Big Data
(Hadoop)

Instructor: Thanh Binh Nguyen

September 1st, 2019

S3Lab
Smart Software System Laboratory

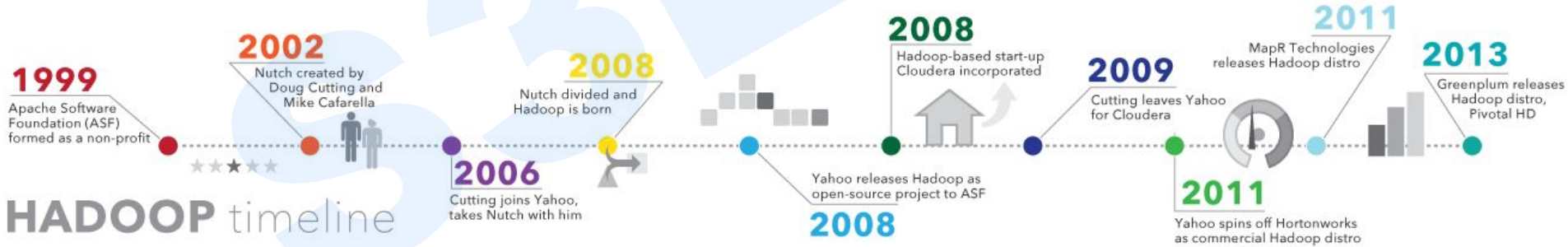

“Big data is at the foundation of all the megatrends that are happening today, from social to mobile to cloud to gaming.”
– Chris Lynch, Vertica Systems

Introduction

- Hadoop is a software framework for distributed processing of large datasets (terabytes or petabytes of data) across large clusters (thousands of nodes) of computers. Included some key components as below:
  - **Hadoop Common**: common utilities
  - **Hadoop Distributed File System** (**HDFS**) (Storage Component): A distributed file system that provides high-throughput access
  - **Hadoop YARN** (Scheduling): a framework for job scheduling & cluster resource management (available from Hadoop 2.x)
  - **Hadoop MapReduce** (Processing): A yarn-based system for parallel processing of large data sets

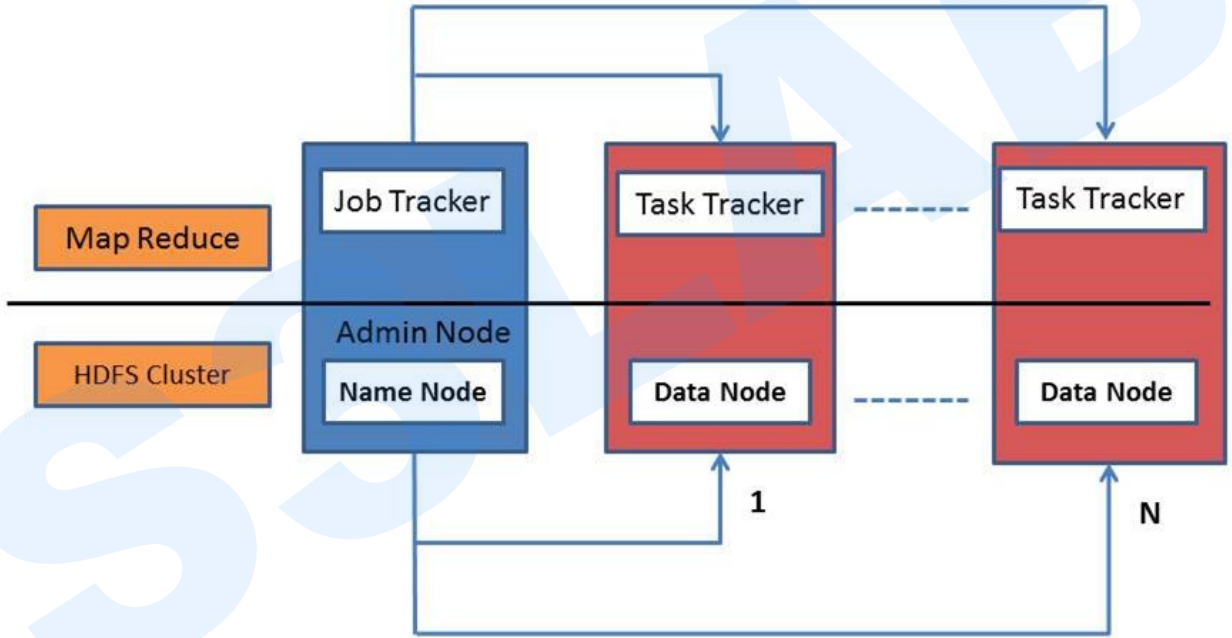
Introduction

- Hadoop is a large and active ecosystem.
- Hadoop emerged as a solution for big data problems.
- Open source under the friendly Apache License
- Originally built as a Infrastructure for the “Nutch” project.
- Based on Google’s mapreduce and google File System.

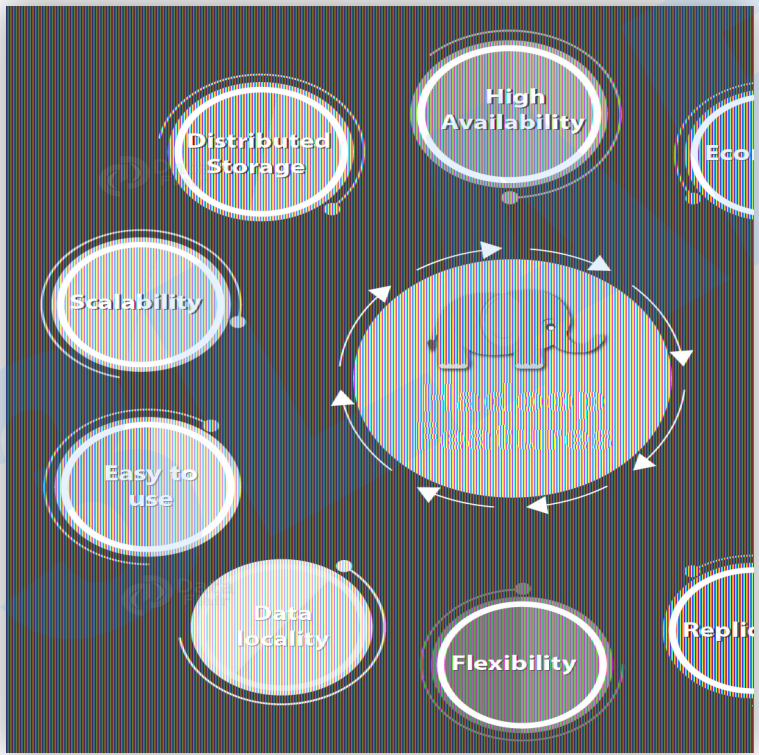


Introduction

Core components

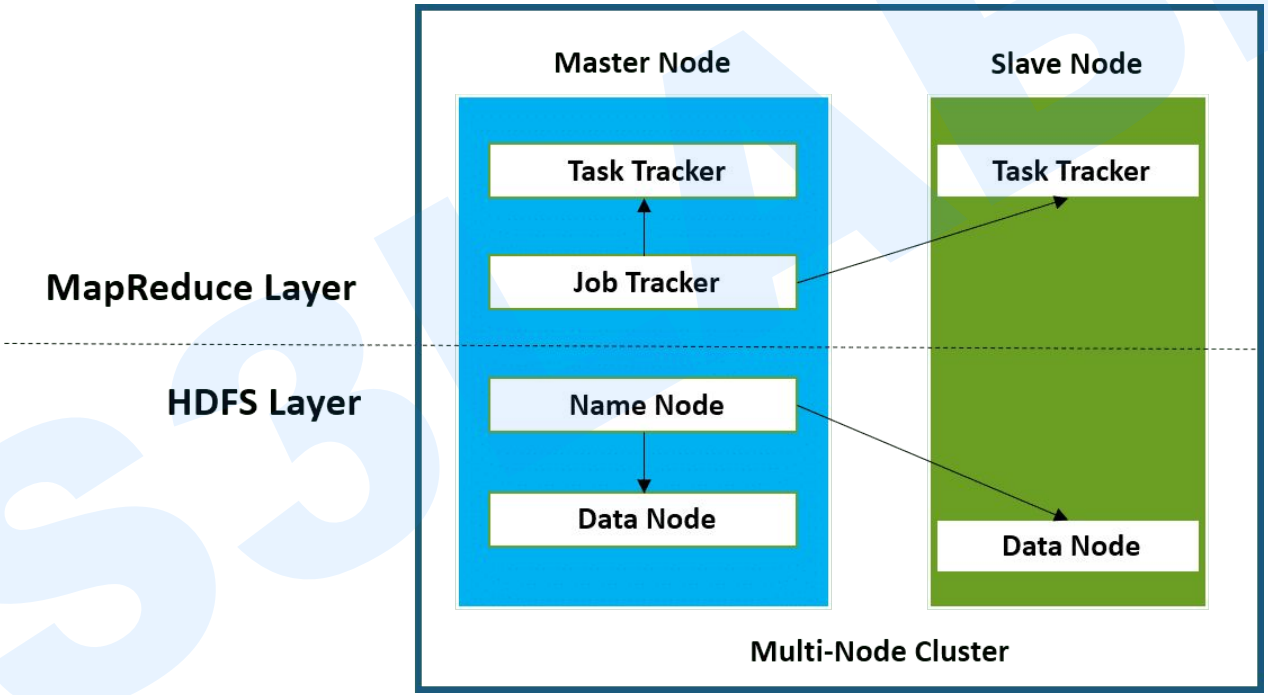


Features


Distributed Storage
High Availability
Scalability
Easy to use
Data locality
Flexibility

Architecture

Multi-Node Cluster

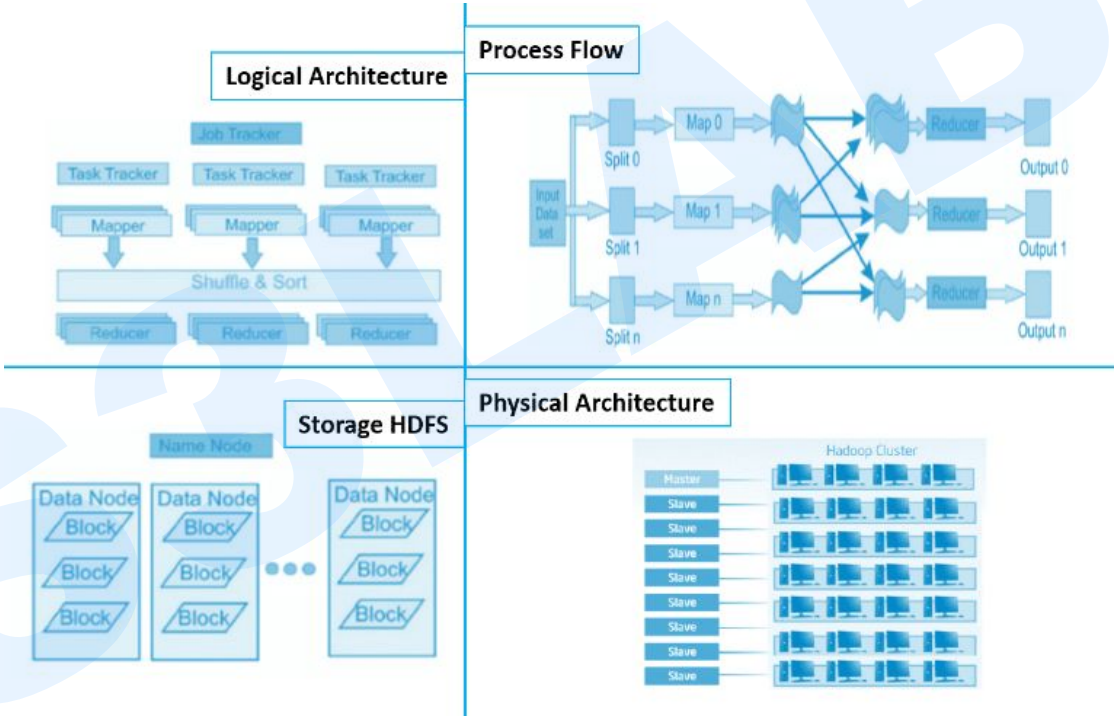


What makes Hadoop unique

- **Data locality and Shared Nothing**: Moving computation to data, instead of moving data to computation. Each node can independently process a much smaller subset of the entire dataset without needing to communicate with one another.
- **Simplified programming model**: allows user to quickly write and test
- **Schema-on-read system** (same as NoSQL platforms) # Schema-on-write system
- Automatic distribution of data and work across machines

Architecture

Architecture in different perspective



HDFS

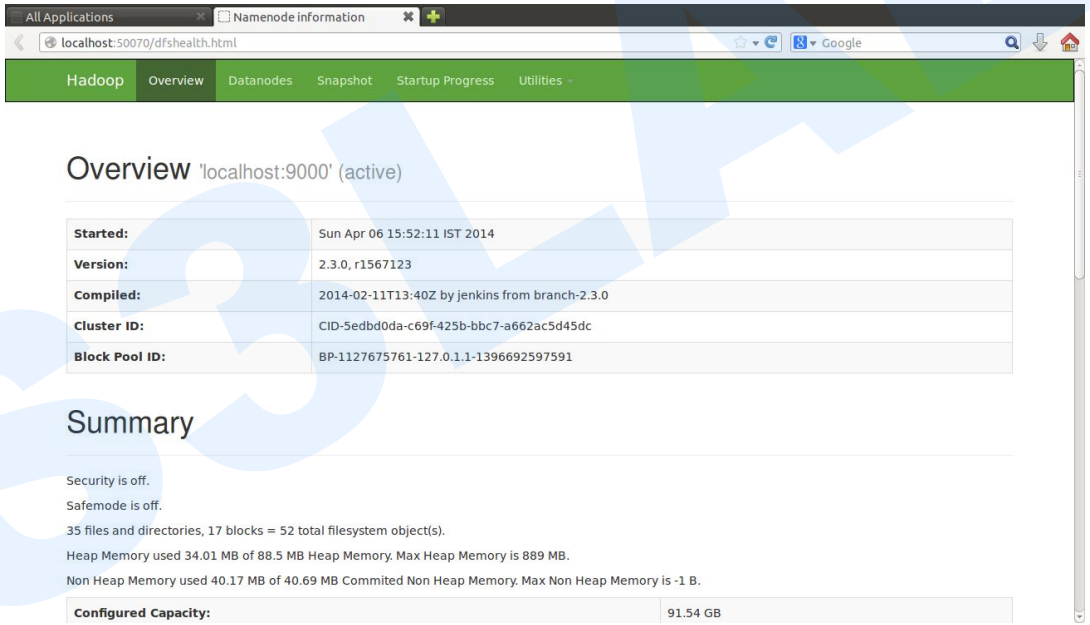
- **Hadoop Distributed File System** (**HDFS**) is designed to reliably store very large files across machines in a large cluster. It is inspired by the Google File System.
- Distribute large data file into blocks
- Blocks are managed by different nodes in the cluster
- Each block is replicated on multiple nodes

HDFS

- NameNode:
  - Master of the system, daemon runs on the master machine
  - Maintains, monitoring and manages the blocks which are present on the DataNodes
  - records the metadata of the files like the location of blocks, file size, permission, hierarchy etc.
  - captures all the changes to the metadata like deletion, creation and renaming of the file in edit logs.
  - It regularly receives heartbeat and block reports from the DataNodes.

HDFS

- All of the Hadoop server processes (daemons) serve a web UI. For NameNode, it was on port **50070**.



HDFS

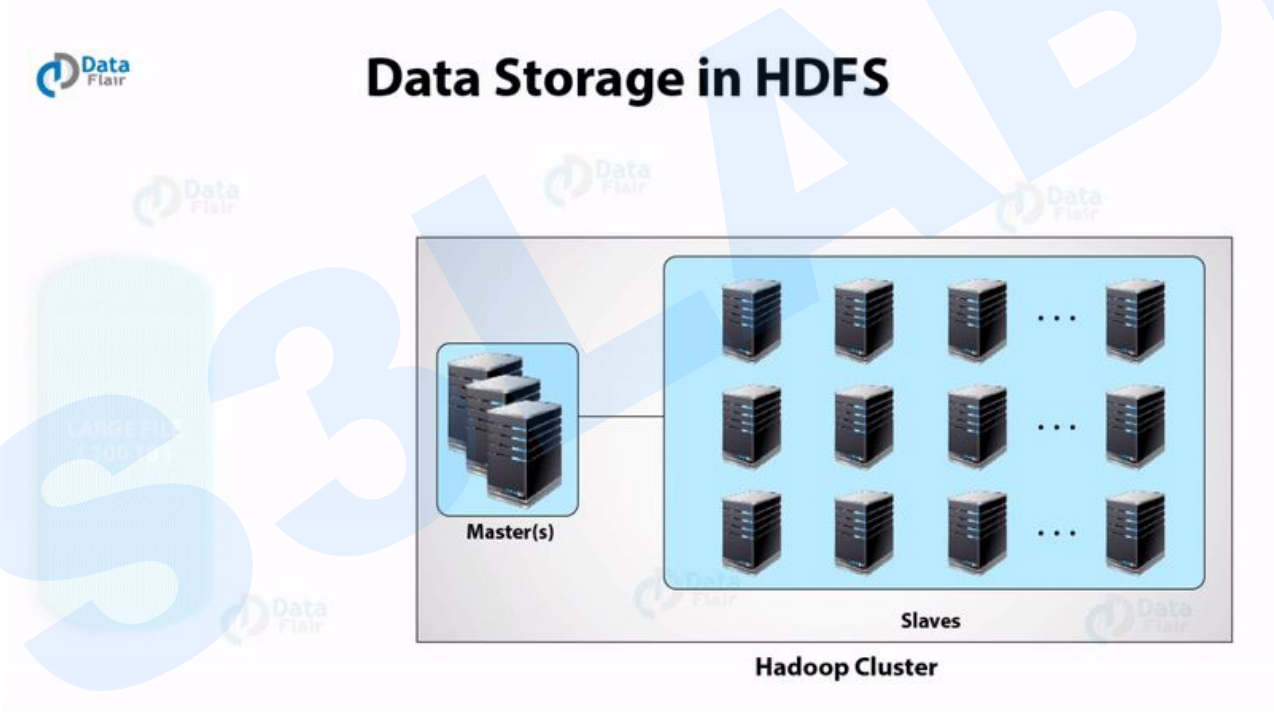
- DataNode:
  - DataNode runs on the slave machine.
  - It stores the actual business data.
  - It serves the read-write request from the user.
  - DataNode does the ground work of creating, replicating and deleting the blocks on the command of NameNode.
  - After every 3 seconds, by default, it sends heartbeat to NameNode reporting the health of HDFS.

HDFS

- Distributed file system for redundant storage
- Designed to reliably store data on **commodity hardware**
- Built to expect hardware failures
- Intended for
  - Large files
  - Batch inserts

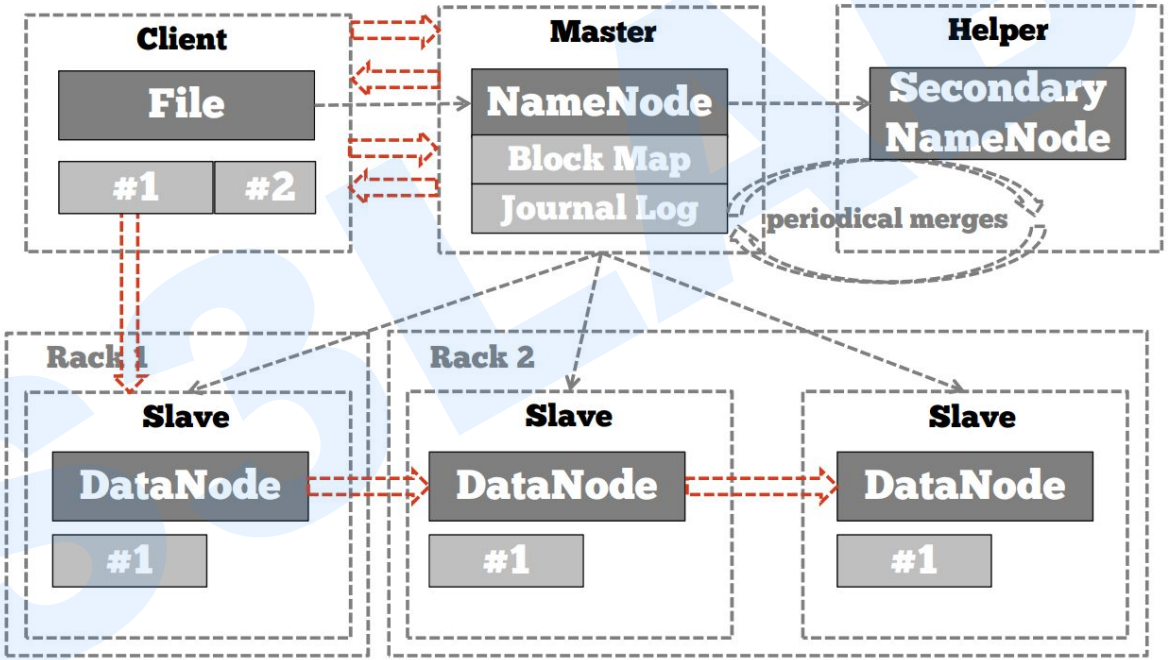
HDFS

Architecture



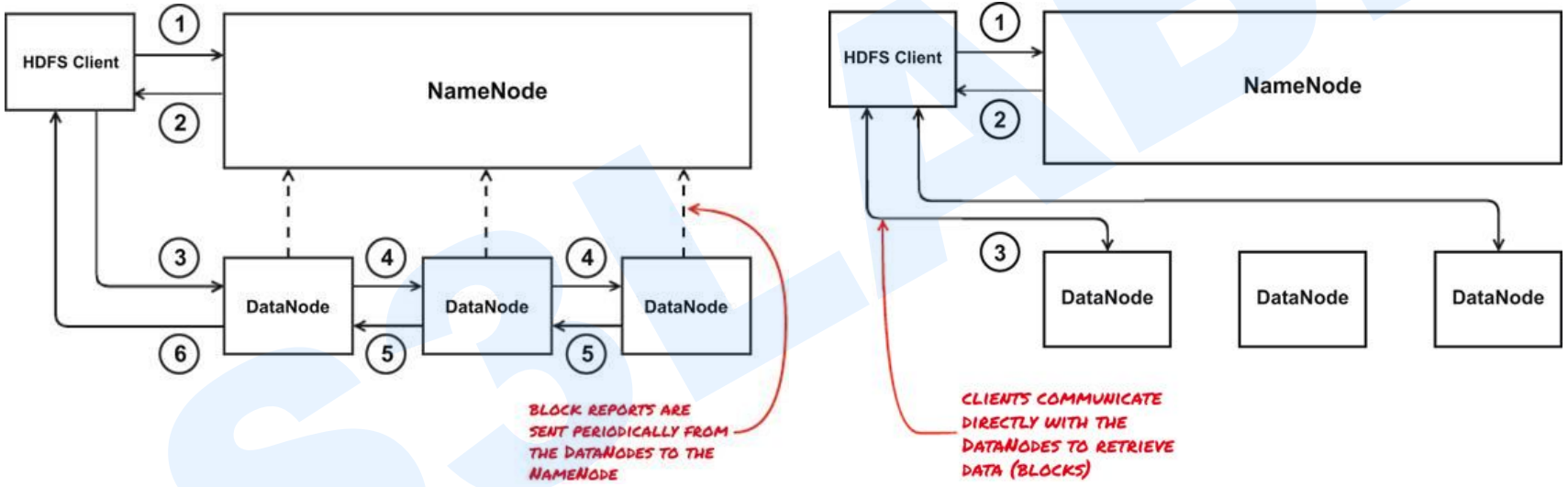
HDFS

Architecture



HDFS

Write & Read files



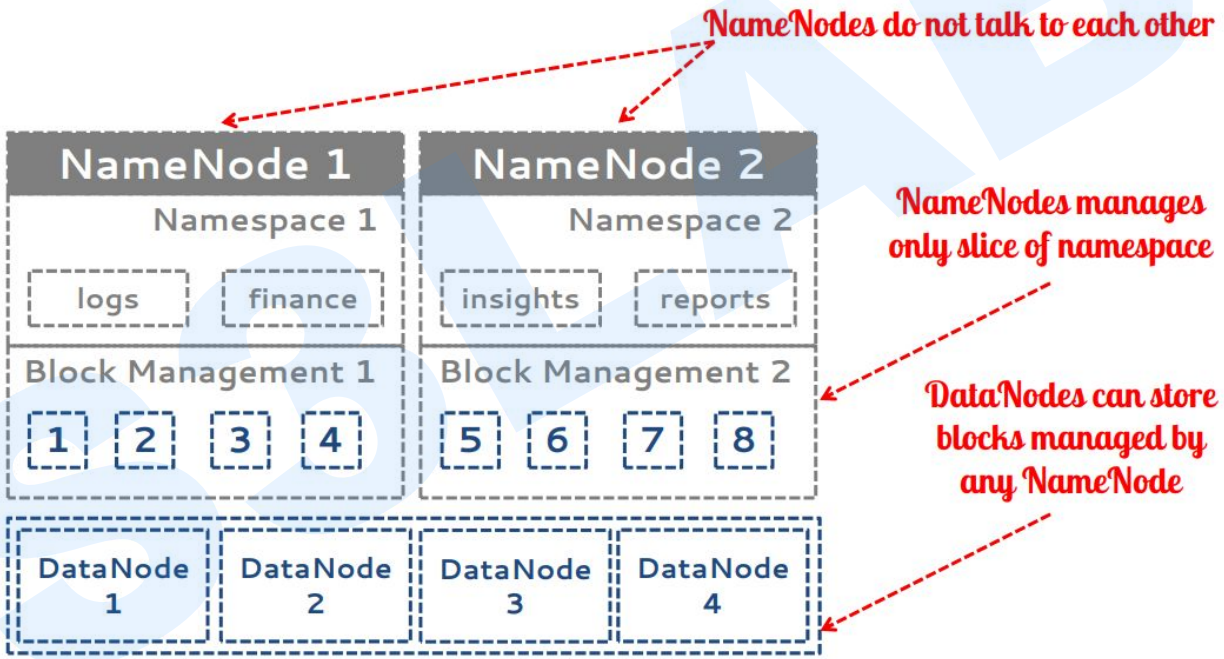
HDFS

HDFS 2 - Federation

- Removes tight coupling of Block
- Storage and Namespace
- Scalability & Isolation
- High Availability
- Increased performance

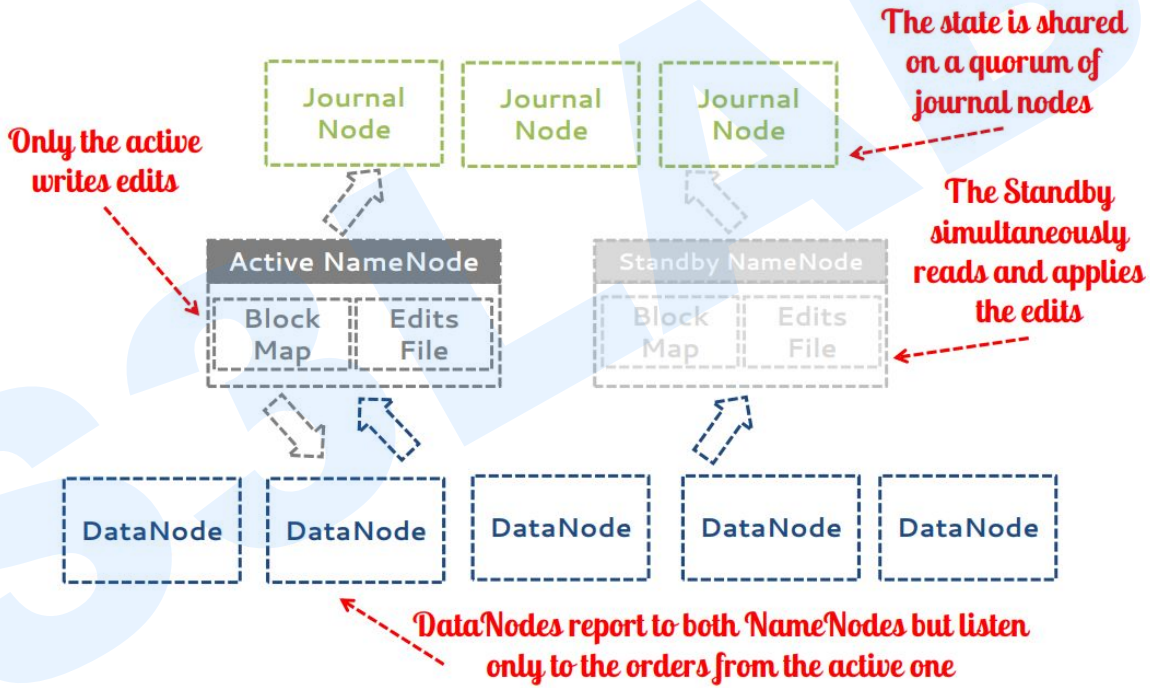
HDFS

HDFS 2 - Federation



HDFS

HDFS 2 (Federation): Quorum based storage

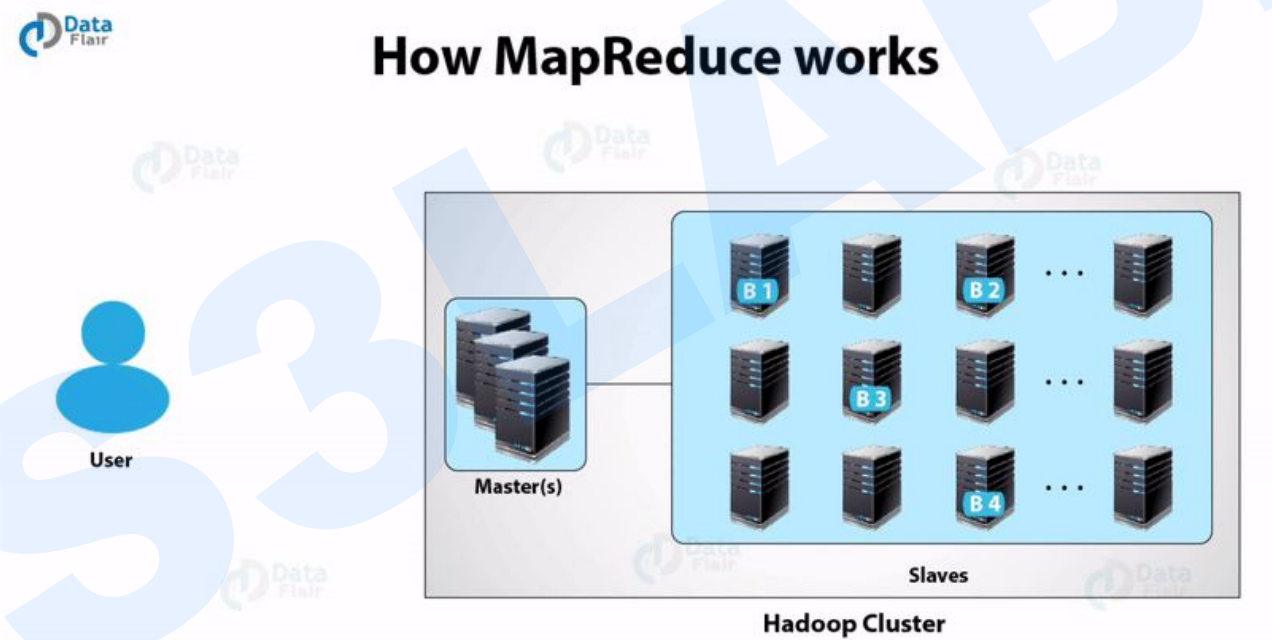


Map Reduce

- Programming model for distributed computations at a massive scale
- Execution framework for organizing and performing such computations
- Data locality is king
- JobTracker:
  - takes care of all the job scheduling and assign tasks to Task Trackers.
- TaskTracker
  - a node in the cluster that accepts tasks - Map, Reduce & Shuffle operations - from jobtracker

Map Reduce

flow

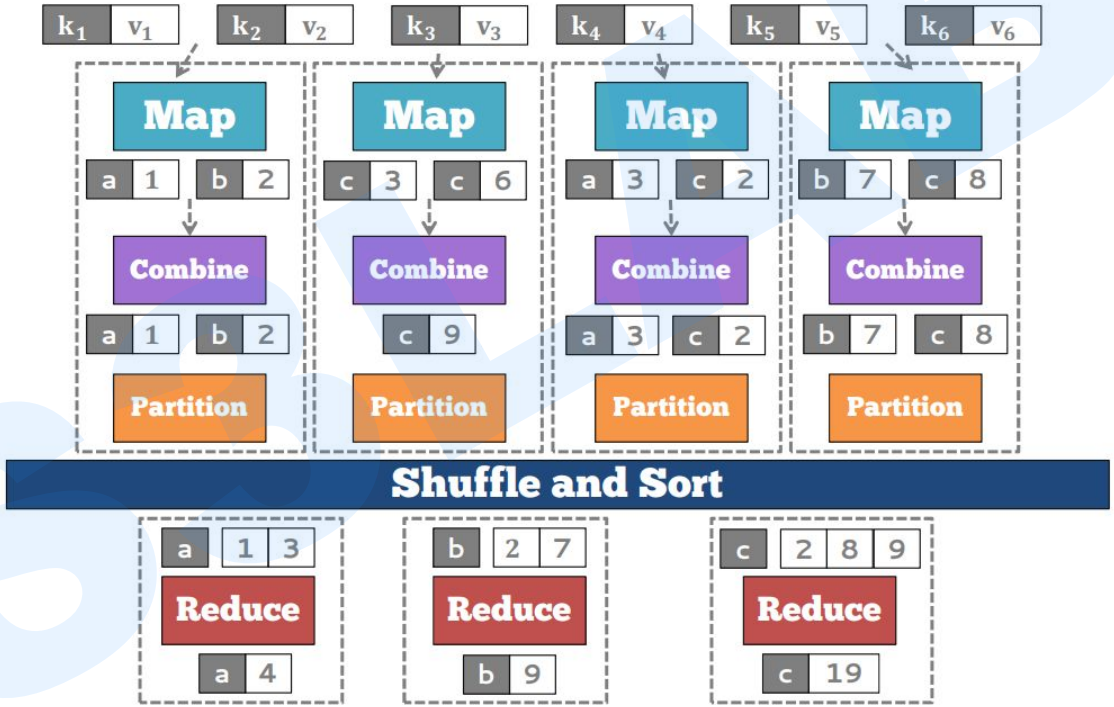


Map Reduce

- The Mapper:
  - Each block is processed in isolation by a map task called mapper
  - Map task runs on the node where the block is stored
  - Iterate over a large number of records
  - Extract something of interest from each
- Shuffle and sort intermediate results
- The Reducer:
  - Consolidate result from different mappers
  - Aggregate intermediate results
  - Produce final output

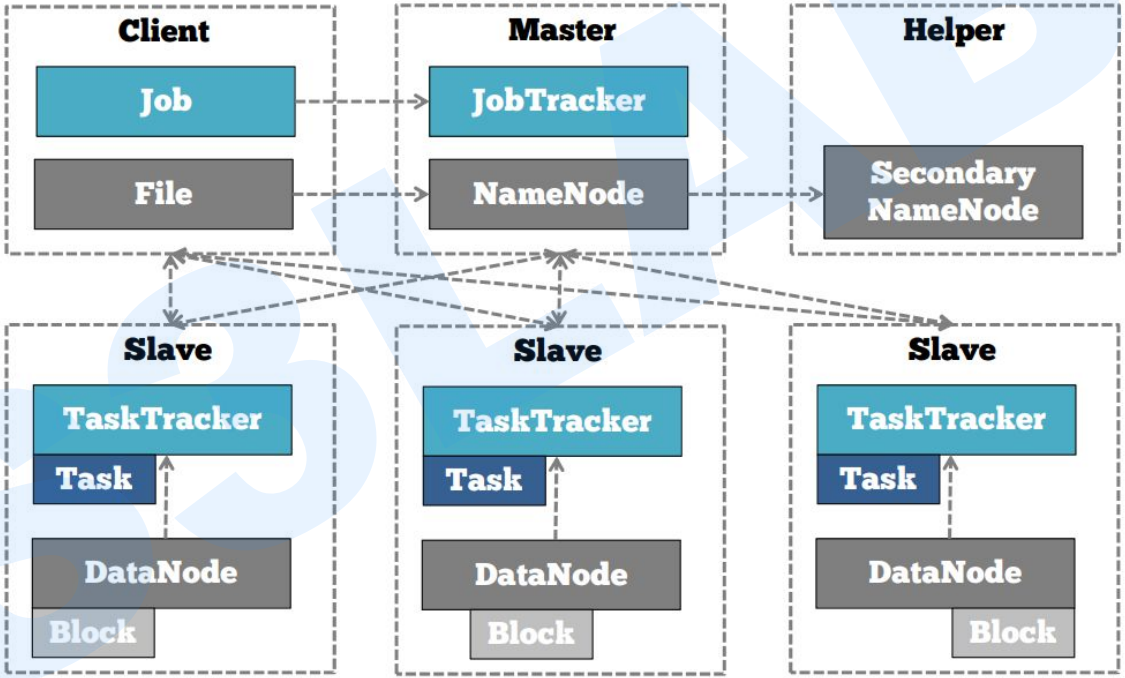
Map Reduce

flow



Map Reduce

Combined Hadoop architecture



Map Reduce

Good for ...

- Embarrassingly parallel algorithms
- Summing, grouping, filtering, joining
- Off-line batch jobs on massive data sets
- Analyzing an entire large dataset

Map Reduce

OK for ...

- Iterative jobs (i.e., graph algorithms)
  - Each iteration must read/write data to disk
  - I/O and latency cost of an iteration is high

Map Reduce

Not good for ...

- Jobs that need shared state/coordination
  - Tasks are shared-nothing
  - Shared-state requires scalable state store
- Low-latency jobs
- Jobs on small datasets
- Finding individual records

Map Reduce

limitations

- Scalability
  - Maximum cluster size ~ 4,500 nodes, concurrent tasks – 40,000
  - Coarse synchronization in JobTracker
- Availability
  - Failure kills all queued and running jobs
- Hard partition of resources into map & reduce slots
  - Low resource utilization
- Lacks support for alternate paradigms and services
  - Iterative applications implemented using MapReduce are 10x slower

Map Reduce

V2

- Basically a porting to the YARN architecture
- MapReduce becomes a user-land library
- No need to rewrite MapReduce jobs
- Increased scalability & availability
- Better cluster utilization

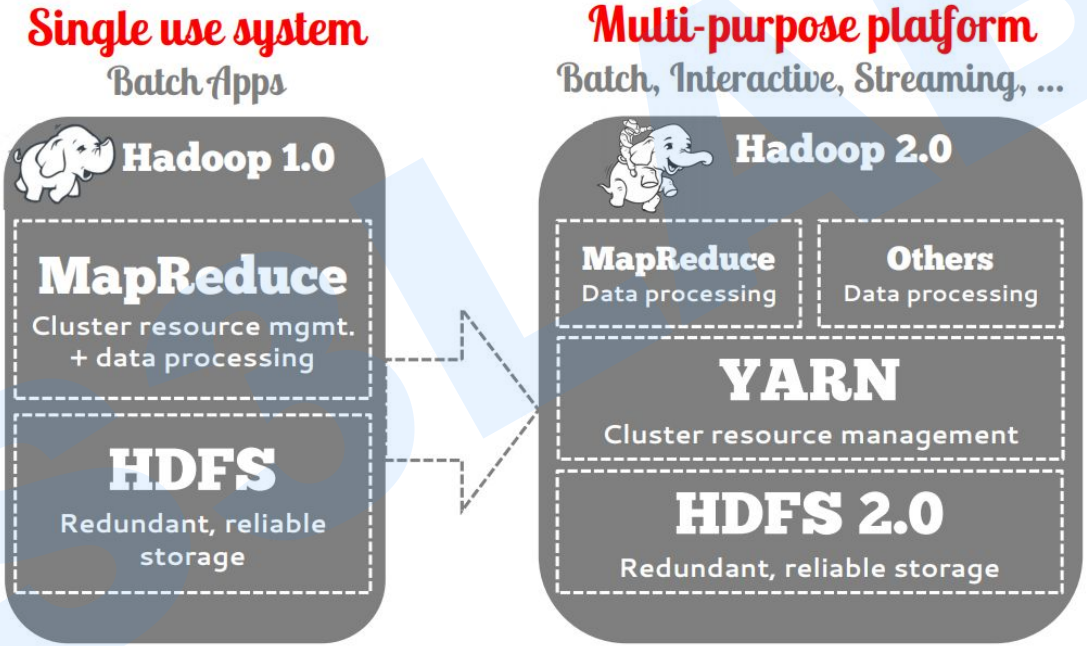
Hadoop 2.0

history

- Originally conceived & architected by the team at Yahoo!
  - Arun Murthy created the original JIRA in 2008 and now is the YARN release manager
- The team at Hortonworks has been working on YARN for 4 years:
  - 90% of code from Hortonworks & Yahoo!
- Hadoop 2.0 based architecture running at scale at Yahoo!
  - Deployed on 35,000 nodes for 6+ months

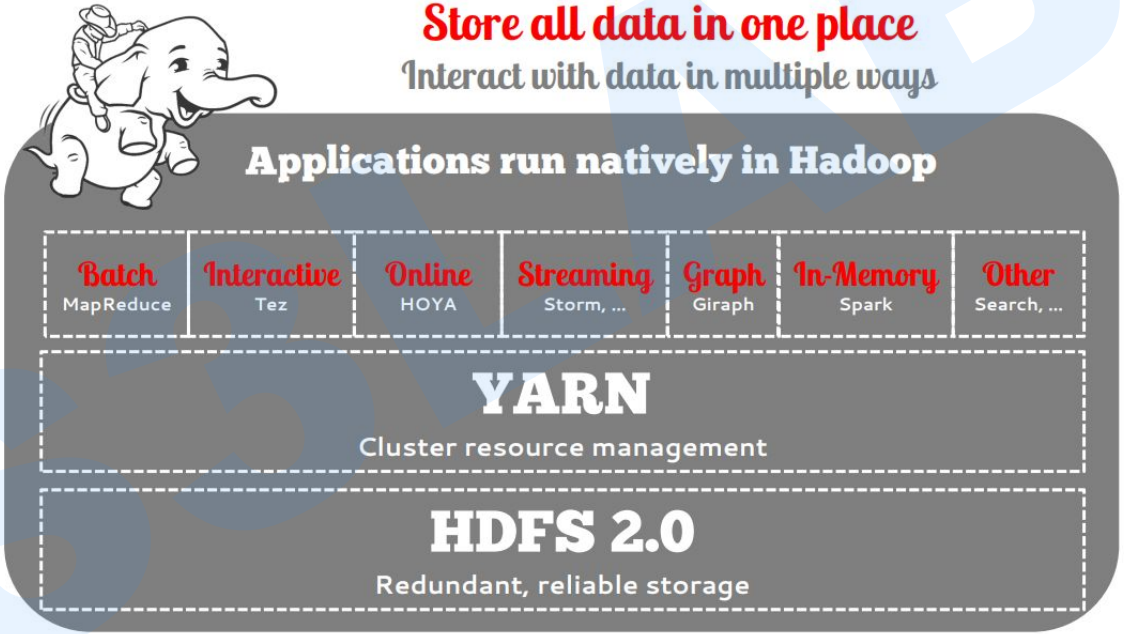
Hadoop 2.0

Next-gen platform



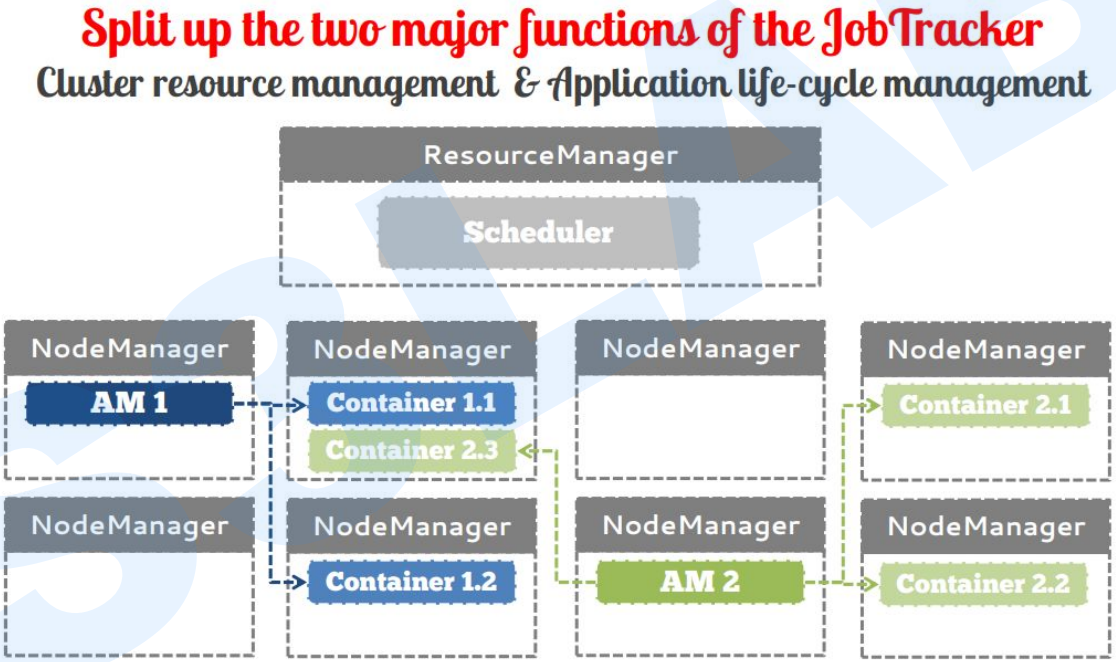
Hadoop 2.0

Taking Hadoop beyond batch



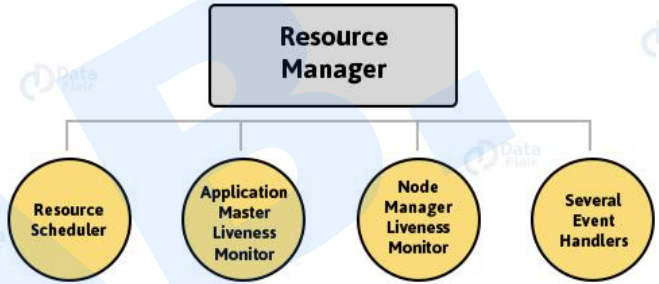
Yarn

Architecture



Yarn

Architecture

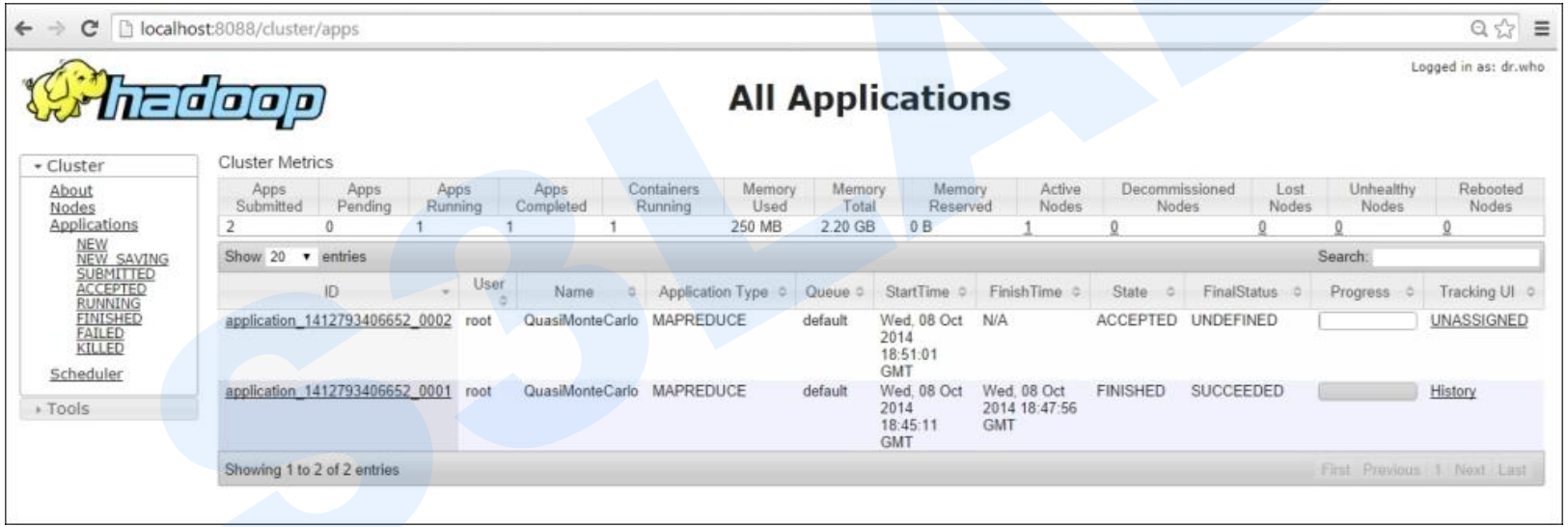


- Resource Manager
  - Resource Manager runs on the master node.
  - It knows where the location of slaves (Rack Awareness).
  - It is aware about how much resources each slave have.
  - Resource Scheduler is one of the important service run by the Resource Manager.
  - Resource Scheduler decides how the resources get assigned to various tasks.
  - Application Manager is one more service run by Resource Manager.
  - Application Manager negotiates the first container for an application.
  - Resource Manager keeps track of the heart beats from the Node Manager.

Yarn

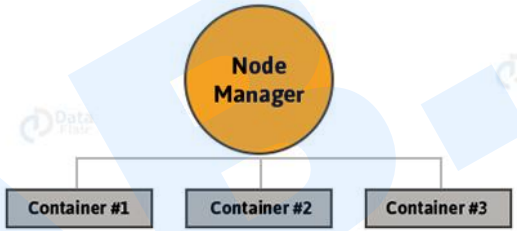
Architecture

- Resource Manager serves an embedded Web UI on port 8088



Yarn

Architecture

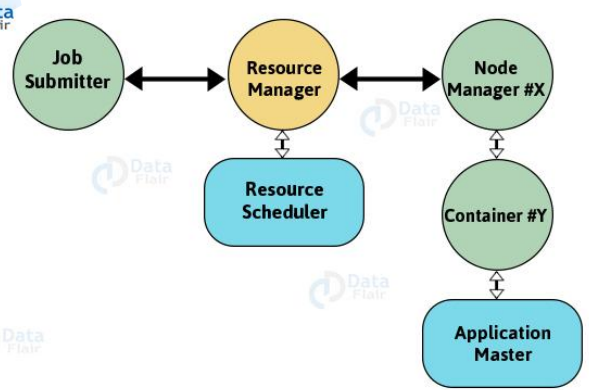


- Node Manager
  - It runs on slave machines.
  - It manages containers. Containers are nothing but a fraction of Node Manager's resource capacity
  - Node manager monitors resource utilization of each container.
  - It sends heartbeat to Resource Manager.

Yarn

Architecture

- Job submitter
  - The client submits the job to Resource Manager.
  - Resource Manager contacts Resource Scheduler and allocates container.
  - Now Resource Manager contacts the relevant Node Manager to launch the container.
  - Container runs Application Master



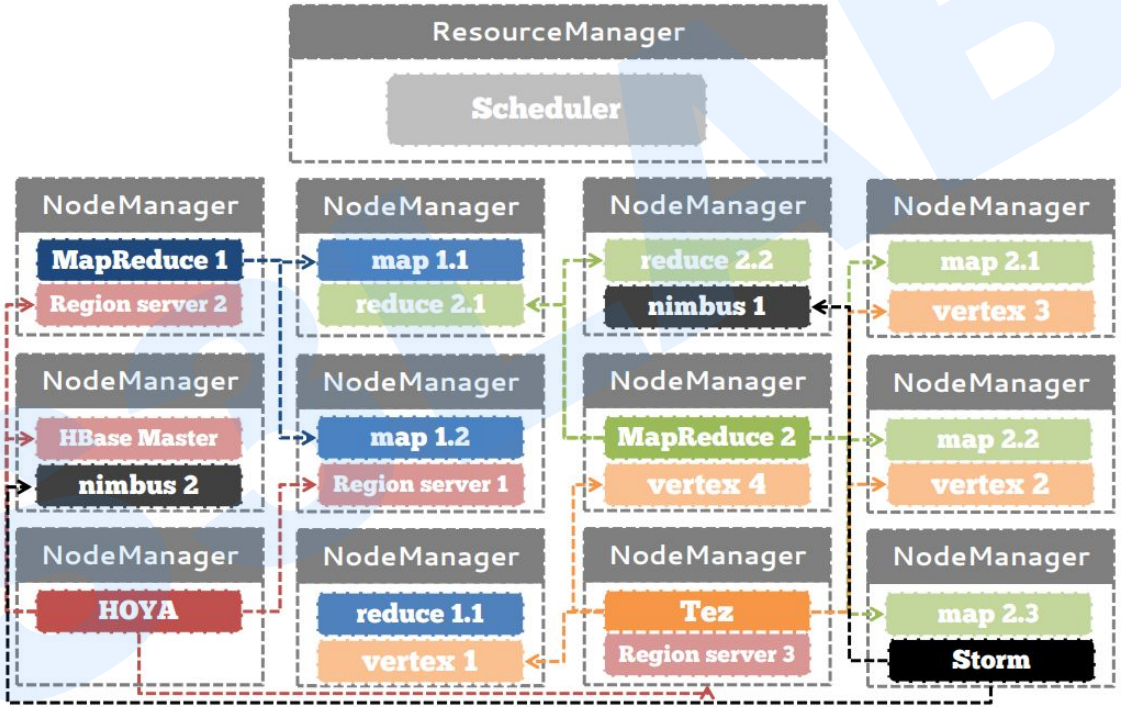
Yarn

Architecture

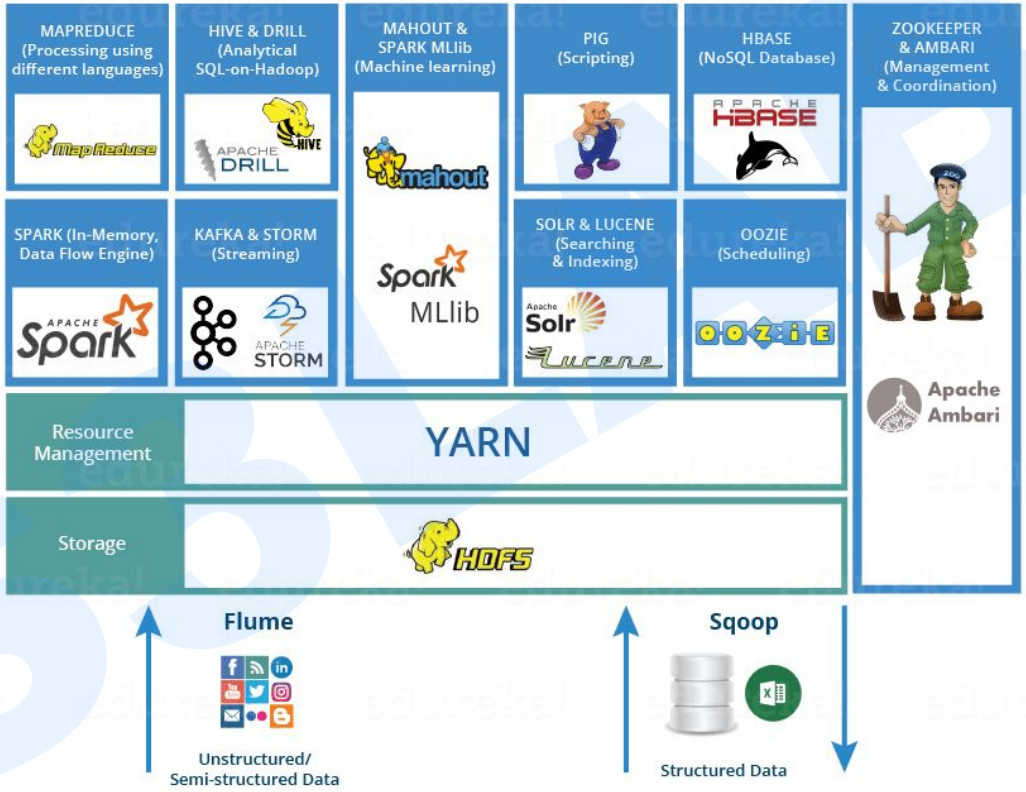
- Application Master
  - Per-application
  - Manages application scheduling and task execution
  - e.g. MapReduce Application Master

Yarn

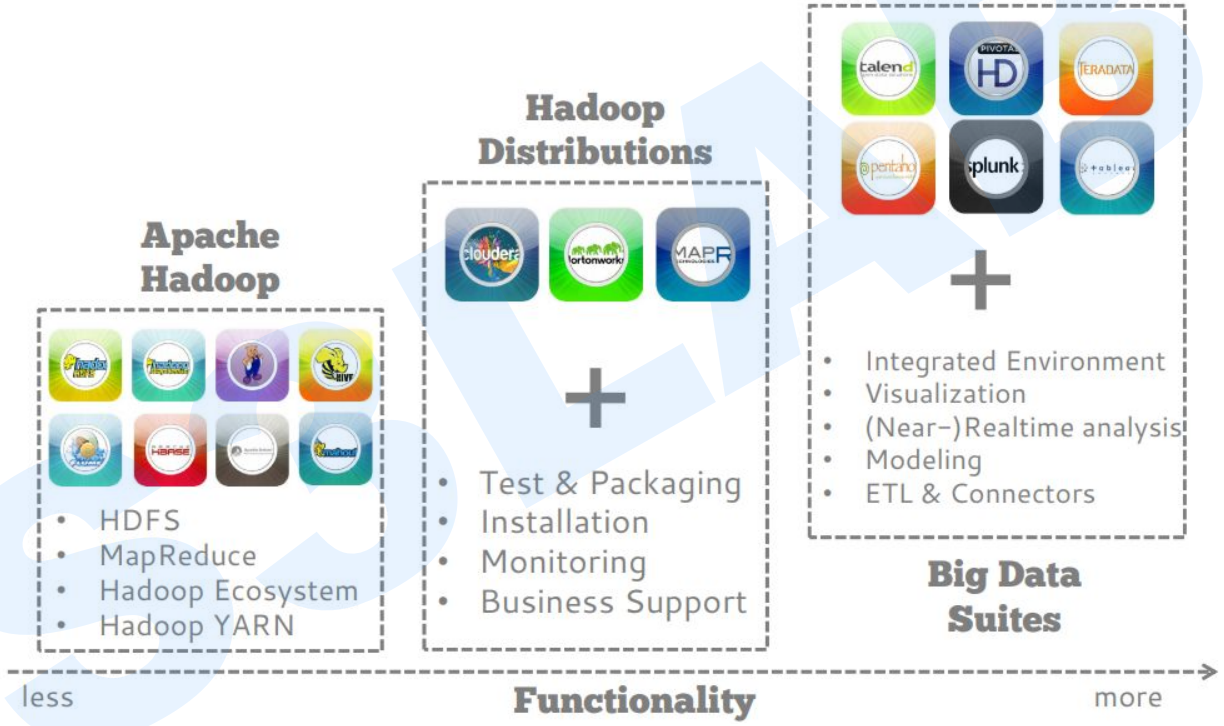
Architecture



Ecosystem


MAPREDUCE (Processing using different languages)
HIVE & DRILL (Analytical SQL-on-Hadoop)
MAHOUT & SPARK MLlib (Machine learning)
PIG (Scripting)
HBASE (NoSQL Database)
ZOOKEEPER & AMBARI (Management & Coordination)
SPARK (In-Memory, Data Flow Engine)
KAFKA & STORM (Streaming)
SOLR & LUCENE (Searching & Indexing)
OOZIE (Scheduling)
Resource Management
YARN
Storage
HDFS
Flume
Unstructured/ Semi-structured Data
Sqoop
Structured Data

Ecosystem


Apache Hadoop
• HDFS
• MapReduce
• Hadoop Ecosystem
• Hadoop YARN
Hadoop Distributions
+
• Test & Packaging
• Installation
• Monitoring
• Business Support
+
• Integrated Environment
• Visualization
• (Near-)Realtime analysis
• Modeling
• ETL & Connectors
Big Data Suites
less
Functionality
more

Ecosystem

- HDFS -> Hadoop Distributed File System
- YARN -> Yet Another Resource Negotiator
- MapReduce -> Data processing using programming
- PIG -> Top-level data processing engine
- HIVE-> Data warehouse on top of hadoop, using Query (SQL-like)
- HBase -> Column oriented NoSQL Database

Core Components

High-level data processing components

NoSQL

Ecosystem

- Apache Drill: Schema-free SQL Query Engine for Hadoop
- Solr & Lucene: High-performance text search engine
- Mahout, Spark MLlib -> Data Analysis and machine Learning
- Avro -> data serialization framework
- Thrift -> Interface definition language and binary communication protocol
- Oozie -> Server-based workflow scheduling system
- HCatalog -> Table and storage management layer

Hadoop Data Analysis components

Data serialization

Management Components

Ecosystem

Hadoop Data transfer (ingest) components

- Flume -> Data collection & aggregation system
- Sqoop -> tool designed for efficiently transferring bulk Data between hadoop and RDBMS
- Chukwa -> Data collection system for monitoring large distributed systems
- Ambari -> Hadoop deployment, management & monitoring tool
- Zookeeper -> Highly reliable distributed coordination system
- Hue (Hadoop user experience) -> Open-source hadoop web interface

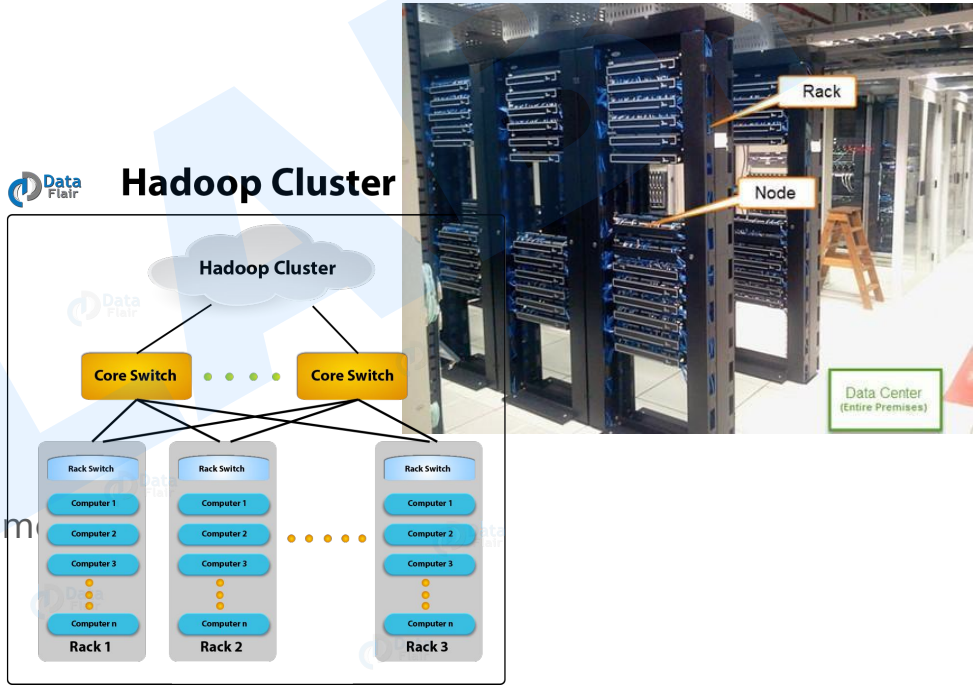
Monitoring components

Ecosystem

- Spark -> in memory dataflow engine
- Kafka, storm -> support stream data

Network Topology In Hadoop

- Topology of network:
  - Performance
  - Availability & handling of failures
- Hadoop cluster:
  - Processes on the same node
  - Different nodes on the same rack
  - Nodes on different racks of the same data center
  - Nodes in different data centers



When Not to Use Hadoop

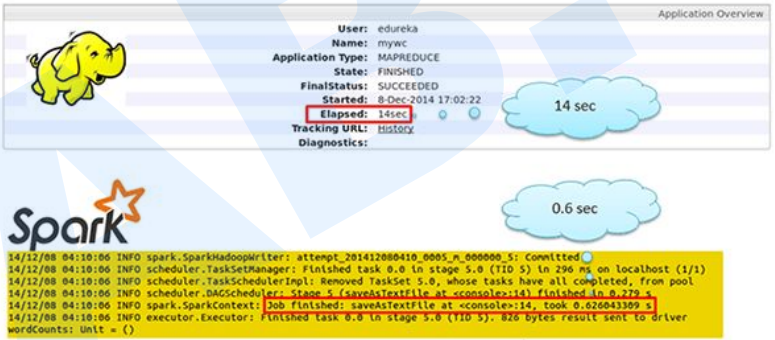
Real-time Analytics

- Hadoop works on batch processing.
- Slow processing speed with mapreduce on large data sets.
- Hadoop is not so efficient for iterative processing, as Hadoop does not support cyclic data flow
- Hadoop is not efficient for caching. In Hadoop, MapReduce cannot cache the intermediate data in memory for a further requirement which diminishes the performance of Hadoop.

When Not to Use Hadoop

Real-time Analytics

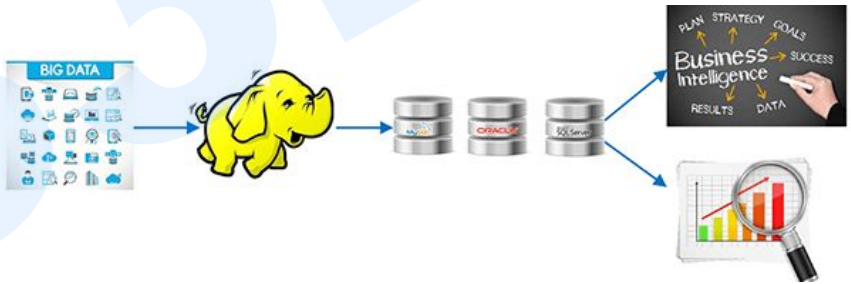
- Solution:
  - Store the big data in HDFS
  - Mount Spark (in-memory processing data) over HDFS (100x faster than mapreduce)
  - Or even using FLINK, it processes faster than spark.



When Not to Use Hadoop

Not a Replacement for Existing Infrastructure

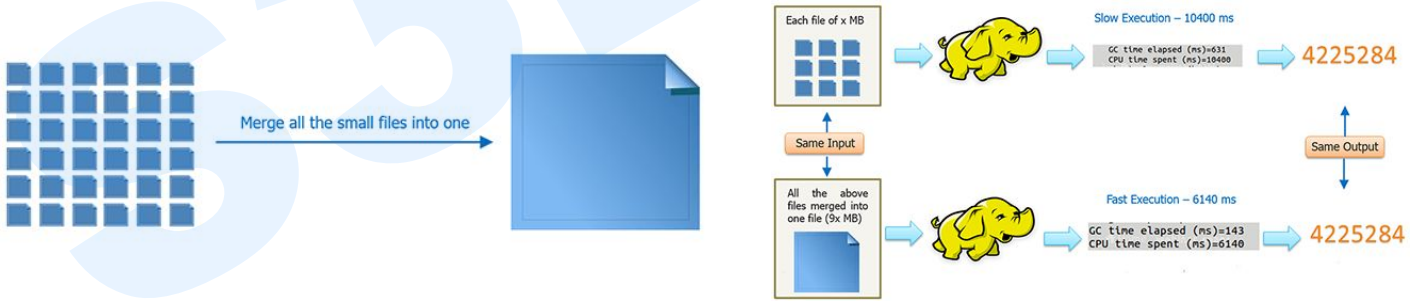
- Hadoop is not a replacement for your existing data processing infrastructure. However, you can use Hadoop along with it.
- Solution:
  - Hadoop is not going to replace your database, but your database isn't likely to replace Hadoop either.
  - Different tools for different jobs, as simple as that.



When Not to Use Hadoop

Multiple Smaller Datasets

- Hadoop is not recommended for small-structured datasets, small files (< default block size: 128MB). For a small data analytics, Hadoop can be costlier than other tools.
- Solution:
  - Merge the small file to create bigger files and then copy bigger files to HDFS



When Not to Use Hadoop

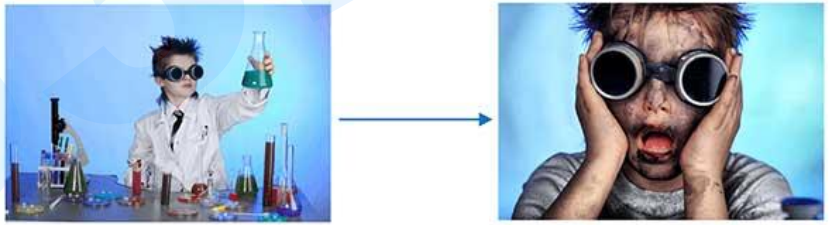
Multiple Smaller Datasets

- Solution:
  - Using HAR files (Hadoop archives)
  - Sequence files: we use filename -> key, file contents -> value. Put them into a single sequence file with a program for files and process them in a streaming fashion operating on the sequence file. MapReduce can break the sequence file into chunks and operate on each chunk independently because the sequence file is splittable.
  - Storing files in HBase. We are not actually storing millions of small files into HBase, rather adding the binary content of the file to a cell.

When Not to Use Hadoop

Novice Hadoopers

- Hadoop is a technology which should come with a disclaimer: “Handle with care”. You should know it before you use it or else you will end up like the kid below.
- Not easy to use, does not have any type of abstraction, mapreduce has no interactive mode.



When Not to Use Hadoop

Novice Hadoopers

- Solutions:
  - Using Spark with RDD (Resilient Distributed DataSet) abstraction for the batch or Flink has dataset abstraction.
  - Spark has interactive mode, Flink has high-level operators.

When Not to Use Hadoop

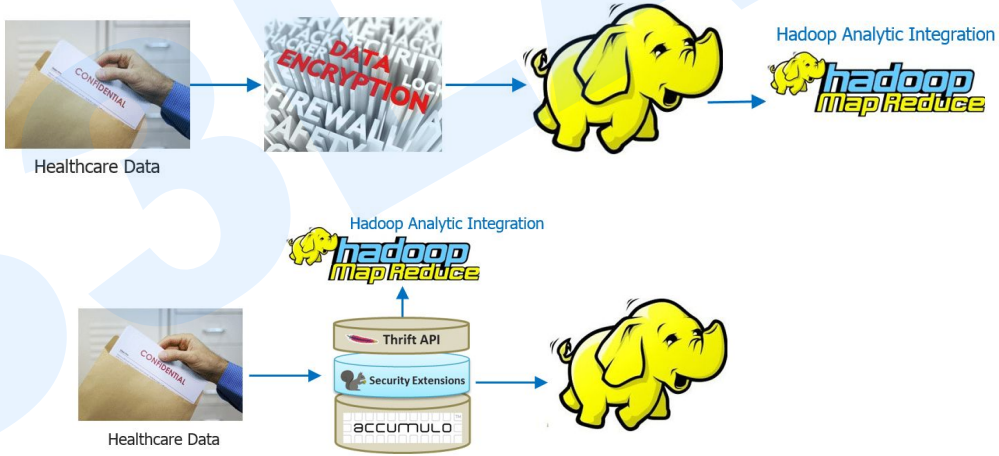
Where Security is the primary Concern?

- Many enterprises — especially within highly regulated industries dealing with sensitive data — aren't able to move as quickly as they would like towards implementing Big Data projects and Hadoop.
- Hadoop is missing encryption.
- Supports Kerberos authentication, which is hard to manage.

When Not to Use Hadoop

Security

- Solution:
  - Using Spark, even spark also can use HDFS ACLs and file-level permissions in HDFS.
  - Spark can run on yarn -> Kerberos authentication



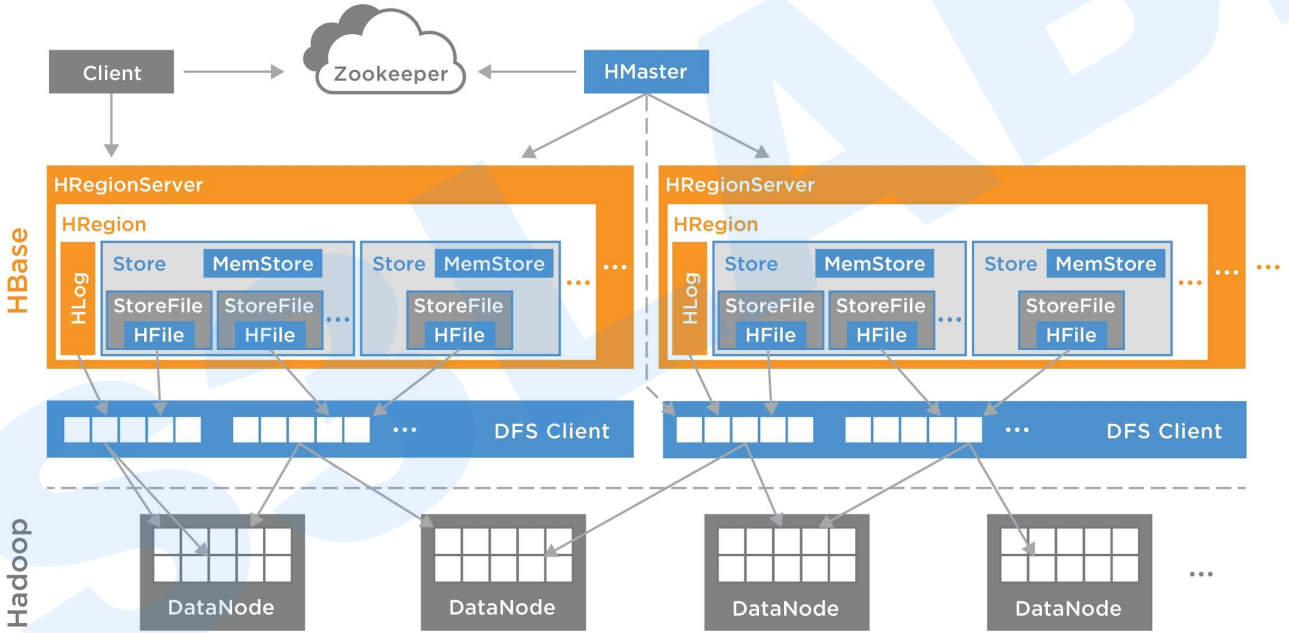
NoSQL Database

HBase

- HBase is an open source, non-relational, distributed database modeled after Google's BigTable.
- It runs on top of Hadoop and HDFS, providing BigTable-like capabilities for Hadoop.

NoSQL Database

HBase



NoSQL Database

HBase *- Features*

- Type of NoSql database (column oriented)
- Strongly consistent read and write
- Automatic sharding
- Automatic RegionServer failover
- Hadoop / HDFS Integration

NoSQL Database

HBase *- Features*

- HBase supports massively parallelized processing via MapReduce for using HBase as both source and sink.
- HBase supports an easy to use Java API for programmatic access.
- HBase also supports Thrift and REST for non-Java front-ends.

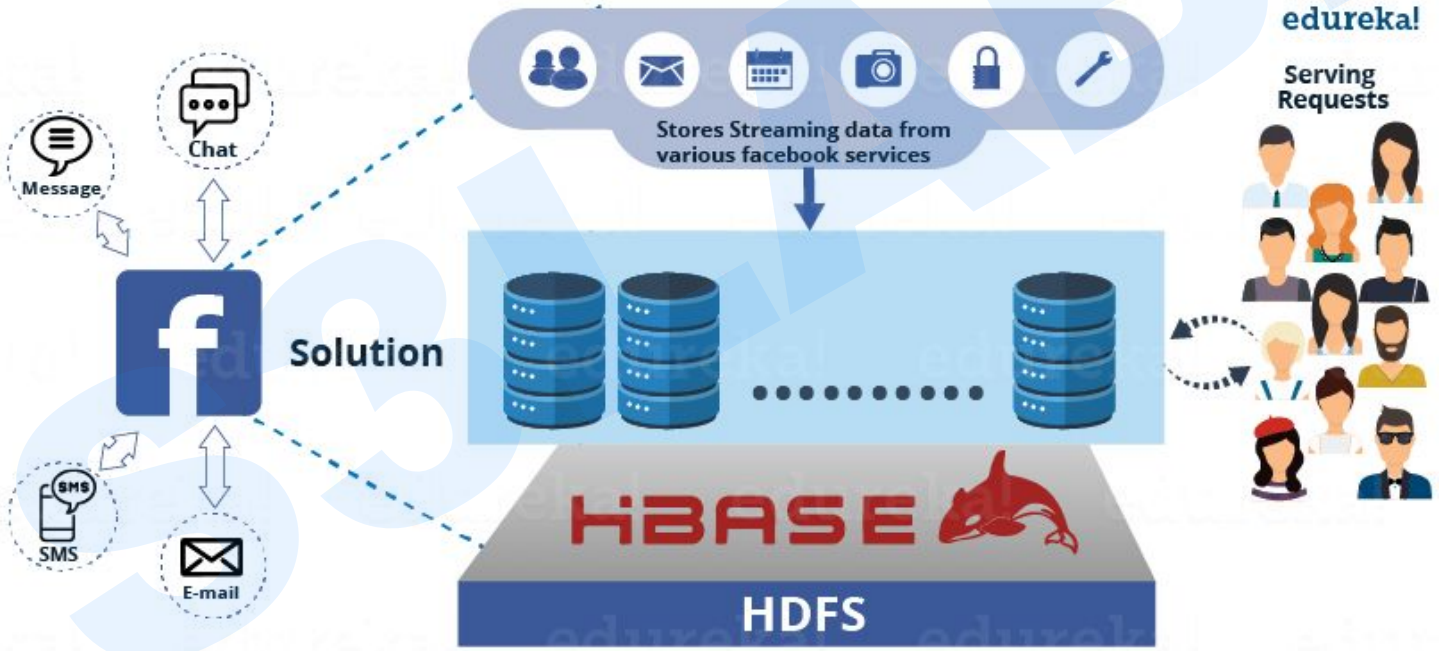
NoSQL Database

***HBase** - When to use*

- When there is real big data: millions or billions of rows, in other way data can not store in a single node.
- When random read/write access to big data
- When require to do thousands of operations on big data
- When there is no need of extra features of RDMS like typed columns, secondary indexes, transactions, advanced query languages, etc.
- When there is enough hardware.

NoSQL Database

HBase - *When to use*



NoSQL Database

*Difference between **Hbase** and HDFS*

HDFS	Hbase
Good for storing large file	Built on top of HDFS. Good for hosting very large tables like billions of rows X millions of column
Write once. Append to files in some of recent versions but not commonly used	Read/write many
No random read/write	Random read/write
No individual record lookup rather read all data	Fast records lookup(update)

NoSQL Database

*Hoya: **HBase** on Yarn*

- Create on-demand HBase clusters
- Configure different HBase instances differently
- Better isolation
- Create (transient) HBase clusters from MapReduce jobs
- Elasticity of clusters for analytic / batch workload processing
- Better cluster resources utilization

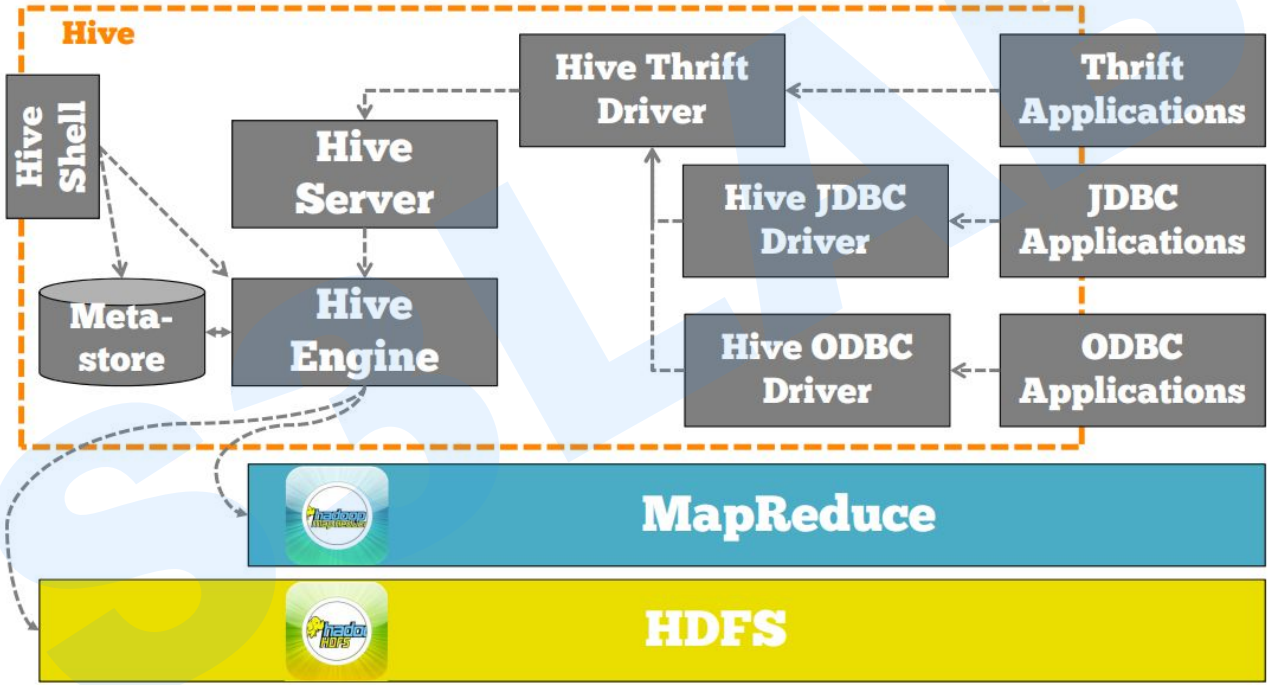
High-Level Data Process Components

Hive

- An sql like interface to Hadoop.
- Data warehouse infrastructure built on top of Hadoop
- Provide data summarization, query and analysis
- Query execution via MapReduce
- Hive interpreter convert the query to Map-reduce format.
- Open source project.
- Developed by Facebook
- Also used by Netflix, Cnet, Digg, eHarmony etc.

High-Level Data Process Components

Hive *- architecture*



High-Level Data Process Components

Hive

- HiveQL example:

  SELECT customerId, max(total_cost) from hive_purchases GROUP BY customerId HAVING count(*) > 3;

High-Level Data Process Components

Pig

- A scripting platform for processing and analyzing large data sets
- Apache Pig allows to write complex MapReduce programs using a simple scripting language.
- Made of two components:
  - High level language: Pig Latin (data flow language).
  - Pig translate Pig Latin script into MapReduce to execute within Hadoop.
- Open source project
- Developed by Yahoo

High-Level Data Process Components

Pig

- Pig Latin example:

```
A = LOAD 'student' USING PigStorage() AS (name:chararray, age:int,
gpa:float);

X = FOREACH A GENERATE name,$2;

DUMP X;
```

High-Level Data Process Components

Hive** & **Pig

- Both requires compiler to generate Mapreduce jobs
- Hence high latency queries when used for real time responses to ad-hoc queries
- Both are good for batch processing and ETL jobs
- Fault tolerant

High-Level Data Process Components

Impala

- Cloudera Impala is a query engine that runs on Apache Hadoop.
- Similar to HiveQL.
- Does not use Map-reduce
- Optimized for low latency queries
- Open source apache project
- Developed by Cloudera
- Much faster than Hive or pig

High-Level Data Process Components

***Impala**, Pig and Hive*

Description of Feature	Pig	Hive	Impala
SQL based query language	No	yes	yes
Schema	optional	required	required
Process data with external scripts	yes	yes	no
Extensible file format support	yes	yes	no
Query speed	slow	slow	fast
Accessible via ODBC/JDBC	no	yes	yes

Data transfer components

Sqoop *(Sql-to-hadoop)*

- Command-line interface for transforming data between RDBMS & Hadoop
- Parallelized data transfer with MapReduce
- Support incremental imports
- Imports use to populate tables in Hadoop
- Exports use to put data from Hadoop into relational database
- Sqoop2 -> Sqoop-as-a-Service: server-based implementation of Sqoop

Data transfer components

Sqoop - *How it works*

- The dataset being transferred is broken into small blocks.
- Map only job is launched.
- Individual mapper is responsible for transferring a block of the dataset.

Data transfer components

Sqoop - *How it works*

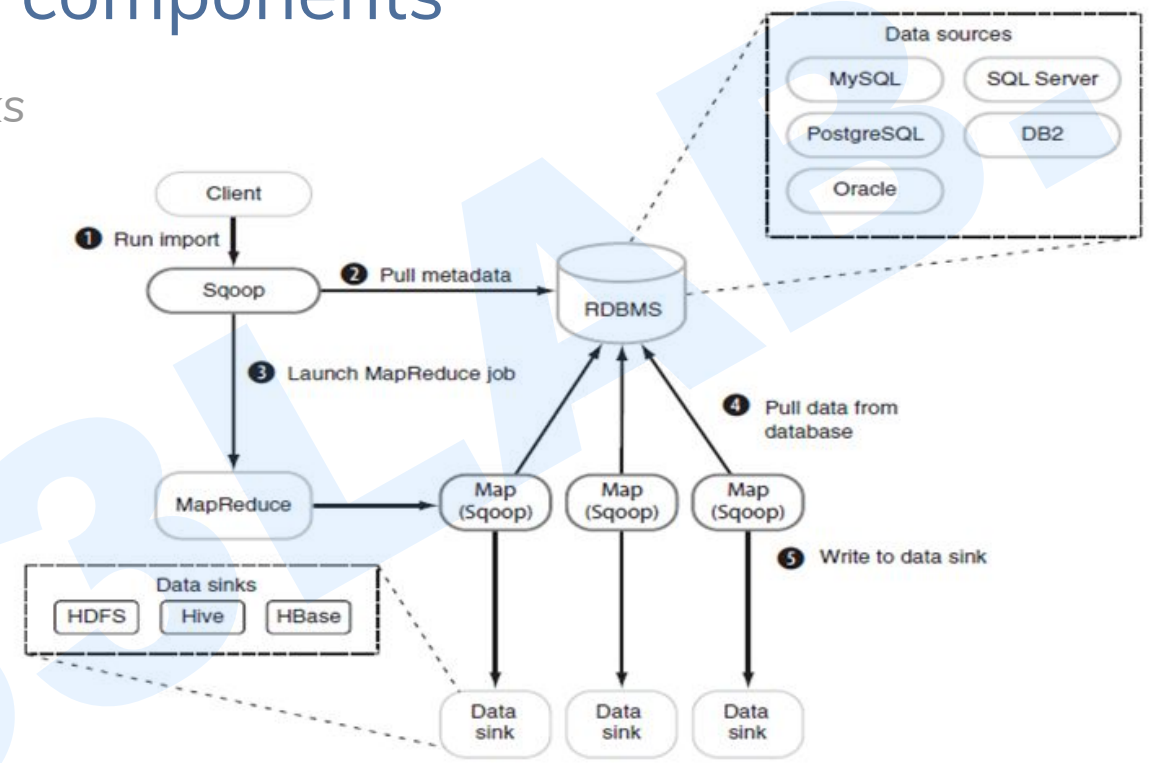


Figure 2.20 **Five-stage Sqoop import overview: connecting to the data source and using MapReduce to write to a data sink**

Data transfer components

Flume

- Apache Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of streaming data (ingest) into the Hadoop Distributed File System (HDFS).

Data transfer components

Flume - *How it works*

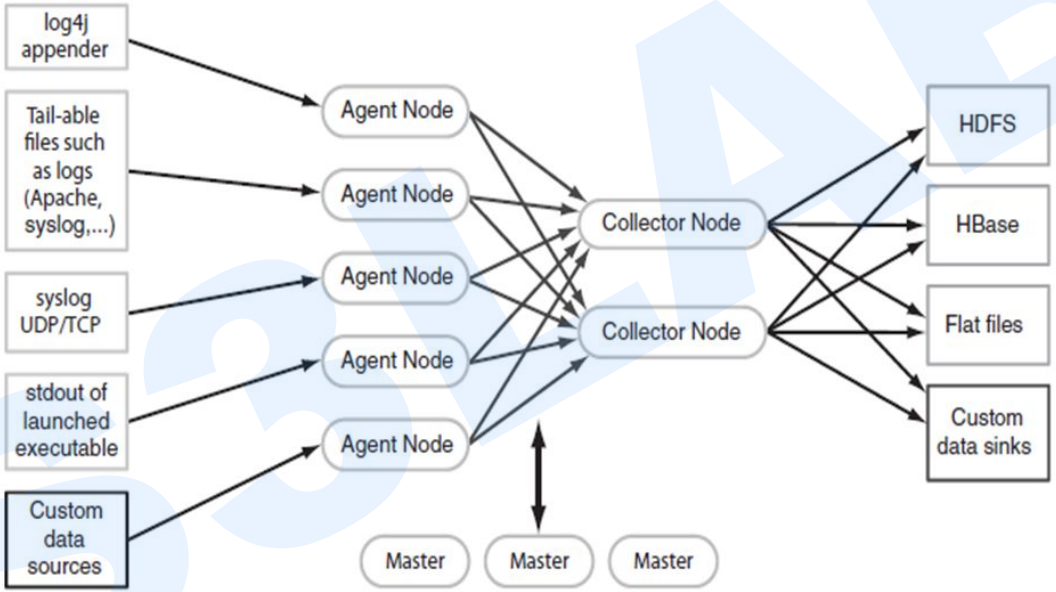


Figure 2.2 **Flume architecture for collecting streaming data**

Data transfer components

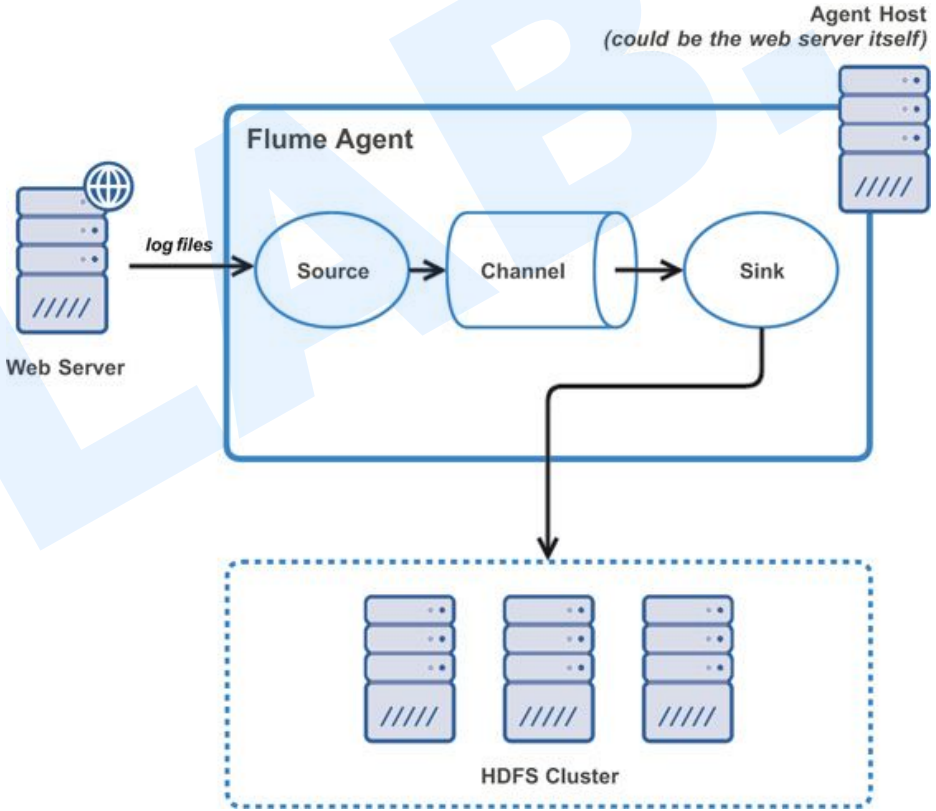
Flume *- How it works*

- Data flows like
- Agent tier -> Collector tier -> Storage tier
- Agent nodes are typically installed on the machines that generate the logs and are data's initial point of contact with Flume. They forward data to the next tier of collector nodes, which aggregate the separate data flows and forward them to the final storage tier.

Data transfer components

Flume - *Agent architecture*

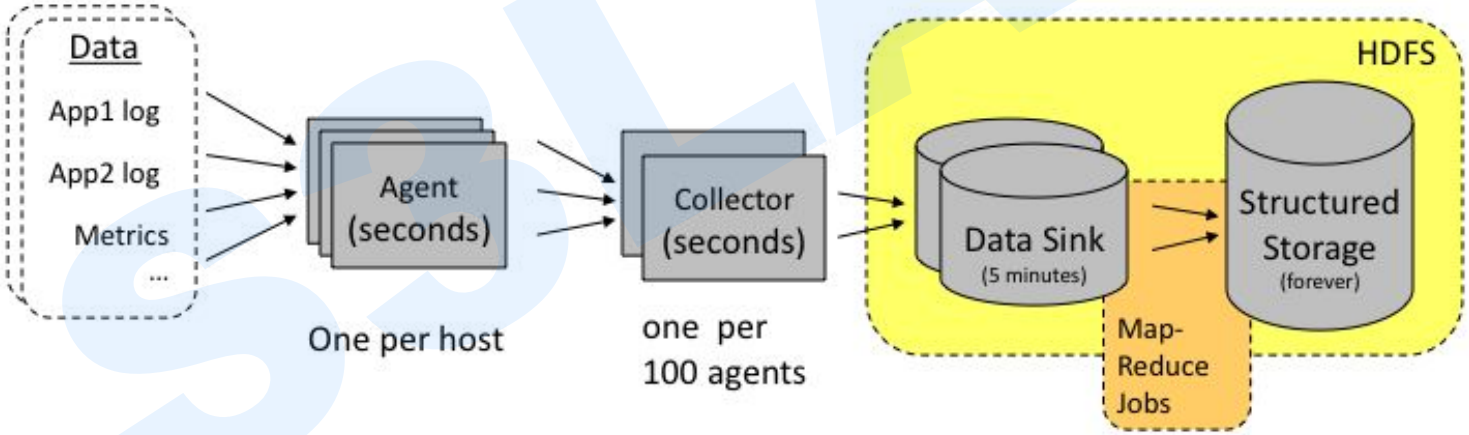
- Sources:
  - HTTP, Syslog, JMS, Kafka, Avro, Twitter - stream api for tweets download, ...
- Sink:
  - HDFS, Hive, HBase, Kafka, Solr, ...
- Channel:
  - File, JDBC, Kafka, ...



Data transfer components

Chukwa

- a scalable system for collecting logs and other monitoring data and processing the data with MapReduce



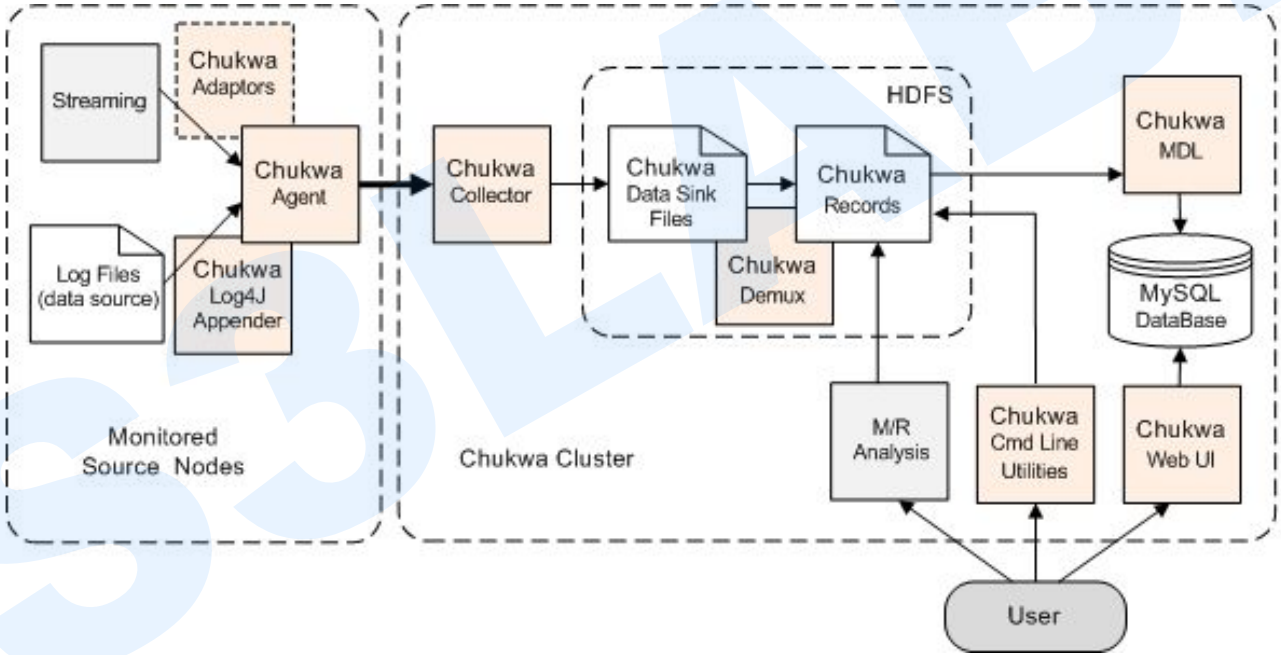
Data transfer components

Chukwa

- **Agents** that run on each machine and emit data.
  - Adaptor: dynamically controllable data sources. Emit data in **Chunks** (a sequence of bytes, with some metadata).
- **Collectors** that receive data from the agent and write it to stable storage.
- **MapReduce jobs** for parsing and archiving the data.
- **HICC**, the Hadoop Infrastructure Care Center; a web-portal style interface for displaying data.

Data transfer components

Chukwa - *HICC*



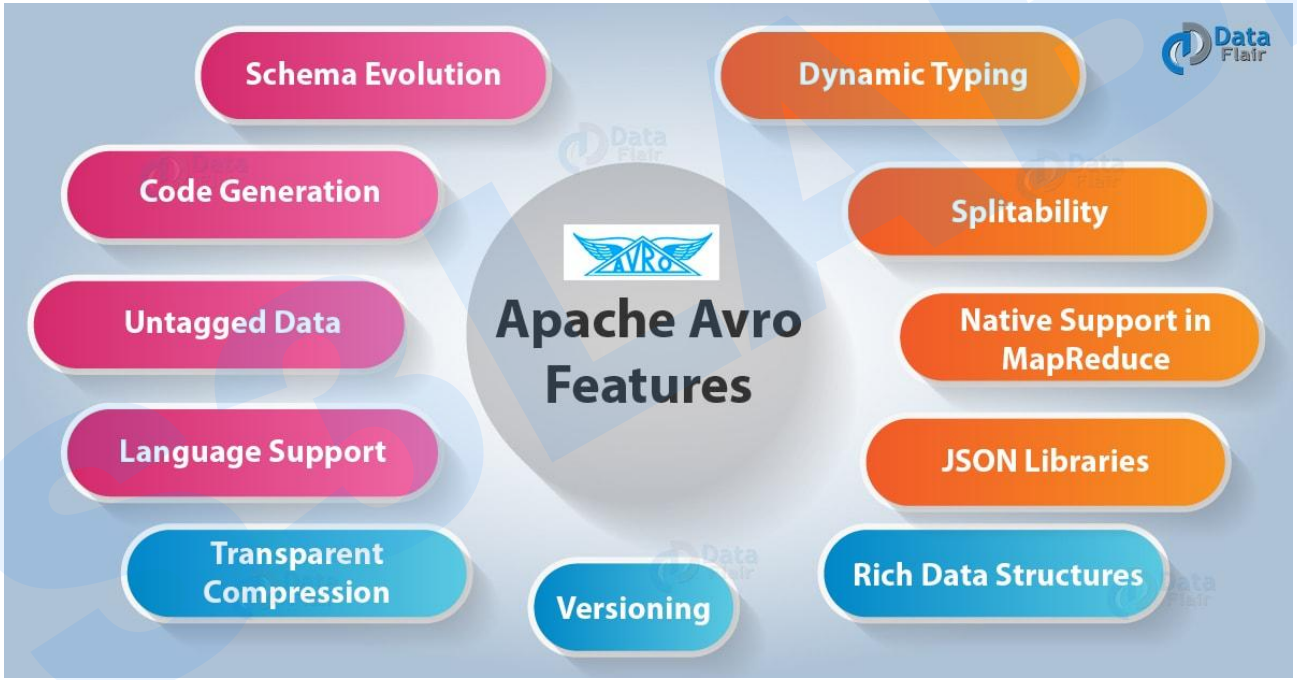
Data Serialization Components

AVRO

- A language-neutral data serialization system.
- Avro uses JSON based schemas
- Uses RPC calls to send data
- During the data exchange, schema’s sent.

Data Serialization Components

Avro - *features*



Data Serialization Components

***Avro** - General working*

- **Step 1 –** Create schemas. Here you need to design Avro schema according to your data.
- **Step 2 – Read the schemas into your program. It is done in two ways –**
  - **By Generating a Class Corresponding to Schema** – Compile the schema using Avro. This generates a class file corresponding to the schema
  - **By Using Parsers Library** – You can directly read the schema using parsers library.

Data Serialization Components

Avro - *General working*

- **Step 3 –** Serialize the data using the serialization API provided for Avro, which is found in the **package org.apache.avro.specific.**
- **Step 4 –** Deserialize the data using deserialization API provided for Avro, which is found in the **package org.apache.avro.specific.**

Data Serialization Components

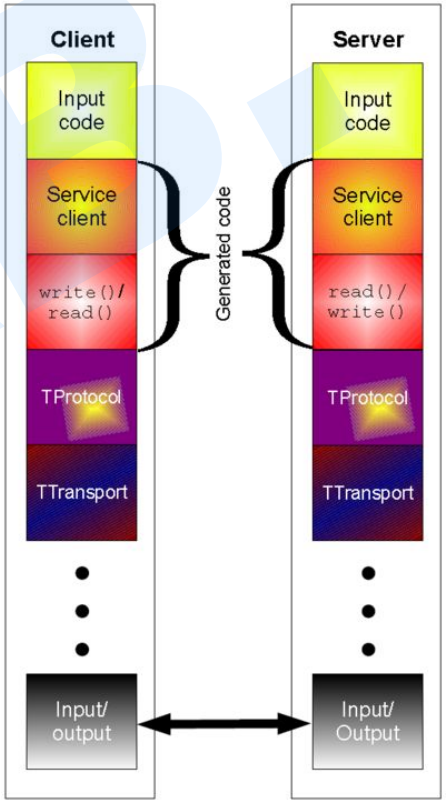
Thrift

- Define data types and service interface with IDL (interface definition language)
- The IDL content are stored as *.thrift*.
- Thrift provides clean abstractions for data transport, data serialization, and application level processing.

Data Serialization Components

Thrift

- Apache Thrift is a set of code-generation tools that allows developers to build RPC clients and servers by just defining the data types and service interfaces in a simple definition file. Given this file as an input, code is generated to build RPC clients and servers that communicate seamlessly across programming languages



Data Serialization Components

Thrift

Technologies	Protobuf	Thrift	Avro
Created	2001 (2008)	2007	2009
Creator / Maintainer	Google / Google	Facebook / Apache	Doug cutting / Apache
Schema evolution	Field Tag	Field Tag	Schema
Static/Dynamic	Yes/No	Yes/No	Yes/Yes
Hadoop support	No	No	Yes
RPC	No	Yes	Yes
Used by	Google	Facebook, Cassandra	Hadoop, Liveperson
Lang support	Good	Great	Good

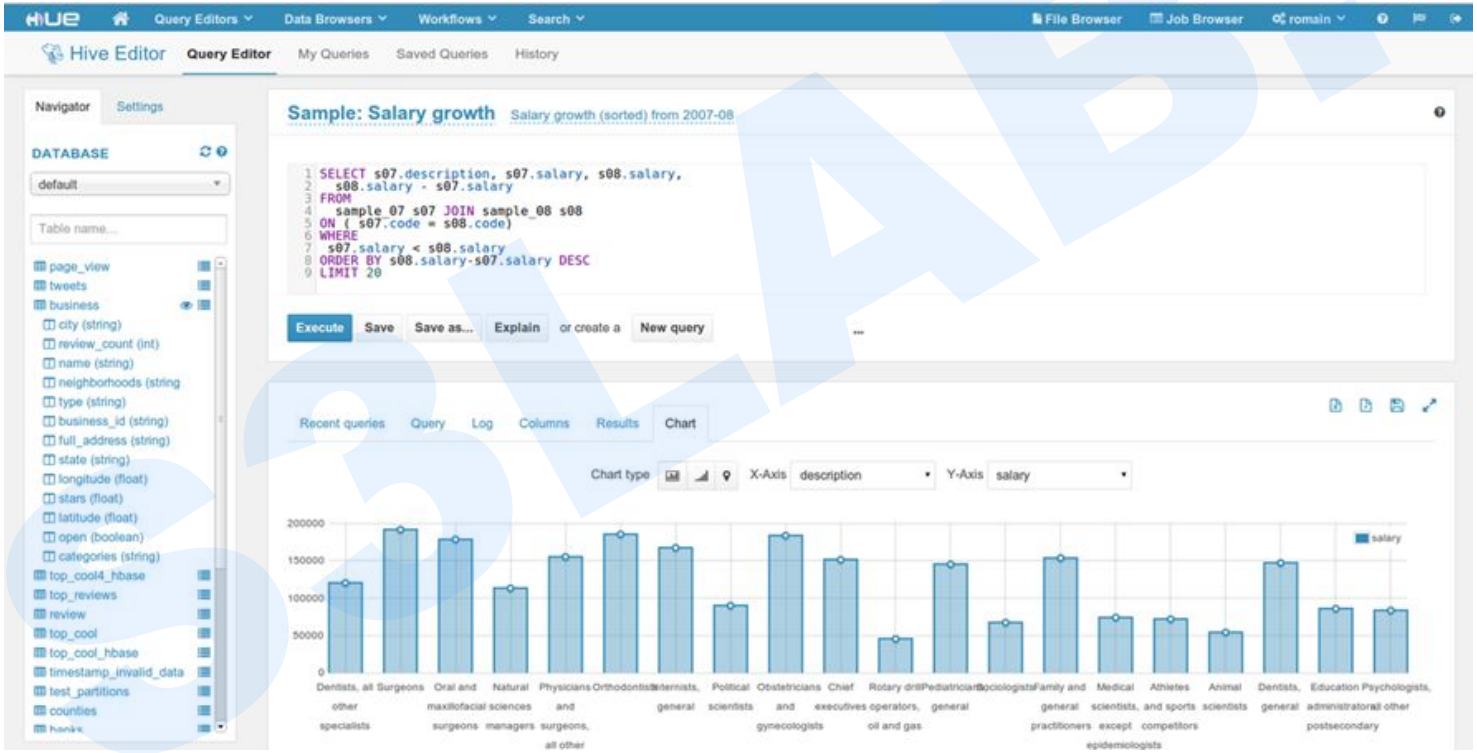
Monitoring Components

Hue

- Graphical front end to the cluster.
- Open source web interface.
- Makes Hadoop platform (HDFS, yarn, Solr, Pig, Map reduce, oozie, Hive, sqoop, impala, etc.) easy to use

Monitoring Components

Hue



Monitoring Components

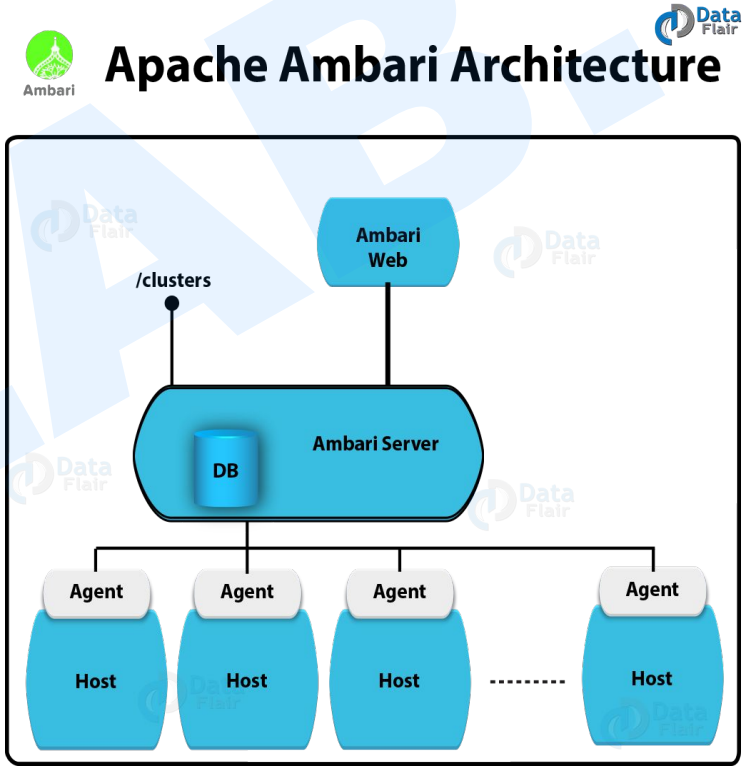
ZooKeeper

- Because coordinating distributed systems is a Zoo.
- ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

Monitoring Components

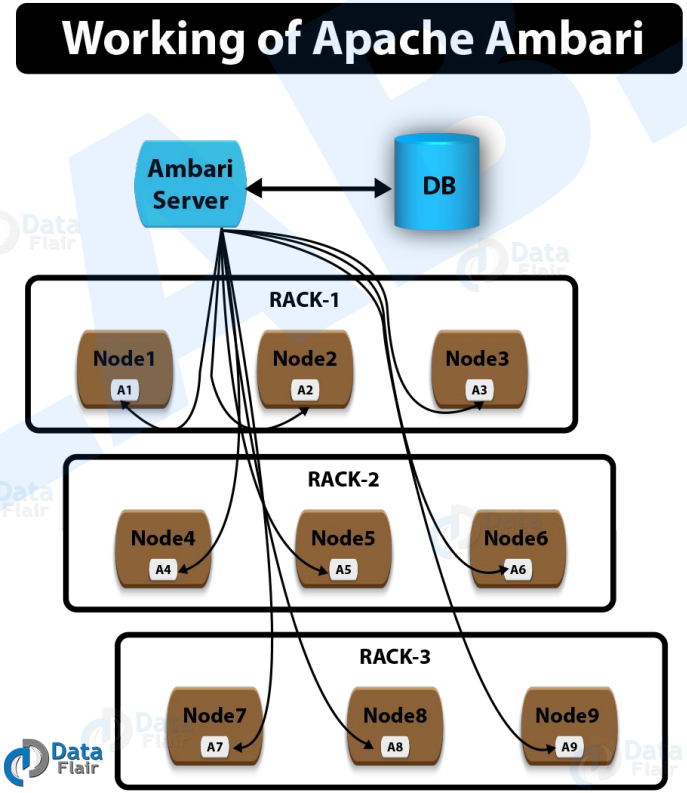
Ambari

- an open source web-based management tool which manages, monitors as well as provisions the health of Hadoop clusters



Monitoring Components

Ambari



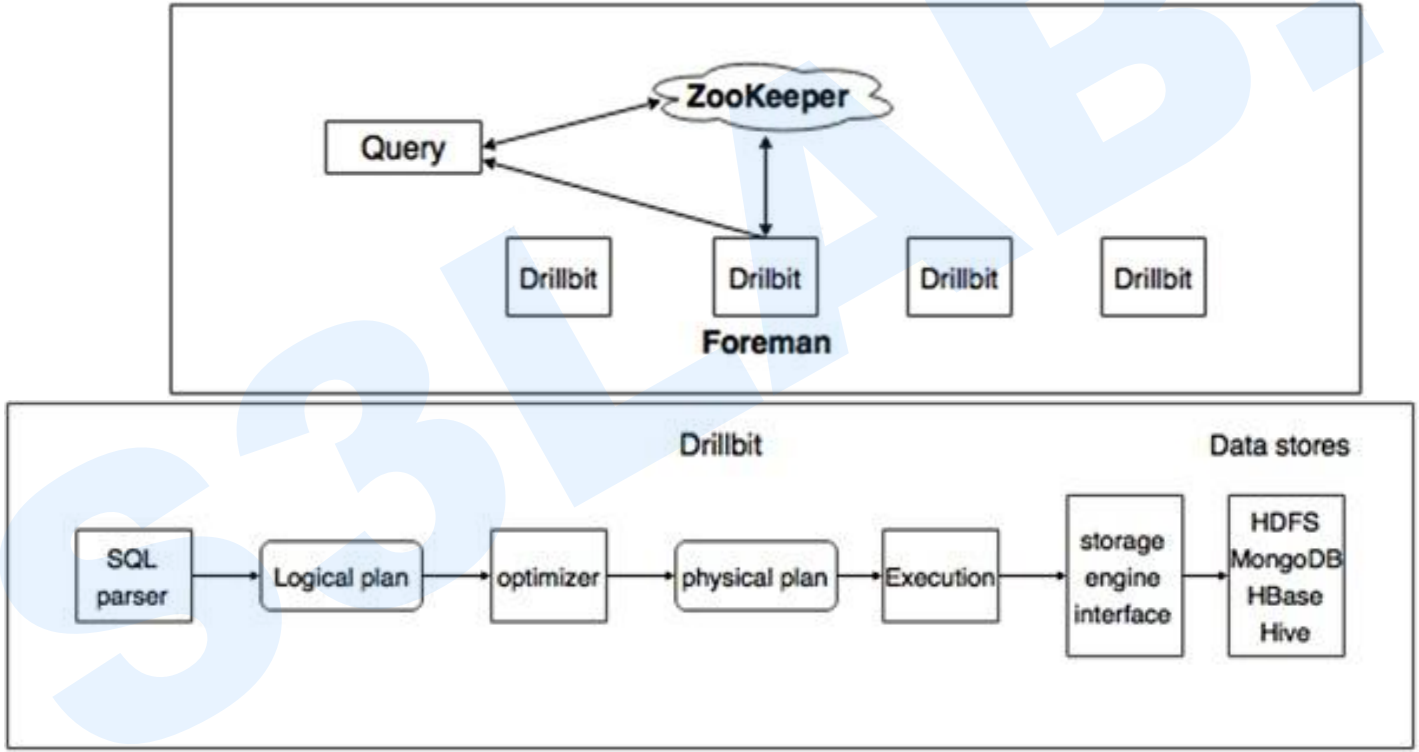
Data Analysis Components

Drill

- A low latency schema-free query engine for big data
- Users can query the data using a standard SQL and BI Tools, which doesn't require to create and manage schemas
- Drill uses a JSON document model internally which allows it to query data of any structure.
- Drill works with a variety of non-relational data stores, including Hadoop, NoSQL databases (MongoDB, HBase), Local files, NAS and cloud storage like Amazon S3, Azure Blob Storage, etc

Data Analysis Components

Drill



Data Analysis Components

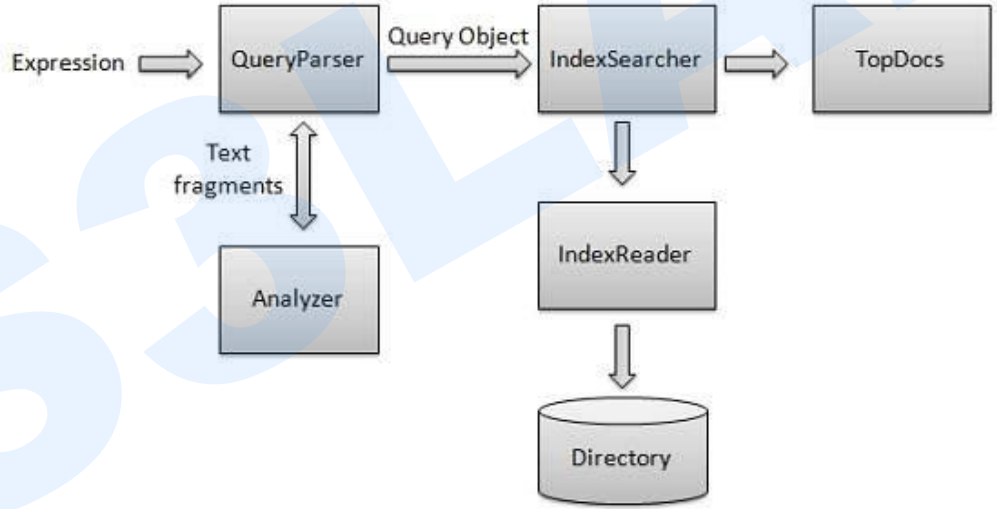
Mahout

- A mahout is one who drives an elephant as its master.
- is an open source project that is primarily used for creating scalable machine learning algorithms, implemented on top of Apache Hadoop® and using the MapReduce paradigm. It implements popular machine learning techniques such as:
  - Recommendation
  - Classification
  - Clustering

Data Analysis Components

Lucene

- Lucene is an open-source Java full-text search library which makes it easy to add search functionality to an application or website.



Searching Process

Management Components

Oozies

- Workflow scheduler to manage hadoop and related jobs
- Developed first in Banglore by Yahoo
- DAG(Direct Acyclic Graph): Acyclic means a graph cannot have any loops and action members of the graph provide control dependency. Control dependency means a second job cannot run until a first action is completed
- Ozzie definitions are written in hadoop process definition language (hPDL) and coded as an xml file (WORKFLOW.XML)

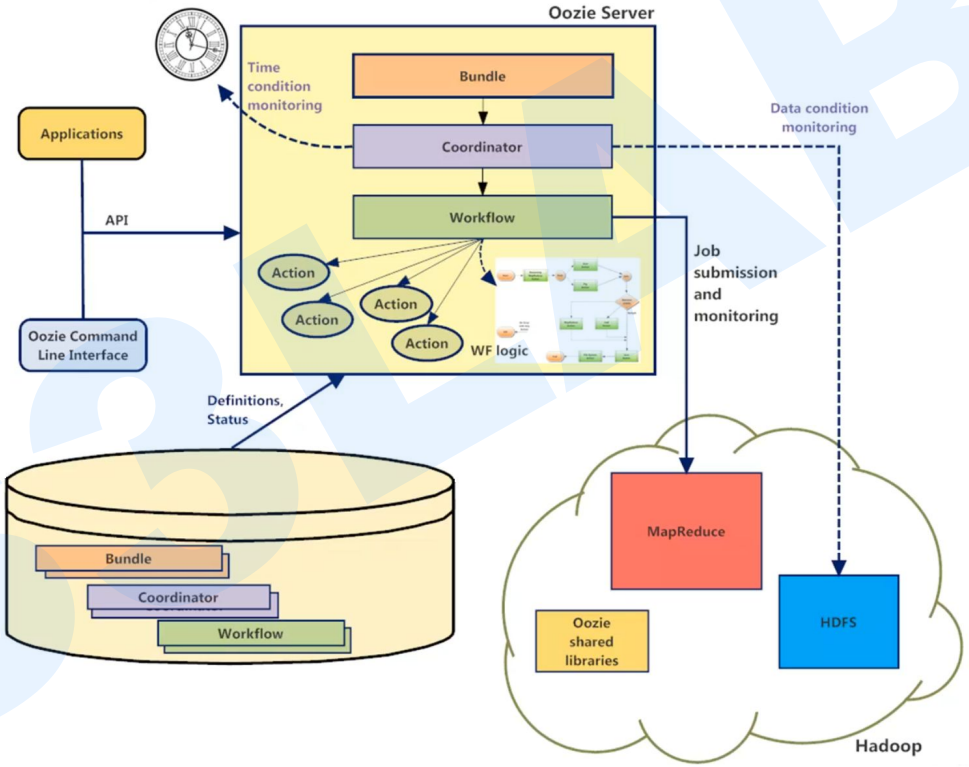
Management Components

Oozies

- Workflow contains:
  - Control flow nodes (defines start, end and execution path of the workflow): START, FORK, JOIN, DECISION, KILL, END
  - Action nodes (trigger execution of tasks): Java MapReduce, Streaming MapReduce, Pig, Hive, Sqoop, FileSystem tasks, Distributed copy, Java programs, Shell scripts, Http, Email, Oozie sub workflows, ...

Management Components

Oozies



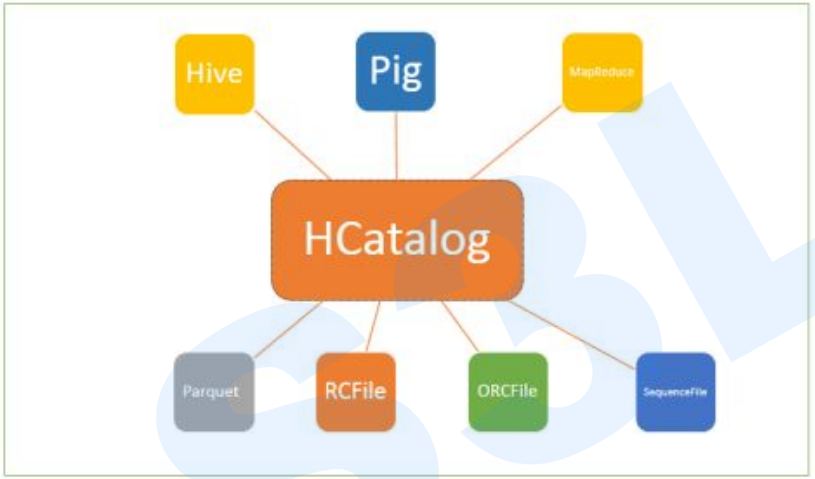
Management Components

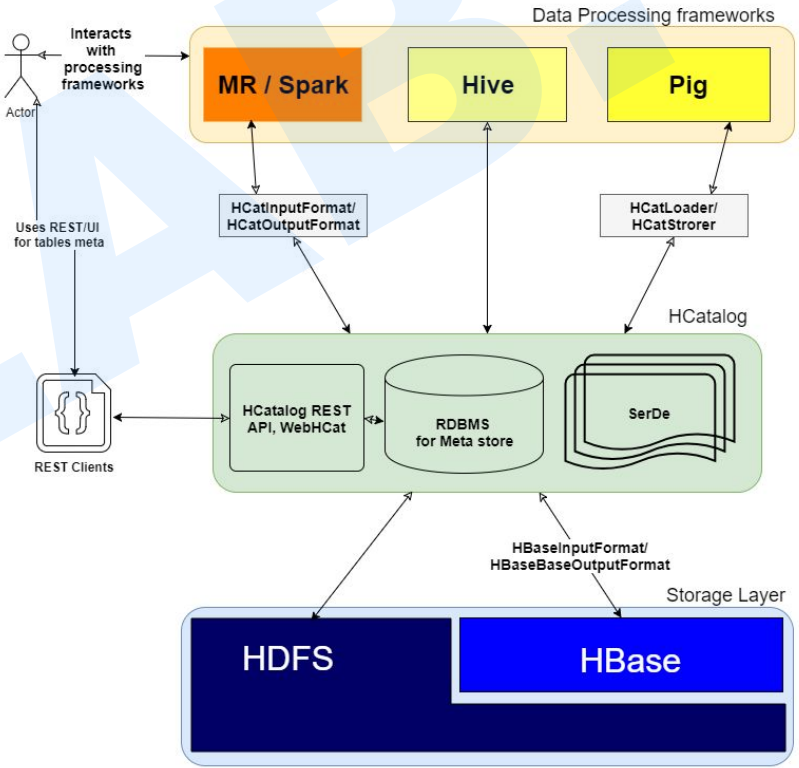
HCatalog

- A table storage management tool for Hadoop.
- It exposes the tabular data of Hive metastore to other Hadoop applications.
- It enables users with different data processing tools (Pig, MapReduce) to easily write data onto a grid. It ensures that users don’t have to worry about where or in what format their data is stored.

Management Components

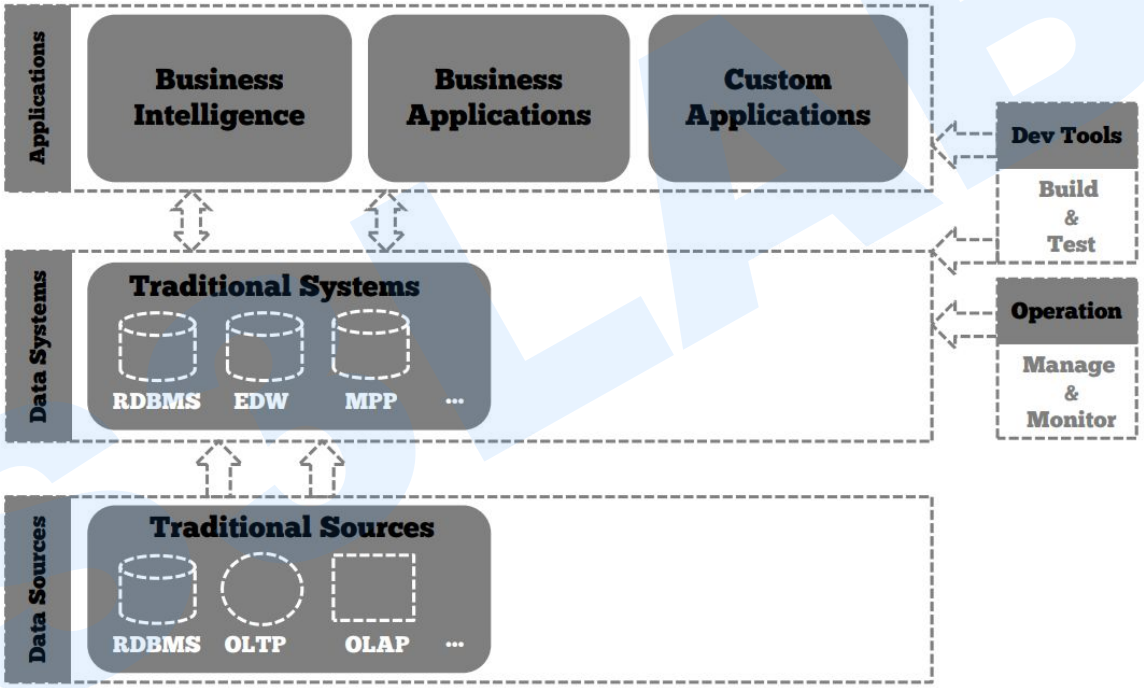
HCatalog





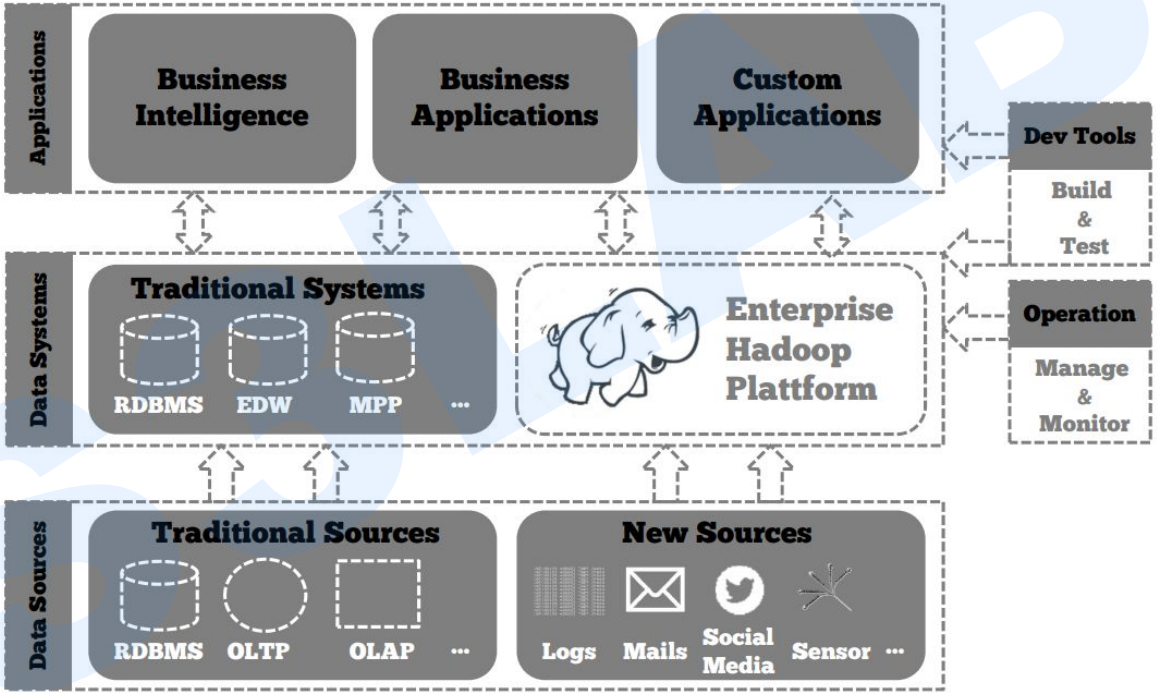
Use case

Classical enterprise platform



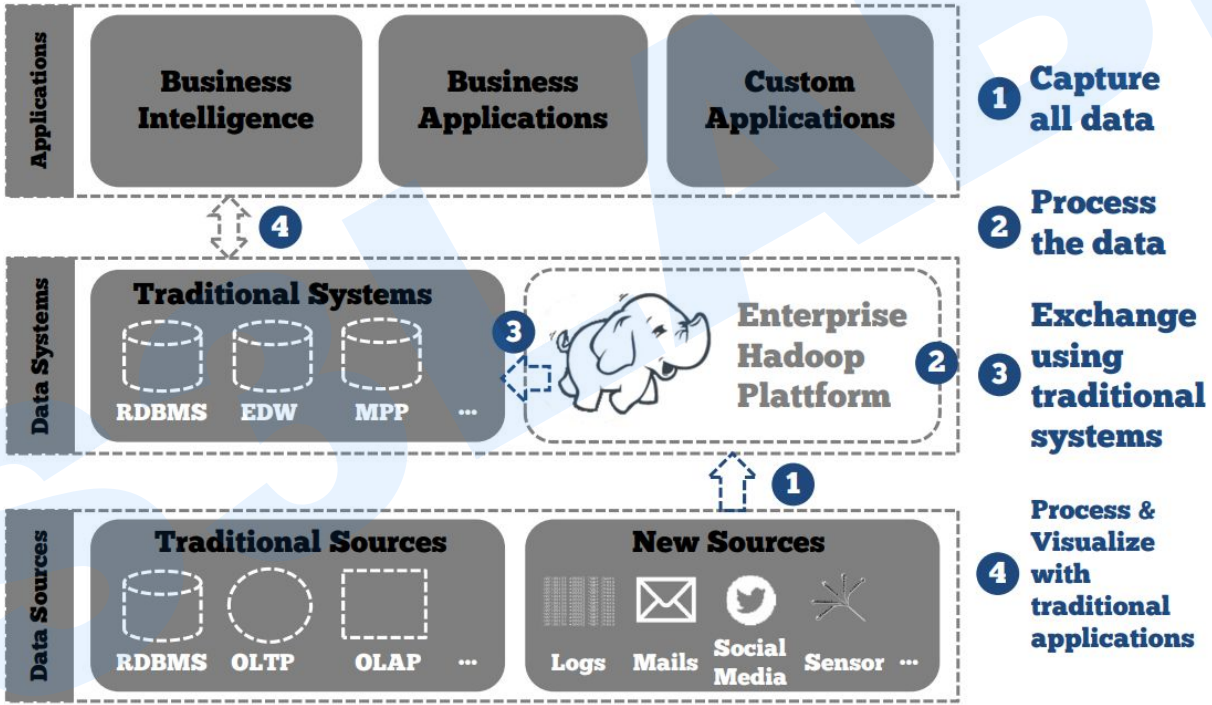
Use case

With big data



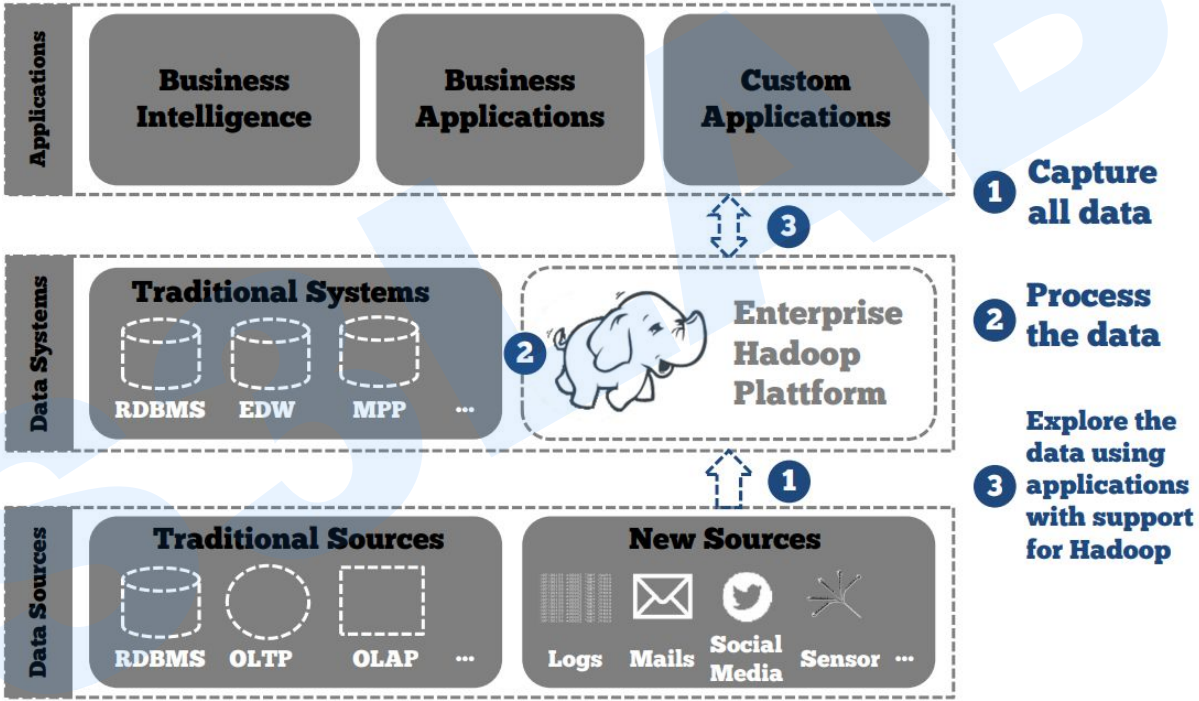
Use case

Refine data



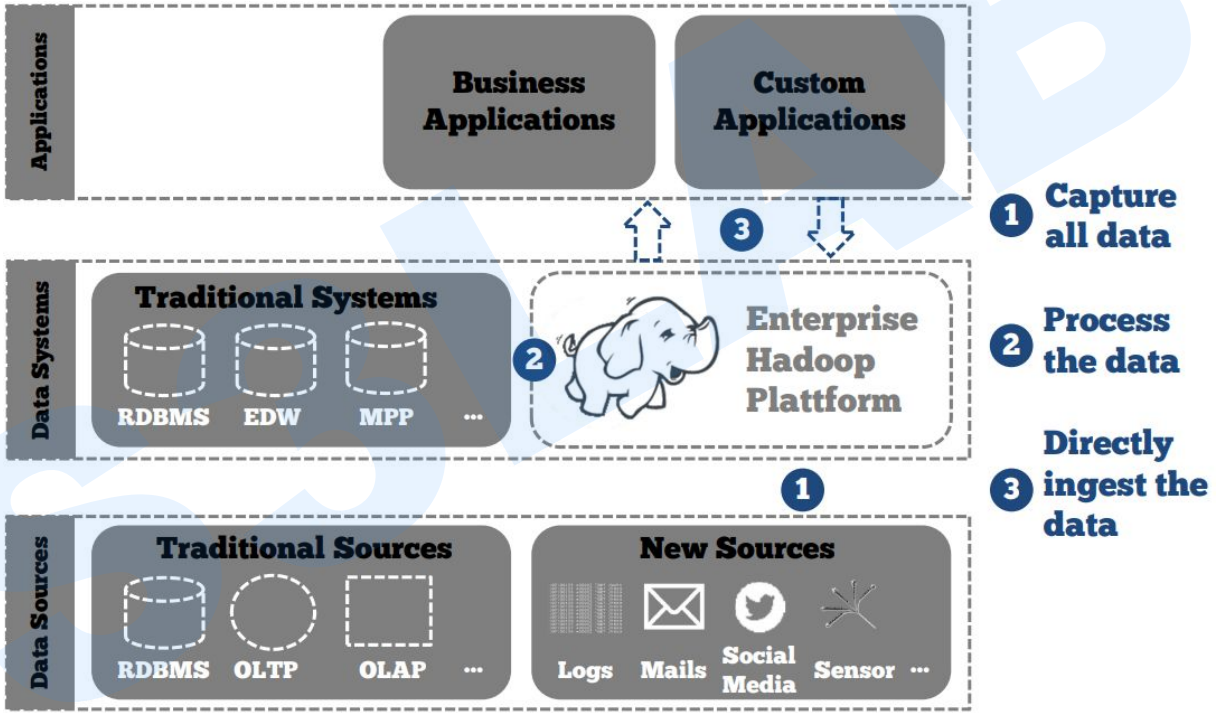
Use case

Explore data



Use case

Enrich data



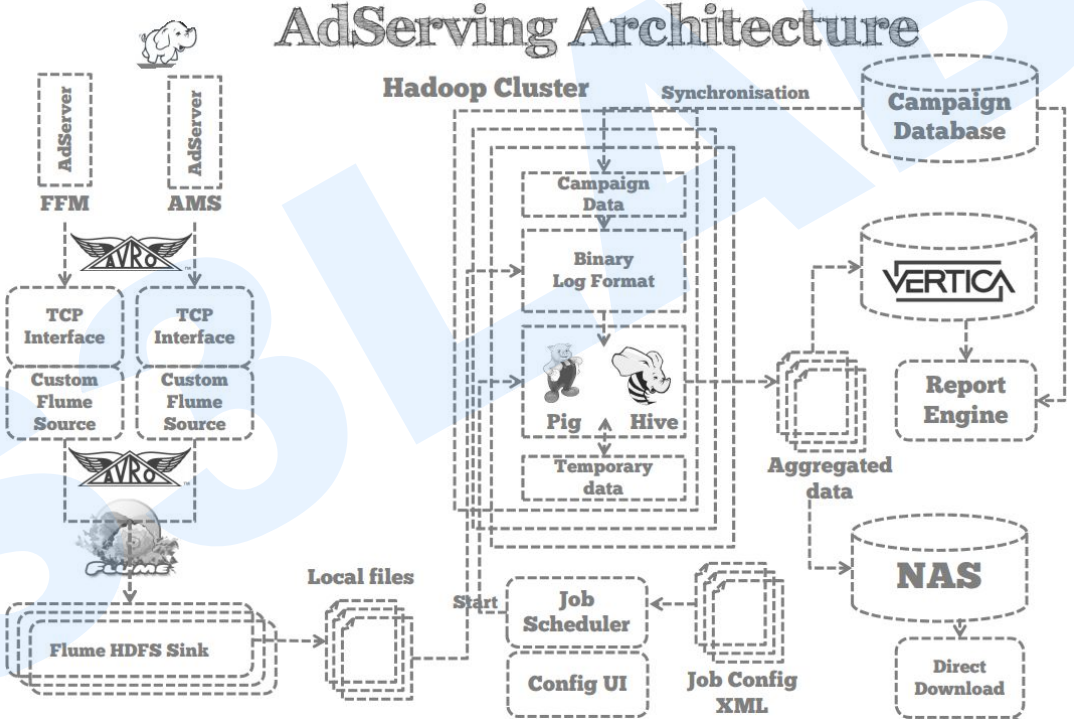
An Example

Digital Advertising

- 6 billion ad deliveries per day
- Reports (and bills) for the advertising companies needed
- Own C++ solution did not scale
- Adding functions was a nightmare

An Example

Digital Advertising



Q & A

Cảm ơn đã theo dõi

Chúng tôi hy vọng cùng nhau đi đến thành công.