 Université IBN TOFAIL

Faculté de Sciences-Kenitra

Département Informatique

*Développement d’une application webpour la gestion des incidents informatique OCP JORF LASFAR EL JADIDA*

Rapport De Projet De Find’Etude

pour l’obtention du diplôme :

Master Big Data & Cloud Computing



Effectue à :

**Office Chérifien des Phosphates - El Jadida**

Encadrent Pédagogique par : Encadrent à l’OCP par :

Dr. Jaafar Abouchabaka M. Youssef Chriai

**Soutenu le ………… devant le jury compose de :**

**Dr. Jaafar Abouchabaka ……Chef de departement informatique**

**Dr. Xxxxxxxxxxxxxxxxxx…..Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx**

**Dr. Xxxxxxxxxxxxxxxxxx……Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx**

**2018-2019**





Ce travail n’aurait pas pu prendre naissance sans l’assistance et l’intervention généreuse de certaines personnes dont les apports ne pourraient être qu’infiniment reconnus. Avant tout développement sur cette expérience professionnelle, il apparaît opportun de commencer ce rapport de stage par des remerciements :

Nous remercions tout d’abord, chaleureusement, notre Encadrant, Mme. Najat RAFALIA, de nous avoir permis de passer notre projet de fin d’étude auprès de lui, ses directives et sa Générosité, aussi pour sa disponibilité continue, à nous faire Partager ses connaissances, son expérience et son savoir- faire, et n’ayant ménagé aucun effort pour nous initier à la vie Professionnelle, par les conseils précieux et les Recommandations qu’il nous a prodigués durant toute la Période de la préparation du projet. Nous témoignons, notre profonde gratitude aux membres du jury, nous avons l’honneur de juger notre travail, et Faculté de sciences Ibn tofail département informatique pour le savoir qu’elle nous offre jour après jour. Enfin, Qu’il nous soit permis de remercier tout le corps professoral et Administratif de la faculté ainsi que toute personne ayant Contribué à bien mener ce Stage de fin d’études.





1. La Gestion des incidents est une partie cruciale de la continuité des activités de toute organisation. Aujourd’hui, avoir un outil de communication qui vous permet de contacter tous les membres de votre organisation, ainsi que surveiller la situation en cours ou d'incident, est essentiel pour être en mesure de faire face à des circonstances imprévues,

A ce titre, s’est inscrit mon PFE qui consiste à réaliser une application de Gestion des incidents afin de restaurer une activité normale le plus rapidement possible et minimiser les interruptions de service au sein de l’entreprise.

Après avoir présenté dans le premier chapitre :

\*la présentation D’Office Chérifien des Phosphates,

\* La description du projet et des problématiques traitées,

Le deuxième chapitre va donner une analyse détaillée de cette problématique.

En fin, je présenterai, dans le troisième chapitre :

\*Les différents outils utilisés lors de la réalisation de l’application

\* les différentes interfaces de l’application.

# 

# Chapitre 1 : Présentation de l’organisme d’accueil

## Historique

Le groupe OCP est un acteur de référence incontournable sur le marché international, il est domicilié au Maroc. OCP S.A est le leader mondial des exportations de phosphates et produits dérivés, avec des activités couvrant l’ensemble de la chaîne de valeur, allant de l’extraction de la roche de phosphate à la transformation chimique en acide phosphorique et différents engrais.

Mondialement, le Maroc détient les réservoirs les plus importants de phosphate dans son sous-sol, il est de l’ordre de 51,8 Milliard de tonnes de minerai, ce qui représente 75 % des réserves mondiales. Ses premières traces ont été découvertes en 1912, dans les régions des OULAD ABDOUN, zone de Khouribga. Son gisement est une superposition de couches de différentes teneurs situées à 120 Km du Sud-est de Casablanca.

Le phosphate provient de la décomposition des fossiles des animaux de mers qui ont vécu, il y a plusieurs millions d’années du fait que les mers et les océans recouvraient une grande partie des continents actuels.

La création de l’OCP fut en 1920 et l’exploitation n’a commencé qu’en 1921 dans la région de Oued-Zem, depuis lors les besoins continus de l’agriculture mondiale en phosphate ont fait de l’office une entreprise qui jusqu’à nos jours n’a cessé de grandir et pour se maintenir sur le plan de la concurrence par rapport aux autres pays producteurs de phosphate et dérives, elle se modernise, se développe continuellement et s’affirme comme le LEADER du marché mondial des phosphates.

## Status Juriqique de l’OCP

L’OCP a été constitué sous la forme d’un organisme semi-publique sous contrôle de L’ETAT. Il fonctionne ainsi comme une société dont le seul actionnaire est l’état Marocain, appliquant les méthodes de gestion privée, dynamique, souple et efficace vu le caractère de ses activités industrielles et commerciales, il est dirigé par un Directeur General nomme par DAHIR, le contrôle est exercé par un conseil d’administration préside par le premier ministre sur le plan fiscal, elle est inscrite sur le registre de commerce soumis à la même obligation que d’autres entreprises privées.

Sur le plan financier ; entièrement indépendante de celle de l’ETAT. Toutes les années, le Groupe, participe au budget de l’Etat par le versement des dividendes. La gestion du personnel est régie par le statut du mineur du 1er janvier 1973. Ce statut a été élaboré en conformité avec le DAHIR n° 16007 du 24 décembre 1960 sur le statut des entreprises minières au MAROC.

Les structures actuelles ont été modifiées par le document 716 du 1/1/1971 signe par le Directeur General du Groupe OCP est d’environ 22677 dont 725 ingénieurs.

## Sites miniers

Le Groupe est présent dans cinq zones géographiques du pays dont trois sites d’exploitation minière Khouribga, Benguerir/Youssoufia, Boucraa/Laayoune et deux sites de transformation chimiques : Safi et Jorf Lasfar.

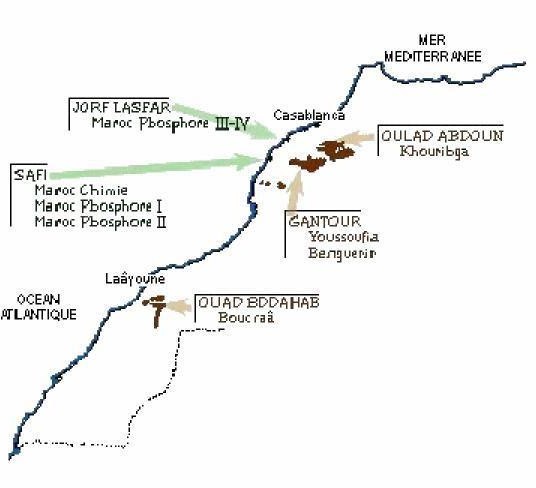


Figure 1. Carte géographique des sites miniers distribués sur le royaume

L’implantation géographique a connu plusieurs mines telles qu’Ouled Abdoun, Gantour, Boucraa… etc.

***OULED ABDOUN :***

C'est la plus importante mine de la production du groupe OCP, située dans la zone de Khouribga, ses réserves en phosphates sont estimées a plus de 35 milliards de mètre cubes. Sa capacité de production est de 19 millions de tonnes par an.

***GANTOUR :***

La Direction des Exploitations Minières de GANTOUR a pour mission l’extraction, le traitement et la livraison du phosphate a partir du gisement de GANTOUR, ce gisement sur 125 Km d’Est à l’ouest et sur 20Km du Nord au Sud. Il couvre une superficie de 2500Km2. Il existe deux centres qui sont en exploitation : le centre de YOUSSOUFIA (depuis 1939) et le centre de BENGURIR (depuis 1980).

***OUED EDDAHAB (BOUCRAA-LAAYOUNE) :***

Situe a 50 Km au Sud-est de la ville de LAAYOUNE, les réserves des gisements des phosphates d’OUED EDDAHAB sont estimées a 1,13 milliards de mètres cubes. Le gisement de Boucraa est en exploitation, compte tenu de ses réserves et de sa teneur.

## Missions

* + **L’extraction** : est une opération qui se fait en découverte, elle consiste à extraire le phosphate de la terre et s’exécute en 4 phases : Forage, sautage, décapage et défruitage.
  + **Le traitement** : c’est une opération qui se fait après extraction et a pour but l’enrichissement du minerai en améliorant sa teneur.
  + **Le transport** : une fois le phosphate extrait puis traite, il est transporté vers le port de CASA, SAFI ou EL JADIDA à destination des différents pays importateurs
  + **La vente** : le phosphate est vendu, selon des demandes des clients, soit brut soit après valorisation (transforme en engrais, acides phosphoriques ou acides sulfuriques), aux industries chimiques

## Impact Economiques

Le Groupe OCP constitue un vecteur de développement régional et national important.

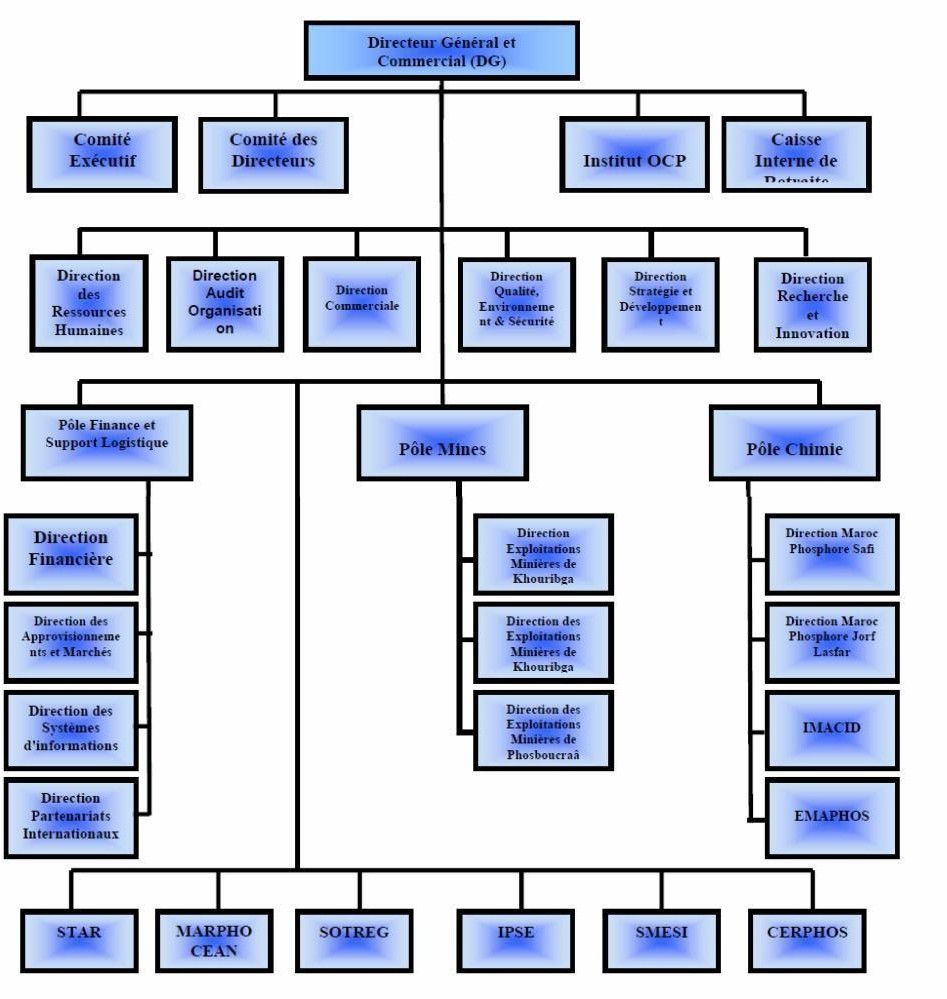
* 1er exportateur mondial de phosphate,
* 1er exportateur mondial d'acide phosphorique,
* 1er exportateur mondial de P2O5 sous toutes formes.
* Sa contribution dans le PIB est de 2 à 3% et dans les exportations marocaines en valeur de 18% à 20%.

## Filiales

En 1975, le groupe OCP a été institué et comprend en plus de l’OCP les sociétés suivantes :

* + **CERPHOS** : Centre d’études et de recherche des phosphates minéraux, sa mission est d’organiser et exécuter toute activité d’analyse, d’étude et de recherche scientifique et technique.
  + **FERTIMA** : Société marocaine des fertilisants, son but est de commercialiser les engrais à l’intérieur du pays en provenance des chimiques du groupe OCP. **MARPHOCEAN** : Il est spécialisé dans le transport maritime de l’acide phosphorique, les engrais et d’autres produits chimiques.
  + **SMESI** : Société marocaine d’études spéciales et industrielles, ses activités principales sont l’étude et la réalisation des installations industrielles (manutention, stockage…).
  + **SOTREG** : Société de transport régional, est chargée du transport du personnel du groupe OCP.
  + **PHOSBUCRAÄ** : Sa mission est l’extraction, le traitement et l’expédition du phosphate.
  + **STAR :** Société de transport et d’affrètement réunis des navires et service annexes.

## Structure et Organigramme



# : Généralités sur les technologies BIG DATA

## Introduction

Chaque jour, 2,5 de téraoctets de données sont générées dans le monde. Aujourd’hui l’information nous provient de toute part (des données postées sur les réseaux sociaux, des vidéos et des images digitales, des transactions des clients, …). Le développement et l’accès à ces données représentent ce qui est appelé le phénomène « Big Data ».

Ce phénomène impacte en particulier les entreprises qui sont amenées à manipuler des téraoctets de données nécessitant une infrastructure spécifique pour leur création, leur stockage, leur traitement, leur analyse et leur récupération.

En d’autres termes, il s’agit du développement en temps réel d’une masse de données volumineuse qui dépasse la capacité des outils de traitement et d’analyse traditionnels (bases de données relationnelles, requêtes SQL …).

« Big Data » fait référence aux technologies, processus et techniques permettant à une organisation de créer, manipuler et gérer des données à grande échelle.

Dans la littérature, le concept de « Big Data » se définit à travers la théorie des 3V :

-Volume : correspond à la taille des flux de données.

-Variété : correspond aux données hétérogènes.

-Vélocité (Vitesse) : correspond à la rapidité de production des données.

Il faut en effet penser à collecter, stocker puis analyser les données d’une façon qui ne peut plus être traitée par une approche traditionnelle pour permettre de satisfaire au 4ème V : la Valorisation des données.

Big Data : exploration de très vastes ensembles de données pour obtenir des renseignements utilisables Le terme Big Data se réfère aux technologies qui permettent aux entreprises d'analyser rapidement un volume de données très important et d'obtenir une vue synoptique. En mixant intégration de stockage, analyse prédictive et applications, le Big Data permet de gagner en temps, en efficacité et en qualité dans l’interprétation de données. Les objectifs de ces solutions d’intégration et de traitements des données sont de traiter un volume très important de données aussi bien structurées que non structurées, se trouvant sur des terminaux variés (PC, smartphones, tablettes, objets communicants...), produites ou non en temps réel depuis n'importe quelle zone géographique dans le monde.

## Caractéristiques du BIG DATA

Les Big Data répondent en principe à quatre caractéristiques:

**Volume :** les Big Data représentent de grosses quantités de données. On dit généralement que 90% des données disponibles aujourd’hui ont été créées ces dernières années.

**Vitesse :** les données sont générées, capturées et partagées à une vitesse toujours plus importante Les délais d’actualisation et d’analyse des données sont toujours plus courts et elles sont le plus souvent traitées en temps réel ou quasi réel.

**Variété (ou hétérogénéité) :** les données analysées ne sont pas forcément structurées. Elles peuvent provenir de sources différentes (et avoir un format différent comme du texte, des images, du contenu multimédia, des traces numériques, etc.) et être combinées entre elles. Des données enregistrées dans un fichier clients interne peuvent être combinées avec des données externes provenant de réseaux sociaux, de moteurs de recherche, de feuilles d’avis officielles ou de portails de données ouvertes gérés par des autorités publiques.

**Valeur :** la dernière caractéristique est la plus-value que l’analyse des données représentent et les usages qu’il est possible d’en faire.

## BIG DATA ET STOCKAGE DE DONNEES

Au terme « Big Data » souvent une question se pose de savoir quelles sont les architectures de stockage les mieux adaptées pour soutenir des processus analytiques à grande échelle. En fait, plusieurs technologies rivalisent aujourd’hui dans ce domaine, chacune a ses propres forces et faiblesses.

La qualification des données est la clé du stockage c’est pour cela que lors de la mise en œuvre de ces types de solutions, une étude sur le stockage est nécessaire. Quelles sont mes données

? Quelle est la volumétrie (max, écart-type, moyenne) ? Quelle est leur croissance ? Quels types de traitements sont effectués ? Doivent-elles être centralisées ?

Toutes ces réponses permettront de catégoriser les données. Ainsi l’optimisation des services de stockage Big data permettent à l’entreprise de maîtriser la variabilité et les performances. L’indexation devient plus facile, la taille des caches est optimisée.

Les solutions de stockage Big data commencent à être adoptées par les entreprises, qui les utilisent sur leurs périmètres stratégiques pour en tirer des gains de performance là où les solutions standards étaient limitantes. L’intégration de ces solutions reste l’étape clé : l’étude des données est nécessaire pour choisir la bonne implémentation Big data, faciliter sa configuration et ainsi profiter pleinement des bénéfices annoncés.

La couche stockage est l’un des premiers facteurs limitant et les problématiques3 associées sont les suivantes :

* **Performance** : l’augmentation du volume à traiter entraîne une diminution des performances.
* **Linéarité :** l’outillage n’étant pas adapté aux gros volumes de données, le modèle de scalabilité n’est pas linéaire, l’ajout de stockage ne permettant pas d’améliorer les performances.
* **Dynamisme** : l’allocation d’espace est fixe avec une faible réactivité pour en ajouter ou en retirer

## HADOOP

### Définition :

HADOOP est une architecture spécifique de bases de données, permettant de traiter en grand nombre tous types de données y compris les données non structurées. On dit qu’elle est organisée sur un mode non-relationnel, c’est-à-dire que les données ne sont pas triées en fonction de leurs relations entre elles.

Elle constitue en quelque sorte la librairie à partir de laquelle on pourra effectuer des calculs parallèles (via MapReduce), sachant que ces données sont distribuées (c’est-à-dire qu’Hadoop les « récupère » depuis diverses sources).

A petite échelle, de l’ordre du Téra octet, les données peuvent être stockées sur des disques durs. Dans le cas du Big Data, les volumes de données peuvent dépasser le Pétaoctet. Les données sont alors stockées dans des sites spécialisés : les Datacenter.

* Projet de la fondation Apache –Open Source, composants complètement libre, tout le monde peut participer
* Modèle simple pour les développeurs: il suffit de développer les tâches MapReduce depuis des interfaces simples accessibles via des bibliothèques dans des langages multiples (Java, Python, C/C++, Rubby, R, etc.)
* Déployable très facilement (paquets Linux pré-configurés). Configuration très simple elle aussi
* S'occupe de toutes les problématiques liées au calcul distribué, comme l'accès et le partage des données, la tolérance aux pannes, ou encoure la répartition des tâches aux machines membres de cluster: le programmeur a simplement à s'occuper du développement logiciel pour l'exécution de la tâche.

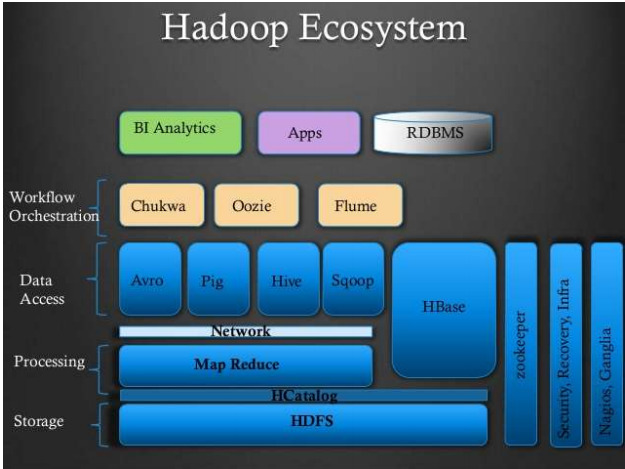
### Qui utilise Hadoop?



### les **caractéristiques Hadoop**

* Le projet Hadoop consiste en deux grandes parties:
* Stockage des données : HDFS (Hadoop Distributed File System)
* Traitement des données : MapReduce
* Hadoop Commun
* Principe :
* Diviser les données (structurer, non structurer, semi structurer)
* Les sauvegarder sur une collection de machines, appelées cluster
* Traiter les données directement là où elles sont stockées, plutôt que de les copier à partir d’un serveur distribué
* Il est possible d’ajouter des machines à votre cluster, au fur et à mesure que les données augmentent

### Ecosystème de Hadoop :



* **HDFS (Hadoop Distributed File System)** :
* Un système de fichiers distribué, extensible et portable développé par Hadoop en Java et basé sur le principe MapReduce,
* conçu pour stocker de très gros volumes de données sur un grand nombre de machines peut couteuses équipées de disques durs banalisés.
* Il permet l'abstraction de l'architecture physique de stockage, afin de manipuler un système de fichiers distribué comme s'il s'agissait d'un disque dur unique.
* **MapReduce** :
* Joue un rôle majeur dans le traitement des grandes quantités de données.
* – La distribution des données au sein de nombreux serveurs permet le traitement parallélisé de plusieurs tâches portant chacune sur des morceaux de fichiers.
* La fonction Map accomplit une opération spécifique sur chaque élément.
* L'opération Reduce combine les éléments selon un algorithme particulier, et fournit le résultat.
* Soulignons que le principe de délégation peut être récursif : les noeuds à qui sont confiées des tâches peuvent aussi déléguer des opérations à d'autres noeuds.
* **Ambari**:
* Destiné à la supervision et à l'administration de clusters Hadoop
* Outil web qui propose un tableau de bord (Visualisation de l'état d'un cluster – état des services, configurations, supervision, executiondes jobs, métriques)
* **Hbase** : Système de gestion de données non-relationnelles distribué de type orientée colonnes avec un accès direct et une lecture/écriture temps réel.
* **Pig**: une plateforme pour analyser des ensembles de gros volumes de données c'est un langage de requêtage de données Hadoop à partir d'un langage de script
* **Hive**: un système d'entrepôt de données pour Hadoop qui offre un langage de requête de type SQL

### Hadoop Distributed FileSystem (HDFS)

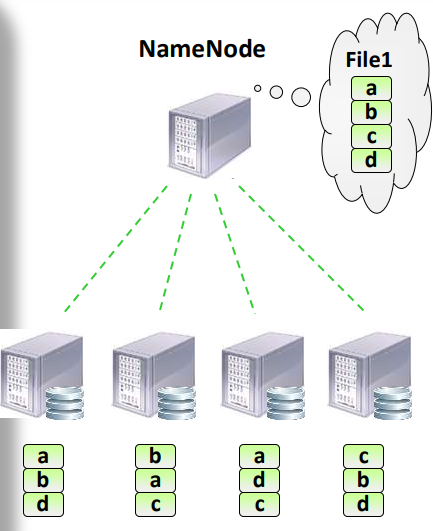
* HDFS est un système de fichiers distribué, extensible et portable écrit en Java
* Inspiré de GFS, un système de fichiers distribué conçu par Google ;
* Pour stocker les données en entrée de nos tâches Hadoop, ainsi que les résultats de nos traitements;
* Permet de stocker de très gros volumes de données sur un grand nombre de machines (nœuds) équipées de disques durs banalisés ⎝ Cluster
* Il s'agit du système de fichiers standard de Hadoop - au même sens que les systèmes de fichiers FAT32, NTFS ou encore Ext3FS, mais :
* Systèmes de fichiers : Les données sont écrites dans des blocs gérés par le FileSystem ;
* HDFS : Les données sont écrites dans des blocs gérés par le HDFS, qu'il est évidemment distribué.

#### Caracteristiques HDFS :



* Distribué, évolutif, tolérant aux pannes, à haut débit
* Accès aux données via MapReduce
* Fichiers divisés en blocs
* 3 répliques pour chaque élément de données par défaut
* Peut créer, supprimer, copier, mais pas mettre à jour
* Conçu pour lire en Streaming, pas pour accès aléatoire
* Data locality : traitement des données sur le support de stockage physique pour réduire la transmission des données

#### HDFS : Architecture :



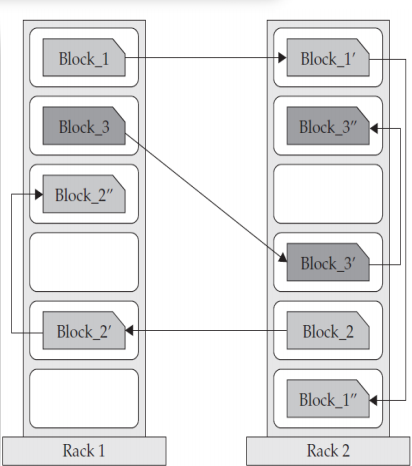
* Architecture (Master/Slave) : Dans une architecture Hadoop, chaque membre pouvant traiter des données est appelé Node
* Un seul d'entre eux peut être Master, il s'agit du NameNode (NN). Il est responsable de la localisation des données dans le cluster.
* Il Gère le système de fichiers
* Espace de noms et métadonnées
* Réglemente l'accès des clients aux fichiers
* Les autres noeuds, stockant les données sous forme de blocs, sont des slaves appelés DataNode (DN) ;
* Plusieurs par cluster
* Gère le stockage attaché aux nœuds
* Rapporte périodiquement l'état à NameNode

#### architecture de stockage maître-esclave

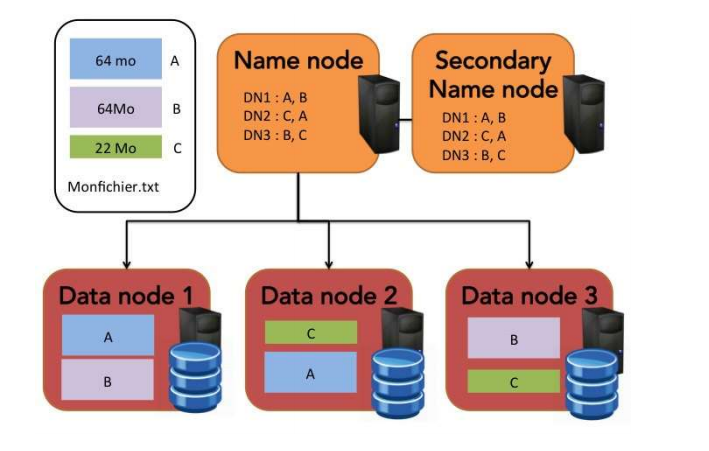
* Le nœud maitre appelé **namenode** contient et stocke tous les noms et blocs des fichiers ainsi que leur localisation dans le cluster. On peut donc le voir comme un gros annuaire.
* Une autre machine, appelée **secondary namenode** sert de **namenode** de secours en cas de défaillance du nœud maître et il a donc pour rôle de faire des sauvegardes régulières de l'annuaire.
* Les autres nœuds, les esclaves, sont les nœuds de stockage en tant que tels. Ce sont les **datanodes** qui ont pour rôle la gestion des opérations de stockage locales (création, suppression et réplication de blocs) sur instruction du namenode.

#### HDFS : Replication :

* Les blocs de données sont répliqués sur plusieurs nœuds
* Par défaut, 3 répliques



* Un réplica sur un nœuds dans le rack local
* Une autre réplique sur un nœud différent dans le rack local
* Et le dernier sur un nœud différent dans un rack différent
* Cela réduit la bande passante du réseau inter-rack, ce qui améliore les performances d'écriture
* Rack : (wikipedia) est une armoire très souvent métallique parfois à tiroirs mais généralement à glissières (ou rails) destinée à recevoir les boîtiers d'appareils, généralement électroniques, réseau ou informatiques de taille normalisée.



### Solution Map-Raduce

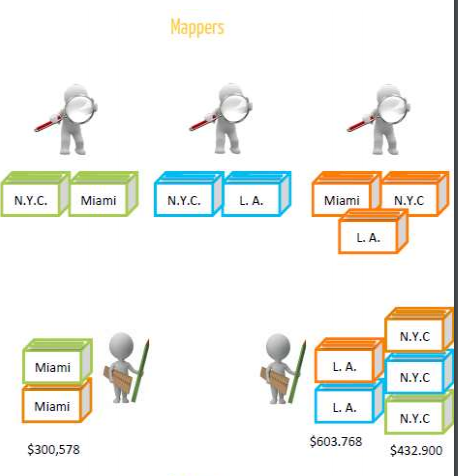
Patron d’architecture de développement permettant de traiter des données volumineuses de manière parallèle et distribuée • A la base, le langage Java est utilisé, mais grâce à une caractéristique de Hadoop appelée Hadoop Streaming, il est possible d’utiliser d’autres langages comme Python ou Ruby.

Au lieu de parcourir le fichier séquentiellement (bcp de temps), il est divisé en morceaux qui sont parcourus en parallèle.

Pour exécuter un problème large de manière distribuée, il faut pouvoir découper le problème en plusieurs problèmes de taille réduite à exécuter sur chaque machine du cluster (stratégie algorithmique dite: diviser pour régner) ;

#### Définition :Map-Raduse

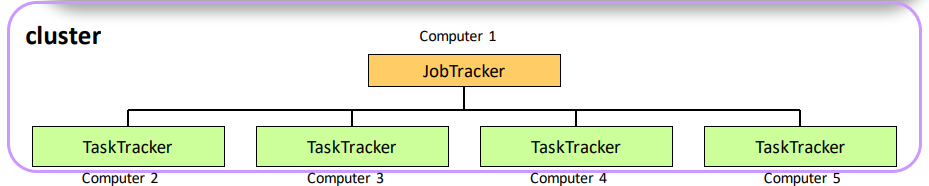
* MapReduce définit deux opérations distinctes à effectuer sur les données d'entrée :
* **Map:**
* Transforme les données d'entrée en une série de couples (key, value) ;
* Regroupe les données en les associant à des clés, choisies de telle sorte que les couples (key, value) aient un sens par rapport au problème à résoudre.
* **Reduce :**
* Applique un traitement à toutes les valeurs de chacune des clés distinctes produite par l'opération Map ;
* Au terme de l'opération Reduce, on aura un résultat pour chacune des clés distinctes. Ici, on attribuera à chacune des machines du cluster une des clés uniques produites par Map, en lui donnant la liste des valeurs associées à la clé. Chacune des machines effectuera alors l'opération Reduce pour cette clé.

****

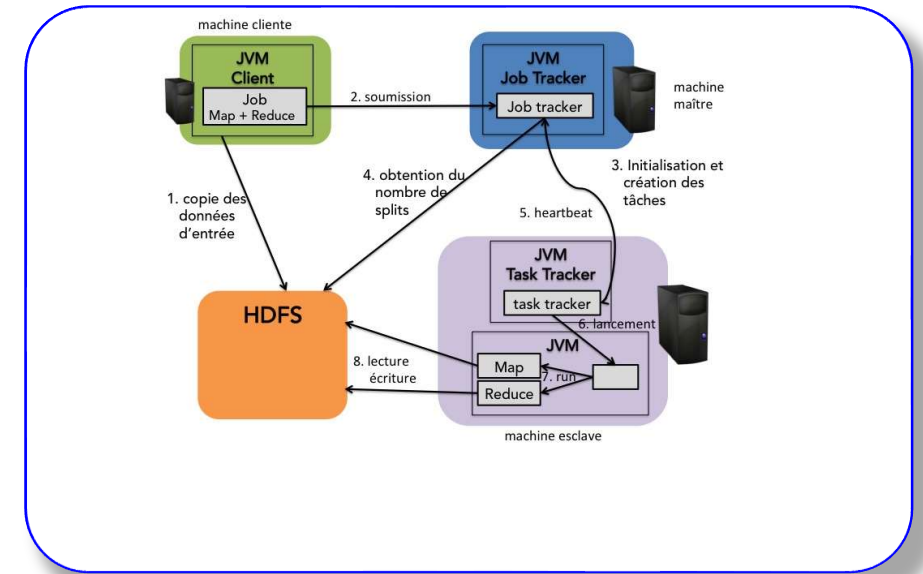
#### Architecture Map-Raduce :

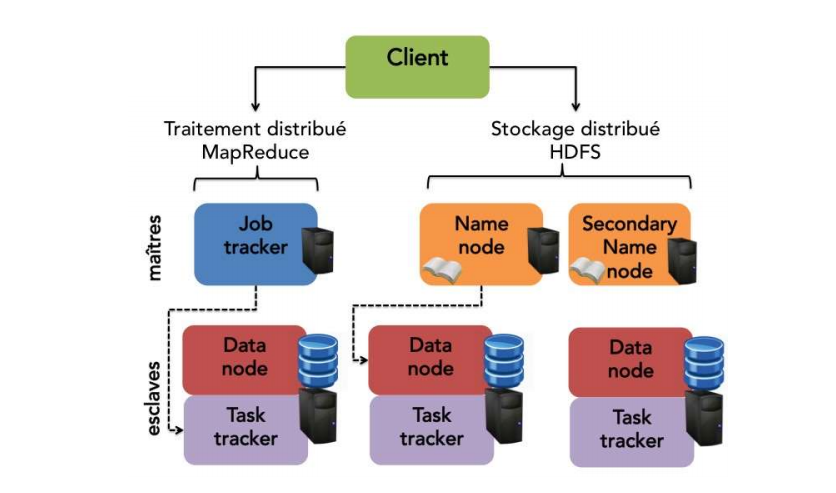
Un seul master (**JobTracker**) contrôle l'exécution des tâches sur plusieurs esclaves (**TaskTrackers**).

* **JobTracker** : Accepte les offres de MapReduce soumises par les clients et Divise le travail sur les Mappers et Reducers, s’exécutant sur les différents noeuds Maintient le travail aussi physiquement proche des données que possible.
* **TaskTracker :** S’exécute sur chacun des noeuds pour exécuter les vraies tâches de Map-Reduce et Choisit en général de traiter (Map ou Reduce) un bloc sur la même machine que lui et S’il est déjà occupé, la tâche revient à un autre tracker, qui utilisera le réseau (rare).

****

#### Schéma soumission d’un JOB MAPREDUCE:



* Un client hadoop copie ses données sur HDFS
* Le client soumet le travail à effectuer au job tracker sous la forme d'une **archive.jar** et des noms des fichiers d'entrée et de sortie.
* Le **job tracker** demande au **name node** où se trouvent les blocs correspondants aux données d'entrée.
* Il détermine alors quels sont les **nœuds Task Tracker** les plus appropriés pour exécuter les traitements (colocalisation ou proximité des nœuds). Il envoie alors au **task tracker** sélectionné et pour chaque bloc de données, le travail à effectuer (Map, Reduce ou Shuffle, fichier .jar).
* Les **task trackers** envoient régulièrement un message (hearbeat) au **job tracker** pour l'informer de l'avancement de la tâche et de leur nombre de slots disponibles.
* Quand toutes les opérations envoyées aux task trackers sont confirmées comme étant effectuées, la tâche est considérée comme effectuée.

# Généralités sur les Bases de donneés NoSql

## C’ est quoi le NoSql ?

Depuis les années 70, la base de données relationnelle était l'incontournable référence pour gérer les données d'un système d'information. Toutefois, face aux 3V (Volume, Velocity, Variety), le relationnel peut difficilement lutter contre cette vague de données. Le NoSQL s'est naturellement imposé dans ce contexte en proposant une nouvelle façon de gérer les données, sans reposer sur le paradigme relationnel, d'où le "**Not Only SQL**". Cette approche propose de relâcher certaines contraintes lourdes du relationnel pour favoriser la distribution (structure des données, langage d'interrogation ou la cohérence).

Dans un contexte bases de données, il est préférable d'avoir un langage de haut niveau pour interroger les données plutôt que tout exprimer en [Map/Reduce](https://fr.wikipedia.org/wiki/MapReduce" \t "_blank). Toutefois, avoir un langage de trop haut niveau comme SQL ne facilite pas la manipulation. Et c'est en ce sens que l'on peut parler de "Not Only SQL", d'autres solutions peuvent être proposées pour résoudre le problème de distribution. Ainsi, le NoSQL est à la fois une autre manière d'interroger les données, mais aussi de les stocker.

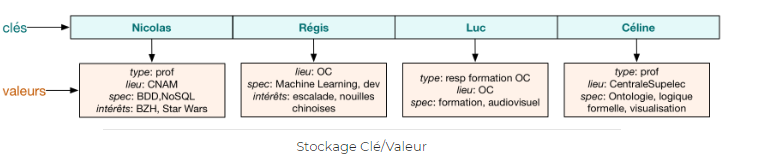
Les besoins de stockage et de manipulation dans le cadre d'une base de données sont variables et dépendent principalement de l'application que vous souhaitez intégrer. Pour cela, différentes familles de bases NoSQL existent : Clé/Valeur, colonnes, documents, graphes. Chacune de ces familles répond à des besoins très spécifiques que nous allons développer par la suite, avec pour but in fine, de vous permettre de choisir **votre** solution NoSQL.

#### Les base de données NoSql clés-valeurs :

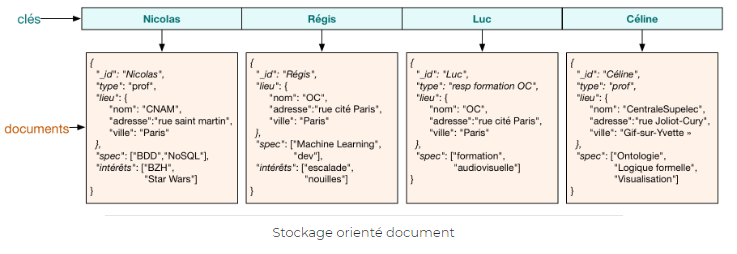
Le but de la famille clé-valeur est l'efficacité et la simplicité. Un système clé-valeur agit comme une énorme table de hachage distribuée sur le réseau. Tout repose sur le couple Clé/Valeur. La clé identifie la donnée de manière unique et permet de la gérer. La valeur contient n'importe quel type de données.

Le fait d'avoir n'importe quoi implique qu'il n'y ait ni schéma, ni structure pour le stockage. D'un point de vue de bases de données, il n'y a pas la possibilité d'exploiter ni de contrôler la structure des données et de fait, pas de langage (SQL = **Structured** Query Language). En soit ce n'est pas un problème si vous savez ce que vous cherchez (la clé) et que vous manipulez directement la valeur.

Les seules opérations de type CRUD peuvent être utilisées :

* Create (key,value)
* Read (key)
* Update (key,value)
* Delete (key)
* types d'applications:
* détection de fraude en temps réel.
* Iot.
* e-commerce.
* gestion de cache.
* transactions rapides.
* fichiers de logs.
* Chat.

#### Les base de données NoSql oriente document :

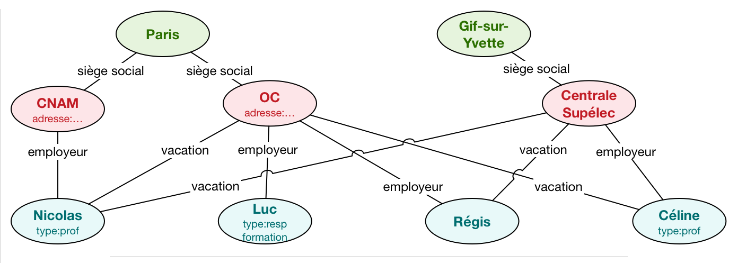
 Les bases orientées documents ressemblent sans doute le plus à ce que l'on peut faire dans une base de données classique pour des requêtes complexes. Le but de ce stockage est de manipuler des documents contenant des informations avec une structure complexe (types, listes, imbrications). Il repose sur le principe du clé/valeur, mais avec une extension sur les champs qui composent ce document.

L'avantage de cette solution est d'avoir une approche structurée de chaque valeur, formant ainsi un document. De fait, ces solutions proposent des langages d'interrogation riches permettant de faire des manipulations complexes sur chaque attribut du document (et sous-documents) comme dans une base de données traditionnelles, tout en passant à l'échelle dans un contexte distribué.

* exemples d'utilisation :
* gestion de contenu (bibliothèques numériques, collections de produits, dépôts de logiciels, collections multimédia, etc.)
* framework stockant des objets
* collection d’événements complexes
* gestion des historiques d’utilisateurs sur réseaux sociaux

#### Les base de données NoSql oriente graphe :

Les trois premières familles NoSQL n'adressent pas le problème de corrélations entre les éléments. Prenons l'exemple d'un réseau social : dans certains cas, il devient très complexe de calculer la distance entre deux personnes non directement connectées. Et c'est ce type d'approche que résolvent les bases orientées Graphe.



Dans la base orientée graphe, les données stockées sont : les nœuds, les liens et des propriétés sur ces nœuds et ces liens. Les requêtes que l'on peut exprimer sont basées sur la gestion de chemins, de propagations, d'agrégations, voire de recommandations. Toutefois, contrairement aux solutions précédentes la distribution des nœuds sur le réseau n'est pas triviale.

* exemples d'utilisation :
* réseaux sociaux (recommandation, plus court chemin, cluster...)
* réseaux SIG (routes, réseau électrique, fret...)
* web social (Linked Data)

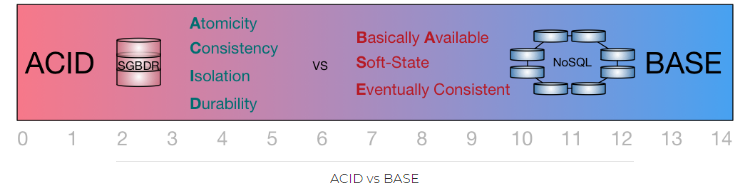
### ACID vs BASE

Pour mettre en rapport les problématiques de la base de données relationnelle, on parle de propriétés **ACID** pour les transactions (séquences d'opérations/requêtes) :

* **A**tomicité : Une transaction s’effectue entièrement ou pas du tout
* **C**ohérence : Le contenu d’une base doit être cohérent au début et à la fin d’une transaction
* **I**solation : Les modifications d’une transaction ne sont visibles/modifiables que quand celle-ci a été validée
* **D**urabilité : Une fois la transaction validée, l’état de la base est permanent (non affecté par les pannes ou autre)

Toutefois, ces propriétés ne sont pas applicables dans un contexte distribué tel que le NoSQL. En effet, prenons l'exemple d'une transaction de cinq opérations (lecture/écriture) : cela implique une synchronisation entre cinq serveurs pour garantir l'atomicité, la cohérence et l'isolation. Au final, cela se traduit par des latences dans les transactions (en cours et en concurrence). Ce qui n'est pas tolérable lorsque justement on veut éviter ces latences en distribuant les calculs.

Le problème s'aggrave encore lorsque l'on distribue les données car il va falloir répliquer chaque donnée. Pourquoi ? Tout simplement parce que si un serveur tombe en panne, il faut pouvoir garantir de retrouver toutes les données présentes sur ce serveur, donc on fait de la réplication. Mais cela veut dire également qu'il va falloir synchroniser toutes mises à jour avec tous les réplicas de la donnée !



### **théorème de CAP :**

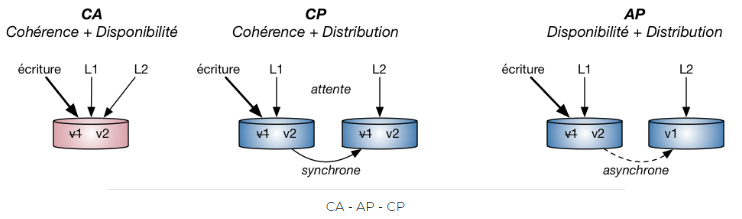
En 2000, [Eric A. Brewer](https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf) a formalisé un théorème très intéressant reposant sur 3 propriétés fondamentales pour caractériser les bases de données (relationnelles, NoSQL et autres) :

* **C**onsistency (Cohérence) : Une donnée n'a qu'un seul état visible quel que soit le nombre de réplicas.
* **A**vailability (Disponibilité) : Tant que le système tourne (distribué ou non), la donnée doit être disponible
* **P**artition Tolerance (Distribution) : Quel que soit le nombre de serveurs, toute requête doit fournir un résultat correct

**Le théorème de CAP dit :**

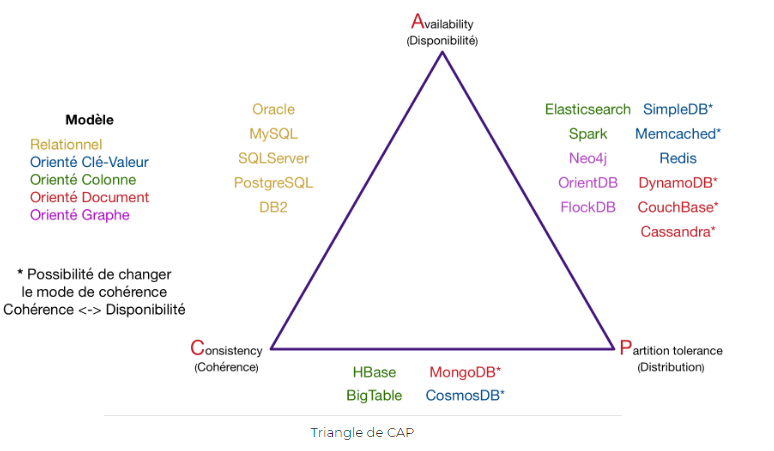
Dans toute base de données, vous ne pouvez respecter au plus que 2 propriétés parmi la cohérence, la disponibilité et la distribution.

Cela s'illustre assez facilement avec les bases de données relationnelles, elles gèrent la cohérence et la disponibilité, mais pas la distribution.



* Prenons le couple **CA** (Consistency-Availability), il représente le fait que lors d'opérations concurrentes sur une même donnée, les requêtes L1 et L2 retournent la nouvelle version (v2) et sans délai d'attente. Cette combinaison n'est possible que dans le cadre de bases de données [transactionnelles](http://sys.bdpedia.fr/conc.html) telles que les SGBDR.
* Le couple **CP** (Consistency-Partition Tolerance) propose maintenant de distribuer les données sur plusieurs serveurs en garantissant la tolérance aux pannes (réplication). En même temps, il est nécessaire de vérifier la cohérence des données en garantissant la valeur retournée malgré des mises à jour concurrentielles. La gestion de cette cohérence  nécessite un protocole de synchronisation des réplicas, introduisant des délais de latence dans les temps de réponse (L1 et L2 attendent la synchronisation pour voir v2). C'est le cas de la base NoSQL MongoDB.
* Le couple **AP** (Availability-Partition Tolerance) à contrario s'intéresse à fournir un temps de réponse rapide tout en distribuant les données et les réplicas. De fait, les mises à jour sont asynchrones sur le réseau, et la donnée est "Eventually Consistent" (L1 voit la version v2, tandis que L2 voit la version v1). C'est le cas de Cassandra dont les temps de réponses sont appréciables, mais le résultat n'est pas garanti à 100% lorsque le nombre de mises à jour simultanées devient important.

#### Le triangle de CAP et les bases de données :

 Grâce à ce théorème de CAP, il est alors possible de classer toutes les bases de données en les plaçant sur le "triangle de CAP", tout en ajoutant des codes couleurs pour chaque modèle de stockage présenté dans le chapitre précédent.

* Nous pouvons constater que les bases de données relationnelles se retrouvent sur la face CA du triangle, combinant disponibilité et cohérence. Nous retrouvons bien MongoDB pour le couple CP (cohérence et distribution) mais également les solutions orientées colonnes comme HBase ou BigTable.
* Le couple AP (Disponibilité et distribution) regroupe le plus grand nombre de solutions NoSQL. En effet, la plupart cherchent les performances en relâchant volontairement la cohérence. Nous y retrouvons principalement des solutions orientées clé-valeur et graphes, mais également orientées documents (clé-valeur étendu).
* Certaines solutions proposent également de modifier la politique de gestion de la concurrence (DynamoDB, CouchBase, Cassandra, MongoDB, CosmosDB...), dans ce cas, ils changent simplement de face sur ce triangle en passant de CP à AP.
* L'avantage de ce triangle CAP est de fournir un critère de choix pour votre application. Vous pouvez ainsi vous reposer sur vos besoins en terme de fonctionnalité, de calculs et de cohérence. Le triangle servira de filtre sur les myriades de solutions proposées.

## la base de données MongoBD :

MongoDB est une base de données NoSQL orientée documents. Comme nous le verrons, l’ensemble du système tourne autour de cette gestion de documents, y compris le langage d’interrogation, ce qui en fait son point fort. Nous allons nous attaquer dès maintenant à la mise en place d’un serveur Mongo et comment intégrer vos données dans cet environnement.



### L’ architecture de ReplicaSet :

Nous avons pu voir comment interroger des données, mais nous n’avons pas encore attaqué le problème de la tolérance aux pannes. MongoDB utilise pour cela une architecture basée sur le principe « maître/esclave .

Le serveur primaire "Primary", à qui toutes les requêtes sont envoyées (lecture/écriture), va s’occuper de gérer la cohérence des données. Ainsi, lors d’une mise à jour, une réplication est effectuée sur les serveurs secondaires "Secondary".



Toutefois, si l’on reste sur une architecture de la sorte, nous ne sommes pas tolérant aux pannes puisque le serveur primaire reste un point critique. Pour cela, nous allons rajouter un arbitre "arbiter", dont la tâche est de vérifier l’état du réseau. Ainsi, dès que le primary tombe, un système de vote va permettre d’élire le secondary qui remplacera le serveur tombé en panne. Nous sommes donc tranquilles : cette architecture tient tant qu’au moins un serveur tourne. C’est ce que l’on appelle un **ReplicaSet**.

Un ReplicaSet doit contenir au minimum 3 serveurs (1 Primary et 2 Secondary) pour garantir un minimum de tolérance aux pannes. Un ReplicaSet peut contenir jusqu’à 50 serveurs ; toutefois lorsqu’un vote a lieu pour élire le nouveau Primary, au maximum 7 peuvent participer à cette élection (les premiers ayant répondu à l’Arbiter).

## MongoDB et Hadoop

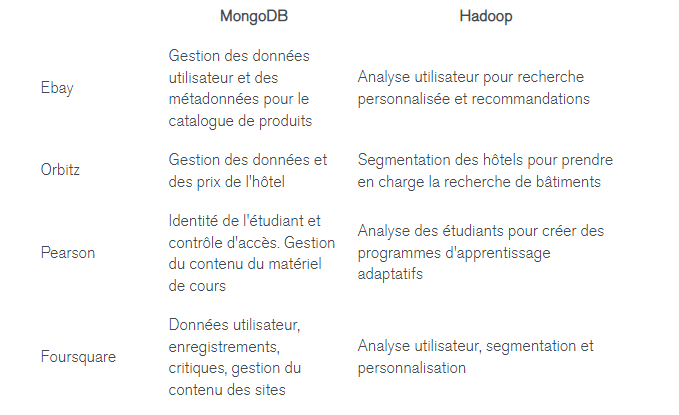
De nombreuses organisations exploitent ensemble les capacités de Hadoop et de MongoDB pour créer des applications Big Data complètes:

* MongoDB alimente l'application opérationnelle en ligne et en temps réel, au service des processus métier et des utilisateurs finaux, en exposant les modèles d'analyse créés par Hadoop aux processus opérationnels.
* *Hadoop utilise les données de MongoDB et les mélange avec des données d'autres sources pour générer des analyses sophistiquées et des modèles d'apprentissage automatique. Les résultats sont transférés à MongoDB pour servir des processus opérationnels plus intelligents et sensibles au contexte - c'est-à-dire proposer des offres plus pertinentes, identifier plus rapidement les fraudes, mieux prédire les taux d'échec des processus de fabrication*.

### Comment les organisations utilisent MongoDB avec Hadoop

Les utilisateurs doivent mettre les résultats analytiques de Hadoop à la disposition de leurs applications opérationnelles en ligne. Ces applications ont des demandes d’accès spécifiques auxquelles HDFS ne peut satisfaire, notamment:

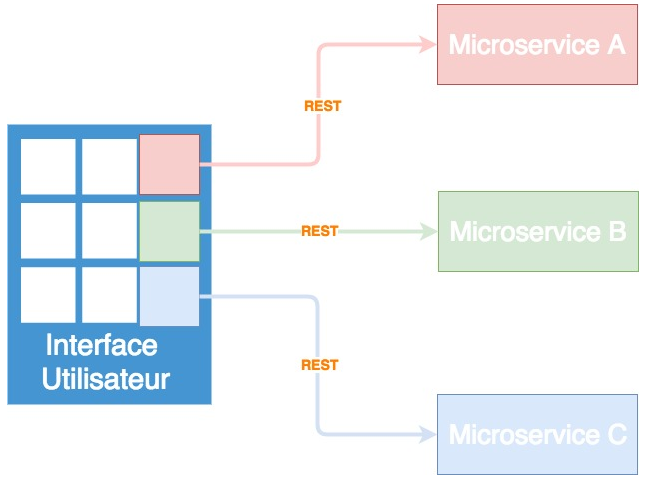
* Réactivité des requêtes en latence milliseconde.
* Accès aléatoire à des sous-ensembles de données indexés.
* Prise en charge en temps réel de requêtes ad hoc et d’agrégations expressives sur les données, rendant les applications en ligne plus intelligentes et contextuelles.
* Mise à jour en temps réel de données qui changent rapidement lorsque les utilisateurs interagissent avec des applications en ligne, sans avoir à réécrire l'intégralité du jeu de données. La gestion des analyses en temps réel depuis Hadoop jusqu'aux applications et aux utilisateurs en ligne nécessite l'intégration d'une couche de base de données opérationnelle hautement évolutive et extrêmement flexible.
* Le tableau suivant fournit des exemples de clients utilisant MongoDB avec Hadoop pour alimenter des applications Big Data.



# Généralités sur les Microservies

## le principe de l'architecture Microservices ?

L'architecture Microservices propose une solution en principe simple : **découper** une application en **petits services**, appelés Microservices,**parfaitement autonomes** qui exposent une API REST que les autres Microservices pourront consommer.



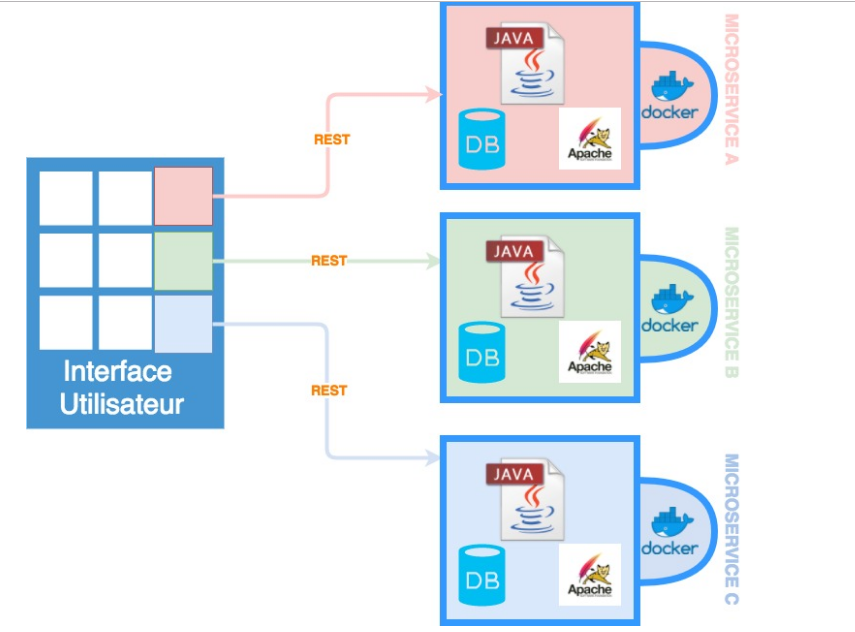
La figure ci-dessus représente un schéma simplifié d'une application basée sur l'architecture Microservices. Cette application affiche par exemple un produit à vendre. Cette "fiche produit" est donc constituée par exemple d'une photo, d'un descriptif et d'un prix. Dans le schéma, l'interface utilisateur fait appel à un Microservice pour chaque composant à renseigner. Ainsi, celle-ci peut faire une requête REST au **Microservice A**, qui s'occupe de la gestion des photos des produits, afin d'obtenir celles correspondant au produit à afficher. De même, les **Microservices B et C** s'occupent respectivement des descriptifs et des pricing.

Vous voyez alors, à travers cet exemple, quelle est l'idée générale de l'architecture Microservices. Dans une architecture traditionnelle, vous avez une application au format WAR qui comporte tous les composants. Lorsqu'un utilisateur demande une fiche de produit, l'application applique sa logique interne et va puiser dans une base de données, puis produit un HTML final.

Dans l'exemple que je vous ai donné, l'interface utilisateur est elle-même un Microservice qui a pour responsabilité d'appeler les autres Microservices et de rassembler donc cette "fiche produit" partie par partie avant de la servir à l'utilisateur final.

**Chaque Microservice est parfaitement autonome :** il a sa propre base de données, son propre serveur d'application (tomcat, jetty, etc.), ses propres librairies et ainsi de suite. La plupart du temps ces Microservices sont chacun dans un container [Docker](https://www.docker.com/), ils sont donc totalement indépendants y compris vis-à-vis de la machine sur laquelle ils tournent.

Voici une vue plus détaillée de notre architecture :



### Comment l'architecture Microservices répond-elle à ces problèmes ?

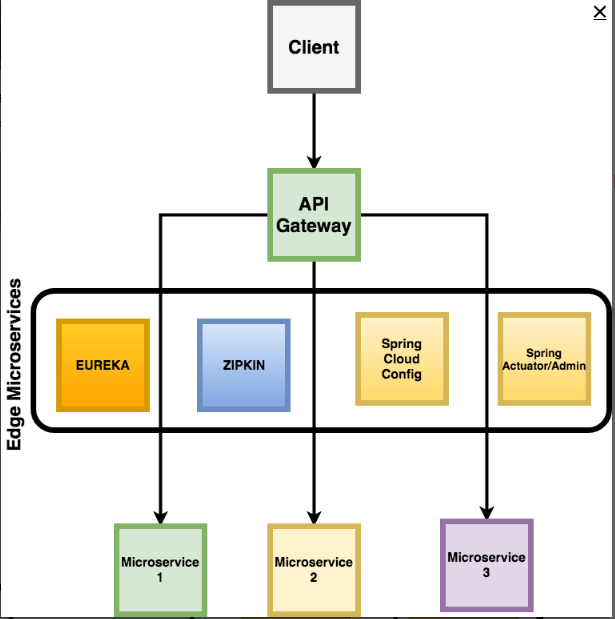
Chacun des Microservices dans notre schéma , comme je l'ai dit plus tôt, est parfaitement autonome, ce qui fait que notre application peut parfaitement profiter de tout ce que le Cloud peut offrir. Ainsi, on peut, par exemple, **dupliquer le Microservice A** sur plusieurs serveurs. Le Microservice qui s'occupe de l'interface utilisateur (UI) va appeler une des instances de ce service et recevoir la même réponse, quelle que soit l'instance choisie. Nous verrons plus tard qu'il existe d'autres **Microservices spécialisés**dans le dispatching des requêtes vers les différentes instances. Au final, notre application est parfaitement élastique : vous pouvez lui envoyer 5 utilisateurs ou 5 millions, il suffit d'augmenter ou de diminuer le nombre d'instances en services.

Quand on voudra mettre à jour l'application, il suffira de **cibler directement le Microservice responsable** de la fonctionnalité en question. Ainsi, si on dispose d'une nouvelle technologie de pricing qui permet de proposer des prix compétitifs en fonction de la concurrence en temps réel, tout ce qu'il y a à faire est de modifier le Microservice C et de le redéployer. À aucun moment vous n'aurez à vous soucier de l'ensemble de l'application ou de quelles répercussions vos changements pourraient avoir sur celle-ci. De cette manière, le processus de mise à jour est d'autant plus rapide que les Microservices sont par nature petits et relativement simples. Le risque de bug ou d'indisponibilité de l'application devient très bas, car vous n'avez pas à builder votre application dans un grand WAR puis à croiser les doigts pour que le nouveau code soit parfaitement compatible et fonctionne correctement une fois déployé.

## Les ****Edge Microservices****

Les **Edge Microservices** sont des Microservices spécialisés dans l'orchestration des Microservices centraux responsables de la logique de l'application.

Les Edge Microservices les plus populaires sont ceux publiés et utilisés par Netflix pour sa plateforme. Ils sont en grande partie regroupés sous Spring Cloud et disposent de fonctionnalités pour fonctionner nativement ensemble.



## Solutions Microservices utilisé

### Spring boot :

Spring Boot est un framework qui facilite le développement d'applications fondées sur Spring en offrant des outils permettant d'obtenir une application packagée en jar , totalement autonome. Ce qui nous intéresse particulièrement, puisque nous essayons de développer des Microservices !

Spring et Spring MVC sont de formidables outils quand on essaye de développer une application web. Néanmoins, un de leurs plus gros problèmes est la configuration. Si vous avez déjà développé une application avec ces outils, vous avez dû remarquer que votre application est bardée de fichiers XML qui indiquent les configurations des servlets, des vues, des contenus statiques, etc. Ces fichiers de configuration deviennent un vrai challenge lorsque vous avez une application complexe.

Pour simplifier cette configuration, Spring Boot propose **2 fonctionnalités principales** que nous allons voir dans la suite de ce chapitre :

* **l'auto-configuration**: Cette fonctionnalité est la plus importante de Spring Boot. Elle permet de configurer automatiquement votre application à partir des jar trouvés dans votre Classpath. En d'autres termes, si vous avez importé des dépendances, Spring Boot ira consulter cette liste puis produira la configuration nécessaire pour que tout fonctionne correctement.
* **les starters:** Les starters viennent compléter l'auto-configuration et font gagner énormément de temps, notamment lorsqu'on commence le développement d'un Microservice. Un starter va apporter à votre projet un ensemble de dépendances, communément utilisées pour un type de projet donné. Ceci va vous permettre de créer un "squelette" prêt à l'emploi très rapidement.



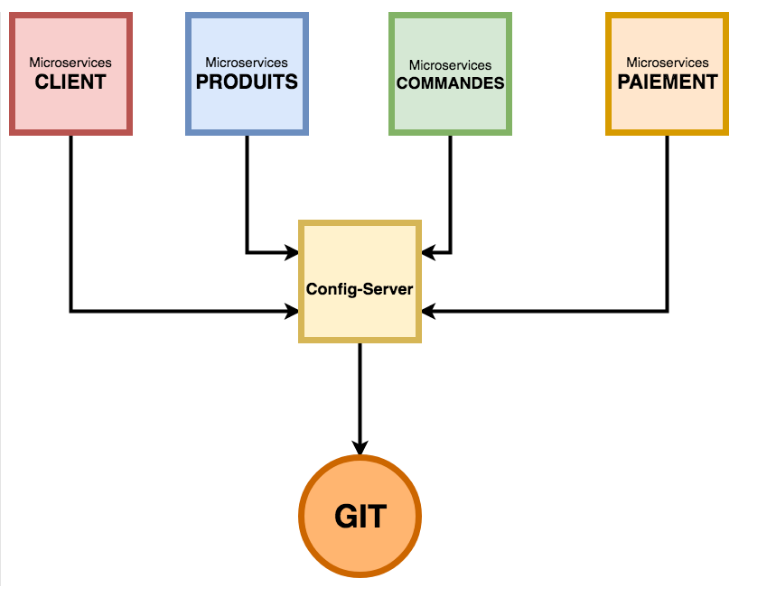
### Spring data MongoDB :

Spring Data MongoDB est un autre framework de Spring qui facilite grandement l'utilisation de JPA. Il va nous permettre de **générer toutes sortes d'opérations vers la base de données**, sans que nous ayons à écrire la moindre requête, ni même la moindre.



### Spring cloud Config :

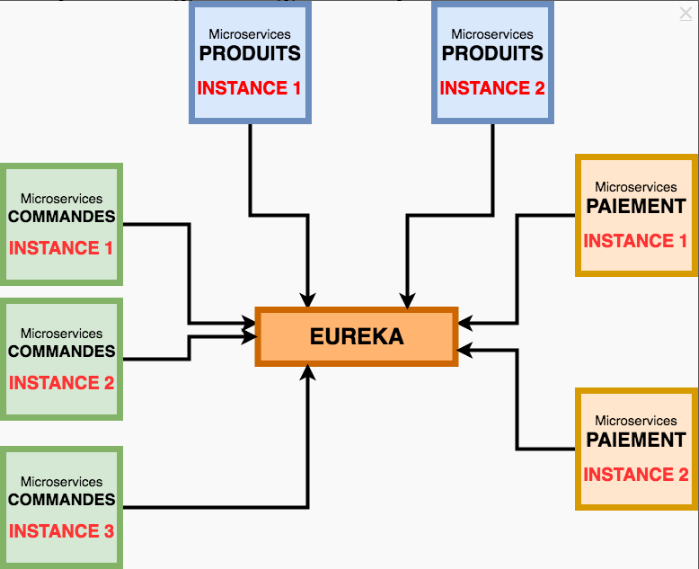
Spring Cloud Config fournit un support côté serveur et côté client pour la configuration externalisée dans un système distribué. Avec Config Server, vous disposez d’un emplacement central pour gérer les propriétés externes des applications dans tous les environnements. Les concepts sur le client et le serveur sont identiques à ceux de Spring **Environment**et **PropertySource** les abstractions, donc elles s’intègrent très bien avec les applications Spring mais peuvent être utilisées avec n’importe quelle application fonctionnant dans toutes les langues Lorsqu'une application parcourt le pipeline de déploiement de dev à test et en production, vous pouvez gérer la configuration entre ces environnements et vous assurer que les applications disposent de tout ce dont elles ont besoin pour s'exécuter lors de leur migration. L'implémentation par défaut du système de stockage du serveur utilise git. Il prend donc facilement en charge les versions étiquetées des environnements de configuration, tout en étant accessible à un large éventail d'outils de gestion du contenu. Il est facile d'ajouter des implémentations alternatives et de les brancher avec la configuration Spring.



### Eureka:

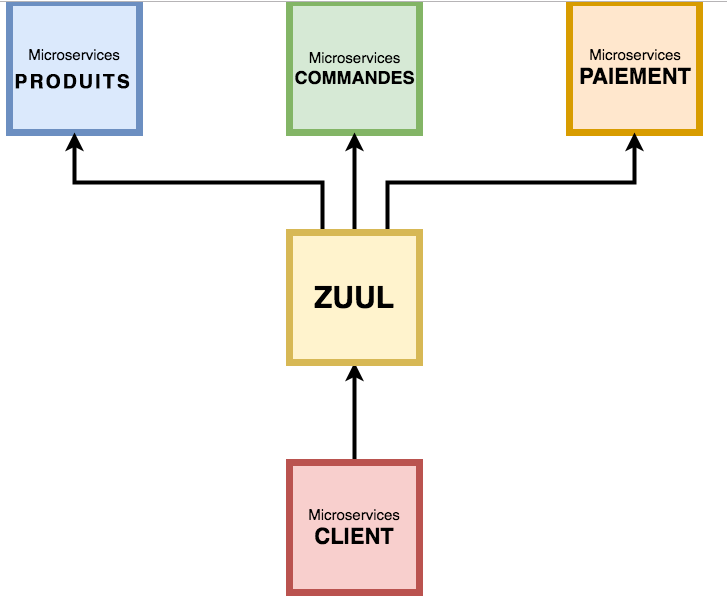
Quand votre application répond à une montée en charge et que vous avez plusieurs instances de chaque Microservice, il est vital de pouvoir garder un **registre de toutes les instances disponibles** afin de distribuer la charge entre celles-ci.

**Eureka** de Netflix remplit précisément cette fonction. Une fois en place, les instances des Microservices viennent s'enregistrer dans le registre d'Eureka. Pour appeler un Microservice, il suffira de piocher dans cette liste d'instances qu'Eureka expose via une API REST, Eureka offre un client capable de réaliser des opérations de récupération des listes d'instances.



### Spring Zuul :

**ZUUL** va se positionner comme le point d'entrée unique de notre application. Ainsi, quand ClientUi, par exemple, voudra appeler les Microservices, il passera par **ZUUL**. Ce dernier s'occupe de dispatcher, modifier et appliquer des filtres et des règles à ces requêtes.

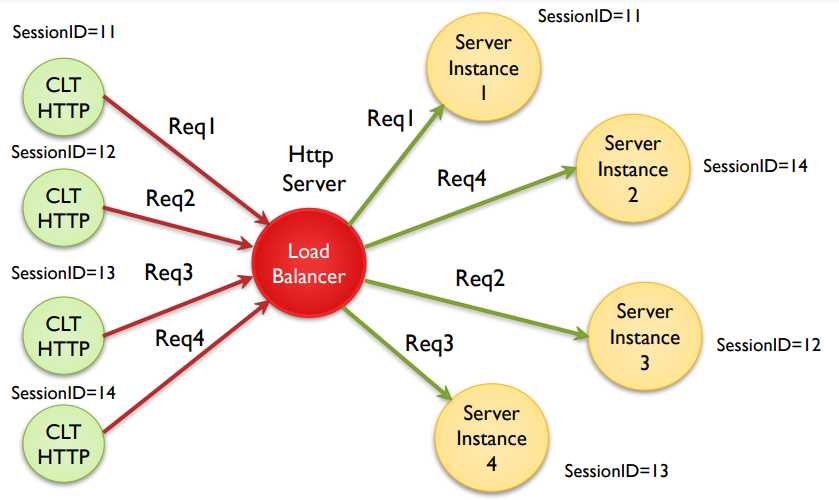


## Sécurité des applications web basées sur les Microservices :

### Systèmes d’authentification :

***Deux types :***

* **Statful** : Les données de la session sont enregistrés coté serveur d’authentification.
* **Statless** : les données de la session sont enregistrés dans un jeton d’authentification délivré au client.

****** Sécurité basée sur les sessions : Problème de montée en charge.

***pour simplifier l'autontification :***

* Gérer la session coté client .
* Chercher un moyen qui permet au serveur de faire confiance aux clients Web
* Sans avoir besoin de stocker sa session coté serveur

### Json Web Token (JWT) :

JSON Web Token (JWT) est un standard (RFC 7519) qui définit une solution **compacte** et **autonome** pour transmettre de manière sécurisée des informations entre les applications en tant qu'objet structuré au format JSON (Java Script Object Notation),Cette information peut être vérifiée et fiable car elle est signée numériquement.

* **Compact:** en raison de leur petite taille, les JWT peuvent être envoyés via une URL, un paramètre POST ou dans un en-tête HTTP. De plus, la plus petite taille signifie que la transmission est rapide.
* **Autonome:** Le JWT contient toutes les informations requises sur l'utilisateur, ce qui évite d'avoir à interroger la base de données plus d'une fois pour connaitre le détail de l’identité d’un client authentifié.

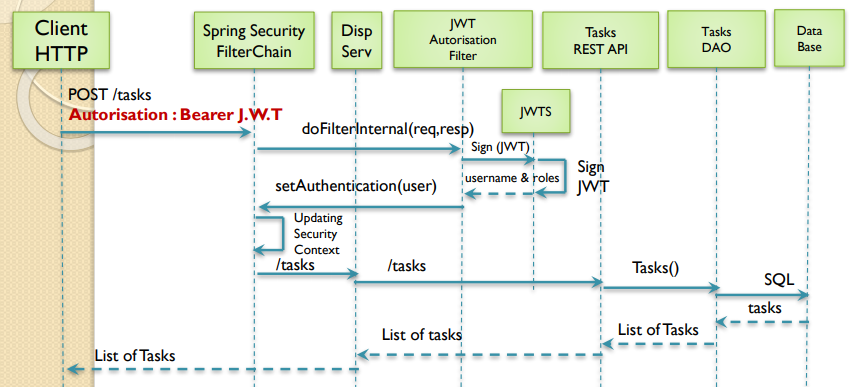
### Structure de JWT :

JWT est constitué de trois parties séparées par un point « . » :

* Header
* Payload
* Signature

### Authentification Spring Security avec JWT:

### Demander une ressource nécessitant l’authentification :



## Framwork Angular :

Angular (communément appelé "Angular 2+" ou "Angular v2 et plus") est un cadriciel (framework) côté client  basé sur [TypeScript](https://fr.wikipedia.org/wiki/TypeScript" \o "TypeScript) dirigée par l'équipe du projet Angular à Google et par une communauté de particuliers et de sociétés. Angular est une réécriture complète de [AngularJS](https://fr.wikipedia.org/wiki/AngularJS" \o "AngularJS), cadriciel construit par la même équipe.



# Partie pratique et Implémentation

## Description des tâches à réaliser :

### Introduction :

la réalisation d'une application web composé deux partie,la partie Front-end et la partie Back-end,partie Back-end composé deux microservices,la premieré gére l'Authentification et deuxiéme gére traitement des incidents informatiques.

Pour assurer la sécurité, les personnels doivent s’identifier avant qu’ils puissent accéder au menu du programme.

* ***Gérer les tickets :*** *une demande d’intervention correspond à un ticket ouvert par un agent, pour cela ce dernier doit accéder au système et remplir un formulaire qui va être par la suite traite et fermer par un technicien informatique.*

*Il existe trois types d’incidents ; matériel, réseau et logiciel. L’agent doit spécifier le type et remplir les champs correspondants et il sera obligé de donner une description du problème afin qu’il puisse valider l’incident.*

*Après la validation de l’incident, l’agent doit avoir à tout moment l’accès aux incidents au cas où il veut faire des modifications ou ajouter plus de détails sur le problème posé ou la fermer au cas où le problème est résolu.*

* ***Gérée les statistiques des incidents :*** *La gestion des statistiques faite par L’administrateur, il a le droit de traite plusieurs cas et les afficher sur des graphes pour avoir une visualisation globale sur le service on a plusieurs graphes :*

*Graphe 1 : présente le nombre des incidents selon les années.*

*Graphe 2 : présente le nombre des incidents selon type (Matériel, Réseau, Logiciel)*

*Graphe 3 : présente le nombre des incidents selon service.*

*Graphe 4 : affiche des statistiques de taux d’incident par mois (taux Résolu, taux au Fournisseur)*

*Graphe 5 : présente l’avis des utilisateurs (Non Satisfait, Satisfait, Très Satisfait).*

* ***Gérer les profils :*** *Les comptes des personnels sont créés initialement par l’administrateur qui définit leur login et leur mot de passe et leur privilèges (Agent/Technicien). Le personnel sera invité après à changer leur mot de passe.*
* ***Le traitement des incidents :*** *chaque incident est traité par un technicien selon sa catégorie (matériel, logiciel, réseau…) le technicien a le droit de consulter et clôturer une incident, si le technicien n’arrive pas à résoudre le problème il a le droit de contacter le fournisseur*
* ***Gérer le système :***

*Le système doit être capable de générer des statistiques et de les représenter sous forme des graphiques par exemple :*

*\*le nombre de visiteurs par jour*

### Les besoins non fonctionnels :

Les besoins non fonctionnels se basent sur le respect des normes de l’ergonomie et des interactions homme / machine qu’on fournit à l’application. Donc les contraintes techniques nécessaires qui garantissent la performance a notre solution se résument en :

* ***La rapidité de traitement:*** *En effet, vu le nombre important des activités et des utilisateurs, il est impérativement nécessaire que la durée d’exécution des traitements soit minimale.*
* ***La portabilité:*** *Il s’agit de minimiser l’effort pour se faire transporter dans un autre environnement matériel et/ou logiciel.*
* ***L’ergonomie des interfaces :*** *Les interfaces doivent être claire et bien structurées a ce propos un thème sera choisi et utilise au cours de développement de l’application pour assurer le bon choix du design. Les interfaces doivent obéir a un ensemble de critère ergonomiques tel que l’accessibilité, la lisibilité et la convivialité afin d’assurer l’interaction entre l’utilisateur et l’application.*
* ***L’intégrité :*** *La fonctionnalité offerte a chaque utilisateur doivent être restreintes a celle qui lui sont autorisées. L’information n’est modifiée que par les personnes y ayant droit dans ce cas nous définissons pour chaque utilisateur ces droit d’accès au système.*
* ***La fiabilité :*** *Le système doit traiter les failles de sécurité d’où le besoin d’un login et un mot de passe pour accéder au système. Les messages d’erreurs doivent identifier tous les cas d’erreurs de saisie et leur source.*

### IDENTIFICATION DES ACTEURS :

*Nous détaillons dans ce qui suit les acteurs intervenant dans notre application :*

Nous parvenons à une étape clé du processus. C’est elle qui grâce a l’étude réalisée dans la partie précédente mettra en valeur le rôle de chaque acteur du système ainsi que les fonctionnalités présentées plus haut

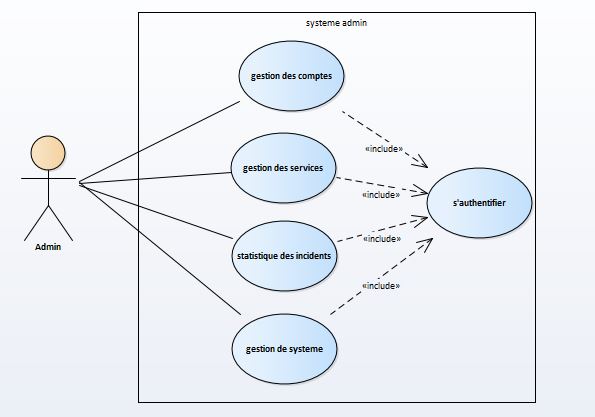
On a trois acteurs :

* *Administrateur*
* *Technicien*
* *Agent*

### Diagramme de cas utilisation

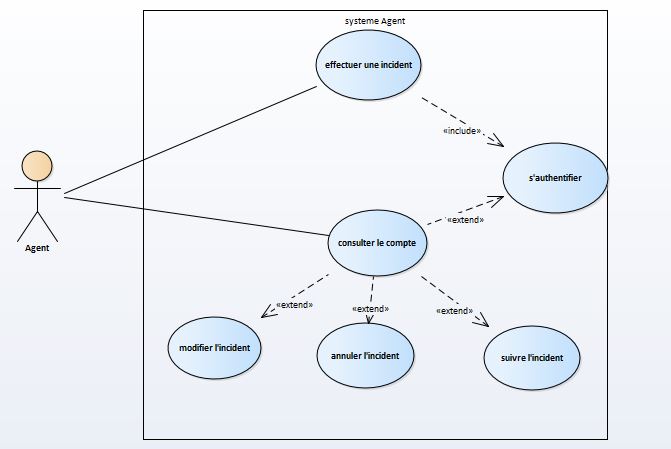
### *Diagramme associé à l’administrateur :*

*Le diagramme ci-dessous présente le diagramme de cas d’utilisation décrivant les différentes fonctionnalités de l’admin :*



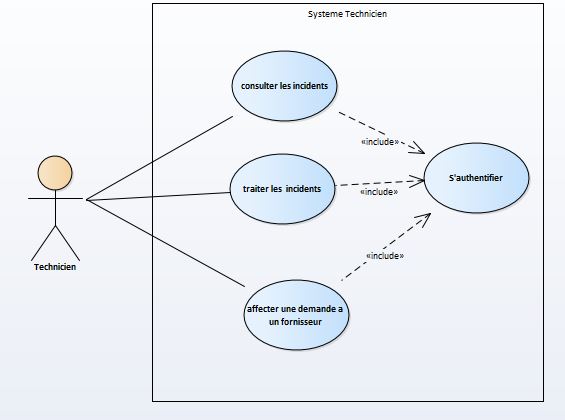
**Diagramme associé à l’agent :**

Le diagramme ci-dessous présente le diagramme de cas d’utilisation décrivant les différentes fonctionnalités de l’agent



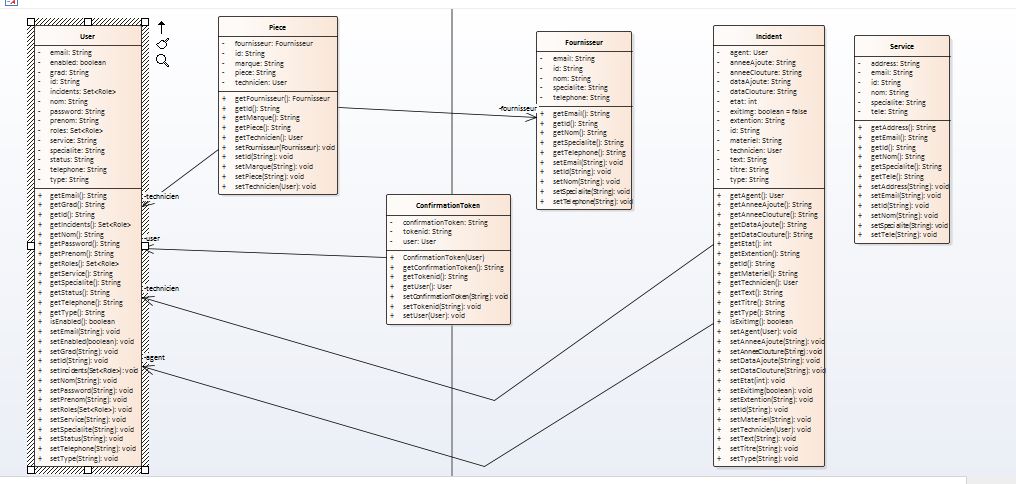
**Diagramme associé au technicien :**

Le diagramme ci-dessous présente le diagramme de cas d’utilisation décrivant les différentes fonctionnalités de technicien



### Diagramme de classes :

Le diagramme de classes constitue un élément très important de la modélisation : il permet de définir quelles seront les composantes du système final, voici le diagramme de classe pour l’application développé.

**