From Relational to Hadoop

Part 2: Sqoop, Hive and Oozie

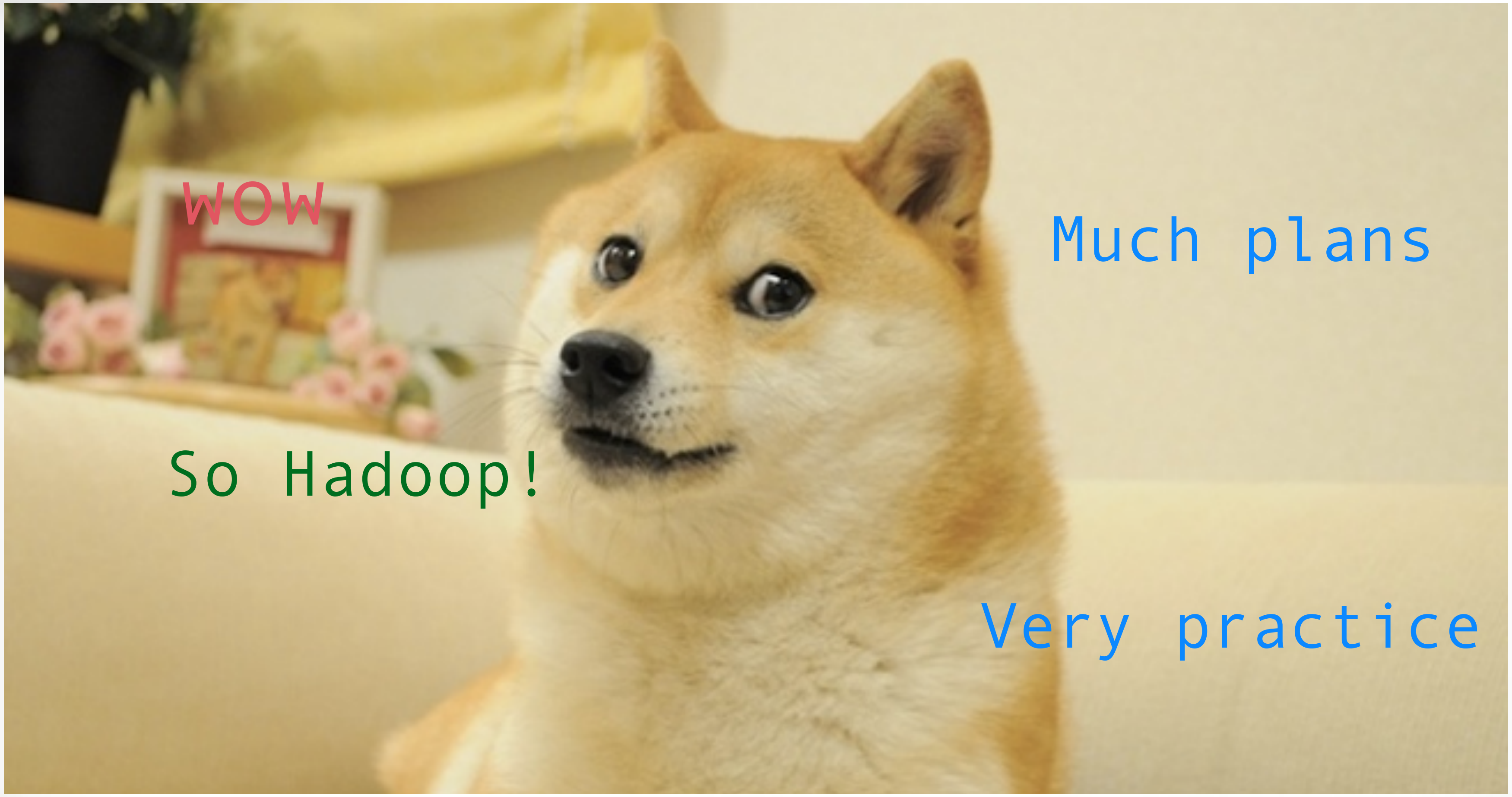
Gwen Shapira , Cloudera
and
Danil Zburivsky, Pythian

Previously we...

- Loaded a file to HDFS
- Ran few MapReduce jobs
- Played around with Hue

Now its time to

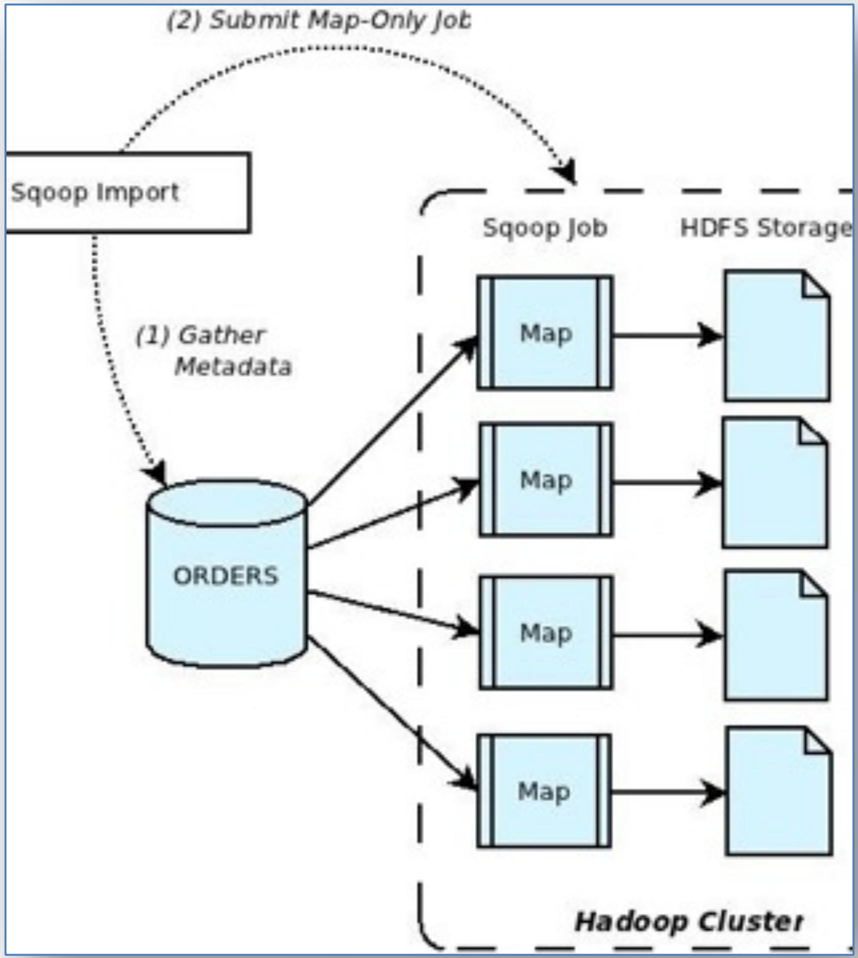
- Use Sqoop to extract data from MySQL
- Create Hive table on JSON data
- Join the two data sets
- Run few aggregations
- Load data back to MySQL
- Sync everything through Oozie


wow
Much plans
So Hadoop!
Very practice

Important things about Sqoop

Reminder:

- Sqoop gets metadata
- Sqoop starts mappers
- Each mapper runs a query
- writes the result to HDFS



Sqoop is customizable

- You can use a custom query
- You can tune number of mappers (careful!)
- You can tune the data split
- You can use direct connectors for your DB

Sqoop vs Sqoop2

Sqoop

- Production
- Client app
- Plain text password
- Lots of connectors

Sqoop2

- Beta
- Client-server
- Can store jobs, configs
- Secure
- No connectors yet

Lets Sqoop

Step 2 - Import with Sqoop

1. Start MySQL
2. List databases using Sqoop
3. List tables in database "world" using Sqoop
4. Load “cities_coords” table to HDFS
5. Load “cities_coords” table to Hive
6. Load “cities” table to Hive using Direct Connector

Tips

- Sudo service mysqld start
- “sqoop help” will give you list of commands
- “sqoop import” is used to get data from the DB to Hadoop
- Take a peek at ~/pl_tutorial/solutions/ex3-sqoop.txt
  and check the JDBC connection string
- If /data doesn’t exist – either use an existing dir, or create it.

No worries

1. `sqoop list-databases --connect jdbc:mysql://localhost --username root`
2. `sqoop list-tables --connect jdbc:mysql://localhost/world --username root`
3. `sqoop import --connect jdbc:mysql://localhost/world --username root --table cities_coords --warehouse-dir /data/world`
4. `sqoop import --connect jdbc:mysql://localhost/world --username root --table cities_coords --hive-import`
5. `sqoop import --connect jdbc:mysql://localhost/world --username root --table City --warehouse-dir /data/world2 --direct`

About Hive

Hive - Not 100% SQL

Extras

- Parallelism (duh)
- Unstructured data
- Columnar + Compression
- Buckets (hash-partition)
- Explode
- Lateral-view
- Nested data types
- Xpath
- Cube rollup

Missing

- Support for subqueries
- Non-equi-joins
- Limited date-time support
- No updates/deletes

Never Metadata I didn’t like

- Hive has a metastore
- Recommended: Put metastore in MySQL
- Translate files into tables
- Any file on HDFS can be a Hive table
- External tables:
  - Datafile itself is not managed by Hive

SerDe

- Instruct Hive on how to read/write data for a certain file format
- JSON, CSV, XML, Avro, Parquet
- SerDes are JAR files. Use “ADD JAR” to load the JAR in Hive

Hive Practice

1. Add the SerDe jar to the classpath using "add jar". List jars to validate.
2. Create an external table for accessing the earthquake data in HDFS using the JSON SerDE
3. Validate the table by selecting the first two rows.
   Then find the avg magnitude of earthquakes in our file
4. Find 10 most populated cities from our cities_coords table in Hive

Extra Credit!

5. Find top 10 cities and their population which have largest earthquakes in our data set. Select only cities that have population of more than 100 people.

Tips:

a. Use properties.place field to find which city is closest to eqrthquake epicenter

b. Use Hive regexp_extract to get city and state from string. Check Hive docs for examples -- https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF

c. Join cities_coors table to get population

Create Table

```
Add jar '….';
Create external table earthquakes (
      type string,
      properties STRUCT<
            mag:string,
…..
)
ROW FORMAT SERDE 'com.cloudera.hive.serde.JSONSerDe'
LOCATION '/etl/earthquakes/landing'
```

No need to load data

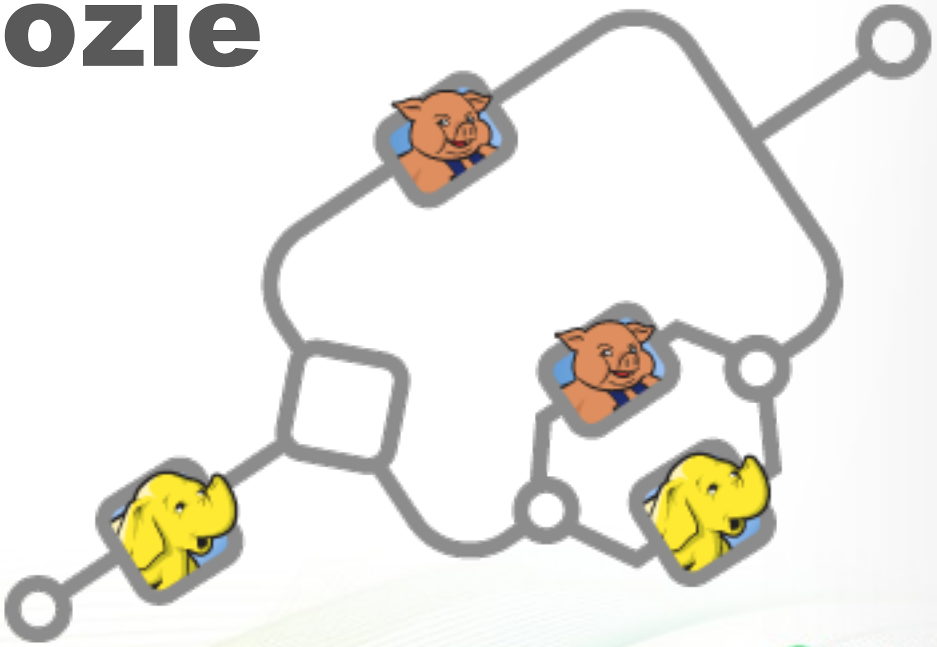
That’s the hadoop magic. Just query the JSON:

- `Select * from earthquakes`
- `select avg(properties.mag) from earthquakes`

- select * from earthquakes limit 2;
- select avg(properties.mag) from earthquakes;

- select name, population from city order by population desc limit 10;

About Oozie



Oozie

- Workflow orchestration
- Why not just use … ?

Oozie + Hue

- The easy way to use Oozie:
  - Use Hue to create workflows
  - Drag and drop actions

Oozie the Hard Way

- Workflows are XMLs

```
<workflow-app xmlns="uri:oozie:workflow:0.2" name="map-reduce-wf">
    <start to="mr-node"/>
    <action name="mr-node">
        <map-reduce>
            <job-tracker>${jobTracker}</job-tracker>
            <name-node>${nameNode}</name-node>
            <configuration>
.....
            </configuration>
        </map-reduce>
        <ok to="end"/>
        <error to="fail"/>
    </action>
    <kill name="fail">
        <message>Map/Reduce failed, error message[$
{wf:errorMessage(wf:lastErrorNode())}]</message>
    </kill>
    <end name="end"/>
</workflow-app>
```

Important Notes

- Everything a workflow needs must be in HDFS
- Job.properties configures everything Oozie client needs
- Watch job status / errors in Hue
- Oozie is best learned from example
  - So feel free to “peek” and play with the provided solution

Oozie Practice

0. Before we start: Drop the Hive table "City". We will be importing it again.

1. Oozie workflows can be executed from any node in the cluster. Therefore everything the workflow will need should be on HDFS.

Create /user/cloudera/workflow directory in HDFS and in it:
(see the list on the exercise text file)
2. create job.properties file - this file should include:
(see the list on the exercise text file)

3. Create workflow.xml file with the following workflow actions:
- Sqoop action to create Hive cities table
- Hive action to create external earthquakes table
- Hive action to create "top 10 most populated" cities

4. Copy workflow.xml to the workflow directory in HDFS.
And use oozie to execute the workflow

Parting Words