Apache Spark

Índex

1. Introducció **1**
1.1. Funcionalitats principals . . . 1
1.2. Altres característiques . . . 2
1.3. Per què és més ràpid que MapReduce . . . 2
1.4. Arquitectura . . . 2
1.5. Abstraccions principals . . . 2
1.6. Mòduls de l'ecosistema . . . 3
1.7. Llenguatges suportats . . . 3
1.8. Relació amb Hadoop . . . 3
1.9. Casos d'ús típics . . . 3

2. Instal·lació d'un clúster Spark **3**
2.1. Arquitectura del miniclúster . . . 4
2.2. Requisits previs (a totes les VM) . . . 4
Actualitza el sistema . . . 4
Instal·la Java . . . 4
Configura `/etc/hosts` . . . 4
Crea un usuari dedicat (opcional però recomanable) . . . 5
Accés SSH sense contrasenya (master → workers) . . . 5
2.3. Instal·lació de Spark (a totes les VM) . . . 6
Descarrega i descomprimeix . . . 6
Variables d'entorn . . . 6
2.4. Configura el clúster (només al master) . . . 7
Fitxer `workers` . . . 7
Fitxer `spark-env.sh` . . . 7
2.5. Arrenca el clúster . . . 8
Verifica l'estat . . . 8
Interfície web . . . 9
2.6. Prova el clúster amb PySpark . . . 9
2.7. Atura el clúster . . . 10
2.8. Converteix en servei systemd (Opcional) . . . 11

3. Resolució de problemes habituals **11**

Índex de figures **12**

Apache Spark és un motor de processament de dades distribuït i de codi obert, dissenyat per fer càlculs sobre grans volums de dades de manera ràpida, tant en batch (per lots) com en streaming (temps real). La seva característica principal és el processament **en memòria**, que el fa molt més ràpid que alternatives com Hadoop MapReduce en molts casos d'ús, especialment els algorismes iteratius.

Figura 1: Apache Spark logo

1. Introducció

Va ser creat per **Matei Zaharia** l'any 2009, a l'**AMPLab de la Universitat de Califòrnia, Berkeley**, com a projecte de recerca. El 2010 es va publicar com a codi obert, i el 2013 va passar a ser un projecte de l'**Apache Software Foundation**, on el 2014 va assolir l'estatus de projecte de nivell superior (*Top-Level Project*).

Spark es distribueix sota la **Apache License 2.0**, una llicència de programari lliure permissiva que permet l'ús, modificació i distribució (fins i tot comercial) del codi, sense obligar a alliberar les modificacions sota la mateixa llicència (a diferència de llicències copyleft com la GPL).

1.1. Funcionalitats principals

Mòdul	Descripció
Spark Core	Nucli del motor: gestió de tasques, planificació, i abstraccions bàsiques (RDD, DataFrames, Datasets). Processament distribuït en memòria amb tolerància a fallades
Spark SQL	Consultes SQL sobre dades estructurades, amb l'optimitzador de consultes Catalyst i el motor d'execució Tungsten
Structured Streaming	Processament de fluxos de dades en temps real (micro-batching), amb la mateixa API que el processament per lots
MLlib	Biblioteca de machine learning distribuïda: classificació, regressió, clustering, sistemes de recomanació, reducció de dimensionalitat, pipelines de ML
GraphX	Processament i anàlisi de grafs (xarxes socials, algorismes com PageRank)
SparkR / PySpark	APIs per treballar amb R i Python, a més de Scala (llenguatge natiu) i Java

1.2. Altres característiques

- **Multillenguatge**: Scala, Java, Python, R
- **Multiplataforma d'execució**: standalone, YARN, Mesos o Kubernetes
- **Compatibilitat amb fonts de dades**: HDFS, S3, Cassandra, HBase, JDBC, Kafka, entre d'altres
- **Escalabilitat**: des d'un sol portàtil fins a clústers de milers de nodes

1.3. Per què és més ràpid que MapReduce

MapReduce escriu resultats intermedis a disc entre cada fase Map i Reduce, cosa que genera molta latència. Spark, en canvi, manté les dades en memòria RAM entre operacions sempre que és possible, cosa que pot arribar a fer-lo **10-100 vegades més ràpid** en determinats casos d'ús (especialment algorismes iteratius com machine learning).

1.4. Arquitectura

- **Driver**: coordina l'execució, crea el pla d'execució i distribueix la feina
- **Cluster Manager**: gestiona els recursos (pot ser el mateix Spark Standalone, YARN, Mesos o Kubernetes)
- **Executors**: processos als nodes worker que executen les tasques i emmagatzemen dades en memòria/disc

1.5. Abstraccions principals

RDD (Resilient Distributed Dataset)

- L'estructura de dades original de Spark: col·lecció distribuïda i immutable, tolerant a fallades
- Nivell baix, més control però més verbós.

DataFrames

- Dades organitzades en columnes (com una taula SQL), amb optimitzacions automàtiques via el **Catalyst optimizer**
- És l'API més utilitzada avui dia.

Datasets

- Combina els avantatges de tipat fort (com RDD) amb les optimitzacions dels DataFrames (disponible principalment a Scala/Java).

1.6. Mòduls de l'ecosistema

- **Spark SQL**: consultes SQL sobre dades estructurades
- **Spark Streaming / Structured Streaming**: processament de dades en temps real (micro-batching)
- **MLlib**: biblioteca de machine learning distribuït
- **GraphX**: processament de grafs

1.7. Llenguatges suportats

Scala (llenguatge natiu de Spark), Python (PySpark, molt popular), Java, i R.

1.8. Relació amb Hadoop

Spark **no substitueix HDFS**; sovint funciona sobre HDFS com a sistema d'emmagatzematge, i pot executar-se sobre YARN com a gestor de recursos. El que substitueix és el motor de processament MapReduce. Molts clústers moderns són “Hadoop” només pel que fa a HDFS+YARN, però amb Spark com a motor de càlcul principal.

1.9. Casos d'ús típics

- ETL a gran escala
- Anàlisi de dades en temps real (streaming)
- Machine learning distribuït
- Processament de grafs (xarxes socials, recomanadors)

2. Instal·lació d'un clúster Spark

Aquest document explica com desplegar un clúster Spark en **mode standalone** utilitzant diverses màquines virtuals Ubuntu Server. És un exercici ideal per entendre l'arquitectura master/worker d'un sistema de processament distribuït, sense necessitat de muntar un clúster Hadoop complet.

NOTA

Es parteix d'un escenari amb 3 VM (`spark-master`, `spark-worker1`, `spark-worker2`), però funciona igual amb 2 o més nodes. Totes les VM han de tenir connectivitat de xarxa entre elles.

2.1. Arquitectura del miniclúster

Node	Rol	Hostname	IP (exemple)
VM 1	Master	spark-master	10.0.2.10
VM 2	Worker	spark-worker1	10.0.2.11
VM 3	Worker	spark-worker2	10.0.2.12

El **master** coordina l'execució dels jobs i exposa una interfície web de monitoratge. Els **workers** executen les tasques reals i reporten l'estat al master.

2.2. Requisits previs (a totes les VM)

Actualitza el sistema

Actualitza la llista de paquets:

```
sudo apt update
```

Actualitza els paquets:

```
sudo apt upgrade
```

Instal·la Java

Spark necessita un JDK (recomanat OpenJDK 17):

```
sudo apt install openjdk-17-jdk
```

Configura `/etc/hosts`

A **totes** les VMs, afegeix les entrades de tots els nodes perquè es resolguin per nom:

```
sudo nano /etc/hosts
```

```
10.0.2.10 spark-master
10.0.2.11 spark-worker1
10.0.2.12 spark-worker2
```

Crea un usuari dedicat (opcional però recomanable)

Crea l'usuari:

```
sudo adduser spark
```

Afegeix l'usuari al grup de sudoers:

```
sudo usermod -aG sudo spark
```

Accés SSH sense contrasenya (master → workers)

Des del `spark-master`, com a usuari `spark`, genera un parell de claus SSH:

```
ssh-keygen -t ed25519
```

Copia la clau pública generada al primer worker:

```
ssh-copy-id spark@spark-worker1
```

Copia la clau pública generada al segon worker:

```
ssh-copy-id spark@spark-worker2
```

CONSELL

Això permetrà als scripts `start-all.sh`/`stop-all.sh` arrencar i aturar tots els workers remotament des del master.

2.3. Instal·lació de Spark (a totes les VM)

Descarrega i descomprimeix

Mou-te al directori `opt`:

```
cd /opt
```

Descarrega Spark:

```
sudo wget -c https://dlcdn.apache.org/spark/spark-4.2.0/spark-4.2.0-
↪  bin-hadoop3.tgz
```

Descomprimeix:

```
sudo tar -xzf spark-4.2.0-bin-hadoop3.tgz
```

Canvia el nom de la carpeta:

```
sudo mv spark-4.2.0-bin-hadoop3 spark
```

Canvia el propietari:

```
sudo chown -R spark:spark /opt/spark
```

Esborra el fitxer descarregat:

```
sudo rm spark-4.2.0-bin-hadoop3.tgz
```

AVÍS

Comprova sempre la versió actual a https://spark.apache.org/downloads.html --- els números de versió canvien amb el temps i l'enllaç pot quedar desactualitzat.

Variables d'entorn

Edita `/etc/profile.d/spark.sh`:

```
sudo nano /etc/profile.d/spark.sh
```

```
export JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64
export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
```

Dona permís d'execució:

```
sudo chmod +x /etc/profile.d/spark.sh
```

Executa l'script `spark.sh` al context (shell) actual:

```
source /etc/profile.d/spark.sh
```

NOTA

Per què és important fer `source` i no simplement executar-lo (./spark.sh o bash spark.sh)? Si executessis l'script normalment (com a subprocés), les variables que defineix (per exemple SPARK_HOME, afegir Spark al PATH, JAVA_HOME, etc.) es perdrien en acabar l'script, ja que només afectarien aquell subprocés. Amb source, aquestes variables queden exportades a la teva shell actual i, per tant, disponibles per a les comandes següents (com spark-submit, pyspark, spark-shell, etc.).

Repeteix aquest pas 2 a totes les VM (master i workers).

2.4. Configura el clúster (només al master)

Fitxer `workers`

```
cd /opt/spark/conf
sudo cp workers.template workers
sudo nano workers
```

Substitueix el contingut per la llista de workers:

```
spark-worker1
spark-worker2
```

Fitxer `spark-env.sh`

```
sudo cp spark-env.sh.template spark-env.sh
sudo nano spark-env.sh
```

Afegeix:

```
export JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64
export SPARK_MASTER_HOST=spark-master
export SPARK_WORKER_CORES=2
export SPARK_WORKER_MEMORY=2g
```

Ajusta `SPARK_WORKER_CORES` i `SPARK_WORKER_MEMORY` segons els recursos assignats a cada VM.

2.5. Arrenca el clúster

Des del `spark-master`:

```
$SPARK_HOME/sbin/start-all.sh
```

Això arrenca el master localment i, via SSH, els workers definits al fitxer `workers`.

Verifica l'estat

Comprova els processos Java actius:

```
jps
```

Al master hauries de veure `Master`, i a cada worker un procés `Worker`.

Sortida esperada al master:

```
19587 Master
19674 Jps
```

Sortida esperada al worker1:

```
19751 Worker
19819 Jps
```

Sortida esperada al worker2:

```
19939 Jps
19871 Worker
```

Interfície web

Obre al navegador:

```
http://spark-master:8080
```

Hi veuràs els workers registrats, els cores i memòria disponibles, i els jobs en execució.

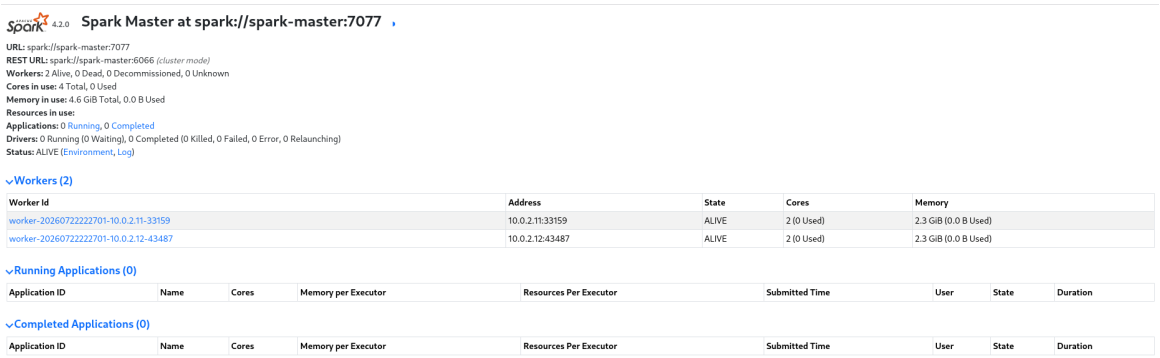


Figura 2: Interfície d'Spark

2.6. Prova el clúster amb PySpark

Instal·la Python:

```
sudo apt install python3-pip
```

Llança una shell PySpark connectada al clúster:

```
pyspark --master spark://spark-master:7077
```

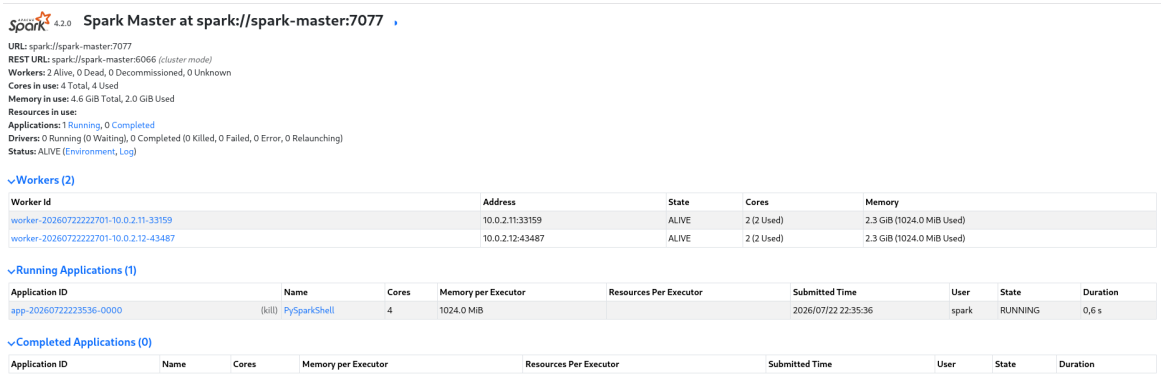
Dins la shell, prova un càlcul distribuït senzill: "*Compta quants múltiples de 7 hi ha entre 1 i 1.000.000*"

```
data = range(1, 1000000)
rdd = spark.sparkContext.parallelize(data)
print(rdd.filter(lambda x: x % 7 == 0).count())
```

NOTA

Crea una seqüència de Python amb els nombres de l'1 a l'1.000.000. Distribueix aquesta seqüència pel clúster d'Spark, creant un RDD (*Resilient Distributed Dataset*). Les dades es divideixen en particions per poder processar-les en paral·lel. Filtra els nombres divisibles entre 7 i retorna quants elements queden després del filtre.
Un detall important: fins que no es crida `count()`, Spark no ha fet realment cap càlcul. Les transformacions només construeixen el pla d'execució.

Si mires la interfície web (`http://spark-master:4040` mentre la shell és oberta), veuràs el job executant-se i repartint-se entre els workers.



Spark Master at spark://spark-master:7077

URL: spark://spark-master:7077
REST URL: spark://spark-master:6066 (cluster mode)
Workers: 2 Alive, 0 Dead, 0 Decommissioned, 0 Unknown
Cores in use: 4 Total, 4 Used
Memory in use: 4.6 GiB Total, 2.0 GiB Used
Resources in use:
Applications: 1 Running, 0 Completed
Drivers: 0 Running (0 Waiting), 0 Completed (0 Killed, 0 Failed, 0 Error, 0 Relaunching)
Status: ALIVE (Environment, Log)

Workers (2)

Worker Id	Address	State	Cores	Memory
worker-20260722222701-10.0.2.11-33159	10.0.2.11:33159	ALIVE	2 (2 Used)	2.3 GiB (1024.0 MiB Used)
worker-20260722222701-10.0.2.12-43487	10.0.2.12:43487	ALIVE	2 (2 Used)	2.3 GiB (1024.0 MiB Used)

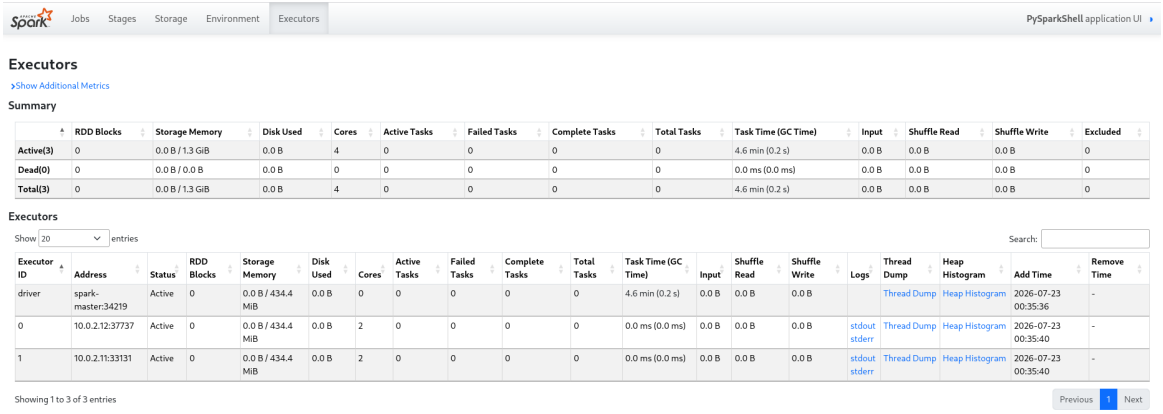
Running Applications (1)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20260722223536-0000 (kill)	PySparkShell	4	1024.0 MiB		2026/07/22 22:35:36	spark	RUNNING	0,6 s

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration

Figura 3: Interfície http://spark-master:8080



Jobs Stages Storage Environment Executors — PySparkShell application UI

Executors

Show Additional Metrics

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(3)	0	0.0 B / 1.3 GiB	0.0 B	4	0	0	0	0	4.6 min (0.2 s)	0.0 B	0.0 B	0.0 B	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(3)	0	0.0 B / 1.3 GiB	0.0 B	4	0	0	0	0	4.6 min (0.2 s)	0.0 B	0.0 B	0.0 B	0

Executors

Show 20 entries — Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump	Heap Histogram	Add Time	Remove Time
driver	spark-master:34219	Active	0	0.0 B / 434.4 MiB	0.0 B	0	0	0	0	0	4.6 min (0.2 s)	0.0 B	0.0 B	0.0 B		Thread Dump	Heap Histogram	2026-07-23 00:35:36	-
0	10.0.2.12:37737	Active	0	0.0 B / 434.4 MiB	0.0 B	2	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump	Heap Histogram	2026-07-23 00:35:40	-
1	10.0.2.11:33131	Active	0	0.0 B / 434.4 MiB	0.0 B	2	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr	Thread Dump	Heap Histogram	2026-07-23 00:35:40	-

Showing 1 to 3 of 3 entries — Previous 1 Next

Figura 4: Interfície http://spark-master:4040

2.7. Atura el clúster

```
$SPARK_HOME/sbin/stop-all.sh
```

2.8. Converteix en servei systemd (Opcional)

Per no haver d'arrencar el clúster manualment cada cop, pots crear una unitat systemd al master:

```
sudo nano /etc/systemd/system/spark-master.service
```

```
[Unit]
Description=Apache Spark Master
After=network.target

[Service]
Type=forking
User=spark
ExecStart=/opt/spark/sbin/start-master.sh
ExecStop=/opt/spark/sbin/stop-master.sh

[Install]
WantedBy=multi-user.target
```

I de forma anàloga a cada worker, amb `start-worker.sh spark://spark-master:7077`.

```
sudo systemctl daemon-reload
sudo systemctl enable --now spark-master
```

3. Resolució de problemes habituals

Símptoma	Causa probable	Solució
El worker no apareix a la UI	SSH sense contrasenya no configurat	Revisar `ssh-copy-id` i permisos `~/.ssh`
`JAVA_HOME not set`	Variable no exportada al servei	Afegir `JAVA_HOME` a `spark-env.sh`
Job es queda “en cua” indefinidament	No hi ha cores/memòria disponibles	Revisar `SPARK_WORKER_CORES`/`SPARK_WORKER_MEMORY`
No es resol el hostname del master	`/etc/hosts` incomplet en algun node	Verificar que totes les VM tenen totes les entrades

Índex de figures

1 Apache Spark logo 1
2 Interfície d'Spark 9
3 Interfície http://spark-master:8080 10
4 Interfície http://spark-master:4040 10

Versions d'aquest document

- HTML
- PDF
- ODT
- MD