ABSTRACT 2

CAPITOLO 1: INTRODUZIONE 3

CAPITOLO 2: ARCHITETTURA 5

2.1 Cloudera 5

2.1.1 CDH 6

2.2 Hadoop 7

2.2.1 Architettura 9

2.3 HDFS 10

2.3.1 Architettura HDFS 12

2.3.2 Caratteristiche HDFS 13

2.4 Map Reduce 16

2.5 YARN 17

2.6 Apache Spark 20

2.6.1 Caratteristiche Apache Spark 23

2.6.2 RDD 24

2.7 Apache Kafka 26

2.8 Apache Zookeeper 29

2.9 Apache Kudu 31

2.10 Apache Impala 34

2.11 Apache Hive 36

2.12 Oozie 38

2.13 Hue 38

CAPITOLO 3: Approccio proposto 38

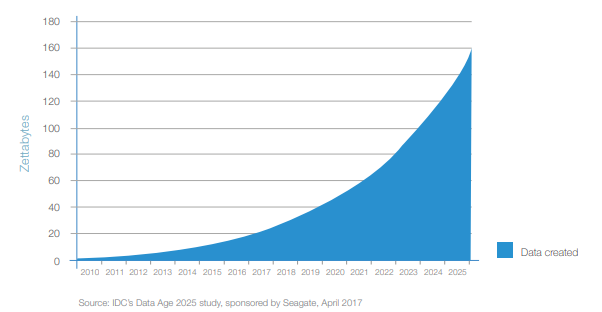
CAPITOLO 4: ANALISI DEI DATI E REALIZZAZIONE HEATMAP 39

# ABSTRACT

La tesi si basa sul lavoro svolto durante il periodo di Stage presso l’azienda Engineering Ingegneria Informatica S.p.a. Si è partiti dalla volontà di sviluppare un’applicazione che permettesse il monitoraggio di un’area geografica riguardante le caratteristiche di copertura di mezzi pubblici nella stessa.

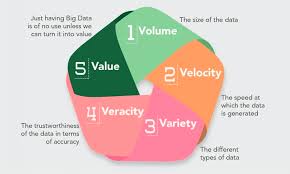
# CAPITOLO 1: INTRODUZIONE

Negli ultimi anni il volume e la complessità delle informazioni ha subito un incremento esponenziale. Uno studio fornito dall’International Data Corporation (IDC) prevede che entro il 2025 la sfera dei dati globale aumenterà fino a 163 miliardi di zettabyte (1ZB = 2^18 Byte), cioè più di 4 volte più grandi dei dati attuali; dei quali più di un quarto saranno real-time ed i dati IoT in tempo reale saranno più del 95%.



Insieme al continuo evolversi del fenomeno dei Big Data anche i metodi per immagazzinare ed analizzare la vasta quantità di dati hanno fatto grossi passi in avanti.

Andando a descrivere i Big Data, tipicamente, non si fa uso di una definizione formale ma si utilizza un modello noto come *Big Data 5V*, rappresentato nella figura.



Andiamo ad analizzare le caratteristiche fondamentali del modello:

* **Volume:** con questo termine si fa riferimento ad un’ingente massa di informazioni, come abbiamo detto pocanzi, dell’ordine dei ZettaByte se non di BrontoByte. Una delle conseguenze dell’aumento del volume è la necessità di andare oltre le sia in termini di capacità di memorizzazione dei dati, sia nella capacità computazionale.
* **Velocità:** una caratteristica molto influente nei Big Data è la rapidità con cui le informazioni vengono prodotte, acquisite ed analizzate. Le tecnologie attuali permettono inoltre di poter andare ad analizzare lo stream di dati anche senza la necessità di doverlo memorizzare in un database.
* **Varietà:** i Big Data comportano l’elaborazione di diversi tipi di dati raccolti da fonti eterogenee. Le fonti di dati possono coinvolgere fonti esterne e unità aziendali interne. In generale i Big Data sono classificati come dati strutturati, semi-strutturati e non strutturati; i primi sono i dati nei quali sono definiti lunghezza, volume e formato mentre i dati semi-strutturati sono quelli che possono essere parzialmente conformi ad un formato di dati specifico. I dati non strutturati sono dati non organizzati e non conformi ai formati di dati tradizionali.
* **Veridicità:** dato l’aumento costante di nuove sorgenti e della mole di dati, esiste la possibilità che non tutti i dati raccolti siano affidabili. Dunque durante l’elaborazione di set è importante che venga verificata la validità dei dati prima di procedere alla loro elaborazione.
* **Valore:** andando soltanto a collezionare i dati, serve a ben poco. Al contrario i dati da cui vengono raccolte informazioni aziendali aggiungono un vero e proprio valore all’azienda. Difatti il valore può essere espresso in base a quanto i dati siano degni di essere analizzati. Effettuare l’aggregazione non equivale all’aggiunta di un valore, ma ciò che conta è una analisi avanzata attraverso la quale puoi trarre informazioni utili.

# CAPITOLO 2: ARCHITETTURA

## 2.1 Cloudera



Cloudera Enterprise è una piattaforma per machine learning e analisi ottimizzata affinché il cloud sia:

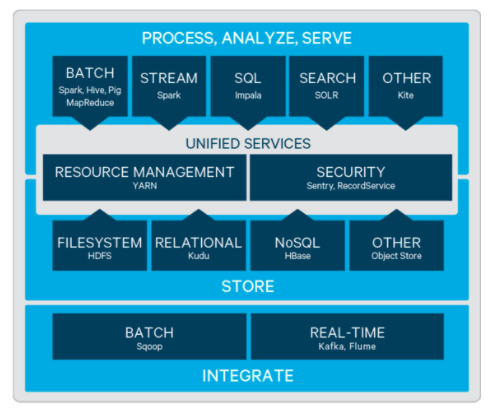
* **Unificato**: Riunisce i Data Warehouse, data scienze, Data Engineering e database su un’unica piattaforma integrata.
* **Ibrido**: Viene utilizzato dove e come è più conveniente ed efficace. Ciò fornisce flessibilità per lavorare sui dati ovunque essi siano, senza copie e senza spostamenti.
* **A livello aziendale**: La piattaforma può mettere a disposizione a più utenti petabyte di dati e fornisce gli strumenti per elaborare ed eseguire query sui dati, e sviluppare rapidamente data models. Vengono forniti diversi livelli di sicurezza a grana fine ed un controllo completo che impedisce l’accesso non autorizzato ai dati e dimostra la responsabilità per le azioni intraprese.

Cloudera Enterprise fornisce dei servizi per poter implementare:

* **Data Warehouse**: Gli utenti possono e esplorare ed iterare rapidamente i dati, eseguire report senza l’assistenza del reparto IT. I dati possono essere raggiunti da più utenti e da più strumenti. Tutto ciò viene implementato grazie a componenti come Apache Impala e Apache Hive.
* **Data Science:** Consente di creare e distribuire soluzioni AI su larga scala in modo efficiente e sicuro. La componente principale che permette ciò è Apache Spark.
* **Data Engineering:** Può supportare carichi di lavoro veloci e flessibili ma anche carichi che vanno ad addestrare modelli di machine learning su larga scala. Tutto ciò è offerto da Apache Spark, Hive su Spark, HDFS, YARN, HUE e Cloudera Manager.
* **Operational Database:** Un servizio di Database sicuro, a bassa latenza ed alta concorrenza, in grado di estrarre informazioni in tempo reale dei dati in continua evoluzione. Il servizio viene implementato grazie ad Apache Spark, Apache Kudu e HBase.

### 2.1.1 CDH

CDH, Cloudera Distribution Including Apache Hadoop, è la distribuzione più testata, completa e popolare di Apache Hadoop. CDH offre gli elementi principali di Hadoop attraverso un’interfaccia Web utente.



CDH fornisce alcune proprietà fondamentali nell’ambito della gestione dei dati, come:

* **Flessibilità:** Immagazzina qualsiasi tipologia di dato e può manipolarlo con framework differenti come elaborazione batch, machine learning e calcoli statistici.
* **Integrazione:** Diventa subito operativo su una piattaforma Hadoop completa, che opera con una vasta gamma di hardware e software.
* **Sicurezza:** Esegue il controllo sui dati sensibili.
* **Scalabilità:** Permette all’utente di ridimensionare ed estendere un’ampia gamma di applicazioni.
* **Disponibilità:** Esegue con sicurezza servizi mission-critical.
* **Compatibilità:** È in grado di sfruttare l’infrastruttura IT esistente.

## 2.2 Hadoop



Apache Hadoop è un framework open source che viene utilizzato per immagazzinare e processare in maniera efficiente una grande quantità dei dati non andando a considerare la potenza di una singola macchina ma sfruttando la potenza di calcolo e lo spazio di archiviazione di macchine facenti parte di un cluster. Il framework è progettato per scalare da un singolo server a migliaia di macchine, ognuna delle quali mette a disposizione le proprie risorse.

Hadoop viene sfruttato nell’ambito Big Data poiché presenta caratteristiche come:

**Scalabilità:** il framework permette di gestire ed elaborare grandi quantità di dati grazie alla possibilità di distribuire il carico di lavoro su più host, detti nodi. Inoltre Hadoop fornisce scalabilità orizzontale, per mezzo della quale è possibile aggiungere nuovi nodi senza periodi di downtime.

**Flessibilità:** Hadoop immagazzina in maniera efficiente i dati, indipendentemente dal fatto che siano strutturati (MySql), semi strutturati (XML, JSON) o non strutturati (Immagini o Video). Ciò significa che non è sempre necessario nel memorizzare i dati si definisca una struttura ma è possibile salvare i dati in qualsiasi formato, per poi gestirlo in maniera opportuna durante le elaborazioni.

**Affidabilità:** In Hadoop i dati sono replicati su più Data Node nel cluster, ciò garantisce la disponibilità dei dati qualora uno dei sistemi dovesse smettere di funzionare. L’affidabilità dunque è preservata dalla fault tolerance. Di default Hadoop crea tre copie di ogni blocco di dati e li immagazzina in nodi differenti, così da rendere i dati sempre disponibili.

**Costi:** Hadoop è open source ed utilizza hardware economico, a differenza dei database relazionali che richiedono hardware costoso e processori di alta fascia. Il costo richiesto per aggiungere nuove macchine, nodi, al cluster non è necessariamente alto data la possibilità di poter girare su hardware economico.

L’ecosistema Hadoop è costituito da varie componenti, le principali sono:

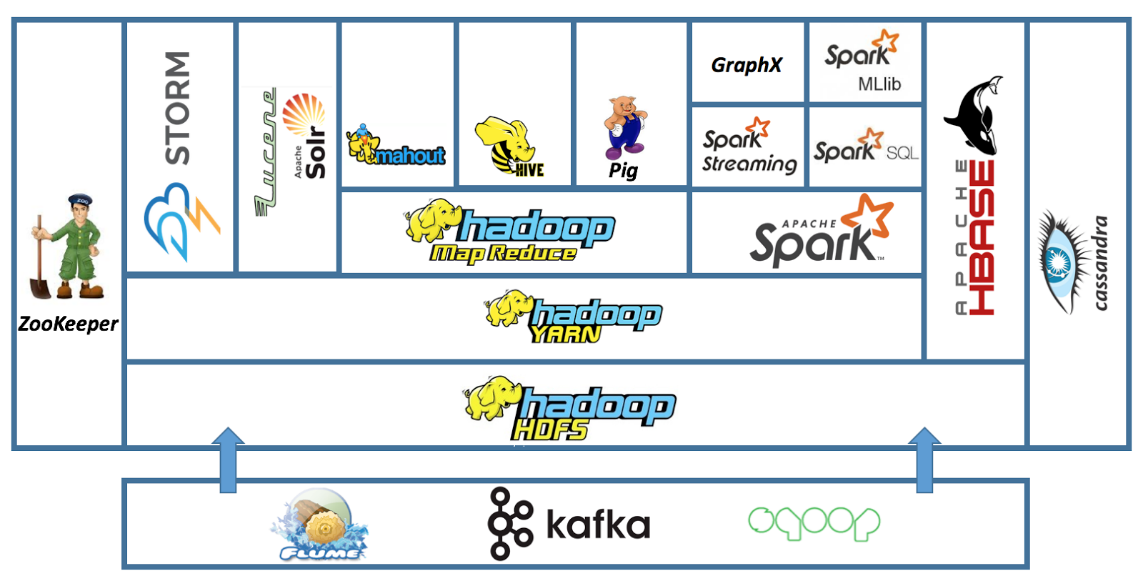
**Hadoop Common:** è una layer di livello software, che va a supportare gli altri moduli Hadoop.

**Hadoop Distributed File System:** è un file system distribuito tra in nodi all’interno di un cluster. Questo modulo esegue la replica dei dati così da renderli disponibili a fronte di eventuali guasti massimizzandone la disponibilità. È sviluppato in modo da poter fornire un elevato throughput e basse latenze per tutte le operazioni I/O.

**Hadoop YARN:** si occupa di gestire e monitorare i nodi appartenenti al cluster e l’utilizzo delle risorse.

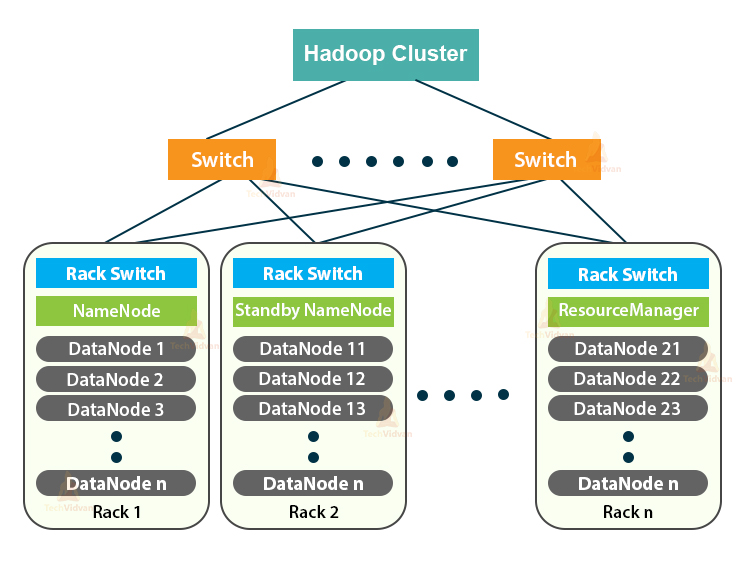
**Hadoop MapReduce:** è un framework che supporta i programmi ad eseguire la parallelizzazione sui dati. Il mapping prende in inputi i dati e li converte in un dataset che viene visto come coppie chiave-valore. L’output successivamente sarà consumato da operazioni di reduce per aggregarlo, fornendo così il risultato desiderato.

Come detto finora Hadoop si basa su quattro componenti ma è stato via via aggiornato con diversi moduli che offrono funzionalità diverse all’utente, così andando a formare l’intero ecosistema (figura)



### 2.2.1 Architettura

In un cluster Hadoop è un insieme di macchina, nodi, che sono interconnessi tra loro per eseguire operazioni in parallelo su una grande quantità di dati. Questa tipologia di cluster si differenzia dagli altri poiché sono progettati specificatamente per archiviare ed analizzare grandi quantità di dati, che siano strutturati o non strutturati. La dimensione del cluster può variare a seconda della necessità. I nodi sono organizzati all’interno dei data center in rack, connessi tra di loro tramite degli switch.



In un cluster Hadoop troviamo due diverse categorie di nodi, nodi master e nodi worker, che gestiscono ed eseguono le operazioni attraverso il file system distribuito di Hadoop(HDFS). I nodi master utilizzano generalmente hardware di qualità superiore ed includono NameNode, Secondary NameNode e JobTracker, ciascuno eseguito su una macchina differente. I nodi worker sono costituiti da macchine virtuali che eseguono i servizi DataNode e TaskTracker su hardware di base.

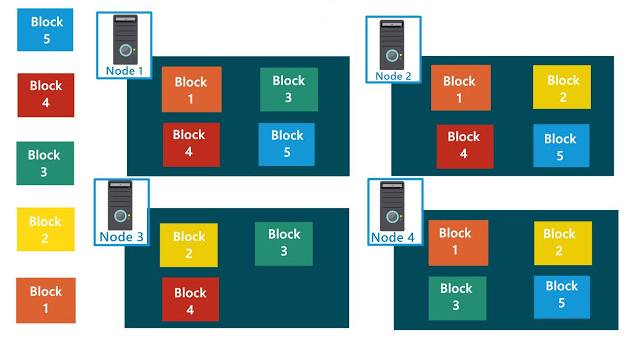
Un ambiente soggetto a grandi elaborazioni di dati e distribuito potrebbe avere la necessità di allocare nuove risorse per raggiungere l’obiettivo. La scalabilità di un cluster Hadoop può essere affrontata secondo due tipologie di approcci:

**Scalabilità orizzontale (scale out):** consiste nell’aggiungere più nodi al cluster. Viene adottata questa politica quando si hanno a disposizione dispositivi non di elevato costo, così da poterli rimpiazzare in caso di necessità.

**Scalabilità verticale (scale in):** consiste nell’aggiungere risorse ai nodi già appartenenti al cluster, andando a migliorare caratteristiche come memoria o CPU. Viene adottata questa politica qualora ci sia un numero limitato di host, ma non si voglia rinunciare alle performance.

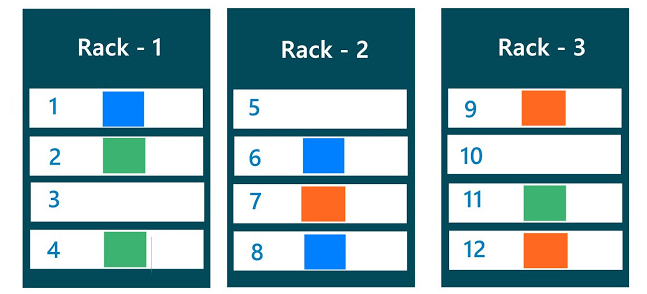
## 2.3 HDFS

HDFS è un file system distribuito basato su Java che permette di memorizzare grandi quantità di dati in un cluster Hadoop in maniera tale da garantire affidabilità e scalabilità. HDFS immagazzina i dati in maniera distribuita; i dati vengono divisi in porzioni e memorizzati nei differenti nodi del cluster. In questo modo HDFS fornisce a MapReduce un modo per andare ad elaborare un sottoinsieme dei dati, parallelamente su diversi nodi. HDFS divide, appunto, i file in piccoli porzioni noti come blocchi. I blocchi HDFS hanno una dimensione predefinita di 128MB, questa può essere aumentata o diminuita a seconda delle necessità. Il maggior vantaggio che si ha nella memorizzazione in HDFS è riguardante l’allocazione dei dati con dimensione inferiore a quella del blocco, difatti in questo caso la dimensione del blocco sarà uguale alla dimensione dei dati. Ciò giova sia in termini di tempi di ricerca del disco, sia nell’elaborazione. Per garantire affidabilità ogni blocco viene replicato su nodi differenti, per indicare il numero di copie esistenti si parla di replication factor.



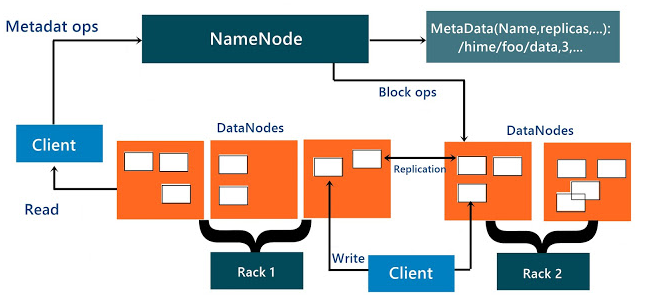
Nella figura sopra si può notare un cluster con 4 nodi che possiedono un fattore di replica pari a 3. La posizione delle repliche è studiata ed effettuata per poter reagire ad ogni tipo di guasto. Poiché i nodi sono organizzati in rack, i blocchi vengono collocati in maniera tale da prevenire sia un guasto su un singolo nodo, nel rack è presente un duplicato, sia ad un guasto sull’intero rack, un’altra replica viene collocata in un rack differente.

La topologia dei nodi all’interno del cluster è fondamentale per massimizzare le performance durante le operazioni di I\O. Il framework, durante le attività, va a minimizzare il consumo di banda e di latenza, garantendo che vengano letti sempre i dati dal nodo più vicino che ha lanciato l’operazione; può accadere che questo sia sulla stessa macchina, o su una copia presente nello stesso rack o nel peggiore dei casi sia presente una copia in un rack vicino.



### 2.3.1 Architettura HDFS

Il file system di Hadoop è basto su un’architettura di tipo master/slave. In ogni cluster è sempre presente un singolo nodo master, chiamato NameNode, e un numero variabile di slave, chiamati DataNode.



**NameNode:** è il nodo master che gestisce i blocchi presenti all’interno dei DataNodes. È un server con alta disponibilità che gestisce il File System Namespace e controlla l’accesso ai file da parte dei clients. Ha il compito di registrare i metadati di tutti i file presenti nel cluster come la dimensione dei blocchi, i permessi e la gerarchia ed ogni loro modifica. Mantiene un registro di tutti i blocchi ed in quali nodi essi si trovino. Il NameNode gestisce il fattore di replica di tutti i blocchi e, qualora sia presente un errore nel DataNode, ha il compito di scegliere nuovi DataNode per creare nuove repliche, per bilanciare l’utilizzo del disco e per gestire il traffico tra i vari Datanode.

**DataNode:** sono i nodi slave nei quali vengono immagazzinati i dati. I DataNode si occupano di gestire le operazioni creazione, cancellazione e replica dei dati; inoltre rispondono alle richieste di lettura e scrittura di basso livello dai client del file system. Per segnalare l’integrità generale di HDFS, i DataNode inviano un segnale periodico di HeartBeat.

### 2.3.2 Caratteristiche HDFS

**ALTA DISPONIBILITA’**

HDFS inizialmente era soggetto ad un problema legato al single point of failure, poiché era presente soltanto un NameNode responsabile all’accesso ai dati sui DataNode. Una volta che questo fosse stato soggetto a guasti il cluster sarebbe divenuto inutilizzabile.

Per fronteggiare il problema è stato introdotta la possibilità di creare due NameNode all’interno del cluster; uno dei due nodi è attivo mentre l’altro è in uno stato di hot standby, cioè è pronto a subentrare prontamente al primo in caso di guasti o malfunzionamenti. Affinché il tutto funzioni correttamente il NameNode attivo e quello in hot standby devono essere sempre sincronizzati riguardo la posizione dei blocchi e il namespace a cui si riferisce. La sincronizzazione è resa possibile grazie ai DataNode che inviano automaticamente le informazioni attraverso due tipi di messaggi: Heartbeat e Block Report.

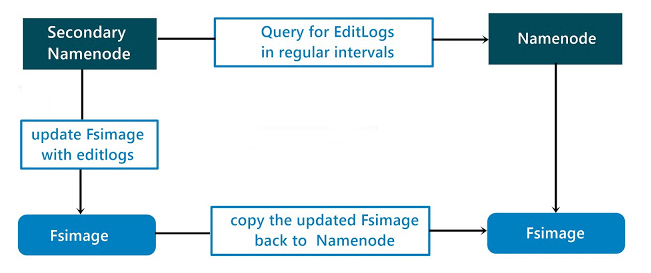
**Heartbeat:** è un messaggio che trasporta informazioni riguardanti il DataNode come la capacità di memoria disponibile o la cache utilizzata. Questo segnale viene inviato periodicamente dal DataNode al NameNode per indicare la presenza all’interno del cluster ed il proprio corretto funzionamento. L’intervallo di trasmissione è settato di default a tre secondi, ma è possibile configurarlo. Nel momento in cui il NameNode non riceve un segnale per un certo timeout prestabilito, considererà quello specifico DataNode come fuori servizio e di conseguenza non gli inoltrerà nessuna richiesta di I\O. I dati presenti su quel DataNode non saranno più disponibili per HDFS e, data la perdita, il fattore di replica di alcuni blocchi potrebbe essere inferiore a quello prestabilito; il NameNode, a questo punto, pianifica ed avvia il processo di creazione di nuove copie di quel blocco.

**BlockReport:** sono messaggi attraverso il quale i DataNode sincronizzano periodicamente i blocchi allocati localmente con metadati che rappresentano quei blocchi. Questi report sono inviati dal DataNode per indicare un set di blocchi validi al NameNode; quest’ultimo confronta la lista ricevuta con i propri metadati. I block report vengono generati in seguito alla scansione del file system, in modo che il NameNode sia sempre aggiornato sullo stato delle repliche dei vari nodi.

**PERSISTENZA**

La gestione del namespace HDFS è affidata al NameNode, questa viene effettuata tramite l’ausilio di due file, FsImage e EditLog. Il primo è un file che contiene la composizione totalitaria del file system fin da quando viene avviato il NameNode; il secondo contiene le modifiche più recenti, relative al file system, facendo riferimento all’FsImage più recente. Ciò avviene così da non avere la necessità di dover creare una FsImage ogni qualvolta sia in atto una modifica. Quando il NameNode si avvia, legge FsImage e EditLog dal disco applica il processo di checkpoint cioè, dopo aver applicato tutte le transazioni da EditLog alla rappresentazione in memoria di FsImage ed essersi scaricato il tutto su una nuova FsImage, tronca il vecchio EditLog poichè, ovviamente, tutte le modifiche sono salvate sulla nuova FsImage. L’operazione di checkpoint viene dunque effettuata ogni qualvolta un nodo viene avviato.

Data la grande importanza di FsImage e di EditLog per HDFS, è necessario assicurarsi che non venga a mancare la loro integrità poiché renderebbe l’intero sistema inutilizzabile. Difatti è possibile configurare i NameNode in modo tale che vadano a creare copie multiple, che vanno tenute aggiornate.



Negli anni è stata introdotta la figura del Secondary NameNode, questo non è una replica ma ha la funzione di eseguire periodicamente l’aggiornamento di FsImage, evitando così che l’EditLog raggiunga dimensioni eccessive, di fatto eseguendo il processo di checkpoint. Come si può vedere dalla figura, il processo si evolve secondo alcuni steps:

1. Il NameNode periodicamente recupera i file di EditLog e di FsImage.
2. Viene prodotto un nuovo FsImage, e viene inviato al NameNode.
3. Al successivo avvio, il NameNode andrà a caricare la nuova versione del file FsImage.

**STAGING**

La richiesta del client per creare un file non raggiunge immediatamente in NameNode, ma i dati inizialmente vengono memorizzati in una porzione temporanea del file system del client. Il NameNode viene effettivamente contattato soltanto quando o il file viene chiuso, oppure quando riceve una quantità di dati tale da riempire un blocco HDFS. A questo punto i dati vengono scritti su HDFS, secondo il seguente procedimento:

1. Il NameNode inserisce il nome del file nella gerarchia del file system e gli assegna un blocco di dati.
2. Il NameNode risponde alla richiesta del client inviando un blocco dati, specificando il destinatario tramite l’identità, al DataNode.
3. Il client scarica il blocco dati dal file temporaneo in locale al DataNode specificato, se il file viene chiuso anche i dati non scaricati vengono trasferiti al DataNode.
4. Il NameNode memorizza l’operazione inserendola nel file nell’EditLog. Nel momento in cui il NameNode dovesse andare in contro a qualche malfunzionamento prima della chiusura del file, questo andrebbe perso.

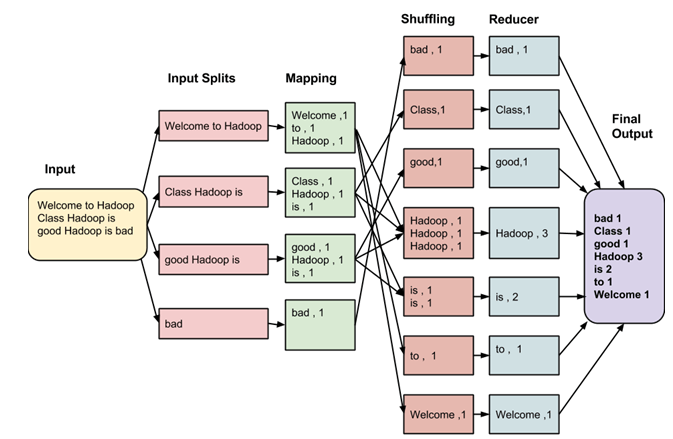
In HDFS, come già visto, il fattore di replica è un parametro configurabile; se questo è maggiore di uno, la fase di staging è seguita da quella di replication pipelining. Quando viene accumulato un blocco completo di dati, il client recupera un elenco contenente i DataNode che ospiteranno una replica di quel blocco. Il primo DataNode inizia a ricevere i dati, li scrive nel suo repository locale e li trasferisce a tutti gli altri DataNode della lista.

## 2.4 Map Reduce

Map Reduce è il layer di elaborazione di Hadoop ed è progettato per elaborare una grande quantità di dati in parallelo suddividendo il lavoro in una serie di attività indipendenti. L’elaborazione è composta da due step fondamentali:

**Map:** i dati di input, vengono inviati al nodo master, questo li divide in blocchi e distribuisce il lavoro ai nodi worker.

**Reduce:** in questa fase, il nodo master raccoglie tutti gli output della fase di map e trasforma le coppie (chiave, valore), in una lista di valori che hanno la stessa chiave svolgendo così la fase di Shuffle. L’ordine può essere lessicografico o può essere impostato dall’utente stesso.



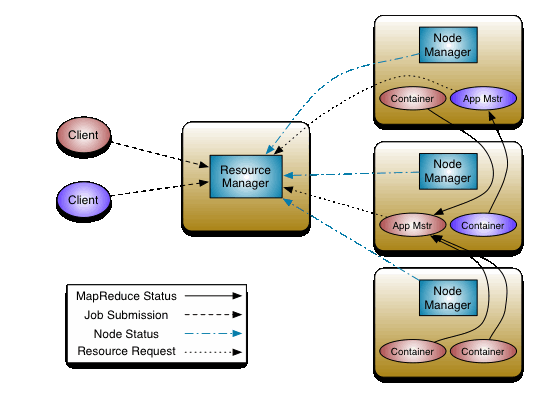
Un esempio semplice che per capire bene come funzioni il layer è il problema del Word Count, mostrato nella figura.

Il file nell’esempio viene suddiviso inizialmente in quattro blocchi. Per ogni blocco viene istanziato un mapper, che riceve una riga di testo alla volta (in questo caso la chiave è irrilevante, è NULL). Ogni mapper processa l’input, divide le parole e produce per ognuna una coppia (parola,1), che sta ad indicare che quella parola appare una sola volta. Queste coppie vengono memorizzate in modo temporaneo sul file system locale dei nodi dei Mapper. A questo punto subentra lo Shuffling,una fase nella quale viene applicata una funzione Hash alla chiave delle coppie. Successivamente, nella face di Reduce, le coppie che condividono la chiave sono aggregate ed inviate allo stesso Reducer in base all’Hash applicato precedentemente. Il Reducer, dopo aver ricevuto il numero di occorrenze della chiave, somma i valori unitari ottenendo le occorrenze finali presenti nel testo. Il risultato finale (parola, occorrenza) viene scritto su HDFS.

Una struttura alternativa è data dall’introduzione di una nuova fase, il Combine, che opera in maniera analoga al Reducer, ma agendo sui file locali prima di effettuare l’invio sulla rete. Dunque non si avrebbero più tante occorrenze di (parola, 1), che potrebbe portare a diminuzione di performance, ma si andrebbe ad eseguire una precomputazione locale, andando a ridurre il lavoro successivo svolto dal Reducer.

## 2.5 YARN

YARN è una delle quattro componenti principali di Apache Hadoop che va a supportare il framework nello scheduling delle varie applicazioni da eseguire e nella gestione delle risorse all’interno del cluster. Con YARN si è ampliato il potenziale uso di Hadoop, in particolare si è introdotto un nuovo approccio che va a slegare la gestione e la pianificazione delle risorse del cluster dalla componente di elaborazione dati di MapReduce, consentendo così al framework di supportare vari tipi di elaborazione ed una più ampia gamma di applicazioni; un esempio è la possibile esecuzione di query interattive ed analisi sui dati in tempo reale. Nel MapReduce l’allocazione delle risorse e la sincronizzazione erano relegati al JobTracker; ciò poteva influire sulle prestazioni andando a creare dei colli di bottiglia e poteva presentarsi un problema legato alla scalabilità con l’aumento delle dimensioni del cluster o del numero di applicazioni associate.



YARN decentralizza l’esecuzione ed il monitoraggio dell’elaborazione andando a separare i compiti a diverse componenti. L’idea principale è quella di avere un ResourceManager Globale ed un ApplicationMaster per ogni applicazione. Il framework per l’elaborazione dei dati è formato da due demoni, il ResourceManager e il NodeManager:

**ResourceManager:** è il demone master, è presente in singola istanza all’interno de cluster e ne controlla la gestione delle risorse; si occupa inoltre dell’assegnazione dei singoli task ai worker.

**NodeManager:** è il demone slave, ed è presente in singola istanza in ogni nodo del cluster. Si occupa del monitoraggio dell’utilizzo delle risorse tenendo aggiornato costantemente il ResourceManager con messaggi di report.

Il ResourceManager a sua volta, presenta due componenti principali, un ResourceScheduler ed un ApplicationsManager:

**ResourceScheduler:** è la componente responsabile dell’allocazione delle risorse richieste dalle applicazioni in esecuzione. Non offre un monitoraggio dello stato dell’applicazione, né una garanzia sul riavvio di attività non andate a buon fine; effettua lo scheduling secondo i requisiti di risorse delle applicazioni e lo fa sulla base di una nozione di un insieme di elementi che comprende memoria, CPU, disco ecc...

**ApplicationsManager:** si occupa della gestione della lista delle applicazioni che vengono inoltrare. Dopo che è stata fatta la richiesta da parte del client, per prima cosa convalida se il requisito dell’applicazione può essere soddisfatto dall’Application Master. Se le risorse sono disponibili, inoltra l’applicazione allo Scheduler, altrimenti questa verrà rifiutata.

Il ResourceManager, in particolare il ResourceScheduler, è responsabile del rilascio di un Container ad un ApplicationMaster; questo è un insieme di risorse quali memoria, CPU e disco. Mentre l’ApplicationMaster è quell’istanza che va a negoziare le risorse con il ResourceManager e coopera con il NodeManager per monitorare le risorse concesse per una determinata applicazione.

Ogni applicazione può essere eseguita attraverso il seguente procedimento:

1. Un client contatta il ResourceManager.
2. L’ApplicationManager negozia un Container ed avvia l’istanza dell’ApplicationMaster.
3. L’ApplicationMaster richiede i Container e comunica con il NodeManager per avviare i Container concessi. Durante l’esecuzione l’applicazione fornisce informazioni riguardo lo stato e sull’avanzamento all’ApplicationMaster.
4. L’ApplicationMaster segnale il completamento dell’applicazione al ResourceManager, i Container vengono deallocati. Alla fine, anche l’ApplicationMaster annulla la registrazione con il ResourceManager, viene rilasciato l’ultimo Container.
5. L’applicazione esce.

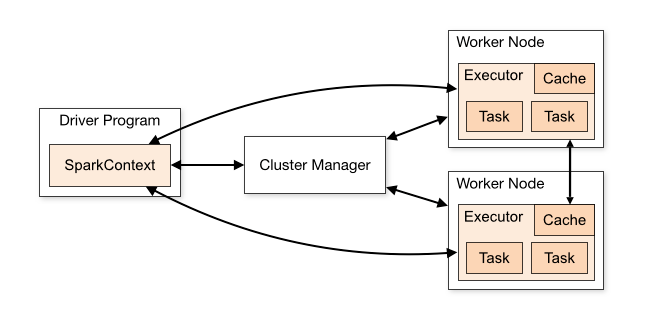
## 2.6 Apache Spark



Apache Spark è un framework general-purpose sviluppato originariamente presso l’UC Berkeley nel 2009 e successivamente donato all’Apache Software Foundation nel 2013. È una piattaforma open source progettata in modo tale da integrarsi con tutte i tool inerenti al mondo dei Big Data.

Apache Spark ha l’obiettivo di andare a processare i dati andando ad estendere ad un livello successivo l’approccio utilizzato da Hadoop MapReduce; difatti è importante l’innovazione che viene introdotta per quanto riguarda la velocità di elaborazione. In MapReduce le operazioni di lettura e scrittura vengono effettuate su disco e ciò influisce sulle performance, Spark, d’altro canto, sfrutta i dischi soltanto quando è necessario e memorizza i dati nella memoria principale che è più veloce e con gli anni si è rilevata anche più economica e grazie a ciò è più veloce fino a 100 volte del metodo utilizzato in MapReduce.

Le applicazioni Spark vengono eseguite come processi indipendenti su un cluster, queste sono coordinate dall’oggetto SparkContext presente nel programma principale, chiamato DriverProgram. Lo SparkContext, per poter interagire con l’intero cluster, necessita di collegarsi al Cluster Manager (Mesos o YARN), che ha il compito di allocare le risorse tra le applicazioni. Una volta connesso, Spark avvia gli Executor all’interno dei nodi del cluster, i Worker Node, questi sono processi responsabili dello svolgimento delle operazioni come l’archiviazione dei dati per l’applicazione o il calcolo sui dati stessi. Di seguito gli Executor, riceveranno il codice del programma ed i task che dovranno eseguire.



Apache Spark è composta da cinque diverse componenti fondamentali:

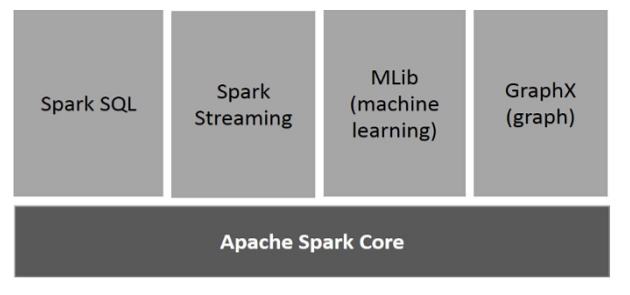
**Spark Core:** è il cuore del framework, fornisce il motore di esecuzione per la piattaforma che è utilizzata dalle altre componenti costruita al livello superiore. Spark Core è responsabile dell’esecuzione di tutte le operazioni di I/O di base e dello scheduling, inoltre, poiché è l’elemento che effettua il monitoraggio, si occupa del ripristino degli errori e della gestione della memoria affinché questa sia ottimale. In Spark Core viene utilizzata una struttura dati chiamata RDD (Resilient Distributed Dataset). Questo tipo di struttura va a superare il problema che era presente in MapReduce riguardante l’archiviazione dei passaggi intermedi e del recupero dei dati; poiché queste strutture dati sono direttamente in memoria e possono essere condivise tra diverse attività all’interno dello stesso processo Spark, si velocizza di gran lunga l’accesso ai dati non eseguendo le operazioni di serializzazione e deserializzazione.

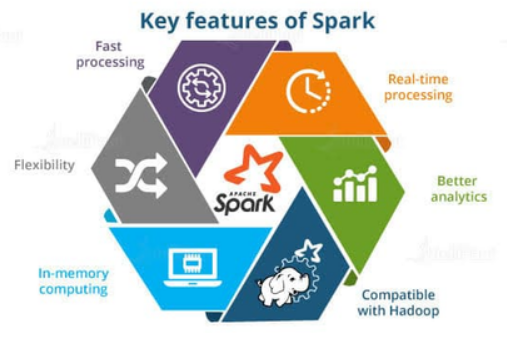
**Spark SQL:** permette di eseguire query sui dati in tramite SQL oltre che alla sua variante HQL (Hive Query Language). Supporta la lettura da sorgenti di dati diversificate tra cui JSON e tabelle Hive. Spark SQL permette di integrare le query SQL con le operazioni di manipolazione dei dati supportate da RDD nei vari linguaggi di programmazione come Python, Scala e Java andando a combinare SQL con analisi complesse. Molto rilevante è la presenza del Catalyst optimizer, uno strumento in grado di andare a migliorare le prestazioni e la produttività delle query.

**Spark Streaming:** è la componente utilizzata per analizzare flussi di dati in tempo reale. Spark Streaming lavora in diverse fasi; la prima riguarda la raccolta del flusso dati che può essere effettuate da sorgenti base, sorgenti disponibili nello Streaming Context, oppure da sorgenti avanzate, come Kafka. La seconda fase riguarda il processing dei dati, i dati raccolti vengono analizzati secondo alcuni algoritmi come il map-reduce. L’ultima fase è l’archiviazione dei dati, questi vengono memorizzati nel file system, nei database piuttosto che nelle dashboard. Lo Spark Streaming è uno strumento affidabile poiché riesce a gestire tutti le tipologie di errori, come il guasto del disco, grazie ai cosiddetti checkpoint. Essi hanno il compito di memorizzare periodicamente lo stato del disco e grazie ad essi si potrebbero raccogliere i dati dal punto in cui si è verificato l’errore.

**MLlib:** è una libreria di machine learning scalabile. MLlib fornisce vari algoritmi di machine learning tra cui classificazione, regressione, clustering oltre a supportare l’importazione dei dati e la valutazione del modello. Tutte le funzioni sono state progettate per distribuire il calcolo su ciascun nodo presente nel cluster.

**GraphX:** è una libreria che viene utilizzata per la gestione dei grafi. GraphX consente di creare un grafo diretto con proprietà arbitrarie e mette a disposizione algoritmi per gestire i grafi stessi.





### 2.6.1 Caratteristiche Apache Spark

In figura si possono notare le principali caratteristiche di Spark, che hanno portato all’attuale espansione del framework per quanto riguarda l’ambito dei Big Data. Tra queste troviamo:

**Fast processing:** la principale caratteristica che ha portato alla scelta del framework per l’analisi dei dati è la propria velocità. I Big Data sono caratterizzati da proprietà viste in precedenza, come il volume e la varietà, e si è in continua ricerca di una elaborazione a velocità sempre maggiore. Apache Spark contiene Spark RDD (Resilient Distributed Dataset), verranno esaminati successivamente, che consentono di risparmiare tempo sia nelle operazioni di lettura che di scrittura andando ad aumentare di quasi 10-100 volte l’analisi fornita da Hadoop.

**Flessibilità**: è un aspetto molto importante poiché permette agli sviluppatori di scrivere le applicazioni in linguaggi come Java, Scala, R o Python fornendo ad essi oltre 80 operatori di alto livello.

**Real-time processing:** con Spark è possibile fornire un’elaborazione dati in tempo reale in grado di produrre risultati immediati, ormai altamente richiesta nell’ambito dei Big Data. Si pensi ad esempio come questo non fosse possibile in MapReduce dove era ammessa soltanto l’analisi dei dati precedentemente immagazzinati.

**Better analytics:** per quanto riguarda le operazioni effettuabili con Apache Spark si allarga il campo d’azione sui dati; Spark contiene un set ricco di query SQL, algoritmi di Machine Learning che portano ad un’analisi dei dati più ampia non limitandosi dunque alle funzioni di Map e Reduce.

**In-memory computing:** viene fornito un modo per effettuare l’elaborazione in memoria. I dati non vengono archiviati in un disco, ma vengono recuperati direttamente dalla RAM, ciò va ad aumentare la velocità di elaborazione. È inoltre fondamentale la presenza dei DAG (Directed Acyclic Graph); questi vanno a supportare il calcolo in memoria poiché fornisce un flusso di dati aciclico che aumenta la velocità ogni qualvolta è necessario andare a recuperare dati in memoria.

**Compatibilità con Hadoop.**

### 2.6.2 RDD

RDD è la struttura dati fondamentale di Apache Spark, è una collezione immutabile di oggetti di diverso tipo. Ogni set di dati in Spark RDD è partizionato logicamente su diversi server in modo tale da poter effettuare l’elaborazione sui diversi nodi del cluster. Andiamo ad analizzare Spark RDD in termini di operazioni concesse e dalla possibile creazione della struttura dati.

Iniziamo l’analisi partendo dal termine RDD che sta per:

**Resilient:** tollerante ai guasti ed è capace di ricostruire partizioni mancanti o danneggiate in cado di guasti.

**Distributed:** i dati sono distribuiti su più nodi del cluster.

**Dataset:** è il record dei dati con i quali si lavora. I set di dati può essere caricato esternamente, questo può essere un file JSON, un file CSV, un file di testo o persino un database.

Esistono tre principali metodi attraverso i quali è possibile creare gli RDD.

**Collezioni parallelizzate:** è possibile effettuare la creazione, andando a considerare delle collezioni esistenti. Sfruttando i linguaggi di programmazione, come Scala, Java o Python, è possibile andare a eseguire il metodo parallelize appartente allo sparkcontext. Questo è il metodo base attraverso il quale viene create in maniera veloce un RDD. Per eseguire questo metodo l’intero dataset deve risiedere all’interno di una sola macchina; proprio per questo è utilizzato raramente al di fuori dell’area di test.

**Dataset esterni:** se ogni sorgente di dati supporta Hadoop, incluso il file system locale, è possibile andare a creare da esso un RDD. Apache Spark supporta numerose sorgenti come file di testo, file csv e altre tipologie di input proprie di Hadoop. È possibile dunque andare a creare un RDD attraverso l’invocazione del metodo relativo allo sparkcontext; questo metodo prende in input l’URL del file e va ad analizzare l’intera collezione della destinazione.

**RDD esistenti:** poiché gli RDD sono collezioni immutabili, non è possibile effettuare cambiamenti, infatti è possibile andare a creare un RDD differente da uno già esistente. Questo processo sostanzialmente effettua una trasformazione, come vedremo successivamente. Come risultato avremo un nuovo RDD e data l’immutabilità viene garantita la consistenza su tutto il cluster.

Una volta creato l’RDD è necessario effettuare operazioni su di esso affinché possa essere effettuata l’analisi, tra le operazioni possibili troviamo:

**Trasformazioni**: a partire da un RDD se ne crea uno nuovo che dunque dipenderà, in qualche modo, da quello di partenza. In questo modo non viene cambiata la struttura dell’RDD in input ma vengono prodotti uno o più RDD. Esempi di trasformazioni sono: map(), flatMap(), filter(), select() ecc...

**Azioni:** sono delle operazioni che restituiscono come un risultato dato da dei calcoli effettuati su un RDD. Sono delle operazioni che producono valori e non RDD. Tra le varie azioni troviamo count() che restituisce il numero di elementi presenti o il metodo collect() che non fa altro che restituire l’intero RDD in memoria.

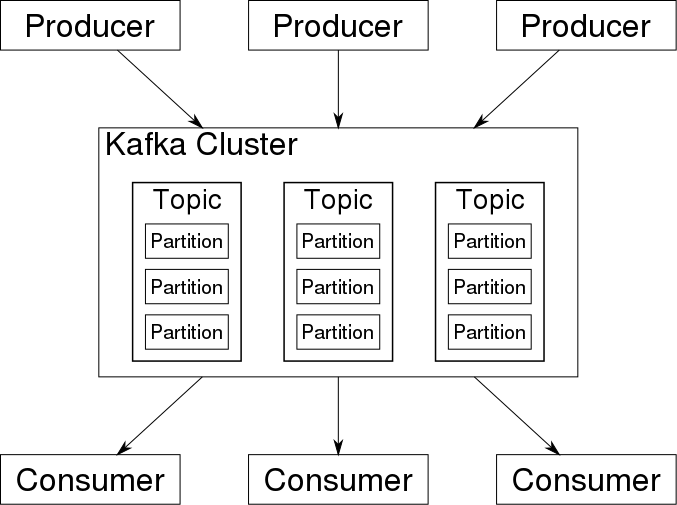
## 2.7 Apache Kafka



Apache Kafka è un sistema open source di messaggistica publisher-subscriber istantanea che consente la gestione di un elevato numero di operazioni in tempo reale da migliaia di client, sia in lettura che in scrittura. Il progetto, inizialmente scritto in Scala, passato poi a Java, è sviluppato dalla Apache Software Foundation e mira a fornire un sistema di gestione dei feed di dati altamente scalabile con caratteristiche quali performance elevate e bassa latenza. Il sistema si basa sul servizio di Apache Zookeeper descritto successivamente.

Kafka comprende alcune componenti, ognuna con uno scopo ben preciso:

* **Kafka Broker**: elementi compongono il cluster di gestione di messaggi, persistenza dei dati e di salvataggio delle informazioni di Apache Kafka, ogni broker possiede un ID univoco;
* **Kafka Java Client**: librerie Java per la scrittura e lettura di messaggi su Kafka;
* **Kafka Stream**: libreria che permette di utilizzare Apache Kafka come un sistema di elaborazione di dati;
* **Kafka Connect**: un framework che permette di connettere Kafka a sistemi esterni come database, file system



Kafka è composta da alcune entità specifiche come:

**Record**: è una coppia chiave-valore e rappresenta l’informazione in Kafka, la chiave è un identificativo univoco, il valore è il messaggio cioè l’informazione stessa. Ad ogni record viene associato un’informazione aggiuntiva come il timestamp, fondamentale per le operazioni di partizionamento e associazione delle informazioni.

**Topic:** il topic è il luogo dove viene realizzata l’architettura publish-subscribe, difatti avviene lo scambio di messaggi(record) tra chi pubblica(producer) e chi si iscrive(consumer) al topic. Sono divisi in un numero di partizioni che contengono record in una sequenza immutabile. Ogni record in una partizione è assegnato e identificato univocamente dal suo offset. Un Topic può avere partizioni multiple permettendo così a più consumers di leggere da un topic parallelamente.

**Log e partition**: per ogni topic, il cluster Kafka mantiene un log partizionato dove ogni partizione è rappresentabile come sequenza ordinata e immutabile di record.

**Producer:** sono processi, locali o remoti, che pubblicano i messaggi sui topic del cluster Kafka.

**Consumer:** sono processi che vanno ad elaborare i record che vengono pubblicati sui topic nei quali sono iscritti.

**Message Broker:** è un software che funge da intermediare e permette di “tradurre” il messaggio dal formato del mittente al formato che viene accettato dal ricevente e viceversa, così da uniformare la comunicazione tra sorgente e destinazione.

**Partizioni di replica:** sono semplicemente delle partizioni di backup. Nelle repliche non vengono mai effettuate operazioni di lettura o scrittura. Vengono utilizzate per prevenire la perdita dei dati.

Kafka è formato da 5 APIs per Java e Scala:

**Admin API**: per gestire ed analizzare topic, broker e altri oggetti kafka

**Producer API**: permette ad un processo di pubblicare un record su uno o più topic.

**Consumer API**: permette ad un processo di iscriversi ad uno o più topic ed a processare ciò che viene prodotto.

**Kafka Streams API**: permette di implementare applicazioni e micro-servizi di streaming dati. È composto da funzioni di alto livello per processare i dati, includendo trasformazioni, aggregazioni e join. Viene letto l’input da uno o più topic per generare l’output su uno o più topic, trasformando di fatto il flusso di dati d’input in flusso di dati

**Kafka Connect API**: per creare ed eseguire connettori riutilizzabili di import/export dei dati che consumano(leggono) o pubblicano(scrivono) stream di eventi da e verso sistemi esterni ed applicazioni in modo che possano integrarsi con Kafka. Uno di questo utilizzato nel caso d’uso è il connettore relativo a Kudu.

Kafka viene ampiamente utilizzato poiché offre servizi come:

**Affidabilità**: poiché è distribuito, partizionato, possiede delle repliche ed è fault-tolerance.

**Scalabilità**: poiché il sistema di messaggistica si ridimensiona facilmente senza tempi di inattività.

**Durabilità**: Kafka utilizzata dei log che mantengono i messaggi persistenti su un disco.

**Performance**: poiché possiede un alto throughput per la pubblicazione sia per la sottoscrizione dei messaggi.

Tramite Cloudera Manager è stato possibile formare un cluster Kafka che è formato da 3 broker ed un server. Per coordinare i kafka broker è stato necessario utilizzare sulle macchine interessate il servizio di Zookeeper che viene principalmente utilizzato per notificare a Producer e Consumer la presenza nel sistema di qualsiasi altro nuovo broker o il fallimento di uno di essi.

## 2.8 Apache Zookeeper



Apache Zookeeper è un servizio che viene utilizzato dal un cluster, per coordinare gli host tra di loro e per mantenere i dati condivisi con robuste tecniche di sincronizzazione. Apache ZooKeeper è un’applicazione distribuita che fornisce servizi per scrivere applicazioni distribuite.

Apache Zookeeper fornisce servizi come:

**Servizio di naming**: serve per identificare i nodi che compongono il cluster attraverso il loro nome.

**Gestione del cluster**: permette sia ad un nodo di entrare o uscire dal cluster sia di monitorarne lo stato.

**Servizio di blocco e sincronizzazione**: viene eseguito il blocco dei dati quando su di essi avviene una modifica. In questo modo qualora si manifesti un errore si procederebbe con il ripristino automatico dello stesso.

**Registro dati altamente affidabile**: è possibile accedere ai dati anche quando uno o più nodi non sono disponibili.

Inoltre garantisce:

**Sincronizzazione**: vi sono meccanismo di mutua esclusione e cooperazione tra i processi del server.

**Messaggi ordinati**: permette di tenere traccia dei messaggi assegnando un numero che ne indica l’ordine.

**Serializzazione**: aiuta nel garantire la coerenza dell’applicazione in esecuzione.

**Affidabilità**: quando uno più nodi del cluster smettono di funzionare, il sistema continua a funzionare.

**Atomicità**: il trasferimento dei dati o avviene con successo o fallisce completamente, non vi è una via di mezzo.

Andando ad analizzare l’architettura di Apache ZooKeeper, che sarà di fondamentale importanza per comprende come avviene la configurazione del cluster, dobbiamo sicuramente menzionare le seguenti componenti:

**Client**: è uno dei nodi che compongono il cluster, accede alle informazioni attraverso il server. Ogni client manda un messaggio al server per notificargli che è attivo, un messaggio Heartbeat.

**Server**: è uno dei nodi nell’ensemble di ZooKeeper che fornisce servizi ai e dà conoscenza circa il proprio stato ai client.

**Ensemble**: insieme dei server di Zookeeper.

**Leader**: è il nodo che esegue il ripristino automatico in caso di guasto di uno dei nodi collegati.

**Follower**: nodo che segue le istruzioni del leader.

## 2.9 Apache Kudu



Kudu è un servizio di storage sviluppato per la piattaforma Apache Hadoop. Viene utilizzato poiché condivide proprietà con le applicazioni facenti parte dell’ecosistema Hadoop, è scalabile e garantisce la continuità di servizio.

Apache Kudu comprende numerosi vantaggi come:

* Un modello di consistenza forte ma flessibile, che consente di scegliere i requisiti di consistenza in base alle richieste.
* Prestazioni elevate per l’esecuzione simultanea di carichi di lavoro sequenziali.
* Facile da amministrare e gestire.
* Alta disponibilità. I Tablet Server e i Master utilizzano l’algoritmo Raft, che garantisce che finché è disponibile più della metà del numero totale di repliche il Tablet è disponibile per la lettura e la scrittura. La scrittura può essere eseguita dai Tablet follower che hanno soltanto il compito della lettura, anche in caso di un guasto al Tablet leader.
* Modello dati dotato di struttura.

Per poter andare ad analizzare l’architettura riguardante Apache Kudu e poter comprendere al meglio l’importanza effettiva del suo utilizzo nel lavoro di Tesi è necessario dover andare ad illustrare i termini fondamentali ed i relativi concetti:

**Database su colonne**: è un sistema di gestione del database che immagazzina le tabelle per colonna piuttosto che per riga. Sia il database su colonne che su righe possono utilizzare i linguaggi tradizionali come SQL per caricare i dati e eseguire le query; utilizzare lo storage per colonne rispetto a quello per righe può più precisamente accedere ai dati necessari per andare a soddisfare un query piuttosto che andare ad analizzare e scartare dati non necessari presenti nelle righe. Il confronto tra i database su colonne e su righe è rilevante nell’efficienza dell’accesso per un dato carico di lavoro, difatti il seek time, cioè il tempo necessario per leggere o scrivere dati, nel database su colonne, se supportato da più dischi, è molto vantaggioso rispetto a quello nel database su righe. Per quanto riguarda l’ambito del Data-Waterhouse l’efficienza della lettura è di vitale importanza poiché porta ad un incremento delle prestazioni notevole, difatti è possibile leggere una colonna singola, o una sua porzione ignorando le altre poco rilevanti allo scopo della query in questione. Ciò porta a poter soddisfare la query leggendo il numero minimo di blocchi necessari memorizzati nel disco, di contro nel database su righe sarebbe necessario leggere l’intera riga andando ad esplorare dati non necessari.

**Compressione dati**: la compressione dei dati avviene in maniera più efficiente poiché nel database a colonne la colonna contiene un solo tipo di dati; la compressione basata su un solo pattern può essere di ordini di grandezza più efficiente rispetto alla compressione di tipi di dati misti che caratterizzano il database su righe. Combinata con l’efficienza della lettura dei dati dalle colonne la compressione permette di soddisfare le query durante lettura dal disco di un numero di blocchi ancora inferiore.

**Tabelle**: è il luogo dove vengono immagazzinati i dati in Kudu. Una tabella possiede uno schema ed un primary key ordinata. Una tabella è divisa in segmenti chiamati tablet.

**Tablet**: Una tablet è un segmento continuo di tabelle. Un dato tablet è replicato su più tablet server, ed ognuna di queste repliche, ad un dato istante, è considerate un tablet leader. Ogni replica può leggere, e per scrivere necessita l’accesso al set di tablet server su cui si appoggia il tablet.

**Tablet Server**: un tablet server immagazzina e fornisce i tablet all’utente. Per un dato tablet, un tablet server funge come leader e gli altri, i follower, fungono da repliche di quel determinato tablet. Soltanto i leader possono eseguire richieste mentre sia leader che followers possono leggere le richieste. I leader vengono eletti utilizzando l’algoritmo “Raft Consensus”. Un tablet server può possedere più tablet.

**Master**: il master tiene traccia dei metadati correlati al cluster ed inoltre monitora i tablet, i tablet server, la Catalog Table.

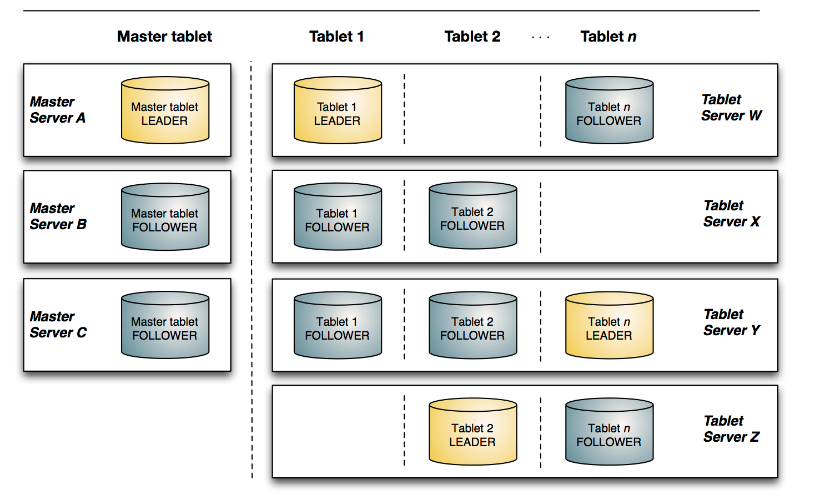
**Algoritmo “Raft Consensus”**: Kudu utilizza l’algoritmo del Raft Consensus come mezzo per garantire la consistenza e la fault-tolerance sia per i tablet che per i dati. Attraverso questo algoritmo numerose repliche di un tablet eleggono un leader, il quale è responsabile della ricezione e della replica delle scritture sulle repliche dei follower.

**Catalog Table**: è il luogo dove vengono memorizzati i metadati di Kudu. Vengono memorizzate informazioni riguardo le tabelle ed i tablets. Le Catalog Table non sono accessibili direttamente ma sono invece accessibili attraverso operazioni di metadati.

**Tablets**: lista dei tablets esistenti (?)

**Repliche Logiche**: Kudu effettua le repliche delle operazioni, non dei dati su disco. Difatti si parla di replica logica e non di replica fisica, ciò ha numerosi vantaggi:

* Le operazioni di inserimento e di aggiornamento trasmettono i dati sulla rete, mentre per le eliminazioni non è necessario che venga trasmesso nessun dato; difatti queste vengono inviate a ciascun tablet server che esegue l’eliminazione in locale.
* Le operazioni fisiche, come la compattazione, non hanno bisogno di trasmettere i sulla rete.
* Non è necessario che i tablet eseguano le compattazioni contemporaneamente o che restino sincronizzati sul livello dell’archiviazione fisica. Ciò riduce la possibilità che tutti i tablet server subiscano un’elevata latenza, a causa di carichi di scrittura pesanti o compattazioni.



## 2.10 Apache Impala



Apache Impala fornisce un metodo per eseguire query SQL a bassa latenza ed elevate performance su dati immagazzinati in host su cui viene eseguito Apache Hadoop. Impala viene integrato con Apache Hive per consentire la condivisione di database ed annesse tabelle tra entrambi i componenti. Alcuni dei vantaggi che introduce Apache impala in cluster sono:

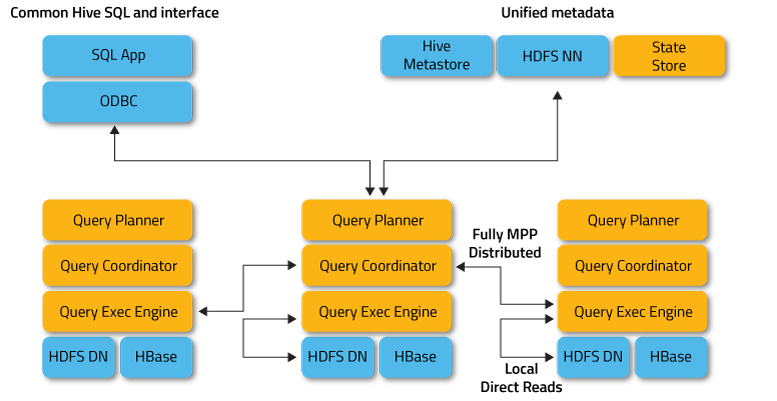
* L’integrazione con l’ecosistema CDH presente, permettendo così l’archiviazione, la condivisione e l’accesso utilizzando le risorse presenti all’interno di CDH.
* Permette l’accesso ai dati archiviati direttamente dal file system HDFS e fornisce un metodo per accedere ai dati archiviati in un database.
* Restituisce i risultati in minor tempo di quanto sia necessario rispetto ad un query eseguita con Hive.

Il server Impala è un motore di ricerca MPP (Massively Parallel Processing) distribuito. È formato da diversi processi deamon che vengono eseguiti sugli host all’interno del cluster CDH; difatti Impala si basa su tre demoni:

**Demone Impala (impalad):** il componente principale di Impala rappresentato fisicamente dal processo impalad. Ad esso sono demandate alcune funzioni come: lettura e scrittura su file di dati, accetta query trasmesse dal comando impala-shell, parallelizza le query e distribuisce il lavoro nel cluster. Una maniera per andare a distribuire i demoni impala è andare a localizzare insieme sia HDFS che Impala; ciò vuol dire che sarà presente un demone impala per ogni host in cui è presente un DataNode. I demoni impala sono in costante comunicazione con Statestore per verificare periodicamente quali deamon sono sani e quali no, così da poter accettare nuovi compiti. Ricevono anche dei messaggi broadcast dal demone catalog, ogni qualvolta un demone impala crea, altera o rilascia qualsiasi tipo di oggetto, oppure quando viene elaborata un’istruzione di tipo *INSERT* o *LOAD DATA* tramite impala. Ciò riesce a coordinare i metadati andando ad eliminare la necessità di istruzioni aggiuntive come la *REFRESH*.

**Demone Statestore (statestored):** controlla l’integrità di tutti i deamon impala all’interno del cluster e ha il compito di trasmettere i risultati a ciascun demone. È necessaria la presenza di un solo demone statestored su un host nel cluster. Se un demone impala dovesse cessare di funzionare, lo Statestore informa tutti gli altri demoni, affinché le richieste future non vengano affidate al demone ormai irraggiungibile. La presenza del demone Statestore non è fondamentale per il normale funzionamento; difatti se questi non è in funzione o è irraggiungibile gli impala daemon continuano a funzionare e distribuire il lavoro, ciò però va ad influire negativamente sulla robustezza del cluster, portando potenzialmente a metadati poco coerenti.

**Demone Catalog (catalogd):** è il demone che si occupa di inoltrare le modifiche apportate ai metadati, ottenute attraverso gli SQL statement, a tutti i demoni impala presenti nel cluster. Questo demone è presente in un solo host nel cluster e poiché le richieste vengono inoltrare al demone Statestore, molto spesso lo statestored e catalogd vengono eseguiti sullo stesso host.



## 2.11 Apache Hive



Apache Hive è un sistema di data warehouse open source per Hadoop, viene utilizzato per eseguire query in un linguaggio simile a SQL, ed analizzare set di dati di grandi dimensioni. Con l’introduzione di Apache Hive è stato possibile superare le nuove sfide che si erano presentate nell’ambito dell’analisi dei dati, come ad esempio la gestione della continua crescente dimensione dei dati generati. FUNZIONALITA’?

Da un punto di visto strutturale i principali componenti di Hive sono:

**Metastore:** memorizza i metadati per le partizioni e per ciascuna tabella, come lo schema e la loro posizione. Immagazzina i dati in formato RDBMS. I metadati conservati sono di fondamentale importanza poiché aiutano il Driver a tener traccia dei dati.

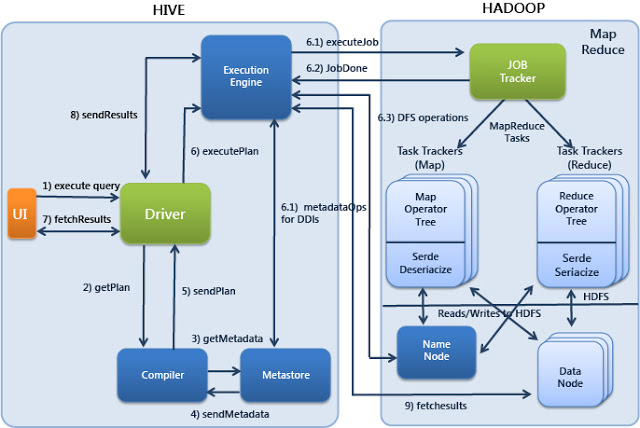
**Driver:** agisce come un controller che riceve le query dagli utenti. Avvia l’esecuzione della query andando a creare una sessione. Ha il compito di monitorare l’avanzamento dell’esecuzione, oltre che andare a memorizzare i metadati necessari generati durante una query.

**Compilatore:** esegue la compilazione delle query Hive. Genera un piano di esecuzione che contiene le attività ed i passaggi che devono essere eseguiti da MapReduce per ottenere l’output della query. Il compilatore inizialmente converte la query in un AST (Abstract Syntax Free) e, dopo aver verificato la compatibilità e la correttezza, converte quest’ultimo in un DAG (Directed Acyclic Graph)

**Ottimizzatore:** applica varie trasformazioni al piano di esecuzione per ottimizzare il DAG. Per fornire migliori prestazioni, può aggregare più trasformazioni, come ad esempio più join in un singolo, oppure può applicare una trasformazione ai dati prima di effettuare una riduzione, dividendo così le attività.

**Esecutore:** una volta che sono complete le fasi di compilazione e di ottimizzazione, vengono eseguiti i task. L’esecutore assegna e coordina le operazioni ai singoli componenti di Hadoop.

**Interfaccia Utente:** per interagire con Hive viene messa a disposizione la CLI (command-line interface), attraverso cui l’utente può eseguire query ed ottenere risultati. Inoltre, per agevolare l’interazione agli utenti, è presente il Thrift Server; questo componente permette di interagire con Hive per mezzo di una connessione di rete.



La figura mostra lo schema architetturale di Hive ed il processo attraverso il quale interagisce con il framework Hadoop. Tramite l’interfaccia utente un client manda una query, questa viene inviata al Driver affinché possa essere eseguita. Il Driver interagisce con il Compiler, una volta accolta la richiesta, si occupa di contattare il Metastore affinché possa ottenere i metadati; la query viene esaminata per controllare la correttezza sintattica e genera il piano di esecuzione che manderà al Driver. Una volta ricevuto il piano di esecuzione, il Driver manda il piano di esecuzione all’esecutore affinché questo possa essere eseguito. L’esecutore invia il task al JobTracker, collocato nel NameNode, e lo assegna al TaskTracker, posizionato nel DataNode; qui viene eseguita la query Map-Reduce. Infine l’esecutore legge il contenuto dei file temporanei direttamente da HDFS, i risultati saranno mandati all’utente tramite il Driver.

## 2.12 Oozie

## 2.13 Hue

# CAPITOLO 3: Approccio proposto

Attraverso la piattaforma cloud Microsoft Azure è stato possibile costruire un cluster che consiste in cinque macchine virtuali necessarie per gestire il lavoro di analisi di dati. È stata scelta la piattaforma Microsoft Azure poiché offre servizi come:

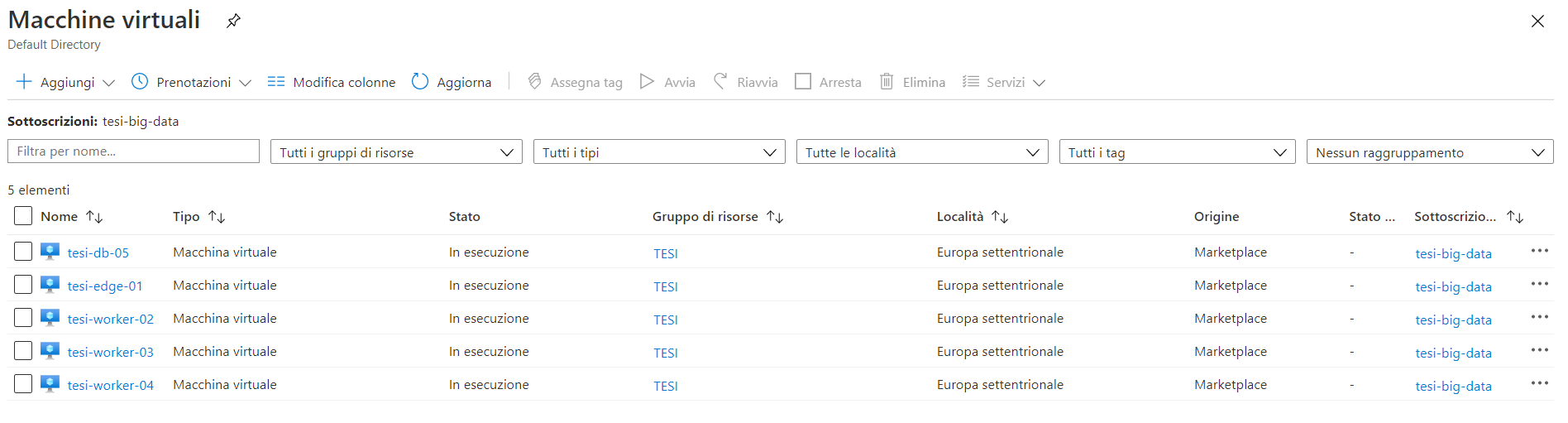
**Privacy**: viene assicurato che soltanto l’utente autorizzato può controllare e distribuire i dati. Viene comunicato come vengono gestiti e dove vengono gestiti i dati.

**Disponibilità**: vi sono numerosi data center che permettono di fornire ridondanza e disponibilità elevate, difatti offre circa il 99% di uptime, che equivale a meno di 5 ore l’anno di inattività.

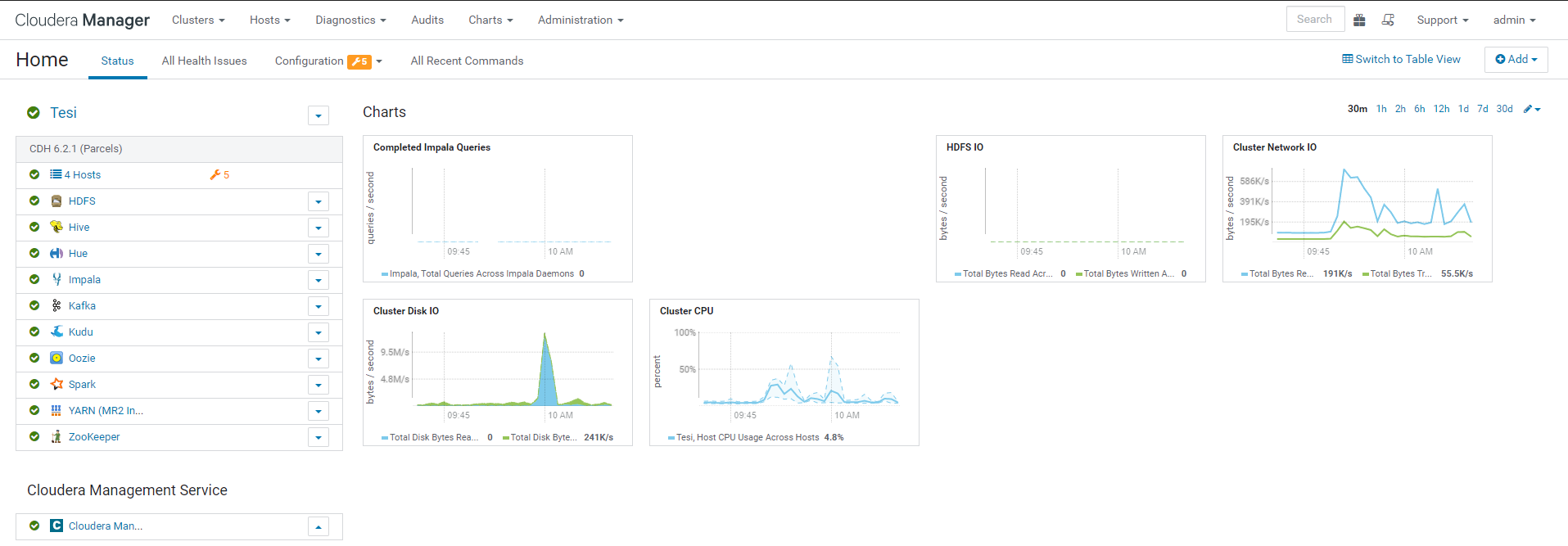
**Scalabilità**: si possono ridimensionare le risorse di calcolo così permettendo di soddisfare le esigenze volatili dell’utente.

**Efficacia dei Costi**: viene offerto un piano di pagamento in base al consumo, il che consente all’utente di avere un controllo migliore sul proprio budget IT. Dunque viene acquistato soltanto ciò che è necessario e vengono pagate le risorse che effettivamente vengono utilizzate.

**Disaster Recovery**: l’infrastruttura cloud decentralizzata facilita la creazione di un piano di disaster recovery. Viene garantito che, i dati e le applicazioni critiche, possano essere gestite da un sito di backup durante il ripristino, non andando così incontro a periodi di inattività dovuti al problema riscontrato.



Il cluster, come si può vedere in figura, è formato da cinque macchine virtuali e sono della dimensione B2ms, con 2 CPU virtuali, 8 GiB di RAM e 4 dischi e con sistema operativo CentOS Linux 7(Core). Queste macchine sono state create con ruoli ben precisi; **Tesi-edge-01** svolgerà il ruolo di edge ed è la macchina in cui, oltre gli altri servizi di cui parleremo in seguito, viene eseguito il Cloudera Manager; le tre worker (Tesi-worker-02, Tesi-worker-03, Tesi-worker-04) sono le macchine nelle quali sono stati installati successivamente i servizi necessari per l’analisi dei dati; **Tesi-db-05,** è la macchina dedicata allo storage dei dati, quindi svolgerà il ruolo di DB e nella quale non verranno utilizzati servizi di analisi dei dati, dunque non farà parte del cluster gestito tramite Cloudera Manager. Le macchine sono state gestite tramite Putty, un client SSH che permette di emulare la gestione in remoto di un sistema informatico. Per iniziare la configurazione è stato necessario rendere univo la localizzazione dell’host sulla rete



# CAPITOLO 4: ANALISI DEI DATI E REALIZZAZIONE HEATMAP