

|  |
| --- |
| 5e année Systèmes Distribués & Big Data |
| Rapport de Projet Intégrateur |
| Reconnaissance d’images satellites dans une infrastructure distribuée |

|  |
| --- |
| Adrian MEGA, Hao HU, Mohamed EL FILALI  2018-2019 |

# Table des matières

[Remerciements 2](#_Toc535847094)

[Contexte et Objectifs 3](#_Toc535847095)

[Fonctionnalités Implémentées 4](#_Toc535847096)

[Démarche 5](#_Toc535847097)

[Résultats 6](#_Toc535847098)

[Conclusion 7](#_Toc535847099)

# Remerciements

Nous souhaitons remercier les contributeurs de ce projet qui nous ont permis de le réaliser dans le cadre de notre formation et ainsi de mettre en application nos compétences.

Merci à Christophe GOUGUENHEIM, travaillant chez Thalès Alenia Space, pour nous avoir proposé ce sujet de projet intégrateur et pour avoir répondu à nos questions.

Merci à notre responsable de spécialité Marie-José HUGUET pour nous avoir encadré sur ce projet ainsi que sur tous les autres projets de cette 5e année en spécialité SDBD.

Merci également aux autres encadrants de ce projet intégrateur, Sami YANGUI et Mohamed SIALA, pour leur soutien durant nos heures de travail au GEI.

Et enfin merci aux autres étudiants de notre classe pour nos multiples échanges sur ce projet. Seul on va plus vite, ensemble on va plus loin !

# Contexte et Objectifs

Le projet intégrateur de 5e année à l’INSA Toulouse doit rassembler toutes les connaissances acquises durant le dernier semestre d’études afin de les mettre en application. Dans le cadre de la spécialité Systèmes Distribués et Big Data, Thales Alenia Space nous a proposé son projet de reconnaissance d’images satellite via une infrastructure distribuée.

Une grande quantité d'images satellite sont générées et traitées par Thalès Alenia Space. Un certain nombre de calculs sont appliqués sur ces images. La taille des images réelles prises par les satellites (plus de 250 Go) nécessite de mettre un place un système distribué à la fois pour leur stockage et pour les diverses analyses souhaitées.

En termes d’analyses d’images, les deux pistes privilégiées sont d’une part la détection de sols (champs, océan, forêt, zone urbaine, …) et d’autre part la détection de cibles (navires, éoliennes, …). Ces analyses peuvent exploiter diverses techniques d’apprentissage automatique.

L'objectif du projet est d’une part de de créer un système distribué de stockage et d’autre part d'analyser ces images pour leur ajouter de la valeur. Le projet inclus les étapes suivantes :

* Création d’une base de données objet (avec Minio)
* Création d’une base d’indexation et de recherche d’information (avec Elasticsearch)
* Développement d’algorithmes Python de classification pour enrichir les métadonnées des images via un système de calcul distribué (avec Spark)

La restitution de ces développements se fera sous la forme de fichiers Docker et Docker Compose. L’ensemble du code sera déposé et rendu disponible sur un dépôt Git.

Les images fournies sont des images de taille réduite pour permettre leur exploitation dans un l’environnement de travail. Une évaluation du passage à l’échelle pourra être envisagée en fonction des premiers résultats obtenus.

Ce rapport a pour but de présenter l’infrastructure distribuée mise en place, les méthodes de reconnaissance d’images développées, la qualité de l’approche et une mise en perspective de l’application. Une soutenance orale viendra également accompagner la restitution du travail effectué.

# Fonctionnalités Implémentées

Ce chapitre a pour but de faire le point sur toutes les fonctionnalités qui ont été implémentées durant la période de temps imparti pour ce projet intégrateur.

## Architecture orientée services

Notre architecture est composée d’instances de containers chargés de traiter et organiser la donnée. Toute cette architecture est inscrite dans le Docker Compose de notre projet. Celui-ci rassemble des containers contenant les images suivantes.

|  |  |  |  |
| --- | --- | --- | --- |
| Nom | Nombre | Port(s) | Description |
| Portainer | 1 | 9000 | Cet outil d’administration nous sert à gérer tous les containers présents sur l’instance Docker locale. A partir de Portainer, nous pouvons voir l’état de fonctionnement de nos containers, les arrêter et redémarrer, regarder les journaux d’événement de chacun (logs), entrer dans le terminal d’un container, etc. |
| Minio | 4 | 9001 à 9004 | Minio est un projet open source de stockage objet. Il a été nativement conçu pour les infrastructures cloud à grande échelle. C’est sur ces 4 instances que nous stockons nos données image. Le fait d’avoir plusieurs instances nous permet d’une part de gagner en efficacité de calcul, et d’autre part d’assurer la persistance des données. |
| spark (Master) | 1 | 7077 | Le Framework open source de calcul distribué Spark nous permet de faire du traitement de données. Cette première image Master sert à la transition de données depuis et vers les serveurs Minio. Elle sert également à l’affectation des travaux de calcul aux autres images Spark. |
| Spark (Worker) | 4 | 8081 à 8084 | Ces autres images Spark quant à elles sont dédiées au traitement des données images. Ce sont elles qui appliquent l’algorithme d’apprentissage actif sur les données annotées. |
| elasticsearch | 1 | 9200 | Elasticsearch est un référenceur de données ainsi qu’un moteur de recherche et d’analyse. C’est lui qui va indexer les données stockées sur les serveurs Minio afin qu’elles puissent être exploitées, notamment par les images Spark. |
| Kibana | 1 | 5601 | Kibana est un greffon à Elasticsearch, servant à la visualisation des données qui y sont référencées. |

## Algorithmes de classification

\*\* Dire ce qui marche pour l’envoi + indexation + traitement d’image \*\*

# Démarche

Afin de compléter notre projet, nous avons exploré différentes pistes sur chacune des fonctionnalités de notre application. Cette partie a pour but de présenter notre démarche de recherche.

## Premier Docker Compose et Portainer

Ce projet a été l’occasion de nous former sur l’utilisation de Docker. Entrevu dans les enseignements à l’INSA, nous n’avons jamais eu de réelle mise en pratique auparavant. Adrian ayant déjà travaillé en entreprise avec cet outil, il a été en mesure d’apporter à l’équipe son expérience avec Docker ainsi que Docker Compose.

Le premier outil ajouté au Docker Compose de notre projet fut Portainer. Bien qu’il ne réponde pas à une exigence du projet, il s’est avéré très utile durant tout le développement du projet. Véritable tableau d’administration de containers, il nous permettait de lancer des tests individuels, de s’assurer du bon fonctionnement de chaque container et de leur bonne communication. Portainer est facile d’installation car une image préconçue est rendue disponible sur les serveurs de Docker.

## Minio et configuration de ports réseau

Minio doit nous servir à stocker les images satellites qui seront ensuite indexées par Elasticsearch. L’installation de Minio peut aussi se faire via une image des serveurs Docker déjà prête à l’emploi. Nous avons trouvé des exemples de projets incluant l’initialisation d’instances Minio sur un Docker Compose et nous nous en sommes inspiré pour notre propre configuration. En discutant avec nos enseignants, ainsi qu’avec l’intervenant de Thalès, il a été défini que pour des raisons de performance, il y aurait quatre instances Minio qui devraient tourner en parallèle. Les données seraient alors partagées entre les 4 serveurs.

Afin d’avoir une configuration cohérente, nous avons beaucoup travaillé sur l’optimisation de l’utilisation des ports afin d’accéder à chaque service. Tout en conservant la configuration de base, nous voulions simplifier au maximum les accès à chaque service. Notons qu’avec Docker, nous pouvons accéder à un service soit par une adresse IP donnée, soit par un port défini sur localhost (ou une autre adresse, il est possible de créer des réseaux autres que le réseau de base sur Docker). Nous avons choisi de mettre tous nos services sur le même serveur local de base, avec un port bien défini par service (la liste de ces ports est disponible dans la partie Fonctions Implémentées de ce rapport).

Enfin, les quatre instances Minio sont capables de communiquer entre elles. Cela permet d’une part de référencer le contenu les autres instances depuis n’importe laquelle (elles ont conscience de fonctionner ensemble), d’autre part de pouvoir faire de la sauvegarde de données sur les autres instances.

## Elasticsearch et Kibana

L’installation de Elasticsearch fut plus complexe que les installations précédentes. A la façon de Minio, nous avons cherché des exemples de configuration de Elasticsearch dans d’autres projets incluant un Docker Compose. La configuration trouvée ne fonctionnait jamais sur nos machines de tests. Nous avons donc fait abstraction des configurations conseillées pour nous concentrer sur les paramétrages essentiels uniquement. Après plusieurs tests, nous avons défini une configuration minimale et efficace pour notre application.

Kibana était souvent associé à Elasticsearch lorsque nous faisions nos recherches sur l’installation de ce dernier. Après avoir étudié son utilisation, nous avons jugé intéressant de l’inclure dans l’architecture de notre projet, afin de visionner les données indexées. Pour les deux services, nous utilisons des images préconçues disponibles sur les servers Docker Elastic.

## Spark

\*\* Quels ont été les procédés pour comprendre le fonctionnement de spark, l’installer et l’utiliser (c’est là que on peut parler de pyspark \*\*

## Classification et indexation

\*\* même question pour les algos de traitement d’image (c’est là qu’on parle de Keras) \*\*

# Résultats

Cette partie du rapport montrera comment installer notre application et la mettre en fonctionnement avec les images déployées sur les serveurs et l’apprentissage actif lancé.

## Initialisation de l’architecture

En se plaçant dans le répertoire de projet, il suffit de lancer la commande suivante :

docker-compose up -d

Ainsi toute l’architecture Docker sera installée sur la machine hôte.

## Ajout d’images sur Minio et indexation du contenu

Etc …

Et conclure sur le score calculé par le model

# Conclusion