**Conociendo Apache Spark**

**Concepto general**

Spark es un framework para distribuido procesamiento de datos, que en la mayoría de veces se ejecuta en un cluster compuesta de pequeñas maquinas ( escalabilidad horizontal) donde hace uso de master/slave arquitectura.

**Concepto técnico**

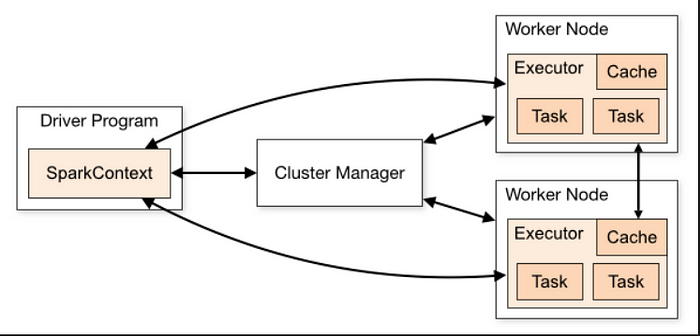


Imagen obtenida de la documentación de spark

Observando la imagen podemos observar 3 componentes.

* Driver (SparkContext):

Driver component es donde se ejecuta la el programa principal, este tambien es el encargado de interactuar con el componente Cluster Manager donde pregunta por los recursos disponibles del cluster. Otra funcion principal de el driver es que divide el programa principal en pequeñas tareas que son enviadas a los Executers

* Cluster Manager:

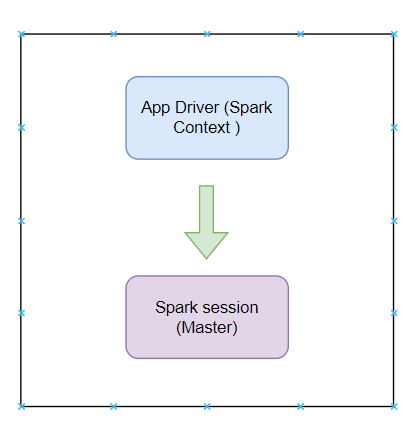
El cluster manager se comunica con el driver para mostrarle los recursos disponible. También se comunica con los executors para obtener status de disponibilidad en caso el executor falle.

* Executors ( Worker node)

Los Executors son workers nodes procesos encargados de ejecutar tareas individuales en un dado Spark job, cuando acaba la tarea el executor guarda los datos procesados y se comunica directamente con el Driver para notificar que ya acabo.

**Como trabaja Spark internamente**

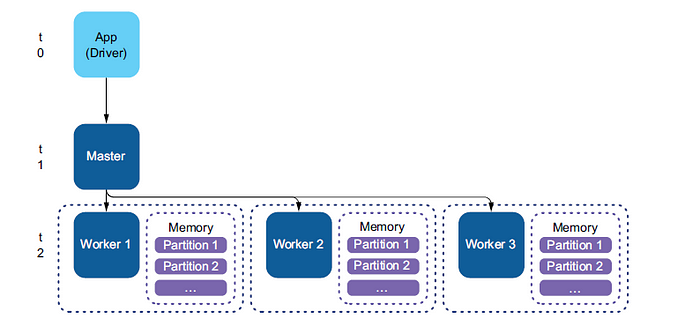
Iniciemos con lo mas básico, cuando creas tu aplicación instancias un Driver (Spark Context). El Spark context se conecta a master para obtener un instancia de Spark session.



Inicio de un programa en Spark

Spark cuando opera en un cluster, usa ingesta distribuida. En mi opinión este concepto va ser un punto de inflexión, en tu mindset de Spark.

En la imagen anterior ya establecimos el inicio de un programa , entonces ahora continuemos con un caso de uso, que nos ayude a entender el comportamiento interno.



Hemos mencionado dos cosas importantes en los anteriores parrafos, Spark implementa la arquitectura master/slave, por la tanto confia en sus workers/slaves para el procesamiento. Y por otro lado mencionamos que Spark trabaja de una forma distribuida, por ejemplo un CSV file puede ser ingestado de una forma distribuida. Como requisito en una ingesta distribuida el archivo fuente, debe estar almancenado en un Shared drive, HDFS (distributed file system) como s3, hadoop, cloud storage, etc.

Teniendo claro los dos puntos importantes, entremos un poco mas a detalle en los workers, como podemos obervar en la imagen cada worker tiene una memoria, y cada memoria tiene secciones dedicadas, llamadas particiones donde se va almacenar los registros de un datasource.

Los workers comienzan a generar tareas para leer los registros de el archivo CSV ( nuestro caso de uso ), cada worker tiene acceso a la memoria para asignar una partición a una tarea especifica. Veamos la imagen de abajo para entender mejor.

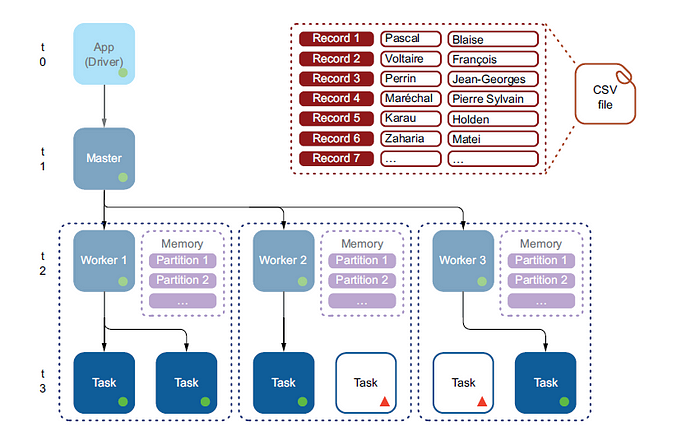


Imagen de Spark Action Book — Jean-Georges Perrin

*Como observamos en la imagen, el Worker 1, ha generado 2 tareas para leer un fragmento de el archivo CSV. Por ultimo las dos tareas en el nodo del Worker 1, tienen acceso a una partición en la memoria.*

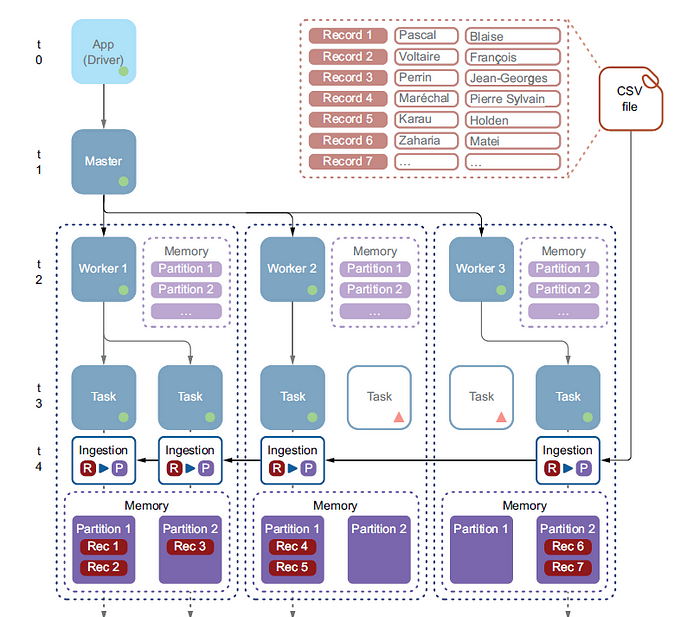


Imagen de Spark Action Book — Jean-Georges Perrin

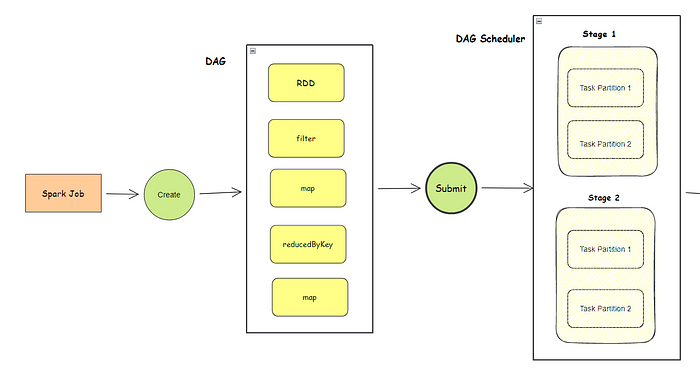
Teniendo como base que son los workers, como se generan las tareas, quien asigna las particiones a las tareas. Podemos ir al ultimo paso, en la parte t4. La caja Ingestion ( R >P) indica que la tarea ( Task ) carga los registros en una partición especifica. Por ejemplo en la imagen anterior podemos observar que la Partition 1 contiene los dos primeros registros del archivo CSV.

Aplicar tunning en jobs de Spark

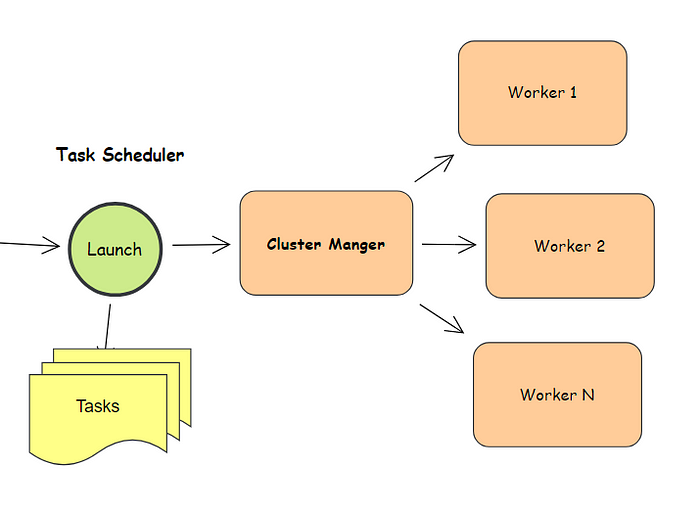
Para entender como funciona por de bajo Spark es necesario entender tres conceptos claves

1. DAG:

Cuando desarrollas una aplicación de Spark, estas creando un DAG. Y un DAG básicamente es un Grafo Acíclico Dirigido (**Directed Acyclic Graph** ) que consta de Vértices y Aristas, donde los vértices en Spark son RDDs y las aristas representan transformaciones que se aplican a los RDD. Por ultimo vamos explicar como funciona Spark dentro de un DAG, actualmente existe un **DAG Scheduler** que divide el grafo en varios Stages de tareas. Estas tareas se generan basado en las particiones de el input data (dataset).



Luego estos stages son pasado a un **Task Scheduler**, que se encargan de enviar las tareas de los stages vía**Cluster Manager** componente para ser ejecutado paralelamente en los workers.



2. Lazy

Antes de definir el comportamiento Lazy de Spark, quiero mencionar o repasar 2 eventos claves dentro de un flujo de Spark.

* Transformaciones

Son funciones (built-in or custom ) que se aplican a un conjunto de datos, por ejemplo tenemos a la funcion filter, u otra mas compleja tenemos a la función reducedByKey. Finalmente tenemos dos tipos de transformaciones:

a) narrow transformation: No necesita shuffle, esto quiere decir que los datos de cada worker no dependen de otro worker, por lo tanto no hay un gasto de I/O network.

b) wirde transformation: Necesita shuffle, por el contrario de narrow este si necesita interactuar con datos de otros workers, para tener una salida consolidada tomando en cuenta toda la informacion de todos los workers.

* Acciones

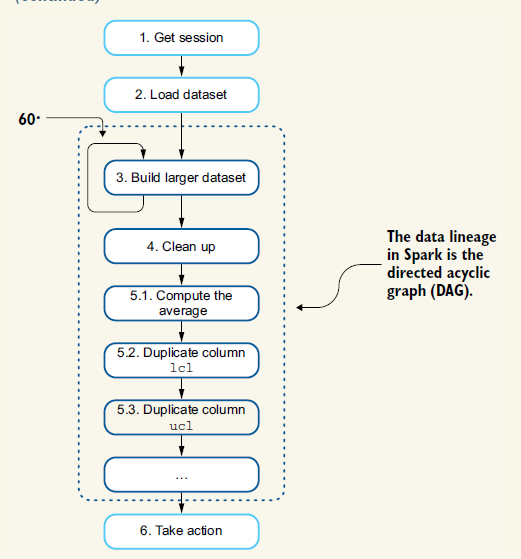
Las acciones en Spark son como activadores, de un flujo de transformaciones.

Lazy comportamiento en Spark hace que cuando se llamen transformaciones en un DAG no se ejecuten inmediatamente, en ves de eso lo registra como una referencia dentro del flujo de el DAG hasta que se llame una accion. Cuando se llama una accion el DAG comienza a ejecutar todas las “referencias” (transformaciones). Se podría concluir que un DAG es como una ruta de waze hacia un destino, y la accion seria salir con tu vehiculo e ir paso a paso (transformacion a transformacion) hasta llegar a tu destino.

3. Data Lineage

data lineage es representado por cada transformación que se registra dentro del flujo en un DAG. Pues como su nombre indica linaje, osea hablamos desde el nacimiento de un dataset y todas las transformaciones que tiene de inicio a fin (accion) .

Abajo podemos observar un ejemplo de un DAG con su lineage.



Catalyst, Spark’s built-in optimizer

Caching

You’ll also learn about caching’s multiple storage levels, allowing

you to fine-tune caching.

Checkpointing

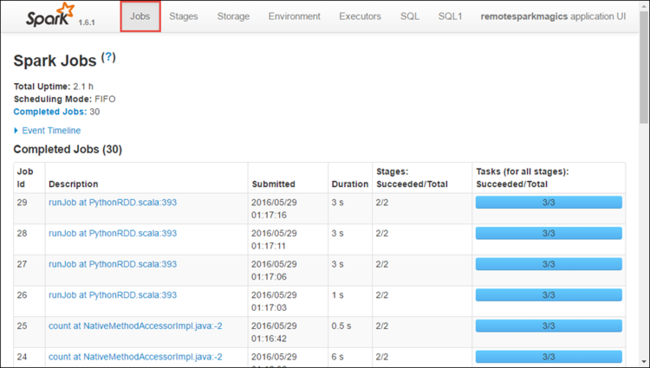
The checkpoint() method will truncate the DAG (or logical plan) and save the

content of the dataframe to disk.

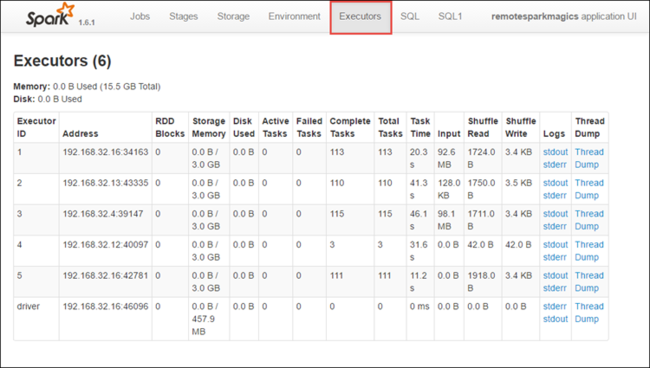
## Seguimiento de una aplicación en la interfaz de usuario de Spark

En la interfaz de usuario de Spark, puede explorar en profundidad los trabajos de Spark generados por la aplicación que ha iniciado anteriormente.

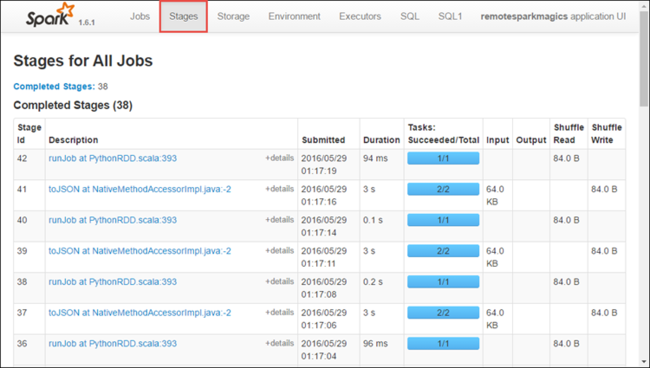
1. Para iniciar la interfaz de usuario de Spark, desde la vista de la aplicación, seleccione el vínculo en la **Dirección URL de seguimiento**, como se muestra en la captura de pantalla anterior. Puede ver todos los trabajos de Spark iniciados por la aplicación que se ejecuta en el cuaderno de Jupyter Notebook.



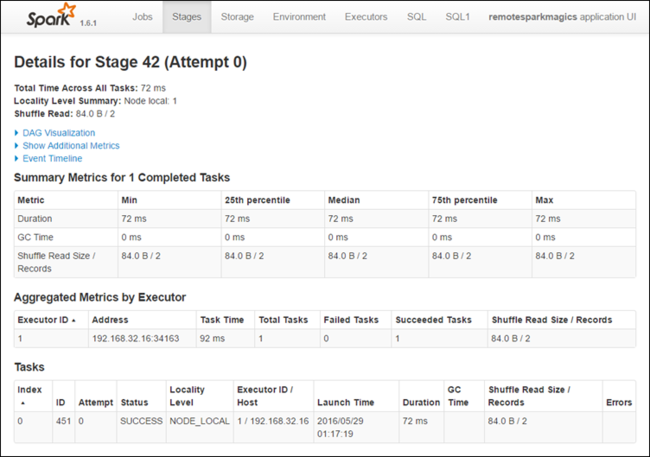
1. Seleccione la pestaña **Executors** (Ejecutores) para ver la información de procesamiento y almacenamiento de cada elemento de ejecución. También puede recuperar la pila de llamadas seleccionando el vínculo **Thread Dump** (Volcado de subprocesos).



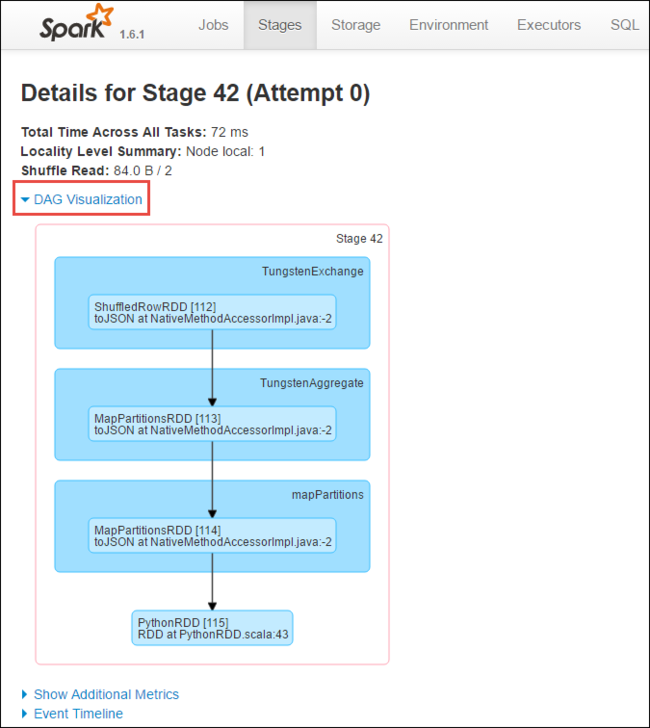
1. Seleccione la pestaña **Stages** (Fases) para ver las fases asociadas a la aplicación.



Cada fase puede tener varias tareas de las que puede ver las estadísticas de ejecución, como se muestra a continuación.

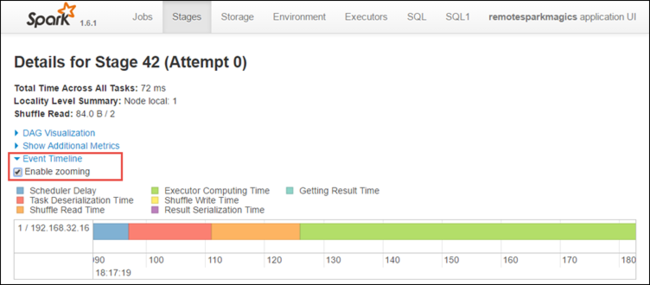


1. En la página de detalles de la fase, puede iniciar la visualización de DAG. Expanda el vínculo **DAG Visualization** (Visualización de DAG) en la parte superior de la página, como se muestra a continuación.



El DAG o grafo acíclico dirigido representa las distintas fases de la aplicación. Cada cuadro azul en el grafo representa una operación de Spark que se invoca desde la aplicación.

1. En la página de detalles de la fase, también puede iniciar la vista de escala de tiempo de la aplicación. Expanda el vínculo **Event Timeline** (Escala de tiempo del evento) en la parte superior de la página, como se muestra a continuación.



Esta imagen permite mostrar los eventos de Spark en forma de una escala de tiempo. La vista de la escala de tiempo está disponible en tres niveles: entre trabajos, dentro de un trabajo y dentro de una fase. La imagen anterior captura la vista de la escala de tiempo para una fase determinada.

**Sugerencia**

Si selecciona la casilla **Enable zooming** (Habilitar zoom), puede desplazarse a izquierda y derecha en la vista de escala de tiempo.

1. Otras pestañas de la interfaz de usuario de Spark proporcionan también información útil acerca de la instancia de Spark.
   * Pestaña Almacenamiento: si la aplicación crea varios RDD, podrá encontrar información sobre estos en esta pestaña.
   * Pestaña Environment (Entorno): esta pestaña proporciona información útil sobre la instancia de Spark, como:
     + La versión de la escala
     + El directorio de registro de eventos asociado con el clúster
     + El número de núcleos del ejecutor de la aplicación