Ciencia e Ingeniería de Datos en el Mundo Real

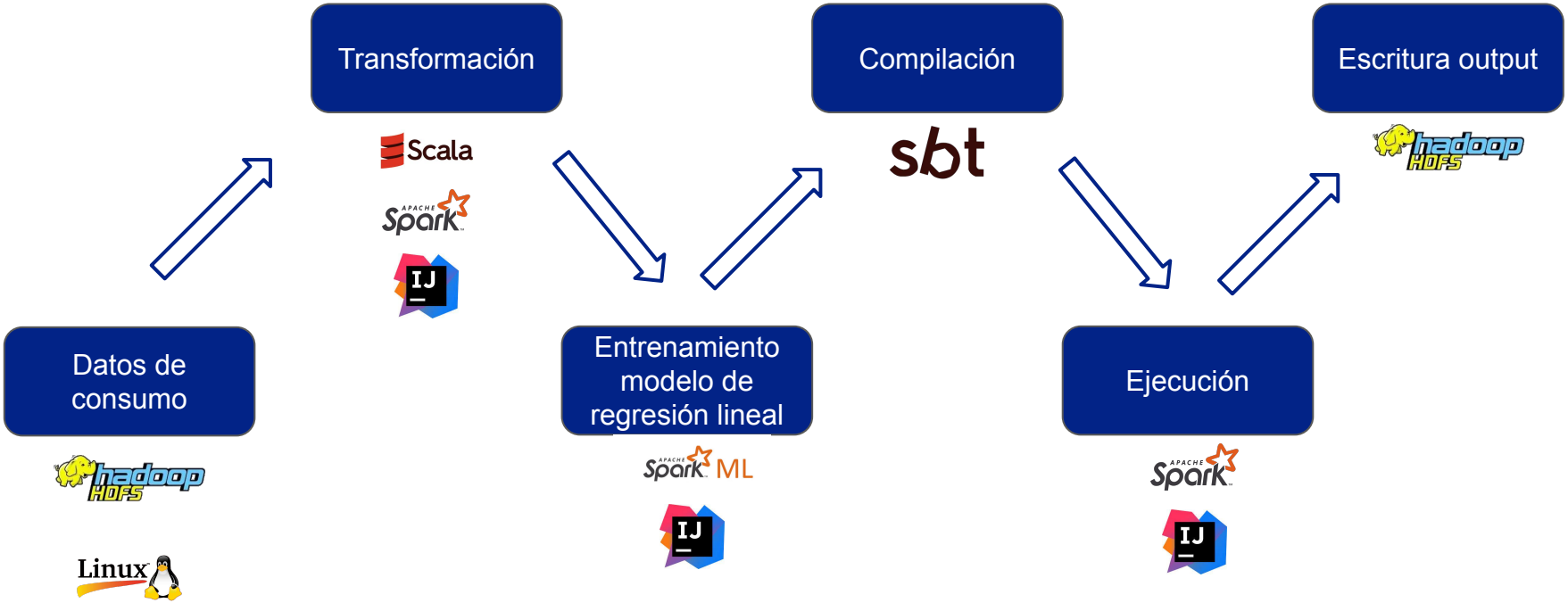
Demo estimación de elasticidad de demanda con Apache Spark

Daniel Bestard Delgado - Lead Data Scientist en Damavis Studio

www.damavis.com | @damavisstudio

Arquitectura de la demo

Arquitectura de la demo



Contenido

Contenido

1. Instalar y configurar **Apache Hadoop** para levantar un **HDFS**
2. Instalar y configurar **Apache Spark** e integrarlo con Apache Hadoop
3. Introducción a **arquitectura de software** aplicada a aplicaciones de **Scala Spark**
4. **Modelo estadístico** de elasticidad de precio
5. Demo entrenamiento modelo de elasticidad en aplicación de **Scala Spark** con **IntelliJ IDEA**
6. Ejecución con **IDE** y **spark-submit**

Apache Hadoop

¿Qué es Apache Hadoop?

Apache Hadoop es un framework open source utilizado para almacenar y procesar big data de forma distribuida y tolerante a fallos

Módulos principales:

- **Hadoop Distributed File System (HDFS)** → almacenamiento de datos
- **Hadoop MapReduce** → procesamiento de datos
- **Apache YARN** → gestión del clúster

Pasos para instalar Apache Hadoop

1. Identificar directorio donde almacenar software de Apache. Generalmente en `/opt`. En esta demo `~/opt`.
2. Buenas prácticas de administrador de sistemas: usuarios específicos lanzar procesos. Por simplicidad se omitirá en esta demo.
3. `wget` del software de Apache Hadoop y descomprimir.
4. Crear link simbólico sin la versión: `ln -s ~/opt/hadoop-3.3.1 ~/opt/hadoop`

Pasos para instalar Apache Hadoop

5. Instalar Java 8: `apt install openjdk-8-jdk -y`
6. Crear directorio `config-files` para desacoplar la configuración de una versión concreta de software
7. Declara la variable de entorno `export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64` en `hadoop/etc/hadoop/hadoop-env.sh` moviendo antes este fichero a `config-files` y creando link simbólico
8. Para evitar replicación de datos en un entorno local, modifica la propiedad `dfs.replication` a 1 en `hadoop/etc/hadoop/hdfs-site.xml`, que una vez más debe ser movido a `config-files` y crear link simbólico

Pasos para instalar Apache Hadoop

9. Especificar que el servicio de HDFS debe ser levantado en local modificando la propiedad `fs.defaultFS` al valor `hdfs://localhost:9000` del fichero `hadoop/etc/hadoop/core-site.xml`, que una vez más debe ser movido a `config-files` y crear link simbólico
10. Comprueba que puedes hacer `ssh localhost`. En caso contrario, copia tu clave ssh pública a `.ssh/authorized_keys` (en caso de no tener clave ssh usa el comando `ssh-keygen` para generarla)
11. Para lanzar scripts de Hadoop desde cualquier directorio, modifica la variable de entorno `PATH` del `.bashrc` añadiendo la localización de los binarios de Hadoop (`hadoop/bin`) y carga los cambios con el comando `source`

Pasos para instalar Apache Hadoop

12. Lanza HDFS ejecutando `hdfs namenode -format` y, posteriormente, ejecutando `hadoop/sbin/start-dfs.sh`
13. Crea directorio de entorno de `pre` y otro de `live` usando `hdfs dfs -mkdir hdfs:///{env}` y dentro crea el directorio `raw`
14. Baja datos de transacciones aquí y descomprime
15. Copia el fichero `region1_company1.csv` a HDFS al directorio `hdfs:///{env}/raw/` usando el comando `hdfs dfs -copyFromLocal ...`

Apache Spark

¿Qué es Apache Spark?

Apache Spark es un framework open source programado en Scala utilizado para computación en clúster para así poder procesar big data de forma distribuida y tolerante a fallos

Modos de lanzamiento:

- **Standalone** → Un solo nodo. Utilizado en la demo de hoy
- **Apache YARN** → Artículos de Damavis, uno de introducción y otro de personalización
- **Kubernetes** → Clúster de contenedores Docker

APIs de programación a parte de Scala:

- Python
- Java
- R

¿Por qué Scala?

No seamos de los que ”para el que tiene martillo todo son clavos”

- Para **Big Data developers** Scala tiene muchas ventajas:
  - Indagar en el código fuente de Spark
  - Sintaxis más intimidatoria que python pero menos que Java o C++
  - Diseñado con paralelismo y concurrencia en mente
  - Usado en el framework de Big Data en tiempo real **Akka**
- Para **científicos de datos** no tantas:
  - En Spark, la ejecución de UDFs (User Defined Functions) es más eficiente en Scala
  - En PySpark se puede requerir más memoria al poder haber ejecuciones fuera de la JVM

Pasos para instalar Apache Spark

1. `wget` del software de Apache Spark sin Hadoop y descomprimir.
2. Crear link simbólico sin la versión: `ln -s ~/opt/spark-3.2.0-bin-without-hadoop ~/opt/spark`
3. Declara la variable de entorno `export SPARK_DIST_CLASSPATH=$(hadoop --config /opt/hadoop/etc/hadoop classpath)` en `spark/conf/spark-env.sh` moviendo antes este fichero a `config-files` y creando link simbólico (previamente hay que borrar `spark/conf/spark-env.sh.template`)
4. Para lanzar scripts de Hadoop desde cualquier directorio, modifica la variable de entorno `PATH` del `.bashrc` añadiendo la localización de los binarios de Hadoop (`spark/bin`) y carga los cambios con el comando `source`
5. Lanza `spark-shell` y lee el fichero que hemos copiado en HDFS con el comando de scala `val df = spark.read.option("header", "true").csv("hdfs://localhost:9000/pre/raw/region1_company1.csv")`

Arquitectura de software aplicada a aplicaciones de Scala Spark

Buenas prácticas en el desarrollo de software

- Principios **SOLID**:
  - **S** → Single responsability
  - **O** → Open closed
  - **L** → Liskov substitution
  - **I** → Interface segregation
  - **D** → Dependency inversion
- Uso de **IDE**:
  - Scala/Java → Intellij IDEA
  - Python → Pycharm

Buenas prácticas en el desarrollo de software

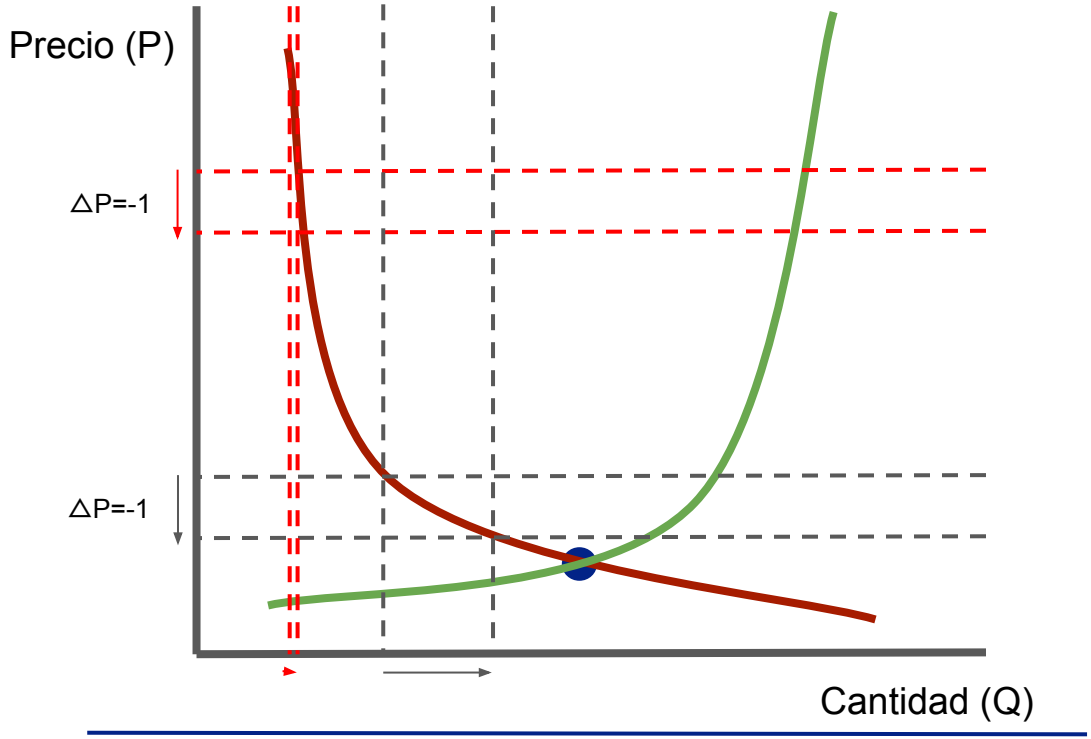
- Gestión de **entornos**:
  - PRE → entorno de prueba y testeo
  - LIVE → entorno de producción
- **Testing** (omitido en esta demo)
  - Es buena práctica tener un alto *code coverage*
- Pasar keys y secrets como **variables de entorno**

Ejemplo arquitectura de software

- `src`
  - `main`
    - `resources` → ficheros de configuración con extensión `.conf`
    - `scala`
      - `com…`
        - `stage` → contiene la lógica de negocio. Se le pasan los datos, por lo que no interacciona ni con `repository` ni con `resource`.
        - `pipeline` → se le pasa el `repository` y los `stages`, los cuales se ejecutan en el orden especificado en esta clase
        - `repository` → usando el `resource` lee/escribe datos concretos de la aplicación.
          - `resource` → lectura y escritura en fuentes. Ajeno a lógica de negocio, es decir, no sabe que lee/escribe, solo sabe de formatos.
      - `main`
  - `test`
- `build.sbt` → Dependencias del aplicativo
- `.gitignore`
- `.scalafmt.conf` → Configuración para formatear el código

Cálculo de elasticidad de demanda

Modelo Demanda Multiplicativo



Ref: 1992, W. Nicholson, Microeconomic Theory: Basic Principles and Extensions

$$Q = A * P^{-b}$$

donde **A** y **b** son parámetros

$$\log(Q) = C - b * \log(P)$$

Interpretación Modelo de Demanda Multiplicativo

$$Q = A * P^{-b}$$

$$\log(Q) = \log(A) - b * \log(P)$$

$$\log(Q_1) - \log(Q_0) = [\log(A) - b * \log(P_1)] - [\log(A) - b * \log(P_0)]$$

$$\log(Q_1) - \log(Q_0) = b * [\log(P_1) - \log(P_0)]$$

$$b = [\log(Q_1) - \log(Q_0)] / [\log(P_1) - \log(P_0)]$$

$b \approx \Delta\%Q / \Delta\%P$

Ref: 1992, W. Nicholson, Microeconomic Theory: Basic Principles and Extensions

Si $A = 1*10^7$ y $b = -3 \rightarrow$ ¿$\triangle Q$, si $P_0=100$ y $P_1=105$?

1

Cálculo Exacto

$Q_0 = (1 * 10^7) * 100^{-3} = 10$

$Q_1 = (1 * 10^7) * 105^{-3} = 8.64$

$Q_1 - Q_0 = -1.36$

$\triangle\%Q = -1.36/10 = -13.6\%$

2

Cálculo Aproximado

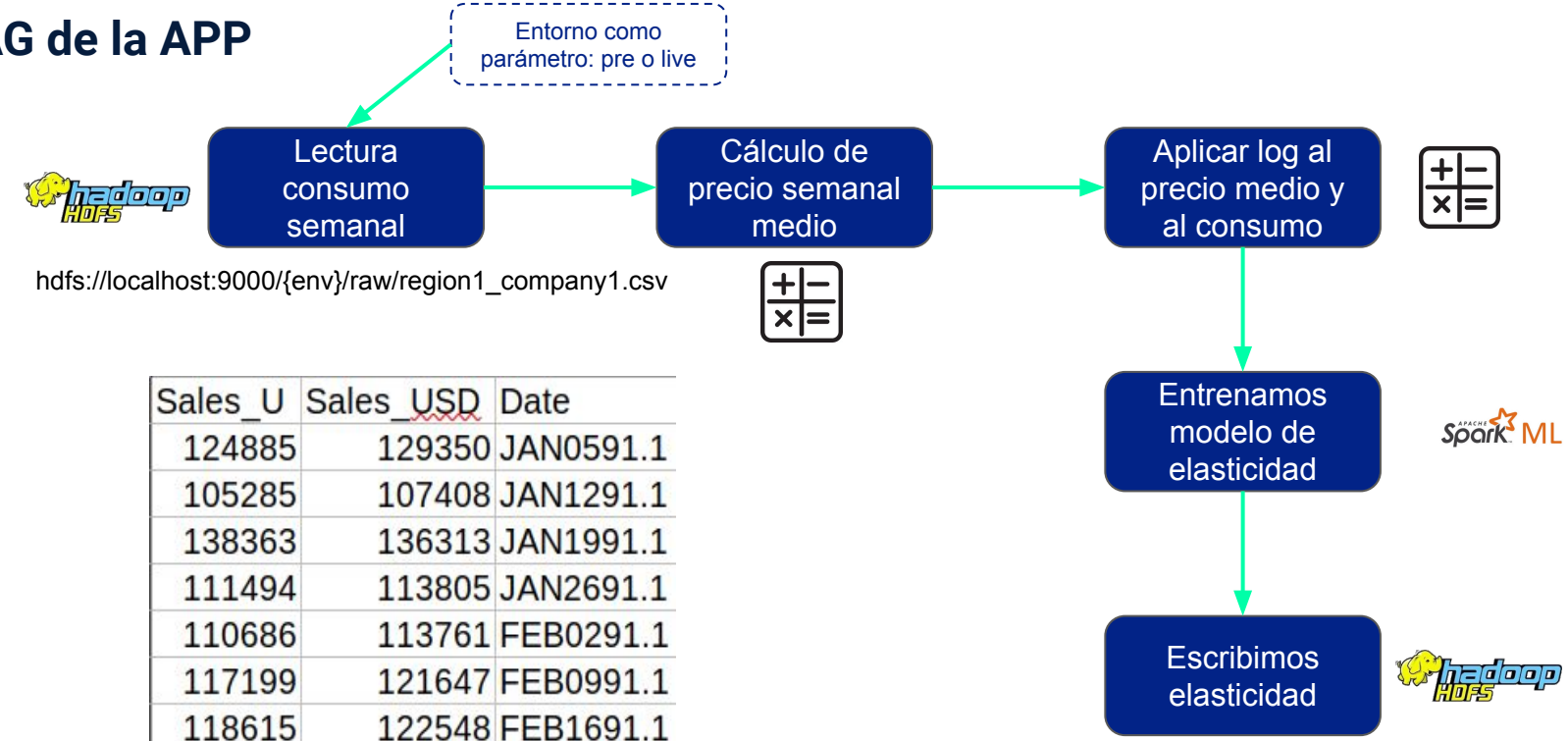
$\triangle\%P = (105 - 100) / 100 = 5\%$

$-3 \approx \triangle\%Q / 5\%$

$\triangle\%Q \approx -15\%$

DAG de la aplicación Scala Spark

DAG de la APP



Sales_U	Sales_USD	Date
124885	129350	JAN0591.1
105285	107408	JAN1291.1
138363	136313	JAN1991.1
111494	113805	JAN2691.1
110686	113761	FEB0291.1
117199	121647	FEB0991.1
118615	122548	FEB1691.1
118847	125338	FEB2391.1

Demo aplicación Scala Spark

Nos bajamos el código de aquí y lo abrimos con **Intellij IDEA**

*Nota: necesitaremos **sbt,** que es un software open source de build de proyectos scala parecido a Apache MAVEN. Instrucciones de instalación aquí

Ejecución en IDE

Ejecución en IDE

- El script requiere de la variable de entorno `MASTER` de Spark
  - Al usar el modo standalone de spark especificamos el valor `local[*]`
- Para ejecutar desde el IDE se deben bajar las dependencias de Apache Spark, por lo que se debe quitar `% Provided` (el Spark instalado en local no es usado).
- Ejecutamos con el comando `runMain main.Main --environment pre` (comando sbt)

Ejecución con Spark submit

Spark submit

- `sbt assembly` genera un paquete `.jar` que contiene toda la aplicación
- Compilamos con las dependencias de Spark como `Provided` dado que emplearemos la instalación que hemos hecho en local, excepto la de Hive, la cual no está contenida en las dependencias del Spark configurado.

```
spark-submit --class main.Main --master local[*] {JAR} --environment pre
```