Procesamiento Distribuido de Datos con Apache Spark

Apache Spark es un potente motor de procesamiento distribuido diseñado para manejar grandes volúmenes de datos de manera rápida y eficiente. Esta presentación explorará los conceptos fundamentales de Spark, desde la configuración inicial hasta la implementación de trabajos complejos.

Analizaremos los componentes clave como RDDs, transformaciones y acciones, además de ver ejemplos prácticos de su aplicación. También examinaremos las opciones disponibles en la nube para implementar soluciones basadas en Spark.

por Kibernum Capacitación S.A.

Ventajas de Apache Spark en Entornos de Datos Masivos

Procesamiento de grandes cantidades de datos

¿Alguna vez has trabajado o escuchado sobre una herramienta que procese grandes cantidades de datos? ¿Qué desafíos crees que enfrenta una empresa cuando intenta analizar millones de registros en poco tiempo?

Análisis en tiempo real

Imagina que estás desarrollando una aplicación que analiza datos en tiempo real (como clics en una página web o transacciones bancarias). ¿Por qué crees que una herramienta como Spark podría ser útil en ese escenario?

Implementación en la nube vs. local

Spark se puede ejecutar en clústeres, tanto localmente como en la nube. ¿Qué ventajas crees que tiene usar Spark en la nube en lugar de en un servidor local?

Configuración, Conexión y Contexto de Spark

Interfaz de Conexión

Proporciona la interfaz para conectarse al clúster y controlar la distribución de tareas.

Configuraciones Clave

Permite definir parámetros esenciales como memoria, núcleos y configuraciones de ejecución.

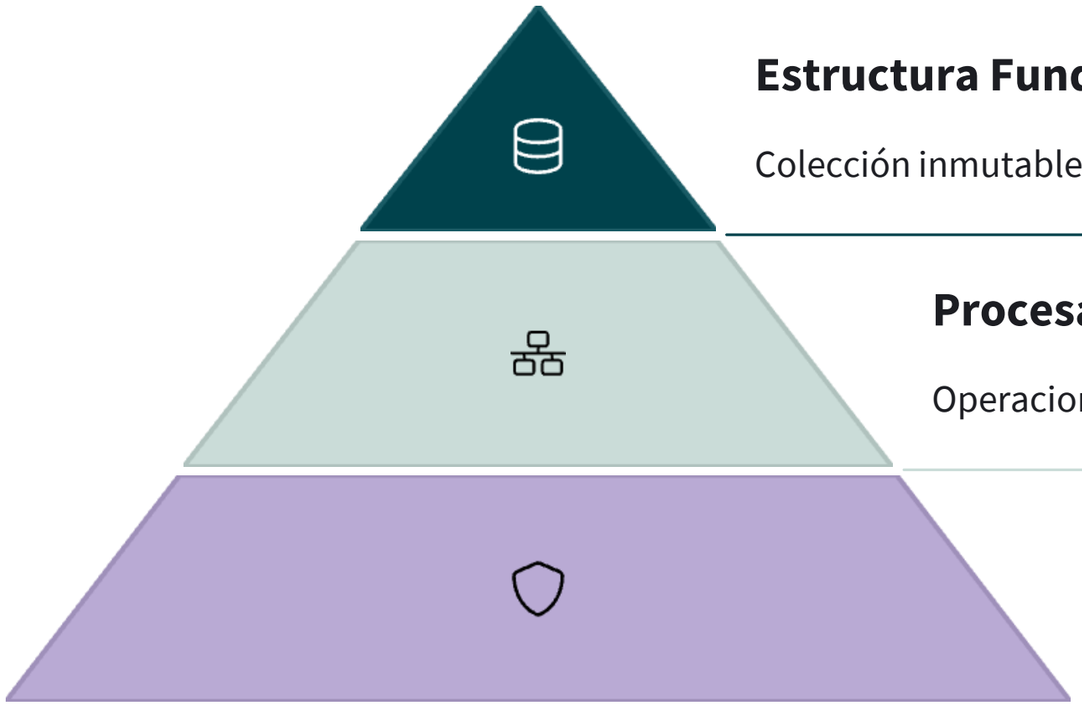
Ejecución de Operaciones

Es esencial para ejecutar transformaciones y acciones en los RDDs y DataFrames.

El SparkContext es la entrada principal que permite interactuar con las funcionalidades de Spark, coordinando la ejecución de tareas en un clúster distribuido. SparkSession, introducido en Spark 2.0, simplifica la inicialización del entorno y proporciona acceso unificado a diferentes APIs (RDD, DataFrame y SQL).

```python
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("MiPrimerContextoSpark") \
    .master("local[*]") \
    .config("spark.executor.memory", "2g") \
    .getOrCreate()

    # .appName  Nombre de la aplicación
    # .master Ejecutar en modo local usando todos los núcleos disponibles
    # .config Configuración de memoria del ejecutor

# Mostrar información sobre la sesión
print("Sesión de Spark creada exitosamente")
print(f"Nombre de la aplicación: {spark.sparkContext.appName}")
print(f"Master: {spark.sparkContext.master}")
print(f"Versión de Spark: {spark.version}")
```

RDD (Resilient Distributed Dataset)



Estructura Fundamental

Colección inmutable de elementos distribuidos

Procesamiento Paralelo

Operaciones eficientes en clúster

Tolerancia a Fallos

Recuperación automática ante errores

Los RDDs son la estructura de datos fundamental en Apache Spark que permite realizar operaciones de procesamiento de forma paralela y eficiente. Se pueden crear a partir de listas en memoria, archivos externos o bases de datos.

Para optimizar el rendimiento, Spark permite almacenar RDDs mediante funciones como **persist()**, que guarda el RDD en memoria o disco según el nivel de persistencia, y **cache()**, que lo almacena en memoria para acceso rápido.

RDD (Resilient Distributed Dataset)sourc

```python
# Almacenar un RDD en memoria (cache)
rdd_lista.cache()

# Persistir un RDD en disco
from pyspark import StorageLevel
rdd_api.persist(StorageLevel.DISK_ONLY)

# Mostrar el almacenamiento
print(f"RDD en cache: {rdd_lista.is_cached}")
print(f"RDD persistido: {rdd_api.getStorageLevel()}")
```

Código completo
https://colab.research.google.com/drive/12a9sEzDgQuLiLT31LxONvnJe4l_uwgoh?usp=sharing

Transformaciones en Spark

Transformaciones Estrechas

Los datos de una partición específica del RDD de entrada son transformados en una sola partición del RDD de salida.

- map()
- filter()
- flatMap()

Transformaciones Amplias

Los datos de múltiples particiones del RDD de entrada son transformados y repartidos en múltiples particiones del RDD de salida.

- groupBy()
- reduceByKey()
- sortBy()

Una transformación en Apache Spark es una operación que toma un RDD o DataFrame como entrada y produce uno nuevo como resultado. Las transformaciones son perezosas (lazy), lo que significa que no se ejecutan inmediatamente cuando se definen, sino que se evalúan solo cuando una acción es invocada.

Transformaciones en Spark

Código completo

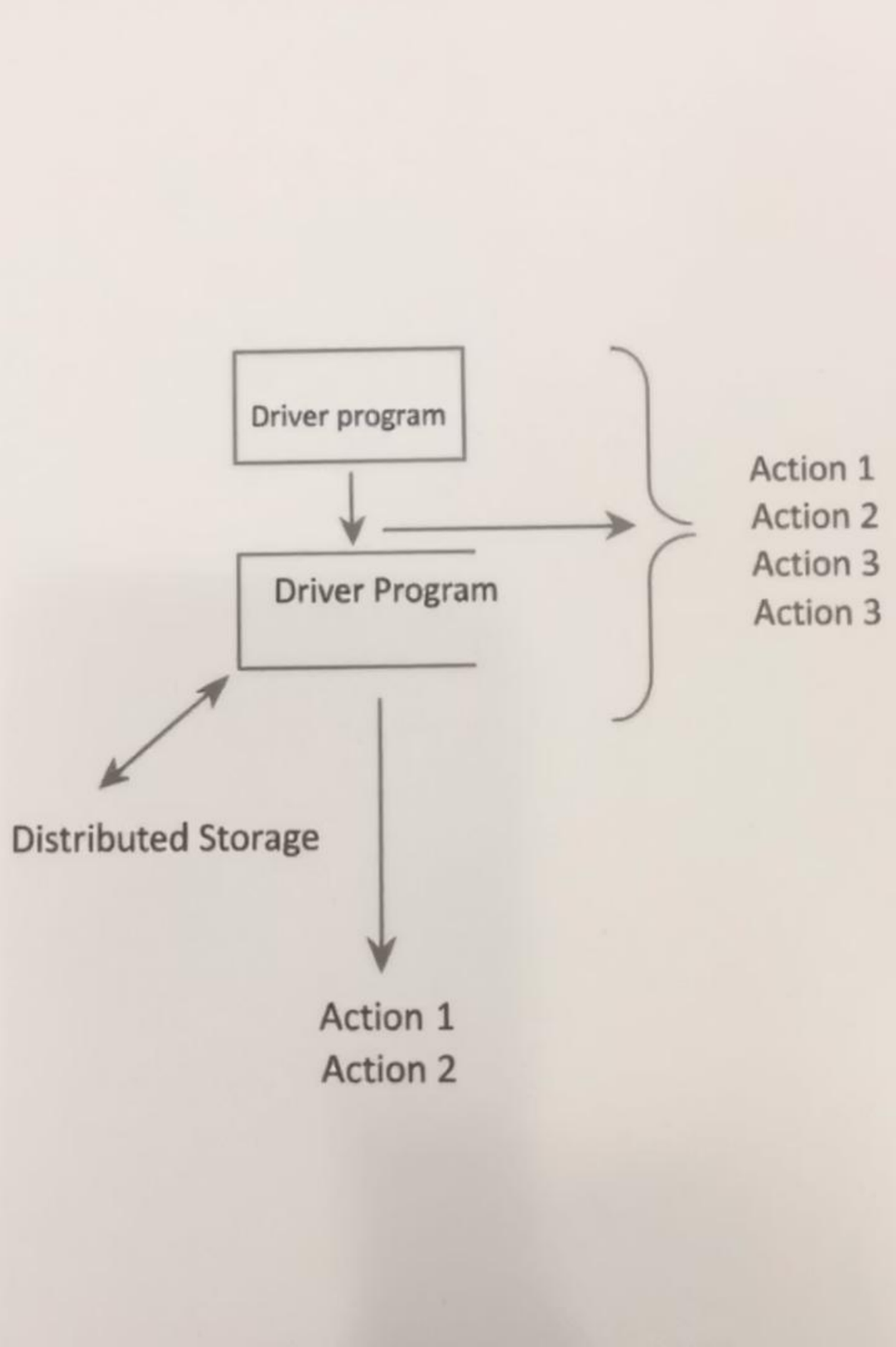
https://colab.research.google.com/drive/1Nlr-L3O7VG5XelW6t05fIej-F9nm5WFl?usp=sharing

```python
from pyspark.sql import SparkSession

# Crear una sesión de Spark
spark = SparkSession.builder \
    .appName("TransformacionesSpark") \
    .master("local[*]") \
    .getOrCreate()

# Crear un RDD de ejemplo
datos = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
rdd = spark.sparkContext.parallelize(datos)

# 1. filter(): Filtrar valores mayores a 5
rdd_filter = rdd.filter(lambda x: x > 5)

# 2. map(): Multiplicar cada número por 2
rdd_map = rdd.map(lambda x: x * 2)

# 3. flatMap(): Descomponer frases en palabras
oraciones = ["Hola mundo", "Apache Spark es increíble"]
rdd_oraciones = spark.sparkContext.parallelize(oraciones)
rdd_flatmap = rdd_oraciones.flatMap(lambda x: x.split(" "))
```

Acciones en Spark

Acción	Descripción	Resultado
collect()	Devuelve todos los elementos del RDD como una lista.	Lista
take(n)	Devuelve los primeros n elementos del RDD.	Lista
count()	Devuelve el número total de elementos del RDD.	Entero
reduce()	Aplica una función de reducción a los elementos.	Valor único
saveAsTextFile()	Escribe los datos del RDD en un archivo de texto.	Archivo

Una acción en Apache Spark es una operación que desencadena la ejecución del DAG (Directed Acyclic Graph) y devuelve un resultado al controlador o escribe los datos en un sistema de almacenamiento. A diferencia de las transformaciones, que son perezosas, las acciones fuerzan la evaluación de los RDDs o DataFrames.

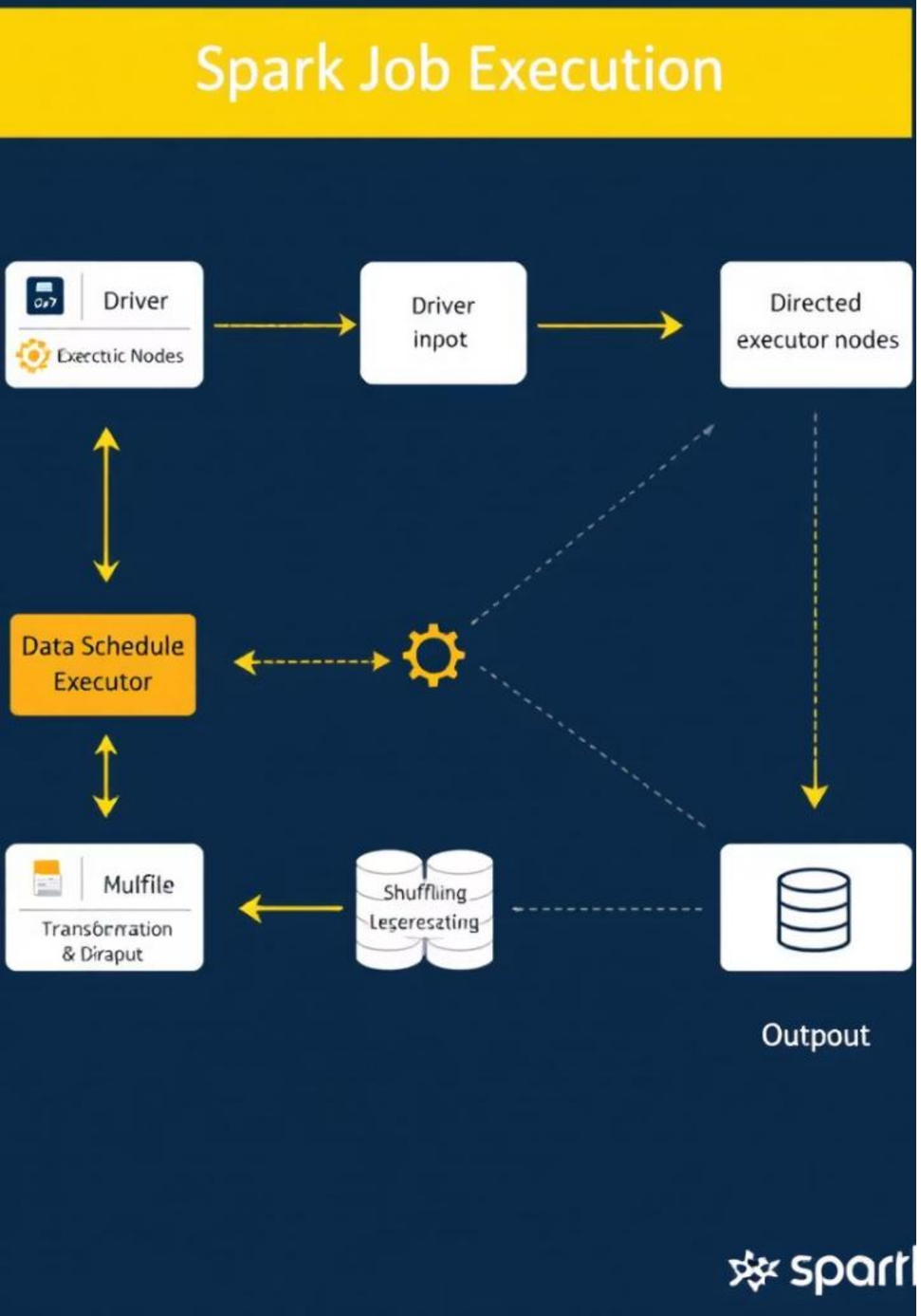


Acciones en Spark

```python
from pyspark.sql import SparkSession

# Crear una sesión de Spark
spark = SparkSession.builder \
    .appName("AccionesSpark") \
    .master("local[*]") \
    .getOrCreate()

# Crear un RDD de ejemplo
datos = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
rdd = spark.sparkContext.parallelize(datos)

# 1. collect(): Obtener todos los elementos del RDD
resultado_collect = rdd.collect()
print("Resultado de collect():", resultado_collect)

# 2. take(n): Obtener los primeros 5 elementos
resultado_take = rdd.take(5)
print("Resultado de take(5):", resultado_take)
```

Código completo:

https://colab.research.google.com/drive/1JHgdvW0SUNTAAjsZkKay8nqOVEWOa8DV?usp=sharing



Job Spark: Ejecución de Tareas

Definición de Transformaciones

Creación del plan de ejecución

Construcción del DAG

Grafo acíclico dirigido de operaciones

Ejecución en Clúster

Distribución de tareas entre nodos

Recolección de Resultados

Agregación de datos procesados

Un Job en Spark es una tarea que se ejecuta como resultado de una acción invocada en un RDD o DataFrame. Cuando se llama a una acción, Spark crea un DAG (Directed Acyclic Graph) de transformaciones y envía ese DAG al clúster para su ejecución, representando una secuencia completa de transformaciones de datos.

Job Spark: Ejecución de Tareas

```python
from pyspark.sql import SparkSession

# Crear sesión de Spark
spark = SparkSession.builder \
    .appName("JobSparkEjemplo") \
    .master("local[*]") \
    .getOrCreate()

# Crear un RDD de ejemplo
datos = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
rdd = spark.sparkContext.parallelize(datos)

# Aplicar transformaciones (Lazy Evaluation - No se ejecutan aún)
rdd_transformado = rdd.map(lambda x: x * 2).filter(lambda x: x > 10)

# Acción que inicia un Job
resultado = rdd_transformado.collect()

# Mostrar resultado
print("Resultado del Job:", resultado)
```

Código completo:

https://colab.research.google.com/drive/1U6Qav9YUpJbwYDfdAqzZ2c9McIm2HllU?usp=sharing

Actividad Práctica Guiada

OBJETIVO: Aplicar los conceptos aprendidos sobre información obtenida de una API pública

Paso 1: Crear una sesión de Spark.

•Paso 2: Obtener datos desde la API JsonPlaceholder.

•Paso 3: Verificar si la respuesta es exitosa.

•Paso 4: Crear un RDD desde los datos obtenidos desde la API.

•Paso 5: Mostrar algunos registros del RDD para validar la carga.

El código con los pasos es el siguiente:

```python
import requests
from pyspark.sql import SparkSession

# Crear una sesión de Spark
spark = SparkSession.builder \
    .appName("ActividadAPITransformacionesSpark") \
    .master("local[*]") \
    .getOrCreate()

# URL de la API pública
url_api = "https://jsonplaceholder.typicode.com/posts"

# Obtener datos desde la API
respuesta = requests.get(url_api)

# Verificar si la respuesta es exitosa
if respuesta.status_code == 200:
    datos_json = respuesta.json()
else:
    raise Exception(f"Error al obtener datos de la API. Código: {respuesta.status_code}")

# Crear un RDD desde los datos obtenidos de la API
rdd_api = spark.sparkContext.parallelize(datos_json)

# Mostrar algunos registros del RDD para validar la carga
print("Primeros 5 registros del RDD desde API:")
for registro in rdd_api.take(5):
    print(registro)
```

Código completo:

https://colab.research.google.com/drive/1DJDg0o63H1j3l-8uGwbEfCrOwaUdJC-S?usp=sharing

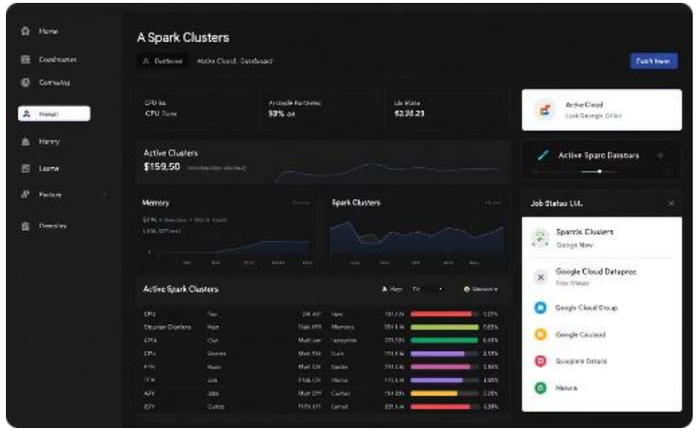
Proveedores de Servicios en la Nube para Apache Spark

Amazon Web Services (AWS)

Ofrece Amazon EMR (Elastic MapReduce) con alta escalabilidad e integración con el ecosistema AWS. Proporciona configuración flexible y opciones de precios bajo demanda o instancias reservadas.

Microsoft Azure

Implementa Spark a través de Azure HDInsight con fuerte integración con herramientas de Microsoft. Destaca por su facilidad de administración y capacidades de seguridad empresarial.



Google Cloud

Proporciona Dataproc para ejecutar Spark con rápido aprovisionamiento y escalado automático. Se integra perfectamente con BigQuery y otras herramientas de análisis de Google.

La elección del proveedor dependerá de factores como la escalabilidad requerida, integración con otros servicios, facilidad de configuración y administración, y los costos asociados. Un análisis comparativo detallado ayudará a determinar la mejor opción para cada caso de uso específico.

Preguntas de Reflexión Final

1) ¿Por qué es importante la evaluación diferida (*Lazy Evaluation*) en Spark?

2) ¿Qué transformaciones y acciones aplicarías para limpiar y analizar grandes volúmenes de datos?

3) ¿Cuál es el beneficio de usar RDDs frente a otros modelos de datos en Spark?