

Uno delle caratteristiche chiave per operare con i **Big Data** è la memorizzazione delle informazioni. Una soluzione tradizionale può risultare un vero e proprio collo di bottiglia.

- ▶ Tra le tecnologie sviluppate per questo scopo vi è sicuramente **Apache Hadoop**, framework open source per applicazioni che elaborano grandi quantità di dati in parallelo, su cluster di grandi dimensioni.
- ▶ Se non si ha accesso ad una rete di nodi che formano un cluster, è possibile utilizzare la piattaforma di container software indipendenti **Docker**.
- ▶ L'implementazione del software **AnagramFinder** consente di individuare quelli che sono i vantaggi che è possibile ottenere adottando dall'utilizzo combinato di tool di sviluppo, opportunamente trattati nel lavoro in esame.



- ▶ **Docker** è una piattaforma open-source per lo sviluppo e l'esecuzione di applicazioni. Essa consente di separare le applicazioni dall'infrastruttura sottostante, in modo da distribuire rapidamente il software prodotto.
- ▶ Offre la possibilità di eseguire un'applicazione in un ambiente isolato, definito **container**. L'isolamento e la sicurezza, offerti dalla piattaforma, consentono di lanciare contemporaneamente più containers su un host.
- ▶ I contenitori sono definiti **leggeri**, poiché non necessitano del carico aggiuntivo di un hypervisor, come accade per le macchine virtuali. Ciò si traduce nella possibilità di eseguire più containers su un determinato hardware rispetto all'utilizzo delle virtual machine.



Consente di lavorare in **ambienti standardizzati** che adottano container locali. In dettaglio risulta interessante esaminare gli effettivi benefici dell'utilizzo di Docker:

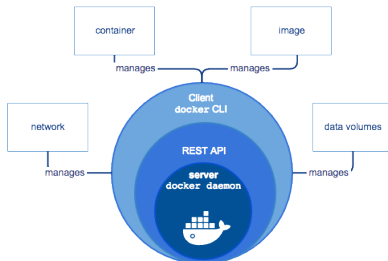
1. Elevata **portabilità del software**. I contenitori possono essere eseguiti sul laptop locale, su macchine fisiche o virtuali. Gli sviluppatori possono scrivere codice locale e condividere il loro lavoro tramite i contenitori Docker.
2. La sua natura leggera semplifica la **gestione dinamica dei carichi di lavoro**, attraverso il ridimensionamento di applicazioni o servizi realizzato quasi in tempo reale, secondo le esigenze.
3. Adatto per implementazioni **software di piccole/medie dimensioni**, in cui è necessario operare con minori risorse disponibili.



Docker Engine

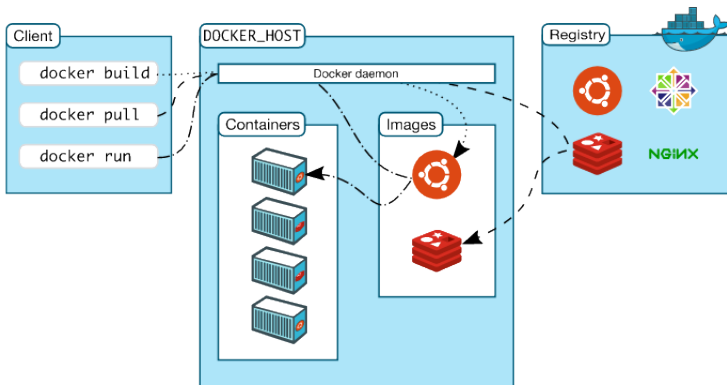
Docker Engine è un'applicazione di tipo client-server che presenta come componenti principali:

- ▶ Un **server**, il quale è un programma ad esecuzione prolungata chiamato processo demone e che corrisponde al comando *dockerd*.
- ▶ Un'**API REST**, che specifica le interfacce che i programmi possono utilizzare per interagire con il demone.
- ▶ Un **client**, basato su interfaccia a riga di comando che è identificato nel comando *docker*.



Architettura Docker

Docker adotta un'architettura di tipo **client-server**. Il client interagisce con il demone Docker, che a sua volta esegue il lavoro di realizzazione, esecuzione e distribuzione dei container.



Architettura Docker(II)

Analizzando l'architettura di Docker possiamo individuare tre componenti fondamentali che la caratterizzano, vale a dire:

- ▶ Il **demone Docker**, ovvero *dockerd*, ascolta le richieste dell'API Docker e gestisce oggetti come immagini, contenitori, reti e volumi. Un demone può inoltre comunicare con altri demoni per gestire i servizi offerti.
- ▶ Il **client Docker** che rappresenta lo strumento attraverso il quale gli utenti interagiscono con la piattaforma. Quando si inseriscono comandi, come ad esempio *docker run*, il client li invia a *dockerd* che li esegue.
- ▶ I **registri Docker**, che memorizzano le immagini Docker. *Docker Hub* è un registro pubblico e Docker, una volta installato, è configurato per cercare immagini su tale registro in modo predefinito.



Quando si lavora con Docker si creano ed utilizzano immagini, contenitori, reti, volumi, plug-in ed altri oggetti di tale piattaforma. In dettaglio possiamo individuare:

- ▶ Le **immagini**, che risultano template di tipo read-only con le istruzioni per creare i container Docker. Di solito un'immagine è basata su un'altra immagine, con alcune ulteriori personalizzazioni. Ad esempio è possibile realizzare un'immagine basata su Ubuntu, ma che a sua volta installa determinate applicazioni, oltre a presentare la configurazione dell'ambiente necessaria per far funzionare tali software.
- ▶ I **contenitori**, che rappresentano istanze eseguibili di un'immagine. Si può creare, avviare, fermare, spostare, o eliminare un container attraverso l'API Docker o l'interfaccia da riga di comando. Inoltre è possibile allargare lo spazio di archiviazione disponibile o creare una nuova immagine in base al suo stato corrente. Un contenitore è definito dalla sua immagine e dalle eventuali opzioni di configurazione fornite durante la creazione o l'avvio.



Analizzando nel dettaglio i container Docker possiamo affermare che:

- ▶ Un **contenitore Docker** può essere considerato come un'unità standard software che impacchetta il codice e tutte le sue dipendenze, in modo che l'applicazione venga eseguita in maniera rapida ed affidabile da un ambiente di elaborazione ad un altro.
- ▶ L'**immagine di un Docker container** è un pacchetto software leggero, autonomo ed eseguibile, il quale include tutto il necessario per eseguire una determinata applicazione: codice, eseguibile, strumenti di sistema, librerie di sistema ed impostazioni di configurazione.
- ▶ In particolare le **immagini diventano container** solamente quando vengono eseguite su Docker Engine.



Container Docker(II)

La tecnologia dei container Docker è disponibile sia per applicazioni basate su Linux che su Windows, il software nel container funzionerà sempre allo stesso modo, indipendentemente dall'infrastruttura. I container Docker che sono eseguiti su Docker Engine risultano:

- ▶ **Standard:** Docker infatti ha creato lo standard industriale per i containers, in modo che essi possano essere lanciati ovunque.
- ▶ **Leggeri:** i container non richiedono un sistema operativo per ogni applicazione, aumentando l'efficienza e riducendo i costi di server e licenze.
- ▶ **Sicuri:** a riguardo Docker offre le più potenti funzionalità di isolamento presenti nel settore.



Container vs Virtual Machine

Container e Virtual Machine hanno simili vantaggi di isolamento e allocazione delle risorse, ma operano in modo differente:

- ▶ I **Container** rappresentano un'astrazione a livello di app che racchiude codice e dipendenze insieme. Più contenitori possono essere eseguiti sullo stesso computer, ciascuno dei quali in esecuzione come processo isolato nello spazio utente.
- ▶ Le **Macchine Virtuali(VM)** sono un'astrazione dell'hardware fisico. Attraverso un ulteriore strato nell'architettura di virtualizzazione, detto hypervisor, è consentito eseguire più macchine virtuali su un singolo computer.



Con il termine *Big Data* si indica dati di dimensioni molto elevate, i quali non possono essere processati con tecniche di elaborazione tradizionali. Possiamo individuare alcune delle fonti da cui essi provengono tra le seguenti aree:

- ▶ **Social network**
- ▶ **E-commerce**
- ▶ **Stazioni metereologiche**
- ▶ **Aziende di telecomunicazione**
- ▶ **Mercato azionario**
- ▶ **Motori di ricerca**



Big Data: Modello delle 5V

Il modello di crescita di tali dati viene inizialmente definito delle 3V, con cui si identifica:

- ▶ **Volume**
- ▶ **Varietà**
- ▶ **Velocità**

In seguito tale modello è stato ulteriormente esteso, introducendo altre due V che rappresentano:

- ▶ **Veridicità**
- ▶ **Valore**



Le tecnologie per i Big Data risultano importanti per fornire analisi più accurate, in modo da avere una maggiore efficienza operativa, una riduzione dei costi ed un successivo abbassamento dei rischi per l'azienda.

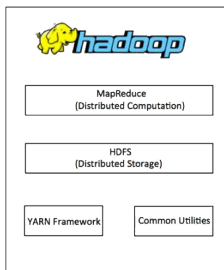
- ▶ Approccio **tradizionale**, in cui si presenta una singola macchina per la memorizzazione e l'elaborazione dei dati.
- ▶ Google ha provveduto a realizzare un algoritmo, definito **MapReduce**, che divide i task in parti più piccole, assegnandole a macchine appartenenti ad un cluster.
- ▶ Basandosi sull'algoritmo MapReduce, Doug Cutting ed il proprio team hanno realizzato un progetto open-source definito **Hadoop**.



Architettura Hadoop

Le componenti principali che caratterizzano l'architettura di Hadoop risultano essere:

- ▶ **HDFS**: fornisce un file system distribuito il quale è progettato per essere eseguito su hardware di largo consumo.
- ▶ **MapReduce**: modello di programmazione parallela per la scrittura di applicazioni distribuite.
- ▶ **YARN**: Yet another Resource Negotiator è un framework utilizzato per il job scheduling e la gestione delle risorse.
- ▶ **Hadoop Common**: librerie Java adottate da altri moduli Hadoop.



Vantaggi Hadoop

I vantaggi di adottare Hadoop possono essere ricondotti ai seguenti aspetti:

- ▶ **Velocità**, poiché in HDFS i dati sono distribuiti all'interno del cluster, e mappati in modo tale da facilitarne il recupero.
- ▶ **Scalabilità**, un cluster Hadoop può essere esteso aggiungendovi semplicemente nodi, senza interruzioni del framework stesso.
- ▶ **Convenienza**, essendo Hadoop un framework open-source che necessita di hardware a basso costo per archiviare ed elaborare i dati.
- ▶ **Compatibilità**, proprietà garantita per gran parte delle piattaforme, essendo Hadoop scritto in Java.
- ▶ **Parallelismo**, attraverso cui il framework opera per l'elaborazione delle informazioni.
- ▶ **Tolleranza ai guasti**, essendo HDFS in grado di replicare i dati all'interno del cluster.

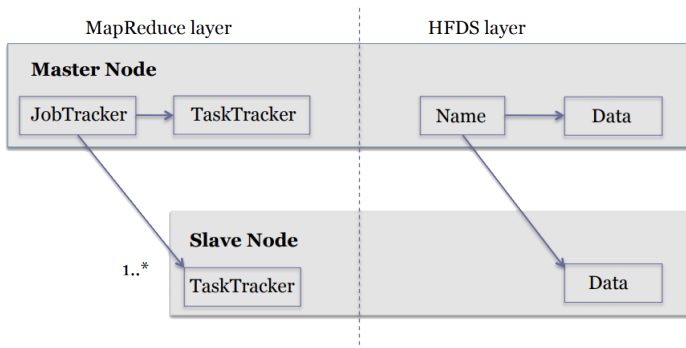


Cluster Hadoop

Analizzando l'architettura Hadoop possiamo affermare che è composta da due livelli principali:

- ▶ Livello di archiviazione(**HDFS**)
- ▶ Livello di elaborazione(**MapReduce**)

Ancor più nel dettaglio un cluster Hadoop è costituito da un singolo master e più nodi slave.



Hadoop File System

Hadoop viene fornito con un file system distribuito chiamato HDFS. Esso è altamente tollerante ai guasti e progettato per essere eseguito su hardware a basso costo. Risulta essere particolarmente adatto quando si opera con:

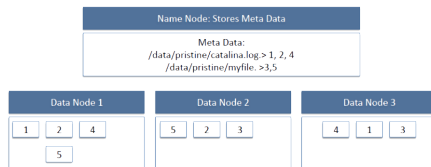
- ▶ **File di grandi dimensioni:** ovvero il size deve essere dell'ordine dei GB o maggiore.
- ▶ **Accesso ai dati in streaming:** vale a dire quando si dà maggiore importanza al tempo di lettura dell'intero dataset, rispetto alla latenza relativa alla lettura del primo elemento.
- ▶ **Hardware economico:** il file system è progettato per lavorare su componenti a basso costo.



Architettura HDFS

L'architettura di Hadoop File System si basa sul modello master/slave, ed è caratterizzata dai seguenti elementi:

- ▶ Il **NameNode**, svolge attività come la gestione del namespace del file system, l'apertura, la chiusura e la ridenominazione dei file e delle directory.
- ▶ I **DataNode**, gestiscono la memorizzazione dei dati all'interno del file system, eseguendo operazioni di lettura e scrittura, secondo le richieste del client.
- ▶ I **blocchi**, rappresentano la quantità minima di dati che può essere letta o scritta in HDFS.



Yet Another Resource Manager

YARN porta la programmazione, all'interno di un cluster Hadoop, ad un livello superiore a quello di Java, consentendo l'esecuzione di applicazioni basate su tecnologie più complesse, come HBase o Spark. Esso è caratterizzato dalle seguenti componenti:

- ▶ **Client:** per la sottomissione dei jobs di tipo MapReduce.
- ▶ **Resource Manager:** per gestire l'uso delle risorse all'interno del cluster.
- ▶ **Node Manager:** per la gestione dei singoli nodi di elaborazione nel cluster.
- ▶ **MapReduce Application Master:** per il controllo delle attività che eseguono il processo MapReduce.



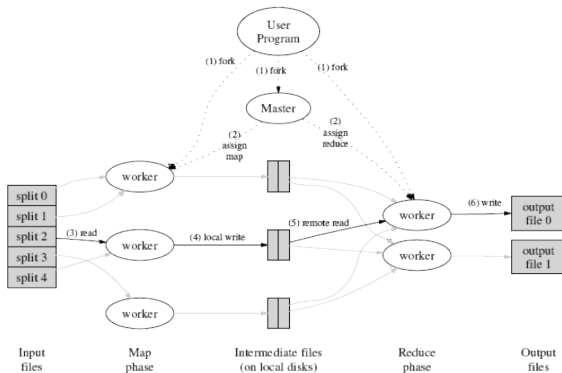
MapReduce è una tecnica di programmazione utilizzata al fine di elaborare i dati parallelamente ed in forma distribuita. Il paradigma MapReduce è composto da due fasi, vale a dire:

- ▶ La **fase Map**, in cui l'input è fornito sotto forma di coppie *chiave – valore* e convertito in un altro set di dati, in cui i singoli elementi sono suddivisi a loro volta in coppie *chiave – valore*.
- ▶ Il corrispettivo output è utilizzato dalla **fase Reduce** come input. L'output ottenuto dalla questa fase, anch'esso in forma *chiave – valore*, risulta essere il risultato finale prodotto dal modello.

	Input	Output
Map	<k1, v1>	list (<k2, v2>)
Reduce	<k2, list(v2)>	list (<k3, v3>)

MapReduce(II)

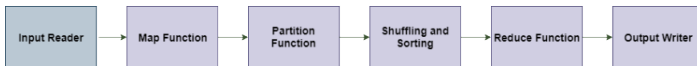
- ▶ Durante l'esecuzione di un processo di tipo MapReduce, Hadoop ha il compito di inviare i **task di Map e Reduce** ai nodi appropriati nel cluster.
- ▶ Gran parte dell'elaborazione avviene su quei nodi i cui dati da elaborare sono memorizzati su **dischi locali**, riducendo il traffico di rete.
- ▶ La capacità di ottenere un'**elevata scalabilità** dei processi su più macchine in un cluster è la peculiarità che spinge ad adottare il modello.



MapReduce: Flusso di dati

Per gestire grandi quantità di informazioni in modo parallelo e distribuito, i dati elaborati con il modello MapReduce devono attraversare varie fasi, vale a dire:

- ▶ L'**Input Reader** legge i dati di ingresso e li divide in blocchi della dimensione appropriata
- ▶ La **Map Function** elabora le coppie *chiave – valore* in ingresso e generano le corrispondenti coppie *chiave – valore* in uscita.
- ▶ La **Partition Function** assegna l'output di ciascuna funzione Map al Reducer appropriato.
- ▶ Lo **Shuffling** prevede che i dati vengano mescolati tra i nodi del cluster. Si applica poi il **Sorting**, in cui i dati sono ordinati in base alla chiave.
- ▶ La **Reduce Function** prende in ingresso i dati intermedi ottenuti dalla fase precedente e li riduce in una soluzione di dimensione inferiore.
- ▶ L'**Output Writer** ha il compito di trascrivere nella memoria persistente il risultato ottenuto dalla funzione Reduce.

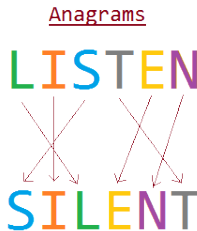


Caso di Studio: AnagramFinder

Un **anagramma** è una parola o frase ottenuta trasponendo le lettere di un'altra parola o frase. Esempi di anagrammi possono essere individuati in:

- ▶ La parola "mary" che può essere riorganizzata in "army".
- ▶ La parola "secure" che può essere riorganizzata in "rescue".
- ▶ La parola "death" che può essere riorganizzata in "hated".

Nel caso in esame, l'obiettivo del programma MapReduce realizzato è quello di trovare anagrammi a partire da un file in input. Esso elabora anagrammi che risultano singole parole, e sono ignorati quei termini minori di una certa lunghezza N.



Configurazione Docker

Hadoop funziona su cluster che vanno da un singolo nodo a migliaia di nodi. Se, come nel caso in esame, si vuole testare un'applicazione Hadoop oppure non si ha accesso ad una grande rete di cluster, è possibile configurare un cluster Hadoop in locale utilizzando **Docker**.

- ▶ Docker consente di creare ed eseguire applicazioni, con i loro ambienti di sviluppo, librerie e dipendenze integrati nei container.
- ▶ I container sono portable, è possibile impostare lo stesso sistema su un'altra macchina, eseguendo alcuni semplici comandi Docker.
- ▶ Risulta facile creare, condividere ed eseguire applicazioni senza dipendere dalla configurazione attuale del sistema operativo.

Se Docker non è già installato localmente, è possibile seguire le istruzioni sulla **homepage ufficiale** di Docker.

```
1 $ docker --version
2 $ docker-compose --version
```



Configurazione Hadoop

Per installare Hadoop in un container Docker, è necessaria un'immagine Docker di Hadoop.

- ▶ Per generare l'immagine viene adottato, nel caso di studio, il repository **Big Data Europe**.
- ▶ Dopo aver scaricato la cartella *docker-hadoop* in locale, bisogna modificare il file **docker-compose.yml**.
- ▶ Per il deploy del cluster Hadoop deve essere lanciato il seguente comando:

```
1 $ docker-compose up -d
```



Configurazione Hadoop(II)

Docker-Compose è un potente tool in Docker, adottato per configurare più container contemporaneamente. Con tale comando viene impostato un cluster Hadoop composto da:

- ▶ tre slave(datanodes)
- ▶ un master(namenode) con il quale gestire i datanodes
- ▶ uno YARN resourcemanager
- ▶ un historyserver
- ▶ un nodemanager

Al termine è possibile verificare la presenza di eventuali container attualmente in esecuzione attraverso il seguente comando:

```
1 $ docker ps
```

Il comando precedente restituisce l'output mostrato in Figura:

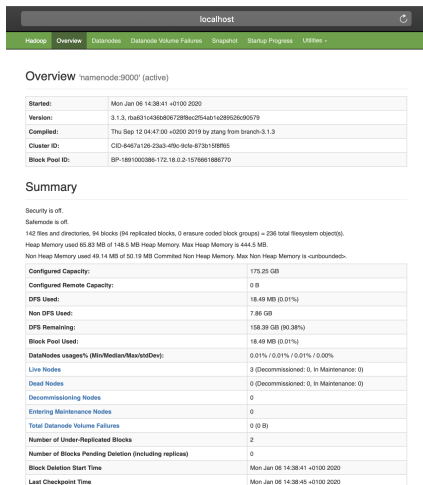
```
MPDIAntonio2@docker-hadoop-master antonioabate$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
86cb712a3653	bde2828/hadoop-historyserver:1.1.0-hadoop2.7.1-java8	"/entrypoint.sh /run-"	2 weeks ago	Up 5 seconds (health: starting)	8188/tcp	Historyserver
c78b0a0913a7	bde2828/hadoop-nodemanager:1.1.0-hadoop2.7.1-java8	"/entrypoint.sh /run-"	2 weeks ago	Up 5 seconds (health: starting)	8032/tcp	nodemanager1
4b7989548a26	bde2828/hadoop-resourcemanager:1.1.0-hadoop2.7.1-java8	"/entrypoint.sh /run-"	2 weeks ago	Up 5 seconds (health: starting)	0.0.0.0:8089->8088/tcp	resourcemanager
858a782b3f38	bde2828/hadoop-datanode:1.1.0-hadoop2.7.1-java8	"/entrypoint.sh /run-"	2 weeks ago	Up 7 seconds (health: starting)	9864/tcp	datanode3
92e83ca3c948	bde2828/hadoop-datanode:1.1.0-hadoop2.7.1-java8	"/entrypoint.sh /run-"	2 weeks ago	Up 7 seconds (health: starting)	9864/tcp	datanode1
169a2130a217	bde2828/hadoop-datanode:1.1.0-hadoop2.7.1-java8	"/entrypoint.sh /run-"	2 weeks ago	Up 7 seconds (health: starting)	9864/tcp	datanode2
e65d8a96c278	bde2828/hadoop-namenode:1.1.0-hadoop2.7.1-java8	"/entrypoint.sh /run-"	2 weeks ago	Up 8 seconds (health: starting)	0.0.0.0:9878->9878/tcp	namenode



Configurazione Hadoop(III)

Visitando poi la pagina <http://localhost:9870>, è possibile visualizzare lo stato corrente del sistema dal namenode.



localhost

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities -

Overview namenode-9000 (active)

Started:	Mon Jan 06 14:38:41 +0100 2020
Version:	3.1.3, rba631c4368b067298ec254ab1e299526c90579
Compiled:	Thu Sep 12 04:47:00 +0200 2019 by ztang from brandh-3.1.3
Cluster ID:	CID-6467a126-23a3-4f9c-9cfe-673b158f965
Block Pool ID:	BP-1691000386-172.18.0.2-1576661886770

Summary

Security is off.
SafeMode is off.
142 files and directories, 94 blocks (94 replicated blocks, 0 erasure coded block groups) = 236 total filesystem object(s).
Heap Memory used 65.83 MB of 148.5 MB Heap Memory. Max Heap Memory is 444.5 MB.
Non Heap Memory used 49.14 MB of 50.19 MB Committed Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	1.75.25 GB
Configured Remote Capacity:	0 B
DFS Used:	18.49 MB (0.01%)
Non DFS Used:	7.86 GB
DFS Remaining:	1.68.39 GB (90.38%)
Block Pool Used:	18.49 MB (0.01%)
DataNodes usage% (Min/Median/Max/stdDev):	0.01% / 0.01% / 0.01% / 0.00%
Live Nodes	3 (Decommissioned: 0, In Maintenance: 0)
Dead Nodes	0 (Decommissioned: 0, In Maintenance: 0)
Decommissioning Nodes	0
Entering Maintenance Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	2
Number of Blocks Pending Deletion (including replicas)	0
Block Deletion Start Time	Mon Jan 06 14:38:45 +0100 2020
Last Checkpoint Time	Mon Jan 06 14:38:45 +0100 2020



Affinché possa essere correttamente eseguito in Hadoop, il programma Anagram Finder realizzato segue il modello MapReduce. In dettaglio, le classi Java sviluppate per la realizzazione del software risultano essere:

- ▶ **AnagramDriver**, la classe principale che sottomette il job.
- ▶ **AnagramMapper**, la classe che implementa la map().
- ▶ **AnagramReducer** la classe che implementa la reduce().



Implementazione: AnagramDriver

AnagramDriver è la classe Java che sottomette il job ad Hadoop, al fine di individuare e contare gli anagrammi. Il metodo più importante della classe risulta il metodo `run()`, il quale riprende i seguenti passi definiti in pseudocodice:

Algorithm 1 AnagramDriver: `run()`

- 1: Creare un nuovo job con la configurazione data
 - 2: Impostare il tipo dell'input del job
 - 3: Impostare il tipo dell'output del job
 - 4: Comunicare al job di utilizzare AnagramMapper come classe Mapper
 - 5: Comunicare al job di utilizzare AnagramReducer come classe Reducer
 - 6: Impostare il path del file di input
 - 7: Impostare il path del file di output
 - 8: Attendere fino al completamento del job
 - 9: Restituire lo stato SUCCESS se tutto è andato correttamente
-



Implementazione: AnagramMapper

La classe **AnagramMapper** legge un record del file di input, dove per record vi si intende un insieme di parole corrispondente ad una linea del documento. Per ogni parola del record ne ordina le lettere e restituisce all'OutputCollector di Hadoop l'output nella forma:

- ▶ Chiave: lettere ordinate della parola
- ▶ Valore: parola stessa

Possiamo sintetizzare i passi seguiti da tale classe attraverso lo pseudocodice seguente:

Algorithm 2 AnagramMapper

- 1: Leggere un record in input(linea di testo del file)
 - 2: Per ogni linea di testo: estrarre le parole che la compongono
 - 3: Per ogni parola individuata: ordinare le lettere che la compongono
 - 4: Per ogni parola ordinata: impostare la coppia <chiave, valore>
 - 5: Scrivere la coppia <chiave,valore> nel contesto Hadoop
-



Implementazione: AnagramReduce

La classe **AnagramReducer** raggruppa i valori delle chiavi ordinate in input, ovvero i valori intermedi ottenuti dalla funzione map. Dopo il raggruppamento di tali valori verifica se, per ogni chiave, vi sono due o più valori. In questo caso si è individuato un anagramma. Possiamo sintetizzare il procedimento descritto nel seguente pseudocodice:

Algorithm 3 AnagramReducer

- 1: Per ogni parola ordinata(chiave): leggere i valori associati
 - 2: Per ogni valore associato ad una chiave: inserire il valore in un insieme
 - 3: Se l'insieme ha cardinalità maggiore di 1: si ha un anagramma
-



Per testare il cluster Hadoop realizzato e l'esecuzione del programma preso in esame, bisogna innanzitutto posizionarsi con una shell all'interno del namenode attivo, con il seguente comando lanciato da terminale locale:

```
1 $ docker exec -it namenode bash
```

In seguito devono essere create le directory che contengono i file da utilizzare, con il comando seguente lanciato dal terminale del namenode:

```
1 $ mkdir -p /home/mp/data-algorithms-book/
```

Successivamente, con il comando *docker container ls* dal terminale locale, viene salvato l'ID che identifica univocamente il container del namenode. Quest'ultimo è utilizzato per la copia dei file necessari dal pc locale al container.

```
1 docker cp src e65d8a96c2f0:/home/mp/data-algorithms-book/src
2 docker cp lib e65d8a96c2f0:/home/mp/data-algorithms-book/lib
3 docker cp anagram e65d8a96c2f0:anagram
```



Esecuzione(II)

In seguito vengono compilati i file Java tramite il comando *javac*, ed il file eseguibile ottenuto è posizionato nella directory */home/mp/data-algorithms-book/dist/*. Vengono poi settate tutte le variabili d'ambiente necessarie per la corretta esecuzione del programma:

```
1 #Cartella home dei file utilizzati
2 export BOOK_HOME=/home/mp/data-algorithms-book
3 #Path del file eseguibile
4 export APP_JAR=$BOOK_HOME/dist/data_algorithms_book.jar
5 #Path di Hadoop
6 export HADOOP_HOME=/opt/hadoop-3.1.3
7 #Path della directory input
8 export INPUT=anagram/input/anagram.txt
9 #Path della directory output
10 export OUTPUT=anagram/output
11 #Definizione classe driver
12 export
    DRIVER=org.dataalgorithms.chapB05.anagram.mapreduce.AnagramDriver
```



Esecuzione(III)

Infine, i file che Hadoop utilizza per l'esecuzione di AnagramFinder sono copiati in HDFS, inserendo dal terminale del namenode i comandi:

```
1 #Creazione in HDFS directory lib
2 hadoop fs -mkdir /lib
3 #Copia da locale del file eseguibile nella cartella lib
4 hadoop fs -copyFromLocal $APP_JAR /lib/
5 #Copia da locale delle librerie necessarie nella cartella lib
6 hadoop fs -copyFromLocal $BOOK_HOME/lib/*.jar /lib/
7 #Creazione in HDFS directory anagram
8 hadoop fs -mkdir anagram
9 #Copia della directory di input nella cartella anagram
10 hadoop fs -put anagram/ anagram
11 #Lettura del file input
12 hadoop fs -cat anagram/input/anagram.txt
```



Al termine delle operazioni preliminari appena esposte, il programma può essere correttamente lanciato in esecuzione con i seguenti comandi dalla shell del namenode:

-
- 1 `hadoop jar $APP_JAR $DRIVER 2 $INPUT $OUTPUT`
 - 2 `hadoop fs -cat anagram/output/part-r-00000`
-



Test: input

L'input utilizzato per testare il programma è caratterizzato da un file testuale suddiviso per righe, al cui interno sono presenti parole che tra loro costituiscono anagrammi. Il file adottato è mostrato nel codice seguente:

```
1 Mary and Elvis lives in Detroit army Easter Listen
2 a silent eaters Death Hated elvis Mary easter Silent
3 Mary and Elvis are in a army Listen Silent detroit
```



Test: output

L'output restituito dall'esecuzione del programma AnagramFinder è costituito dalla lista di chiavi, ovvero le lettere delle parole ordinate, cui è associato un elenco di valori, ovvero le parole individuate come anagramma.

```
root@e65d8a96c2f0:/# hadoop fs -cat anagram/output/part-r-00000
WARNING: HADOOP_PREFIX has been replaced by HADOOP_HOME. Using value of HADOOP_PREFIX.
2020-01-07 09:50:57,252 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrusted = false, remoteHostTrusted = false
adeht      [death, hated]
aeerst     [eaters, easter]
amry       [army, mary]
eillnst    [silent, listen]
eillsv     [lives, elvis]
```



Grazie per l'attenzione