statistical Offices (NSOs) have been working across a wide range of themes (online job vacancies, smart energy, tracking ships, etc.) in order to put together the building blocks for using these sources in actual production processes and identify their limitations [13]. However, while a substantial amount of work has been devoted to developing methodological frameworks [9, 36], quality guidelines [20] as well as devising business architectures that make third-party data acquisition more secure [33], not much has been said about the IT infrastructures and skills needed to properly deal with these new objects.

The characteristics of big data sources make them particularly complex to process, be it their volume, their velocity (speed of creation or renewal) or their variety (structured but also unstructured data, such as text and images). Besides, the "skills and competencies to automate, analyse, and optimize such complex systems are often not part of the traditional skill set of most National Statistical Offices" [3]. Not incidentally, an increasing number of public statisticians trained as data scientists have joined NSOs in recent years. Within its multiple meanings, the term “data scientist” reflects the increased involvement of statisticians in the IT development and orchestration of their data processing operations, beyond merely the design or validation phases [7]. However, based on our observations at Insee and other French statistical offices, the ability of these new data professionals to derive value from big data sources and machine learning methods is limited by several challenges.

A first challenge is related to the lack of proper IT infrastructures to tackle the new data sources that NSOs now have access to as well as the accompanying need for new statistical methods. For instance, big data sources require huge storage capacities and often rely on distributed computing frameworks to be processed

[25]. Similarly, the adoption of new statistical methods based on machine learning algorithms often require IT capacities — in particular, GPUs (graphical processing units) — to massively parallelize computations [35]. Such resources are not readily available in traditional IT infrastructures. Furthermore, these new infrastructures generally require specific skills — especially to build and maintain them — that are not easily found in NSOs.

Another major challenge lies in equipping statisticians with development environments that enable them to experiment more freely. The essence of innovation in statistical work lies in the ability to swiftly adapt to and incorporate new tools and methodologies. This agility is hampered when statisticians depend excessively on IT departments to provision resources or install new software packages. In traditional setups — personal computers or virtual desktops on centralized architectures — IT departments generally prioritize security and system stability over the provision of new services, which limits the innovation potential. Besides, these rigid environments make it harder to implement development best practices, such as collaborative work — which requires environments where experiments can be easily shared with peers — and reproducibility.

A third challenge is related to the difficulty of transitioning from innovative experiments to production-ready solutions. Even when statisticians have access to development environments in which they can readily experiment, the step towards deploying an application or a model is generally very large. Production environments often differ from development environments in such a way that the additional development costs are needed to go from a proof of concept to an industrialized solution that actually serves users can limit the feasibility of this transition. Furthermore, in the case of machine learning projects, models that have been deployed require a proper monitoring to ensure that they maintain their accuracy and utility over time, and generally require periodic or continuous improvements. Again, this pleads for more flexible environments that enable statisticians to manage the complete lifecycle of their data science projects in a more continuous way.

We argue that these various challenges have an underlying common theme: the need for more autonomy. The ability of data science methods to improve and potentially transform the production of official statistics crucially depends on the ability of statisticians to carry out innovative experiments more freely. To do so, they need to have access to substantial and diverse computing resources that enable them to tackle the volume and diversity of big data sources and leverage machine learning methods to better deal with these data. Such experimental projects require, in turn, flexible development environments that foster collaborative work in order to capitalize the diversity of profiles and skills that compose project teams. Finally, to derive value from these experiments, statisticians require tools to deploy applications as proof-of-concepts and orchestrate their statistical operations autonomously.

Against this background, we developed Onyxia: an open source project that enables organizations to deploy data science platforms that foster innovation by giving statisticians more autonomy[1]. This paper aims at describing the full thought

[1] https://github.com/InseeFrLab/onyxia

process that led to this project and at exemplifying how it empowers statisticians at Insee, thus becoming a cornerstone of our innovation strategy. Section 2 provides an in-depth analysis of the data ecosystem's latest developments, casting light on the technological choices that have shaped the development of a modern data science environment tailored to the specific needs of statisticians. In particular, we show how cloud-native technologies — particularly containers and object storage — are key to building scalable and flexible environments that can enhance autonomy while promoting reproducibility in the production of official statistics. However, despite their appealing attributes for modern data science applications, the complexity of configuring and utilizing cloud technologies often poses barriers to their broad adoption. In section 3, we detail the core of the Onyxia project: how we made cloud technologies accessible to statisticians through a user-friendly interface and an extensive catalogue of ready-to-use data science environments, while circumventing potential vendor lock-in effects for both the institution and their users. We also show how providing an open-innovation instance of Onyxia, the SSP Cloud, greatly facilitated the adoption of these technologies and fostered improved development practices. Finally, through the case study of the classification of French companies' activity, section 4 illustrates how leveraging these technologies greatly facilitated the deployment of machine learning models at Insee in alignment with the industry best practices — namely, MLOps principles.

2 Principles for building a modern and flexible data architecture for official statistics

With the emergence of big data sources and new methodologies offering significant promise to improve the production process of official statistics, statisticians trained in data science techniques are eager to innovate. However, their ability to do so is limited by several challenges. Central among these challenges is the need for greater autonomy — be it in scaling resources to match statistical workloads, deploying proofs of concept with agility and in a collaborative manner, etc. Against this background, our aim was to design a data science platform that not only manages big data efficiently but also empowers statisticians by enhancing their autonomy. To achieve this, we delved into the evolving data ecosystem in order to identify significant trends with the potential to overcome the aforementioned limitations[2]. Our findings indicate that leveraging cloud-native technologies, particularly containers and object storage, is key to building infrastructures capable of handling large and varied

[2] As a preamble to this review, we should note that, although we did our best to ground our insights in the academic literature, a lot of it stems from informal knowledge gathered through diligent and ongoing technology watch. In the rapidly evolving data ecosystem, traditional research papers are increasingly giving way to blog posts as the primary references for cutting-edge developments. This shift is largely due to the swift pace at which big data technologies and methodologies are advancing, making the lengthy publication process of formal research often not the preferred way of disseminating timely insights and innovations.

datasets in a flexible and cost-effective manner. Furthermore, these technologies significantly enhance autonomy, facilitating innovation and promoting reproducibility in the production of official statistics.

2.1 Limitations of traditional big data architectures

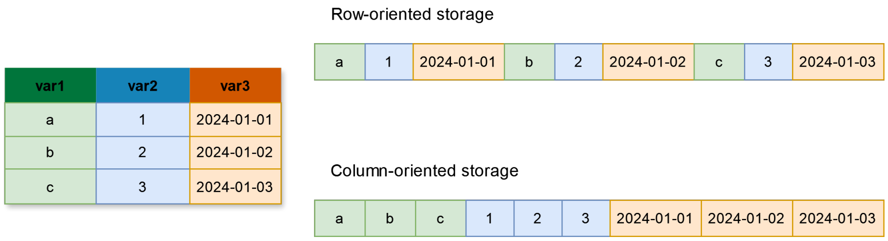
Over the last decade, the landscape of big data has dramatically transformed. Following the publication of Google's seminal papers that introduced the MapReduce paradigm [16, 8], Hadoop-based systems rapidly became the reference architecture of the big data ecosystem, celebrated for their capability to manage extensive datasets through the use of distributed computing. The inception of Hadoop marked a revolutionary step, enabling organizations to process and analyze data at an unprecedented scale. Basically, Hadoop provided companies with all-rounded capabilities for big data analytics: tools for ingestion, data storage (HDFS), and computing capacities (Spark, among others) [11], thus explaining its rapid adoption across industries.

In the late 2010s, Hadoop-based architectures have experienced a clear decline in popularity. In traditional Hadoop environments, storage and compute were co-localized by design: if the source data is distributed across multiple servers (horizontal scaling), each section of the data is directly processed on the machine hosting that section, so as to avoid network transitions between servers. In this paradigm, scaling the architecture often meant a linear increase in both compute and storage, regardless of the actual demand. In a recent article provocatively titled "Big Data is Dead"[3], Jordan Tigani, one of the founding engineers behind Google BigQuery, explains why this model doesn't fit the reality of most data-centric organizations anymore. First, because "in practice data sizes increase much faster than compute sizes". While the amount of data generated and thus needing to be stored may grow linearly over time, it is generally the case that we only need to query the most recent portions of it, or only some columns and/or groups of rows. Besides, Tigani points out that "the big data frontier keeps receding": advancements in server computing capabilities and declining hardware costs mean that the number of workloads that don't fit on a single machine — a simple yet effective definition of big data — has been continually decreasing. As a result, by properly separating storage and compute functions, even substantial data processing jobs may end up using "far less compute than anticipated [...] and might not even need to use distributed processing at all".

These insights strongly align with our own observations at Insee in recent years. For instance, an Insee team set up a Hadoop cluster as an alternative architecture to the one already in use to process sales receipt data in the context of computing the consumer price index. An acceleration of data processing operations by up to a factor of 10 was achieved, for operations that previously took several hours to perform [21]. But despite this increase in performance, this type of architectures were not reused later for other projects, mainly because the architecture proved to

[3] `https://motherduck.com/blog/big-data-is-dead/`

be expensive and complex to maintain, necessitating specialized technical expertise rarely found within NSOs [39]. Interestingly, subsequent projects involving large datasets didn't suffer much from this change, as their needs were actually very much in line with Tigani's observations: the performance bottleneck for these projects was generally on the side of computational needs rather than storage capacity, making Hadoop-style clusters less relevant. Furthermore, although these projects could still involve substantial data volumes, we observed that effective processing could be achieved using conventional software tools (R, Python) on single-node systems by leveraging recent important innovations from the data ecosystem. First, by using efficient formats to store the data such as Apache Parquet [14], which properties — columnar storage [1] (see figure1), optimisation for "write once, read many" analytics, ability to partition data, etc. — make it particularly suited to analytical tasks such as those generally performed in official statistics [2]. Second, by performing computations using optimised in-memory computation frameworks such as Apache Arrow [15] or DuckDB [32]. Also based on columnar representation — thus working in synergy with Parquet files — both of these frameworks greatly improve data queries performance through the use of "lazy evaluation": instead of doing lots of separate operations (e.g. selecting columns and/or filtering rows, then computing new columns, then performing agregations, etc.), they process them all at once in a more optimised way. As a result, computations are limited to the data effectively needed by the queries, enabling much larger-than-memory data processing on usual single-node machines.



var1	var2	var3
a	1	2024-01-01
b	2	2024-01-02
c	3	2024-01-03

Row-oriented storage

a	1	2024-01-01	b	2	2024-01-02	c	3	2024-01-03

Column-oriented storage

a	b	c	1	2	3	2024-01-01	2024-01-02	2024-01-03

Fig. 1 Row-oriented and column-oriented representation of a same dataset.

Note: Many statistical operations are analytical (OLAP) in nature: they involve selecting specific columns, computing new variables, performing group-based aggregations, etc. Row-oriented storage is not well-suited to analytical operations as it requires the full dataset to be read in memory to query it. Conversely, column-based storage allows only relevant data columns to be queried, significantly reducing read and processing times for analytical workloads. In practice, popular columnar formats such as Parquet use a hybrid-representation: they are primarily column-oriented but also implement clever row-based grouping to optimize filtering queries.

2.2 Embracing cloud-native technologies

In light of this evolution of the big data ecosystem, there has been a notable shift in recent years within the industry towards more flexible and loosely coupled architectures. The advent of cloud technologies has been instrumental in facilitating this shift. Unlike the era where Hadoop was prominent, network latency has become much less of a concern, making the traditional model of on-premise and co-located storage and compute solutions less relevant. Regarding the nature of the data that needs to be processed, we observe an evolution that some have described as moving "from big data to flexible data". Modern data infrastructures are required not only to process large volumes but also to be adaptable in multiple dimensions. They must accommodate various data structures (ranging from structured, tabular formats to unstructured formats like text and images), ensure data portability across multi-cloud and hybrid cloud environments, and support a diverse range of computational workloads (from parallel computations to deep learning models necessitating GPUs, as well as the deployment and management of applications) [24]. In recent years, two technologies have emerged in the data ecosystem as foundational technologies for achieving such flexibility in cloud-based environments: containerization and object storage.

In a cloud environment, the computer of the user becomes a simple access point to perform computations on a central infrastructure. This enables both ubiquitous access to and scalability of the services, as it is easier to scale a central infrastructure — usually horizontally, i.e. by adding more servers. However, such centralized infrastructures have two well-identified limitations that need to be dealt with: the competition between users in access to physical resources and the need to properly isolate deployed applications. The choice of containerization is fundamental as it tackles these two issues [4]. By creating “bubbles” specific to each service, containers guarantee application isolation while remaining lightweight, as they share the support operating system with the host machine (see. graph 2). In order to manage multiple containerized applications in a systematic way, containerized infrastructures generally rely on an orchestrator software — the most prominent one being Kubernetes, an open-source project initially developed by Google to manage its numerous containerized workloads in production [40]. Orchestrators automate the process of deploying, scaling, and managing containerized applications, coordinating their execution across various servers. Interestingly, this property makes it possible to handle very large volumes of data in a distributed way: containers break down big data processing operations into a multitude of small tasks, organized by the orchestrator. This minimizes the required resources while providing more flexibility than Hadoop-based architectures [41].

The other fundamental choice in a data architecture is the nature of data storage. In the cloud ecosystem, so-called "object storage" has become the *de facto* reference [37] [4]. In this paradigm, files are stored as "objects" consisting of data, an identifier and metadata. This type of storage is optimized for scalability, as objects are not limited

[4] Mainly because of Amazon’s "S3" (Simple Storage Service) implementation.

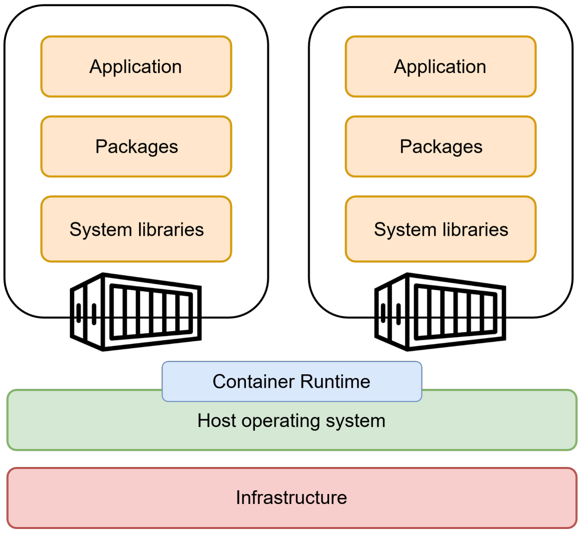


Fig. 2 Architecture of a containerized environment.

Note: A container is a logical grouping of resources that makes it possible to encapsulate an application (e.g. Python code), the packages used (e.g. Pandas, NumPy) and system libraries (the Python interpreter, other OS-dependent libraries, etc.), in a single package. Containerized applications are isolated from one another through virtualization, which makes it possible to attribute specific physical resources to each application while guaranteeing complete independence between them. But contrary to virtual machines which also virtualize the operating system (OS), containers rely on a lightweight form of virtualization: the container shares the OS of the host infrastructure through the container runtime (e.g. Docker). As a result, containers are much more portable and can be readily deployed and redistributed.

in size and the underlying technology enables cost-effective storage of (potentially very) large files. It is also instrumental in building a decoupled infrastructure such as discussed before: the data repositories — referred to as "buckets" — are directly searchable using standard HTTP requests through a standardized REST API. In a world where network latency is not the main bottleneck anymore, this means that storage and compute don't have to be on the same machines or even in the same location, and can thus scale independently according to specific organization demands. Finally, object storage is a natural complement to architectures based on containerized environments for which it provides a persistence layer — containers being stateless by design — and easy connectivity without compromising security, or even with strengthened security compared with a traditional storage system [28].

2.3 Leveraging cloud technologies to increase autonomy and foster reproducibility

Understanding how the technological choices described in the technical discussion above are relevant in the context of official statistics require an in-depth review of statisticians' professional practices in their use of computing environments. At the end of the 2000s, with micro-computing at its peak, many of the technical resources used by statisticians at Insee were local: the code and processing software were located on individual computers, while data was accessed through a file-sharing system. Because of the the limited scalability of personal computers, this setup greatly limited the ability of statisticians to experiment with big data sources or computationally intensive statistical methods, and involved security risks because of the widespread data dissemination within the organization. In order to overcome these limitations, a transition was made towards centralised IT infrastructures, concentrating all — and thus overall much more — resources on central servers. Such infrastructures, made available to statisticians through a shared, virtual desktop environment for ease of use, remains the dominant method for conducting statistical computations at Insee at the time of writing this lines.

Through our observations and discussions with fellow statisticians, it became obvious that although the current IT infrastructure adequately supported the core activities of statistical production, it noticeably restricted statisticians' capacity to experiment freely and innovate. The primary bottleneck in this organization is the dependency of statistical projects on centralized IT decision-making, such as the allocation of computing resources, access to shared data storage, the use of pre-configured programming languages and packaging environments, etc. Besides, such dependencies often lead to a well-known phenomenon within the software development community that lies at the heart of the DevOps approach, where the priorities of developers — iterate rapidly to improve functionality in a continuous manner — often clash with IT's focus on security and process stability. On the contrary, it is our understanding that modern data science practices reflect an increased involvement of statisticians in the IT development and orchestration of their data processing operations, beyond merely the design or validation phases. New data science infrastructures must take this expanded role of their users into account, giving them more autonomy than conventional infrastructures.

We argue that cloud technologies stand out as a powerful solution to give statisticians much more autonomy in their daily work, enabling a culture of innovation. Through object storage, users gain control over the storage layer, allowing them to experiment with diverse datasets without being constrained by the limited storage spaces typically allocated by IT departments. Containerization empowers users to customize their working environments to their specific needs — be it programming languages, system libraries, or package versions — while also providing the flexibility to scale their applications according to the required computing power and storage capacities. By design, containers also foster the development of portable applications, which enables smoother transitions between environments (development, qualification, production), ensuring that applications can be moved seamlessly without the hurdles of environmental inconsistencies. Finally, with orchestration tools like Kubernetes, statisticians can more readily deploy applications and APIs and automatize the whole building process. This capability aligns with the DevOps approach, enabling quicker iteration and building minimal prototypes as proofs of concept (POCs) rather than building the optimal (but time-consuming) solution for a pre-defined objective [22].

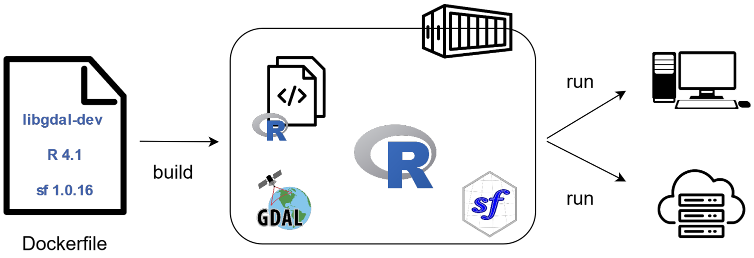


Fig. 3 Containers foster reproducibility and portability by design.

Note: In a containerised environment, applications are created through script specifications — a paradigm known as "infrastructure as code". In a text file conventionally named "Dockerfile", data scientists can specify the working environment of their application: the application code, the software to be included (e.g. R), the packages used for their processing operations (e.g. the R package for geospatial computation *sf*), and the OS-dependent system libraries that are called by these packages (e.g. GDAL, the translator library for raster geospatial data formats used by most packages dealing with geospatial data). Importantly, the versions of the software and packages that were used to develop the application can be precisely specified, which guarantees reproducibility of the computations performed. A build step then creates an image associated to the Dockerfile, i.e. a packaged and compressed form of the working environment of the application. Images created this way are portable: they can be readily distributed — usually through a container registry — and executed in a reproducible manner on any infrastructure that has a container runtime.

Besides scalability and autonomy, these architectural choices also foster reproducibility of statistical computations. The concept of reproducibility — namely the ability to reproduce the result of an experiment by applying the same methodology to the same data — is a fundamental criterion of scientific validity [27]. It is also highly relevant in official statistics, as it serves as a foundation for transparency, which in turn is crucial for building and maintaining the public's trust [12]. Fostering reproducibility in statistical production involves devising processing solutions that can produce reproducible statistics on the one hand, and that can be shared with peers on the other hand [26]. Traditional IT infrastructures — either a personal computer or a shared infrastructure with remote desktop access — fall short in this regard, as building a project or just computing a statistical indicator there generally involves a series of manual steps (installing system libraries, the programming language binary, projects packages, dealing with potentially conflicting versions, etc.) that can not be fully reproduced across projects. In comparison, containers are reproducible by design, as their build process involves defining precisely all the needed resources as a set of processing operations in a standardized manner, from the "bare machine" to the running application [29]. Furthermore, these reproducible environments can be easily shared to peers as they can be readily published on open registries (for example, a container registry such as DockerHub) along with the source code of the application (for example, on a public software forge like GitHub or GitLab). This approach significantly enhances the re-usability of code projects, fostering a community-driven model of development and innovation.

3 Onyxia: an open source project to build cloud-native data science platforms

The Onyxia project, initiated by the French Public Service and available at onyxia.sh, is an open source project aimed at creating self-sufficient data science environments in the cloud or on-premises. This project can be seen as a “Platform as a Package” (PaaP) solution for organisations wishing to create a data science environment based on cloud technologies.

3.1 Making cloud-technologies accessible to statisticians

Our technology watch and literature review highlighted cloud-native technologies, in particular containerization and object storage, as instrumental in building a data science platform that is both scalable and flexible. Building on these insights, we established our initial on-premise Kubernetes cluster in 2020, integrating it with MinIO, an open-source object storage system designed to work seamlessly with Kubernetes. Yet, our first experiments highlighted a significant barrier to the widespread adoption of cloud-native technologies: the complexity of their integration. This is an important consideration when building data architectures that prioritize modularity — an essential feature for the flexibility we aim to achieve. For instance, due to MinIO’s compatibility with the Amazon S3 API, the storage source could easily be switched to one managed by another public cloud provider, without requiring substantial modifications. However, modularity of the architecture components also entails that any data application launched on the cluster must be configured so as to communicate with all the components. For instance, in a big data setup, configuring Spark to operate on Kubernetes while interacting with datasets stored in MinIO requires an intricate set of configurations (specifying endpoints, access tokens, etc.), a skill set that typically lies beyond the expertise of statisticians.

This insight is really the base of the Onyxia project : choosing technologies that foster autonomy won’t actually foster autonomy if their complexity acts as a deterrent from widespread adoption in the structure. In recent years, statisticians at Insee already needed to adapt to a changing environment in terms of their everyday tools : transitioning from proprietary software (SAS®) to open-source ones (R, Python), acculturating to technologies that improve reproducibility (version control with Git), consuming and developing APIs, etc. These changes, making their job more and more akin to the one of software developers, already imply significant training and changes in daily work practices. Against this background, adoption of cloud-technologies was utterly dependent on making them readily accessible.

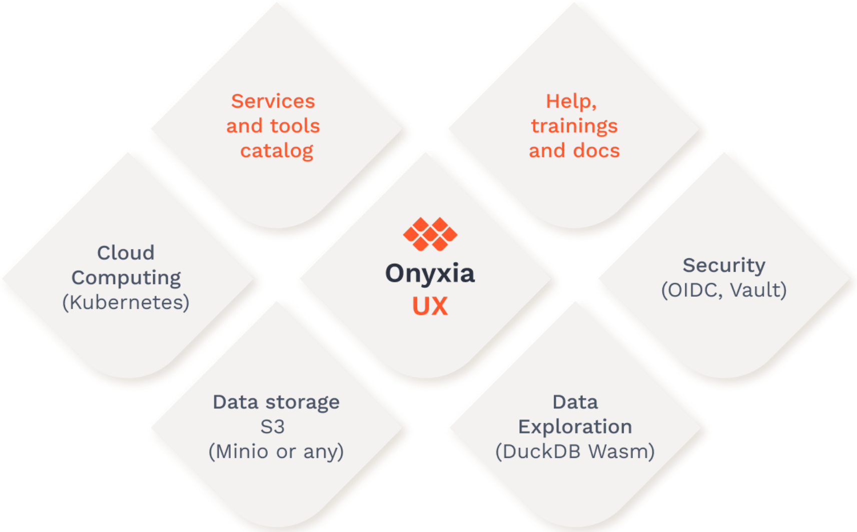

Services and tools catalog
Help, trainings and docs
Cloud Computing (Kubernetes)
Onyxia
UX
Security (OIDC, Vault)
Data storage S3 (Minio or any)
Data Exploration (DuckDB Wasm)

Fig. 4 Onyxia is the technical binder between cloud-native modular components

To bridge this gap, we developed Onyxia, an application that essentially acts as interface between the modular components that compose the architecture (see fig 4). The main entrypoint of the user is a user-friendly web application[5] that enables users to launch services from a data science catalog (see section 3.3) as running containers on the underlying Kubernetes cluster. The interface between the UI and Kubernetes is done by a lightweight custom API[6], that essentially transforms the application request of the user into a set of manifests to deploy Kubernetes resources. For a given application, these resources are packaged under the form of Helm charts, a popular way of packaging potentially complex applications on Kubernetes [18]. Although users can configure a service to tailor it to their needs, they will most of the time just launch an out-of-the-box service with default settings and start developing straightaway. This point really illustrates the added value of Onyxia in facilitating the adoption of cloud technologies. By injecting authentication information and configuration into the containers at the initialization, we ensure that users can launch and manage data science services in which they can interact seamlessly with the

[5] `https://github.com/InseeFrLab/onyxia-ui`

[6] `https://github.com/InseeFrLab/onyxia-api`

data from their bucket on MinIO, their sensitive information (tokens, passwords) stored in Vault, etc. This automatic injection, coupled with the pre-configuration of data science environments in Onyxia's catalogs of images[7] and associated helm-charts[8], make it possible for users to execute potentially complex workloads - such as running distributed computations with Spark on Kubernetes using data stored in S3, or training deep-learning models using a GPU - without getting bogged down by the technicalities of configuration.

3.2 Architectural choices aimed at fostering autonomy

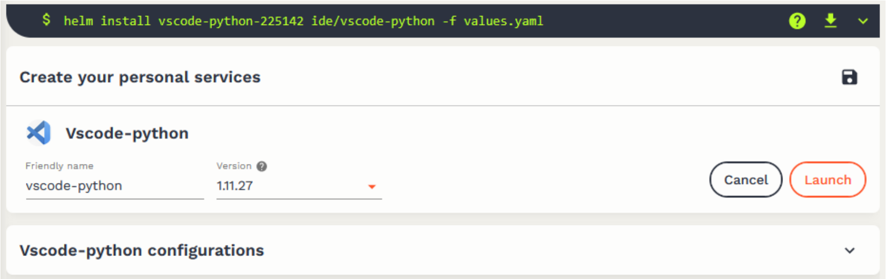
The Onyxia project is based on a few structuring principles, with a central theme: fostering autonomy, both at the organizational and individual levels. First, at the level of the organization by preventing vendor lock-in. In order to get a competitive edge, many commercial cloud providers develop applications and protocols that customers need to use to access cloud resources but that are not interoperable, greatly complexifying potential migrations to another cloud platform [31]. Recognizing these challenges, there is a trend towards endorsing cloud-neutral strategies [30] in order to reduce reliance on a single vendor's specific solutions. In contrast, the use of Onyxia is inherently not restrictive: when an organization chooses to use it, it chooses the underlying technologies - containerization and object storage - but not the solution. The platform can be deployed on any Kubernetes cluster, either on-premise or in public clouds. Similarly, Onyxia was designed to be used with MinIO because it is an open-source object-storage solution, but is also compatible with objects storage solutions from various cloud providers (AWS, GCP).

Onyxia also fosters autonomy at the level of users. Proprietary softwares that have been used intensively in official statistics - such as SAS or STATA - also produce a vendor lock-in phenomenon. The costs of licensing are high and can evolve quickly, and users are tied in certain ways of performing computations, preventing progressive upskilling. On the contrary, Onyxia aspires to be removable; we want to enhance users' familiarity and comfort with the underlying cloud technologies rather than act as a permanent fixture in their workflow. An illustrative example of this philosophy is the platform's approach to user actions: for tasks performed through the UI, such as launching a service or managing data, we provide users with the equivalent terminal commands, promoting a deeper understanding of what actually happens on the infrastructure when triggering something. Furthermore, all the services offered through Onyxia's catalog are open-source.

Naturally, the way Onyxia makes statisticians more autonomous in their work depends on their needs and familiarity with IT skills. Statisticians that just want to have access to extensive computational resources to experiment with new data sources or statistical methods will have access in a few clicks to easy-to-use, pre-

[7] `https://github.com/InseeFrLab/images-datascience`

[8] `https://github.com/InseeFrLab/helm-charts-interactive-services`



```
$ helm install vscode-python-225142 ide/vscode-python -f values.yaml
```

Create your personal services

Vscode-python

Friendly name: vscode-python

Version: 1.11.27

Vscode-python configurations

Fig. 5 Launching a service through Onyxia's UI.

Note: Services from Onyxia's catalog can either be used vanilla or configured by the users to tailor them to their specific needs. In order to limit the dependence of users on Onyxia, each action performed by the user on the UI is accompanied by the actual command that is executed on the Kubernetes cluster.

configured data science environments, so that they can directly start to work and prototype their solution. However, many users want to go deeper and build actual prototypes of production applications for their projects: configuring initialization scripts to tailor the environments to their needs, deploying an interactive app that delivers data visualisation to users of their choice, deploying other services than those available in our catalogs, etc. For these advanced users to continue to push the boundaries of innovation, Onyxia gives them access to the underlying Kubernetes cluster. This means that users can freely open a terminal on an interactive service and interacts with the cluster - within the boundaries of their namespace - in order to apply custom resources and deploy custom applications or services.

Besides autonomy and scalability, the architectural choices of Onyxia also foster reproducibility of statistical computations. In the paradigm of containers, the user must learn to deal with resources which are by nature ephemeral, since they only exist at the time of their actual mobilization. This fosters the adoption of development best practices, notably the separation of the code — put on an internal or open-source forge such as GitLab or GitHub — the data — stored on a specific storage solution, such as MinIO — and the computing environment. While this requires an entry cost for users, it also helps them to conceive their projects as pipelines, i.e. a series of sequential steps with well-defined inputs and outputs (akin to directed acyclic graph (DAG)). The projects developed in that manner are usually more reproducible and portable — they can work seamlessly on different computing environments — and thus also more readily shareable with peers.

3.3 An extensive catalogue of services to cover the entire lifecycle of data science projects

In developing the Onyxia platform, our intention was to provide statisticians with a comprehensive environment designed to support end-to-end development of data science projects. As depicted in Figure 6, the platform offers a vast array of services that span the complete lifecycle of a data science project.

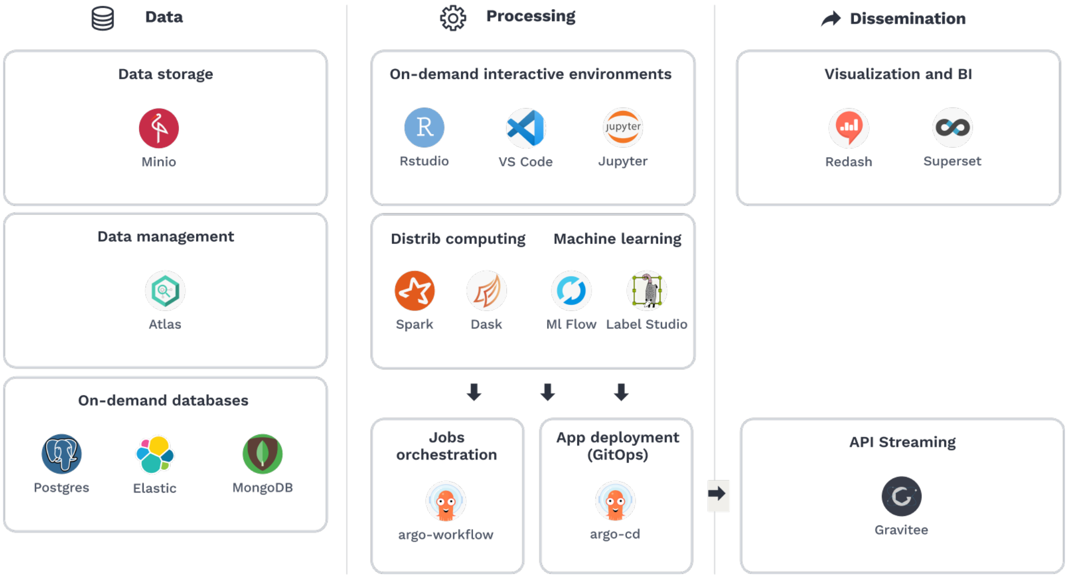


Fig. 6 Onyxia's catalog aims at covering the entire lifecycle of data science projects

The primary usage of the platform is the deployment of interactive development environments (IDE), such as RStudio, Jupyter, or VSCode. These IDEs come equipped with the latest kernels of major open-source programming languages commonly employed by public statisticians (R, Python, Julia), as well as an extensive collection of packages commonly used in data science for each language. In order to ensure that services remain up-to-date and consistent between them, we maintain our own stack of underlying Docker images and rebuilt it weekly. The stack of images is fully open-source[9] and can thus be reused outside of Onyxia.

As discussed in previous sections, the persistence layer of these interactive environments is mainly carried out by MinIO, Onyxia's default object storage solution. As it is based on a standardized REST API, files can be easily queried directly from R or Python using high-level packages. This in itself is an important step of ensuring reproducibility: the input files of a project are not mounted manually and

[9] https://github.com/InseeFrLab/images-datascience

then specified via paths adherent to a specific infrastructure and filesystem. Rather, files are specified as HTTP queries, making the overall structure of projects much more extendable. In our experience, the object-storage paradigm covers very well the needs of most statistical projects we accompany. However, additional database services such as PostgreSQL and MongoDB are available for applications with specific needs, such as those requiring online transaction processing (OLTP) capabilities or document-oriented storage.

As Onyxia was developed to allow experimentation with big data sources and machine learning methods, we also provide services optimized for scalability. For instance, frameworks like Spark and Trino that enable to perform distributed computations within Kubernetes. These services come pre-configured to integrate seamlessly with S3 storage, thus facilitating building integrated and efficient data pipelines.

Beyond mere experimentation, our goal is to empower statisticians to transition from trial phases to production-ready projects. In lines with principles from the DevOps approach, this involves facilitating the deployment of prototypes and their continuous improvement over time. To this end, we provide a set of open-source tools aimed at automatizing and industrializing the process of deploying data-intensive applications (ArgoCD, Argo-Workflows, MLFlow). For projects leveraging machine-learning models, statisticians can serve their models through APIs, deploy them using the aforementioned tools, and manage their lifecycle using an API manager such as Gravitee. Section 4 will delve into how these tools, particularly MLFlow, have been central in putting machine learning models in production at Insee, in accordance with MLOps principles.

In section 3.2, we stressed that one of Onyxia's fundamental design principle was to avoid vendor lock-in. In line with this idea, organizations that implement Onyxia are free to customize catalogs to suit their specific requirements, or even opt to construct their own catalogs independent of Onyxia's default offerings. This flexibility ensures that organizations are not confined to a single solution or provider, and can adapt the platform to their evolving needs.

3.4 Building commons : an open-source project and an open-innovation platform

As a fully open-source initiative, the Onyxia project aims at building "knowledge commons" by promoting and building software that can be easily reused in official statistics and beyond [38]. This concerns, first of all, the components on which Onyxia are based: both its constitutive technological bricks (Kubernetes, MinIO, Vault) as well as all the services from the catalog are open-source. But more crucially, all the code of the project is available openly on GitHub[10]. Alongside an in-depth documentation[11], this greatly facilitates the potential for other organiza-

[10] https://github.com/InseeFrLab/onyxia

[11] https://docs.onyxia.sh/

tions to create instances of data science platforms built upon the Onyxia software and tailor it to their respective needs (see figure 7). This enabled the project to attract a growing community of contributors from official statistics (Statistics Norway), NGOs (Mercator Ocean), research centres and even industry, thus transitioning progressively towards a more decentralized governance of the project. In the next years, the involvement of NSIs from the European Statistical System is expected to increase as Onyxia was chosen as the reference data science platform in the context of the AIML4OS project, a "One-Stop-Shop" for Artificial Intelligence/Machine Learning for Official Statistics in the European Statistical System.

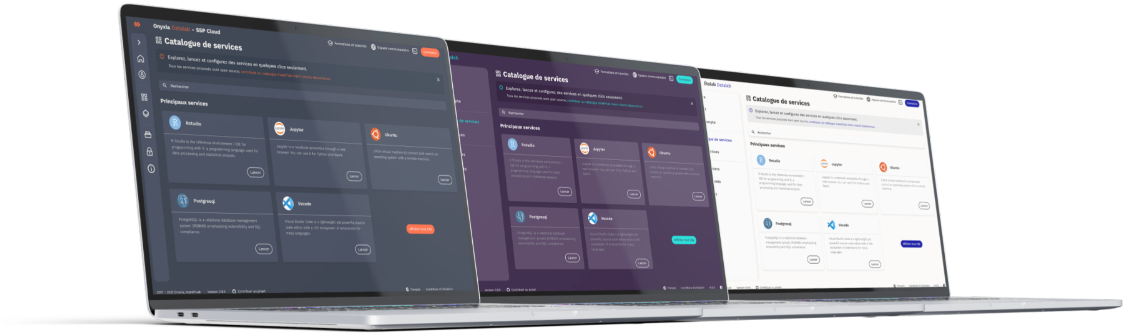


Fig. 7 One project, multiple instances: the UI is adaptable to the graphic identity of the organization

Another major way in which we try to build commons is by developing and maintaining a showcase instance of the Onyxia project, the SSP Cloud[6]. This platform, equipped with extensive and scalable computational resources[12], is designed to be a sandbox for experimenting with cloud technologies and new data science methods. The full catalog of services of Onyxia is available on the platform, enabling motivated users to go beyond mere experimentation by producing "proof of concepts", with full autonomy regarding the configuration and orchestration of their services.

Beyond its technical capabilities, the SSP Cloud is an endeavour at embodying the principles of open-innovation[5]. Deployed on internet[13], it is open not only to Insee employees, but also more broadly to all French governmental agencies and other European NSIs, and will serve as an official sandbox for the One-Stop-Shop program of the ESS described above. As a result, the platform is now widely used in the French National Statistical System and even beyond, with about 800 unique users per month. These users form a dynamic community thanks to a centralized discussion canal; they help improve the user experience by reporting bugs, by suggesting new features, and by contributing directly to the project. The fundamentally collaborative nature of

[12] On the physical side, the SSP Cloud consists in a Kubernetes cluster of about 20 servers, for a total capacity of 10 TB of RAM, 1100 CPUs, 34 GPUs and 150 TB of storage.

[13] `https://datalab.sspcloud.fr/`

the SSP Cloud has proven especially beneficial for organizing innovative events such as hackathons - both at the national and international levels - and in the academic sphere. It has become an integral resource for several universities and Grandes Ecoles in France, fostering the use of cloud-native and reproducible environments, and preventing vendor lock-in effect due to the over-reliance of educational organizations on proprietary cloud solutions. Finally, the SSP Cloud plays a significant role in leveraging open data, particularly as it highlights and makes use of the growing abundance of datasets openly published by various statistical agencies.

4 Case-study : using MLOps to improve NACE classification

This chapter aims, through a concrete example, to illustrate how Insee managed to deploy its first machine learning (ML) model into production. It will provide an in-depth description of the MLOps approach that this project strived to adhere to, focusing on the various technologies that were employed. In particular, we will highlight how cloud technologies were instrumental in building a solution iteratively and how Onyxia greatly facilitated this process by providing flexible development environments as well as tools to deploy and monitor models, promoting a continuous improvement loop. The entire project is available in open source[14] and remains under active development.

4.1 Improving the NACE classification process using ML methods

4.1.1 Motivation

Coding tasks are common operations for NSOs and can sometimes be challenging due to the size of nomenclatures. At Insee, a sophisticated coding tool called Sicore was developed in the 1990s to perform various classification tasks. It consists in a coding engine containing a large number of deterministic rules which identify ground-truth labels. Each input label goes through these rules and when a ground-truth label is recognized, the associated code is assigned. When the label is not recognized, it must be manually classified by an Insee agent.

Two main reasons drove the experimentation of new coding methods. Firstly, there was an internal change with the redesign of the Sirene registry, which lists all companies in France and assigns them a unique identifier, the Siren number, used by public institutions. The main goals of this revamping were to improve the daily management of the registry for Insee agents and to reduce waiting times for companies. Additionally, at the national level, the government launched a one-stop shop for business formalities, allowing more flexibility for business owners in

[14] https://github.com/orgs/InseeFrLab/teams/codification-ape/repositories

describing their main activities. Initial testing exercises revealed that Sicore was no longer the suitable tool for performing NACE classification, as only 30% of the input data were being automatically coded.

Three stakeholders were involved in this project: the *business team*[15] responsible for managing the Sirene registry, the *IT team* developing software related to the registry's operation, and the *innovation team* responsible for implementing the new coding tool. The latter team is the Insee Lab, which was created in 2017 with the objective of providing support to other teams on innovation topics to streamline their various projects.

4.1.2 Classification task

The project we are describing consists in a standard NLP classification problem. Starting from a textual description of the activity, we want to predict the associated class in the NACE Rev. 2 nomenclature. This nomenclature has the particularity of being hierarchical and contains 5 different levels[16]: section, division, group, class, and subclass. In total, 732 subclasses are included in the nomenclature, which is the level at which we aim to perform the classification. Table 1 provides an example of this hierarchical structure.

Level	**NACE**	**Title**	**Size**
Section	H	Transportation and storage	21
Division	52	Warehousing and support activities for transportation	88
Group	522	Support activities for transportation	272
Class	5224	Cargo handling	615
Subclass	**5224A**	**Harbour handling**	**732**

Table 1 NACE Nomenclature

With the establishment of the one-stop shop, business owners now describe their activity description with a free-text field. As a result, the new labels are very different from the harmonized labels that were previously received. Therefore, it was decided to work with ML models, that are known to be effective on supervised text classification tasks [23]. This represents a significant paradigm shift from Insee's perspective, as ML was not traditionally used in the actual production of official statistics. Besides, the perspective of putting the new model in production was considered from the outset, guiding numerous methodological and technical choices. As such, several strategic choices had to be made from the outset, including the methodology, the choice of a development environment consistent with the target production environment, and the adoption of collaborative work methods.

[15] We refer to the business team, the team that has business knowledge i.e. who knows the NACE perfectly.

[16] Actually, there are 5 different levels in France but only 4 at the European level.

4.1.3 Methodology

Text classification from the free-text field provided by business owners is a complex task: the activity descriptions are relatively short and thus contain limited statistical information, can contain spelling mistakes, and often require domain knowledge to be properly classified. On such task, traditional text analysis methods such as count vectorization or TF-IDF often fall short whereas neural-network-based embedding methods tend to perform better [23]. However, such architectures impose greater demands on the production setting, as they are much larger and often require specific hardware such as GPUs to be fine-tuned and perform inference with acceptable latency. These constraints led us away from the most powerful language models, such as Transformer models, and instead directed us towards the fastText model [19], a simpler embedding-based classifier. The fastText model is extremely fast to train, even from scratch, and inference doesn't require a GPU to achieve low latency time. Besides, the model yielded excellent performance results for our use-case which, considering the time and human resource constraints, were more than sufficient to enhance the existing process. Finally, the model's architecture is relatively simple, which simplified communication and adoption within the various Insee teams.

The fastText model relies on a bag-of-words model to obtain embeddings and a classification layer based on logistic regression. The bag-of-words approach involves representing a text as the set of vector representations of each of its constituent tokens. With FastText, the embedding of a sentence is computed as a function of the individual embeddings, typically the average. In the case of supervised text classification, the embedding matrix and the classifier's parameters are learned simultaneously during training by gradient descent, minimizing the cross-entropy loss function. The specificity of the fastText model compared to other embeddings-based approaches is that embeddings are not only computed on words but also on word n-grams and character n-grams, providing more context and reducing biases due to spelling mistakes. Figure 8 represents the full pipeline of operations performed by fastText on an example text input.

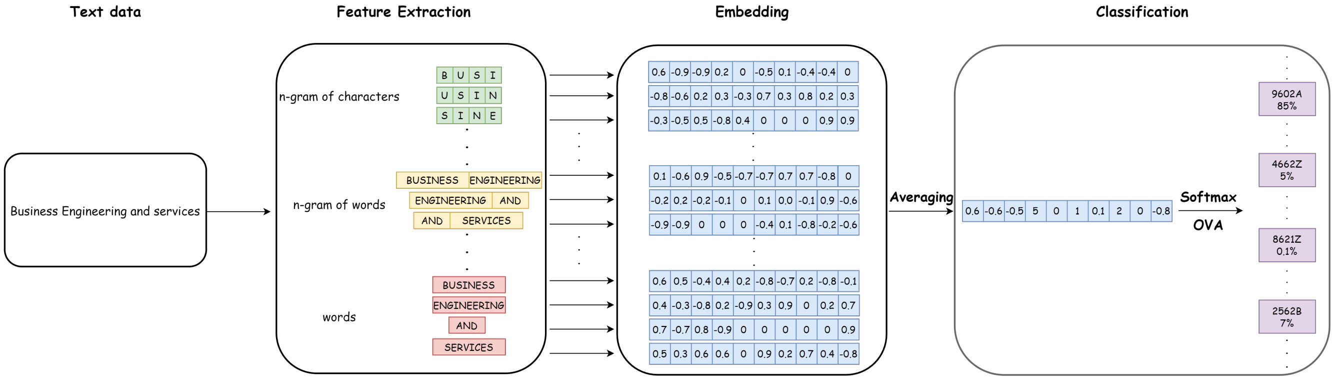


Fig. 8 Simplified architecture of the fastText model

4.2 A production-first approach with MLOps

From the very onset of this project, the target was to go beyond mere experimentation and put the model in production. Besides, the goal with this pilot project was also to build a template for future ML projects at Insee. We thus strove to enforce best development practices from the very beginning of the project: following community standards for code quality, using scripts-based development over notebooks, building a modular package-like structure, etc. However, compared to traditional development projects, machine learning projects have specific features that make it necessary to apply a complementary set of best practices, gathered under the name of MLOps.

4.2.1 From DevOps to MLOps

DevOps is a set of practices designed to foster collaboration between development (Dev) and operations (Ops) teams. The fundamental idea is to integrate the full lifecycle of a project in a single automated continuum. An important tool to achieve this continuity is CI/CD pipelines. With continuous integration (CI), each commit of new source code will trigger a pipeline of standardized operations, such as building the application, testing it and making it available as a release. Then, continuous deployment (CD) consists in tools to automate the deployment of the new code and limit manual intervention, while ensuring proper monitoring to guarantee process stability and security. This approach promotes a more rapid, continual release of necessary feature changes or additions. Furthermore, by encouraging collaboration between teams, DevOps also promotes a quicker cycle of innovation, allowing teams to address issues as they arise and incorporate feedback effectively throughout the project lifecycle.

The MLOps approach can be seen as an extension of DevOps, developed to address the specific challenges related to managing the lifecycle of ML models. Fundamentally, both DevOps and MLOps aim at building software in a more automated and robust manner. The main difference is that in MLOps, this software also has a machine learning component. Consequently, the lifecycle of the project gets more complex. The underlying ML model needs to be trained, often periodically in order to prevent potential performance losses over time. Data ingestion must also be included in the pipeline, as new data may be used to improve performance. Figure 9 presents the steps of a ML project using the continuous representation traditionally seen in DevOps. This illustrates a fundamental principle of MLOps, the need for continuous improvement, described in more details in section 4.2.2.

4.2.2 Principles of MLOps

MLOps is defined by a few core principles that are crucial for building production-ready and scalable ML applications. These principles are designed to address the specific challenges associated with ML workflows.

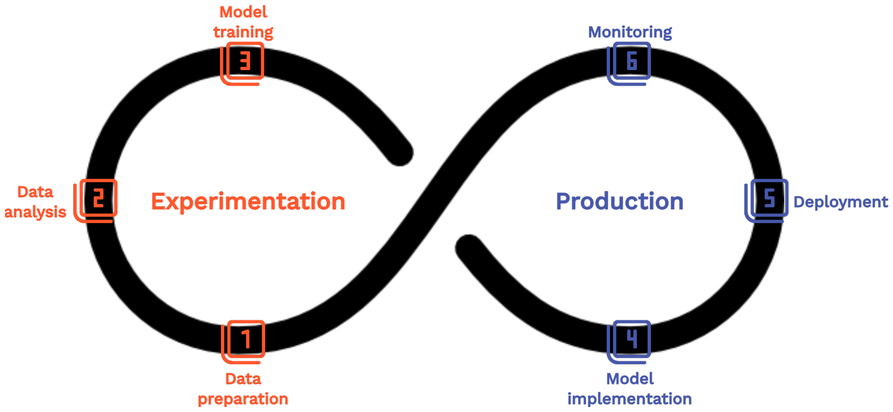


Fig. 9 The MLOps approach promotes a continuous management of ML projects lifecycle

The most fundamental principle of MLOps is continuous improvement, in order to reflect the iterative nature of ML projects. In the experimentation phase, the model is developed using a training dataset, which will generally differ somewhat from the target data. When a model is deployed in production, the new data that the model needs to perform prediction one can reveal insights about the model's performance and potential shortcomings. These insights necessitate a return to the experimentation phase, where data scientists adjust or redesign their models to address any discovered issues or to improve accuracy. This principle thus emphasizes the importance of building a feedback loop that enables ongoing enhancements throughout the lifecycle of a model. Automation, particularly through the use of CI/CD pipelines, plays a crucial role in this process by making the transition between experimentation and production phases more continuous. Monitoring is also an essential part of this process: a model deployed in production needs to be continuously assessed so as to detect data or concept drifts that may reduce the predictive performance of the model and thus necessitate further adjustments, such as re-training or fine-tuning the model.

Another major goal of MLOps is to promote reproducibility, which ensures that any ML experiment can be reliably repeated and will produce the same results. MLOps tools thus facilitate thorough logging of ML experiments, including data pre-processing steps, model hyper-parameters, and training algorithms. Data, models, and code are versioned, enabling teams to revert to previous versions if an update does not perform as expected. Finally, these tools help producing detailed specifications of the computing environment used to produce these experiments — such as versions of libraries — and often rely on containers to help replicate the same conditions in which the original model was developed.

Finally, MLOps aims at fostering collaborative work. ML-based projects generally involve a wider range of profiles: business units and data science teams on the one hand, developers and operations teams on the other. Like DevOps, MLOps thus emphasizes the need for a collaborative culture and to avoid working in silos.

MLOps tools generally include collaborative features, such as centralized stores for ML models or ML features which facilitate the sharing of components between team members and limit redundancy.

4.2.3 Implementation with MLFlow

Numerous tools have been developed to implement the MLOps approach in actual projects. All of these frameworks aim at enforcing, in some form, the core principles described above. In this project, we chose to rely on a popular open-source framework named MLFlow[17]. This choice doesn't indicate any inherent superiority of MLFlow over alternative software, but reflects a set of good properties associated with MLFlow that made it a very relevant solution for our application. First, it is Kubernetes-native, reflecting our choice of the underlying infrastructure for the project. Second, it aims at covering the entire lifecycle of ML projects, while other tools may be more specialised in some parts of it. Third, it exhibits great interoperability as it is well-interfaced with with popular ML libraries — such as PyTorch, Scikit-learn, XGBoost, etc. — and supports multiple programming languages — including Python, R, and Java, thus covering the spectrum of programming languages copmmonly used at Insee. Finally, it proved to be very user-friendly and thus encouraged adoption among the project members and facilitated continuous collaboration between them.

MLFlow provides a cohesive framework to operationalize MLOps principles effectively within ML projects. Data scientists can encapsulate their work in MLFlow Projects, which package ML code and its dependencies, ensuring that each project is reproducible and can be consistently re-executed. A project relies on an MLFlow Model, a standard format that is compatible with most ML libraries and offers a normalized way of serving the model, e.g. via an API. This interoperability and standardization are instrumental in supporting continuous improvement of the project, as models trained with a variety of packages can be readily compared or switched by one another without breaking any code. As experiments with various models progress, the Tracking Server logs detailed information about each run — hyperparameters, metrics, and outputs artifacts and metrics — which there again promotes reproducibility but also facilitates the model selection phase through a user-friendly interface. After this experimentation phase, selected models are integrated into the Model Registry, where they are versioned and staged for deployment. This registry serves as a centralized model store that enables the different project members or teams to collaboratively manage the lifecycle of the project. Figure 10 shows the core components of MLFlow and how they facilitate a more continuous and collaborative workflow inside a ML project.

[17] `https://github.com/MLflow/MLflow`

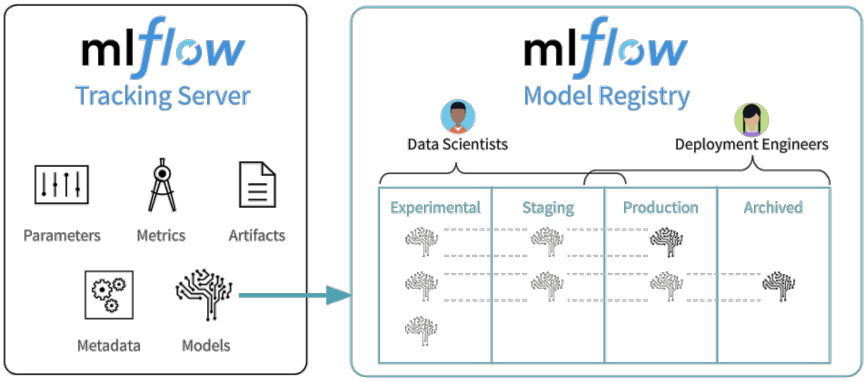


Fig. 10 Core components of MLFlow. Source: Databricks.

4.3 Leveraging cloud technologies to foster iterative development

While continuous improvement is a fundamental principle of MLOps, it is also a very demanding one. In particular, it requires to design and build our project as an integrated pipeline whose various stages are mainly automated, from data ingestion to monitoring the model in production. In this context, iterative development is essential in order to build a minimum viable product that is then refined over time. This section aims to show how cloud-native technologies, through the Onyxia project, were instrumental in building the project from the start as a collection of modular connected components, thus greatly enhancing the capacity for continuous refinement over time.

4.3.1 A flexible development environment

In a ML project, the flexibility of the development environment is essential. First, due to the diversity of tasks to be performed — data collection, preprocessing, modeling, evaluation, inference, monitoring, etc. Second, because ML is a fast-evolving field, it is preferable to build a ML app as a collection of modular components so as to be able to update components without disrupting the entire pipeline. As discussed in section 2.2, cloud-native technologies enable the creation of modular and scalable development environments.

However, as also discussed in section 3, access to such resources is not enough. A ML project that is developed following MLOps principles requires a wide variety of tools — data storage, interactive development environments to experiment freely, automation tools, monitoring tools, etc. While these tools can be installed on a Kubernetes cluster, making them available to data scientists in an integrated and pre-configured manner is essential to facilitate their adoption. Through its catalog of services and the automatic injection of configuration in the services, Onyxia enables building projects that rely on multiple cloud-native components that can communicate readily with each other.

The way model training was carried out for this project emphasizes the flexibility provided by Onyxia in the experimentation phase. All the code performing the training is written in Python in the context of a VSCode service. As personal S3 credentials are injected in each service at startup, the various users of the projects can interact directly with the training data stored on a S3 bucket on MinIO, Onyxia's default object storage solution. All experiments performed for the model selection phase are logged on a shared instance of MLFlow, which stores logged data on a PostgreSQL instance automatically launched on Kubernetes and artifacts (trained models and associated metadata) on MinIO. The model was trained using grid-search for hyperparameter tuning and evaluated through cross-validation, a combination that is known to provide a better evaluation of the generalization performance of the model but also requires a lot of computing resources, due to the combinatory nature of testing many hyperparameters combinations. In our case, we leveraged Argo Workflows, an open-source workflow engine designed to orchestrate parallel jobs on Kubernetes, each job being specified as an independent container. It then becomes straightforward to compare performances of the different trained models and select the best one using the comparison and visualization tools available in the UI of MLFlow.

In summary, the training stage was made efficient and reproducible thanks to the use of numerous modularly connected components — a distinctive feature of cloud-native technologies — readily made available to data scientists by Onyxia.

4.3.2 Deploying a model

Once candidate models have been optimized, evaluated and a best-performing model has been selected, the next step is to make it available to the application end users. Simply providing the trained model as an artifact or even just the code to train the model is not a convenient way to serve it, as it assumes that users have the resources, infrastructure, and knowledge required for training it under the same conditions. The goal is therefore to make the model available in a simple and interoperable manner, in the sense that it should be possible to query it with various programming languages. Furthermore, it should be possible for other applications to query the model in a programmatic way.

Against that background, we opted to serve the model through a REST API. This technology has become a standard way to serve ML models as they offer several benefits. First, they fit very well a cloud-oriented environment: similarly to the other components of our stack, it makes it possible to query the model using standard HTTP requests, which contributes to the modularity of the system. It also means that they are interoperable: as they rely on standards technologies for queries (HTTP requests) and responses (generally, a JSON-formatted string), they are mostly agnostic to the programming language used to request them. Finally, they offer great scalability because of their stateless design. As each request must contain all the information needed to understand and process the request, REST APIs can easily be duplicated on different machines to balance a challenging load — a process known as horizontal scaling.

To develop the API that serves the model, we used FastAPI[18], a fast and well-documented web framework for building APIs with Python. The API code and required software dependencies are encapsulated into a Docker image so that it can be deployed as a container on the Kubernetes cluster. An important benefit of using Kubernetes is the ability to scale the API — through the number of API pods effectively deployed — to the demand and provide automatic load-balancing. Upon startup, the API automatically retrieves the correct model from the MLflow model registry, which acts as a proxy from the actual artifact of the production model, stored on MinIO. Using a MLflow model enables to integrate the pre-processing step before each prediction to the API, greatly simplifying inference regardless of the ML framework used. This deployment process is summarized in Figure 11.

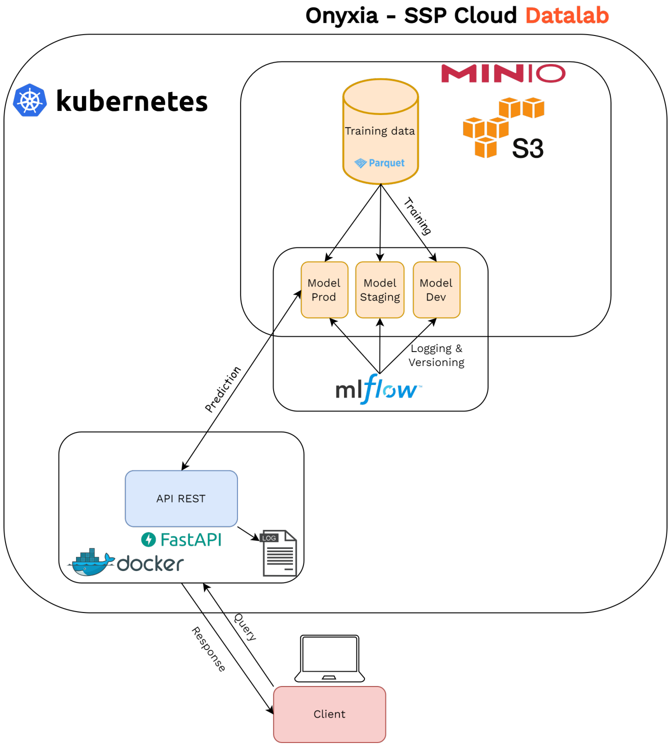


Fig. 11 A cloud-native approach to serve a ML Model using a REST API

[18] `https://fastapi.tiangolo.com`

4.3.3 Building an integrated pipeline

The architecture we built at this stage already reflects some important principles of MLOps. The use of containerization to deploy the API as well as the use of MLFlow to track the experiments while developing the model ensures reproducibility of the predictions. Using the central model registry provided by MLFlow facilitates the management of the lifecycle of the models in a collaborative way. Furthermore, the modularity of our architecture leaves room for further improvement as modular components can be easily added or modified without breaking the structure of the application as a whole. As we shall see in subsequent sections, this property was essential in building the application iteratively, enabling to add a monitoring layer (Section 4.3.4) and an annotation component (Section 4.3.5) to promote continuous improvement of the model.

However, the ability to refine the base architecture iteratively also requires more continuity in the process. At this stage, the deployment process involves several manual operations. For instance, adding a new feature to the API would require to build a new image, tag it, update the Kubernetes manifests used to deploy the API and enforce them on the cluster to replace the existing one with minimum downtime. Similarly, a change of the model served through the API would require a very simple modification of the code but several manual steps to update the version on the cluster. As a result, data scientists are not fully autonomous when it comes to prototyping and testing updated versions of the model or the API, which limit the potential for continuous improvement.

In order to automate this process, we built a CI/CD pipeline — a concept already presented in Section 4.2.1) — integrating these various steps. Figure 12 illustrates our specific implementation of a CI/CD pipeline. Any change in the code of the API repository triggers a CI build process (implemented with GitHub Actions) of the associated docker image, which is then released on a public container registry (DockerHub). This image can then be fetched and deployed by the container orchestrator (Kubernetes), by specifying and applying manually new manifests to update the Kubernetes resources of the API. However, the downside of this approach is that it limits reproducibility of the deployment, since each resource is handled independently by the orchestrator, so that the lifecycle of the API deployment as a whole is not managed. To overcome this shortcoming, we integrate the deployment part in a CD pipeline based on the GitOps approach: the resources manifest of the API are stored on a Git repository. The state of this "GitOps" repository is monitored by a Kubernetes operator (ArgoCD), so that any change in the application manifests is directly propagated to the deployment on the cluster. In this integrated pipeline, the only action needed for the data scientist to trigger an update of the API is to change the tag of the API image indicating the version to be deployed.

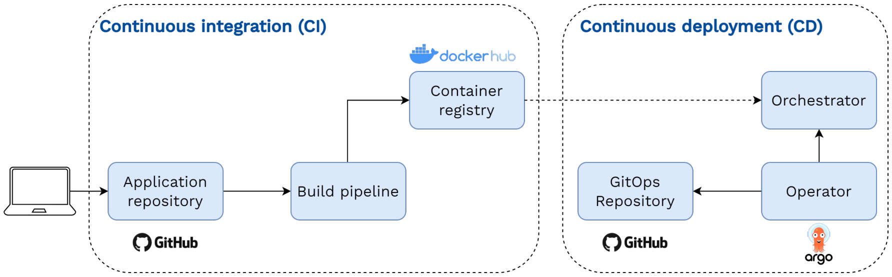


Fig. 12 Our implementation of a CI/CD pipeline

4.3.4 Monitoring a model in production

Once the modeling phase has been completed, along with training, optimization, and deployment of the model to be accessible to users, it's crucial to understand that the data scientist's responsibilities extend further. Traditionally, the responsibility of the data scientist is often limited to selecting the model to deploy, with the deployment task then delegated to data engineers. However, once in production, the model has not yet reached the end of its lifecycle, and it must be continuously monitored to prevent undesirable performance degradation.

The various components of MLOps, namely data (DataOps), models (ModelOps), and code (DevOps), add complexity to the lifecycle of a ML model by involving multiple stakeholders. Generally, three main actors are involved:

- Data scientists/data engineers
- IT teams
- Business teams

Sometimes, data scientists are integrated into business teams and data engineers into IT teams to foster better collaboration. This approach aims to streamline skills, unify terminologies, and align project objectives. In any cases, effective communication remains essential to ensure proper management of the lifecycle of a ML model, particularly concerning monitoring its behavior in a production environment. Continuous monitoring of the deployed model is extremely important to ensure the conformity of results to expectations, anticipate changes in data, and iteratively improve the model. The monitoring is a crucial element of the MLOps approach.

The concept of monitoring can take on different meanings depending on the context of the involved team. For IT teams, it primarily involves verifying the technical validity of the application, including aspects such as latency, memory consumption, or disk usage. Conversely, for data scientists or business teams, the focus is more on methodological monitoring of the model. However, real-time tracking of the performance of a ML model is often a complex task, given that ground truth is usually not known at the time of prediction. Therefore, it is common to use proxies to detect any signs of performance degradation. Two main types of degradation of a ML model are generally distinguished: data drift and concept drift.

- **Data drift**: Data drift occurs when the data used during inference in production exhibits significant differences compared to the data used during training, that is, $\mathbb{P}_{train}(X) \neq \mathbb{P}_{inference}(X)$.
- **Concept drift**: Concept drift is evoked when a change in the statistical relationship between the features (X) and the target variable (Y) is observed over time, that is, $\mathbb{P}_{train}(Y|X) \neq \mathbb{P}_{inference}(Y|X)$ and $\mathbb{P}_{train}(X) = \mathbb{P}_{inference}(X)$.

In our use case, our objective is to achieve the highest number of correctly classified textual description while minimizing the number of textual description requiring manual intervention. Thus, our goal is to distinguish correct predictions from incorrect ones without prior access to ground truth. To accomplish this, we have developed a confidence index, which represents the difference between the two highest scores. Therefore, for a given textual description, if the confidence index exceeds a determined threshold, the textual description is automatically coded. Otherwise, the textual description is manually coded by an Insee's agent who has access to the five most probable codes according to the model to optimize manual intervention. Defining the threshold for automatic coding of textual descriptions is crucial in production deployment. It involves making trade-offs between achieving a high automatic coding rate and preserving model performance, and vice versa.

To monitor the behavior of our model in production, we have developed an interactive dashboard that enables visualization of several metrics of interest for the business teams. Among these metrics are the number of requests per day and the rate of automatic coding per day based on a given confidence index threshold. This visualization allows business teams to understand the rate of automatic coding they would have obtained if they had chosen different thresholds. It is also possible to visualize the distribution of obtained confidence indices and to compare two temporal windows to determine if there are changes in the distributions of predictions returned by the model[19].

Confidence indices can be analyzed at finer levels of granularity based on the aggregation level of the nomenclature, to determine which classes are most difficult to predict and which have more or less occurrences. Another part of the dashboard focuses more on the input data of the model and details, for example, the average number of words per textual description, the average number of sentences per textual description, as well as the most frequent words, etc.

Although there are several solutions for monitoring ML models such as Evidently or Sifflet, we preferred to implement our own solution. Indeed, at the time of our trials, none of the open-source solutions available on the market seemed optimal to us. However, the field of monitoring ML models is still emerging and actively developing, so the statements made today may no longer be true at the time of reading these lines. Before opting, as we did, for an internal solution, we recommend exploring the available tools first.

[19] To compare distributions, you need to calculate the distances between them using, for example the Bhattacharyya distance, the Kullback-Leibler divergence, the Hellinger distance or to perform statistical tests such as the Kolmogorov–Smirnov test, or the χ^2-test.

In our case, we chose to use tools that we already mastered and that seemed suitable for our problem. We thus used DuckDB for optimized management of our data and the Dashboard feature of Quarto for creating the interactive dashboard.

First, to monitor in real-time the activities of your API, it is essential to integrate logs, whether to monitor technical or methodological aspects. We made the decision, which may be subject to debate, to include *business information* such as the output code and its corresponding confidence index in the API logs. The objective was to be able to build our dashboard based on the logs returned by the API. To exploit these logs, we set up a kind of Extract-Transform-Load (ETL) process in Python, which retrieves the API logs and transforms them into partitioned parquet files exploitable by the dashboard. Ideally, we would have preferred to rely on a tool capable of performing this parsing natively and more structured than us. Parseable could have been such a solution, but unfortunately, it was not installable on the SSP Cloud for technical reasons. We then use Quarto and its Dashboard feature to create an interactive dashboard. The various metrics present in the dashboard are calculated by making SQL queries directly on our parquet files with DuckDB. The dashboard is deployed in the same way as our API, that is, in a container based on an image that is automatically built via Github Actions as soon as the code associated with the dashboard is modified. Figure 13 depicts the setup of our monitoring system, highlighting the various technologies employed.

In summary, our monitoring pipeline consists of:

1. Including *business logs* in the API for every request.
2. Utilizing ArgoCD to schedule a nightly ETL job, generating partitioned parquet files stored on MinIO.
3. Employing ArgoCD for continuous deployment, where the dashboard dynamically retrieves data from the parquet file via DuckDB, computes metrics, and offers interactive visualization.

4.3.5 Promoting continuous improvement of the model

In all ML projects, it is well-established that data is key, and in supervised ML problems, annotations play a crucial role in model performance. At the project's onset, time constraints compelled us to swiftly achieve results, precluding the luxury of conducting a comprehensive annotation campaign. Consequently, we never had access to a truly annotated gold-standard test sample on which we could rely 100%. To evaluate the performance of our model, we relied on a subset of our training dataset (test/train split), knowing that its labeling was not perfect. After several months of model deployment, it became imperative to build such a gold-standard test sample. Firstly, this sample allows us to have an unbiased view of the model's performance in production on real data, particularly on data that has been automatically coded. Additionally, this sample enabled us to identify and correct inconsistencies present in our initial training dataset.

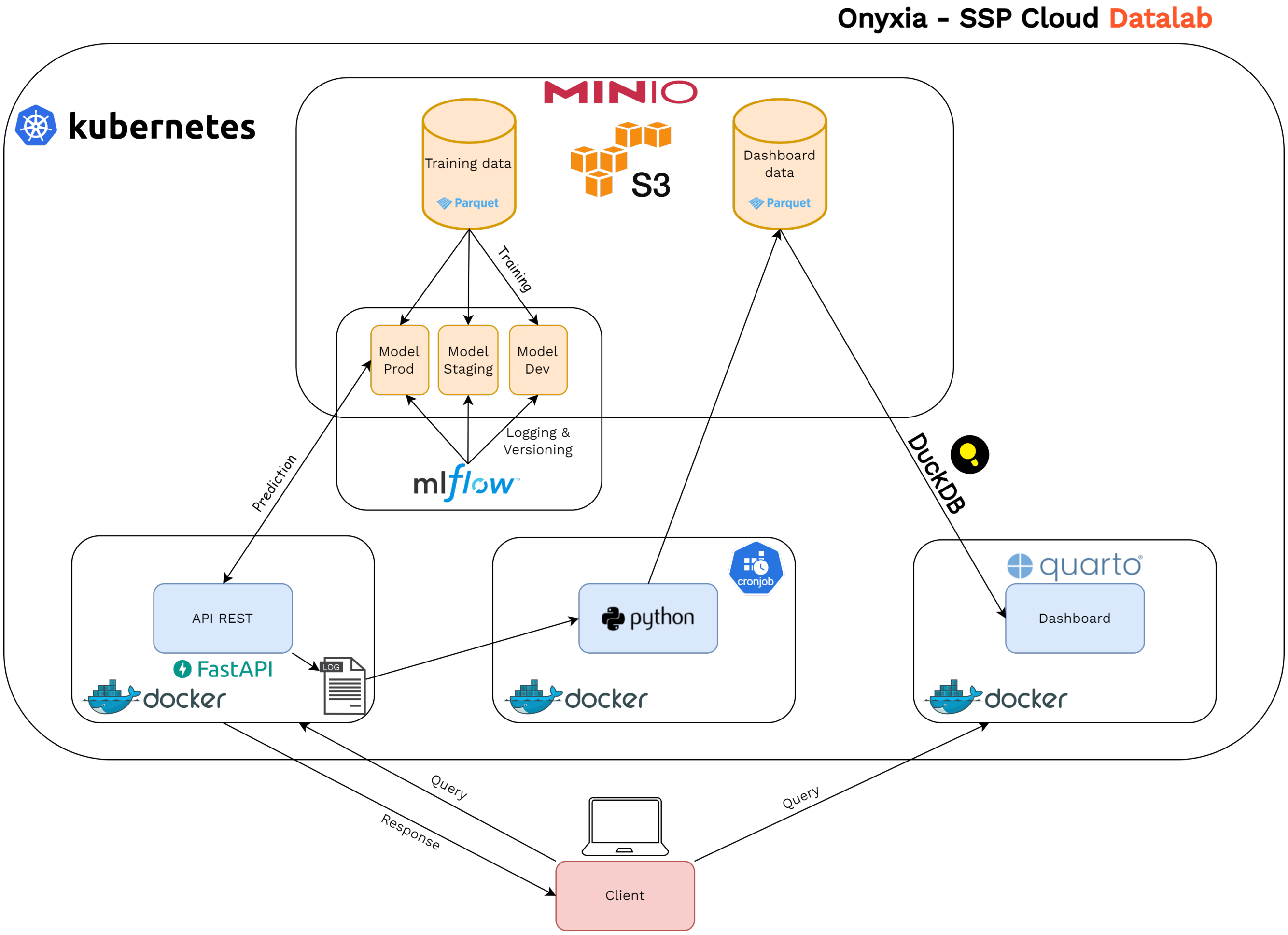


Fig. 13 In-house implementation of a monitoring system

Convincing the business teams of the significance of annotation proved challenging, as they were reluctant to reallocate new human resources, which necessitated specialized expertise, on a tedious task. However, the model's increased automation in coding enabled the teams to shift their focus to annotation tasks.

Another reason that motivated the implementation of annotation campaigns is the redesign of the NACE nomenclature in 2025. From 2025 onwards, European statistical institutes will be required to use the latest version of NACE, namely NACE Rev. 2.1. This revision brings substantial changes that will require retraining a new model. For this, a new training dataset with the correct annotations is peremptory. A back-propagation work on the old training dataset is also considered to increase the size of the dataset. Thus, a dual annotation campaign was initiated in early 2024 on the SSP Cloud platform.

To carry out these annotation campaigns, we used the Label Studio service, available on the SSP Cloud. Label Studio is an open-source platform designed to facilitate data annotation. The first annotation campaign aims to create a test dataset allowing continuous evaluation of our model and improvement of the training dataset if necessary. To do this, we create a pool of text descriptions randomly sampled from the data passed through the API over the past three months. This sample is then sent to annotation by NACE experts using the UI of Label Studio. The annotation results are automatically saved on MinIO, transformed into parquet format, and then integrated into the monitoring dashboard to compute and observe various model performance metrics. The second annotation campaign is dedicated to the transition to the new version of NACE. The objective is to allocate certain experts to annotation in the new version of NACE to create a training dataset for the future model. In this revision, some codes are univocal, meaning for example that code 0112Z will become 0112Y, while others are multivocal, such as 0119Z, which can become either 0113Y or 0119Y. The challenge lies primarily with the latter, which is why, to streamline this annotation process, only the textual descriptions of multivocal codes are sent for annotation on Label Studio.

Once again, the presence of a cloud-native technology on the platform, such as Label Studio, allowed us to swiftly kickstart this annotation campaign while seamlessly integrating it with the various components of our architecture, as depicted in Figure 14.

5 Discussion

Initially developed as an internal project, Onyxia has gained recognition beyond the realm of Insee or the French administration. Aware of the necessity to foster autonomy in order to leverage the full potential of data science, several organizations now have a production instance of Onyxia running, and multiple others are in the process of either testing or implementing one. Besides, the choice of Onyxia as the reference data science platform in the context of the One-Stop-Shop for Artificial Intelligence/Machine Learning for Official Statistics (AIML4OS) will further facilitate its adoption within the ESS. This trend is naturally very beneficial to the Onyxia project, as it moves from a project developed in open-source - but mainly at Insee - to a full open-souce project with a growing base of contributors. This in turn facilitates its adoption by other organizations, since it gives more guarantees on its sustainability independently from Insee's strategy. The governance of the project is currently evolving to reflect this trend. For instance with the organization of monthly community calls and the creation of a public channel and roadmap for the project[20].

[20] All information are available on the GitHub depository of the project : https://github.com/InseeFrLab/onyxia

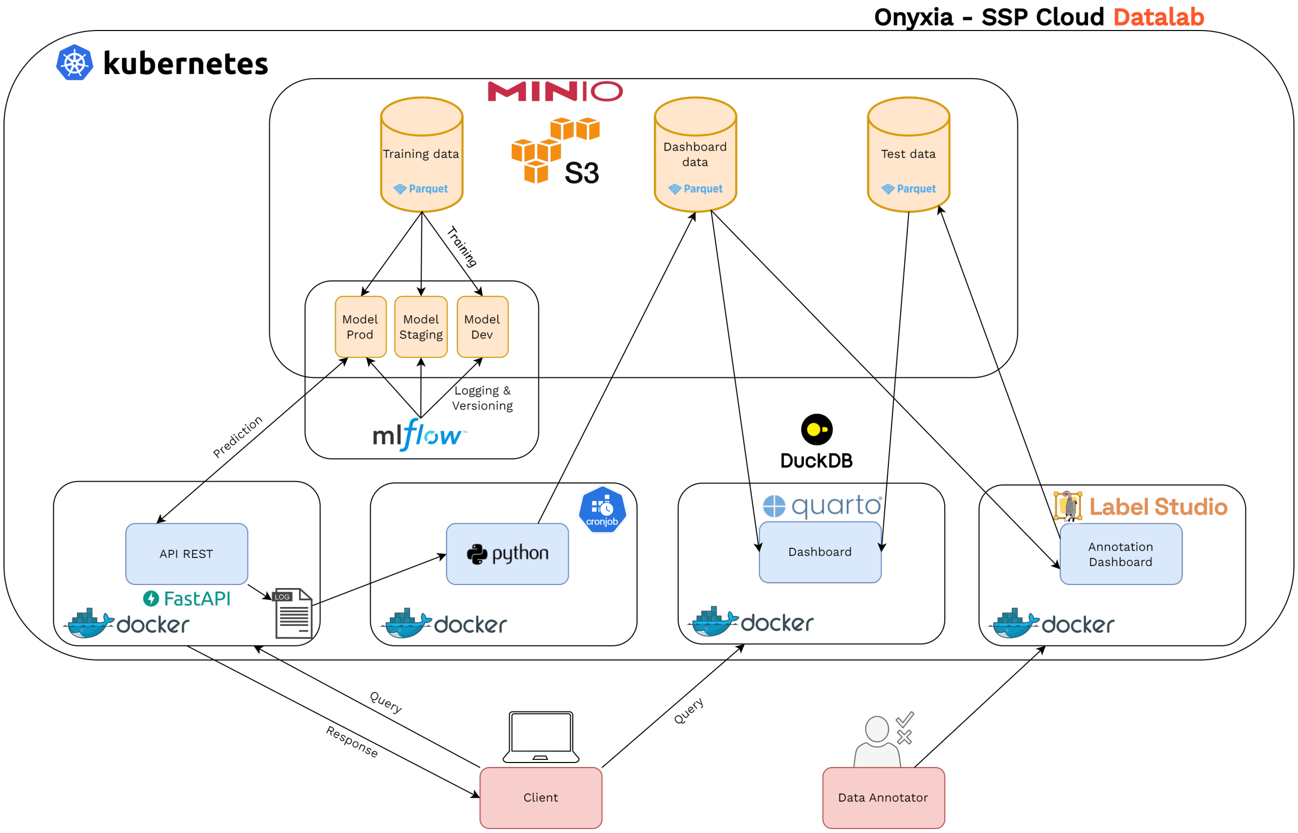


Fig. 14 Implementation of data quality control using Label Studio

Despite this success, we observe several limitations to the widespread adoption of the project in organizations. First, it is essential to remind that the fundamental choice made by organizations that adopt Onyxia is not the software itself, but the underlying technologies: containerization (through Kubernetes) and object storage. These technologies can represent substantial entry costs for organizations, as they demand a significant commitment to developing and maintaining skills which are not readily found in NSOs. Yet, the general trend towards cloud-native solutions among data-centric organizations suggests a favorable shift that could mitigate these challenges over time.

Similarly, the transition towards cloud-native technologies induces entry costs for statisticians. First, they often deal with a loss of references regarding where computations actually happen: while they may be accustomed to performing computation on centralized servers rather than a personal computer, the container adds a layer of abstraction that make the location hard to grasp at first. But the major perceived change in this paradigm is the loss of data persistence. In traditional setups - either a personal computer or a server accessed through a virtual desktop - the code, the data and the computing environment are kind of mixed in a black box fashion. On the contrary, containers have no persistence by design. While object storage provides this persistence, a proper use of these infrastructures for statistical projects require a variety of tools and corresponding skills: using a version control system for the code, interacting with the object storage API to store the data, providing configuration files or secrets as inputs, etc. In a way, these entry costs can be seen as the "price" of autonomy: thanks to cloud-native technologies, statisticians now have access to scalable and flexible environments that enable them to experiment more freely, but this autonomy requires a significant skills upgrade which may be overwhelming at first and limit adoption. However, our experience at Insee suggests that this effect can largely be mitigated through a combination of training statisticians to development best practices and accompanying statistical projects when transitioning to cloud infrastructures.

While Onyxia has significantly democratized access to cloud-native technologies for statisticians, the journey towards integrating data science within NSOs encompasses broader organizational challenges beyond the technical realm. In particular, the deployment of our first machine learning model in production highlighted the necessity of overcoming skill compartmentalization across IT, business, and innovation teams. Effective collaboration and the seamless integration of machine learning into statistical processes require not only shared goals and methodologies but also a convergence of diverse skills and knowledge. Addressing these challenges involves strategic measures such as embedding data science capabilities within business teams for better alignment with project objectives, and initiating cross-disciplinary training to harmonize the toolsets and languages across teams. Ultimately, the successful transition to a data science-driven approach in statistical production is contingent upon a balanced strategy that marries technical solutions like Onyxia with comprehensive organizational adjustments, fostering a culture of collaboration, continuous learning, and innovation.

References

1. Daniel Abadi, Peter Boncz, Stavros Harizopoulos, Stratos Idreos, Samuel Madden, et al. The design and implementation of modern column-oriented database systems. *Foundations and Trends® in Databases*, 5(3):197–280, 2013.
2. Abdullah I Abdelaziz, Kent A Hanson, Charles E Gaber, and Todd A Lee. Optimizing large real-world data analysis with parquet files in r: A step-by-step tutorial. *Pharmacoepidemiology and Drug Safety*, 2023.
3. Afshin Ashofteh and Jorge M Bravo. Data science training for official statistics: A new scientific paradigm of information and knowledge development in national statistical systems. *Statistical Journal of the IAOS*, 37(3):771–789, 2021.
4. Ouafa Bentaleb, Adam SZ Belloum, Abderrazak Sebaa, and Aouaouche El-Maouhab. Containerization technologies: Taxonomies, applications and challenges. *The Journal of Supercomputing*, 78(1):1144–1181, 2022.
5. Henry William Chesbrough. *Open innovation: The new imperative for creating and profiting from technology*. Harvard Business Press, 2003.

6. Frédéric Comte, Arnaud Degorre, and Romain Lesur. Sspcloud: a creative factory to support experimentations in the field of official statistics. *Courrier des Statistiques, INSEE*, 7:68–85, 2022.
7. Thomas H Davenport and DJ Patil. Data scientist. *Harvard business review*, 90(5):70–76, 2012.
8. Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
9. Pascaline Descy, Vladimir Kvetan, Albrecht Wirthmann, and Fernando Reis. Towards a shared infrastructure for online job advertisement data. *Statistical Journal of the IAOS*, 35(4):669–675, 2019.
10. DGINS. Bucharest memorandum on official statistics in a datafied society. `https://ec.europa.eu/eurostat/documents/13019146/13237859/\The+Bucharest+Memorandum+on+Trusted+Smart+Statistics+FINAL.pdf/7a8f6a8f-9805-e77c-a409-eb55a2b36bce?t=1634144384767`, 2018.
11. Bijesh Dhyani and Anurag Barthwal. Big data analytics using hadoop. *International Journal of Computer Applications*, 108(12):0975–8887, 2014.
12. European Commission. European statistics code of practice — revised edition 2017. `https://ec.europa.eu/eurostat/web/products-catalogues/-/ks-02-18-142`.
13. EUROSTAT. Essnet big data 2 - final technical report. `https://wayback.archive-it.org/12090/20221110013641/https://ec.europa.eu/eurostat/cros/system/files/\wpa_deliverable_a5_final_technical_report_2021_06_29.pdf`, 2021.
14. Apache Software Foundation. Apache parquet. `https://parquet.apache.org/`, 2013.
15. Apache Software Foundation. Apache arrow. `https://arrow.apache.org/`, 2016.
16. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, 2003.
17. Taeke Gjaltema. High-level group for the modernisation of official statistics (hlg-mos) of the united nations economic commission for europe. *Statistical Journal of the IAOS*, 38(3):917–922, 2022.
18. Shivani Gokhale, Reetika Poosarla, Sanjeevani Tikar, Swapnali Gunjawate, Aparna Hajare, Shilpa Deshpande, Sourabh Gupta, and Kanchan Karve. Creating helm charts to ease deployment of enterprise application and its related services in kubernetes. In *2021 international conference on computing, communication and green engineering (CCGE)*, pages 1–5. IEEE, 2021.
19. Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.
20. Alexander Kowarik and Magdalena Six. Quality guidelines for the acquisition and usage of big data with additional insights on web data. In *4th International Conference on Advanced Research Methods and Analytics (CARMA 2022)*, pages 269–269. Editorial Universitat Politècnica de València, 2022.
21. Marie Leclair et al. Using scanner data to calculate the consumer price index. *Courrier des statistiques*, 3:61–75, 2019.
22. Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. A survey of devops concepts and challenges. *ACM Computing Surveys (CSUR)*, 52(6):1–35, 2019.
23. Qian Li, Hao Peng, Jianxin Li, Congying Xia, Renyu Yang, Lichao Sun, Philip S Yu, and Lifang He. A survey on text classification: From traditional to deep learning. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(2):1–41, 2022.
24. Yun Li, Manzhu Yu, Mengchao Xu, Jingchao Yang, Dexuan Sha, Qian Liu, and Chaowei Yang. Big data and cloud computing. *Manual of digital earth*, pages 325–355, 2020.
25. Ling Liu. Computing infrastructure for big data processing. *Frontiers of Computer Science*, 7:165–170, 2013.
26. Sybille Luhmann, Jacopo Grazzini, Fabio Ricciato, Mátyás Mészáros, Jean-Marc Museux, and Martina Hahn. Promoting reproducibility-by-design in statistical offices. In *2019 New Techniques and Technologies for Statistics (NTTS) conference*, 03 2019.

27. Marcia McNutt. Reproducibility. *Science*, 343(6168):229–229, 2014.
28. Mike Mesnier, Gregory R Ganger, and Erik Riedel. Object-based storage. *IEEE Communications Magazine*, 41(8):84–90, 2003.
29. David Moreau, Kristina Wiebels, and Carl Boettiger. Containers for computational reproducibility. *Nature Reviews Methods Primers*, 3(1):50, 2023.
30. Justice Opara-Martins, M Sahandi, and Feng Tian. A holistic decision framework to avoid vendor lock-in for cloud saas migration. *Computer and Information Science*, 10(3), 2017.
31. Justice Opara-Martins, Reza Sahandi, and Feng Tian. Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. *Journal of Cloud Computing*, 5:1–18, 2016.
32. Mark Raasveldt and Hannes Mühleisen. Duckdb: an embeddable analytical database. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1981–1984, 2019.
33. Fabio Ricciato, Freddy De Meersman, Albrecht Wirthmann, Gerdy Seynaeve, and Michail Skaliotis. Processing of mobile network operator data for official statistics: the case for public-private partnerships. In *104th DGINS conference*, 2018.
34. Fabio Ricciato, Albrecht Wirthmann, Konstantinos Giannakouris, Michail Skaliotis, et al. Trusted smart statistics: Motivations and principles. *Statistical Journal of the IAOS*, 35(4):589–603, 2019.
35. Anam Saiyeda and Mansoor Ahmad Mir. Cloud computing for deep learning analytics: A survey of current trends and challenges. *International Journal of Advanced Research in Computer Science*, 8(2), 2017.
36. David Salgado, Luis Sanguiao-Sande, Sandra Barragán, Bogdan Oancea, and Milena Suarez-Castillo. A proposed production framework with mobile network data. In *ESSnet Big Data II - Workpackage I - Mobile Network Data*, 2020.
37. S Samundiswary and Nilma M Dongre. Object storage architecture in cloud for unstructured data. In *2017 International Conference on Inventive Systems and Control (ICISC)*, pages 1–6. IEEE, 2017.
38. Charles M Schweik. Free/open-source software as a framework for establishing commons in science. 2006.
39. Steven Vale. International collaboration to understand the relevance of big data for official statistics. *Statistical Journal of the IAOS*, 31(2):159–163, 2015.
40. Rafael Vaño, Ignacio Lacalle, Piotr Sowiński, Raúl S-Julián, and Carlos E Palau. Cloud-native workload orchestration at the edge: A deployment review and future directions. *Sensors*, 23(4):2215, 2023.
41. Qi Zhang, Ling Liu, Calton Pu, Qiwei Dou, Liren Wu, and Wei Zhou. A comparative study of containers and virtual machines in big data environment. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 178–185. IEEE, 2018.