PRUEBA TÉCNICA XXX (Extracto)

Los puntos a tratar en la prueba técnica serían 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 . Si estás en Windows, te recomiendo utilizar WLS2.
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*:
    a. 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/dockercompose.yml
    b. Levanta directamente el clúster haciendo uso de Docker Compose (desde la ruta donde hayas descargado el archivo docker-compose.yml): docker-compose up
    c. A partir de ahí, desde tu navegador podrás accecer a la interfaz web de Apache Spark a través de la URL http://localhost:8080/.
3. 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:
    a. Utiliza la siguiente imagen de Docker para realizar un montaje de Apache Airflow en local: https://github.com/puckel/docker-airflow
    b. Para ello, primero bájate la imagen directamente desde tu Docker local: docker pull puckel/docker-airflow
    c. 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 puckel/docker-airflow webserver
    d. A partir de ahí, 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) además de una serie de ejemplos que seguro que te servirán para desarrollar los tuyos propios.
4. 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 tipica de un proyecto de ETL en Spark que por ejemplo puede ser: 1)subir datos y metadatos a storage que de soporte al proceso Spark, 2) ejecutar proceso Spark que represente a un dataflow,3) bajar datos resultado del storage que de soporte a Spark al destino final). Para ello, haciendo uso de tu despliegue de Apache Spark y de los operadores adecuados de Apache Airflow (SparkSubmitOperator, por ejemplo), lanza contra el clúster de Spark el siguiente programa a construir:
    a. **Descripcion**: El programa realizado en spark / scala debería, dirigido por los metadatos que se comentarán a continuación, ejecutar un proceso que cargará los orígenes descritos, ejecutara 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 autoexplique 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 via los metadatos propuestos y que no sea un desarrollo adhoc.

b. **Datos de Entrada** (fichero json ejemplarizante)

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

c. **Datos de Salida STANDARD_OK** (fichero json ejemplarizante)

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

d. **Datos de Salida STANDARD_KO** (fichero json)

```
{"name":"Xabier","age":39,"office":"","dt":"2020-12-29 09:00:00",” arraycoderrorbyfield”:{...}}
{"name":"Miguel","age":,"office":"RIO","dt":"2020-12-29 09:00:00",” arraycoderrorbyfield”:{...}}
```

e. **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",
                "paths": [
                    "/data/output/events/person"
                ],
                "format": "JSON",
                "saveMode": "OVERWRITE"
            },
            {
                "input": "validation_ko",
                "name": "raw-ko",
                "paths": [
                    "/data/output/discards/person"
                ],
                "format": "JSON",
                "saveMode": "OVERWRITE"
            }
        ]
    }
  ]
}
```

Si tienes que priorizar en que te enfocas de la prueba técnica porque no tienes suficiente tiempo, el principal objetivo es que nos desmuestres tus capacidades de Scala y Spark.