Toward a datalake

Conception d'une architecture de datalake

Rapport de Master 2 - mention Mathématiques et Applications & Informatique
Statistiques et Informatique Décisionnelle

à l'Université Paul Sabatier (UPS) – Toulouse III

19 août 2020

Vincent-Nam DANG

Laboratoire d'accueil : Institut de Recherche en Informatique de Toulouse (IRIT)
Directeur de recherche : Responsable de stage : Pr. TESTE Olivier, Dr. THIEBOLT François
Équipe d'accueil : Systèmes Multi-Agents Coopératifs / neOCampus

Mots-clés : Business Intelligence, data processing, Big Data, données hétérogènes, cloud computing, IoT, time series, IA, workflow

---

Résumé : Rapport de stage du projet "Toward a datalake" dans le cadre de l'opération neOCampus : développement, conception et mise en place d'une architecture de data lake pour les différents acteurs de neOCampus.

Table des matières

Résumé

Ce rapport décrit le projet "Toward a datalake" lors du stage de fin d'étude de Master 2. Il reprend en 3 parties le processus de conception et de mise en place d'une preuve de concept pour une architecture de stockage de données type data lake dans le cadre de l'opération neOCampus. La première phase est la conception et la modélisation d'une solution purement logicielle avec les différents besoins et les réponses à ces besoins. Une fois cette solution conçue et réfléchie, l'étape suivante est l'implementation de l'architecture dans OSIRIM, une plateforme de calcul de l'IRIT. La réflexion sur la sécurité et les aspects réseaux sont décrits ainsi que des réflexions sur la partie matérielle et les performances de la solution. Enfin, le coeur d'action d'un data lake, les traitements de données sont abordés. Les conditions exceptionnelles du stage n'ont pas permis pour la rédaction de ce rapport de fournir des implémentations de ces traitements, cependant ils sont réfléchis et prévus pour la fin du stage.

This report describes the results obtained for the project Tooward a data lake, which had been deve- lopped during this Master 2 internship. The development of a prototype intending to satisfy the requests expressed by the potential users can be divided in three main parts. First, we will discuss the overall software design and analyse the interactions between the different entities. Then we will focus on the architecture implementation on OSIRIM and we will see networks security, hardware requirements and benchmarks on the solution. Finally, we will explain differents data processing scheduled for the end of the internship that could be useful in this architecture.

Remerciements

Je souhaiterais tout d'abord remercier : Dr. François THIEBOLT qui m'a permis de réaliser ce stage et qui m'a fait participer aux projets en tant que membre à part entière de l'équipe, J'ai eu la chance de pouvoir réaliser le projet très librement grâce à sa confiance.
Pr. Olivier TESTE qui a été mon responsable de stage qui m'a aidé et accompagné dans ce projet et m'a prodigué de nombreux conseils très utiles.
Dr. Marie-Pierre GLEIZES pour m'avoir accueilli ainsi que les doctorants affiliés neOCampus-pour m'avoir accueilli intégré dans leur lieu de travail.
l'IRIT et son personnel et notamment Dr. Michel DAYDE, directeur de l'IRIT, qui m'a ouvert les portes de l'établissement pour ce stage.
Enfin, tous les enseignants de la formation SID de l'Université PaulSabatier pour la formation ainsi que le soutien qui m'a été fourni et plus précisément le MdC.CHOUQUET Cécile en tant responsable de section pour son soutien tout au long de l'année et le Pr. PINEL-SAUVAGNAT Karen pour m'avoir conseillé et soutenu dans une grande partie mon parcours universitaire.

Introduction

0.1 Contexte du stage

Ce stage de fin d'étude de 2ème année de Master en Statistiques et Informatique Décisionnelle (SID) à l'Université Toulouse III - Paul-Sabatier a été financé par l'opération neOCampus et par l'équipe SMAC de l'Institut de Recherche en Informatique de Toulouse (IRIT).

0.1.1 IRIT

L'Institut de Recherche en Informatique de Toulouse (IRIT), une des plus imposantes Unité Mixte de Recherche (UMR) au niveau national, est l'un des piliers de la recherche en Occitanie avec ses 700 membres, permanents et non-permanents. De par son caractère multi-tutelle (CNRS, Universités toulousaines), son impact scientifique et ses interactions avec les autres domaines, le laboratoire constitue une des forces structurantes du paysage de l'informatique et de ses applications dans le monde du numérique, tant au niveau régional que national. Les recherches de l'IRIT se structurent autour de cinq grands sujets scientifiques :

— Conception et construction de systèmes (fiables, sûrs, adaptatifs, distribués, communicants, dynamiques…)
— Modélisation numérique du monde réel
— Concepts pour la cognition et l'interaction
— Etude des systèmes autonomes adaptatifs à leur environnement
— Passage de la donnée brute à l'information intelligible

Six domaines d'application stratégiques matérialisent ces recherches :

— Santé, Autonomie, Bien-être
— Ville Intelligente
— Aéronautique, Espace, Transports
— Médias Sociaux, Écosystèmes Sociaux Numériques
— E-éducation pour l'Apprentissage et l'Enseignement
— Sécurité du Patrimoine et des Personnes

Ainsi qu'une action stratégique : Calcul, Masse de Données, IA

D'un point de vue organisationnel, l'IRIT est structurée en 7 départements de recherche qui regroupent les 24 équipes du laboratoire :

— Département SI : Signaux, Images (5 équipes)
— Département GD : Gestion des Données (3 équipes)
— Département ICI : Intelligence Collective, Interaction (2 équipes)
— Département IA : Intelligence Artificielle (3 équipes)
— Département CISO : Calcul Intensif, Simulation, Optimisation (2 équipes)

— Département ASR : Architecture Systèmes Réseaux (5 équipes)
— Département FSL : Fiabilité des Systèmes et des Logiciels (4 équipes)
—

sur des sujets tels que la sécurité, l'internet mobile, l'internet des objets (IoT), le Cloud, le Big Data et l'Intelligence Artificielle. L'IRIT est réparti sur 4 université et 6 sites. (cf. [1])

Sur des sujets tels que la sécurité, l'internet mobile, l'internet des objets, le Cloud, le Big Data et l'Intelligence Artificielle, dans un continuum qui s'instancie localement depuis les infrastructures jusqu'à l'aide à la décision, l'IRIT, réparti sur 4 universités et 6 sites, est l'un des rares laboratoires doté de toutes les compétences en son sein.

0.1.2 SMAC : Systèmes Multi-Agents Coopératifs

L'équipe SMAC [2] s'intéresse à la modélisation et à la résolution de problèmes dans les systèmes complexes grâce à la technologie multi-agent. Les systèmes complexes sont des systèmes composés d'un grand nombre d'entités en interaction locale dont on ne peut prévoir l'évolution autrement que par l'expérience ou la simulation. Autrement dit, malgré une connaissance parfaite des composants élémentaires du système, on ne connaît pas de programme capable de prévoir son comportement qui soit plus court que celui décrivant le système de manière exhaustive. Maîtriser le fonctionnement de tels systèmes comprenant des non-linéarités provenant de boucles de rétroaction, ainsi que le couplage entre niveaux (le macro-niveau du système est produit par les entités du microniveau, mais il contraint aussi ce micro-niveau) est l'objectif de nos travaux de recherche (illustrés par différents travaux de thèse, projets et applications).

Ces travaux s'articulent autour de six champs fortement liés les uns aux autres par les concepts d'adaptation, d'auto-organisation et d'émergence.

— Axe « auto-adaptation « . Le comportement adaptatif d'un système lui permet de réagir à l'interaction avec un environnement dynamique, soit pour continuer à réaliser sa tâche, soit pour améliorer son fonctionnement [Robertson, 2000]. Le défi est alors de concilier l'autonomie d'un système face à des situations imprévues et la maîtrise nécessaire que l'humain doit en avoir.
— Axe « auto-organisation, coopération et émergence « . L'auto-organisation est un mécanisme ou processus qui permet à un système de changer son organisation en cours de fonctionnement et sans contrôle explicite externe [A. Di Marzo, M.P. Gleizes, A. Karageorgos, TFGSO 2005]. Dans un système multi-agent, l'auto-organisation se traduit par un changement de l'organisation entre les agents. Le défi est alors de trouver les comportements locaux des agents (des lois au niveau individuel), les règles des agents qui mettent en oeuvre l'auto-organisation et qui permettent de concevoir des systèmes fonctionnellement adéquats. L'adéquation fonctionnelle est le jugement positif porté par des utilisateurs finals sur l'activité du système.
— Axe « conception et modélisation « . L'objectif de cet axe est d'améliorer et de fournir une assistance pour le développement d'applications à base d'agents coopératifs.
— Axe « coordination adaptative « . Elle permet de gérer les dépendances entre les activités des agents.

— Axe » résolution collective de problèmes « . L'objectif est ici d'étudier la manière dont les systèmes multi-agents adaptatifs apportent une solution à la résolution de problèmes multi-disciplines, multi-objectifs et multi-échelles. Cette approche permet, en environnement dynamique, de proposer une nouvelle configuration à partir d'une solution précédente en minimisant les modifications endogènes et, surtout, sans recommencer une nouvelle résolution à chaque perturbation.
— Axe » simulation « . La problématique de cet axe est d'étudier et de développer des outils et méthodes permettant d'explorer des modèles. A long terme, on souhaite notamment intégrer à une plate-forme de simulation multi-agent, les outils nécessaires à l'étude de propriétés émergentes.

0.1.3 neOCampus

L'opération scientifique neOCampus, initiée juin 2013 comporte actuellement 11 laboratoires de l'université : Cesbio, Cirimat, CRCA, Laboratoire d'écologie fonctionnelle et environnement, Irit, LA, Laas, Laplace, Lerass, LMDC, LCC. Ces laboratoires ont pour objectif de croiser leurs compétences pour améliorer le confort au quotidien pour la communauté universitaire tout en diminuant l'empreinte écologique des bâtiments et en réduisant les coûts de fonctionnement (fluide, eau, électricité...).

L'opération de recherche supportée par neOCampus a pour objet de concevoir les produits et services associés aux systèmes ambiants. L'équipement de neOCampus consiste en de nombreux dispositifs logiciels et matériels interconnectés pour le campus numérique de demain, durable et intelligent alliant matériels pédagogiques innovants, capteurs, systèmes de stockage, de localisation, de simulation et des matériaux innovants au sein de bâtiments universitaires et du campus pour augmenter la qualité de vie des usagers et réduire les consommations de fluides.

En Mai 2019, un Groupement d'Intérêt Scientifique autour de neOCampus composé de 12 institutions et 17 laboratoires de recherche a été créé. Ce GIS a pour but de mettre l'accent sur l'aspect interdisciplinaire du travail réalisé dans neOCampus. Il a été approuvé par le Conseil d'administration en mai 2019 et est actuellement en cours de signature.

0.2 Objectifs du stages

0.2.1 Différentes tâches

L'objectif de ce stage est, dans le cadre de l'opération neOCampus, de concevoir et de mettre en place une architecture de gestion de données grande échelle sous la forme d'un datalake. Ce projet a pour but de mettre en place une preuve de concept (Proof of Concept - POC) de cette architecture en vue d'un déploiement opérationnel pour la gestion des données des différents acteurs de neOCampus.

Le déploiement de ce POC a été réalisé grâce aux ressources de la plateforme de calcul présent dans l'IRIT : OSIRIM (Observatoire des Systèmes d'Indexation et de Recherche d'Information Multimédia). OSIRIM est principalement soutenue par le fonds européen de développement régional (FEDER), le gouvernement français, la région Occitanie et le Centre National de la Recherche Scientifique (CNRS). Cette plateforme est composée d'une zone de stockage d'une capacité d'environ 1 Po et d'un cluster de calcul de 1120 cœurs et 31 GPUs.

Les sources de données et acteurs en lien avec ce projets sont multiples et différents :

— les données provenant de neOCampus : composés de capteurs, de données de recherches et d'expérimentations (véhicules autonomes, données de recherches, jeux de données d'apprentissage pour l'intelligence artificielle)
— les données provenant du SGE de l'Université Toulouse III - Paul-Sabatier : données de capteurs de flux présents sur de nombreux sites, des données provenant de bases de données relationnelles
— les données provenant de l'Université Toulouse III - Paul-Sabatier : données provenant de stations météo et capteurs présents sur l'université

En plus d'être hétérogènes, ces données ont des flux et des besoins différents. Certaines sources sont des flux continus d'informations tandis que d'autres sont des données "offlines" sauvegardées à intervalle régulier (journalier). On retrouve dans ces données des données dites "sensibles" et / ou soumises à des accords de confidentialité.

Ainsi, le stage s'est concentré sur plusieurs aspects :
— la conception d'une solution logicielle pour le stockage d'informations très hétérogènes
— la mise en place d'outils de transformations de données
— la réflexion sur les outils pour un dimensionnement pouvant passer à l'échelle et pour accueillir une très grande quantité de données de sources différentes
— le développement de flux de travail (workflow) pour le traitement des données pour la validation de la valorisation des données
— la compatibilité de l'architecture avec des outils d'administration pour l'authentification et l'identification d'utilisateurs selon des polices différentes liées à des restrictions légales ou d'accord de confidentialité

0.2.2 Organisation du travail

Dû à des conditions exceptionnelles (crise du Coronavirus), le début du stage a été un peu chaotique. Il y a eu un grand nombre d'imprévus administratifs qui ont retardé le début du travail. La totalité du stage a été fait en télétravail en dehors du site de l'IRIT.

Pour le développement du projet, il s'est déroulé en 2 grandes parties :
— développement d'un POC dans des conteneurs
— déploiement de ce POC sur la plateforme OSIRIM

La soutenance se déroulant environs 3 semaines avant la fin du stage, de nombreuses tâches n'ont pas encore été réalisées et sont prévues pour après la soutenance.

Ci-dessous, le diagramme de Gantt de l'organisation du travail durant ce stage.

Toward a datalake

neOCampus

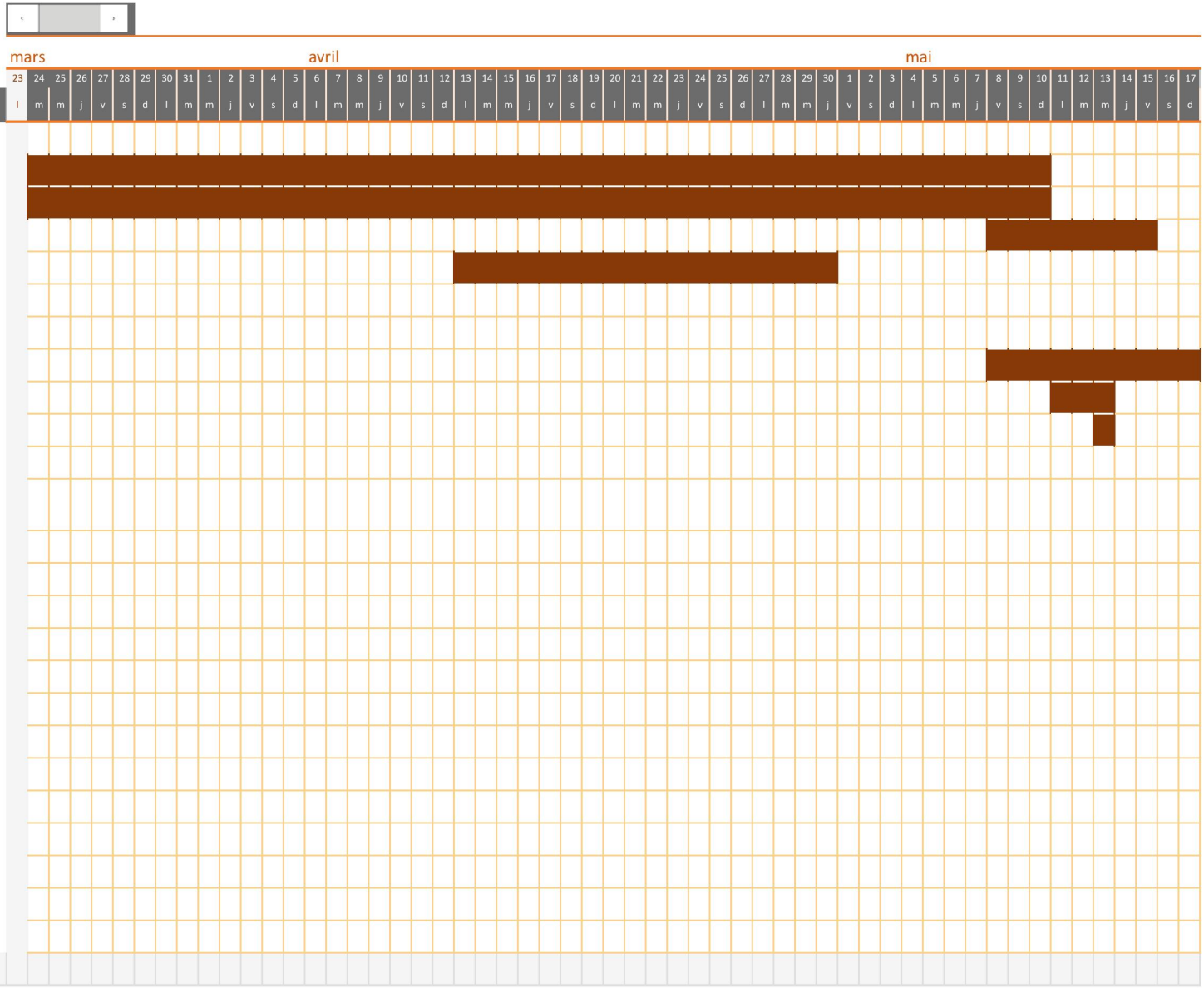
DANG Vincent-Nam

Date de début du projet : 23/03/2020

Incrément de défilement : 0

Marqueur de jalon : 0

Description du jalon	Avancement	Début	Nombre de jours
Mise en place			
Administratif		#########	50
Recherches		#########	50
Récupération des ressources		#########	10
Recherches d'outils et définition des besoins		#########	20
Rédaction et préparation soutenance		#########	22
Développement prototype en conteneurs			
Réflexion sur l'architecture		#########	40
Mise en place Openstack Swift en conteneurs		#########	5
Mise en place Apache Airflow		#########	3
Mise en place autres bases de données		#########	4
Workflows et interconnexions de services minimal		#########	24
Déploiement sur OSIRIM			
Installation Openstack Swift		#########	50
Installation Airflow		#########	10
Installation autres bases de données		#########	6
Développement des workflows		#########	35
Test et mise en place d'use cases	10%	#########	23
Développement de features			
Analyse d'image / détection automatique	30%	#########	23
analyse de document textes JSON	5%	#########	23
Augmentation des services Openstack	0%	#########	23
API REST	5%	#########	23



Toward a datalake

neOCampus

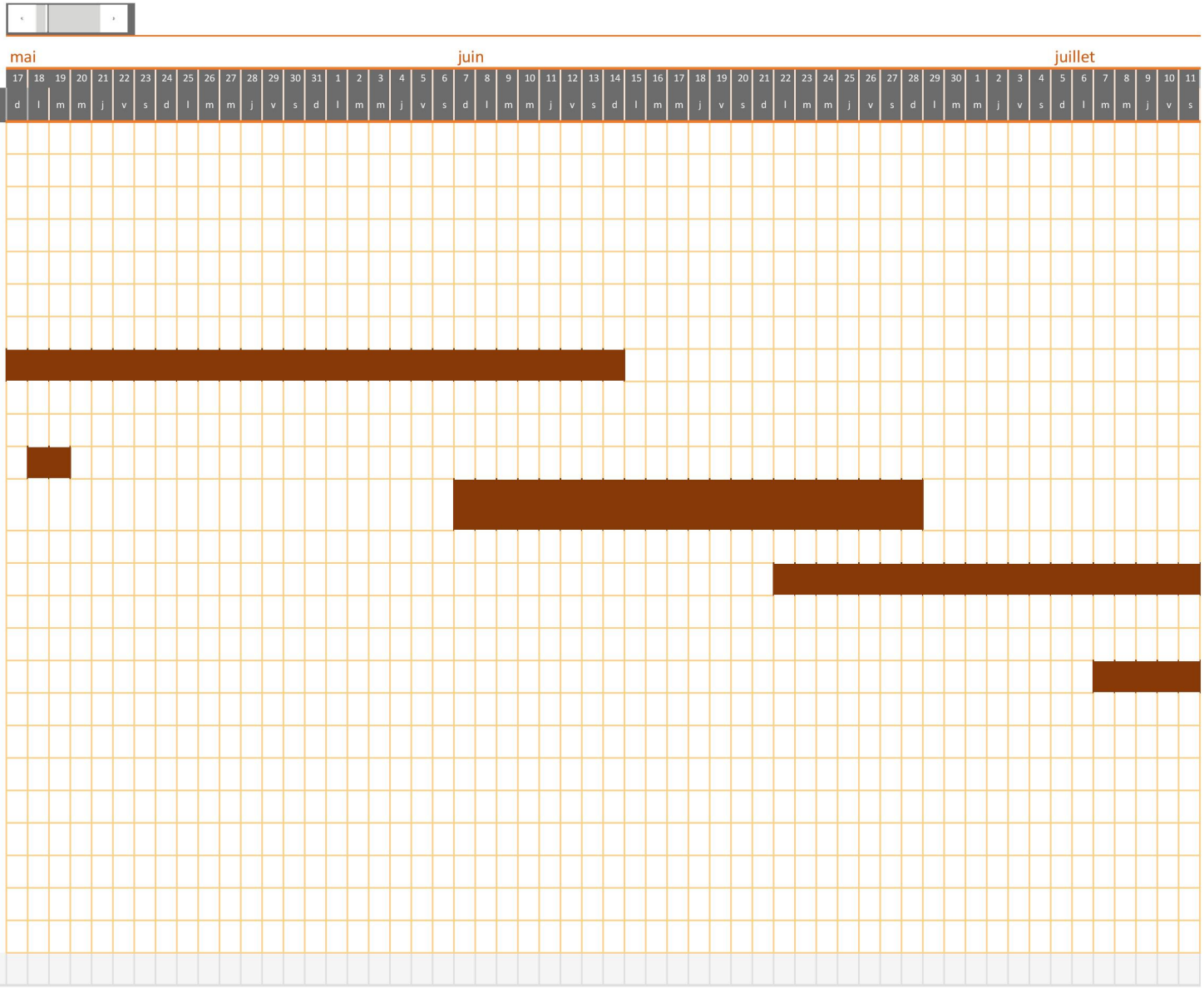
DANG Vincent-Nam

Date de début du projet : 23/03/2020

Incrément de défilement : 55

Marqueur de jalon : 0

Description du jalon	Avancement	Début	Nombre de jours
Mise en place			
Administratif		#########	50
Recherches		#########	50
Récupération des ressources		#########	10
Recherches d'outils et définition des besoins		#########	20
Rédaction et préparation soutenance		#########	22
Développement prototype en conteneurs			
Réflexion sur l'architecture		#########	40
Mise en place Openstack Swift en conteneurs		#########	5
Mise en place Apache Airflow		#########	3
Mise en place autres bases de données		#########	4
Workflows et interconnexions de services minimal		#########	24
Déploiement sur OSIRIM			
Installation Openstack Swift		#########	50
Installation Airflow		#########	10
Installation autres bases de données		#########	6
Développement des workflows		#########	35
Test et mise en place d'use cases	10%	#########	23
Développement de features			
Analyse d'image / détection automatique	30%	#########	23
analyse de document textes JSON	5%	#########	23
Augmentation des services Openstack	0%	#########	23
API REST	5%	#########	23



Toward a datalake

neOCampus

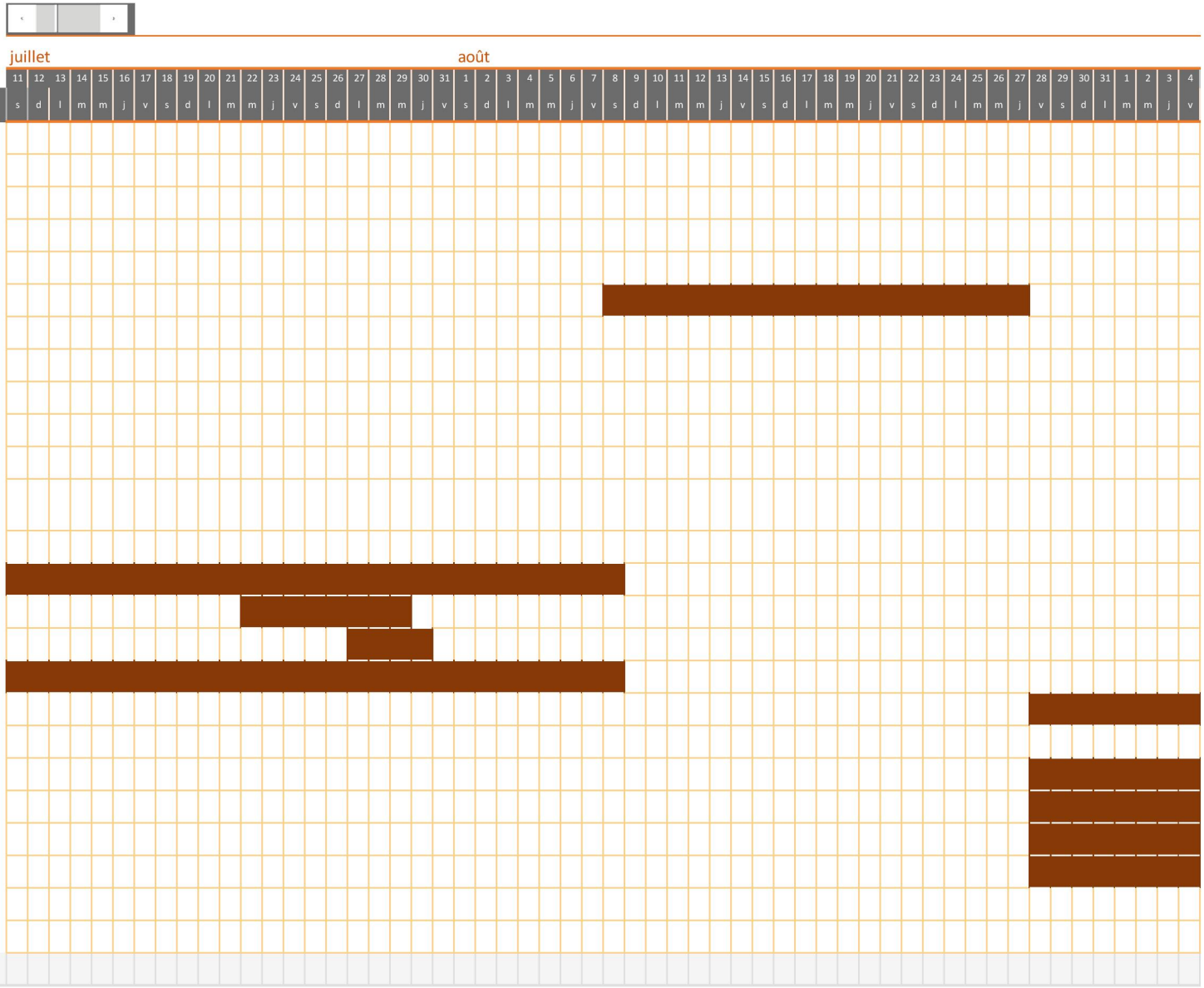
DANG Vincent-Nam

Date de début du projet : 23/03/2020

Incrément de défilement : 110

Marqueur de jalon : 0

Description du jalon	Avancement	Début	Nombre de jours
Mise en place			
Administratif		#########	50
Recherches		#########	50
Récupération des ressources		#########	10
Recherches d'outils et définition des besoins		#########	20
Rédaction et préparation soutenance		#########	22
Développement prototype en conteneurs			
Réflexion sur l'architecture		#########	40
Mise en place Openstack Swift en conteneurs		#########	5
Mise en place Apache Airflow		#########	3
Mise en place autres bases de données		#########	4
Workflows et interconnexions de services minimal		#########	24
Déploiement sur OSIRIM			
Installation Openstack Swift		#########	50
Installation Airflow		#########	10
Installation autres bases de données		#########	6
Développement des workflows		#########	35
Test et mise en place d'use cases	10%	#########	23
Développement de features			
Analyse d'image / détection automatique	30%	#########	23
analyse de document textes JSON	5%	#########	23
Augmentation des services Openstack	0%	#########	23
API REST	5%	#########	23



Chapitre 1

Solution logicielle

Tout le travail qui a été réalisé se trouve sur un dépôt de gestion de version Github [3].

1.1 Data lake : contexte

Le lac de données ou "data lake" est un concept intrinsèquement lié à celui de Big Data. Il s'agit d'une méthode de stockage de données utilisée dans un cas de très grand volume de données avec une grande variété de types de données.
On peut retrouver dans un data lake des données :

— des donnes structurées issues principalement de bases de données relationnelles
— des données semi-structurées : JSON, CSV, XML, journaux...
— des données non structurées : emails, documents, PDF...
— des fichiers de type blob : images, audio, vidéo notamment

Très proche des "data mart", des "data warehouse", le data lake est destiné à gérer tous types de données mais demande un plus grand investissement et n'est pas adapté à tout type d'entreprise. De plus, s'il est mal conçu ou qu'il n'est pas administré correctement, il peut très vite devenir un marécage de données ("data swamp"), un amas de données inutilisables et donc un mauvais investissement.
On retrouve le terme dès 1999 mais le Big Data et la volumétrie suffisante pour trouver une utilité au data lake n'étant arrivée qu'une dizaine d'années plus tard, il y a eu peu de recherches publiques dessus. Il n'existe pour l'instant aucun standard d'architecture de data lake.

Cependant, on trouve un consensus sur les caractéristiques que doit avoir un data lake :

— pouvoir gérer des données de n'importe quel format
— pouvoir gérer n'importe quelle quantité de données
— permettre de générer un niveau d'analyse sur ces données

1.2 Data lake : besoins

Du fait de la généricité de ces critères, la conception d'une telle architecture est très libre. Le contexte du stage n'offre pas de besoins clients concrets. En effet, des solutions sont déjà en place pour les besoins des projets, cependant on se retrouve pour N projets avec N solutions distinctes alors qu'une seule pourrait être mutualisée pour les besoin du laboratoire et des différentes équipes. Avec la supervision du Pr. TESTE Olivier, il a été mis en place plusieurs critères plus concrets pour la conception de cette architecture :

— Avoir plusieurs "zones" fonctionnelles différentes indépendantes : une zone d'ingestion, une zone de travail et une zone de valorisation des données.
  — une zone d'ingestion de données brute avec pour but d'ingérer de manière la plus rapide possible les données sans prendre en compte le contenu ou le format. L'objectif de cette zone est de gérer une potentielle volumétrie importante. De plus, cette zone a un objectif d'archivage des données en cas de problème dans les différentes bases de données.
  — une zone de travail (appelé aussi "workzone") qui permette d'avoir un suivi facile des opérations effectuées sur les données pour pouvoir avoir une maintenance la plus simple possible.
  — une zone de valorisation de données, qu'on peut aussi trouver sous le nom de "Gold zone". Cette zone a pour but de conserver les données sous une forme adéquate pour les analyses. C'est cette zone qui permet aux données d'être utiles, notamment pour les preneurs de décisions.
— Avoir un historique des traitements effectués ainsi que la conservation du maximum de méta-données sur les données conservées : Il est important de ne pas se retrouver dans une situation où des données ne peuvent être utilisées et transformer l'architecture en data swamp.
— Fournir une solution open-source et gratuite. L'open-source est une vision qui peut permettre aux projets de vivre leur propre vie. De plus, étant dans un laboratoire public sur un projet exploratoire, il peut être difficile d'avoir des fonds pour des licences et des solutions payantes / propriétaires.
— Avoir une grande liberté dans les transformations de données : Les données amenées à être stockées dans ce data lake ne sont pas définies, il est nécessaire d'avoir une architecture qui puisse être modulaire et s'adapter au plus grand nombre de besoins potentiels.
— Pouvoir gérer des politiques de sécurité et d'authentification : au sein de l'IRIT et de neOCampus, il existe des projets confidentiels ou qui possèdent une politique de restriction sur l'accès aux données. Il est donc nécessaire de pouvoir gérer qui accède à quelle données.

1.3 Data lake : architecture

Du point de vue logiciel, le data lake se construit comme suit (cf. Fig. 1.1) :
— La zone d'ingestion : composée de 2 entités logicielle, i.e. Openstack Swift et MongoDB. Cette zone peut être vue telle une zone tampon ("buffer") qu'on peut retrouver dans les protocoles réseau. En effet, les traitements sur une donnée peuvent être coûteux en temps et en ressource. De ce fait, si l'entrée de ces données était faite directement dans la zone de travail, on pourrait avec des données entrantes s'accumulant en entrée de la zone de travail. Une fois la taille maximum atteinte dans les files d'attente, les données sont perdues. On souhaite ne pas rencontrer cette situation.
— La zone de travail : composée d'une seule entité logicielle : Apache Airflow. Cette zone est la charnière du data lake entre les données brutes et les données transformées.
— La "Gold zone" : composée de plusieurs bases de données différentes. En production, les bases de données à installer sont définies par les besoins. Dans le cadre du stage, il a mis en place 3 bases de données : Neo4J, MongoDB, InfluxDB.

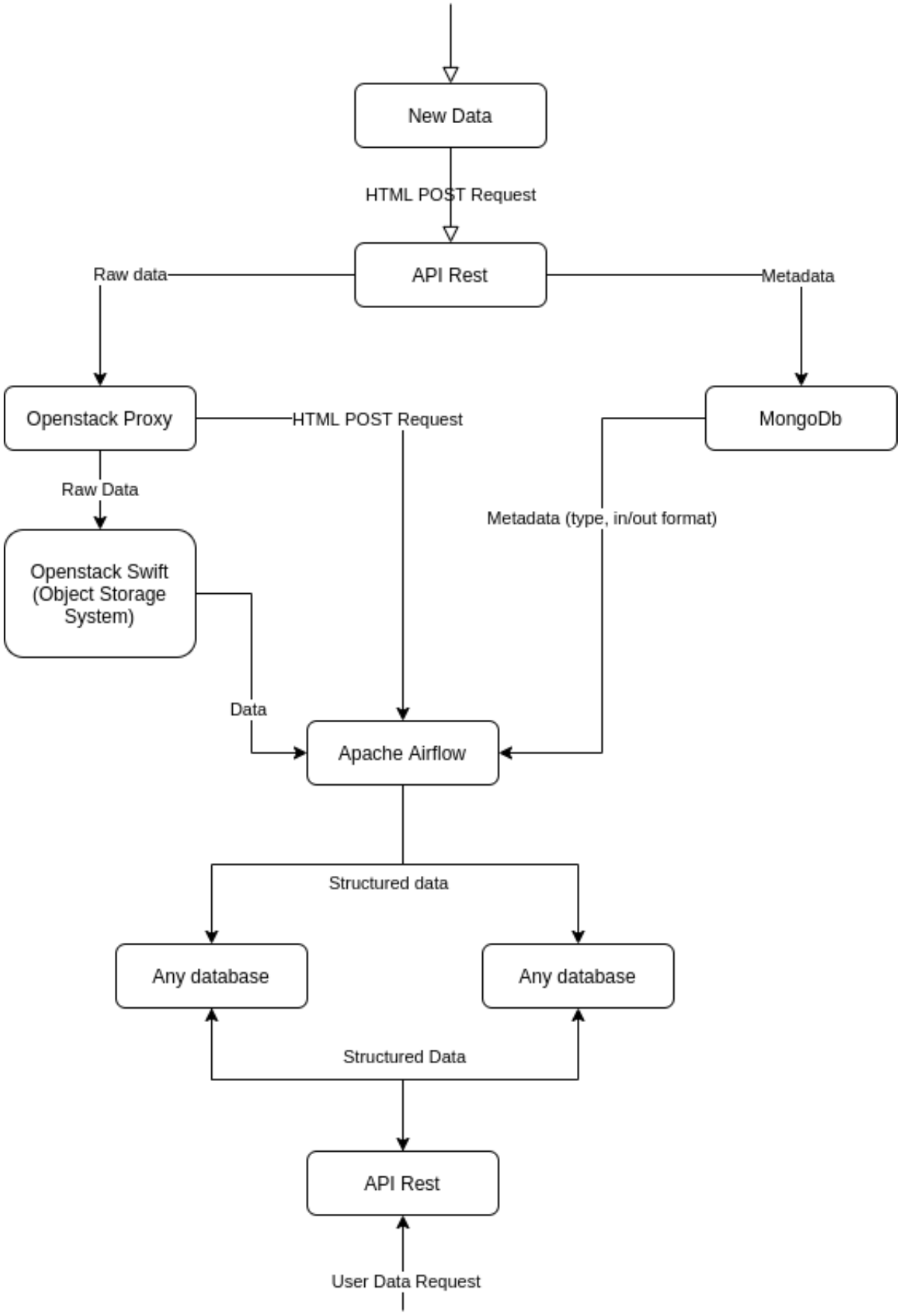


FIGURE *1.1* — Diagramme global d'interaction du data lake

Les API REST définies permettent d'avoir un contrôle d'accès. Il n'y a qu'un seul accès aux ressources demandées permettant ainsi de contrôler mieux les utilisateurs en cas de soucis. Il est possible d'ouvrir un direct accès aux bases de données en cas de besoin spécifique.

1.3.1 Data lake : Zone d'ingestion - Openstack Swift

Le Cloud Computing va souvent de paire avec le Big Data et leurs besoins sont très souvent similaires.

Openstack Swift est un service de "Object-based storage" proposé par le projet Openstack. Ce projet est un projet open-source d'outils de déploiement d'infrastructure de Cloud Computing. Le principe de l'Object-based storage est de conserver la donnée de manière agnostique. On prend la donnée telle un flux d'octets et on conserve les méta-données nécessaires à la reconstruction ou la relecture de l'information, tel que le type de données, l'encodage, etc... Le choix de Openstack Swift a été orienté par plusieurs critères.

L'approche Object-based Storage s'adapte très bien au besoin en volumétrie. La conservation de manière générique permet de sauvegarder la donnée le plus rapidement possible quelle que soit la donnée. De cette façon, n'importe quelle donnée peut être stockée uniquement en fonction de sa taille.
L'approche Cloud Computing offerte par Openstack apporte plusieurs critères importants :

- la "scalability", ou la capacité de passage à l'échelle, critère essentielle de toute solution Big Data
- l'élasticité, même si fondamentalement non-nécessaire, dans le cadre d'un POC, il est important d'augmenter ou diminuer les ressources allouées. Les besoins peuvent changer au cours du projet. De plus, on assure une vraie flébilité au niveau de ce service.
- les services d'identification et d'authentification : ces services peuvent être intégrés aux services préexistants de l'IRIT grâce notamment aux compatibilités avec les méthodes d'authentification standards telles que LDAP ou Kerberos mais il existe aussi des systèmes d'authentification propre à Openstack permettant d'augmenter le nombre de services différents d'authentification et par conséquent le niveau de sécurité mis en place.

De plus, grâce à sa communauté grande et active, Openstack propose une offre de service très grande et qui s'étoffe régulièrement. Il peut être possible d'augmenter facilement les service du data lake. On peut penser notamment aux services de télémétrie ou de traitements de données compatibles avec la plateforme Hadoop.

1.3.2 Data lake : Zone d'ingestion - Openstack MongoDB

MongoDB est une base de données NoSQL type document avec une licence open-source. Elle est utilisée en coopération avec Swift pour conserver les méta-données de chaque donnée sous forme de document JSON. On conserve ainsi les méta-données de chaque donnée dans un document propre afin de pouvoir retrouver par requêtes les documents qui nous intéressent via leurs méta-données. Il est possible de conserver jusqu'à 16 Mo de données dans un seul document. On va vouloir conserver :

- la localisation de la donnée dans le service Openstack Swift
- le type de donnée
- un historique des opérations effectuées sur cette donnée
- toute information supplémentaire importante insérée par l'utilisateur

On ne possède pas l'indépendance physique dans ce type de base de données. Cependant, il est possible d'avoir une très grande flexibilité sur la structure des documents stockés. De plus, ces bases de données sont faites pour de grandes quantités de données.

1.3.3 Data lake : Zone de travail - Apache Airflow

Apache est une fondation basée sur le principe du logiciel open-source regroupant de nombreux projets. Apache Airflow est un projet de l'Apache Incubator, il s'agit d'un logiciel de gestion de flux de travail ("workflow") écrit en Python. Ce logiciel est notamment utilisé par l'entreprise AirBnB. Il permet de gérer des flux de travail basés sur les graphes orientés acyclique (DAG - Directed Acyclic Graphs) (Cf. Fig. 1.3) et de créer facilement des pipelines ETL sur mesure. Un noeud du graphe est une tâche. Chaque flux de travail peut être programmée à intervalle régulier pour permettre la mise en place de routines.

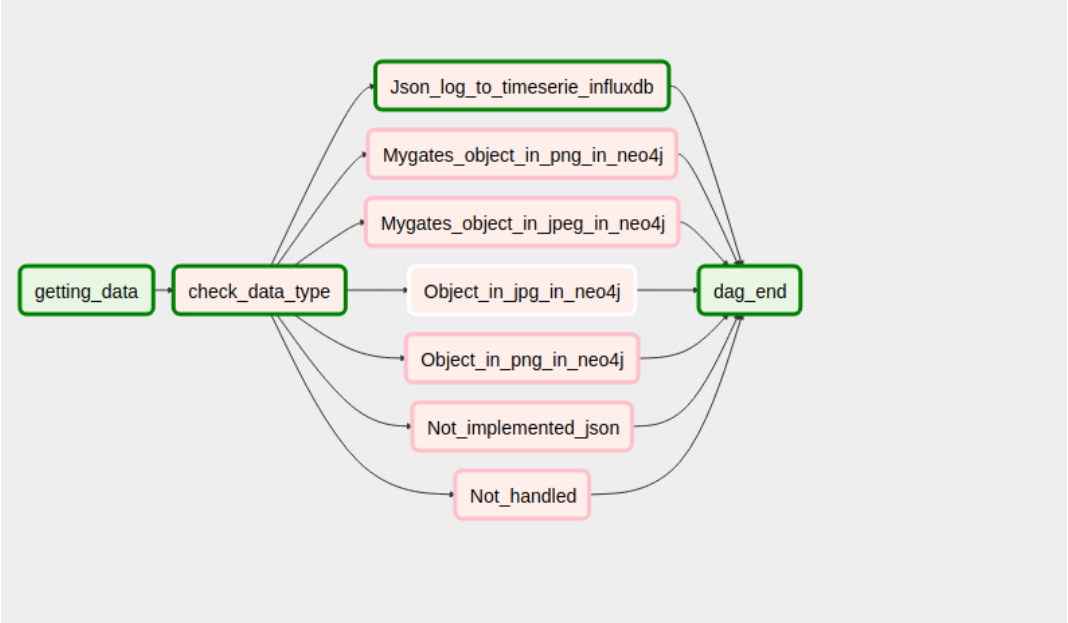


FIGURE *1.2* — Exemple de DAG dans Apache Airflow : pipeline pour une nouvelle donnée

Une tâche est définie par une classe Operator qui permet d'intégrer de nombreux outils répandus comme le Python, le shell script mais aussi des outils tels que Hadoop, Apache Spark, Cassandra ou bien même la suite AWS ou la plateforme de cloud Azur [4]. C'est cette grande diversité d'outils utilisables qui a orienté le choix de cet outil, à l'inverse d'autres outils d'intégration de données très intégrés (comme Talend à priori) ne permettant pas facilement d'intégrer des technologies autres que celle définies au départ. Airflow possède de nombreux autres avantages :

— son interface graphique pour le suivi des flux de travail **??**
— une journalisation intégrée des tâches
— un API REST permettant de s'interfacer avec tous les outils permettant des requêtes HTML

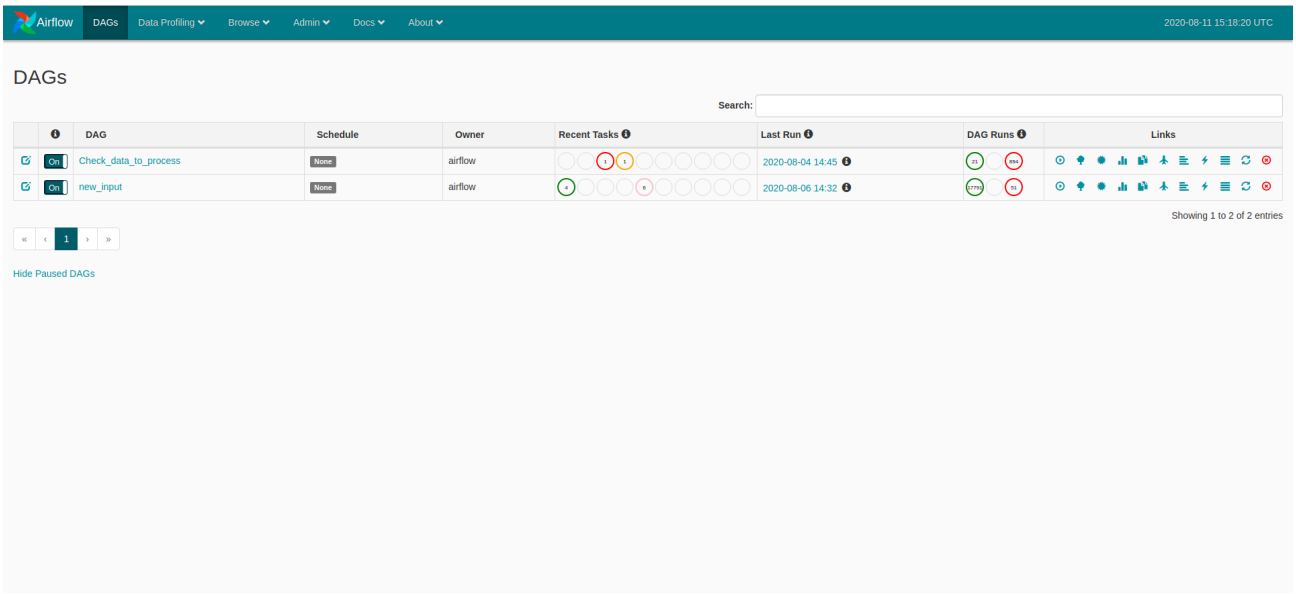


FIGURE *1.3* — Interface graphique de Apache Airflow

1.4 Data lake : la "Gold Zone"

Il a été mis en place pour le POC 3 bases de données différentes : MongoDB, Neo4J et InfluxDB.

Neo4J est une base de données noSQL type graphe. L'objectif derrière cette base de données est de fournir la possibilité de stocker des informations sur d'éventuelles images insérées. On souhaite conserver les éléments constituants de chaque image (données par l'utilisateur ou par un service de vision par ordinateur (cf. Section 3.2) grâce à de la détection automatique d'objets) permettant d'avoir par exemple toutes les photos et images comportant un ou plusieurs vélo. Une fois insérées, ces informations permettent de créer un service de recommandation d'images pour pouvoir créer des datasets d'image similaires.

neOCampus est une opération avec beaucoup de projets intégrant ou centrés sur les capteurs. On se retrouve avec une grande proportion des informations provenant de ce réseau de capteur. Pour pouvoir gérer ces données, il existe une représentation de données adéquate : les time series. InfluxDB a été mis en place pour gérer ces séries temporelles. On peut ainsi facilement visualiser ces données et les stocker de manière optimisée. Il est possible de développer un outil de visualisation très simplement grâce aux outils intégrés avec cette base de données (i.e. Telegraf et Grafana).

MongoDB est une base de données par défaut qui ne possède pas de cas d'utilisation à proprement parler. Cependant, de nombreux projets fonctionnent avec (comme le projet MyGates de neOCampus, projet de barrières connectées) et il s'agit d'une base de données très populaire. De plus, originellement les relevés de capteurs dans neOCampus étaient stockés dans MongoDB. Il s'agit donc d'une solution facile pour la migration de ces données tout en conservant le fonctionnement des applications.

Une base de données relationnelle a été envisagée, notamment pour les données du SGE (cf. Section 3.0.1). Cependant, aucune donnée n'étant disponible pour un potentiel test à l'heure actuelle, ce type de base de données n'a pas été mis en place. Les bases de données relationnelles sont des bases de données historiquement utilisées dans de nombreux services et leur utilité semble très probable.

Chapitre 2

Implémentation dans Osirim

2.0.1 Architecture Réseau et sécurité

Le data lake est hébergé dans la plateforme OSIRIM. Cette plateforme se trouve dans le réseau de l'IRIT et pour y accéder il faut passer plusieurs sécurité qui lui permettent d'assurer un meilleur contrôle d'accès sur les données. En effet, pour accéder il faut (cf Fig. 2.1) :

— Avoir accès au réseau de l'IRIT, celui ci protégé dans un premier temps par les pare-feux de l'Université Toulouse III - Paul-Sabatier, déjà conçus avec des contraintes de sécurité forte. L'accès à ce réseau demande d'y être authentifié avec un identifiant différent de celui de l'université, augmentant ainsi la sécurité. De plus, ces accès sont restreints à un certain nombre d'applications grâce aux ports limités qui y sont autorisés (22 (ssh - dans peu de cas), 80 (http), 443 (https)) qui intègrent leurs propres systèmes de contrôle d'accès.

— Avoir un accès à la plateforme OSIRIM. En effet, les ressources de la plateforme OSIRIM sont en accès restreint. Il est donc nécessaire d'avoir un compte authentifié pour accéder aux ressources.

— Avoir un accès au réseau virtuel local du data lake. Cet accès ne se fait pour l'instant que par SSH protégé par un compte sur les machines virtuelles sur whitelist. Il est possible en plus d'utiliser les clés SSH (en cryptage RSA avec des clés de chiffrage (par défaut 2048 bits)) pour s'assurer d'un plus haut niveau de sécurité.

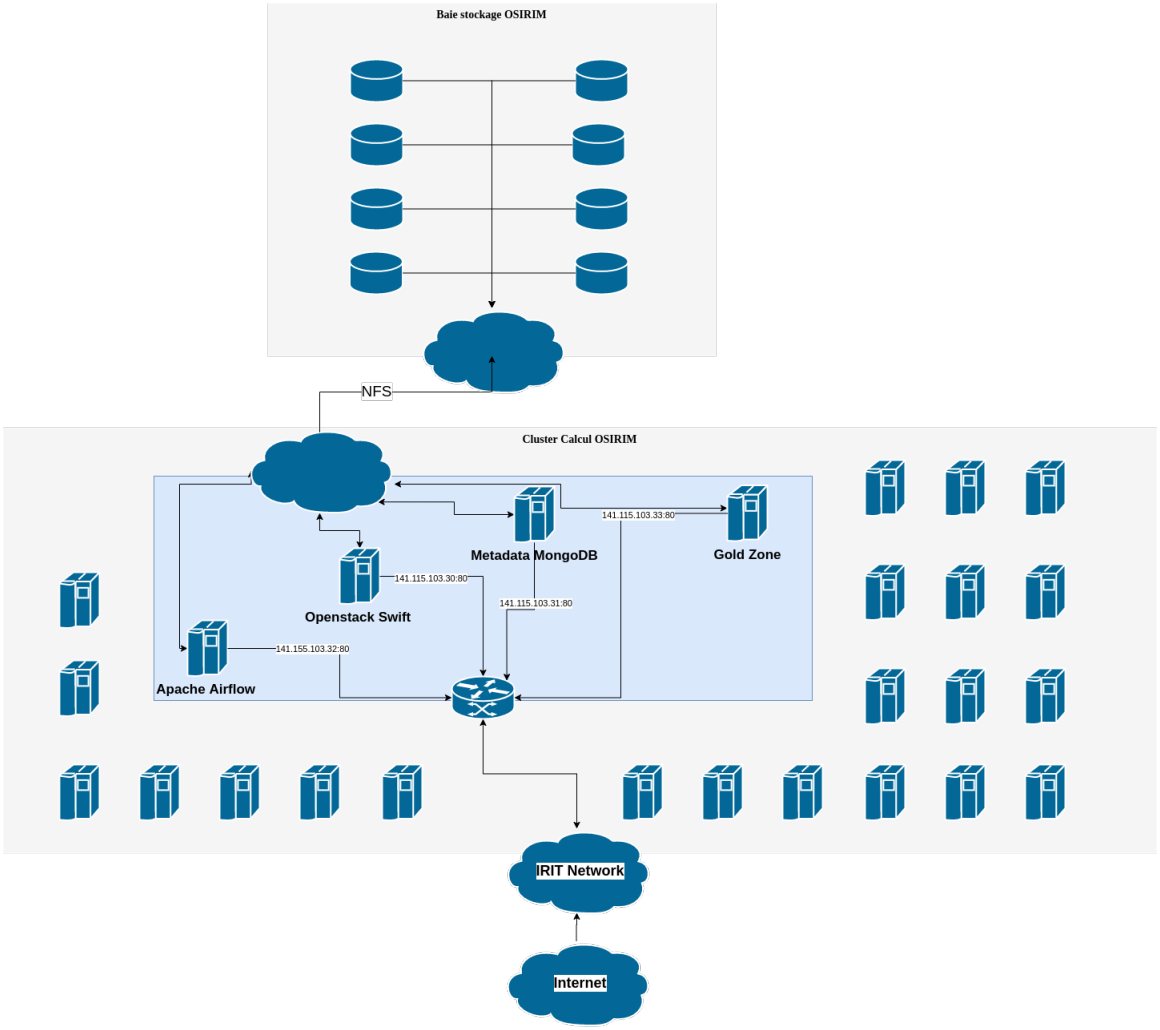


FIGURE 2.1 — Diagramme réseau de l'implémentation dans Osirim

De plus, les données sont stockées sur une baie de stockage NFS (Network File System) qui n'est pas en accès libre. Il est donc nécessaire d'accéder aux données par les machines virtuelles du data lake. Il est nécessaire d'avoir donc un accès physique pour le réseau de l'université ou un accès physique au réseau de l'IRIT (ou à distance en utilisant le proxy de l'IRIT), une authentification sur le réseau de l'IRIT, une authentification sur le réseau OSIRIM et une authentification sur le réseau privé du data lake avec 3 passages de pare-feux différents ce qui réduit considérablement les probabilités d'accès non autorisé aux données pour une personne extérieur aux projets. De plus, pour les personnes internes aux différents projets de l'IRIT / OSIRIM, il est nécessaire d'avoir une authentification valide pour les conteneurs Openstack Swift liée à une politique de sécurité DAC et / ou RBAC, selon les besoins des projets. Il est possible de mettre en place, en plus, un serveur Kerberos permettant d'augmenter le taux de fidélité des authentification mises en place.

Si toutefois, un individu arrivait à s'introduire dans le réseau sans pour autant accéder aux conteneurs Swift, d'autres mécanismes sont possibles :

— du point de vue transfert de données : il est possible de mettre en place du chiffrement sur les communications ainsi que des lignes de communications sécurisées au niveau de la couche transport par le protocole SSL. On augmente ainsi l'intégrité et la confidentialité des données pour un éventuel intrus dans le réseau privé du data lake.

— du point de vue applicatif : chaque base de données et service logiciel intègre des mécanismes propres d'authentification, d'identification et de chiffrement limitant ainsi les possibilité d'attaque de type "Man in the middle". De plus, les API REST sont déjà en place avec HTTPS.

2.0.2 Chemin d'une donnée

L'ingestion d'une donnée se fait par un seul point d'accès : l'API REST d'entrée. Il y a 4 étapes principales avant d'être valorisée (Cf. Fig. 2.2).

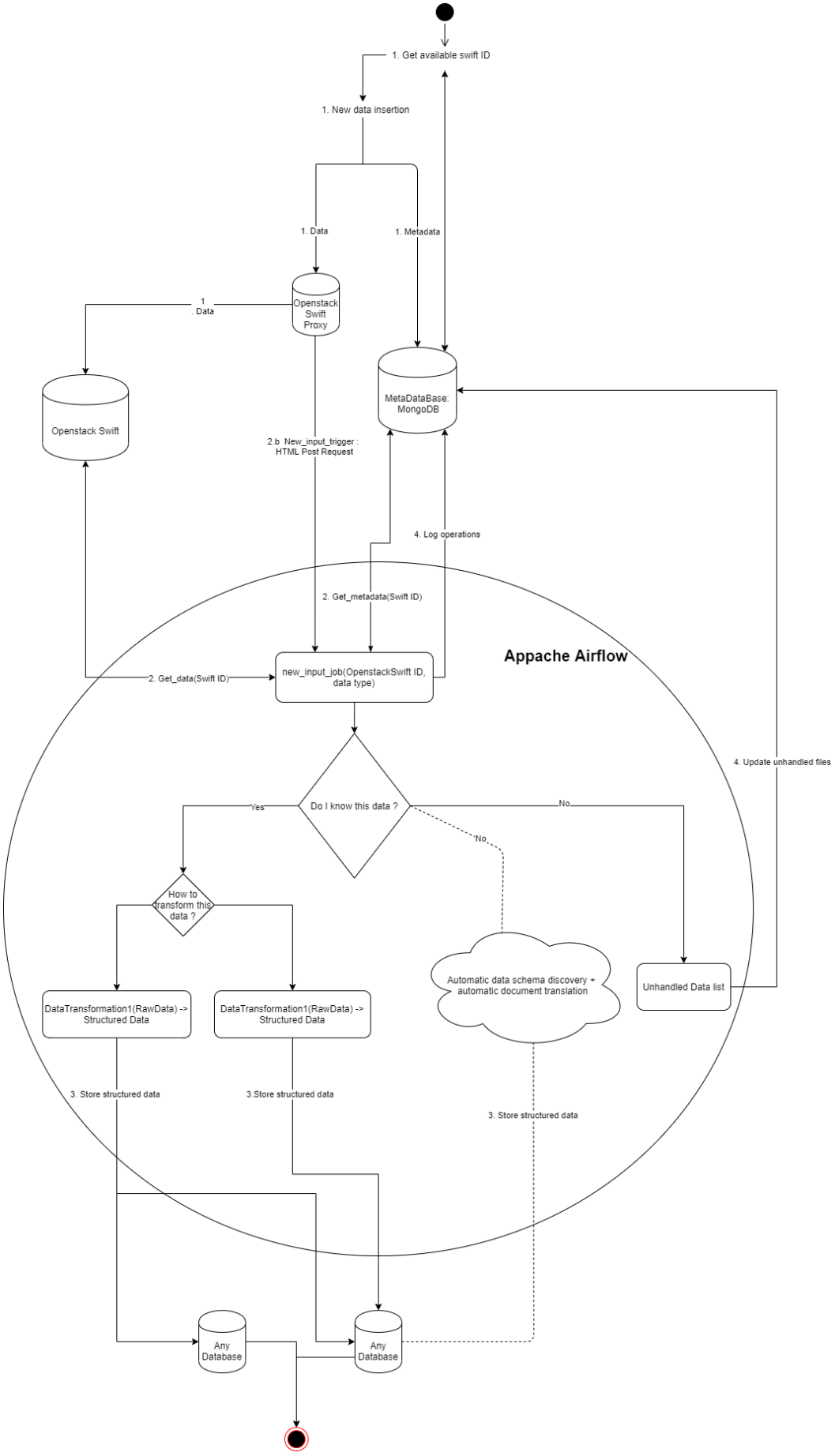


FIGURE 2.2 — Diagramme de séquence de l'intégration d'une donnée

2.0.2.1 Etape 1 : Ingestion brute

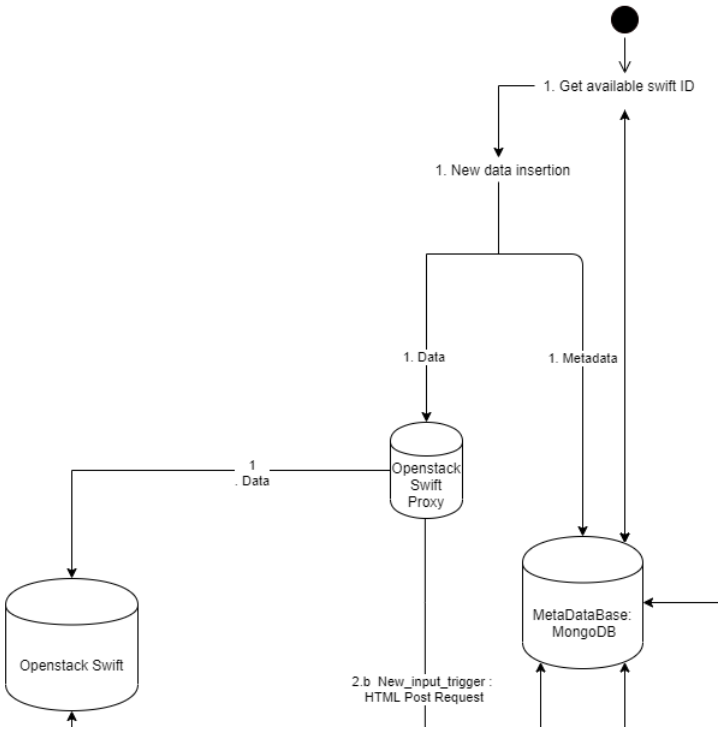


FIGURE 2.3 — Diagramme de séquence de l'intégration d'une donnée : Etape 1

La données ainsi que ses méta-données sont stockées de manière naïve dans Openstack Swift et dans la base de méta-données MongoDB. Même si on peut considérer conceptuellement que ces insertions sont simultanées, il n'y a pas été trouvé de solution pour transformer cette insertion en transaction.

- On récupère dans MongoDB le numéro d'identifiant par lequel on renomme la donnée.
- On tente d'insérer la donnée renommée dans le conteneur adéquat Openstack Swift. On conserve :
  - la donnée sous forme de tableau d'octets
  - le type de données : le type est conservé avec le standard IETF "MIME Type" [6]. Ce type est inséré de 2 manières : soit manuellement par l'utilisateur soit automatiquement en utilisant les extensions de fichier dans le cas contraire.
- Si l'insertion dans Openstack Swift s'est correctement déroulée, on insère dans MongoDB les méta-données correspondantes.
  - le type de données
  - l'utilisateur et le conteneur Openstack Swift dans lequel la donnée est stockée
  - une information "application" qui permet de donner une description textuelle de l'application pour laquelle cette donnée est utile. Si elle n'est pas donnée, il est conservé le conteneur et l'utilisateur. On supposera que le conteneur est le nom de l'équipe / projet auxquel appartient l'utilisateur, à l'instar des groupes UNIX mis en place sur OSIRIM.
  - une date d'insertion et une date de dernière modification
  - une liste des opérations réussies et une liste des opérations échouées

— un champ "other_data" permettant de stocker un sous document contenant n'importe quelle donnée jugée nécessaire par l'utilisateur

On souhaite conserver toutes les données, même si elles sont redondantes. Cette zone de stockage a aussi pour but d'archiver les données. L'un des intérêts est de pouvoir reconstruire toutes informations générées à partir de ces données brutes en cas de problème d'une base de données.

2.0.2.2 Etape 2 : Traitement de la donnée

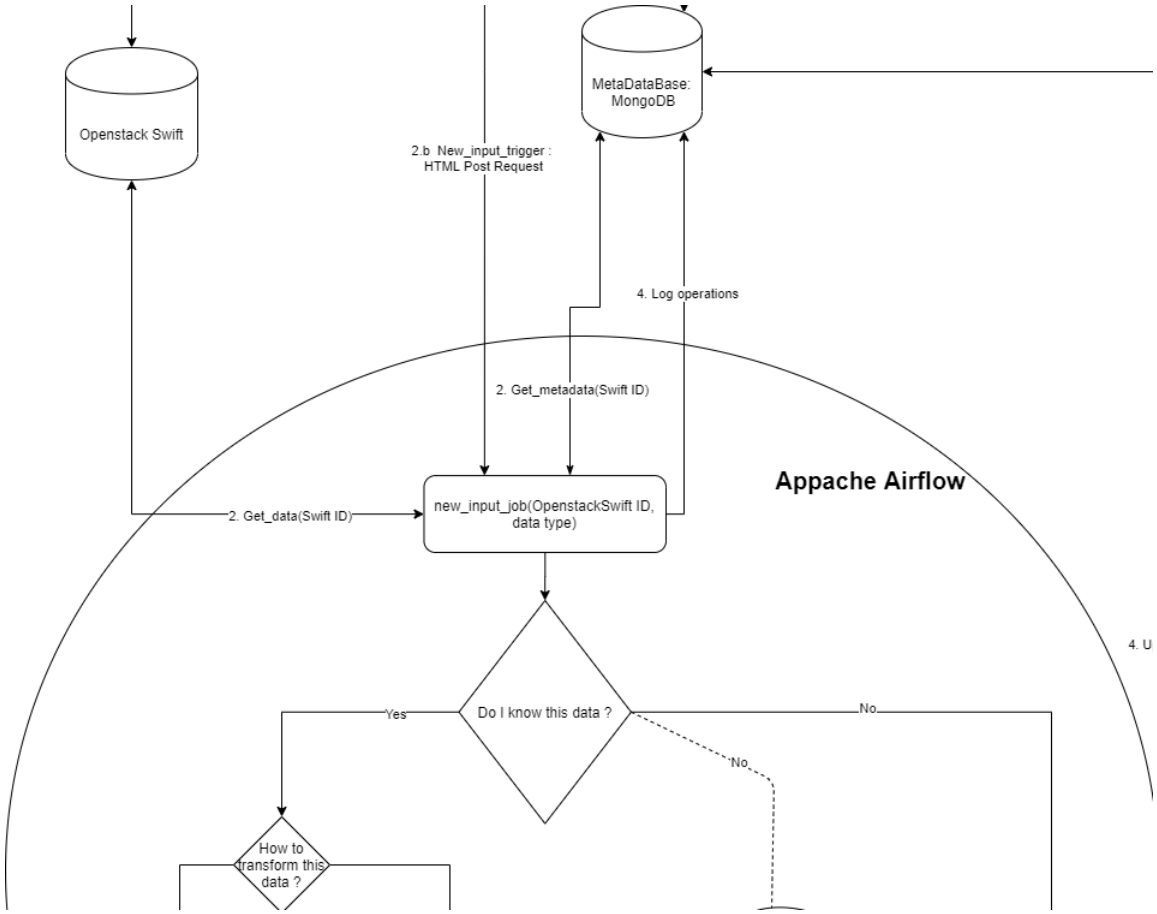


FIGURE 2.4 — Diagramme de séquence de l'intégration d'une donnée : Etape 2

Une fois les données insérées dans la zone d'ingestion, les données sont traitées par les pipelines de traitements mis en place dans Apache Airflow. Il y a 2 possibilités pour le lancement d'un travail par Airflow :

— un déclencheur web ("web trigger") est lancé par le service de proxy de Openstack Swift qui, à l'insertion d'une donnée, lancera le pipeline de nouvelle donnée ("new_input") avec l'identifiant Swift correspondant.

— un minuteur programmé à intervalle régulier ira vérifier la liste des données à traiter dans MongoDB. Pour toute nouvelle donnée à traiter, ce pipeline lancera le pipeline de nouvelle donnée avec l'identifiant Swift correspondant.

Le pipeline de nouvelle donnée (cf. Fig. 1.3) vérifie le type de donnée qui vient d'être insérée grâce aux méta données. Puis, en fonction du type puis du conteneur, effectuera le traitement spécifié pour cette donnée. Il est ainsi possible de traiter 2 données de même type avec des traitements différents en fonction des besoins de chaque projet. Si aucun pipeline n'est défini pour un projet, le pipeline par défaut est utilisé. Il est possible d'enchaîner autant de tâches voulues pour traiter une donnée. Chaque donnée ne prend qu'un seul chemin. Il est nécessaire que ce DAG soit considéré comme un automate déterministe (et ne pas avoir de branchement aléatoire ou avec des comportements non contrôlé et impactant le branchement vers la prochaine étape) pour que le suivi des traitements soit faisable et fiable.

Les traitements qui ont été développés pour le déploiement permettent de traiter 2 types de données :

— les données semi-structurées type JSON avec les relevés de capteurs de neOCampus qui permettent de transformer les relevés de capteurs MongoDB en series temporelles dans InfluxDB.
— les images de type PNG et JPG en ajoutant pour chaque image les éléments présents dans cette image insérées sous forme de graphe dans Neo4J

2.0.2.3 Etape 3 : Conservation des informations

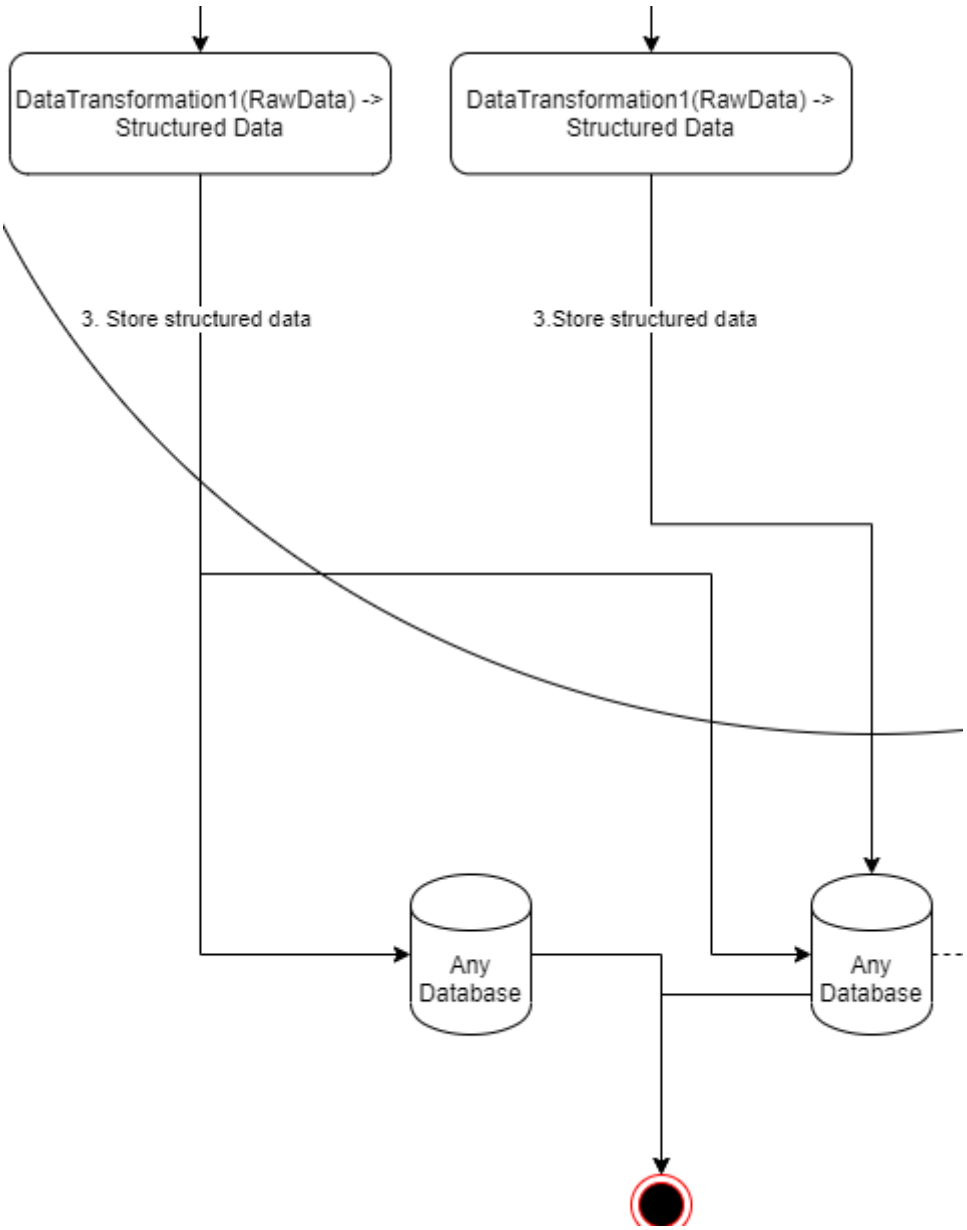


FIGURE 2.5 — Diagramme de séquence de l'intégration d'une donnée : Etape 3

Pour les preneurs de décisions, cette étape est la plus cruciale. En effet, c'est cette étape qui permet, à partir des données, d'avoir des informations qui leur permettent d'orienter leurs choix pour une entreprise. On stocke l'information transformée dans la base de données adéquate. Il existe de nombreux types de bases de données différentes et pour des besoins différents. On peut penser :

— aux bases de données relationnelles utilisant le langage SQL (i.e. MariaDB, mySQL, Oracle Database, etc..) avec un but applicatif. De nombreuses applications fonctionnent historiquement avec des SGBD Relationnels.

— aux bases de données noSQL type Document (i.e. MongoDB, CouchDB, etc..) adaptés aux documents semi-structurés type JSON ou XML qu'on retrouve très largement dans le web.
— aux bases de données noSQL type Graphe (i.e. Neo4J, OrientDB, etc..) pour les données fortement liées permettant les traitements se basant sur la théorie des graphes
— aux bases de données type clé/valeurs et type colonnes (i.e. Redis, BigTable, HBase, etc..) qui peuvent être utilisées d'un point de vue applicatif pour les recherches dans de grands jeux de données

Cependant, il existe de nombreux autres types de base de données pour des besoins différents. Il est possible d'intégrer n'importe quelle base de données grâce à la flexibilité de Apache Airflow.

2.0.2.4 Etape 4 : Journalisation

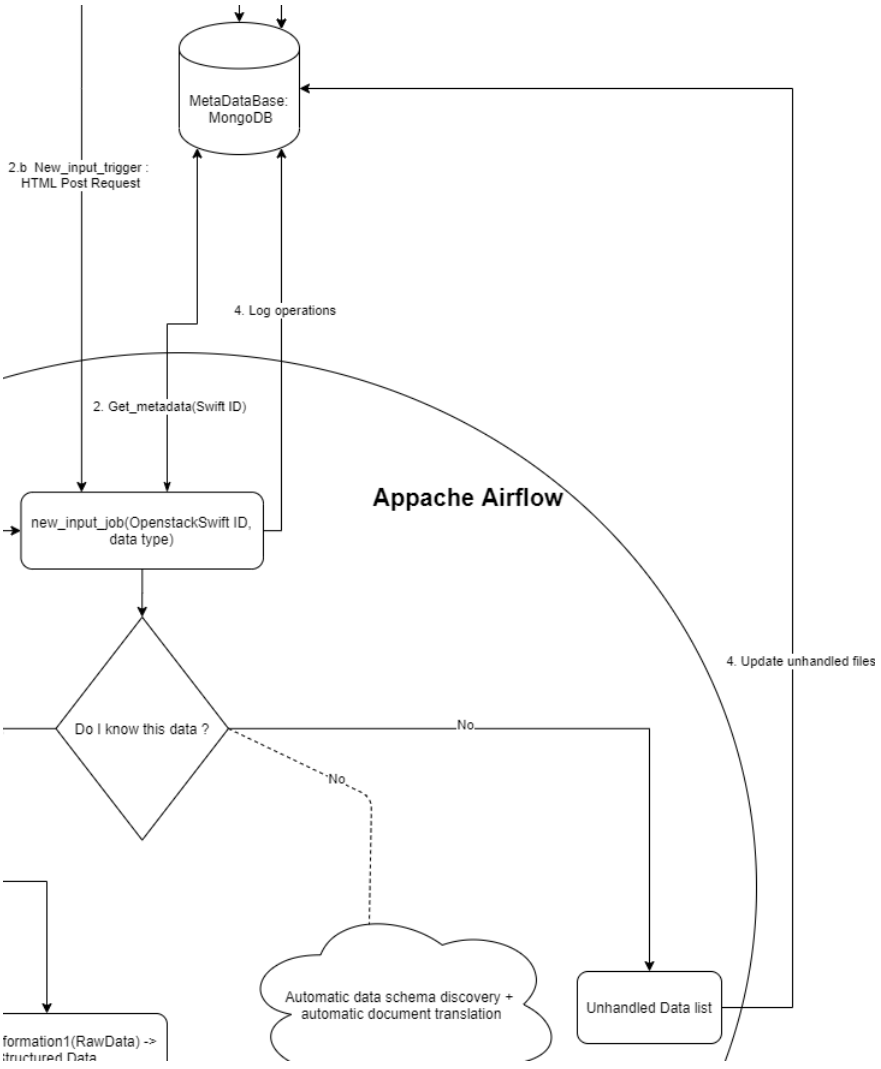


FIGURE 2.6 — Diagramme de séquence de l'intégration d'une donnée : Etape 4

Une fois la valorisation des données effectuée, on inscrit dans la base de méta données MongoDB les opérations réussies et échouées pour chaque donnée ainsi que les les opérations effectués dans le document stockant les opérations effectuées pour ce jour. On obtient grâce à cette étape un suivi possible des données et on réduit les probabilités d'avoir des données inutilisables.

2.0.3 Réflexion sur le matériel

Dans le cadre du stage, l'implémentation s'est faite sur des machines virtuelles sur la plateforme OSIRIM. Le choix du matériel n'étant pas possible, il ne devient possible que d'avoir des réflexions sur le matériel et sur l'architecture de ce matériel. Nous nous intéresserons aux entités critiques que sont Openstack Swift et Apache Airflow. Les autres bases de données seront beaucoup moins sollicitées que ces 2 entités, les probabilités de sous-dimensionnement sont moins grandes.

2.0.3.1 Réflexion sur le matériel : Openstack Swift

Openstack Swift est implémenté sur une machine virtuelle avec :
— 8Go RAM
— 4 Coeurs
Cependant, dû aux fonctionnement des plateforme de cloud et de virtualisation, il est difficile de savoir quelles sont les caractéristiques de la RAM ou des CPU (pouvant être cadencé à 1.6Ghz, 2.8Ghz ou 3Ghz). En effet, les ressources allouées ne sont pas forcément constamment les mêmes, il peut y avoir un roulement des CPU alloués (principe de vCPU) ainsi qu'un surcoût significatif dû au partage des ressources par l'ordonnanceur de la plateforme qui ont un impact sur les capacités difficile à estimer. Le stockage étant effectué sur une baie NFSv3, on a un accès au stockage se faisant par un réseau d'interconnexion. On augmente la latence entre le serveur et son stockage. De plus, NFSv3 n'étant pas la version la plus à jour de NFS, l'accès au attributs étendus (xattr) n'est pas disponible. Il est donc nécessaire de passer par une surcouche en créant un périphérique virtuel ("loopback device") avec un système de fichier permettant l'accès à ces attributs, ici XFS, un système 64-bit journalisé de haute performance, très adapté aux entrées-sorties parallèle par sa conception. De plus, Openstack Swift n'est pas adapté aux techniques de virtualisation de stockage distribué de données sur plusieurs disques (la technologie RAID) communément utilisé sur les baies de stockage réseau. L'architecture matérielle sur Osirim n'est pas très bien adapté à Openstack Swift.

Openstack Swift fonctionne sous forme de plusieurs services en coopération. Ces services tournent tous sur la même machine mais devraient avoir leur propres machines d'hébergement dans un cadre d'optimisation matériel.
— les proxy-server : ils examinent toutes les requêtes et redirigent les demandes vers les bons serveurs. Ils sont responsables de nombreuses autres actions configurables dans leur pipeline. Ils sont les points d'entrée dans le service.
— les objects servers : ils sont responsable de la majorité du travail à effectuer, l'écriture des données sous forme de blob.
— les containers servers et les account server : ils sont responsables du stockage des informations sur les conteneurs ainsi que les comptes sous forme de base de données SQLite database. Solicités pour des opérations bien moins lourdes, ils pourraient être hébergés sur la même machine.

La majorité du travail est réalisé par les proxy-server ainsi que par les "objects servers". Pour le proxy-server, il est possible mettre plusieurs serveurs pour augmenter les capacités d'entrée et de répartir le travail via un load balancer (pour avoir une répartition équitable des charges). Ces problématiques sont des problématiques d'architecture réseaux et peuvent être résolues grâce à de nombreuses solutions (matériel propriétaire ou par virtualisation (SDN - Software Defined Networks)). Pour les objects servers, le besoin matériel est différent. On se place dans un contexte où on souhaite réduire au maximum les erreurs dues à un manque de serveur disponible. L'objectif de ce type de plateforme est de fournir une haute disponibilité pour un potentiel grand nombre de clients. Un mécanisme de réplication ("replication") et de fragmentation ("sharding") des informations est mis en place pour s'assurer d'une meilleure disponibilité ainsi qu'une bonne résilience aux pannes en mettant de la redondance sur les données ainsi que sur les serveurs pouvant répondre aux requêtes sur une donnée.
Les architectures type Hadoop ou les architectures à mémoire partagée pour les base de données parallèles sont adaptées à cette problématique. On souhaite avoir de nombreux serveurs avec leur mémoire propre et leur stockage propre quitte à avoir des performances modérées sur chaque serveur. De cette façon, chaque serveur possède un accès son espace de stockage avec le moins de latence possible. La technologie Hadoop semblerait être compatible avec Openstack Swift qui peut s'interfacer en tant que surcouche de Hadoop. L'idéal pour l'interconnexion de ces serveurs serait de mettre en place des liens type "Infinite band" permettant de réduire au maximum les latences d'échanges. Cependant, ce type de connexion est cher.

2.0.3.2 Réflexion sur le matériel : Apache Airflow

Apache Airflow fonctionne sur des briques logicielles indépendantes pour la parallélisation des tâches et de nombreuses solutions sont disponibles. Parmi ces solutions, Celery executor, une solution distribuée d'ordonnancement de tâches. Les demandes sont encore différentes des services de Openstack Swift. Pour ce service, on souhaite une grande parallélisation des tâches ainsi qu'une grande quantité de mémoire vive. Les traitements pouvant être lourd, il est nécessaire de pouvoir fournir suffisamment de mémoire à ces processus. Le besoin en stockage est très faible il n'est seulement nécessaire d'avoir du stockage pour les méta-données et quelques dizaines de giga-octets seront largement suffisants pour longtemps. Dans l'absolu, on souhaite avoir un grand nombre de coeurs haute performance et une mémoire vive assez grande et rapide. Ainsi, Apache Airflow s'adapte bien aux plateformes de virtualisation car il est aisé d'ajouter un grand nombre de vCPU et une mémoire en fonction des besoins et de pouvoir réaliser un dimensionnement sur mesure en plus de pouvoir s'abstraire de la partie orchestration de plusieurs serveurs distincts, ce qui pourrait être nécessaire pour augmenter le nombre de coeurs. Apache Airflow supporte Hadoop et il pourrait être intéressant de mettre en place une plateforme Hadoop pour faire fonctionners Apache Airflow, permettant ainsi d'avoir une très grande parallélisation des tâches et en permettant d'intégrer Hadoop aux outils disponibles dans les tâches de traitement.

2.0.4 Tests et benchmarks

2 étapes cruciales de la mise en place de l'architecture du data lake sont les partie test et benchmark. La majorité des entités logicielles mises en place sont déjà testées par les développeurs des applications. Cependant, pour l'assurance d'un bon fonctionnement de l'architecture et dans un souci de suivi de qualité, il est nécessaire de mettre en place des tests d'intégration système pour s'assurer du bon fonctionnement des communications entre entités. On peut penser à 2 solutions pour mettre en place ces tests.

— développer les tests dans un DAG Apache Airflow qui serait programmé à intervalle régulier (tous les jours ou toutes les semaines, en fonction des besoins et de la fréquence d'ajout de service).
— mettre en place des outils de la mouvance DevOps. Un worflow type (cf. Fig. 2.8) est :
  — mettre en place tout d'abord un dépôt de gestion de version Git qui permettrait de conserver les fichiers de l'architecture logicielle.
  — mettre en place un serveur de build (type Jenkins) qui permettra d'effectuer le build et les tests d'intégration sur l'architecture à chaque changement du dépôt Git. Cette étape peut être effectuée grâce aux technologies de virtualisation via des conteneurs logiques (cf. Section 2.0.5) qui permettra de tester les fonctionnalités en s'abstrayant de la partie physique.
  — une fois le build effectué, les modifications sont déployées sur les serveurs de production par le serveur de build.

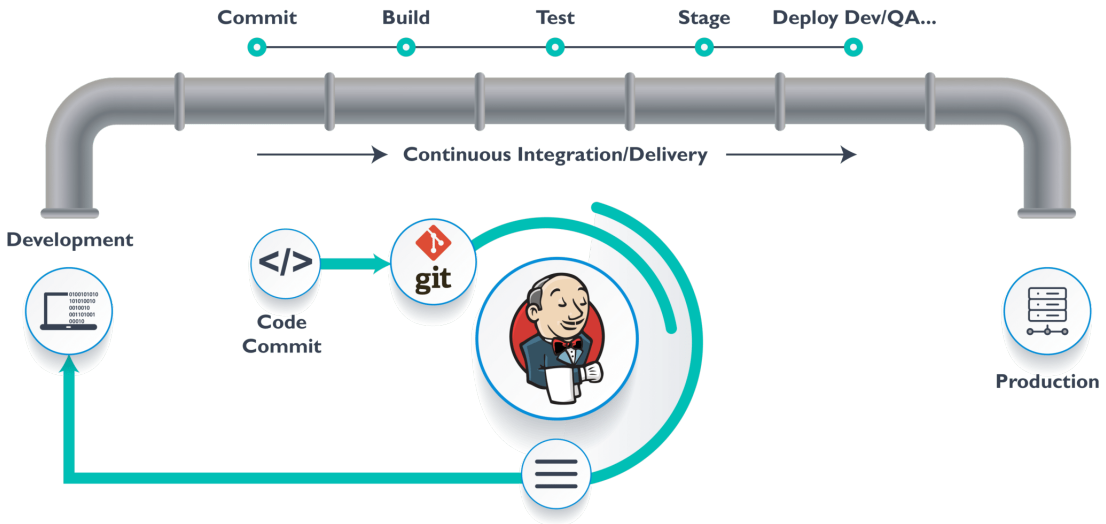


FIGURE 2.7 — Schéma d'un pipeline CI/CD grâce à Jenkins

Les benchmarks permettent d'avoir une idée de la quantité de données qui peut être gérée par le data lake. L'objectif est de générer des flux de données à insérer dans le data lake et de surveiller les indicateurs de performance. Cela donne une estimation des capacités de l'architecture en fonction de l'implémentation et du matériel fourni. On souhaite simuler plusieurs types de trafic :

— des trafics volumineux, afin de tester la capacité de la zone d'ingestion à gérer les gros volumes de données
— des trafics non volumineux mais nombreux, afin d'estimer le nombre de clients maximum en simultané pour la zone d'ingestion
— des trafics de données réelles permettant de tester les capacités de la zone de travail ainsi que la "Gold zone". L'objectif est de générer le trafic le plus gros possible qui ne fasse pas exploser les files d'attentes de traitement de données.

Des outils fournis par Openstack permettent de réaliser des benchmarks pour Swift. Il est possible de régler la taille des données insérées, le nombre d'objet ainsi que le nombre de connexion en simultannée.

```
[vdang@co2-dl-swift swift-bench]$ swift-bench swift-bench_SAIO.conf
swift-bench INFO Creating containers with storage policy: gold
swift-bench INFO Auth version: 1.0
swift-bench INFO Auth version: 1.0
swift-bench INFO 7 PUTS [0 failures], 3.3/s
swift-bench INFO 212 PUTS [0 failures], 12.4/s
swift-bench INFO 407 PUTS [0 failures], 12.6/s
swift-bench INFO 660 PUTS [0 failures], 13.9/s
swift-bench INFO 869 PUTS [0 failures], 13.9/s
swift-bench INFO 1000 PUTS **FINAL**
                [0 failures], 14.3/s
swift-bench INFO Auth version: 1.0
swift-bench INFO 563 GETS [0 failures], 281.4/s
swift-bench INFO 4839 GETS [0 failures], 284.6/s
swift-bench INFO 9093 GETS [0 failures], 284.1/s
swift-bench INFO 10000 GETS **FINAL**
                [0 failures], 283.9/s
swift-bench INFO Auth version: 1.0
swift-bench INFO 312 DEL [0 failures], 155.9/s
swift-bench INFO 1000 DEL **FINAL**
                [0 failures], 155.7/s
swift-bench INFO Auth version: 1.0
```

Ce benchmark a été réalisé cette configuration :

- 10 clients
- une taille d'objet de 1000 octets
- jusqu'à 1000 insertion d'objets
- jusqu'à 10 000 lecture d'objets

La taille des objets n'influence pas les vitesses en sortie. Cependant, si on augmente la quantité de données insérées (en augmentant la taille des objets), on augmente considérablement le nombre d'échecs d'insertions. Une partie des échecs semble être due à des mécanismes de sécurité de limitation du nombre de connexions sur une période pour éviter des attaques comme des attaques basées sur le scan de ports TCP. Cependant, on arrive à voir qu'il est aisément possible d'insérer au moins 1000 données d'1 Ko par 10 clients différents à une vitesse de 14.8 entrées par secondes. Des benchmarks plus poussés demandent d'être effectués pour estimer correctement les capacités. De plus, il est nécessaire de mettre en place des benchmarks sur toutes les entités logicielle ainsi que sur l'architecture dans sa globalité et voir les capacités de l'architecture sur un chemin complet d'ingestion d'une donnée.

Il existe une deuxième solution pour estimer les capacités de l'architecture par une étude statistique des files d'attente en utilisant la théorie des files d'attente. Ce type d'étude permet de modéliser et mettre en place un dimensionnement de l'architecture en fonction du besoin. Cependant, ces études demandent du temps et une grande connaissance du matériel. Le temps imparti par ce stage ne permet pas de mettre en place cette étude.

2.0.5 POC dans des Conteneur Docker

La virtualisation et plus précisément les outils d'isolation logicielle sont des outils très intéressants pour la conception de solutions logicielles à moindre coût. En effet, il devient possible sur une unique machine d'exécuter plusieurs entités logicielles, chacune étant isolée logiquement des autres pour simuler plusieurs machines physiques. Docker est un outil de conteneurisation logicielle se basant sur les Cgroups Linux, fonctionnalité apparue dans la version 2.6.24 du noyau Linux permettant une isolation logicielle des processus exécutés dans un environnement GNU/Linux en utilisant les espaces de nommage Linux. Pour valider, tester mais aussi implémenter sans impact sur les serveurs de production, un système conteneurisé de l'application a été développé et composé de :

— un conteneur pour Openstack Swift
— un conteneur MongoDB
— un contenur Apache Airflow
— un conteneur par base de données de la "Gold zone"

Ces conteneurs ont été créé grâce à Docker et ont été orchestrés entre eux grâce à docker-compose. 2 réseaux distincts ont été créés, un réseau "northbound" qui permet de lier les entités de la zone d'ingestion avec Apache Airflow et un réseau "southbound" qui relie Apache Airflow à toutes les bases de données de la "gold zone". Conceptuellement, Apache Airflow est la charnière entre donnée brute et information, ou donnée traitée. On s'assure ainsi de n'avoir aucun échange entre les 2 zones de stockage de données et il devient bien plus simple de suivre l'historique de modification des données avec Apache Airflow comme point de passage obligatoire.

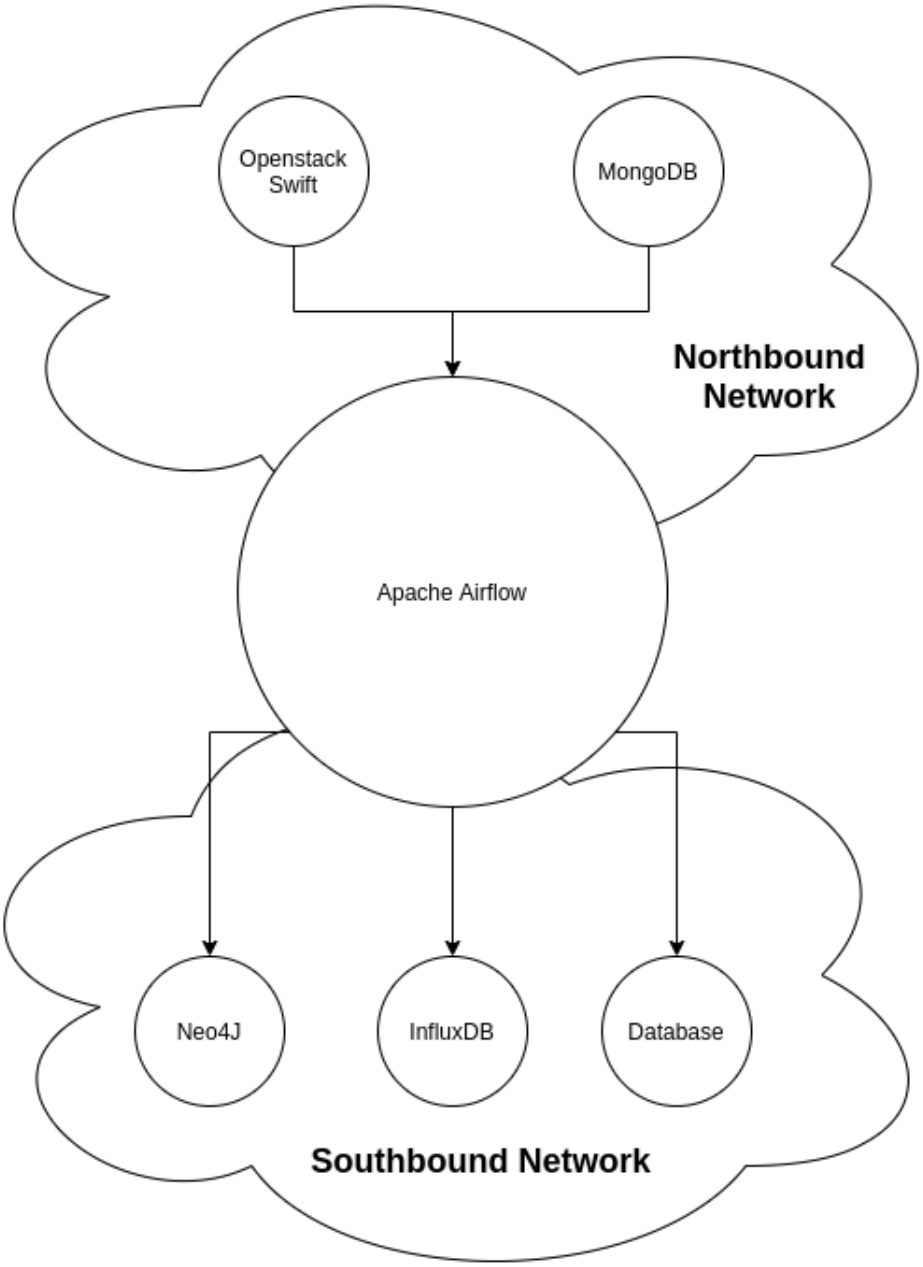


FIGURE 2.8 — Orchestration des conteneurs entre eux

L'utilisation de Docker a plusieurs avantages :

— Il est facilement possible de tester et développer des features et services et voir les interactions entre services. En effet, grâce à l'utilisation d'un gestionnaire de version et d'un serveur de build (cf. Section 2.0.4), il est possible de mettre un pipeline automatique de développement et de test de features et de services en assurant de ne pas dégrader la version de production.

- Cette virtualisation permet d'avoir une portabilité de l'architecture sur n'importe quelle machine fonctionnant dans un environnement GNU/Linux. Étant donné qu'un conteneur se base sur les outils noyaux et embarque les outils nécessaires à son fonctionnement à partir du noyau, un conteneur Linux fonctionnera tout aussi bien sur une distribution Centos que sur une distribution Ubuntu.
- Le déploiement sur des serveurs physiques est possible en quelques instant en utilisant les conteneurs et la technologie Docker. En effet, si les machines hôtes du data lake venaient à être arrêtées (pour maintenance, panne ou autre), une mise en place très rapide d'une solution de secours temporaire sur d'autres machines pour continuer à maintenir les applications fonctionnelles est possible.

La version conteneurisée de l'architecture se trouve dans la branche "docker" du dépôt Github.

Chapitre 3

Traitement de données

Le rendu du rapport et la soutenance de stage s'effectuant avant la fin du stage, la majorité des outils de traitement de données n'ont pas intégrés. La priorité du stage était de fournir une architecture viable. Cependant, une majorité des outils disponibles seront mis en place d'ici la fin du stage.

3.0.1 Intégration de données : données relationnelles du SGE

Le Service de Gestion et Exploitation de l'université Toulouse III - Paul-Sabatier possède de nombreux capteurs pour les différents flux de l'université et de nombreux autres sites. Parmi ces flux, on compte notamment l'eau et l'électricité. Ces données sont à accès restreint et demandent d'avoir un accord de confidentialité. Ces données possèdent une volumétrie quotidienne conséquente. L'objectif est de pouvoir intégrer ces relevés de données de manière offline, i.e. non pas sous forme de flux continu. L'architecture du SGE déjà mise en place est très intégrée et fonctionne grâce à des logiciels propriétaires qui font office de sur-couche sur une base de données relationnelle réduisant la lisibilité de ces données. De plus, les méta-données correspondantes à ces données ne sont pas facilement accessibles et il est difficile de retrouver à quoi correspond quelle donnée.

Pour répondre à cette problématique, la meilleure solution est de stocker les archives quotidiennes des transferts de données et d'importer telles quelles ces données dans une base de données relationnelle pour pouvoir conserver le fonctionnement des applications. De cette manière et grâce au stockage brut des données dans Openstack Swift, il est possible à posteriori de retravailler sur ces données pour pouvoir les valoriser au mieux en fonction des besoins tout en conservant les besoins opérationnels.

Cet échange de données est une négociation sur le long terme dû au caractère "sensible" de ces données et n'a pas encore été effectué. Cependant, le projet a été une des considérations pour la conception du data lake et l'architecture proposée est capable d'ingérer des données.

3.0.2 Analyse de texte : traduction automatique de JSON

Dans le monde du web sémantique, des réseaux de capteurs et plus largement d'Internet, le texte est une grande part des données qui transitent dans les projets informatiques. Parmi les formats, on retrouve très largement des données sous forme de JSON ou de XML pour le transfert des informations. En effet, le JSON est un format textuel dérivé de la notation objet du langage JavaScript, langage largement utilisé dans le web. Il a été normalisé par l'IETF [7] ainsi que par l'ECMA [8] et réutilisé dans de nombreux autres contextes pour les échanges de données grâce à son caractère complet et simple d'utilisation.
Dans le cadre plus précis des réseaux de capteurs, le format JSON est très largement utilisé avec le protocole MQTT [9] qui permet de faire transiter des messages grâce à ce format. Dans ces documents, on retrouve toujours au moins :

— une estampille temporelle
— un identifiant de capteur
— une valeur mesurée

En effet, même si ces informations ne sont pas des informations nécessaires au protocole MQTT, elles sont nécessaires à minima pour pouvoir stocker des informations cohérentes à partir du moment où plus de 1 capteur est présent dans la même base de données. La clé par laquelle est conservée chaque donnée n'est pas normée ou standardisée, il est donc possible d'un réseau de capteurs ou d'un projet à un autre de retrouver pour la même information 2 clés différentes. L'hypothèse est que ces clés font partie d'ensembles de termes finis et disjoints (au moins en majorité). Ces termes ne sont pas pris au hasard et ont pour but d'être humainement compréhensibles donc doivent être au moins sémantiquement disjoints. On retrouve par exemple :

— pour les valeurs de relevés : value, values, payload, mesure, measure, etc...
— pour les identifiants de capteurs : ID, capteurID, sensorID, sensID, num_capteur, etc...
— pour les estampilles temporelle : date, datemesure, time, timestamp, etc...

Ainsi, pour une même valeur, il est possible de se retrouver avec une dizaine de mises en forme différentes ne permettant pas d'avoir des informations qui soient retrouvables programmatiquement. L'objectif de ce projet est de passer de N documents avec des structures différentes à 1 document unique, agrégat de ces N documents (cf. Fig. 3.1). Ce projet est plus orienté recherche qu'ingénierie et le temps imparti pour ce stage n'a pas permis de mettre en place un début de solution viable. Cependant de nombreuses pistes à explorer sont envisageables. Ces approches ont été motivées par des sujets de recherches sur des sujets différents mais qui pourraient être transposés au problème présent ou bien des discussions avec des personnes travaillant dans le monde de l'intelligence artificielle et sur l'analyse d'information.

Une première approche est sur un travail de traduction des clés :

— en se basant sur les expressions régulière, la création d'un glossaire, l'utilisation de POS-tagging [10] et de typage sémantique
— en se basant sur réseaux de neurones pour traduire les clés en entrée vers une clé représentant le groupe sémantique auquel appartiennent les clés en entrée avec :
  — les Recurrent Neural Network [**?**]
  — les Word Embedding et d'embeddings BERT afin d'obtenir l'espace vectoriel permettant de distinguer chaque catégorie sémantique de clé
  — les Bag-of-words
  — une association des différents algorithmes de Machine Learning et de Deep Learning (comme par exemple une adaptation de l'algorithme de Viterbi) pour obtenir une classification de chaque terme

— en se basant sur des algorithmes de classification et de clustering

Une deuxième approche est sur un travail sur les valeurs et non pas sur les clés :
— en utilisant les réseaux de neurones :
  — avec une partie des outils de la première approche
  — une classification des valeurs et des types de données (LSTM, réseaux d'attention, réseaux récursif, etc...)
— des outils d'intelligence artificielle comme les arbres de décision

Une dernière approche est de travail sur la structure du document et d'utiliser la sémantique de la structure des documents JSON :
— LSTM et réseaux d'attention pour apprendre la structure des documents et inférer une classe en fonction de la position de l'information et de la structure du document [11]
— adopter une approche de l'étude du web sémantique et utiliser les ontologies. Il devient possible de mettre en place des algorithmes de la théorie des graphes et / ou avoir une approche différente des solutions précédentes

Cependant, la plupart des solutions demandent un grand nombre de données. Même si les données dans neOCampus représentent une centaine de millions d'entrées, il n'y a pas de variété dans les clés permettant d'avoir un dataset sur le travail des clés possible à l'heure actuelle. De plus, ces approches ne sont pas triviales à mettre en place et le temps n'est pas suffisant pour un stage de 6 mois que cela soit pour la collecte des données ou bien de la mise en place d'une solution viable.
De plus, même si cela semble contre-intuitif, il se peut que l'hypothèse de base d'avoir des ensembles sémantiques de termes disjoints soit fausse.
Trouver une solution à cette problématique permettrait de pouvoir intégrer les données de n'importe quel réseau de capteurs. Le data lake deviendrait beaucoup plus adaptatif avec un besoin humain réduit. De plus, en généralisant l'approche, on pourrait se permettre d'intégrer automatiquement un grand nombre de documents et d'informations semi-structurés.

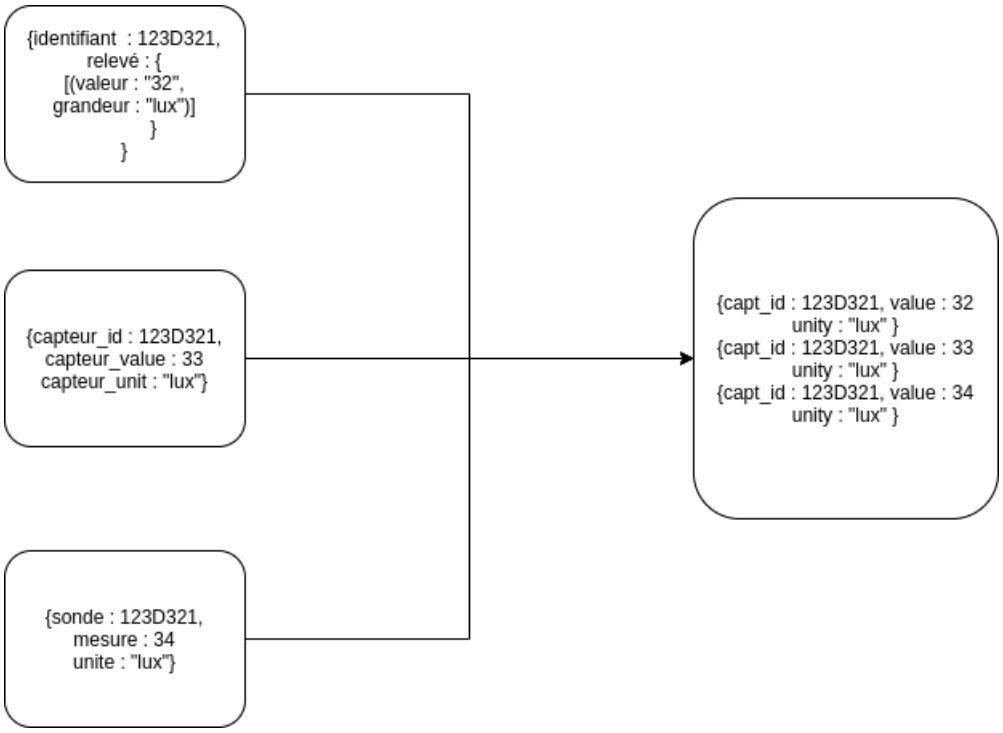


FIGURE *3.1* — Schéma de transformation des documents de relevés de capteurs

3.0.3 Traitement de données : mise sous plusieurs formes

Il peut être intéressant d'avoir, pour une seule donnée, plusieurs formes différentes de cette même donnée. Prenons en exemple un relevé de capteur. On supposera que ces relevés comportent :

- le relevé de la valeur de ce capteur ainsi que l'estampille temporelle de ce relevé
- le type de capteur et l'unité
- l'emplacement de ce capteur avec un identifiant de bâtiment, de salle et / ou emplacement dans la salle

Il peut être intéressant d'avoir les relevés de capteurs sous forme de série temporelle pour pouvoir visualiser ces données de manière pertinente. Cependant, il peut être intéressant d'avoir une représentation de ces données sous forme de graphe. On peut ainsi obtenir la liste des capteurs les plus utilisés ou les plus liés à un bâtiment.

En effet de la même information, on peut obtenir :

- des informations sur la donnée mesurée et avoir la possibilité de mettre en place des détections d'anomalies.
- des informations sur le type de capteur qui revient le plus souvent et dans quel bâtiment. On obtient ainsi les endroits et / ou les données les plus relevés, des pertinences de relevés d'informations, etc...

3.0.4 Analyse d'image : outils disponibles

L'analyse d'image et le domaine de vision par ordinateur est un domaine qui est étudié depuis de nombreuses années. Il a été développé des outils qui peuvent être mis en place dans les pipelines de traitement d'image de Apache Airflow pour permettre d'ajouter des features intéressantes.

3.0.4.1 Convolutionnal Neural Network et Detectron - Facebook AI software system

Facebook est un acteur important de la vision par ordinateur et a développé de nombreux outils disponibles pour faire du traitement d'image. Detectron 2 [12] est un réseau de type Mask-RCNN pré-entraîné développé par Facebook AI. Cet outil implémente l'état de l'art de l'analyse d'image par CNN. Cet outil permet à la fois de mettre en place sur une image de la détection d'objet parmi une liste de 80 classes (provenant du dataset COCO). Il est possible de "fine-tune" le réseau pour lui faire détecter une liste de classe personnalisée.

FIGURE 3.2 — Exemple de fonctionnement de Detectron2 provenant du github du projet[12]

Detectron 2 permet de faire de la reconnaissance, de la détection et de la segmentation sur des images avec de grandes résolutions. Par défaut, les images sont redimensionnées en 800x800 3.2. Une grande partie des CNN qu'on trouve fonctionnent avec des images jusqu'en 256x256 par souci d'efficacité. Il est possible de détecter aussi des caractères et de l'entraîner à reconnaître des langues. L'objectif derrière est de pouvoir automatiquement détecter les objets d'une image et ajouter ces informations dans Neo4J avec pour but de créer un outil de recommandation d'images pour création de datasets.

Comme tous les réseaux de neurones à l'heure actuelle, la validation des modèles est compliquée et le comportement n'est pas complètement déterministe. Il peut être donc nécessaire à l'issue du traitement d'avoir une supervision humaine des résultats pour s'assurer de leur validité. Cependant, comme tous les datasets demandent d'être retravaillé avant d'être utilisés, les anomalies ont une grande probabilité d'être détectées dans le pipeline de traitement de données quel qu'il soit.

3.0.4.2 Analyse d'image sans réseaux de neurones

Avant que les réseaux de neurones ne deviennent une solution viable avec l'augmentation des performances de calcul du matériel informatique et des cartes graphiques, de nombreux outils ont été développés pour faire du traitement d'image. Les descripteurs SIFT ou les descripteurs de Fourier sont des outils plus anciens pour la mise en production qui permettent de traiter des images. Il est notamment possible avec ces outils de détecter des points d'intérêts communs entre 2 images. Il existe des outils permettant de travailler sur les formes et les couleurs. En transformant les images en descripteurs, il devient possible de les comparer et de retourner une liste d'images ayant des descripteurs proches et donc des points communs.

3.0.5 Analyse de données : BU Hygrométrie et contrôle statistique des procédés

La bibliothèque universitaire de l'université Toulouse III - Paul-Sabatier comporte de nombreux documents dans leur bâtiments d'archives considérés comme fragiles. L'hygrométrie y est surveillée avec beaucoup d'attention. Si le taux d'hygrométrie augmente ou diminue trop, de nombreux documents pourraient être abîmés voire détruits.
Des capteurs neOCapteur ont récemment été mis en place pour suivre l'hygrométrie des salles d'archives. Dans une volonté de contrôle du procédé de gestion de l'hygrométrie, le but est de mettre en place un contrôle statistique des procédés pour permettre de détecter des dérives de cette mesure de l'hygrométrie avant qu'elle ne dépasse un seuil critique. Ce contrôle statistique permettrait de mettre en place des alertes sur le niveau d'hygrométrie afin de prévenir tout changement non contrôlé de cette mesure.
Ainsi, il sera nécessaire de développer une interface graphique comportant plusieurs cartes de contrôle et de mettre en évidence des alertes en suivant les différentes règles de Shewhart. Cette interface graphique sous forme de tableau de bord (avec les outils fournis par les technologies web afin de s'assurer d'une portabilité sur de nombreux appareils pouvant accéder au web) devra reprendre :

— l'affichage des courbes de suivi des différents capteurs
— les cartes de contrôle nécessaires
— un récapitulatif visuel des alertes

Le data lake propose tous les outils nécessaires à la mise en place du stockage et l'analyse de ces informations. Il deviendrait nécessaire de mettre en place un nouveau serveur pour l'hébergement de l'interface graphique qui soit accessible de l'extérieur du réseau.

Conclusion

Bilan

A l'issue du stage, une architecture de data lake opensource a été mise en place sur la plateforme Osirim et une version dans des conteneur a été développée. Cette plateforme s'oganise autour de 3 services distincts :

- Openstack Swift : un service de stockage objet fourni par la suite de logiciel de plateforme de cloud Openstack
- Apache Airflow : un service d'ordonnancement de tâches qui sert de l'équivalent des ETL dans les entrepôts de données
- MongoDB : une base de données pour la gestion des méta-données et de la journalisation des tâches

L'implémentation n'en est qu'au stade de preuve de concept et la plateforme sur laquelle est mise en place le data lake n'est pas adapté pour tous les besoins de l'application. Les cas d'utilisations n'étant pas très clairement définis, le développement du projet s'est fait de façon plutôt exploratoire. Les pipelines de traitements de données restent à mettre en place en fonction des besoins des futurs projets hébergés dans le data lake.

Perspectives

La soutenance de stage se déroulant avant la fin de mon stage, il reste encore du travail à réaliser sur la partie traitement de données et validation de l'architecture par tests. Un CDD de 1 an pour un enrichissement des services de la plateforme OSIRIM m'a été proposé dans la continuité de ce stage et pourra permettre de continuer ce projet d'architecture de data lake et d'ajouter de nombreux services (Openstack et traitements des données à priori) à l'architecture déjà en place. Un réel travail sur le choix du matériel pourrait être intéressant s'il est possible.

Bilan personnel

Dans le cadre de ce projet, j'ai appris beaucoup sur la gestion des données, du point de vue logiciel comme matériel. En effet, j'ai réalisé une grande réflexion sur les choix des outils en fonction des besoins qui m'avait été exprimés ce qui m'a permis d'y voir plus clair sur le stockage de données dans une vision Big Data et ses contraintes. La mise en place de Openstack Swift a été une épreuve que je suis heureux d'avoir pu mener à terme. En effet, les services Openstack sont souvent très complets et très complexes à mettre en place et m'ont permis d'avoir une meilleure compréhension du fonctionnement de ces services tout en approfondissant mes compétences dans le domaine du Cloud et des systèmes distribués. Dû aux circonstances particulières du début du stage avec la crise du Coronavirus, j'ai pu m'atteler à réfléchir à des problèmes de traitement de l'information avec ce projet de traduction automatique de JSON. Même si aucune solution n'est sortie de cette réflexion, il m'a permis de découvrir de nombreux aspects de l'intelligence artificielle ainsi que de nombreuses solutions pour un même problème. Les approches sont nombreuses et différentes mais permettent toutes de potentiellement fournir une solution à cette problématique. J'ai appris aussi beaucoup sur la conception de solution ayant des contraintes opérationnelles fortes. En effet, les besoins en confidentialité de certains projets ont vraiment orienté certains aspects de la conception ainsi que le choix des outils.

Bibliographie

[1] Daydé, M., *Présentation*, depuis https://www.irit.fr, ouvert en Août 2020

[2] *Présentation*, depuis https://www.irit.fr/departement/intelligence-collective-interaction/equipe-smac/, ouvert en Août 2020

[3] Dépôt de gestion de version Github https://github.com/vincentnam/docker_datalake

[4] Site officiel de Apache Airflow - https://airflow.apache.org/, ouvert en Août 2020

[5] Documentation sur les pré-requis matériel d'une plateforme Openstack Swift à grande échelle, https://www.swiftstack.com/docs/admin/hardware.html, ouvert en Juillet 2020

[6] RFC 2046 : MIME Type, standardisation de l'IETF, https://tools.ietf.org/html/rfc2046

[7] RFC 8259 : norme JSON par l'IETF https://tools.ietf.org/html/rfc8259

[8] ECMA-404, norme JSON par l'ECMA http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[9] Standardisation OASIS du protocole MQTT version 5 https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

[10] *A Universal Part-of-Speech Tagset*, Slav Petrov, Dipanjan Das, Ryan McDonald, 11 Apr 201, Google Research and Carnegie Mellon University https://arxiv.org/abs/1104.2086

[11] *Learning Structured Text Representations*, Yang Liu and Mirella Lapata Institute for Language, Cognition and Computation, School of Informatics, University of Edinburgh, https://www.aclweb.org/anthology/Q18-1005.pdf

[12] Detectron 2 github Repository - Image analysis with Mask-RCNN https://github.com/facebookresearch/Detectron