**Interacción de los Desarrolladores con la Arquitectura de Big Data**

**1. HDFS (Hadoop Distributed File System)**

* **Propósito:** Almacenamiento distribuido de archivos.
* **Interacción:**
  + **Subir Archivos:** Los desarrolladores pueden usar comandos de hdfs dfs para cargar datos en el sistema.

bash

Copiar código

hdfs dfs -put local\_file.txt /user/hadoop/

* + **Leer Archivos:** Leer archivos desde HDFS utilizando los mismos comandos.

bash

Copiar código

hdfs dfs -cat /user/hadoop/local\_file.txt

* + **Eliminar Archivos:**

bash

Copiar código

hdfs dfs -rm /user/hadoop/local\_file.txt

* **Uso Diario:** Continuo.
* **Encendido/Apagado:** Los servicios HDFS (NameNode y DataNode) deben estar siempre activos.

**2. Apache Spark**

* **Propósito:** Procesamiento distribuido de grandes volúmenes de datos.
* **Interacción:**
  + **Ejecutar Jobs de Spark:** Utilizando scripts en PySpark, Scala o Java.

bash

Copiar código

spark-submit --master spark://spark-master:7077 my\_spark\_job.py

* + **Interfaz Web:** Los desarrolladores pueden monitorear trabajos en la interfaz web de Spark en el puerto 8080.
* **Uso Diario:** Según sea necesario para ejecutar tareas de procesamiento.
* **Encendido/Apagado:** Spark Master y Spark Workers deben estar siempre activos. Los trabajos se envían y se ejecutan cuando sea necesario.

**3. Apache Kafka**

* **Propósito:** Sistema de mensajería distribuida para ingestión de datos en tiempo real.
* **Interacción:**
  + **Producir Mensajes:** Los desarrolladores pueden escribir datos en un topic de Kafka.

python

Copiar código

from kafka import KafkaProducer

producer = KafkaProducer(bootstrap\_servers='kafka:9092')

producer.send('my\_topic', b'Hello Kafka')

* + **Consumir Mensajes:** Leer datos desde un topic.

python

Copiar código

from kafka import KafkaConsumer

consumer = KafkaConsumer('my\_topic', bootstrap\_servers='kafka:9092')

for message in consumer:

print(message.value)

* **Uso Diario:** Continuo para procesamiento en tiempo real.
* **Encendido/Apagado:** Kafka y Zookeeper deben estar siempre activos.

**4. Apache Airflow**

* **Propósito:** Orquestación de flujos de trabajo de datos.
* **Interacción:**
  + **Definir DAGs:** Los desarrolladores definen flujos de trabajo mediante archivos Python.

python

Copiar código

from airflow import DAG

from airflow.operators.bash\_operator import BashOperator

from datetime import datetime

default\_args = {

'owner': 'airflow',

'start\_date': datetime(2024, 7, 31),

'retries': 1

}

dag = DAG('example\_dag', default\_args=default\_args, schedule\_interval='@daily')

task1 = BashOperator(

task\_id='print\_date',

bash\_command='date',

dag=dag)

* + **Monitoreo:** Utilizando la interfaz web en el puerto 8080.
  + **Ejecutar DAGs:** Los DAGs se ejecutan según el cronograma o manualmente desde la interfaz web.
* **Uso Diario:** Según la programación de los DAGs.
* **Encendido/Apagado:** Airflow debe estar siempre activo.

**5. Prometheus y Grafana**

* **Propósito:** Monitorización y visualización del rendimiento del sistema.
* **Interacción:**
  + **Definir Metricas en Prometheus:** Los desarrolladores configuran reglas de scrape en Prometheus.

yaml

Copiar código

scrape\_configs:

- job\_name: 'kubernetes'

static\_configs:

- targets: ['localhost:9090']

* + **Crear Dashboards en Grafana:** Los desarrolladores pueden crear dashboards personalizados en la interfaz web de Grafana en el puerto 3000.
* **Uso Diario:** Continuo para monitorización.
* **Encendido/Apagado:** Prometheus y Grafana deben estar siempre activos.

**6. Jenkins**

* **Propósito:** Automatización de CI/CD.
* **Interacción:**
  + **Configurar Jobs:** Los desarrolladores configuran trabajos de Jenkins en la interfaz web en el puerto 8080.

groovy

Copiar código

pipeline {

agent any

stages {

stage('Build') {

steps {

sh 'echo Building...'

}

}

stage('Test') {

steps {

sh 'echo Testing...'

}

}

stage('Deploy') {

steps {

sh 'echo Deploying...'

}

}

}

}

* + **Disparar Builds:** Los trabajos se disparan automáticamente por cambios en el repositorio o manualmente.
* **Uso Diario:** Según el ciclo de desarrollo.
* **Encendido/Apagado:** Jenkins debe estar siempre activo.

**Proceso de Desarrollo Diario**

1. **Desarrollo y Pruebas:**
   * Los desarrolladores crean y prueban sus scripts y DAGs en sus máquinas locales utilizando contenedores Docker similares a los del entorno de producción.
   * Utilizan spark-submit para probar trabajos de Spark y airflow dags test para DAGs de Airflow.
2. **Commit y Push:**
   * Una vez que las pruebas locales son exitosas, los desarrolladores hacen commit y push de sus cambios al repositorio de código.
   * Jenkins detecta los cambios y dispara el pipeline de CI/CD.
3. **Ejecución de Pipelines de CI/CD:**
   * Jenkins ejecuta el pipeline que incluye build, test y despliegue de los cambios.
   * Los resultados de las pruebas y los despliegues se monitorizan en Prometheus y Grafana.
4. **Monitorización y Ajustes:**
   * Los desarrolladores utilizan Grafana para visualizar métricas y logs de Prometheus y ELK Stack para asegurarse de que todo funcione correctamente.
   * Cualquier error o problema se aborda y se corrige en el ciclo de desarrollo siguiente.

**Encendido y Apagado**

* **Servicios Permanentes:** HDFS, Spark, Kafka, Airflow, Prometheus, Grafana, Jenkins.
  + Estos servicios deben estar siempre activos para garantizar la disponibilidad continua de la arquitectura y permitir la ingestión, procesamiento y monitorización de datos en tiempo real.
* **Escalado de Recursos:**
  + Los componentes en Kubernetes pueden escalarse dinámicamente según la carga de trabajo.
  + Utilizar Kubernetes Horizontal Pod Autoscaler (HPA) para escalar automáticamente los despliegues basados en métricas.

Este enfoque asegura que los desarrolladores puedan trabajar de manera eficiente y productiva en un entorno bien definido y gestionado, con una arquitectura de datos robusta y escalable que soporta sus necesidades de desarrollo y operaciones diarias.