Mettre les technologies cloud au service de la production statistique

Romain Avouac
Insee
romain.avouac@insee.fr

Thomas Faria
Insee
thomas.faria@insee.fr

Frédéric Comte
Insee
frederic.comte@insee.fr

2025-02-23

Résumé French

1 Introduction

L'exploitation de nouvelles sources de données en complément des enquêtes traditionnelles est une orientation majeure du Système Statistique Européen (SSE) pour améliorer les processus de production statistique. Cette évolution s'accompagne d'innovations non seulement méthodologiques mais également des systèmes d'information afin de tirer parti du potentiel de ces sources — plus grande disponibilité, résolution spatio-temporelle accrue, etc. — tout en faisant face à leur complexité et à leurs limites. Parmi ces innovations figurent les méthodes d'apprentissage automatique et leurs applications prometteuses dans les domaines de la codification, des redressements et de l'imputation [1]. Les multiples défis auxquels font face les instituts statistiques dans ce contexte d'évolution sont abordés dans le *Mémorandum de Bucarest sur les statistiques publiques dans une société numérisée*, qui anticipe que « la variété des nouvelles sources de données, paradigmes computationnels et outils nécessitera des adaptations de l'architecture métier statistique, des processus, des modèles de production, des infrastructures informatiques, des cadres méthodologiques et de qualité, ainsi que des structures de gouvernance correspondantes », et invite en conséquence le SSE à évaluer les adaptations requises et à les prioriser [2]. Cette évolution est également largement visible dans le cadre du service statistique public (SSP), dont elle constitue l'une des lignes directrices de la stratégie à horizon 2025 [3].

Face à ces transformations, de nombreux travaux ont été menés dans le cadre de projets innovants visant à qualifier l'utilisation de sources de données non-traditionnelles dans la production de statistiques publiques. Dans le cadre des projets ESSnet Big Data I (2016-2018) et II (2018-2020), les instituts statistiques nationaux (INS) ont travaillé sur une large gamme de thématiques (offres d'emploi en ligne, transactions financières, traces GPS, etc.) afin de poser les premiers jalons pour l'intégration de ces sources dans les processus de production et d'identifier leurs limites [4]. A l'Insee, les travaux sur l'exploitation des données mobiles [5] ou des données de caisse [6] ont

permis d'illustrer le potentiel de ces sources pour construire de nouveaux indicateurs ou raffiner des indicateurs existants. Néanmoins, si un travail conséquent a été consacré au développement de cadres méthodologiques [7], [8], de lignes directrices sur la qualité [9], ainsi qu'à la conception de processus sécurisant l'acquisition de données dans le cadre de partenariats avec des acteurs privés [10], les infrastructures informatiques et les compétences nécessaires pour gérer ces nouveaux objets sont restées peu abordées dans la littérature.

On désigne généralement par « *big data* » les données qui se distinguent par leur volume (souvent de l'ordre de plusieurs centaines de Go voire du To), leur vélocité (vitesse de génération, parfois proche du temps réel) ou leur variété (données structurées mais aussi non structurées, telles que du texte ou des images). Cette caractérisation s'applique naturellement aux données massives générées de manière automatique par les comportements individuels (données mobiles, données de caisses) ou encore aux données récupérées depuis internet via des méthodes de *web scraping*. Mais elle est également pertinente pour caractériser certaines sources de nature administrative déjà utilisées pour la production statistique. Le projet Résil en est une bonne illustration dans la mesure où il repose sur l'appariement de multiples sources administratives volumineuses (D-SN, POTE, etc.) devant être accueillies avec différentes temporalités — dont certaines en continu, comme le RNIPP — et dans des formats hétérogènes, plus ou moins structurés [11]. Dans les deux cas, l'intégration de telles sources dans un processus de production statistique pose des défis qui se situent au confluent de la méthodologie statistique et de la technique informatique, et relèvent ainsi du domaine de la *data science*. Dans ses multiples acceptions, le terme *data scientist* reflète en effet l'implication croissante des statisticiens dans le développement informatique et l'orchestration de leurs opérations de traitement des données, au-delà des seules phases de conception ou de validation [12]. Hors si l'on observe un nombre croissant de statisticiens publics formés aux méthodes de *data science*, leur capacité à tirer pleinement parti des sources non-traditionnelles pour la production statistique se heurte à plusieurs défis.

Un premier défi réside dans l'absence d'infrastructures informatiques adaptées aux nouvelles sources de données et aux nouvelles méthodes exploitées par les INS. Par exemple, les sources *big data* nécessitent de très grandes capacités de stockage et s'appuient souvent sur des infrastructures et des méthodes de calcul distribués pour être traitées en temps raisonnable [13]. De même, l'adoption de nouvelles méthodes statistiques basées sur des algorithmes d'apprentissage automatique requiert de la puissance de calcul, en particulier des GPUs (processeurs graphiques) dans le cadre du traitement du texte ou de l'image [14]. De telles ressources sont rarement disponibles dans les infrastructures informatiques traditionnelles. Lorsque des infrastructures de calcul adaptées sont disponibles, comme les supercalculateurs (HPC) utilisés dans certains domaines de recherche, elles nécessitent des compétences spécifiques, notamment pour leur mise en place et leur maintenance, qui sont rarement disponibles au sein des INS. Pour lever cette barrière, il est nécessaire d'adopter des infrastructures informatiques qui reflètent les besoins des projets de *data science* actuels en permettant de découpler le stockage du traitement de la donnée, afin de s'adapter rapidement à l'évolution des besoins.

Un autre défi majeur est de mettre à disposition des statisticiens des environnements de développement leur permettant d'expérimenter librement. Cette agilité est limitée lorsque les environ-

nements de calcul dépendent excessivement des départements informatiques pour provisionner des ressources ou installer de nouveaux logiciels. Dans les configurations traditionnelles où les statisticiens effectuent leurs calculs sur des ordinateurs personnels ou des bureaux virtuels sur des architectures centralisées[1], les départements informatiques privilégient généralement la sécurité et la stabilité du système là où l'innovation réside dans la capacité à intégrer rapidement de nouveaux outils. De plus, la rigidité de ces environnements rend difficile la mise en œuvre de certaines bonnes pratiques de développement. Par exemple, la reproductibilité des traitements statistiques n'est pleinement possible que dans des environnements dont les statisticiens peuvent eux-mêmes spécifier les caractéristiques (version du langage statistique, version des packages, librairies système nécessaires, etc.) [15]. L'enjeu est donc de concevoir des environnements agiles qui favorisent l'innovation sans compromettre la sécurité des processus de production.

Un troisième défi concerne la difficulté de passer des expérimentations innovantes à des solutions en production. Même lorsque les statisticiens ont accès à des environnements « bac à sable » leur permettant par exemple de développer une application interactive ou d'entraîner un modèle, la transition vers le déploiement en production de tels objets est coûteuse. Les environnements de production diffèrent souvent des environnements de développement, ce qui peut entraîner des coûts de développement supplémentaires importants pour passer d'une preuve de concept à une solution industrialisée qui rend du service à des utilisateurs dans la durée. Par exemple, dans le cas des projets d'apprentissage automatique, les modèles déployés nécessitent un suivi continu pour s'assurer qu'ils conservent leur performance et leur utilité au fil du temps, ce qui requiert généralement des ré-entraînements périodiques. Ainsi, notre choix doit s'orienter vers des infrastructures informatiques qui d'une part permettent de mettre à disposition des environnements de développement calqués sur les environnements de production, et qui d'autre part favorisent une collaboration plus continue entre statisticiens et équipes informatiques dans la gestion du cycle de vie des projets de *data science*.

Ces différents défis ont un thème sous-jacent commun : le besoin d'une plus grande autonomie. La capacité des méthodes de *data science* à améliorer la production des statistiques publiques dépend crucialement de la capacité à intégrer de nouvelles sources et de nouvelles méthodes de traitement de la donnée dans les processus statistiques. Et cette capacité dépend à son tour de la disponibilité d'environnements de calcul adaptés aux besoins de la *data science* moderne (capacités de stockage suffisantes, infrastructures distribuées, disponibilité de GPUs) permettant aux statisticiens de s'approprier de nouvelles méthodologies et de nouveaux outils hors des contraintes des environnements informatiques traditionnels. Avec cet article, notre objectif est de montrer comment les technologies *cloud* apportent une réponse globale à ces défis, en offrant aux statisticiens des environnements qui les rendent autonomes dans l'accès aux ressources nécessaires à leurs traitements en *self* tout en les rapprochant des infrastructures de production déployées à l'Insee.

[1] `AUSv3` est un exemple d'une telle infrastructure. Les utilisateurs y accèdent via leur poste de travail, qui sert de point d'accès à un bureau virtuel qui « reproduit » l'expérience habituelle du poste de travail. Néanmoins, les calculs qui sont lancés — via `R` ou `Python` par exemple — sont effectués sur des machines virtuelles (VM) de calcul dédiées, et non sur le poste de travail lui-même.

La Section 2 offre une analyse approfondie des derniers développements de l'écosystème de la donnée, mettant en lumière les choix technologiques qui ont façonné le développement d'un environnement de calcul moderne à l'Insee, adapté aux besoins spécifiques de la *data science*. Nous montrons comment certaines technologies dites *cloud-native*, comme la conteneurisation et le stockage objet, permettent de créer des environnements évolutifs et flexibles qui favorisent l'autonomie tout en promouvant la reproductibilité des projets statistiques. Malgré leurs atouts, la disponibilité de ces technologies n'implique pas leur adoption dans l'organisation, dans la mesure où elles s'avèrent complexes à configurer du point de vue informatique et nécessitent une adaptation de la part des statisticiens pour les exploiter dans le cadre de projets statistiques. Dans la Section 3, nous détaillons comment le projet Onyxia, développé à l'Insee, a permis de mettre les technologies *cloud* au service des statisticiens grâce à une interface dynamique et un catalogue étendu de services de *data science* prêts à l'emploi. Enfin, la Section 4 illustre l'application pratique de ces technologies à un projet innovant de l'Insee : la classification des activités des entreprises françaises (APE) à l'aide de méthodes d'apprentissage automatique. Ce retour d'expérience vise à montrer comment l'utilisation de ces technologies permet de faciliter et fiabiliser la mise en production de modèles d'apprentissage en permettant d'appliquer les bonnes pratiques issues du *MLOps*.

2 Construire un environnement de *data science* moderne grâce aux technologies cloud

L'intégration des nouvelles sources de données et des méthodologies innovantes dans les processus de production statistique pose plusieurs défis. Le premier est celui de l'infrastructure : ces sources de données sont souvent massives, donc coûteuses en stockage, et les méthodologies permettant de les traiter sont généralement gourmandes en ressources de calcul. L'autre défi est de permettre aux statisticiens d'accéder à ces ressources à travers des environnements qui favorisent l'autonomie — qu'il s'agisse de dimensionner les ressources de calcul en fonction des chaînes de productions statistiques à traiter, de déployer des preuves de concept avec agilité, de travailler de manière collaborative, etc. Dans ce contexte, l'objectif est de concevoir une plateforme de *data science* qui non seulement rende possible le traitement des données massives, mais qui permette également aux statisticiens de tester et déployer de nouveaux outils de manière autonome. Pour se faire, la première étape a été d'analyser les évolutions de l'écosystème de la donnée afin d'en extraire les tendances significatives qui marquent les choix des organisations. L'analyse de ces évolutions indique une tendance générale vers l'adoption des technologies *cloud*, en particulier les conteneurs et le stockage objet, qui permettent de construire des infrastructures capables de gérer des ensembles de données volumineux et variés de manière flexible et efficiente. Ces technologies se révèlent particulièrement pertinentes pour construire des environnements de calcul destinés à la statistique publique, dans la mesure où elles favorisent l'autonomie et la reproductibilité des traitements.

2.1 Limites des architectures informatiques traditionnelles

Afin de comprendre la prédominance des technologies *cloud* dans les infrastructures actuelles de traitement de la donnée, il est nécessaire de rappeler l'historique — sans prétendre à l'exhaustivité

— des architectures développées au cours des dernières décennies afin de traiter les données de plus en plus volumineuses générées par la numérisation des activités humaines. Historiquement, les données ont été stockées dans des bases de données, c'est à dire des systèmes de stockage et d'organisation de la donnée. Ces objets ont vu le jour dans les années 1950, et ont connu un essor particulier avec les bases de données relationnelles dans les années 1980. Cette technologie se révélant particulièrement pertinente pour organiser le stockage des données « métier » des entreprises, elle a été à la base des *data warehouses*, qui ont longtemps constitué la référence des infrastructures de stockage de la donnée. Si leur implémentation technique peut être de nature variée, leur principe est simple : des données de sources multiples et hétérogènes sont intégrées dans un système de bases de données relationnel selon des règles métier grâce à des processus dits *ETL* (*extract-transform-load*), afin de les rendre directement accessibles pour une variété d'usages (analyse statistique, *reporting*, etc.) à l'aide d'un langage normalisé : SQL [16].

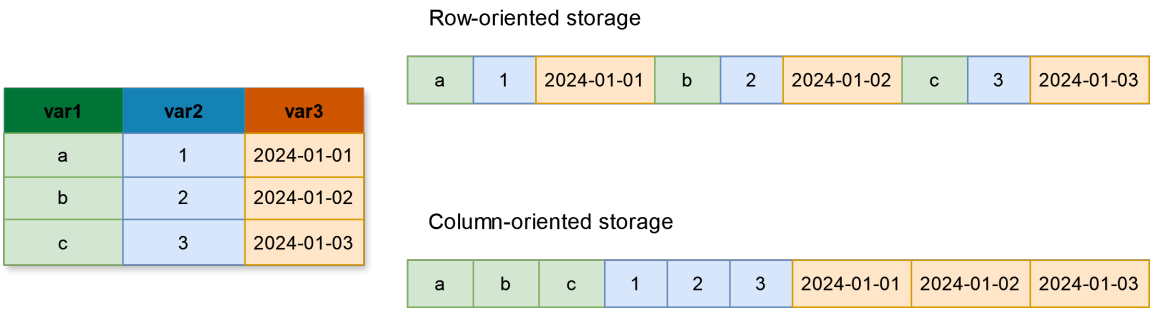
Au début des années 2000, la montée en puissance des usages de nature *big data* dans les entreprises met en lumière les limites des *data warehouses* traditionnels. D'une part, les données traités présentent une diversité croissante de formats (structurés, semi-structurés et non structurés), et cette réalité rentre difficilement dans le monde ordonné des *data warehouses*, qui nécessite de spécifier *a priori* le schéma des données. Pour pallier ces limites, de nouvelles infrastructures de stockage vont être développées : les *data lakes*, qui permettent la collecte et le stockage de quantités massives de données de nature diverse. D'autre part, la taille considérable de ces données rend de plus en plus difficile leur exploitation sur une unique machine. C'est dans ce contexte que Google publie le paradigme MapReduce [17], [18], posant les bases d'une nouvelle génération de systèmes permettant de traiter de larges volumes de données de manière distribuée. Dans les infrastructures traditionnelles, le passage à l'échelle était réalisé selon un principe de scalabilité verticale, c'est à dire en augmentant la puissance d'une machine de calcul ou en choisissant une machine plus performante. Cette approche devient néanmoins rapidement très coûteuse et se heurte aux limites physiques des composants. A l'inverse, les architectures distribuées adoptent le principe de scalabité horizontale : en installant des serveurs — chacun d'une puissance limitée — en parallèle et en adaptant les algorithmes à cette logique distribuée, on parvient à traiter des données massives avec du matériel standard. Dans la lignée de ces travaux, émerge l'écosystème Hadoop qui offre une combinaison de technologies complémentaires : un *data lake* (HDFS - *Hadoop Distributed File System*), un moteur de calcul distribué (MapReduce) et des outils d'intégration et de transformation de la donnée. Cet éco-système est progressivement complété par des outils qui vont démocratiser la capacité à traiter des données *big data* : Hive, qui convertit des requêtes SQL en traitements MapReduce distribués, puis Spark, qui lève certaines limites techniques de MapReduce et fournit des API dans plusieurs langages (Java, Scala, Python, etc.). Le succès de l'éco-système Hadoop dans les entreprises est considérable dans la mesure où il permet de traiter des volumes de données sans précédent — jusqu'au péta-octet — et des vélocités considérables — jusqu'au temps réel — à l'aide de langages de programmation non réservés aux seuls informaticiens.

À la fin des années 2010, les architectures basées sur Hadoop connaissent néanmoins un net déclin de popularité. Dans les environnements Hadoop traditionnels, le stockage et le calcul sont co-localisés par construction : si les données à traiter sont réparties sur plusieurs serveurs, chaque

section des données est directement traitée sur la machine hébergeant cette section, afin d'éviter les transferts réseau entre serveurs. Dans ce paradigme, la mise à l'échelle de l'architecture implique une augmentation linéaire à la fois des capacités de calcul et de stockage, indépendamment de la demande réelle. Dans un article volontairement provocateur et intitulé « *Big Data is Dead* » [19], Jordan Tigani, l'un des ingénieurs fondateurs de Google BigQuery, explique pourquoi ce modèle ne correspond plus à la réalité de la plupart des organisations exploitant intensivement de la donnée. Premièrement, parce que « dans la pratique, la taille des données augmente beaucoup plus rapidement que les besoins en calcul ». Même si la quantité de données générées et nécessitant donc d'être stockées croît de manière rapide au fil du temps, il n'est généralement pas nécessaire d'interroger l'ensemble des données stockés mais seulement les portions les plus récentes, ou seulement certaines colonnes et/ou groupes de lignes. Par ailleurs, Tigani souligne que « la frontière du *big data* ne cesse de reculer » : les avancées dans les capacités des serveurs et la baisse des coûts du matériel signifient que le nombre de charges de travail ne tenant pas sur une seule machine — une définition simple mais efficace du *big data* — a diminué de manière continue. En conséquence, en séparant correctement les fonctions de stockage et de calcul, même les traitements de données substantiels peuvent finir par utiliser « beaucoup moins de calcul que prévu [...] et pourraient même ne pas avoir besoin d'un traitement distribué du tout ». Ces enseignements plaident donc de manière générale pour le choix d'infrastructures dans lesquelles ressources de calcul et de stockage sont le plus faiblement couplées possibles.

Cette tendance est par ailleurs renforcée par certaines avancées technologiques récentes issues de l'éco-système de la donnée qui permettent de traiter des volumes de plus en plus importants de données sur des machines standards en utilisant des langages de programmation usuels comme R ou Python. Une première innovation concerne l'apparition de formats de stockage beaucoup plus efficients que les formats traditionnels, comme Apache Parquet [20]. Ce format de stockage orienté-colonne (voir Figure 1) est notamment optimisé pour les analyses *WORM* (*write once, read many*). Il est ainsi particulièrement adapté aux traitements statistiques, où les données sont généralement figées après la collecte, mais les analyses ultérieures à partir de ces données potentiellement nombreuses. De plus, la possibilité de partitionner ces données selon une variable de filtrage fréquente, comme la région ou le département par exemple, permet d'optimiser les traitements — à la manière d'un index dans une base SQL — et d'accroître encore leur efficience. Ces différentes propriétés rendent le format Parquet très pertinent pour les traitements analytiques qui caractérisent la production de statistique publique [21], [22].

Figure 1. – Représentation orientée ligne et orientée colonne d'un même jeu de données.



Note: De nombreuses opérations statistiques sont analytiques (dites OLAP - *Online Analytical Processing*) par nature : elles impliquent la sélection de colonnes spécifiques, le calcul de nouvelles variables, la réalisation d'agrégations basées sur des groupes, etc. Le stockage orienté ligne - comme dans un fichier CSV - n'est pas adapté à ces opérations analytiques, car il nécessite de charger l'ensemble du jeu de données en mémoire afin d'effectuer une requête. À l'inverse, le stockage orienté colonne permet de ne lire que les colonnes de données pertinentes, ce qui réduit considérablement les temps de lecture et de traitement pour ces charges de travail analytiques. En pratique, les formats colonnes populaires tels que Parquet utilisent une représentation hybride : ils sont principalement orientés colonne, mais intègrent également un regroupement astucieux basé sur les lignes pour optimiser les requêtes de filtrage.

Le format Parquet rend le stockage de données au format tabulaire beaucoup plus efficient. Mais pour pleinement bénéficier de cette structure de données, il est également nécessaire de s'intéresser à l'étape suivante : le traitement des données en mémoire. Deux outils majeurs ont émergé à cette fin au cours des dernières années. Le premier est Apache Arrow, un format tabulaire de données en mémoire interopérable entre de nombreux langages (Python, R, Java, etc.). Le second est DuckDB, un système de base de données portable et également interopérable. Ces deux outils, bien que techniquement très différents en termes d'implémentation, présentent des avantages et des gains de performance semblables. D'abord, ils sont tous deux orientés-colonne (voir Figure 1) et travaillent ainsi en synergie avec le format Parquet, dans la mesure où ils font persister les bénéfices de ce format de stockage dans la mémoire. Par ailleurs, ils permettent tout deux d'augmenter considérablement les performances des requêtes sur les données grâce à l'utilisation de la *lazy evaluation* (« évaluation paresseuse »). Là où les opérations sur des données sont généralement exécutées de manière linéaire par les langages de programmation — par exemple, sélectionner des colonnes et/ou filtrer des lignes, puis calculer de nouvelles colonnes, puis effectuer des agrégations, etc. — Arrow comme DuckDB exécutent quant à eux ces dernières selon un plan d'exécution pré-calculé qui optimise de manière globale la chaîne de traitements. Dans ce paradigme, les calculs sont non seulement beaucoup plus performants, mais également beaucoup plus efficients dans la mesure où ils n'impliquent de récupérer que les données effectivement nécessaires pour les traitements demandés. Ces innovations permettent ainsi d'envisager des traitements basés sur des données dont le volume total dépasse la mémoire RAM effectivement disponible sur une machine.

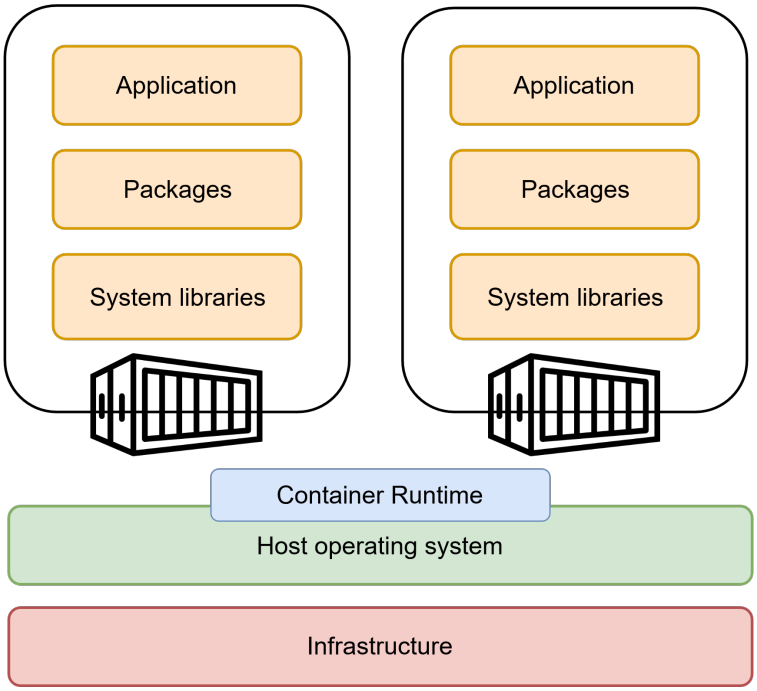
2.2 L'apport des technologies *cloud*

Dans la lignée des observations de Tigani, on observe ces dernières années une transition marquée des organisations vers des architectures plus flexibles et faiblement couplées. L'avènement des technologies *cloud* a joué un rôle déterminant dans cette transition, et ce pour plusieurs raisons. D'abord, une raison technique : par rapport à l'époque où Hadoop constituait l'infrastructure *big data* de référence, la latence des flux réseaux est devenue une préoccupation bien moindre, rendant le modèle de co-localisation du stockage et des ressources de calcul sur de mêmes machines moins pertinent. Ensuite, une raison liée aux usages : si le volume des données générées continue de croître, c'est surtout la diversification des données exploitées qui marque l'évolution récente de l'éco-système. Les infrastructures modernes doivent doivent non seulement être capables de

traiter de grands volumes, mais aussi être adaptables sur de multiples dimensions. Elles doivent pouvoir prendre en charge diverses structures de données (allant des formats structurés et tabulaires aux formats non structurés comme le texte, les images, le son et la vidéo) et permettre une large gamme de techniques computationnelles, du calcul parallèle aux modèles d'apprentissage profond qui nécessitent des GPU, ainsi que le déploiement et la gestion d'applications [23]. Ces dernières années, deux technologies intimement liée au *cloud* — justifiant leur qualificatif de technologies *cloud-native* — ont émergé comme des solutions essentielles pour atteindre ce besoin d'environnements de calcul plus flexibles : la conteneurisation et le stockage objet.

Dans un environnement *cloud*, l'ordinateur de l'utilisateur devient un simple point d'accès pour effectuer des calculs sur une infrastructure centralisée. Cela permet à la fois un accès universel aux ressources et un passage à l'échelle des services, car il est plus facile de faire passer à l'échelle une infrastructure centrale — là encore de manière horizontale, c'est-à-dire en ajoutant davantage de serveurs. Cependant, ces infrastructures centralisées présentent deux limites importantes : la concurrence entre utilisateurs pour l'accès aux ressources physiques et la nécessité d'isoler correctement les applications déployées. Le choix de la conteneurisation est fondamental, car il répond à ces deux enjeux [24]. En créant des « bulles » spécifiques à chaque service, les conteneurs garantissent l'herméticité des applications tout en restant légers, puisqu'ils partagent le système d'exploitation avec la machine hôte (voir Figure 2). Initialement développés dans le cadre du noyau Linux, les conteneurs ont fortement gagné en popularité au début des années 2010 grâce au moteur Docker qui a permis la démocratisation de leur utilisation - à tel point que l'on emploie parfois le terme « *dockerisation* » pour désigner la *conteneurisation*.

Figure 2. – Architecture d'un environnement conteneurisé.



Note: Un conteneur est un groupe logique de ressources permettant d'encapsuler une application (par exemple, un *batch* R), les bibliothèques utilisées (ggplot, dplyr, etc.) et les librairies système nécessaires (l'interpréteur R, d'autres bibliothèques dépendantes du système d'exploitation, etc.) dans un contenant auto-suffisant. Les applications conteneurisées sont isolées les unes des autres grâce à la virtualisation, qui permet d'attribuer des ressources physiques spécifiques à chaque application tout en garantissant leur herméticité. Contrairement aux machines virtuelles (VM), qui virtualisent également le système d'exploitation (OS), les conteneurs s'appuient sur une forme de virtualisation légère : le conteneur partage l'OS de l'infrastructure hôte via le moteur de conteneurisation, dont le plus connu est Docker. En conséquence, les conteneurs sont beaucoup plus portables que les VM et peuvent être déployés et redistribués facilement.

Pour gérer plusieurs applications conteneurisées de manière systématique, les infrastructures conteneurisées s'appuient généralement sur un logiciel orchestrateur — le plus utilisé étant Kubernetes, un projet open source initialement développé par Google pour gérer ses nombreuses charges de travail conteneurisées en production [25]. Les orchestrateurs automatisent le processus de déploiement, de mise à l'échelle et de gestion des applications conteneurisées, coordonnant leur exécution sur différents serveurs. Cette propriété permet notamment de traiter de très grands volumes de données de manière distribuée : les conteneurs décomposent les opérations de traitement des données massives en une multitude de petites tâches, organisées par l'orchestrateur. Cela minimise les ressources requises tout en offrant une flexibilité supérieure aux architectures basées sur Hadoop [26]. Cette transition s'est d'ailleurs observée de manière très empirique à l'Insee dans le cadre du projet d'exploitation des données de caisse. Face aux difficultés computationnelles que posaient l'utilisation de ces données dans le calcul de l'IPC, un cluster Hadoop a d'abord été mis en place comme alternative à l'architecture existante. Une accélération des opérations de traitement des données pouvant aller jusqu'à un facteur 10 avait été observée, pour des opérations qui prenaient auparavant plusieurs heures [27]. Malgré cette amélioration des performances, ce type d'architectures n'a pas été pérennisée, principalement du fait de leur complexité et donc de leur coût de maintenance. La solution finalement retenue a été de mettre en place un environnement Spark sur Kubernetes, permettant de bénéficier des gains de performance liés à la distribution des calculs tout en bénéficiant de la flexibilité des infrastructures *cloud*.

L'autre choix fondamental dans une architecture *cloud* concerne le mode de stockage des données. Les conteneurs étant par construction sans état (*stateless*), il est nécessaire de prévoir une couche de persistence pour stocker à la fois les données brutes en entrée des traitements et les données transformées en sortie de ces derniers. Dans l'écosystème des infrastructures de données conteneurisées, le stockage dit « objet » s'est progressivement imposé comme référence, largement popularisée par l'implémentation S3 (*Amazon Simple Storage Service*) d'Amazon [28]. Afin de comprendre cette prédominance, il est utile de comparer ce mode de stockage aux autres modes existants.

Schématiquement, on peut distinguer trois grandes approches en matière de stockage : le stockage de fichiers (*filesystem*), le stockage par bloc (*block storage*) et le stockage d'objets (*object storage*). Le stockage de fichiers est le plus intuitif : les données sont organisées sous forme d'une structure hiérarchique de répertoires et de fichiers — comme sur un ordinateur personnel. Facile

d'utilisation et adapté aux environnements traditionnels, ce mode de stockage passe difficilement à l'échelle et requiert des interventions manuelles pour monter et gérer les accès aux fichiers, ce qui restreint l'autonomie des utilisateurs et n'est pas adapté aux environnements de traitement éphémères comme les conteneurs. Le stockage par bloc propose un accès de bas niveau aux données sous forme de blocs contigus — à l'image du stockage sur un disque dur — garantissant des performances élevées et une faible latence. Il s'avère donc très pertinent pour des applications qui exigent un accès rapide aux données stockées, comme une base de données. En revanche, il passe là encore difficilement à l'échelle du fait du coût de la technologie et de la difficulté à faire croître horizontalement ce type de stockage. Enfin, le stockage objet divise quant à lui les fichiers de données en morceaux appelés « objets » qui sont ensuite stockés dans un référentiel unique, qui peut être distribué sur plusieurs machines. Chaque objet se voit attribuer un certain nombre de métadonnées (nom de l'objet, taille, date de création, etc.) dont un identifiant unique qui permet au système de retrouver l'objet sans la nécessité d'une structure hiérarchique comme celle d'un *filesystem*, ce qui réduit drastiquement le coût du stockage.

Les différentes propriétés du stockage objet le rendent particulièrement pertinent pour construire une infrastructure conteneurisée pour la *data science*. D'abord, il est optimisé pour la scalabilité : les objets stockés ne sont pas limités en taille et la technologie sous-jacente permet un stockage efficient de fichiers potentiellement très volumineux, si besoin en les distribuant horizontalement. Ensuite, il est source d'autonomie pour les utilisateurs : en stockant les données sous forme d'objets enrichis de métadonnées et accessibles via des API REST standardisées, il permet aux utilisateurs d'interagir directement avec le stockage via leur code applicatif (en R, Python, etc.) tout en offrant une gestion très fine des permissions — jusqu'aux droits sur un fichier — vie des jetons d'accès, garantissant ainsi une traçabilité accrue des opérations effectuées. Enfin, le stockage objet joue un rôle clé dans l'objectif de construction d'une infrastructure découplée comme celle évoquée précédemment. Dans la mesure où les dépôts de données — appelés *« buckets »* — sont interrogeables via des requêtes HTTP standards, les environnements de calcul peuvent importer par le biais du réseau les données nécessaires aux traitements réalisés. Ainsi, les ressources de stockage et de calcul n'ont plus besoin d'être sur les mêmes machines ni même nécessairement dans le même lieu, et peuvent ainsi évoluer indépendamment en fonction des besoins spécifiques de l'organisation.

2.3 Les technologies *cloud* favorisent l'autonomie et la reproductibilité

Comprendre comment les choix technologiques décrits dans la discussion technique ci-dessus se révèlent pertinents dans le contexte des statistiques publiques nécessite un examen critique de l'évolution des environnements informatiques mis à disposition des statisticiens à l'Insee.

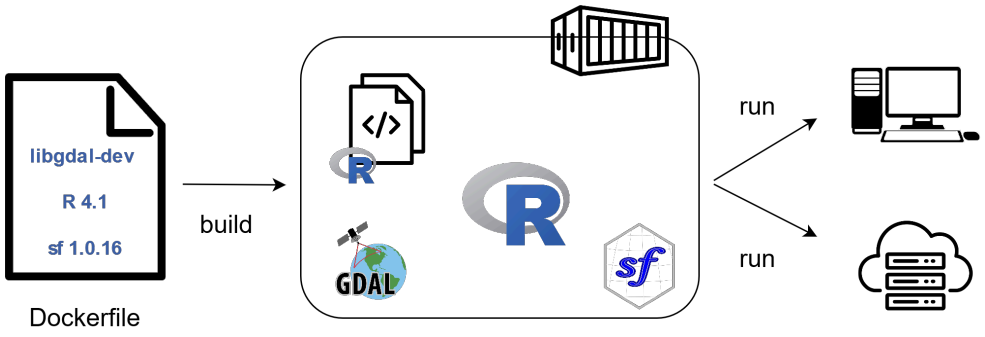
À la fin des années 2000, alors que la micro-informatique est à son apogée, une grande partie des ressources techniques utilisées par les statisticiens sont locales : le code et les logiciels de traitement sont situés sur des ordinateurs personnels, tandis que les données sont accessibles via un système de partage de fichiers. En raison de la puissance limitée des ordinateurs personnels, cette configuration restreignait considérablement la capacité des statisticiens à expérimenter avec des sources *big data* ou des méthodes statistiques intensives en calculs. Par ailleurs, cet état de fait impliquait des risques de sécurité liés à la dissemination des données confidentielles au sein

de l'organisation. Afin de pallier ces limites, une transition progressive est opérée vers des infrastructures centralisées avec le projet d'Architecture Unifiée Statistique (AUS). Ce centre de calcul propose une forme de transition entre une informatique locale et une informatique centralisée : elle concentre toutes les ressources sur des serveurs centraux, mais recrée l'expérience du poste de travail via un accès par bureau distant au centre de calcul. La centralisation des ressources de calcul et la technologie des coffres sécurisés permettent aux statisticiens d'effectuer des traitements en *self* sur des données à la fois volumineuses et confidentielles qui étaient jusqu'alors impossibles. A ce jour, AUS — dans sa troisième version — reste l'infrastructure de référence pour les traitements réalisés « en *self* » : les environnements disponibles couvrent une majorité des besoins des statisticiens et contribuent de manière essentielle à la production statistique de l'institut.

Bien que l'infrastructure informatique actuelle soutienne adéquatement les activités fondamentales de production statistique, elle restreint de manière notable la capacité des statisticiens à expérimenter librement et à innover. Le principal goulot d'étranglement dans cette organisation réside dans la dépendance des projets statistiques à la prise de décision centralisée en matière d'informatique, notamment en ce qui concerne l'allocation des ressources de calcul, l'accès au stockage partagé, l'utilisation de langages de programmation préconfigurés etc. En outre, ces dépendances conduisent souvent à un phénomène bien connu dans la communauté du développement logiciel, où les priorités des développeurs — itérer rapidement pour améliorer continuellement les fonctionnalités — entrent souvent en conflit avec l'objectif des équipes informatiques de garantir la sécurité et la stabilité des processus. À l'inverse, nous comprenons que les pratiques modernes en *data science* reflètent une implication accrue des statisticiens dans le développement et l'orchestration informatique de leurs opérations de traitement de données, au-delà de la simple phase de conception ou de validation. Les nouvelles infrastructures de *data science* doivent donc prendre en compte ce rôle élargi de leurs utilisateurs, en leur offrant plus d'autonomie que les infrastructures traditionnelles.

Nous soutenons que les technologies *cloud* sont une solution puissante pour offrir aux statisticiens une autonomie bien plus grande dans leur travail quotidien, favorisant ainsi une culture de l'innovation. Grâce au stockage objet, les utilisateurs obtiennent un contrôle direct sur la couche de stockage, leur permettant d'expérimenter avec des sources de données diverses sans être limités par les espaces de stockage souvent restreints et alloués par les départements informatiques. La conteneurisation permet aux utilisateurs de personnaliser leurs environnements de travail selon leurs besoins spécifiques — qu'il s'agisse de langages de programmation, de bibliothèques système ou de versions de packages — tout en leur offrant la flexibilité nécessaire pour adapter leurs applications à la puissance de calcul et aux capacités de stockage requises. Par construction, les conteneurs favorisent également le développement d'applications portables, permettant des transitions plus fluides entre les environnements de *self* et de production (Figure 2). Enfin, avec des outils d'orchestration tels que Kubernetes, les statisticiens peuvent déployer des applications interactives et des API comme preuves de concept, tout en automatisant le processus de construction. Cette capacité s'aligne avec l'approche DevOps, qui préconise la création de preuves de concept de manière itérative, plutôt que de chercher à développer la solution optimale (mais chronophage) pour un objectif préalablement défini [29].

Figure 3. – Construction d'une image et déploiement d'un conteneur



Note: Dans un environnement conteneurisé, les applications sont créées à partir de spécifications écrites dans des manifestes — un paradigme connu sous le nom d'*"infrastructure as code"*. Dans un *script*, conventionnellement nommé *Dockerfile*, les *data scientists* spécifient l'environnement de travail de leur application : le code de l'application, les logiciels à inclure (par exemple, R), les packages utilisés pour leurs opérations de traitement (par exemple, le package sf pour le calcul géospatial), ainsi que les librairies système dépendant de l'OS appelées par ces packages (par exemple GDAL, une bibliothèque standard pour traiter les formats utilisés par la majorité des packages traitant des données géospatiales). En particulier, les versions de ces différentes dépendances de l'application peuvent être précisément spécifiées, ce qui garantit la reproductibilité des calculs effectués. Une fois le *Dockerfile* correctement spécifié, une étape de construction (*build*) génère une image, c'est-à-dire une forme empaquetée et compressée de l'environnement qui permet de lancer l'application à l'identique. Les images créées de cette manière sont portables : elles peuvent être facilement distribuées — via un registre d'images, comme celui de GitLab à l'Insee — et exécutées de manière reproductible sur n'importe quelle infrastructure disposant d'un moteur de conteneurisation. L'exécution de l'image donne naissance à un conteneur, c'est à dire une instance vivante et déployée de l'application contenue dans l'image.

Outre le passage à l'échelle et l'autonomie, ces choix architecturaux favorisent également la reproductibilité des calculs statistiques. Le concept de reproductibilité — à savoir la capacité de reproduire le résultat d'une expérience en appliquant la même méthodologie aux mêmes données — est un critère fondamental de validité scientifique [30]. Il est également essentiel dans le domaine des statistiques publiques, car il constitue un facteur de transparence, essentielle pour établir et maintenir la confiance du public [31]. Favoriser la reproductibilité dans la production statistique implique de concevoir des solutions de traitement capables de produire des statistiques reproductibles, tout en étant partageables entre pairs [32]. Les infrastructures informatiques traditionnelles — qu'il s'agisse d'un ordinateur personnel ou d'une infrastructure partagée avec un accès à distance — sont limitées à cet égard. Construire un projet ou calculer un indicateur statistique dans ces environnements implique généralement une série d'étapes manuelles (installation des bibliothèques système, des binaires du langage de programmation, des packages du projet, gestion des versions conflictuelles, etc.) qui ne peuvent pas être pleinement reproduites du fait de la persistence de l'environnement sous-jacent. En comparaison, les conteneurs sont reproductibles par construction : leur processus de construction implique de définir précisément toutes

les ressources nécessaires sous la forme d'un ensemble d'opérations standardisées, allant de la machine quasi-nue à l'application en cours d'exécution [33]. De plus, ces environnements reproductibles peuvent être facilement partagés avec des pairs en les publiant sur des registres ouverts (par exemple, un registre de conteneurs comme DockerHub ou celui de GitLab) en plus du code source de l'application (par exemple, sur une forge logicielle comme GitHub ou GitLab). Cette approche améliore considérablement la réutilisation des projets de données, favorisant un modèle de développement et d'innovation basé sur la collaboration communautaire.

3 Un projet *open source* pour faciliter l'adoption des technologies *cloud*

Notre revue de la littérature et de l'évolution de l'éco-système de la donnée mettent en évidence les technologies *cloud*, en particulier la conteneurisation et le stockage objet, comme des éléments clés pour construire une plateforme de *data science* à la fois scalable et flexible. Néanmoins, les arguments qui justifient d'investir dans ce type d'infrastructure en tant qu'organisation ne suffisent pas à eux seuls à garantir leur adoption dans les pratiques. Cette section revient sur la genèse et le développement d'Onyxia, un projet développé à l'Insee qui vise à démocratiser l'accès aux technologies *cloud* en fournissant aux statisticiens des environnements de *data science* prêts à l'emploi qui favorisent l'expérimentation. Enfin, nous montrons comment les principes fondamentaux qui sous-tendent le projet Onyxia — innovation ouverte, licence *open-source* absence d'enfermement propriétaire — s'inscrit dans une volonté de construire des communs numériques (« *commons* ») facilement réutilisables par les organisations. Ces principes ont permis le développement du projet à la fois à l'Insee et en dehors de l'Insee avec plusieurs instances en production et de nombreuses autres à l'essai.

3.1 Rendre les technologies *cloud* accessibles aux statisticiens

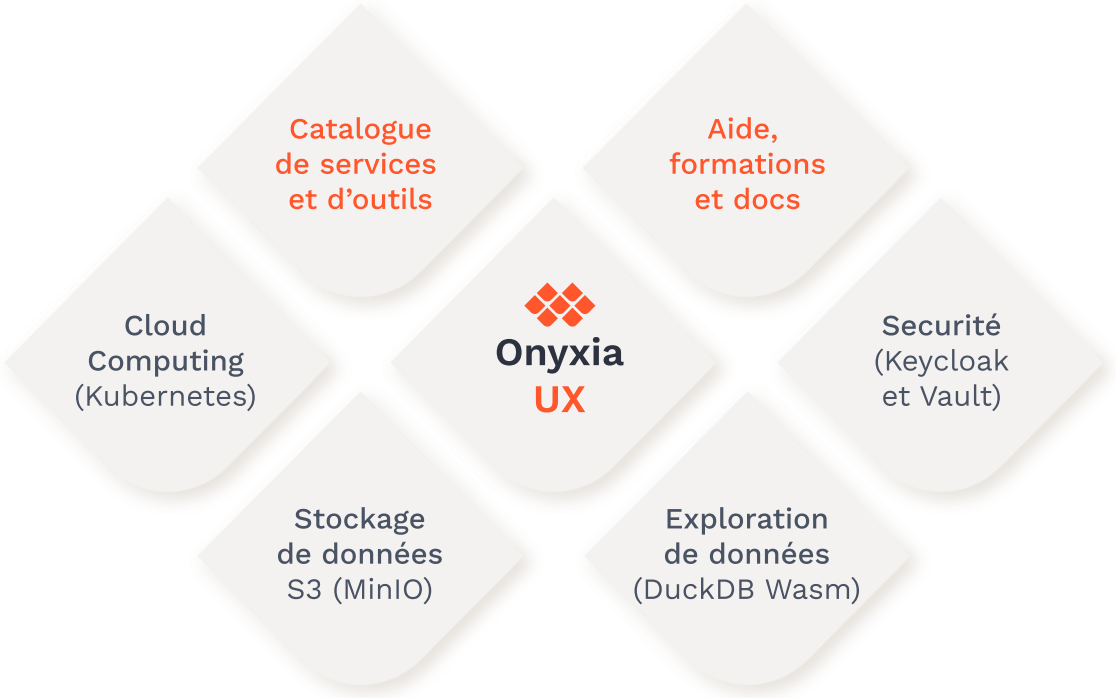
En marge de la conférence NTTS organisée en 2017 par Eurostat, un *hackathon* « *big data* » propose aux équipes de différents INS d'exploiter des offres d'emploi en ligne pour réfléchir aux problèmes d'adéquation entre l'offre et la demande de compétences sur le marché du travail à un niveau régional. Une équipe regroupant des agents de la DMCSI et de la DSI se constitue pour appréhender ces données volumineuses d'un type nouveau, anticipant l'insuffisance de la plateforme AUSv1, alors infrastructure de référence pour les traitements réalisés en *self* à l'Insee. Cette composition mixte de l'équipe participante permet de mettre en lumière un certain paradoxe dans la mise à disposition des ressources de calcul à l'Insee. Là où les *data scientists* de l'équipe ne demandent qu'à avoir accès à un environnement de calcul leur fournissant à la fois un minimum de ressources de calcul et leur permettant d'installer de nouveaux outils librement pour pouvoir mettre en place leurs idées, les informaticiens disposent quant à eux de l'ensemble de la puissance du *data center* mais n'en utilisent qu'une fraction en pratique. Au-delà de la belle réussite finale — l'équipe Insee obtient la deuxième place — ce hackathon illustre l'intrication croissante des sujets méthodologiques et informatiques dans le paradigme de la *data science*, et amène la DSI à s'interroger quant à la manière de redonner de l'autonomie aux statisticiens dont le métier évolue vers des pratiques plus coûteuses en ressources. Ce petit groupe de pirates [34] s'est alors saisi d'une double opportunité : une nouvelle technologie de rupture — les technologies *cloud* — parfaitement

adaptée au enjeux de la *data science* et des serveurs décommissionnés à la Direction Générale. Cette initiative aboutit à la mise en place d'un premier cluster Kubernetes dans les locaux de l'Insee en 2020.

Cependant, les premières expérimentations révèlent un obstacle majeur à l'adoption généralisée des technologies *cloud* : la complexité de leur intégration. Les infrastructures *cloud* sont par nature modulaires, faites de différents composants faiblement couplés. C'est précisément cette modularité qui favorise le passage à l'échelle et l'évolutivité de ces architectures. Cette modularité a néanmoins une contrepartie importante : la nécessité de configurer finement ces différents composants pour leur permettre de communiquer entre eux. Ainsi, un simple service RStudio lancé sur le *cluster* ne suffit pas en soi : il doit pouvoir communiquer avec la couche de stockage des données, le service de gestion des secrets, d'autres services éventuels permettant par exemple l'ordonnancement des traitements, etc. Ces configurations sont complexes et nombreuses et ne sauraient être demandées aux statisticiens désireux de bénéficier des avantages des infrastructures *cloud* dans le cadre de leurs traitements.

Ce constat est capital : choisir des technologies qui favorisent l'autonomie ne suffit pas à atteindre cet objectif si leur complexité constitue une barrière trop importante à leur adoption dans l'organisation. Ces dernières années, les statisticiens de l'Insee ont déjà dû s'adapter à un environnement en forte évolution en ce qui concerne leurs outils quotidiens : passer de logiciels propriétaires (SAS) à des outils open source (R, Python), s'approprier des technologies qui améliorent la reproductibilité (contrôle de version avec Git), consommer voire développer des API, etc. Ces changements, qui tendent à rapprocher la nature de leur travail de celui des développeurs informatiques, impliquent déjà des efforts considérables en termes de formation et des modifications substantielles des pratiques. Dans ce contexte, l'adoption des technologies *cloud* dans le cadre de la modernisation du *self* et, par là, l'opportunité pour l'organisation d'en tirer les bénéfices attendus, dépend fortement de notre capacité à les rendre accessibles.

Figure 4. – Onyxia agit comme un « liant » technique entre des composants *cloud native*



C'est dans ce contexte que s'inscrit le développement du projet Onyxia, une application légère qui agit essentiellement comme une interface entre les composants modulaires qui composent l'architecture (voir Figure 4). Le point d'entrée principal pour l'utilisateur est une application web ergonomique[2] qui permet de lancer des services à partir d'un catalogue de *data science* (voir Chapitre 3.3)[3]. Ces services sont alors immédiatement déployés sous forme de conteneurs sur un *cluster* Kubernetes sous-jacent. Le lien entre l'interface utilisateur (UI) et Kubernetes est assurée par une API[4], dont le rôle est de transformer la demande de lancement de service de l'utilisateur en un ensemble de manifestes nécessaires pour déployer les ressources Kubernetes nécessaires à ce service. Pour une application donnée, ces ressources sont regroupées sous la forme d'un *chart* Helm[5], une librairie largement utilisée pour empaqueter des applications potentiellement complexes sur Kubernetes [35] et gérer le cycle de vie de ces objets vivants.

Cette capacité à mettre à disposition des services prêts à l'emploi et ainsi d'abstraire à l'utilisateur la complexité des technologies *cloud* sous-jacentes est véritablement la valeur ajoutée du projet Onyxia. Bien que les utilisateurs puissent configurer un service pour l'adapter à leurs besoins, la plupart du temps, ils se contentent de lancer un service prêt à l'emploi avec des paramètres par défaut et commencent à développer immédiatement. En injectant automatiquement les informations d'authentification et de configuration dans les conteneurs lors de leur initialisation,

[2] https://github.com/InseeFrLab/onyxia
[3] https://github.com/InseeFrLab/images-datascience
[4] https://github.com/InseeFrLab/onyxia-api
[5] https://github.com/InseeFrLab/helm-charts-interactive-services

Onyxia permet aux utilisateurs d'interagir sans difficulté avec les données de leur *bucket* sur MinIO, leurs informations sensibles (jetons, mots de passe) contenues dans un outil de gestion des secrets tel que Vault, etc. Cette injection automatique, associée à la pré-configuration des environnements de *data science* mis à disposition dans le catalogue d'images pour couvrir la plupart des usages courants de *data science*, permet aux utilisateurs d'exécuter des applications potentiellement complexes — comme des calculs distribués avec Spark sur Kubernetes à l'aide de données volumineuses stockées sur MinIO, ou encore l'entraînement de modèles d'apprentissage profond nécessitant un GPU — sans se heurter aux difficultés techniques liées à la configuration des composants nécessaires.

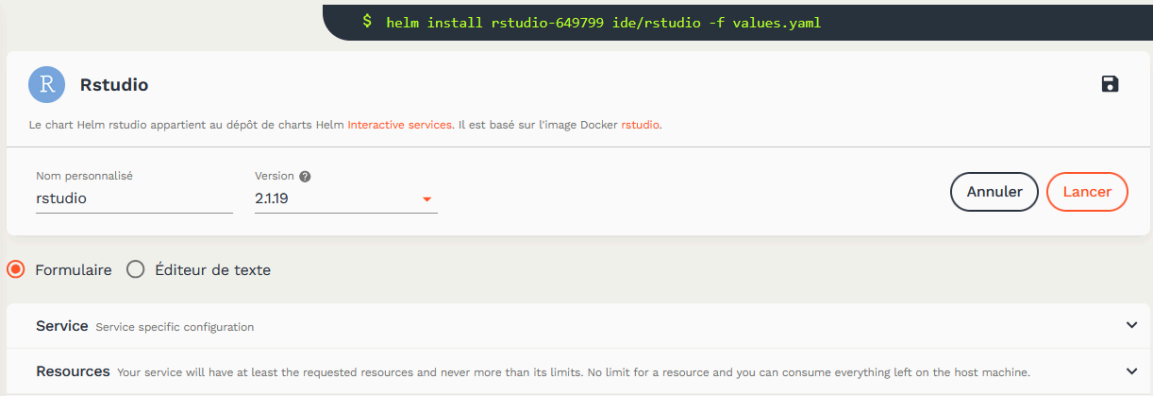
3.2 Des choix architecturaux visant à favoriser l'autonomie

Le projet Onyxia repose sur quelques principes structurants, avec un thème central : favoriser l'autonomie, à plusieurs niveaux. Ce principe s'applique d'abord niveau de l'organisation, en évitant l'enfermement propriétaire. Afin d'obtenir un avantage concurrentiel, une pratique courante que de nombreux fournisseurs de *cloud* commerciaux développent est d'imposer l'utilisation de certaines applications ou protocoles pour accéder aux ressources *cloud*. Souvent, ces dernières ne sont néanmoins pas interopérables : les scripts et pratiques qui fonctionnent avec un fournisseur ne marcheront pas à l'identique avec un autre, compliquant considérablement les migrations potentielles vers une autre plateforme *cloud* [36]. Face à ce constat, une tendance émerge vers l'adoption de stratégies dites « neutres » vis-à-vis des *clouds* afin de réduire la dépendance à des solutions spécifiques d'un seul fournisseur [37]. Dans cette optique, l'utilisation d'Onyxia est pensée de manière à être intrinsèquement non-enfermante : lorsqu'une organisation choisit de l'utiliser, elle choisit les technologies sous-jacentes — la conteneurisation et le stockage d'objets — mais pas la solution en elle-même. Le logiciel Onyxia peut être déployé sur n'importe quel cluster Kubernetes, qu'il soit *on-premise* ou issu d'une offre gérée de *clouds* commerciaux. De même, le choix de MinIO comme solution de stockage contribue à limiter l'enfermement propriétaire. En effet, MinIO est d'une part une solution de stockage objet *open source*, et d'autre part une solution basée sur l'API S3 d'Amazon, qui est progressivement devenu un standard de l'éco-système de la donnée. Ainsi, dans la mesure où les stockages proposés par les divers fournisseurs de *cloud* (AWS, GCP, etc.) s'assurent de leur compatibilité avec cette API, ces choix favorisent une position agnostique qui facilite toute migration ultérieure vers une solution *cloud* différente.

La volonté du projet Onyxia de favoriser l'autonomie s'illustre également au niveau du choix des services. Les logiciels propriétaires qui ont été intensivement utilisé dans les statistiques publiques et la recherche — comme SAS ou STATA — induisent également un phénomène d'enfermement propriétaire. Les coûts des licences, substantiels, peuvent évoluer rapidement et ce dans un contexte de marges de négociation réduites pour l'organisation si son système d'information dépend de manière sensible du code propriétaire. Par ailleurs, si ces logiciels ont l'avantage de mettre à disposition des utilisateurs des procédures statistiques faciles d'utilisation et stables dans la durée, elles contraignent dans le même temps à un ensemble de procédures proposées et maintenues par l'entreprise et limitent donc l'appropriation des nouveaux outils produits par l'écosystème. Par ailleurs, la nature fermée de leur code source empêche d'auditer certaines des procédures en question. A l'inverse, les choix réalisés dans le projet Onyxia visent à minimiser au

maximum l'effet d'enfermement des pratiques. D'abord, en n'incluant dans son catalogue que des services *open-source*, Onyxia promeut des logiciels dont il est possible d'auditer le code, ce qui favorise la transparence et la reproductibilité des statistiques produites. Si le catalogue des services offerts est par nature limité, le choix de ces derniers est également transparent : les services proposés doivent être standards, *open source* et s'intégrer avec Kubernetes ; le catalogue est par ailleurs lui-même ouvert à de nouvelles demandes ou des contributions. Enfin, et de manière capitale, Onyxia limite l'enfermement propriétaire en étant conçu de sorte à être amovible. L'objectif final est d'améliorer la familiarité et le confort des utilisateurs avec les technologies *cloud* sous-jacentes et les services standards de *data science*, de sorte à ce qu'ils puissent continuer à utiliser ces services si une autre infrastructure était adoptée. Un exemple illustratif de cette philosophie est l'approche de la plateforme concernant les actions des utilisateurs : pour les tâches effectuées via l'interface , comme le lancement d'un service ou la gestion des données, nous fournissons aux utilisateurs les commandes terminal équivalentes, promouvant ainsi une compréhension fine de ce qui se passe réellement lors de la réalisation d'une action (voir Figure 5).

Figure 5. – Lancer un service via l'interface web d'Onyxia.



Note: Les services du catalogue d'Onyxia peuvent être utilisés tels quels ("*out-of-the-box") ou configurés par les utilisateurs pour répondre à leurs besoins spécifiques. Afin de limiter la dépendance des utilisateurs vis-à-vis d'Onyxia, chaque action effectuée par l'utilisateur via l'interface utilisateur est accompagnée de la commande exacte exécutée sur le cluster Kubernetes sous-jacent.

Si le projet Onyxia a été initialement développé afin de permettre l'expérimentation avec des outils de *data science* à l'état de l'art, les environnements qu'il permet de mettre à disposition se veulent pour autant généralistes. Un objectif majeur du projet est de couvrir une large gamme de besoins exprimés par les statisticiens, de la production statistique courante réalisée en *self* aux usages les plus expérimentaux. Ainsi, cohabitent dans le catalogue de services des environnements de développement usuels — RStudio pour l'usage de R, Jupyter et VSCode pour l'usage de Python — et des environnements pour des usages avancés, mais déployés à travers les mêmes environnements interactifs. Par exemple, lancer des traitements distribués avec Spark en Python se fera également à travers l'usage de Jupyter ou VSCode ; les services de base peuvent ainsi servir de porte d'entrée à des usages plus avancés par la suite. Du fait de la diversité des besoins couverts, les bénéfices à

migrer seront différenciés selon les profils. Les utilisateurs dont les besoins actuels sont déjà bien couverts par les infrastructures de *self* existantes bénéficieraient essentiellement de la capacité à spécifier finement les ressources allouées à leur service et ainsi limiter les risques de rentrer en concurrence avec les processus d'autres utilisateurs en cas de saturation de la machine. Les bénéfices à migrer seront en revanche beaucoup plus marqués pour les utilisateurs souhaitant aller plus loin et développer de véritables prototypes d'applications pour leurs projets : configurer des scripts d'initialisation pour adapter les environnements à leurs besoins, déployer une application interactive (par exemple, en R Shiny) pour publier des visualisations de données dynamiques, ou encore déployer des services sur-mesure comme des bases de données à la demande ou bien des protoypes d'API. Pour permettre à ces utilisateurs avancés, souvent limités par la rigidité des environnements de calculs traditionnels, de continuer à faire progresser la structure via des usages innovants, Onyxia offre — selon la politique de sécurité de l'organisation — de larges possibilités de configuration des services et un accès étendu au cluster Kubernetes sous-jacent. Cela signifie que les utilisateurs peuvent ouvrir librement un terminal sur un service interactif et interagir avec le cluster — dans les limites de leur *namespace* — afin d'appliquer des ressources personnalisées et déployer des applications ou d'autres usages de leurs choix.

Au-delà de l'autonomie et de la scalabilité, les choix architecturaux d'Onyxia favorisent également la reproductibilité des calculs statistiques et la portabilité des applications. Dans le paradigme des conteneurs, l'utilisateur doit apprendre à gérer des ressources qui sont par nature éphémères, puisqu'elles n'existent qu'au moment de leur mobilisation effective dans le cadre d'un traitement. Cette non-persistance implique une séparation claire du code — hébergé sur une forge de code, comme GitLab ou GitHub — des données — stockées sur une solution de stockage spécifique, comme MinIO — et de la configuration et des secrets — passés au conteneur sous la forme de variables d'environnement. Au moment du lancement du traitement, le conteneur doit récupérer ces différents éléments (entrées), et produit par construction les mêmes sorties (production de données, modifications dans une base de données, visualisations, etc.) à partir d'un même jeu d'entrées. Cette séparation est un critère fondamental de qualité des projets de code dans la mesure où elle favorise à la fois la reproductibilité et de portabilité. Un conteneur étant nécessairement identifié par un *tag* (label), il est possible dans ce paradigme de versionner non plus seulement le code d'un projet, mais l'ensemble de l'environnement d'exécution qui garantit la reproduction des résultats obtenus à partir des mêmes données en entrée, ce qui constitue un gage de reproductibilité du chiffre statistique.

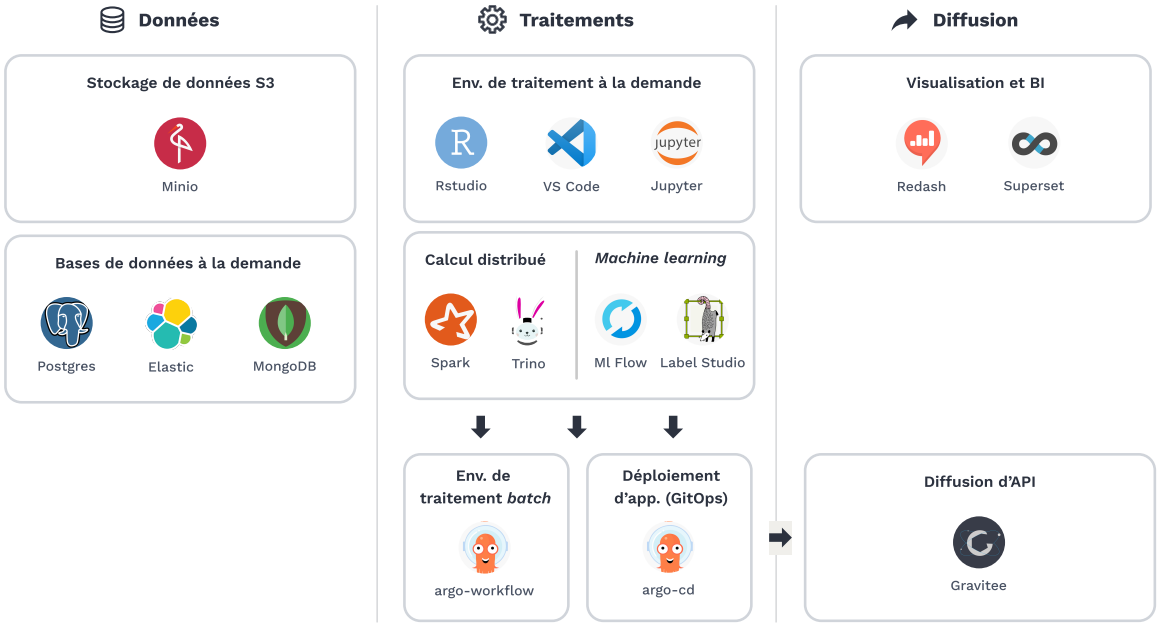
Le fait de développer dans des services conteneurisés favorise également la portabilité des projets, c'est à dire leur capacité à s'exécuter de manière homogène quelle que soit l'infrastructure de calcul sous-jacente. Ainsi, et sous réserve d'un accès homogène aux données d'entrée et aux ressources de calcul, un conteneur s'étant exécuté correctement sur une infrastructure de développement en *self* s'exécutera de la même manière sur une infrastructure de production. La conteneurisation permet donc fondamentalement de réduire l'écart existant entre les infrastructures de développement et les infrastructures de production, limitant ainsi les coûts de développement supplémentaires qui caractérisent souvent le passage en production des projets. Cette technologie peut également permettre d'envisager des modes de collaboration plus continus entre les équipes métiers et informatiques. Le fait de s'échanger des conteneurs, c'est à dire des environnements

dont on garantit le fonctionnement, plutôt que du code seul, rarement portable en soi, permet aux différentes parties prenantes de se spécialiser sur leur cœur de métier : la spécification, l'écriture de la logique métier, et le débogage éventuel côté équipe métier ; le développement applicatif, les tests fonctionnels et le *monitoring* côté informatique — limitant par là le besoin coûteux mais fréquent de fait de recoder des applications d'un langage statistique vers un langage informatique.

3.3 Un catalogue de services qui couvre le cycle de vie complet des projets de *data science*

L'intention du projet Onyxia était de fournir aux statisticiens un environnement complet conçu pour accompagner le développement de bout en bout des projets de *data science*. Comme illustré dans Figure 6, le catalogue par défaut propose une vaste gamme de services couvrant l'ensemble du cycle de vie d'un projet, du développement à la diffusion.

Figure 6. – Le catalogue d'Onyxia vise à couvrir l'ensemble du cycle de vie des projets de data science



L'utilisation principale de la plateforme est le déploiement d'environnements de développement interactifs (IDE), tels que RStudio, Jupyter ou VSCode. Ces IDE sont voulus comme étant « prêts à l'emploi » : ils sont équipés des dernières versions des principaux langages de programmation *open source* couramment utilisés par les statisticiens publics (R, Python, SQL, Julia) ainsi que des librairies les plus fréquemment utilisées pour chaque langage. Afin de garantir que les services restent à jour et cohérents entre eux, l'Insee maintient un dépôt d'images Docker sous-jacentes et les met à jour chaque semaine. Ces images sont entièrement *open source*[6] et peuvent donc être réutilisée en dehors du projet.

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

Comme décrit dans les sections précédentes, la couche de persistance de ces environnements interactifs est principalement assurée par MinIO, une solution de stockage objet *open source*. Cette solution étant basée sur une API REST standardisée, les fichiers peuvent être facilement interrogés depuis R ou Python à l'aide de librairies de haut niveau. Cela représente en soi une étape importante pour garantir la reproductibilité : les données ne sont pas sauvegardées localement, puis appelées via des chemins propres à une infrastructure ou un système de fichiers particulier. Au contraire, les requêtes aux fichiers sont spécifiées sous forme de requêtes HTTP standards, rendant la structure globale des projets plus évolutive. Les traitements rencontrés dans le service statistique public reposant très largement sur des fichiers de données, le paradigme du stockage objet répond très bien aux besoins de la plupart des projets que nous accompagnons sur ces infrastructures. Des services de bases de données supplémentaires, tels que PostgreSQL et MongoDB, sont également proposés pour les applications ayant des besoins spécifiques, notamment celles nécessitant de traitement géospatial en base (PostGIS) ou un stockage de données orienté documents.

Onyxia ayant été développé afin de permettre l'expérimentation avec des sources de données volumineuses, le catalogue propose également des services facilitant le passage à l'échelle. Pour les projets faisant intervenir des données volumineuses — de l'ordre de dizaines ou de centaines de millions de ligne — les différents services disposent nativement des librairies Arrow et DuckDB, permettant de traiter efficacement les données stockées au format Parquet en mémoire (cf. section XXX). Pour des usages plus conséquents, faisant intervenir des données massives, des logiciels comme Spark et Trino permettent d'effectuer des calculs distribués au sein d'un cluster Kubernetes via un simple lancement de service interactif ou d'un *batch* de traitement. Dans les deux cas, ces services sont préconfigurés pour s'intégrer naturellement avec le stockage S3, facilitant ainsi la création de *pipelines* de données à la fois efficients et intégrés de bout en bout. Enfin, différents services pré-configurés pour l'utilisation d'une GPU sont également proposés pour les projets basés sur des méthodes d'apprentissage automatique intensives en calcul. Le catalogue d'Onyxia fournit donc un point d'entrée unique pour mettre à disposition ces ressources rares ainsi que des librairies spécialisées — par exemple, les librairies de *deep learning* PyTorch et Tensorflow — avec la même simplicité que le lancement d'un service interactif de base.

Au-delà de la phase expérimentation, l'objectif est également de permettre aux statisticiens de produire des projets dits de « quasi-prod », au sens où ils préfigurent un passage en production afin d'en faciliter sa réalisation. Conformément aux principes de l'approche *DevOps*, cela implique de faciliter le déploiement de prototypes et leur amélioration continue au fil du temps. À cette fin, le catalogue d'Onyxia propose un ensemble de services *open source* visant à automatiser et industrialiser le processus de déploiement d'applications (ArgoCD), ordonnancer des traitements séquentiels et/ou parallèles (Argo-Workflows), ou encore déployer et gérer le cycle de vie des modèles d'apprentissage automatique (MLflow). La Section 4 illustre comment ces outils ont joué un rôle central dans la mise en production de premiers modèles d'apprentissage automatique à l'Insee, conformément aux principes du MLOps.

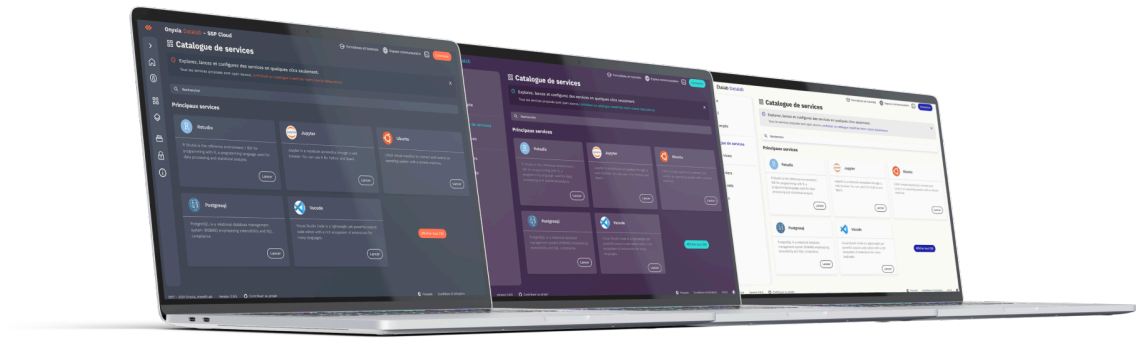
Dans la Chapitre 3.2, nous avons souligné qu'un des principes fondamentaux d'Onyxia était d'éviter l'enfermement propriétaire. Dans cette optique, les organisations qui instancient Onyxia

sont libres de personnaliser les catalogues pour répondre à leurs besoins spécifiques, ou même de créer leurs propres catalogues indépendamment de l'offre par défaut. Cette flexibilité garantit aux organisations le fait de ne pas être limité à une solution ou à un fournisseur unique, et ainsi de pouvoir adapter la plateforme à l'évolution de leurs besoins ou de migrer vers une autre solution *cloud* dans le futur si nécessaire.

3.4 Construire des communs numérique : un projet, plusieurs instances

En tant qu'initiative entièrement *open source*, le projet Onyxia vise à construire des « communs numériques » en promouvant et en développant des logiciels facilement réutilisables dans le service statistique publique et plus largement [38]. Cela concerne, tout d'abord, les composants sur lesquels repose Onyxia : à la fois ses briques technologiques (Kubernetes, MinIO, Vault) de même que l'ensemble des services du catalogue sont *open source*. De même, le code source du projet Onyxia est disponible publiquement sur GitHub[7] sous licence MIT ce qui, associé à une documentation détaillée[8], favorise les réutilisations du projet. Enfin, cette orientation est sensible à travers les principes architecturaux qui ont guidé le développement du projet. La notion de « région », qui permet de paramétriser finement la configuration du logiciel et son interaction avec le cluster Kubernetes sous-jacent, la possibilité de définir simplement des identités graphiques spécifiques (voir Figure 7), et la possibilité d'adapter la catalogue de services (cf. supra) facilitent les ré-instanciations et l'appropriation du projet par les organisations.

Figure 7. – Un projet, de multiples instances : l'interface web est adaptable à l'identité graphique de chaque organisation



Ainsi, deux instances d'Onyxia cohabitent actuellement à l'Insee, qui couvrent chacune des besoins différents. L'instance historique, nommée SSP Cloud, est une instance de démonstration du projet Onyxia. Equipée de ressources de calcul conséquentes[9] et du catalogue de services le plus complet, cette plateforme est conçue comme un bac à sable permettant d'expérimenter en toute autonomie des nouvelles méthodes et outils de *data science*[34]. Au-delà de ses capacités techniques, le SSP Cloud incarne les principes de l'innovation ouverte [39]. Déployé sur internet[10], il

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

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

[9] Sur le plan matériel, le SSP Cloud est constitué d'un cluster Kubernetes d'environ 20 serveurs, pour une capacité totale de 10 To de RAM, 1100 processeurs, 34 GPU et 150 To de stockage.

[10] https://datalab.sspcloud.fr/

est accessible non seulement aux agents de l'Insee mais également, plus largement, aux agents des ministères, aux universités et grandes écoles françaises et aux autres INS européens. La nature fondamentalement collaborative du SSP Cloud s'est avérée particulièrement bénéfique pour l'organisation d'événements innovants, tels que des hackathons — tant au niveau national qu'international — et dans le domaine académique. Il est devenu une ressource intégrale pour plusieurs universités et Grandes Écoles en France, favorisant l'utilisation d'environnements *cloud* et reproductibles, tout en évitant l'effet d'enfermement propriétaire dû à une dépendance excessive des institutions éducatives envers des solutions *cloud* propriétaires. En conséquence, la plateforme est désormais largement utilisée dans le service statistique publique français et plus largement, avec environ 1000 utilisateurs uniques par mois début 2025. Ces utilisateurs forment une communauté dynamique grâce à un canal de discussion centralisé[11] ; ils contribuent à améliorer l'expérience utilisateur en signalant des bugs, en proposant de nouvelles fonctionnalités et en contribuant ainsi directement au projet.

Si le côté résolument ouvert du SSP Cloud rend cette instance particulièrement adaptée à la préfiguration de nouveaux usages, il impose en contrepartie l'utilisation de données ouvertes dans les projets qui y sont menés. A ce titre, cette instance ne pouvait servir plateforme de traitements de données confidentielles à l'Insee.

Associée à une cette transparence facilite grandement la possibilité pour d'autres organisations de créer des instances de plateformes de *data science* basées sur le logiciel Onyxia et de les adapter à leurs besoins spécifiques (voir Figure 7). Cela a permis au projet d'attirer une communauté croissante de contributeurs issus des statistiques publiques (Statistique Norvège), des ONG (Mercator Ocean[12]), des centres de recherche et même de l'industrie, favorisant ainsi une transition progressive vers une gouvernance plus décentralisée du projet. Dans les prochaines années, l'implication des INS (Instituts Nationaux de Statistique) du système statistique européen devrait augmenter, puisque le SSPCloud a été choisie comme plateforme *data science* de référence dans le cadre du projet AIML4OS[13].

4 Faire du *machine learning* en production : l'exemple de la codification de l'APE

Ce chapitre vise à illustrer comment l'Insee a réussi à déployer son premier modèle de machine learning (ML) en production. Il propose une description détaillée de l'approche MLOps à laquelle ce projet s'est efforcé d'adhérer, en mettant l'accent sur les différentes technologies employées. En particulier, nous soulignons le rôle crucial des technologies *cloud* qui ont permis la construction du projet de manière itérative, ainsi que la manière dont Onyxia a grandement facilité cette construction en fournissant des environnements de développement flexibles et des outils pour entraîner, déployer et surveiller les modèles des modèles d'apprentissage automatique. De plus, la convergence entamée des environnement de *self* (LS^3), de développement (KubeDev) et de

[11] Lien vers les canaux de discussion https://www.tchap.gouv.fr/#/room/#SSPCloudXDpAw6v:agent.finances.tchap.gouv.fr et https://join.slack.com/t/3innovation/shared_invite/zt-19tht9hvr-bZGMdW8AV_wvd5kz3wRSMw

[12] Lien vers l'instance Onyxia de Mercator Ocean : https://datalab.dive.edito.eu/

[13] Plus d'informations à propos de ce projet disponibles à https://cros.ec.europa.eu/dashboard/aiml4os

production (KubeProd) constitue une réelle avancée pour faciliter la mise en production d'autres modèles d'apprentissage automatique[14]. Le projet présenté est disponible en open source[15] et reste en cours de développement actif.

4.1 Améliorer la codification de l'APE à l'aide de méthodes d'apprentissage automatique

4.1.1 Motivation

Les tâches de codification sont des opérations bien connues des instituts statistiques, et peuvent parfois être complexes en raison de la taille des nomenclatures. À l'Insee, un outil sophistiqué appelé Sicore a été développé dans les années 1990 pour effectuer diverses tâches de classification [40]. Cet outil repose sur un ensemble de règles déterministes permettant d'identifier les codes corrects à partir d'un libellé textuel en se basant sur un fichier de référence comprenant un certain nombre d'exemples. Chaque libellé d'entrée est soumis à ces règles et, lorsqu'un code correct est reconnu, il est attribué au libellé. En revanche, si le libellé n'est pas reconnu, il est soumis à une procédure de reprise manuelle par des gestionnaires de l'Insee.

Deux raisons principales ont motivé l'expérimentation de nouvelles méthodes de codification. Premièrement, un changement interne est survenu avec la refonte du répertoire statistique des entreprises en France (Sirene), qui liste toutes les entreprises et leur attribue un identifiant unique utilisé par les administrations publiques, le numéro Siren. Les principaux objectifs de cette refonte étaient d'améliorer la gestion quotidienne du répertoire pour les agents de l'Insee et de réduire les délais d'attente pour les entreprises. Par ailleurs, au niveau national, le gouvernement a lancé dans le cadre de la loi PACTE (n° 2019-486 du 22 mai 2019) un guichet unique pour les formalités des entreprises, offrant aux chefs d'entreprises plus de flexibilité dans la description de leurs activités principales mais les rendant ainsi mécaniquement plus verbeux que précédemment. Les tests initiaux ont révélé que Sicore n'était plus adapté pour effectuer la codification APE, puisque seulement 30% des liasses d'entreprises étaient automatiquement codées, et donc 70% partaient en reprise manuelle. Les équipes en charge du répertoire Sirene, déjà confrontées à des charges de travail importantes et à de fortes contraintes opérationnelles, ne pouvaient pas voir leur charge augmentée par une re-codification manuelle du fait du caractère très chronophage de la tâche. Ainsi, en mai 2022, la décision a été prise d'expérimenter de nouvelles méthodes pour effectuer cette tâche de codification, avec pour objectif de les utiliser en production dès le 1er janvier 2023, date de lancement du nouveau répertoire Sirene.

Trois parties prenantes étaient donc impliquées dans ce projet : l'équipe métier (division RIAS[16]), responsable de la gestion du répertoire statistique des entreprises ; l'équipe informatique, en charge du développement des applications liés au fonctionnement du répertoire ; et l'équipe d'innovation (l'unité SSP Lab), responsable de la mise en œuvre du nouvel outil de codification.

[14] Un modèle similaire pour améliorer la codification de la PCS dans la nomenclature PCS2020 a ainsi également été déployé en production récemment.

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

[16] Répertoire Interadministratif Sirene

4.1.2 La tâche de codification

Le projet consiste en un problème classique de classification dans le cadre du traitement du langage naturel (NLP). À partir d'une description textuelle de l'activité d'une entreprise, l'objectif est de prédire la classe associée dans la nomenclature APE. Cette classification présente la particularité d'être hiérarchique et comporte cinq niveaux différents[17] : section, division, groupe, catégorie et sous-catégorie. Au total, la nomenclature comprend 732 sous-classes, ce qui correspond au niveau le plus fin de la nomenclature et pour lequel on souhaite réaliser la codification. La table Table 1 fournit un exemple de cette structure hiérarchique.

Table 1. – Exemples d'éléments à différents niveaux de la nomenclature APE

Niveau	NAF	Libellé	Taille
Section	H	Transports et entreposage	21
Division	52	Entreposage et services auxiliaires des transports	88
Groupe	522	Services auxiliaires des transports	272
Catégorie	5224	Manutention	615
Sous-catégorie	**5224A**	**Manutention portuaire**	**732**

Avec la mise en place du guichet unique, les chefs d'entreprise décrivent désormais leur activité dans un champ de texte libre. Par conséquent, les nouveaux libellés diffèrent fortement des libellés harmonisés précédemment reçus. Il a donc été décidé de travailler avec des modèles d'apprentissage automatique, reconnus pour leur efficacité sur les tâches de classification supervisée de texte [41]. Cela représente un changement de paradigme significatif pour l'Insee, puisque le *machine learning* n'est pas traditionnellement utilisé dans la production des statistiques publiques. De plus, la perspective de mettre le nouveau modèle en production a été envisagée dès le début du projet, orientant de nombreux choix méthodologiques et techniques. Ainsi, plusieurs décisions stratégiques ont dû être rapidement prises, notamment en ce qui concerne la méthodologie, le choix d'un environnement de développement cohérent avec l'environnement de production cible, et l'adoption de méthodes de travail collaboratif.

4.1.3 Méthodologie

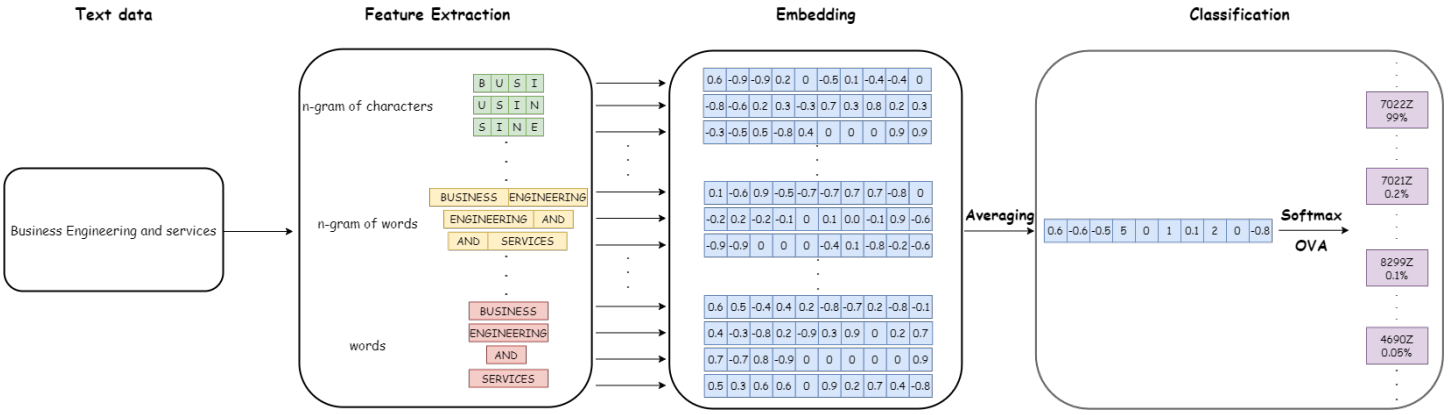
La classification textuelle à partir des champs de texte libre fournis par les chefs d'entreprise est une tâche complexe : les descriptions d'activité sont relativement courtes et contiennent donc peu d'information statistique, peuvent inclure des fautes d'orthographe et nécessitent souvent une expertise métier pour être correctement codées. Pour une telle tâche, les méthodes traditionnelles d'analyse textuelle, comme la vectorisation par comptage ou TF-IDF, sont souvent insuffisantes, tandis que les méthodes d'intégration basées sur des réseaux de neurones tendent à donner de meilleurs résultats [41]. Cependant, ces architectures nécessitent souvent des ressources de calcul importantes, et peuvent exiger du matériel spécifique, comme des GPUs, afin obtenir une latence acceptable lors de l'inférence. Ces contraintes ont, dans un premier temps, éloigné des modèles

[17] En réalité, il existe cinq niveaux en France, mais seulement quatre au niveau européen.

plus performants, tels que les modèles Transformer, et ont orientés le choix vers le modèle fastText [42], un réseau de neurone plus simple basé sur des plongements lexicaux. Le modèle fastText est extrêmement rapide à entraîner, et l'inférence ne nécessite pas de GPU pour obtenir un temps de latence faible. En outre, le modèle a donné d'excellents résultats pour le cas d'usage présenté, qui, compte tenu des contraintes de temps et de ressources humaines, étaient largement suffisants pour améliorer le processus existant. Enfin, l'architecture du modèle est relativement simple, ce qui facilite la communication et l'adoption au sein des différentes équipes de l'Insee.

Le modèle fastText repose sur une approche de sac de mots (*bag-of-words*) pour obtenir des plongements lexicaux (*word embeddings*) et une couche de classification basée sur la régression logistique[18]. L'approche sac de mots consiste à représenter un texte comme un ensemble de représentations vectorielles de chacun des mots qui le composent. La spécificité du modèle fastText, par rapport à d'autres approches basées sur des principes similaires, est que les plongements lexicaux ne sont pas seulement calculés sur les mots, mais aussi sur des *n-grams* de mots et de caractères, fournissant ainsi plus de contexte et réduisant les biais liés aux fautes d'orthographe. Le plongement lexical d'une phrase est calculé comme une fonction des plongements lexicaux des mots (et n-grams de mots et de caractères), généralement une moyenne. Dans le cas de la classification textuelle supervisée, la matrice de plongement et les poids du classifieur sont appris simultanément lors de l'entraînement par descente de gradient, en minimisant la fonction de perte d'entropie croisée. La Figure 8 présente la *pipeline* complete des opérations effectuées par fastText sur un exemple de texte en entrée.

Figure 8. – Aperçu simplifié du processus derrière la classification textuelle avec fastText



4.2 Une approche orientée production et MLOps

Dès le début du projet, l'objectif était d'aller au-delà de la simple expérimentation et d'aboutir à une mise en production rapide du modèle. Par ailleurs, ce projet pilote avait également pour but de servir de référence pour les futurs projets de *machine learning* à l'Insee. L'équipe du projet a donc cherché à appliquer les bonnes pratiques de développement dès les premières étapes du projet : respect des standards communautaires de qualité du code, utilisation de scripts pour le dévelop-

[18]Pour une introduction (en Anglais) à la fois visuelle et intuitive aux plongements lexicaux, voir : https://jalammar.github.io/illustrated-word2vec/

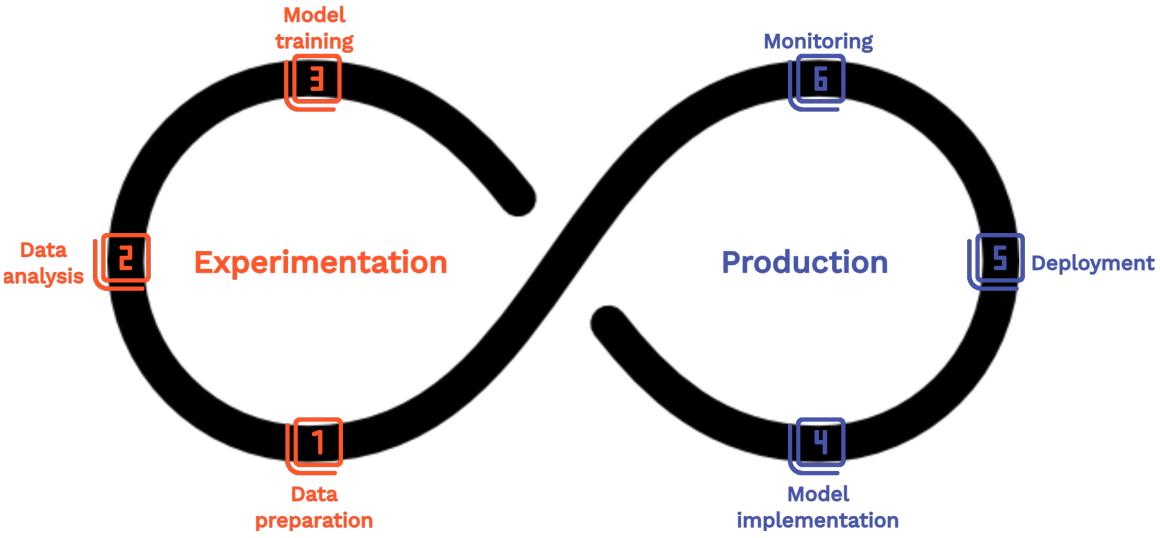
pement au lieu de notebooks, construction d'une structure modulaire semblable à un *package*, etc. Cependant, par rapport aux projets de développement traditionnels, les projets de *machine learning* présentent des caractéristiques spécifiques qui nécessitent l'application d'un ensemble de bonnes pratiques complémentaire, regroupées sous le nom de MLOps.

4.2.1 Du DevOps au MLOps

Le DevOps est un ensemble de pratiques conçu pour favoriser la collaboration entre les équipes de développement (*Dev*) et d'opérations (*Ops*). L'idée fondamentale est d'intégrer tout le cycle de vie d'un projet dans un continuum automatisé. Un outil important pour atteindre cette continuité sont les *pipeline*s CI/CD. Avec l'intégration continue (*Continuous Integration* ou CI), chaque *commit* de sur le code source du projet déclenche un processus d'opérations standardisées, telles que la construction de l'application, son test et la mise à disposition d'une version de l'application. Ensuite, le déploiement continu (*Continuous Deployment* ou CD) consiste en des outils pour automatiser le déploiement du nouveau code et limiter les interventions manuelles, tout en garantissant une supervision appropriée pour assurer la stabilité et la sécurité des processus. Cette approche favorise un déploiement plus rapide et continu des modifications ou ajouts nécessaires de fonctionnalités. En outre, en encourageant la collaboration entre les équipes, le DevOps accélère également le cycle de développement, permettant aux équipes de résoudre les problèmes au fur et à mesure qu'ils surviennent et d'intégrer efficacement les retours tout au long du cycle de vie du projet.

L'approche MLOps peut être vue comme une extension du DevOps, développée pour relever les défis spécifiques liés à la gestion du cycle de vie des modèles de ML. Fondamentalement, DevOps et MLOps partagent le même objectif : construire des logiciels de manière plus automatisée et robuste. La principale différence réside dans le fait qu'avec le MLOps, le logiciel inclut également une composante de *machine learning* (ML). Par conséquent, le cycle de vie du projet devient plus complexe. Le modèle de ML sous-jacent doit être réentraîné régulièrement afin d'éviter toute perte de performance au fil du temps. L'ingestion des données doit également être intégrée au processus, car de nouvelles données peuvent être utilisées pour améliorer les performances. La Figure 9 présente les étapes d'un projet de ML en utilisant une représentation continue, comme cela se fait traditionnellement en DevOps. Cela illustre un principe fondamental du MLOps : la nécessité d'une amélioration continue, décrite plus en détail dans Chapitre 4.2.2.

Figure 9. – L'approche MLOps favorise une gestion plus continue du cycle de vie des projets de ML



4.2.2 Principes du MLOps

Le MLOps repose sur quelques principes fondamentaux qui sont essentiels pour construire des applications de *machine learning* évolutives et adaptées au passage en production.

Le principe essentiel du MLOps est l'amélioration continue, reflétant la nature itérative des projets de ML. Lors de la phase d'expérimentation, le modèle est développé à partir d'un ensemble de données d'entraînement, qui diffère généralement des données de production. Une fois le modèle déployé en production, les nouvelles données sur lesquelles le modèle doit effectuer des prédictions peuvent révéler des informations sur ses performances et ses éventuelles lacunes. Ces informations nécessitent un retour à la phase d'expérimentation, où les *data scientists* ajustent ou redéfinissent leurs modèles pour corriger les problèmes découverts ou améliorer la précision. Ce principe souligne donc l'importance de construire une boucle de rétroaction permettant des améliorations continues tout au long du cycle de vie d'un modèle. L'automatisation, en particulier grâce à l'utilisation de *pipeline*s CI/CD, joue un rôle crucial en rendant la transition entre les phases d'expérimentation et de production plus fluide. La surveillance (*monitoring*) est également une composante essentielle de ce processus : un modèle déployé en production doit être continuellement analysé pour détecter d'éventuelles dérives importantes susceptibles de réduire ses performances prédictives et nécessitant des ajustements supplémentaires, comme un ré-entraînement.

Un autre objectif majeur du MLOps est de promouvoir la reproductibilité, en garantissant que toute expérience de ML puisse être reproduite de manière fiable avec les mêmes résultats. Les outils de MLOps facilitent ainsi une sauvegarde détaillée des expériences de ML, incluant les étapes de prétraitement des données, les hyperparamètres des modèles utilisés et les algorithmes d'entraînement. Les données, modèles et codes sont versionnés, permettant aux équipes de revenir à des versions antérieures si une mise à jour ne donne pas les résultats escomptés. Enfin, ces outils aident à produire des spécifications détaillées de l'environnement informatique utilisé pour

produire ces expériences — comme les versions des bibliothèques — et reposent souvent sur des conteneurs pour reproduire les mêmes conditions que celles dans lesquelles le modèle initial a été développé.

Enfin, le MLOps vise à favoriser le travail collaboratif. Les projets basés sur le ML impliquent généralement une gamme plus large de profils : équipes métier et équipes de *data science* d'un côté, développeurs et équipes de production informatique de l'autre. Comme le DevOps, le MLOps met donc l'accent sur la nécessité d'une culture collaborative et d'éviter le travail en silos. Les outils de MLOps incluent généralement des fonctionnalités collaboratives, telles que des stockages centralisés pour les modèles de ML ou les caractéristiques (*features*) de ML, qui facilitent le partage des composants entre les membres des équipes et limitent la redondance des développements.

4.2.3 Implémentation avec MLflow

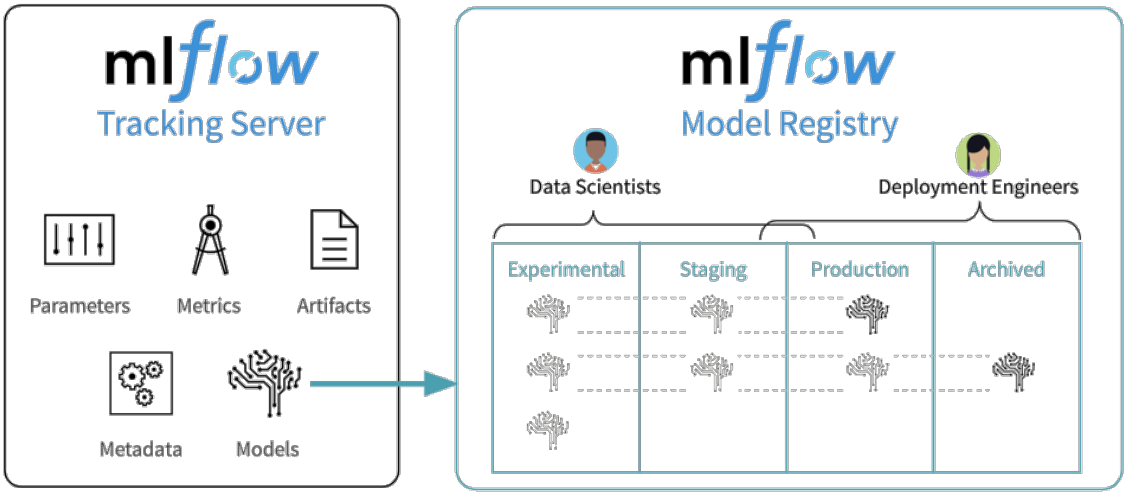
De nombreux outils ont été développés pour mettre en œuvre l'approche MLOps dans des cas d'usage concrets. Tous visent à appliquer, sous une forme ou une autre, les principes fondamentaux décrits précédemment. Dans ce projet, le choix a été fait de s'appuyer sur un outil open-source populaire nommé MLflow[19]. Ce choix ne reflète pas une supériorité inhérente de MLflow par rapport à d'autres outil, mais s'explique par un ensemble de bonnes propriétés associées à MLflow, qui en font une solution particulièrement pertinente pour le cas d'usage. Tout d'abord, il couvre l'intégralité du cycle de vie des projets de ML, tandis que d'autres outils peuvent être plus spécialisés sur certaines parties seulement. Ensuite, il offre une grande interopérabilité grâce à une bonne interface avec les bibliothèques populaires de ML — telles que PyTorch, Scikit-learn, XGBoost, etc. — et prend en charge plusieurs langages de programmation — notamment Python, R et Java, couvrant ainsi le spectre des langages couramment utilisés à l'Insee. Enfin, MLflow s'est révélé généralement très accessible, encourageant ainsi son adoption par les membres du projet et facilitant la collaboration continue entre eux.

MLflow fournit un ensemble d'outils intégrés permettant d'implémenter les principes du MLOps efficacement au sein des projets de ML. Les *data scientists* peuvent encapsuler leur travail dans des objects *Projects* qui regroupent le code ML et ses dépendances, garantissant que chaque expérimentation soit reproductible et puisse être ré-exécutée de manière identique. Un projet s'appuie sur un objet *Model*, un format standardisé compatible avec la plupart des bibliothèques de ML et offrant une méthode normalisée pour déployer le modèle, par exemple via une API. Cette interopérabilité et cette standardisation sont essentielles pour soutenir l'amélioration continue du projet, puisque les modèles entraînés avec une multitude de packages peuvent être facilement comparés ou remplacés les uns par les autres sans casser le code existant. Le *Tracking Server* enregistre des informations détaillées sur chaque expérimentation — hyperparamètres, métriques, artefacts et données — ce qui favorise d'une part la reproductibilité et facilite la phase de sélection des modèles grâce à une interface utilisateur permettant de comparer simplement les performances. Une fois la phase d'expérimentation terminée, les modèles sélectionnés sont rajoutés dans le *Model Registry*, où ils sont versionnés et prêts pour le déploiement. Cet entrepôt sert de « magasin » centralisé pour les modèles, permettant aux différents membres ou équipes du projet

[19] https://github.com/MLflow/MLflow

de gérer collaborativement le cycle de vie du projet. La Figure 10 illustre les composants principaux de MLflow et la manière dont ils facilitent un flux de travail plus continu et collaboratif au sein d'un projet de ML.

Figure 10. – Composants principaux de MLflow. Source : Databricks.



4.3 Faciliter le développement itératif avec les technologies cloud

Bien que l'amélioration continue soit un principe fondamental du MLOps, il s'agit également d'un principe exigeant. En particulier, celui-ci nécessite de concevoir et de construire un projet sous la forme d'un *pipeline* intégré, dont les différentes étapes sont principalement automatisées, de l'ingestion des données jusqu'à la surveillance du modèle en production. Dans ce contexte, le développement itératif est essentiel pour construire un produit minimum viable qui sera ensuite affiné et amélioré au fil du temps. Cette section illustre comment les technologies *cloud*, via le projet Onyxia, ont été déterminantes pour construire le projet sous forme de composants modulaires interconnectés, renforçant ainsi considérablement la capacité d'amélioration continue.

4.3.1 Un environnement de développement flexible

Dans un projet de ML, la flexibilité de l'environnement de développement est essentielle. Premièrement, en raison de la diversité des tâches à accomplir : collecte des données, prétraitement, modélisation, évaluation, inférence, surveillance, etc. Deuxièmement, parce que le domaine du ML évolue rapidement, il est préférable de construire une application de ML sous forme d'un ensemble de composants modulaires afin de pouvoir mettre à jour certains éléments sans perturber l'ensemble du *pipeline*. Comme discuté dans la Section 2.2, les technologies *cloud* permettent de créer des environnements de développement modulaires et évolutifs.

Cependant, comme également abordé dans la Section 3, l'accès à ces ressources ne suffit pas. Un projet de ML nécessite une grande variété d'outils pour se conformer aux principes du MLOps : stockage des données, environnements de développement interactifs pour expérimenter librement, outils d'automatisation, outils de surveillance, etc. Bien que ces outils puissent être installés sur un cluster Kubernetes, il est essentiel de les rendre disponibles aux *data scientists* de manière

préconfigurée pour faciliter leur adoption. Grâce à son catalogue de services et à l'injection automatique de configurations dans les services, Onyxia permet de construire des projets qui reposent sur plusieurs composants *cloud* capables de communiquer facilement entre eux.

La manière dont l'entraînement du modèle a été réalisé pour ce projet illustre bien la flexibilité offerte par Onyxia pendant la phase d'expérimentation. Tout le code utilisé pour l'entraînement est écrit en Python au sein d'un service VSCode. Grâce à l'injection automatique des jetons d'accès S3 dans chaque service au démarrage, les différents utilisateurs du projet peuvent interagir directement avec les données d'entraînement stockées dans un *bucket* S3 sur MinIO. Toutes les expériences menées lors de la phase de sélection du modèle sont consignées dans une instance partagée de MLflow, qui enregistre les données sur une instance PostgreSQL automatiquement lancée sur Kubernetes, tandis que les artefacts (modèles entraînés et métadonnées associées) sont stockés sur MinIO.

Le modèle a été entraîné en utilisant une recherche exhaustive (*grid-search*) pour l'ajustement des hyperparamètres et évalué par validation croisée (*cross-validation*). Cette combinaison, reconnue pour offrir une meilleure évaluation des performances de généralisation du modèle, nécessite cependant d'importantes ressources de calcul en raison de la nature combinatoire du test de nombreuses combinaisons d'hyperparamètres. Sur le plan technique, ce processus de recherche a été implémenté à l'aide d'Argo Workflows, un moteur de *workflows* open source conçu pour orchestrer des tâches parallèles sur Kubernetes, chaque tâche étant spécifiée comme un conteneur indépendant. Cela a permis de comparer facilement les performances des différents modèles entraînés et de sélectionner le meilleur en utilisant les outils de comparaison et de visualisation disponibles dans l'interface utilisateur de MLflow.

En résumé, la phase d'entraînement a été rendue à la fois efficace et reproductible grâce à l'utilisation de nombreux composants modulaires interconnectés — une caractéristique distinctive des technologies *cloud* — mis à disposition des *data scientists* grâce à Onyxia.

4.3.2 Déploiement d'un modèle

Une fois que les modèles candidats ont été optimisés, évalués et qu'un modèle performant a été sélectionné, l'étape suivante consiste à le rendre accessible aux utilisateurs finaux de l'application. Fournir simplement le modèle entraîné sous forme d'artefact, ou même uniquement le code pour l'entraîner, n'est pas une manière optimale de le transmettre car cela suppose que les utilisateurs disposent des ressources, de l'infrastructure et des connaissances nécessaires pour l'entraîner dans les mêmes conditions. L'objectif est donc de rendre le modèle accessible de manière simple et interopérable, c'est-à-dire qu'il doit être possible de l'interroger avec divers langages de programmation et par d'autres applications de manière programmatique.

Dans ce contexte, la solution retenue a été de déployer le modèle via une API REST. Cette technologie est devenue une norme pour servir des modèles de ML, car elle présente plusieurs avantages. Tout d'abord, elle s'intègre parfaitement dans un environnement orienté *cloud* : comme les autres composants de l'architecture choisie, elle permet d'interroger le modèle en utilisant des requêtes HTTP standard, ce qui contribue à la modularité du système. De plus, elle est interopérable : reposant sur des technologies standards pour les requêtes (requêtes HTTP) et les réponses

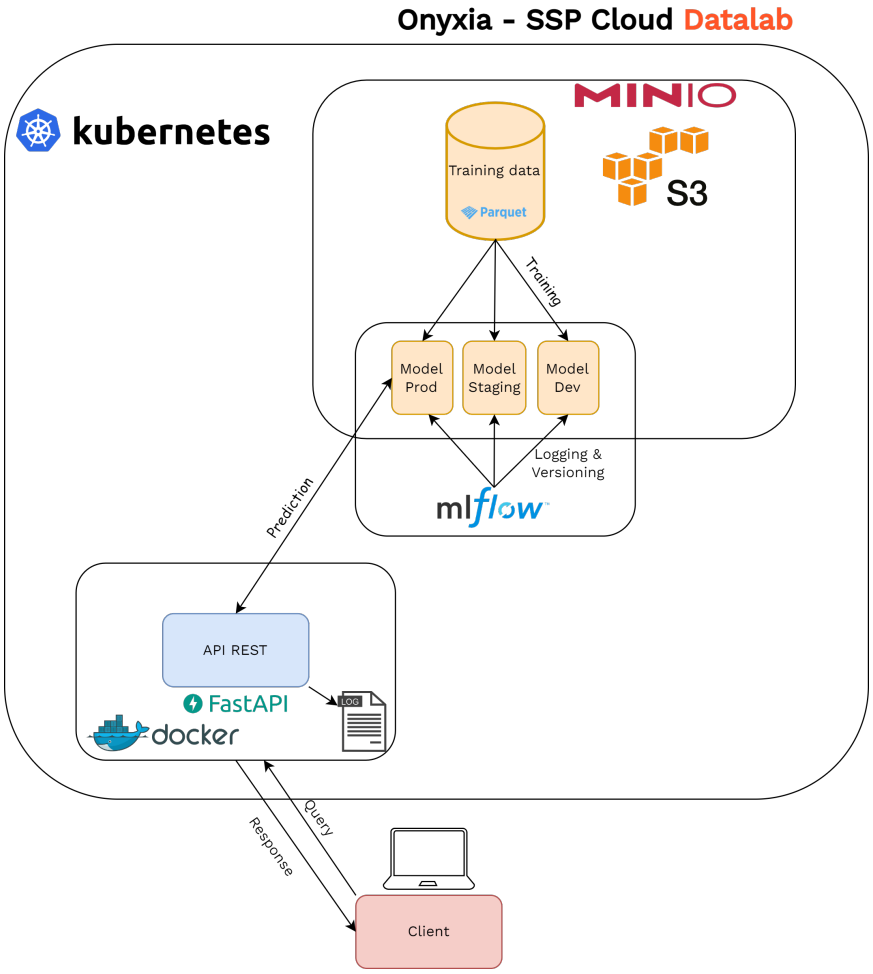
(généralement une chaîne formatée en JSON), elle est largement indépendante du langage de programmation utilisé pour effectuer les requêtes. Enfin, les API REST offrent une grande évolutivité grâce à leur conception sans état (*stateless*)[20]. Chaque requête contient toutes les informations nécessaires pour être comprise et traitée, ce qui permet de dupliquer facilement l'API sur différentes machines pour répartir une charge importante, dans une logique de scalabilité horizontale.

L'API servant le modèle a été déployée avec FastAPI[21], un framework web accessible et bien documenté pour construire des APIs avec Python. Le code de l'API et les dépendances logicielles nécessaires sont encapsulés dans une image Docker, ce qui permet de la déployer sous forme de conteneur sur le cluster Kubernetes. L'un des avantages majeurs de Kubernetes est sa capacité d'adapter la puissance de l'API — via le nombre de pods d'API effectivement déployés — en fonction de la demande, tout en fournissant un équilibrage de charge automatique. Au démarrage, l'API récupère automatiquement le modèle approprié depuis l'entrepôt de modèles MLflow stocké sur MinIO. Enfin, comme le code de l'application est *packagé* en utilisant l'API standardisée de MLflow — permettant par exemple d'intégrer directement l'étape de prétraitement dans chaque appel API — le code d'inférence reste largement uniforme, quel que soit le framework de ML sous-jacent utilisé. Ce processus de déploiement est résumé dans la Figure 11.

[20]La conception sans état (*stateless*) fait référence à une architecture système où chaque requête d'un client au serveur contient toutes les informations nécessaires pour comprendre et traiter la requête. Cela signifie que le serveur ne stocke aucune information sur l'état du client entre les requêtes, ce qui permet de traiter chaque requête indépendamment. Cette conception simplifie l'évolutivité et renforce la robustesse du système, car n'importe quel serveur peut gérer une requête sans dépendre des interactions précédentes.

[21]https://fastapi.tiangolo.com

Figure 11. – Une approche basé sur des technologies *cloud* pour servir un modèle de ML via une API REST



4.3.3 Construction d'un *pipeline* intégré

L'architecture construite à ce stade reflète déjà certains principes importants du MLOps. L'utilisation de la conteneurisation pour déployer l'API, ainsi que celle de MLflow pour suivre les expérimentations pendant le développement du modèle, garantissent la reproductibilité des prédictions. L'utilisation de l'entrepôt central de modèles fourni par MLflow facilite la gestion du cycle de vie des modèles de manière collaborative. De plus, la modularité de l'architecture laisse de la place pour des améliorations ultérieures, puisque des composants modulaires peuvent être ajoutés ou modifiés facilement sans casser la structure du projet dans son ensemble. Comme nous le verrons dans les sections suivantes, cette propriété s'est avérée essentielle pour construire le projet de manière itérative, permettant d'ajouter une couche de surveillance du modèle (Chapitre 4.3.4) et un composant d'annotation (Chapitre 4.3.6) afin de favoriser l'amélioration continue du modèle en intégrant « l'humain dans le cycle de vie du modèle de ML » (*human in the loop*).

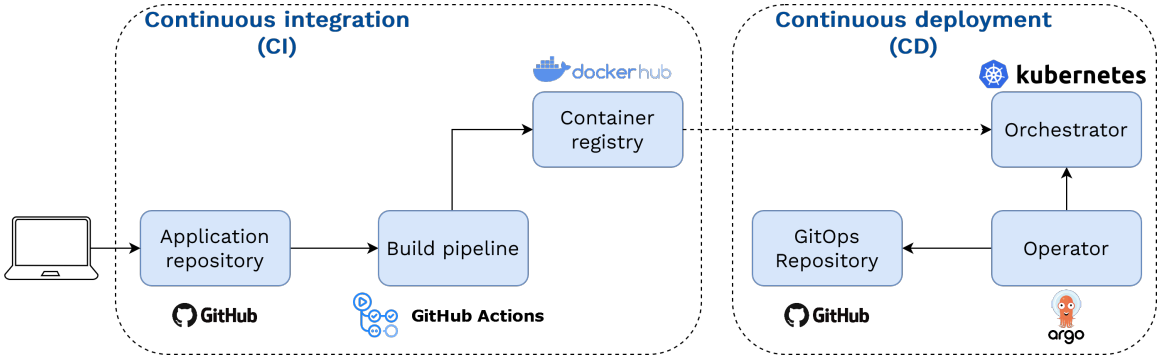
Cependant, la capacité à affiner l'architecture de base de manière itérative nécessite également une plus grande continuité dans le processus. À ce stade, le processus de déploiement implique

plusieurs opérations manuelles. Par exemple, l'ajout d'une nouvelle fonctionnalité à l'API nécessiterait de construire une nouvelle image, de la taguer, de mettre à jour les manifestes Kubernetes utilisés pour déployer l'API et de les appliquer sur le cluster afin de remplacer l'instance existante avec un temps d'arrêt minimal. De même, un changement de modèle servi via l'API nécessiterait une simple modification du code, mais plusieurs étapes manuelles pour mettre à jour la version sur le cluster. En conséquence, les *data scientists* ne sont pas encore totalement autonomes dans cette configuration pour prototyper et tester des versions mises à jour du modèle ou de l'API, ce qui limite le potentiel d'amélioration continue.

Afin d'automatiser ce processus, un *pipeline* CI/CD — un concept déjà présenté dans Chapitre 4.2.1 — a été construit afin d'intégrer ces différentes étapes. La Figure 12 illustre l'implémentation spécifique du *pipeline* CI/CD retenue pour le projet. Toute modification du code sur le dépôt de l'API, associée à un nouveau *tag*, déclenche un processus de CI (implémenté avec GitHub Actions) qui construit l'image Docker et la publie sur un registre public de conteneurs (DockerHub). Cette image peut ensuite être récupérée et déployée par l'orchestrateur de conteneurs (Kubernetes) en spécifiant et en appliquant manuellement de nouveaux manifestes pour mettre à jour les ressources Kubernetes de l'API.

Cependant, cette approche présente un inconvénient : elle limite la reproductibilité du déploiement, car chaque ressource est gérée indépendamment par l'orchestrateur, et le cycle de vie du déploiement de l'API dans son ensemble n'est pas contrôlé. Pour pallier cette lacune, la partie déploiement a été intégrée dans un *pipeline* CD basée sur l'approche GitOps : les manifestes des ressources de l'API sont également stockés dans un dépôt Git. L'état de ce dépôt « GitOps » est surveillé par un opérateur Kubernetes (ArgoCD), de sorte à ce que toute modification des manifestes de l'application soit directement propagée au déploiement sur le cluster. Dans ce nouveau *pipeline* intégré de bout en bout, la seule action nécessaire pour déclencher une mise à jour de l'API est de modifier le tag de l'image de l'API indiquant la version à déployer.

Figure 12. – Le *pipeline* CI/CD implémenté dans le projet



4.3.4 Surveillance d'un modèle en production

Une fois la phase initiale de développement du projet terminée — incluant l'entraînement, l'optimisation et le déploiement du modèle pour les utilisateurs —, il est crucial de comprendre que les responsabilités du *data scientist* ne s'arrêtent pas là. Traditionnellement, le rôle des *data scientists* se limite souvent à l'entraînement et à la sélection du modèle à déployer, le déploiement étant

généralement délégué au département informatique. Cependant, une spécificité des projets de ML est que, une fois en production, le modèle n'a pas encore atteint la fin de son cycle de vie : il doit être surveillé en permanence afin d'éviter toute dégradation indésirable des performances. La surveillance continue du modèle déployé est essentielle pour garantir la conformité des résultats aux spécifications, anticiper les changements dans les données et améliorer le modèle de manière itérative. Même en production, la compréhension fine de la problématique métier que cherche à résoudre l'approche ML reste entièrement nécessaire.

Le concept de surveillance peut avoir différentes significations selon le contexte. Pour les équipes informatiques, il s'agit principalement de vérifier l'efficacité technique de l'application, notamment en termes de latence, de consommation de mémoire ou d'utilisation du disque de stockage. En revanche, pour les *data scientists* ou les équipes métier, la surveillance est davantage centrée sur le suivi méthodologique du modèle. Cependant, le suivi en temps réel des performances d'un modèle de ML est souvent une tâche complexe. Contrairement à la phase d'entraînement, lors de laquelle on dispose de données labellisées qui permettent d'étalloner le modèle, en production cette « vérité terrain » (*ground truth*) n'est généralement pas connue au moment de la prédiction. Il est donc courant d'utiliser des proxys pour détecter les signes éventuels de dégradation des performances. Deux types principaux de dégradation d'un modèle ML sont généralement distingués. Le premier est le *data drift*, qui se produit lorsque la distribution des données utilisées pour l'inférence diffèrent significativement de celle des données utilisées lors de l'entraînement. Le second est le *concept drift*, qui survient lorsqu'un changement dans la relation statistique entre les variables explicatives et la variable cible est observé au fil du temps. Par exemple, le mot « Uber » était habituellement associé à des codes liés aux services de taxis. Cependant, avec l'apparition des services de livraison de repas comme « Uber Eats », cette relation entre le libellé et le code associé a changé. Il est donc nécessaire de repérer au plus tôt ces changements afin de ne pas dégrader la codification.

Dans le cadre du projet, l'objectif est d'atteindre le taux le plus élevé possible de libellés correctement classifiés, tout en minimisant le nombre de descriptions nécessitant une intervention manuelle. Ainsi, le but est de distinguer les prédictions correctes des prédictions incorrectes sans avoir accès au préalable à la vérité terrain. Pour y parvenir, on calcule un indice de confiance, défini comme la différence entre les deux scores de confiance les plus élevés parmi les résultats renvoyés par le modèle. Pour une description textuelle donnée, si l'indice de confiance dépasse un seuil déterminé, la description est automatiquement codée. Sinon, elle est codée manuellement par un agent de l'Insee. Cette tâche de codification manuelle est néanmoins assistée par le modèle ML : via une application qui interroge l'API, l'agent visualise les cinq codes les plus probables selon le modèle. Le seuil choisi pour l'indice de confiance est un paramètre que l'équipe métier peut utiliser pour arbitrer entre la charge de travail qu'elle est disposée à assumer pour la reprise manuelle et le taux d'erreurs qu'elle est prête à tolérer.

4.3.5 Définition du seuil de codification automatique et surveillance en production

La définition du seuil pour la codification automatique des descriptions textuelles a été une étape cruciale de ce processus, nécessitant un compromis entre un taux élevé de codification automatique et une erreur minimale de classification. Afin de surveiller le comportement du modèle

en production, un tableau de bord interactif a été développée pour permettre de visualiser plusieurs métriques d'intérêt pour les équipes métier. Parmi ces métriques figurent le nombre de requêtes par jour et le taux de codification automatique quotidien, pour un seuil donné de l'indice de confiance. Cette visualisation permet aux équipes métier de connaître le taux de codification automatique qu'elles auraient obtenu si elles avaient choisi différents seuils. Le tableau de bord représente également la distribution des indices de confiance obtenus et compare des fenêtres temporelles afin de détecter des changements dans les distributions des prédictions renvoyées par le modèle[22]. Enfin, les indices de confiance peuvent être analysés à des niveaux de granularité plus fins, basés sur les niveaux d'agrégation de la classification statistique, pour identifier les classes les plus difficiles à prédire et celles qui sont plus ou moins fréquentes.

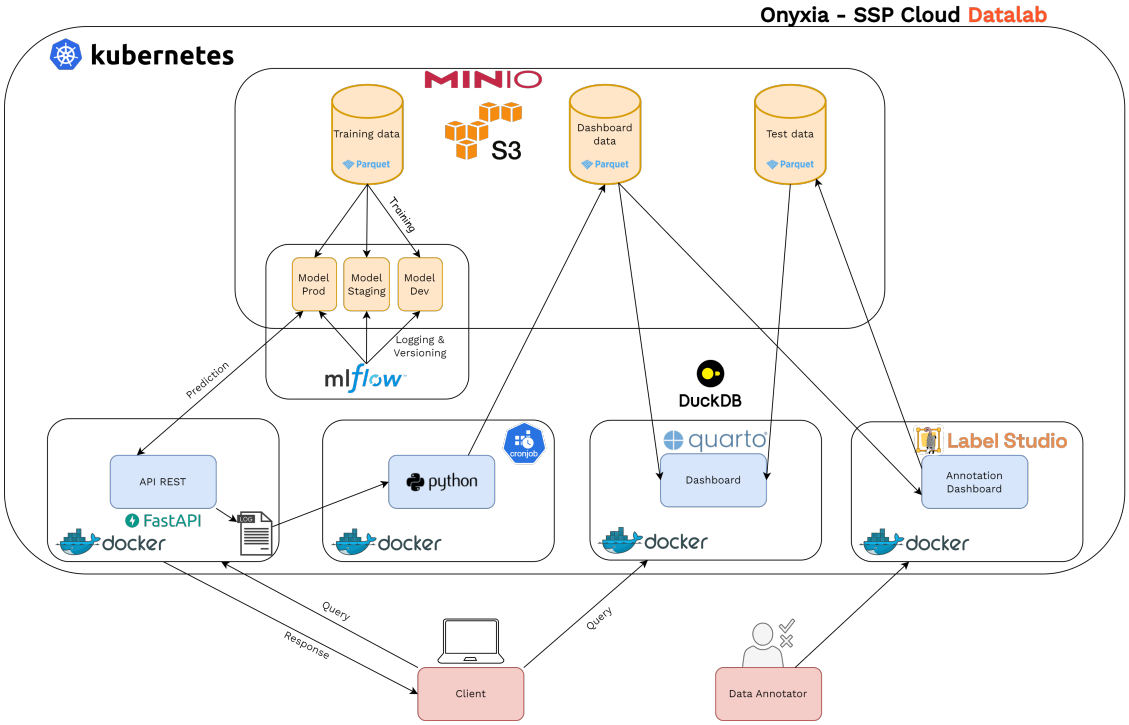
La Figure 13 détaille les composants ajoutés à l'architecture du projet pour fournir le tableau de bord de surveillance décrit ci-dessus. Tout d'abord, un processus Python (deuxième composant de la rangée inférieure) a été développé pour récupérer quotidiennement les logs de l'API et les transforme en fichiers partitionnés au format Parquet[23]. Ensuite,[24] pour construire un tableau de bord interactif (troisième composant de la rangée inférieure). Pour calculer les diverses métriques présentées dans le tableau de bord, les fichiers Parquet sont interrogés via le moteur optimisé DuckDB. À l'instar de l'API, le tableau de bord est construit et déployé sous forme de conteneur sur le cluster Kubernetes, et ce processus est également automatisé grâce à un *pipeline* CI/CD. Le composant d'annotation (quatrième composant de la rangée inférieure) est discuté dans la section suivante.

[22]Ces changements de distribution sont généralement vérifiés en calculant des distances statistiques — telles que la distance de Bhattacharyya, la divergence de Kullback-Leibler ou la distance de Hellinger — et/ou en effectuant des tests statistiques — tels que le test de Kolmogorov–Smirnov ou le test du khi-deux.

[23]Idéalement, les *frameworks* existants devraient être privilégiés par rapport aux solutions sur mesure pour adopter des routines standardisées. Lors de la construction de ce composant du *pipeline*, on a pu constater que les frameworks *cloud* existants pour l'analyse des logs présentaient d'importantes limites. Cela constitue une piste d'amélioration pour le projet.

[24]Successeur de R Markdown, Quarto est devenu un outil essentiel. Il unifie les fonctionnalités de plusieurs packages très utiles de l'écosystème R Markdown tout en offrant une prise en charge native de plusieurs langages de programmation, dont Python et Julia en plus de R. Il est de plus en plus utilisé à l'Insee pour produire des documents reproductibles et les exporter dans divers (HTML, PDF, ODT, etc.).

Figure 13. – Implémentation d'une architecture MLOps retenur pour le projet



4.3.6 Favoriser l'amélioration continue du modèle

La composante de surveillance du modèle fournit une vue détaillée et essentielle de l'activité du modèle en production. En raison de la nature dynamique des données de production, les performances des modèles de ML ont tendance à diminuer avec le temps. Pour favoriser l'amélioration continue du modèle, il est donc essentiel de mettre en place des stratégies permettant de surmonter ces pertes de performance. Une stratégie couramment utilisée est le réentraînement périodique du modèle, nécessitant la collecte de nouvelles données d'entraînement plus récentes et donc plus proches de celle observées en production.

Plusieurs mois après le déploiement de la première version du modèle en production, le besoin de mettre en œuvre un processus d'annotation continue est devenu de plus en plus évident pour deux raisons principales. Premièrement, un échantillon de référence (*gold standard*) n'était pas disponible lors de la phase d'expérimentation. Un sous-ensemble des données d'entraînement a donc été utilisé pour l'évaluation, tout en sachant que la qualité de la labelisation n'était pas optimale. La collecte continue d'un échantillon de référence permettrait ainsi d'obtenir une vue réaliste des performances du modèle en production sur des données réelles, en particulier sur les données codifiées automatiquement. Deuxièmement, la refonte de la nomenclature statistique APE prévue en 2025 impose aux INS d'adopter la dernière version. Cette révision, qui introduit des changements importants, nécessite une adaptation du modèle et surtout la création d'un nouveau jeu de données d'entraînement. L'annotation de l'ancien jeu de données d'entraînement selon la nouvelle nomenclature statistique est donc indispensable.

Dans ce contexte, une campagne d'annotation a été lancée début 2024 pour construire de manière continue un jeu de données de référence. Cette campagne est réalisée sur le SSP Cloud[25] en utilisant le service Label Studio, un outil *open source* d'annotation offrant une interface ergonomique et disponible dans le catalogue d'Onyxia. La Figure 13 montre comment le composant d'annotation (quatrième composant de la rangée inférieure) a pu être intégré facilement dans l'architecture du projet grâce à sa nature modulaire. En pratique, un échantillon de descriptions textuelles est tiré aléatoirement des données passées par l'API au cours des trois derniers mois. Cet échantillon est ensuite soumis à l'annotation par des experts APE via l'interface de Label Studio. Les résultats de l'annotation sont automatiquement sauvegardés sur MinIO, transformés au format Parquet, puis intégrés directement dans le tableau de bord de surveillance pour calculer et observer diverses métriques de performance du modèle. Ces métriques offrent une vision beaucoup plus précise des performances réelles du modèle sur les données de production, et permet notamment de détecter les cas les plus problématiques.

En parallèle, une campagne d'annotation pour construire un nouveau jeu d'entraînement adapté à la NAF 2025 a également été réalisée. En exploitant à la fois les nouvelles données d'entraînement et les métriques de performance dérivées de l'échantillon de référence, on peut espérer améliorer la précision du modèle de manière itérative grâce à des réentraînements périodiques et automatique dès lors que le moteur de codification aura migré sur le cluster Kubernetes de production.

5 Discussion

Le développement des méthodes de *data science* offre un potentiel considérable pour la statistique publique. Cependant, notre capacité à tirer profit de ces nouvelles méthodes dépend essentiellement de notre aptitude à produire des chaînes de production de qualité, robustes et adaptés à leurs objectifs. Cette évolution nécessite une réflexion approfondie sur ce qui constitue une infrastructure moderne et évolutive pour la *data science* dans le domaine des statistiques publiques. Cet article présente le projet Onyxia, une proposition pour une telle plateforme développée à l'Insee. En exploitant des technologies *cloud* devenues des standards dans l'écosystème de la donnée, le projet vise à accroître l'autonomie des statisticiens dans l'orchestration de leurs traitements statistiques, tout en favorisant la reproductibilité des statistiques produites. Les technologies *cloud* étant notoirement difficiles à configurer, la valeur principale d'Onyxia réside dans leur accessibilité pour les statisticiens grâce à une interface ergonomique, simple d'utilisation, et un catalogue de services préconfigurés couvrant les usages les plus courants d'un statisticien public. À travers un projet interne visant à refondre le processus de codification de l'Activité Principale de l'Entreprise (APE) en utilisant des méthodes d'apprentissage automatique, nous illustrons comment Onyxia permet de construire de manière itérative des projets de *machine learning* prêts à passer en production, favorisant l'amélioration continue, un principe fondamental de l'approche MLOps.

Initialement développé comme un projet interne, Onyxia a acquis une reconnaissance dépassant le cadre de l'Insee ou de l'administration française. Convaincues du potentiel des technologies *cloud* pour renforcer l'autonomie et exploiter pleinement le potentiel de la *data science*, plusieurs organisations disposent désormais d'une instance de production d'Onyxia (comme c'est le cas à

[25] https://datalab.sspcloud.fr/

l'Insee avec le déploiement récent de LS^3), et de nombreuses autres sont en phase de test ou d'implémentation. Par ailleurs, le choix d'Onyxia comme plateforme de *data science* de référence dans le cadre du projet AIML4OS devrait encore accroître son adoption au sein du SSE. Cette tendance est naturellement très bénéfique pour le projet Onyxia, qui passe d'un projet développé en *open source* — mais principalement à l'Insee — à un véritable projet *open source* avec une base croissante de contributeurs. Cela, en retour, facilite son adoption par d'autres organisations, en offrant davantage de garanties sur sa pérennité indépendamment de la stratégie de l'Insee. La gouvernance du projet évolue actuellement pour refléter cette tendance, notamment avec l'organisation de réunions communautaires mensuelles et la création d'un canal public et d'une feuille de route pour le projet[26].

Malgré ce succès, nous constatons plusieurs limites à l'adoption généralisée du projet au sein des organisations. Tout d'abord, il est essentiel de rappeler que le choix fondamental fait par les organisations qui adoptent Onyxia ne porte pas sur le logiciel en lui-même, mais sur les technologies sous-jacentes : la conteneurisation (via Kubernetes) et le stockage d'objets. Ces technologies peuvent représenter des coûts d'entrée important pour les organisations, puisqu'elles nécessitent un investissement dans le développement et le maintien de compétences qui ne sont pas toujours présentes dans les INS. Cependant, on constate que les organisations qui manipulent de la donnée tendent de plus en plus à s'orienter vers des solutions *cloud* qui pourrait atténuer ces défis à long terme.

De même, la transition vers les technologies *cloud* impose des coûts d'entrée pour les statisticiens. Tout d'abord, ils sont souvent confrontés à une perte de repères quant à l'endroit où les calculs sont réellement effectués : bien qu'ils soient habitués à effectuer des calculs sur des serveurs centralisés plutôt que sur un ordinateur personnel, le conteneur ajoute une couche d'abstraction qui rend cet emplacement difficile à appréhender au départ. Mais le changement le plus perturbant dans ce paradigme est la perte de persistance des données. Dans les configurations traditionnelles — qu'il s'agisse d'un ordinateur personnel ou d'un serveur accessible via un bureau virtuel — le code, les données et l'environnement de calcul sont souvent mélangés dans une sorte de boîte noire. À l'inverse, les conteneurs, par construction, n'ont pas de persistance. Si le stockage d'objets fournit cette persistance, une utilisation adéquate de ces infrastructures exige une variété d'outils et de compétences : utilisation d'un système de contrôle de version pour le code (e.g. Git), interaction avec l'API de stockage d'objets pour enregistrer les données, gestion de fichiers de configuration et/ou de secrets et variables d'environnement, etc. En un sens, ces coûts d'entrée peuvent être considérés comme le « prix » de l'autonomie : grâce aux technologies *cloud*, les statisticiens ont désormais accès à des environnements évolutifs et flexibles leur permettant d'expérimenter plus librement, mais cette autonomie exige une montée en compétences significative, qui peut être intimidante et, *in fine*, limiter cette adoption. Cependant, notre expérience à l'Insee montre que cet effet peut être largement atténué grâce à une combinaison de formation des statisticiens aux bonnes pratiques de développement et d'accompagnement des projets statistiques lors de leur transition vers des infrastructures *cloud*.

[26] Toutes les informations sont disponibles sur le dépôt GitHub du projet : https://github.com/InseeFrLab/onyxia

Bien qu'Onyxia ait significativement démocratisé l'accès aux technologies *cloud* pour les statisticiens, l'intégration effective des méthodes de *data science* dans la production statistique des INS soulève des défis plus larges, d'ordre organisationnel. Une leçon majeure tirée du déploiement de notre premier modèle de *machine learning* en production est la nécessité de surmonter la segmentation des compétences entre les équipes informatiques, métier et innovation. Par nature, les projets de *machine learning*, pour pouvoir passer en production, impliquent un large éventail de compétences — connaissance du domaine métier, entraînement et amélioration des modèles ainsi que leur déploiement et supervision — et nécessitent donc une collaboration efficace entre des professionnels aux cultures de travail et langages de programmation variés. Notre expérience montre que les technologies *cloud*, en favorisant l'autonomie des *data scientists*, apportent plus de continuité aux projets d'apprentissage automatique et facilitent cette collaboration essentielle entre les différents profils. Toutefois, répondre pleinement à ces défis nécessite des choix qui dépassent le domaine technique. Par exemple, intégrer certaines compétence en *data science* directement au sein des équipes métier, en complément des équipes d'innovation centralisées, pourrait favoriser une meilleure collaboration. De même, recruter des profils qui ne sont pas traditionnellement présents dans les INS, comme les *data engineers* ou les *ML engineers*, pourrait apporter de nouvelles compétences à l'intersection des méthodologies statistiques et des techniques informatiques. Au final, la transition vers une approche axée sur la *data science* dans la production statistique doit s'appuyer sur une stratégie équilibrée qui lie des solutions techniques comme Onyxia avec des ajustements organisationnels et humains, favorisant une culture de collaboration, de formation continue et d'innovation.

Bibliographie

[1] T. 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*, vol. 38, nº 3, p. 917-922, 2022.

[2] DGINS, « Bucharest Memorandum on Official Statistics in a Datafied Society ». 2018.

[3] INSEE, « Horizon 2025 ». 2016.

[4] EUROSTAT, « ESSnet Big Data 2 - Final Technical Report ». 2021.

[5] B. Sakarovitch, M.-P. d. Bellefon, P. Givord, et M. Vanhoof, « Estimating the residential population from mobile phone data, an initial exploration », *Economie et Statistique*, vol. 505, nº 1, p. 109-132, 2018.

[6] M. Leclair, I. Léonard, G. Rateau, P. Sillard, G. Varlet, et P. Vernédal, « Scanner data: advances in methodology and new challenges for computing consumer price indices », *Economie et Statistique*, vol. 509, nº 1, p. 13-29, 2019.

[7] P. Descy, V. Kvetan, A. Wirthmann, et F. Reis, « Towards a shared infrastructure for online job advertisement data », *Statistical Journal of the IAOS*, vol. 35, nº 4, p. 669-675, 2019.

[8] D. Salgado, L. Sanguiao-Sande, S. Barragán, B. Oancea, et M. Suarez-Castillo, « A proposed production framework with mobile network data », in *ESSnet Big Data II - Workpackage I - Mobile Network Data*, 2020.

[9] A. Kowarik et M. 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)*, 2022, p. 269-270.

[10] F. Ricciato, F. De Meersman, A. Wirthmann, G. Seynaeve, et M. Skaliotis, « Processing of mobile network operator data for official statistics: the case for public-private partnerships », in *104th DGINS conference*, 2018.

[11] O. Lefebvre, M. Soulier, et T. Tortosa, « L'accueil des données administratives: un processus structurant », *Courrier des statistiques*, p. 141-142, 2024.

[12] T. H. Davenport et D. Patil, « Data scientist », *Harvard business review*, vol. 90, nº 5, p. 70-76, 2012.

[13] L. Liu, « Computing infrastructure for big data processing », *Frontiers of Computer Science*, vol. 7, p. 165-170, 2013.

[14] A. Saiyeda et M. A. Mir, « Cloud computing for deep learning analytics: A survey of current trends and challenges. », *International Journal of Advanced Research in Computer Science*, vol. 8, nº 2, 2017.

[15] E. L'Hour, R. Le Saout, et B. Rouppert, « Quelques bonnes pratiques de développement logiciel à l'usage du statisticien selfeur », *Courrier des statistiques*, p. 86-106, 2022.

[16] S. Chaudhuri et U. Dayal, « An overview of data warehousing and OLAP technology », *ACM Sigmod record*, vol. 26, nº 1, p. 65-74, 1997.

[17] S. Ghemawat, H. Gobioff, et S.-T. Leung, « The Google file system », in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, 2003, p. 29-43.

[18] J. Dean et S. Ghemawat, « MapReduce: simplified data processing on large clusters », *Communications of the ACM*, vol. 51, nº 1, p. 107-113, 2008.

[19] J. Tigani, « Big data is dead ». [En ligne]. Disponible sur: https://motherduck.com/blog/big-data-is-dead/

[20] A. S. Foundation, « Apache Parquet ». 2013.

[21] A. Dondon et P. Lamarche, « Quels formats pour quelles données? », *Courrier des statistiques*, p. 86-103, 2023.

[22] A. I. Abdelaziz, K. A. Hanson, C. E. Gaber, et T. A. Lee, « Optimizing large real-world data analysis with parquet files in R: A step-by-step tutorial », *Pharmacoepidemiology and Drug Safety*, 2023.

[23] Y. Li *et al.*, « Big data and cloud computing », *Manual of digital earth*, p. 325-355, 2020.

[24] O. Bentaleb, A. S. Belloum, A. Sebaa, et A. El-Maouhab, « Containerization technologies: Taxonomies, applications and challenges », *The Journal of Supercomputing*, vol. 78, nº 1, p. 1144-1181, 2022.

[25] R. Vaño, I. Lacalle, P. Sowiński, R. S-Julián, et C. E. Palau, « Cloud-native workload orchestration at the edge: A deployment review and future directions », *Sensors*, vol. 23, nº 4, p. 2215-2216, 2023.

[26] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, et W. Zhou, « A comparative study of containers and virtual machines in big data environment », in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, p. 178-185.

[27] M. Leclair et others, « Using Scanner Data to Calculate the Consumer Price Index », *Courrier des statistiques*, vol. 3, p. 61-75, 2019.

[28] S. Samundiswary et N. M. Dongre, « Object storage architecture in cloud for unstructured data », in *2017 International Conference on Inventive Systems and Control (ICISC)*, 2017, p. 1-6.

[29] L. Leite, C. Rocha, F. Kon, D. Milojicic, et P. Meirelles, « A survey of DevOps concepts and challenges », *ACM Computing Surveys (CSUR)*, vol. 52, nº 6, p. 1-35, 2019.

[30] M. McNutt, « Reproducibility », *Science*, vol. 343, nº 6168, p. 229-230, 2014.

[31] European Commission, « European Statistics Code of Practice — revised edition 2017 ».

[32] S. Luhmann, J. Grazzini, F. Ricciato, M. Mészáros, J.-M. Museux, et M. Hahn, « Promoting reproducibility-by-design in statistical offices », in *2019 New Techniques and Technologies for Statistics (NTTS) conference*, 2019, p. . doi: 10.5281/zenodo.3240198.

[33] D. Moreau, K. Wiebels, et C. Boettiger, « Containers for computational reproducibility », *Nature Reviews Methods Primers*, vol. 3, nº 1, p. 50-51, 2023.

[34] F. Comte, A. Degorre, et R. Lesur, « SSPCloud: a creative factory to support experimentations in the field of official statistics », *Courrier des Statistiques, INSEE*, vol. 7, p. 68-85, 2022.

[35] S. Gokhale *et al.*, « 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)*, 2021, p. 1-5.

[36] J. Opara-Martins, R. Sahandi, et F. Tian, « Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective », *Journal of Cloud Computing*, vol. 5, p. 1-18, 2016.

[37] J. Opara-Martins, M. Sahandi, et F. Tian, « A holistic decision framework to avoid vendor lock-in for cloud saas migration », *Computer and Information Science*, vol. 10, nº 3, 2017.

[38] C. M. Schweik, « Free/open-source software as a framework for establishing commons in science », 2006.

[39] H. W. Chesbrough, *Open innovation: The new imperative for creating and profiting from technology*. Harvard Business Press, 2003.

[40] E. Meyer et P. Rivière, « SICORE, un outil et une méthode pour le chiffrement automatique à l'INSEE », in *Actes de la 4ème Conférence Internationale des Utilisateurs de Blaise*, mai 1997, p. 280-293. Consulté le: 4 juillet 2023. [En ligne]. Disponible sur: http://www.blaiseusers.org/1997/papers/meyer97.pdf

[41] Q. Li *et al.*, « A survey on text classification: From traditional to deep learning », *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 13, nº 2, p. 1-41, 2022.

[42] A. Joulin, E. Grave, P. Bojanowski, et T. Mikolov, « Bag of Tricks for Efficient Text Classification », *arXiv preprint arXiv:1607.01759*, 2016.