

Procesamiento Masivo de datos: Hadoop

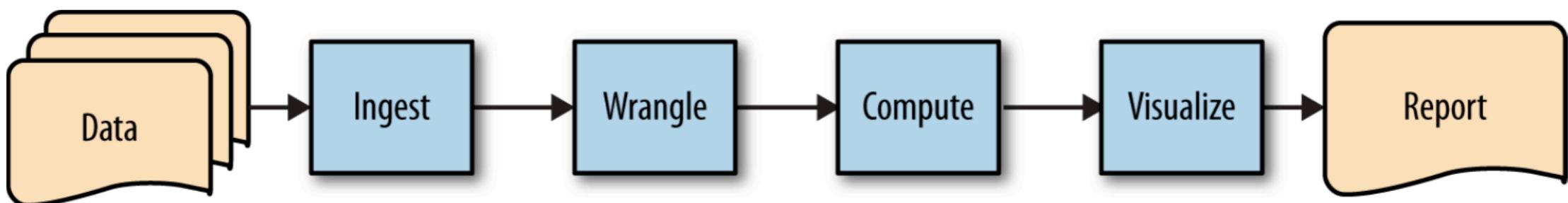
Alex Di Genova

07/10/2024

Hadoop

Introducción

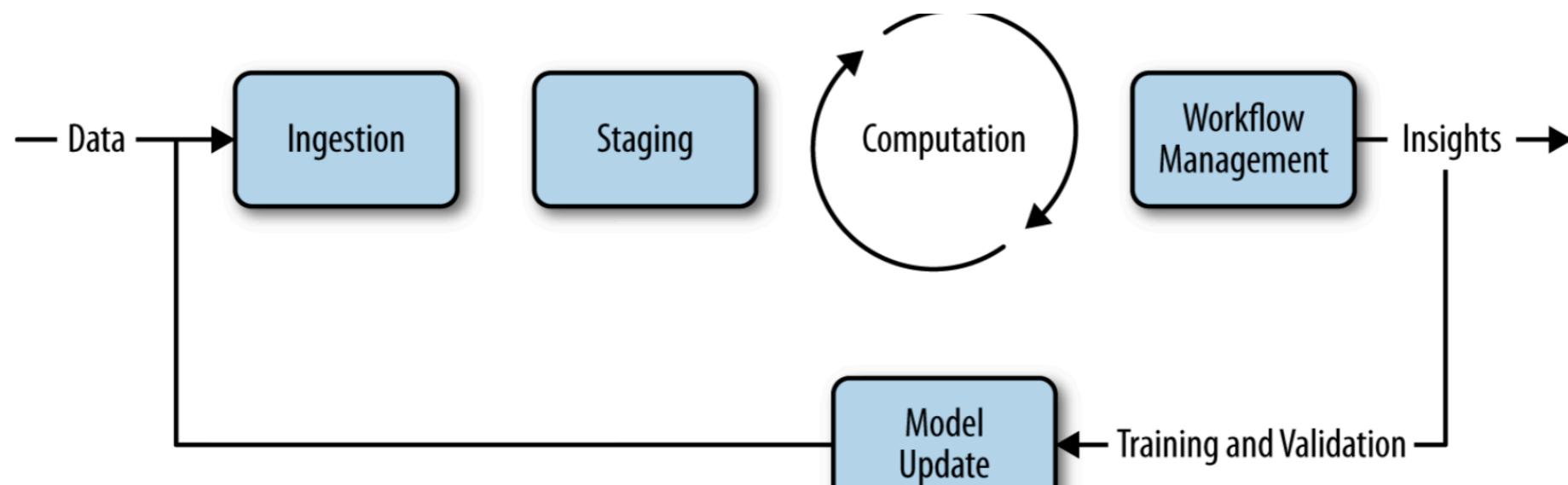
- Data science workflow.
 - 44 zettabytes (2022)
- El software de procesamiento de datos tradicional es inadecuado para manejar grandes cantidades de datos.
- El análisis de datos masivos requiere un software paralelo que se ejecuta en varios servidores.



Hadoop

Big Data workflows

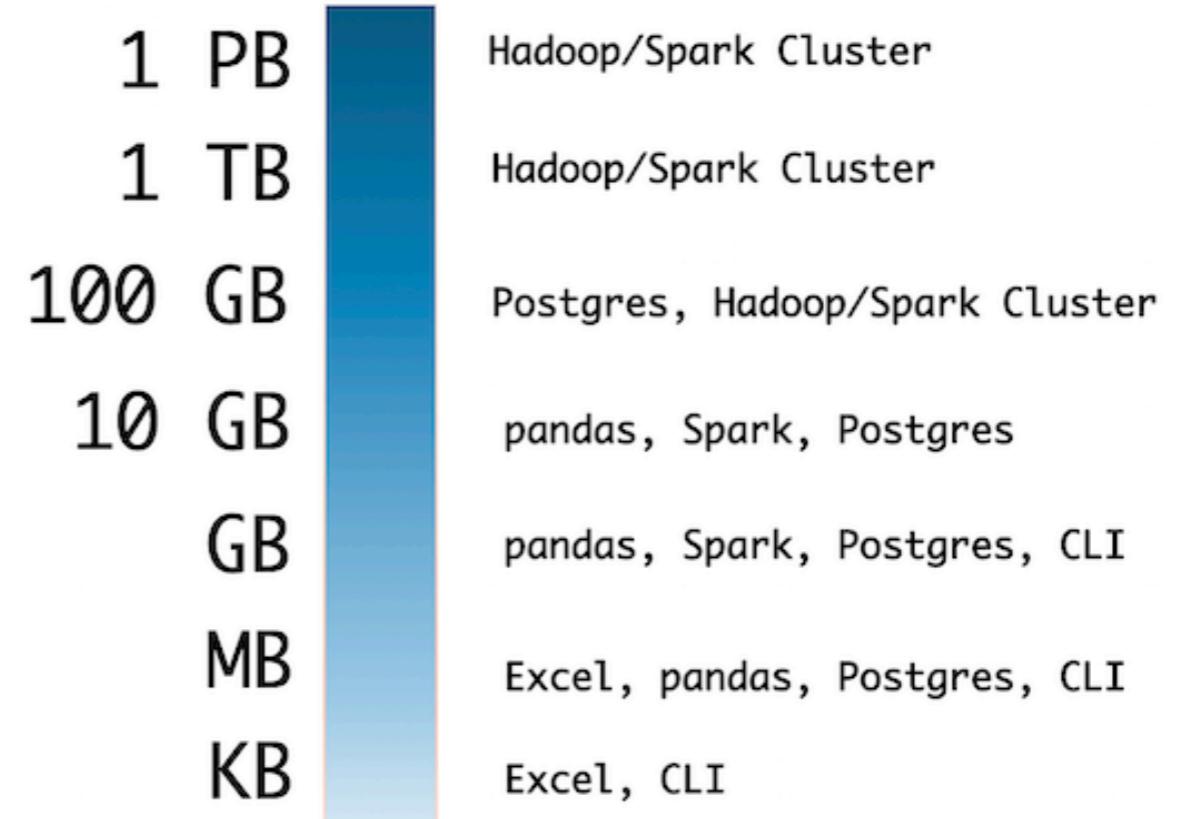
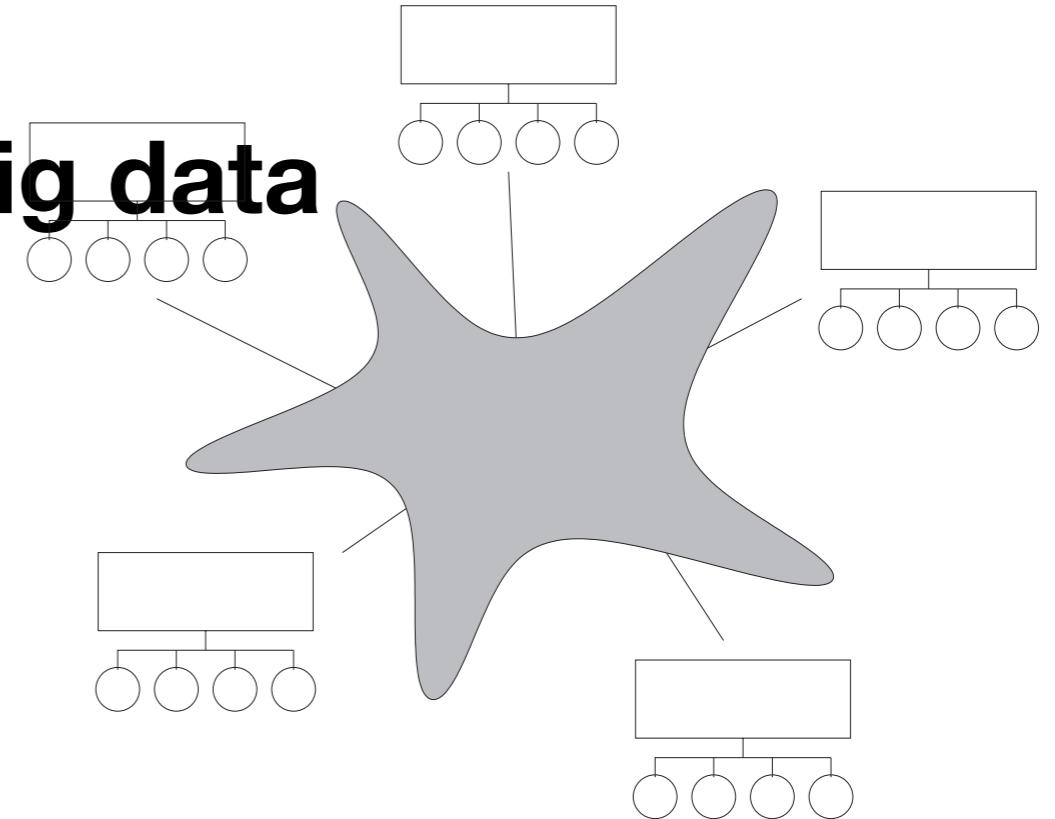
- Escalable y automatico.
- De datos a información.
- Estapas:
 - Fase de consumo
 - Se especifican las ubicaciones de los datos o se anotan datos.
 - Fase de puesta a punto (staging)
 - Se realizan transformacion a los datos para almacenarlos y dejarlos disponibles para procesamiento (normalización y estandarización).
 - Fase de computo
 - Procesar los datos para obtener información.
 - Desarrollando reportes
 - Construyendo modelos para recomendacion, clustering o clasificacion.
 - Fase de control de workflow
 - Realiza la abstraccion, orquesta y automatiza tareas para permitir que los pasos del worflow sean operacionales (script, applicacion, job).



Hadoop

Un sistema operativo para big data

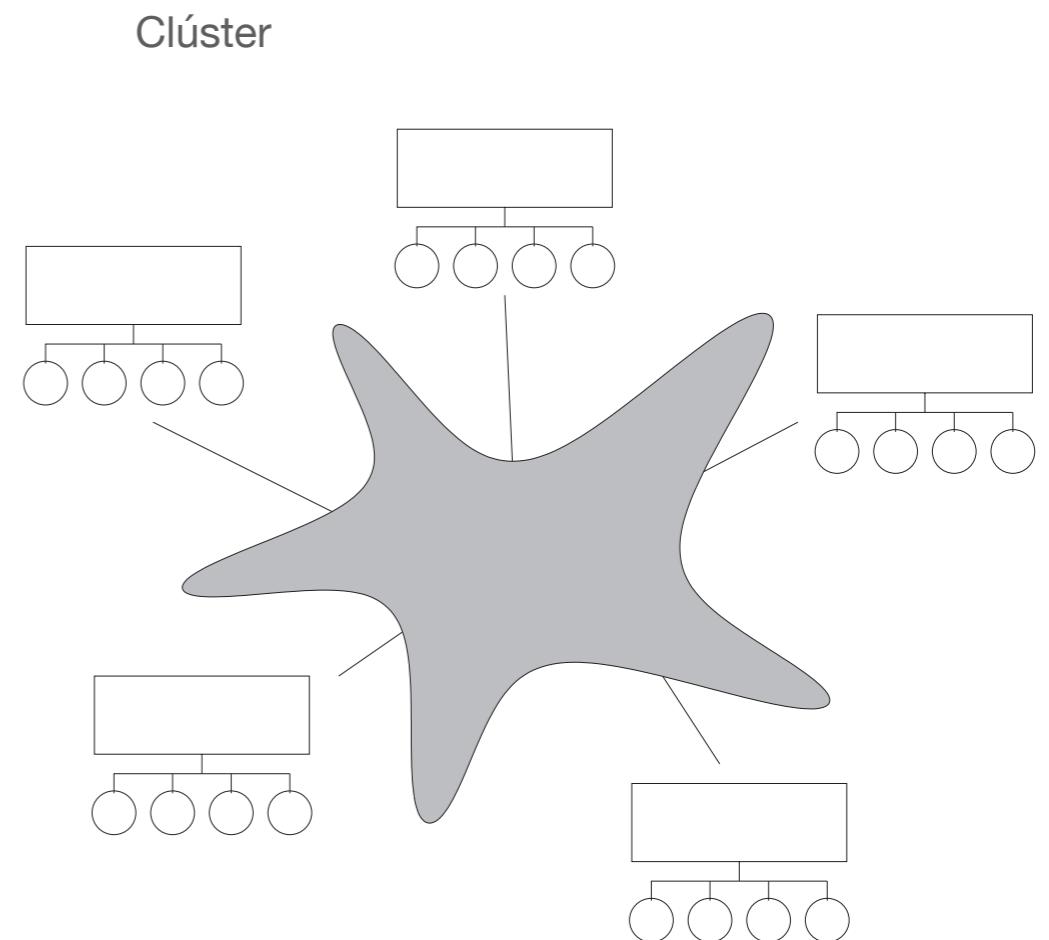
- Hadoop proporciona aplicaciones robustas para operar con datos.
- Hadoop implementa **Map/Reduce**, donde la aplicación se divide en muchos fragmentos pequeños de trabajo, cada uno de los cuales se puede ejecutar o volver a ejecutar en cualquier nodo del clúster.
- Proporciona un sistema de archivos distribuido (HDFS) que almacena datos en los nodos de cómputo,
 - Acceso rápido y eficiente a los archivos.
- Tanto Map/Reduce como HDFS están diseñados para manejar errores en los nodos.
- Hadoop distribuye el cálculo en muchas máquinas que operan simultáneamente en su propia sección de datos.



Hadoop

Un sistema operativo para big data

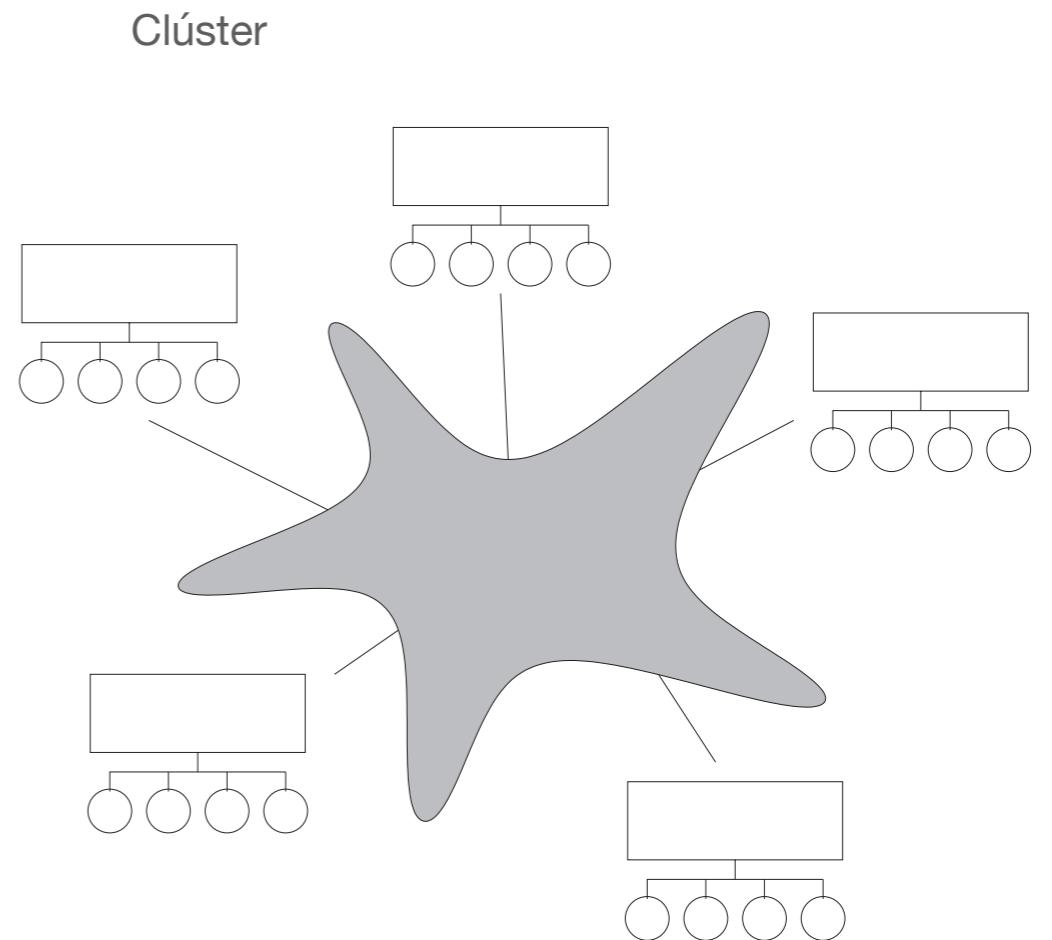
- Los datos se distribuyen inmediatamente cuando se agregan al clúster y se almacenan en varios nodos. Los nodos procesan los datos que se almacenan localmente para minimizar el tráfico en la red.
- un sistema distribuido debe cumplir con los siguientes requisitos:
 - **Tolerante a fallas**
 - Si un componente falla, no debería resultar en la falla de todo el sistema.
 - **Recuperable**
 - En caso de falla, no se deben perder datos.
 - **Consistente**
 - La falla de un trabajo o tarea no debe afectar el resultado final.
 - **Escalable**
 - Agregar carga (más datos, más cómputo) conduce a una disminución en el rendimiento, no a fallas; aumentar los recursos debería resultar en un aumento proporcional de la capacidad procesamiento.



Hadoop

Un sistema operativo para big data

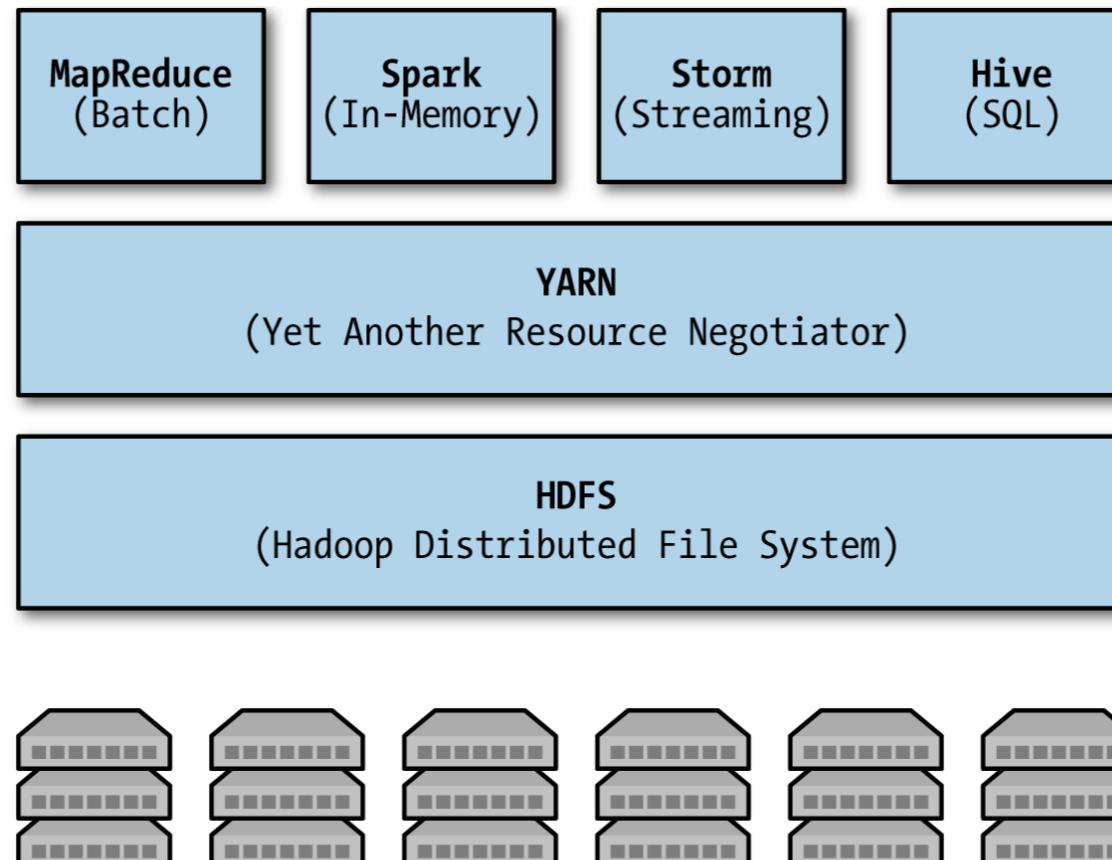
- Los datos se almacenan en bloques de un tamaño fijo (normalmente 128 MB) y cada bloque se duplica varias veces en el sistema para proporcionar redundancia y seguridad de datos.
- Los trabajos se dividen en tareas donde cada nodo individual realiza la tarea en un bloque de datos.
- Los trabajos se escriben a un alto nivel sin preocuparse por la programación de la red, el tiempo o la infraestructura.
 - Hadoop permite centrarse en los datos y la computación en lugar de los detalles de la programación distribuida.
- El sistema debe minimizar de forma transparente la cantidad de tráfico de red entre los nodos.
 - Tareas independientes, evitar “deadlock”.
- Los trabajos son tolerantes a fallas a través de la redundancia de tareas.
- Los programas maestros asignan trabajo a los nodos trabajadores de modo que muchos nodos trabajadores puedan operar en paralelo, cada uno en su propia porción del conjunto de datos.



Hadoop

Arquitectura

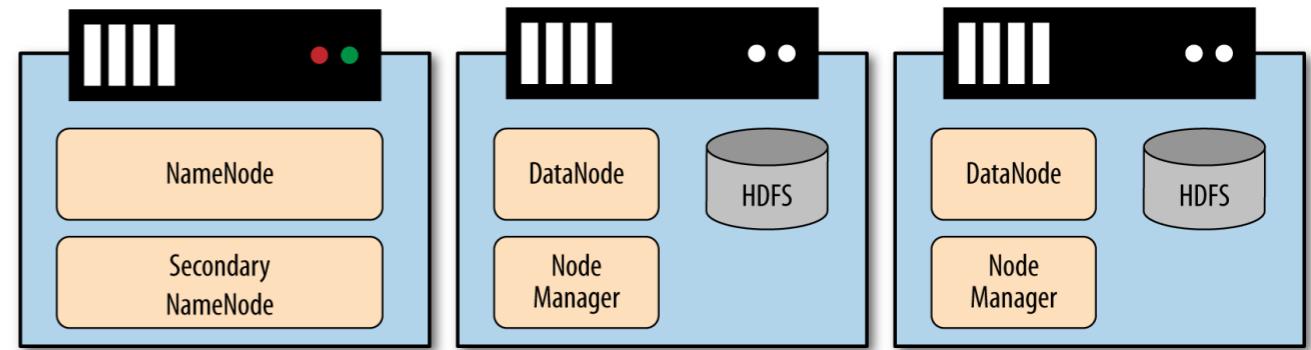
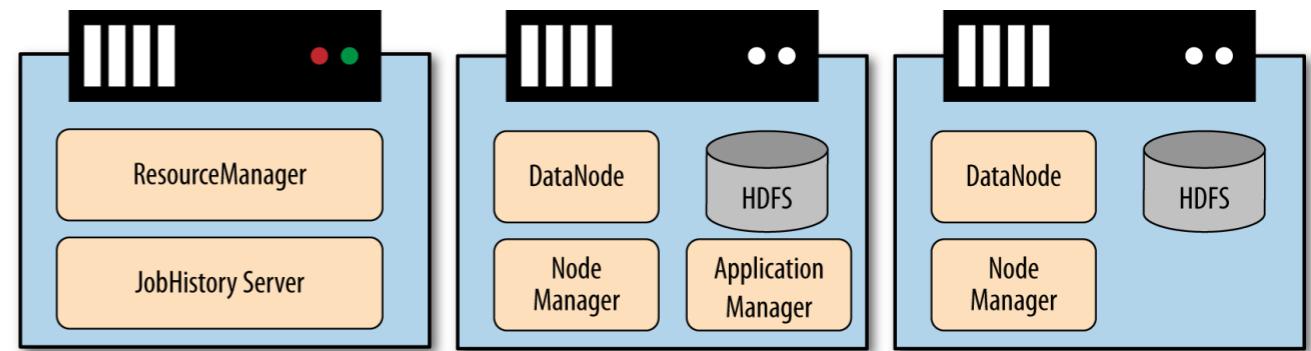
- Hadoop posee dos componentes principales que implementan los conceptos básicos de computación y almacenamiento distribuido.
- **HDFS** (a veces abreviado como DFS) es el sistema de archivos distribuidos de Hadoop, responsable de administrar los datos almacenados en discos en todo el clúster.
- **YARN** actúa como un administrador de recursos del clúster, asignando recursos computacionales (disponibilidad de procesamiento y memoria en los nodos trabajadores) a las aplicaciones que desean realizar una computación distribuida.
- HDFS y YARN funcionan en conjunto
 - Minimizando la cantidad de tráfico de red en el clúster,
 - Garantizar que los datos sean locales para el cálculo requerido.
 - La duplicación de datos y tareas garantiza la tolerancia a fallas, la capacidad de recuperación y la consistencia.
- HDFS y YARN son tecnologías para construir aplicaciones de big data.



Hadoop

A Hadoop cluster

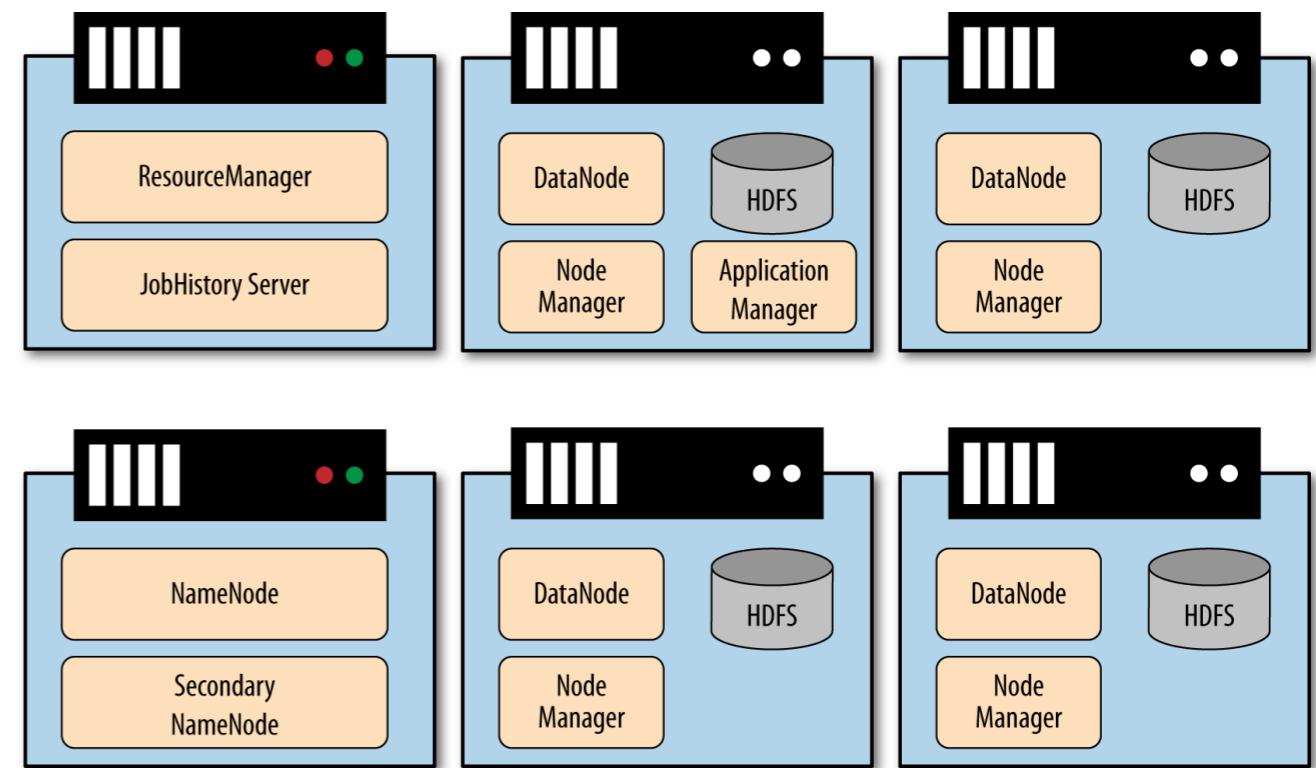
- YARN y HDFS se implementan mediante varios procesos (demonios), es decir, software que se ejecuta en segundo plano y no requiere intervención del usuario.
- Los procesos de Hadoop son servicios, se ejecutan todo el tiempo en un nodo de clúster y aceptan entradas y entregan salidas a través de la red (similar a un servidor HTTP)
- Cada uno de estos procesos se ejecuta dentro de su propia máquina virtual Java (JVM), por lo que cada demonio tiene su propia asignación de recursos del sistema.
- Nodos maestros:
 - Estos nodos ejecutan servicios de coordinación y, por lo general, son los puntos de entrada para el acceso de los usuarios al clúster.
- Nodos trabajadores:
 - Los nodos trabajadores ejecutan servicios que aceptan tareas de los nodos maestros, ya sea para almacenar o recuperar datos o para ejecutar una aplicación en particular.



Hadoop

A Hadoop cluster

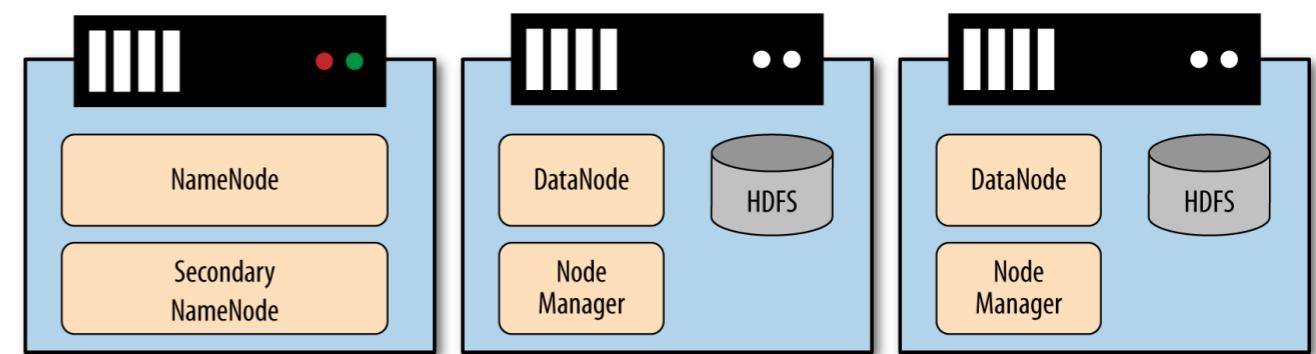
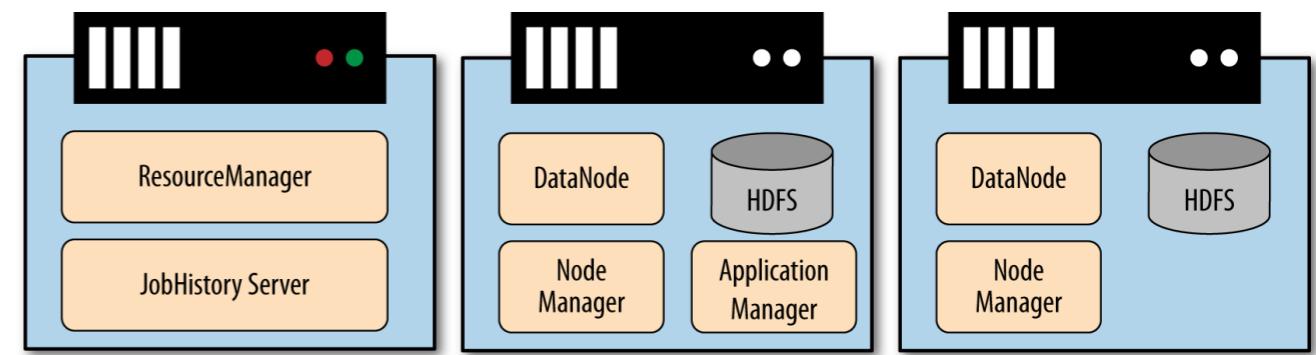
- Para HDFS, los servicios maestro y trabajador son los siguientes:
 - NameNode (Master)
 - Almacena el árbol de directorios del sistema de archivos, los metadatos del archivo y las ubicaciones de cada archivo en el clúster.
 - Secondary NameNode (Master)
 - Realiza tareas de limpieza y puntos de control para el NameNode
 - DataNode (Worker)
 - Almacena y administra bloques HDFS en el disco local



Hadoop

A Hadoop cluster

- YARN tiene múltiples servicios maestros y un servicio de trabajador:
 - ResourceManager (Master)
 - Asigna y supervisa los recursos de clúster disponibles a las aplicaciones, además de manejar la programación de trabajos en el clúster.
 - ApplicationMaster (Master)
 - Coordina una aplicación particular que se ejecuta en el clúster según lo programado por ResourceManager.
 - NodeManager (Worker)
 - Ejecuta y administra tareas de procesamiento en un nodo individual y también informa sobre el estado y el estado de las tareas a medida que se ejecutan.



Google Colab

The screenshot shows a Google Colab notebook titled "Rust-C-basics.ipynb". The notebook contains two sections: "Hola Mundo" and "Arreglos y funciones".

Hola Mundo:

```
[ ] 1 !apt install rustc cargo
[ ] 1 %%writefile helloworld.rs
2 // This is a comment, and is ignored by the compiler
3
4 // This is the main function
5 fn main() {
6     // Statements here are executed when the compiled binary is called
7     // Print text to the console
8     println!("Hola Mundo!");
9 }
10

Writing helloworld.rs

[ ] 1 !rustc /content/helloworld.rs
2 ./helloworld

[ ] Hola Mundo!
```

Arreglos y funciones:

```
[ ] 1 %%writefile arreglos.rs
2
3 use std::mem;
4
5 // This function borrows a slice
6 fn analyze_slice(slice: &[i32]) {
7     println!("primer elemento del arreglo: {}", slice[0]);
8     println!("el ultimo elemento es: {}", slice[slice.len()-1]);
9     println!("el arreglo tiene {} elementos", slice.len());
10 }

11
12 fn main() {
13     // arreglo de largo 5 fijo
14     let xs: [i32; 5] = [1, 2, 3, 4, 5];
15     // arreglo de largo 500 inicializado en 0
16     let ys: [i32; 500] = [0; 500];
17
18     // indices comienzan en 0
19     println!("primer elemento arreglo: {}", xs[0]);
20     println!("tercer elemento del arreglo: {}", xs[2]);
21
22     // `len` retorna el largo del arreglo
23     println!("numero de elemenos arreglo xs: {}", xs.len());
24     println!("numero de elementos arreglo ys: {}", ys.len());
25
26     // Arrays are stack allocated
27     println!("tamaño en memoria xs {} bytes", mem::size_of_val(&xs));
28     println!("tamaño en memoria ys {} bytes", mem::size_of_val(&ys));
29     // Arrays can be automatically borrowed as slices
30     println!("pasamos el arreglo por puntero a una función");
31     analyze_slice(&xs);
32     analyze_slice(&ys);
33
34     println!("pasamos una slice del array");
35     analyze_slice(&ys[1 .. 4]);
36
37     //Se puede acceder a las matrices de forma segura mediante `.get`,
38     //que devuelve un
39     //`Opción`. Esto se puede combinar como se muestra a continuación, o se puede usar con
40     //`.expect()` si desea que el programa finalice con un buen
41     //mensaje en lugar de continuar.
42     for i in 0..xs.len() + 1 { // iteramos un elemto mas del largo
43         match xs.get(i) {
44             Some(xval) => println!("{}: {}", i, xval),
45             None => println!("Fuerza de indice! {} no existe!", i),
46         }
47     }
```

<https://github.com/adigenova/uohpmd/blob/main/code/Hadoop.ipynb>

Consultas?

Consultas o comentarios?

Muchas gracias