Big Data Senior Engineer

Prueba técnica

Documento guión para poder desarrollar la prueba técnica cuyo objetivo es poder analizar la pericia técnica del candidato desde diferentes prespectivas relevantes para la posición de Big Data Senior Engineer en la práctica de Data Technologies de SDG Group.

Prueba técnica para la posición de Big Data Senior Engineer en la práctica de Data Technologies de SDG Group

El objetivo de la prueba técnica es que puedas demostrar tus capacidades de desarrollo con Spark, conceptos básicos del ciclo de vida de data engineering - pipelining - y toda la ingeniería e infraestructura que pueda girar alrededor para desarrollar la tarea de Data Engineer en los ámbitos más contemporáneos ligados a la tendencia de Data Ops.

Somos conscientes de que algunos de los componentes tecnológicos que proponemos para esta prueba técnica quizá no los conozcas, pero pensamos que por tus capacidades podrás adquirir un conocimiento básico en estos días para sacar adelante la prueba (la cual, por otro lado, es bastante libre).

Los puntos a tratar en la prueba técnica son los siguientes:

1. Si no lo has usado nunca, familiarizarte con el uso de Docker para la ejecución de sistemas y aplicaciones contenerizadas. Este punto es importante porque te servirá para montar el entorno que proponemos a continuación. Docker se maneja mejor en entornos Linux, pero es perfectamente viable utilizarlo en Windows siempre que tengas Windows 10 o superior. Si estás en Windows, te recomiendo utilizar WLS 2.

2. Utilizar Docker para desplegar Apache Spark en local. Para ello, puedes utilizar por ejemplo esta imagen, siguiendo las instrucciones que vienen ahí para levantar un clúster en local con dos *workers*:
    1. Descárgate la especificación del despliegue de Docker Compose (el ejemplo utiliza cURL, de ahí también la recomendación de utilizar WLS2 si estás en Windows):

        ```
        curl -LO https://raw.githubusercontent.com/bitnami/bitnami-docker-spark/master/docker-compose.yml
        ```

    2. Levanta directamente el clúster haciendo uso de Docker Compose (desde la ruta donde hayas descargado el archivo docker-compose.yml):

        ```
        docker-compose up
        ```

    3. A partir de ahí, desde tu navegador podrás acceder a la interfaz web de Apache Spark a través de la URL http://localhost:8080/.

3. Monta un sistema HDFS mediante docker-compose. Tienes un ejemplo de cómo hacerlo en este enlace.

4. Después, utilizar Docker para desplegar Apache Airflow. Apache Airflow es un orquestador de procesos que, a diferencia de otros, se basa en la especificación de esos flujos de ejecución a través de *scripts* de Python. Para facilitarte esta labor, te doy las siguientes pistas:
    1. Utiliza la siguiente imagen de Docker para realizar un montaje de Apache Airflow en local: https://hub.docker.com/r/apache/airflow

    2. Para ello, primero bájate la imagen directamente desde tu Docker local:

        ```
        docker pull apache/airflow
        ```

3. Después, puedes levantar un contenedor de Docker que lance Apache Airflow y cargue los DAGs (los flujos) que posteriormente tú vayas implementando desde una ruta local (C:\Users\Lenovo\Desktop\dags en mi caso) mediante el siguiente comando:

   ```
   docker run -d -p 8090:8080 --name airflow -e LOAD_EX=y -v
   C:\Users\Lenovo\Desktop\dags:/usr/local/airflow/dags apache/airflow bash -c
   "airflow db init && airflow webserver"
   ```

4. Una vez el la instalación haya finalizado, entra en el contenedor con el comando:

   ```
   docker exec -ti airflow bash
   ```

5. Una vez finalizado y estando dentro del contenedor, debes crear un usuario para airflow y especificarle el password cuando lo pregunte con el comando:

   ```
   airflow users create --username admin --firstname FIRST_NAME --lastname
   LAST_NAME --role Admin --email admin@example.org
   ```

6. El último paso de la configuración del airflow es arrancar el scheduler. Para ello, estando dentro del contenedor, ejecuta el comando:

   ```
   airflow scheduler
   ```

7. Una vez tengas todo configurado, desde tu navegador podrás acceder a la interfaz web de Apache Airflow a través de la URL http://localhost:8090/, donde verás los DAGs que hayas metido en tu carpeta local (si hay alguno)

5. Finalmente, utiliza docker para montar un Confluent – Kafka. Puedes usar esta imagen: https://hub.docker.com/r/confluentinc/cp-kafka

   Nota: Esta infraestructura se debe ver por red para interactuar entre sus diferentes componentes; hazlo como consideres, una opción podría ser orquestar todo mediante docker-composer pero quizás tengas que analizar si debes partir de otras imágenes de contenedores que existan y no las que te hemos dado.

6. Una vez que tengas Apache Airflow desplegado en local y te hayas familiarizado con su interfaz, la forma de construir DAGs y la forma de ejecutarlos, te propongo que construyas un DAG que represente la ejecución típica de un proyecto de ETL en Spark. Tu DAG debería realizar la siguiente secuencia de procesos:

   1. Levantar el clúster de Spark
   2. Levantar el clúster HDFS
   3. Levantar el clúster Kafka
   4. Subir los ficheros de datos y metadatos necesarios para el proceso de Spark a HDFS
   5. Ejecutar el proceso python/Spark - Apache Spark y de los operadores adecuados de Apache Airflow (SparkSubmitOperator, por ejemplo),
   6. Bajar los resultados de HDFS a Local
   7. Apagar clúster Spark
   8. Apagar clúster HDFS

9. Apagar clúster Kafka

7. En proceso Spark a construir es el siguiente:

1. Descripción:

   El programa realizado en Spark / Scala o Python debería, dirigido por los metadatos que se comentarán a continuación, ejecutar un proceso que cargará los orígenes descritos, ejecutará una serie de transformaciones (en este caso lo definido como dataflow: dos validaciones funcionales y técnicas de dos campos y añadir una columna) y escribir los datos resultado , tanto los que han pasado la validación, como por otro lado los que no las han pasado y el código de error que auto explique porque no lo ha pasado. Es muy importante que sea un programa que, o genera código leyendo metadatos que posteriormente se ejecuta, o que cambie su comportamiento dinámicamente vía los metadatos propuestos y que no sea un desarrollo ad hoc.

2. Datos de Entrada (fichero json ejemplarizante)

```
{"name":"Xabier","age”:39,"office":""}
{"name":"Miguel","office":"RIO"}
{"name":"Fran","age":31,"office":"RIO"}
```

3. Datos de Salida STANDARD_OK (fichero json ejemplarizante)

```
{"name":"Fran","age":31,"office":"RIO","dt":"2020-12-29 09:00:00" }
```

4. Datos de Salida STANDARD_KO (fichero json)

```
{"name":"Xabier","age":39,"office":"","arraycoderrorbyfield":{...}}
{"name":"Miguel","age":,"office":"RIO","arraycoderrorbyfield":{...}}
```

5. Metadatos que debe usar el programa

```
{
 "dataflows": [
  {
   "name": "prueba-acceso",
   "sources": [
    {
     "name": "person_inputs",
     "path": "/data/input/events/person/*",
     "format": "JSON"
    }
   ],
   "transformations": [
    {
     "name": "validation",
     "type": "validate_fields",
     "params": {
```

```
      "input" : "person_inputs",
      "validations": [
       {
        "field": "office",
        "validations": ["notEmpty"]
       },
       {
        "field": "age",
        "validations": ["notNull"]
       }
      ]
     }
    },
    {
     "name": "ok_with_date",
     "type": "add_fields",
     "params": {
      "input": "validation_ok",
      "addFields": [
       {
        "name": "dt",
        "function": "current_timestamp"
       }
      ]
     }
    }
   ],
   "sinks": [
    {
     "input": "ok_with_date",
     "name": "raw-ok",
     "topics": [
      "person"
     ],
     "format": "KAFKA"
    },
    {
     "input": "validation_ko",
     "name": "raw-ko",
     "paths": [
      "/data/output/discards/person"
     ],
     "format": "JSON",
     "saveMode": "OVERWRITE"
    }
   ]
  }
 ]
}
```

Consideraciones finales y entrega

Si tienes que priorizar en que te enfocas de la prueba técnica porque no tienes suficiente tiempo, el principal objetivo es que nos demuestres tus capacidades de Scala/Python y Spark e interacción con Kafka, así que nos gustaría indicar que la framework de ingesta y validación y transformación dirigida por metadatos que diseñes e implementes es en lo que más vamos a valorar en términos técnicos y en términos de presentación. Sin embargo, como hemos comentado, la infraestructura es algo indispensable y que hay que cuidar y más en ambientes Big Data de alto rendimiento; esta es la razón por la que te hemos propuesto montar un pequeño cluster de Spark y otro de Airflow e integrarlos. Además de esta manera veremos tus capacidades de infraestructura.

El objetivo de la prueba es la presentación en la reunión de una hora que hemos convocado, donde nos podrás explicar como has elaborado el código y organizado la infraestructura, pero valoraremos muy positivamente si el día anterior a la reunión de presentación nos envías por mail un repositorio privado de Git (en GitHub por ejemplo) con el código de la prueba y el resto de componentes necesarios para su ejecución.

Si en algún momento tienes algo que comentar con nosotros, o necesitas que movamos la fecha o la hora de presentación por algún tema personal, no dudes en contactarnos. Por favor, confírmanos que has recibido la prueba técnica y si la aceptas.

Sin más, muchas gracias por la entrevista y esperamos vernos contigo de nuevo en breve.

SDG Group

W. www.sdggroup.com
M. info.es@sdggroup.com