BUSINESS INTELLIGENCE AND DATA ANALYTICS
– WE PREDICT THE FUTURE –

University of Petra

Institute of Analytics

University of Petra

307401
Big Data and Data Warehouses

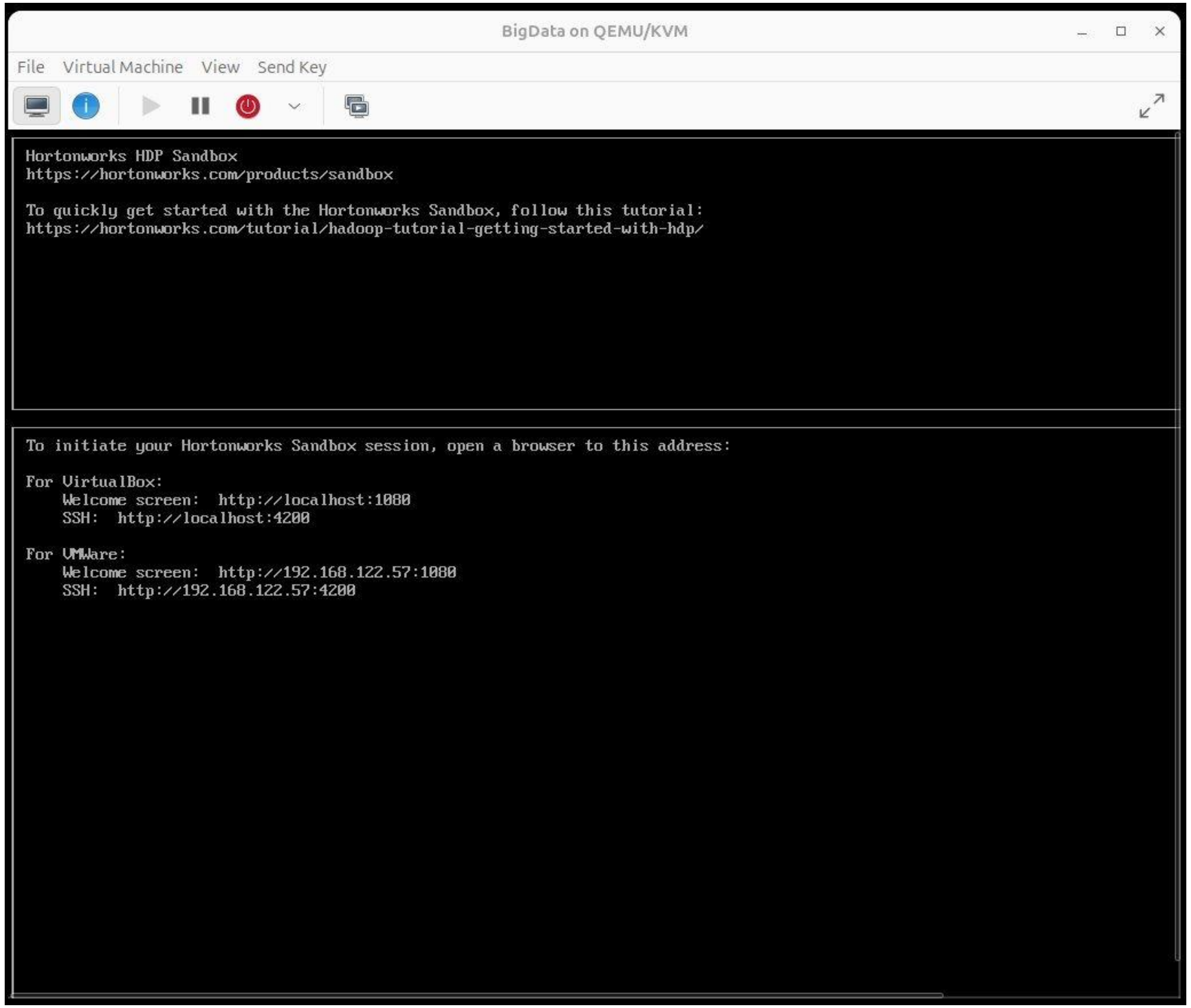
Introduction to Hadoop – Practical Examples

hadoop

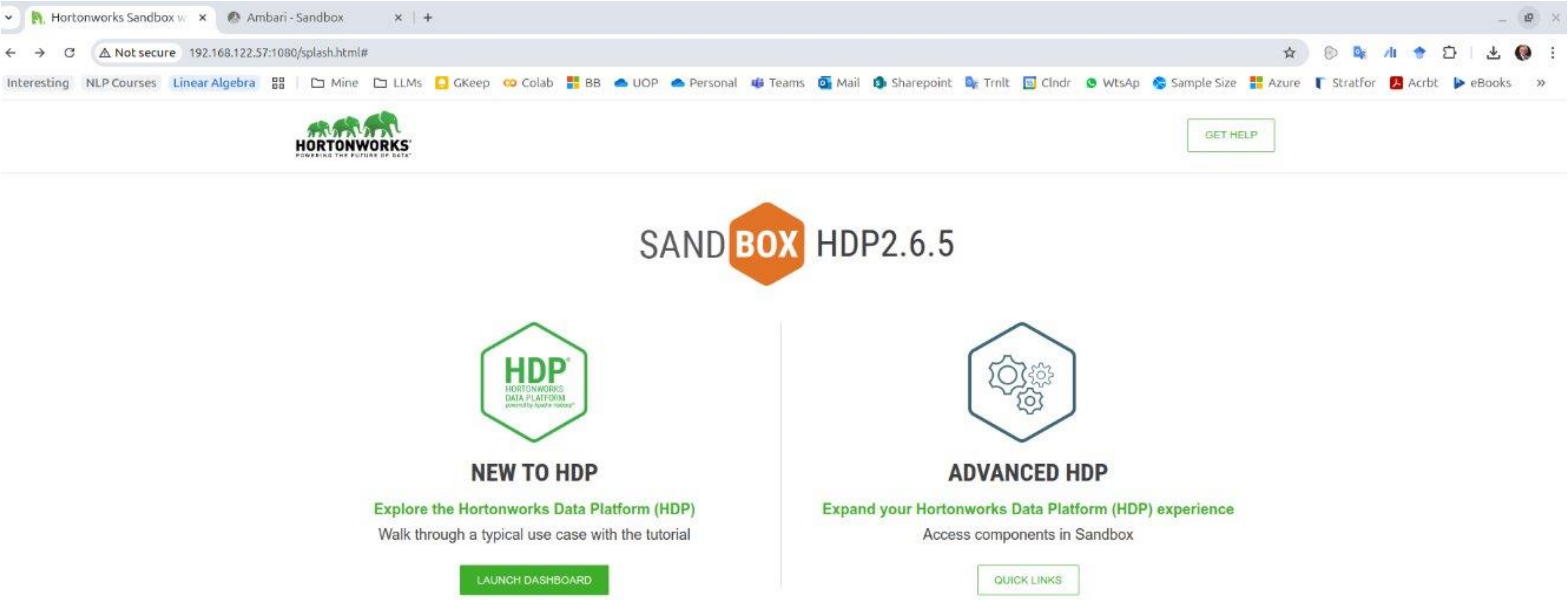
Hands On Practice using Horton Works Virtual Machine

Prepare Test Environment

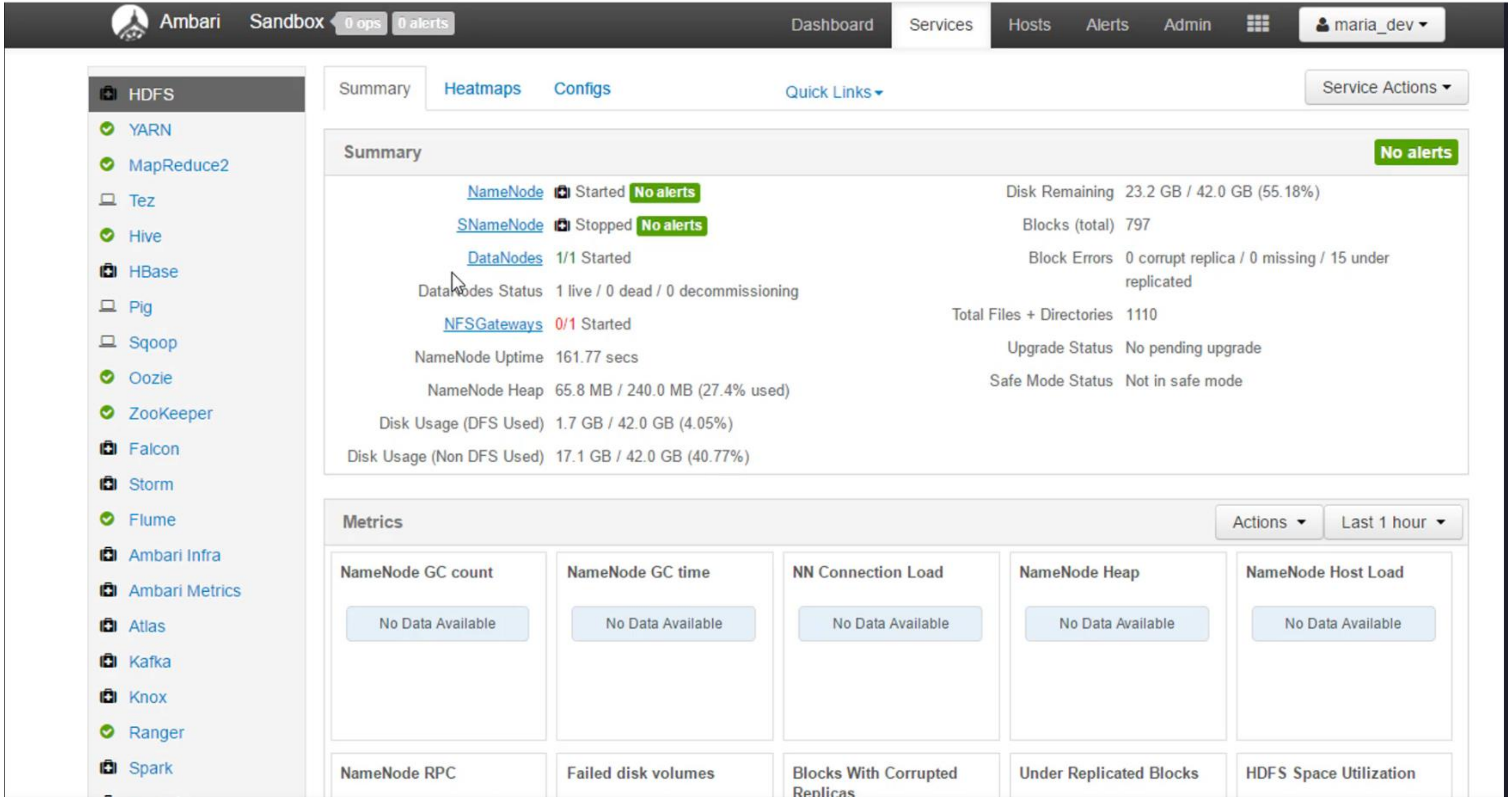
1. Download Virtual Box: https://www.virtualbox.org/wiki/Downloads
2. Or Works virtual machine Version 2.6.5: https://archive.cloudera.com/hwx-sandbox/hdp/hdp-2.6.5/HDP_2.6.5_virtualbox_180626.ova
3. Import HortonWorks (.ova file) you downloaded in step 1 into Virtual Box.
4. Open your browser, goto: http://localhost:1080, launch HORTONWORKS main page.



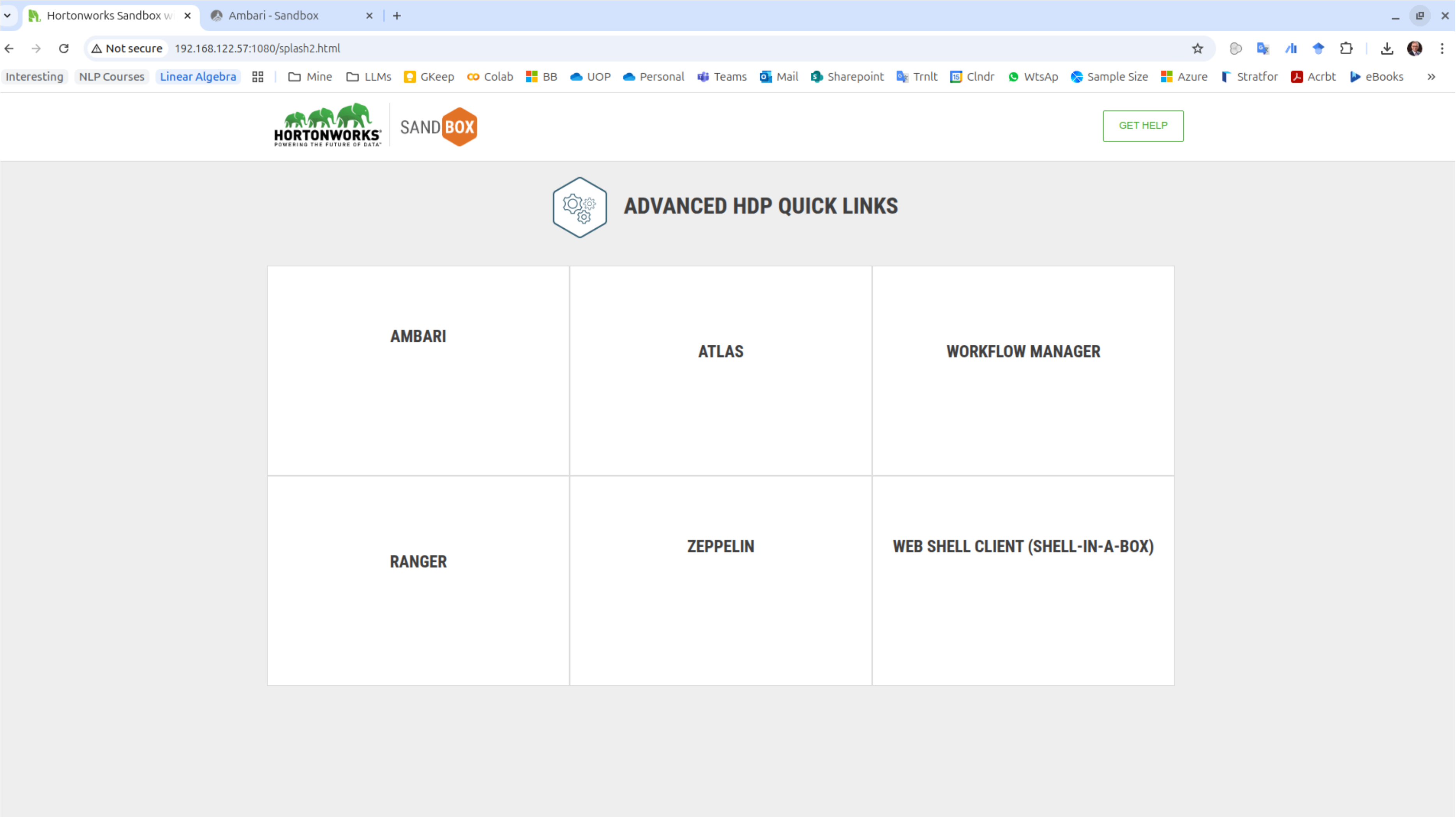
Main Sand Box Web Page – localhost:1080 (port 1080)



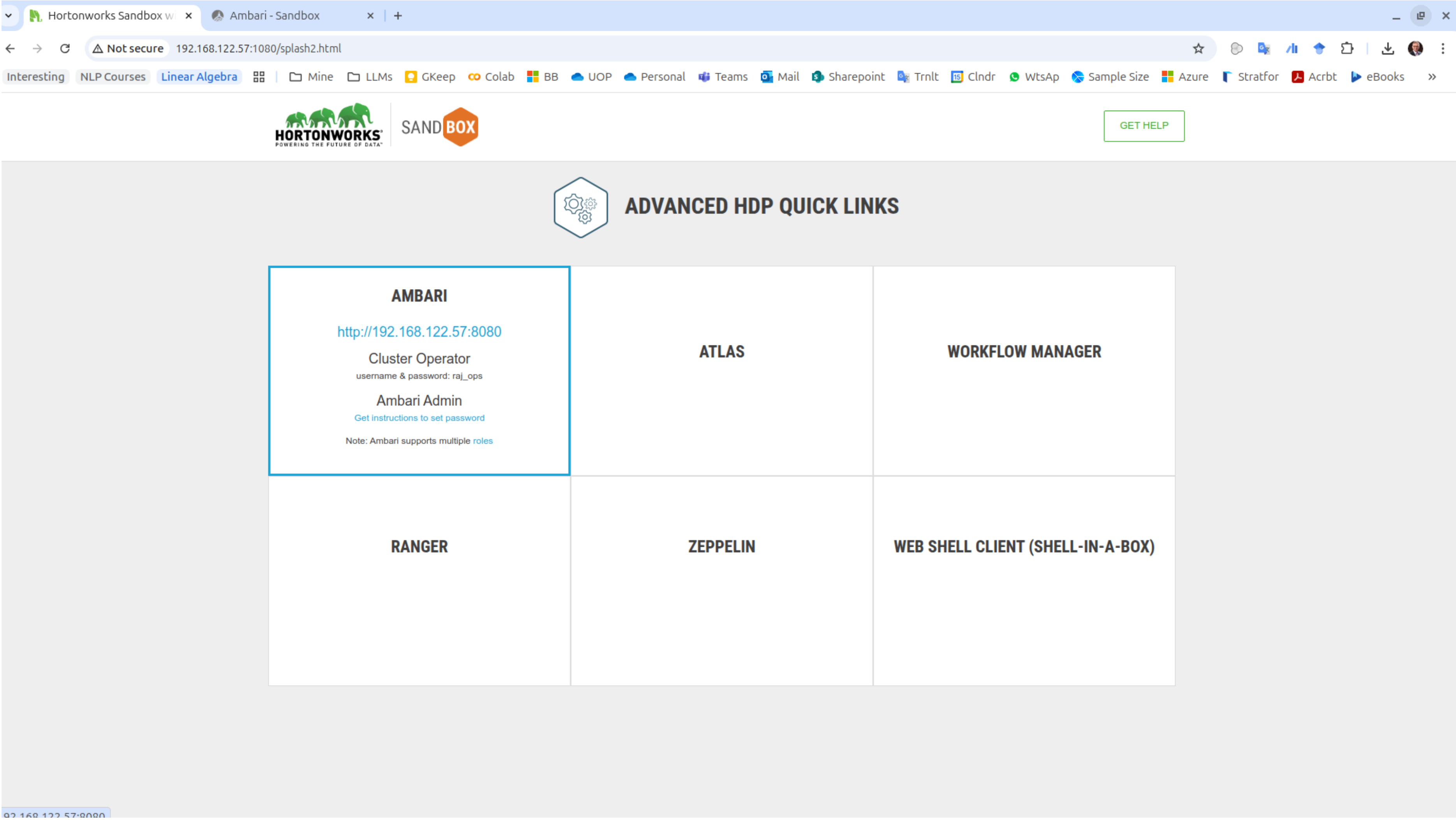
Ambari Dashboard



Advanced HDP Page



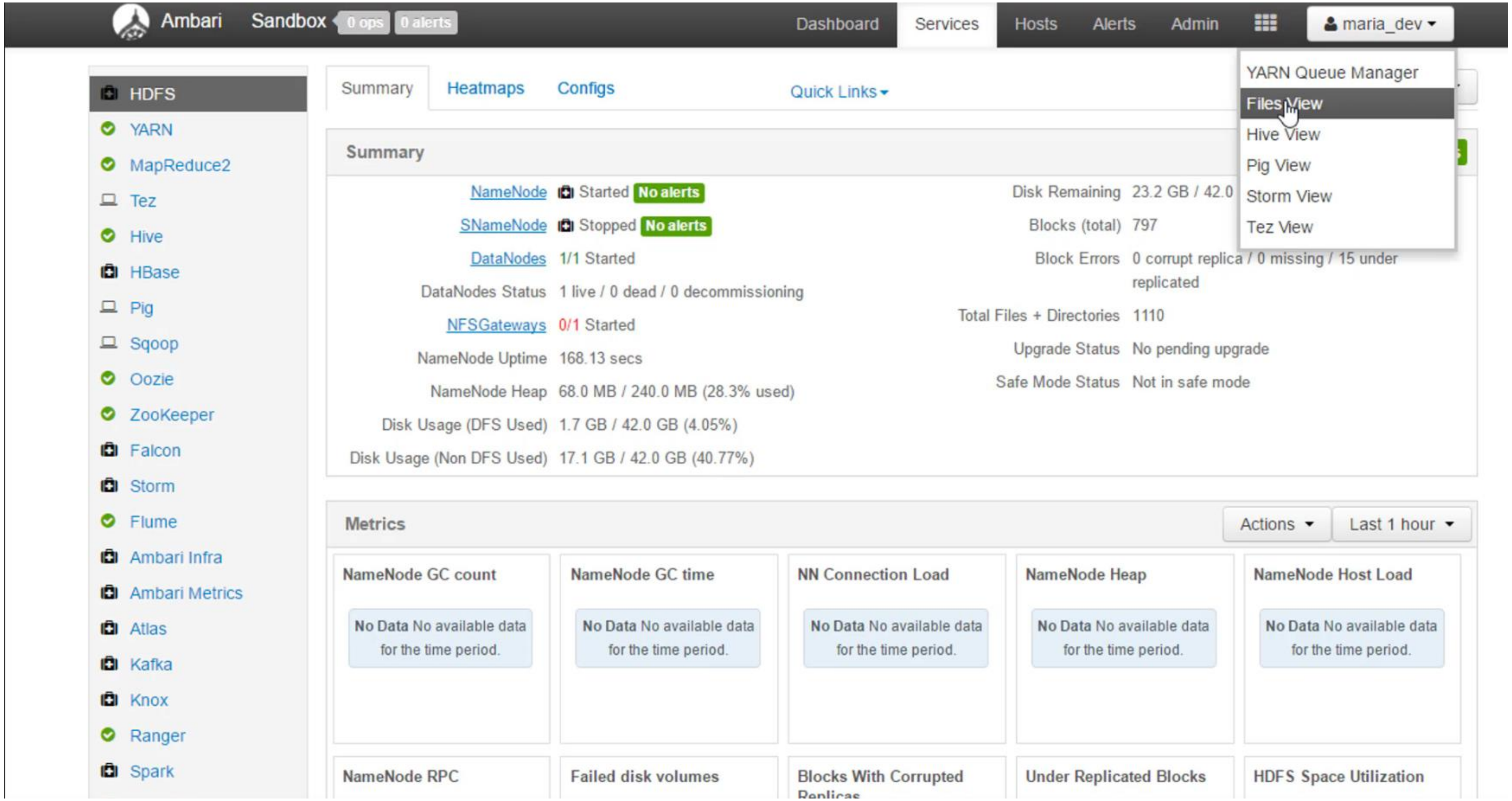
Open Ambari Dashboard



Practical hadoop HDFS

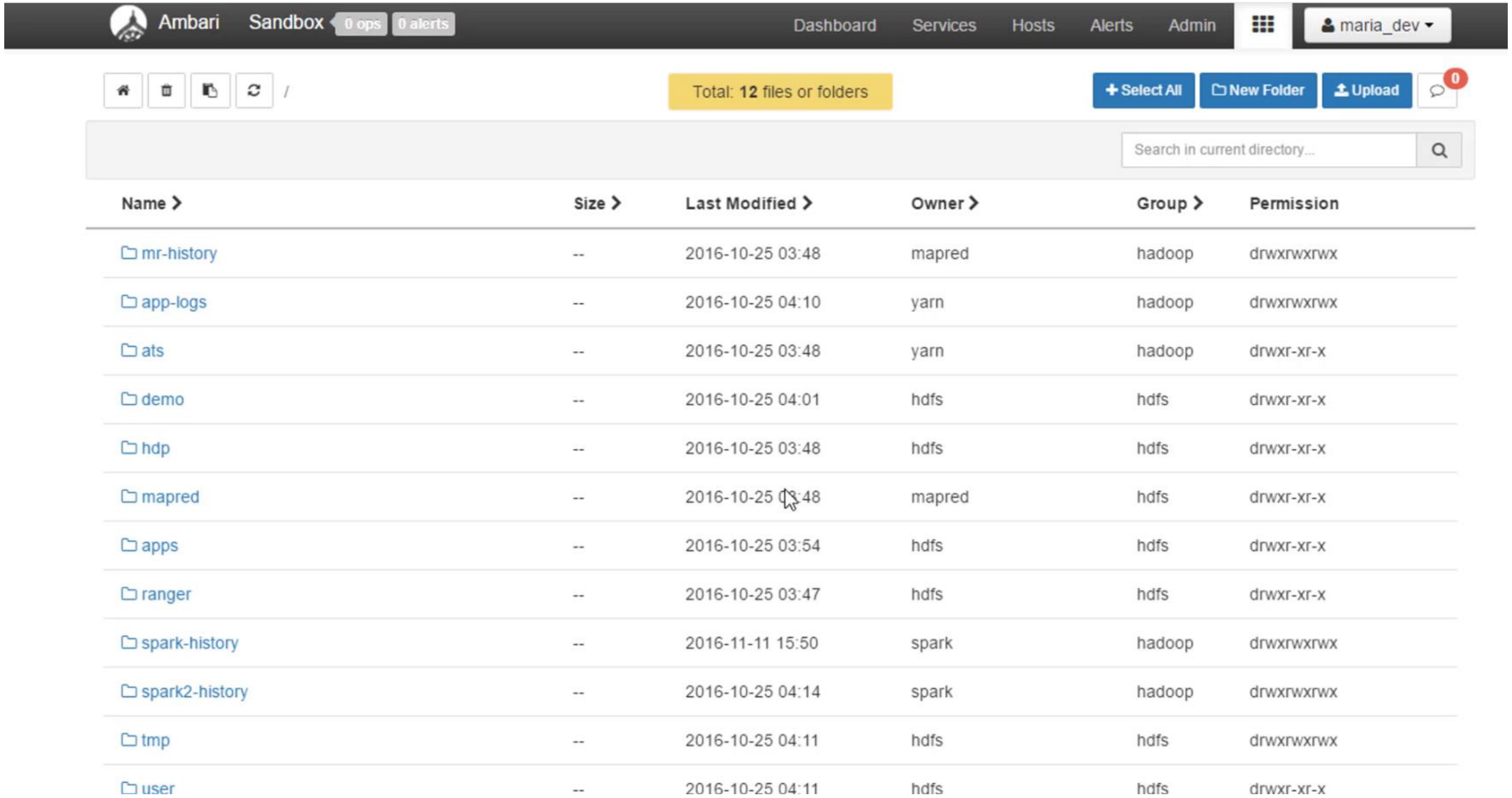
Testing HDFS

Select File View in Ambari



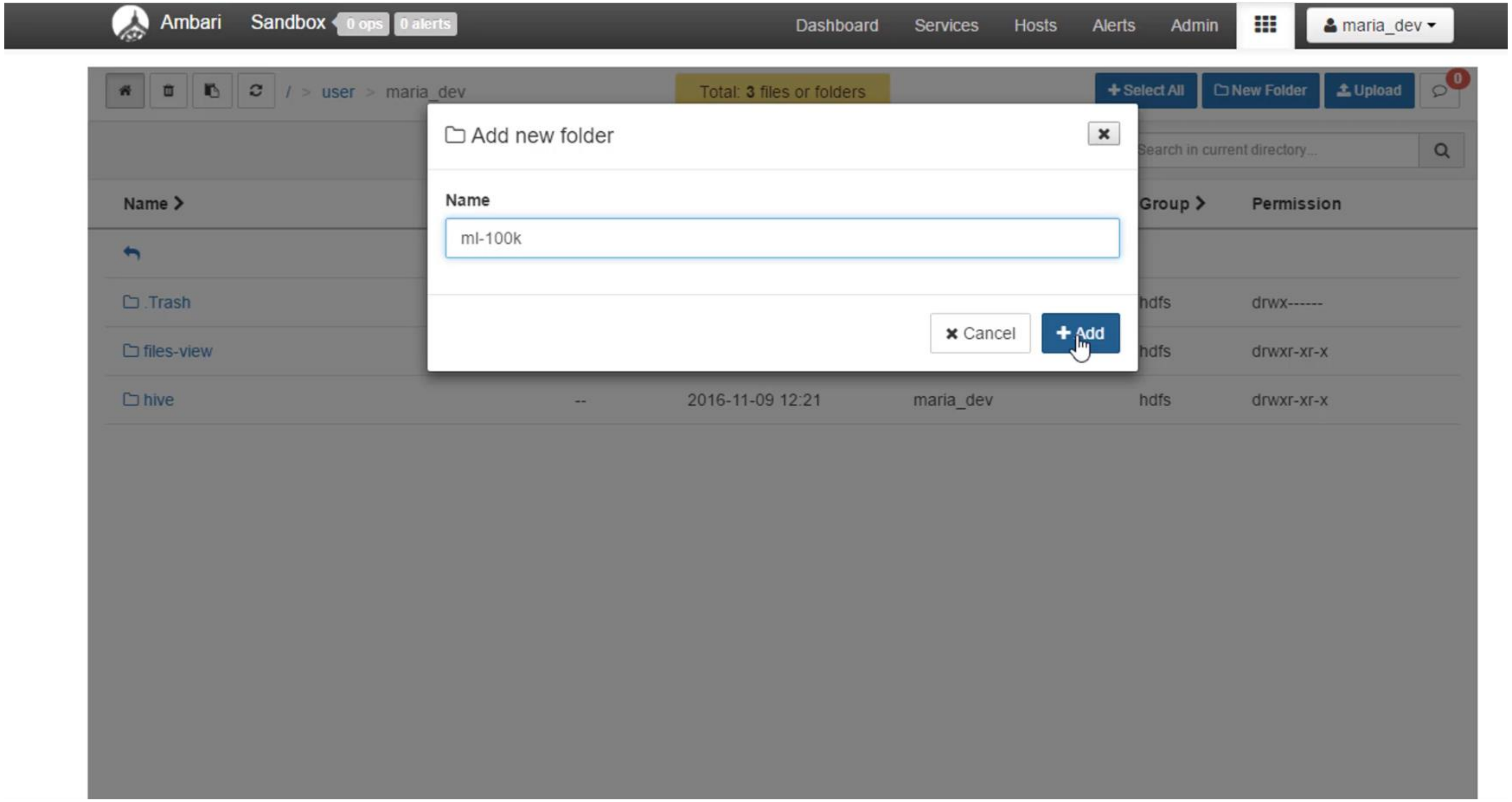
HDFS File System

Selecting file views displays the entire file system in a graphical manner



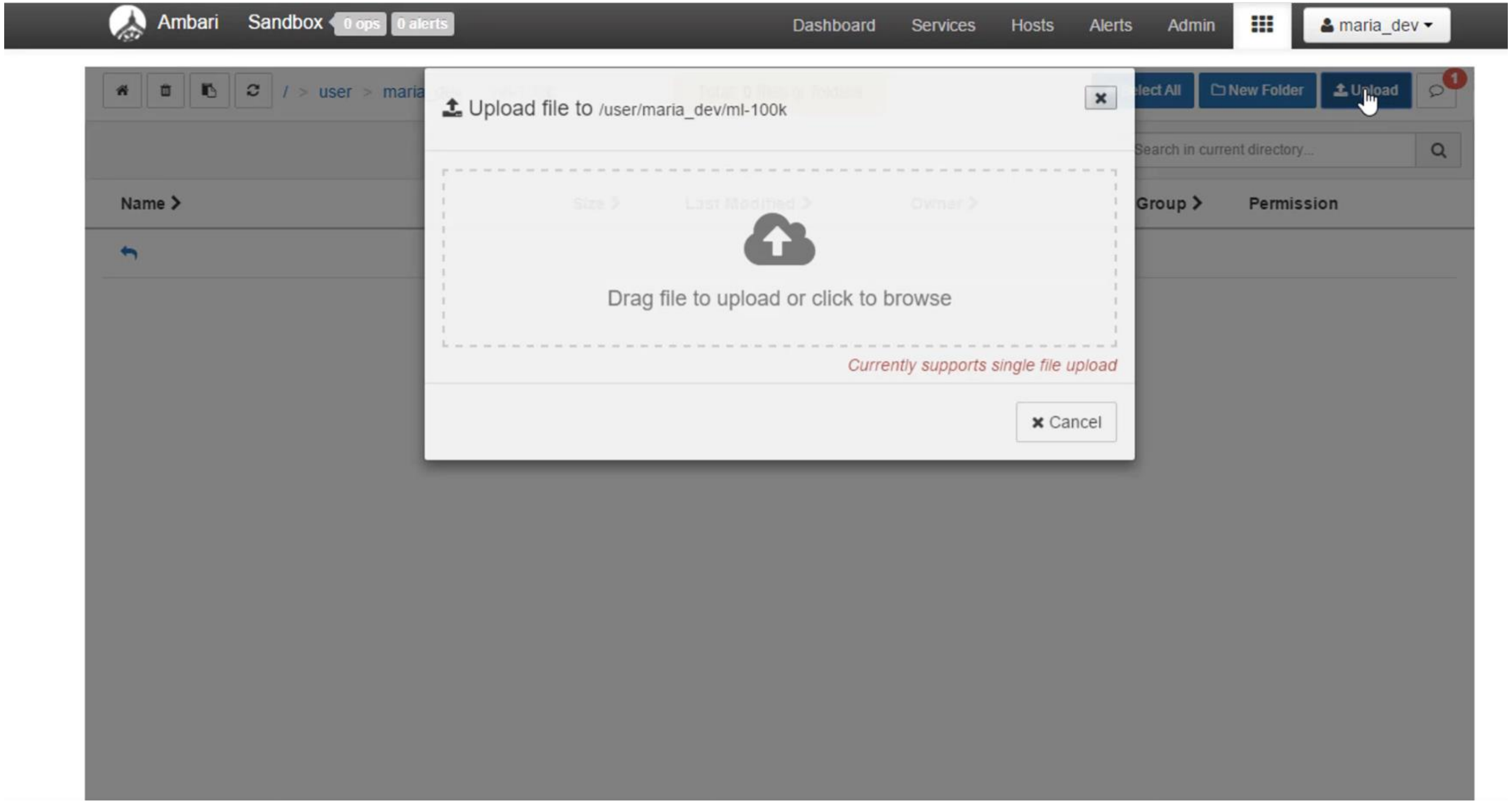
HDFS File System

We can create a new folder to store our files.



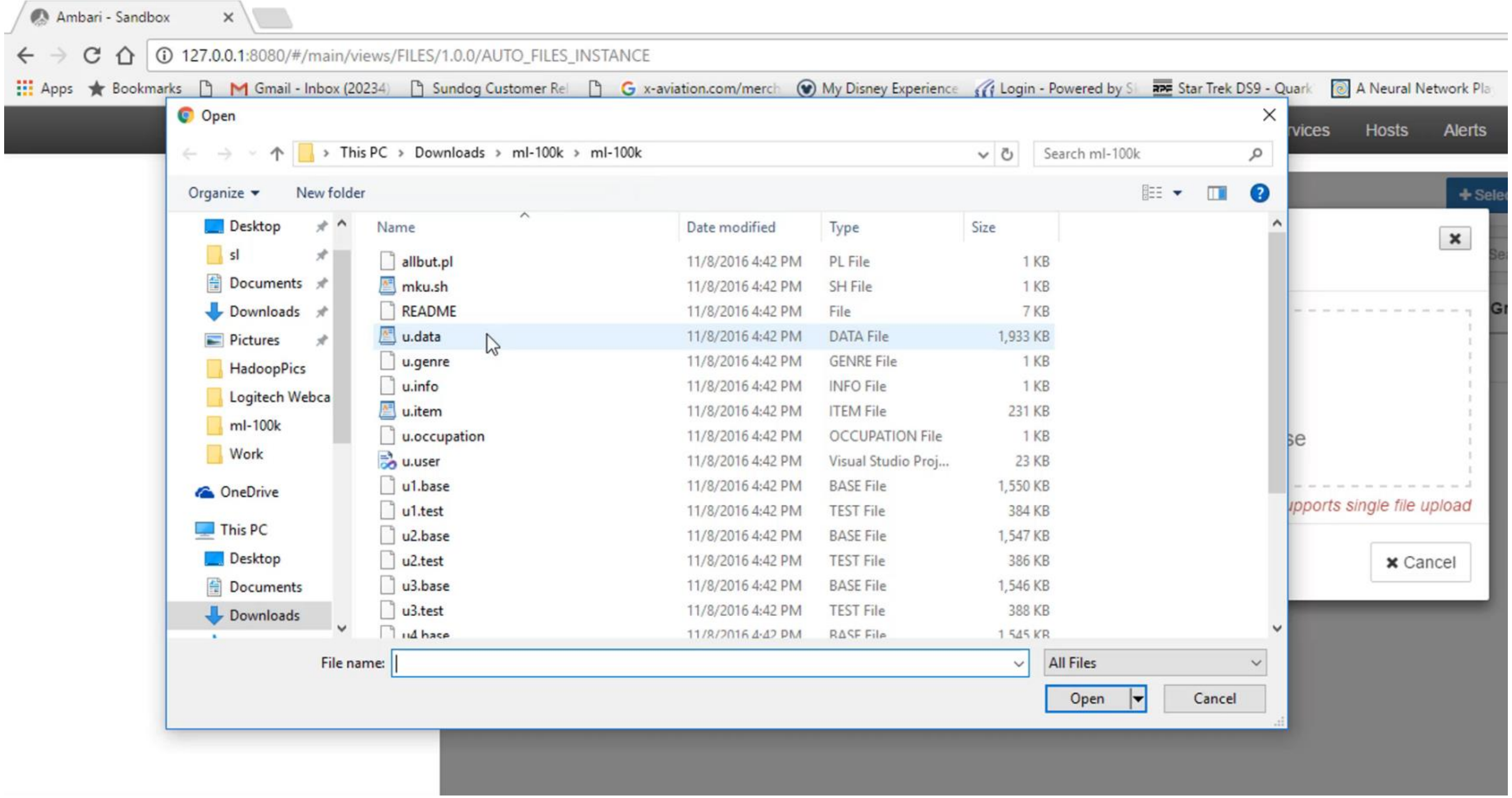
HDFS File System

Now we can upload our data to the Hadoop cluster



HDFS File System

One we upload a file, HDFS distribute it across our cluster (Data Nodes) depending on file size and cluster size

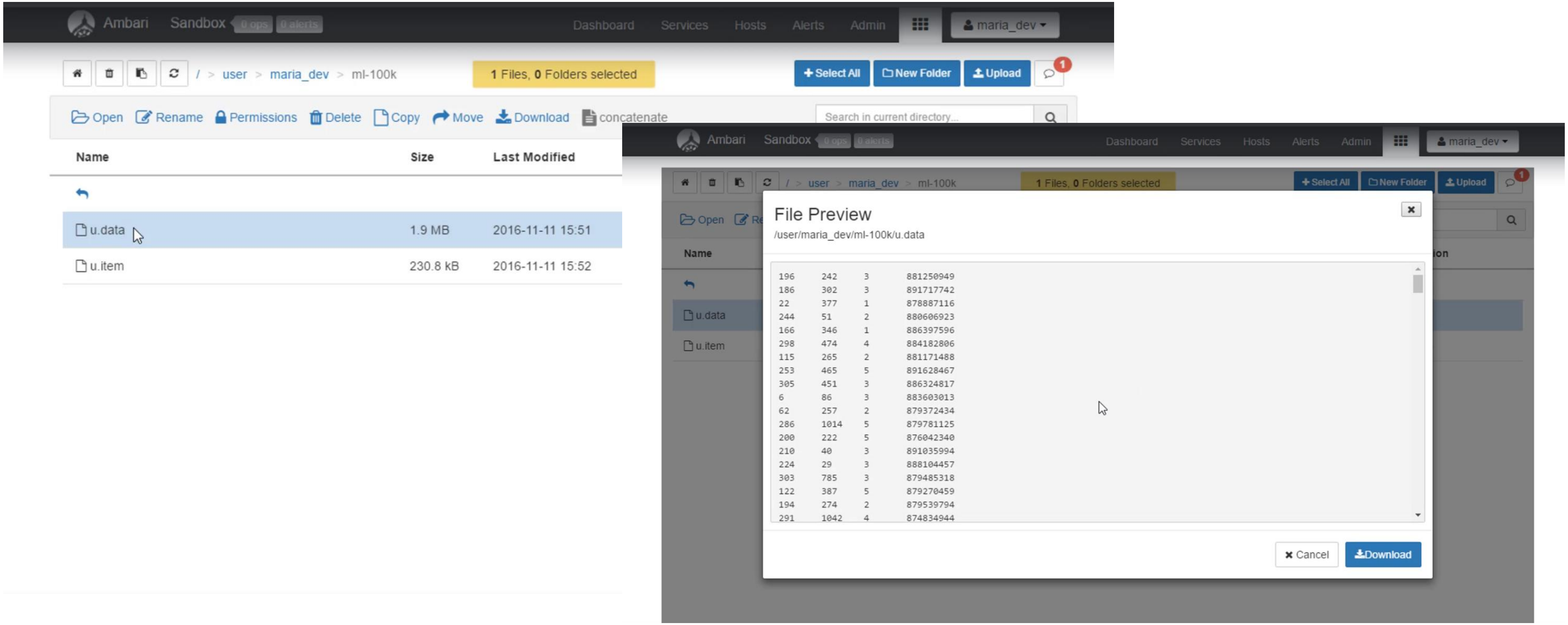


BUSINESS INTELLIGENCE AND DATA ANALYTICS

Institute of Analytics Recognized Programme

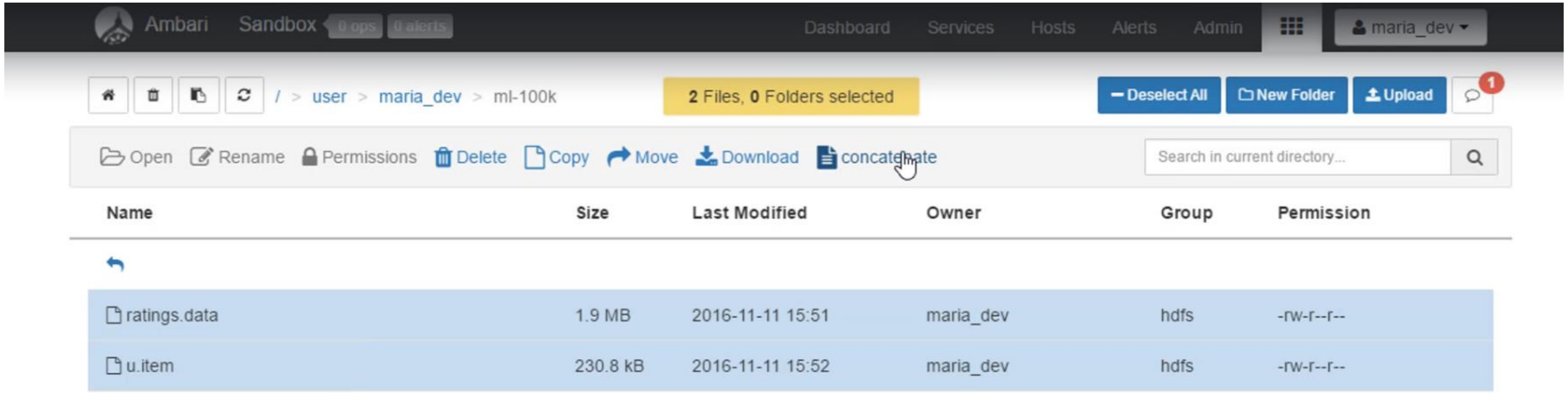
HDFS File System

We can do file operations using Ambari interface (Open, Rename, Change Permissions, Delete, Copy, Download…etc.)



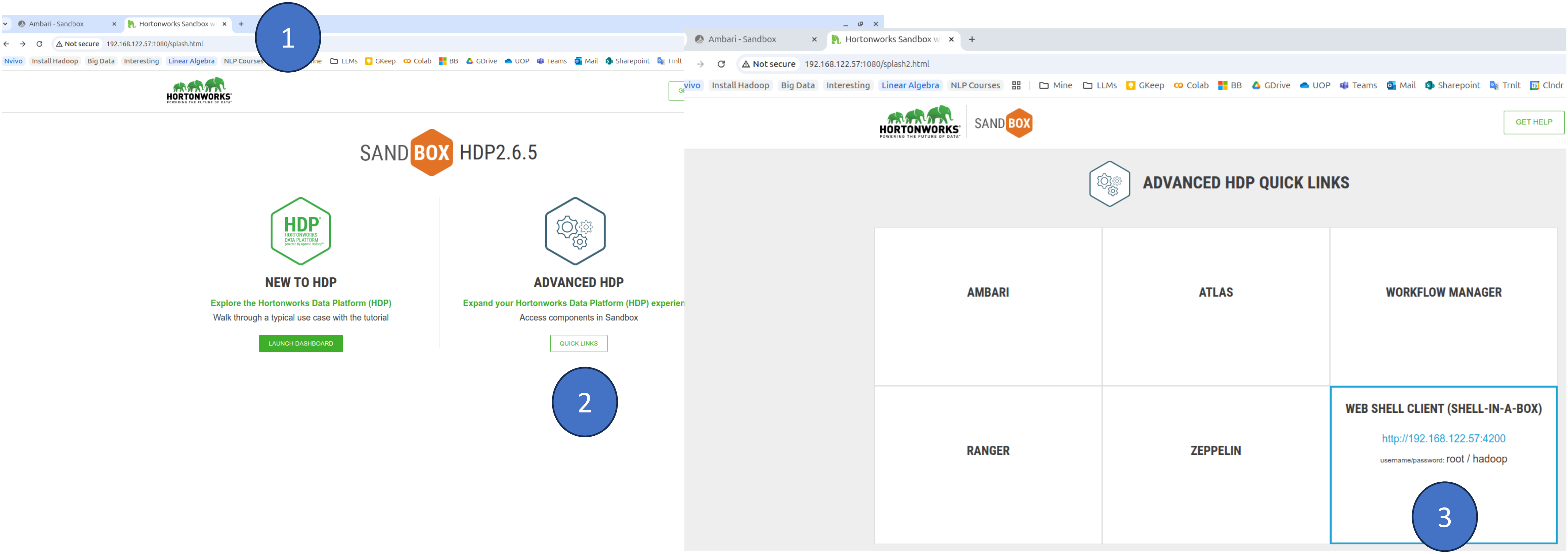
HDFS File System

We can even concatenate multiple files to download them together



HDFS Command Line Access

Open HortonWorks Sandbox (usually http://localhost:1080, Open Advanced HDP Quick Links Page, Open Web Shell



Basic HDFS Commands

1. Check HDFS directory structure

```bash
hdfs dfs -ls /
```

2. Create a directory in HDFS

```bash
hdfs dfs -mkdir /user/pig/logs
```

3. Upload a local file to HDFS

```bash
hdfs dfs -put access_log.txt /user/pig/logs/
```

4. List contents of a directory

```bash
hdfs dfs -ls /user/pig/logs
```

5. View contents of a file

```bash
hdfs dfs -cat /user/pig/logs/access_log.txt
```

6. Copy a file from HDFS to your local machine

```bash
hdfs dfs -get /user/pig/logs/access_log.txt ./
```

7. Remove a file or directory

```bash
hdfs dfs -rm /user/pig/logs/access_log.txt
hdfs dfs -rm -r /user/pig/logs
```

Check Disk Usage

```bash
hdfs dfs -du -h /user/
```

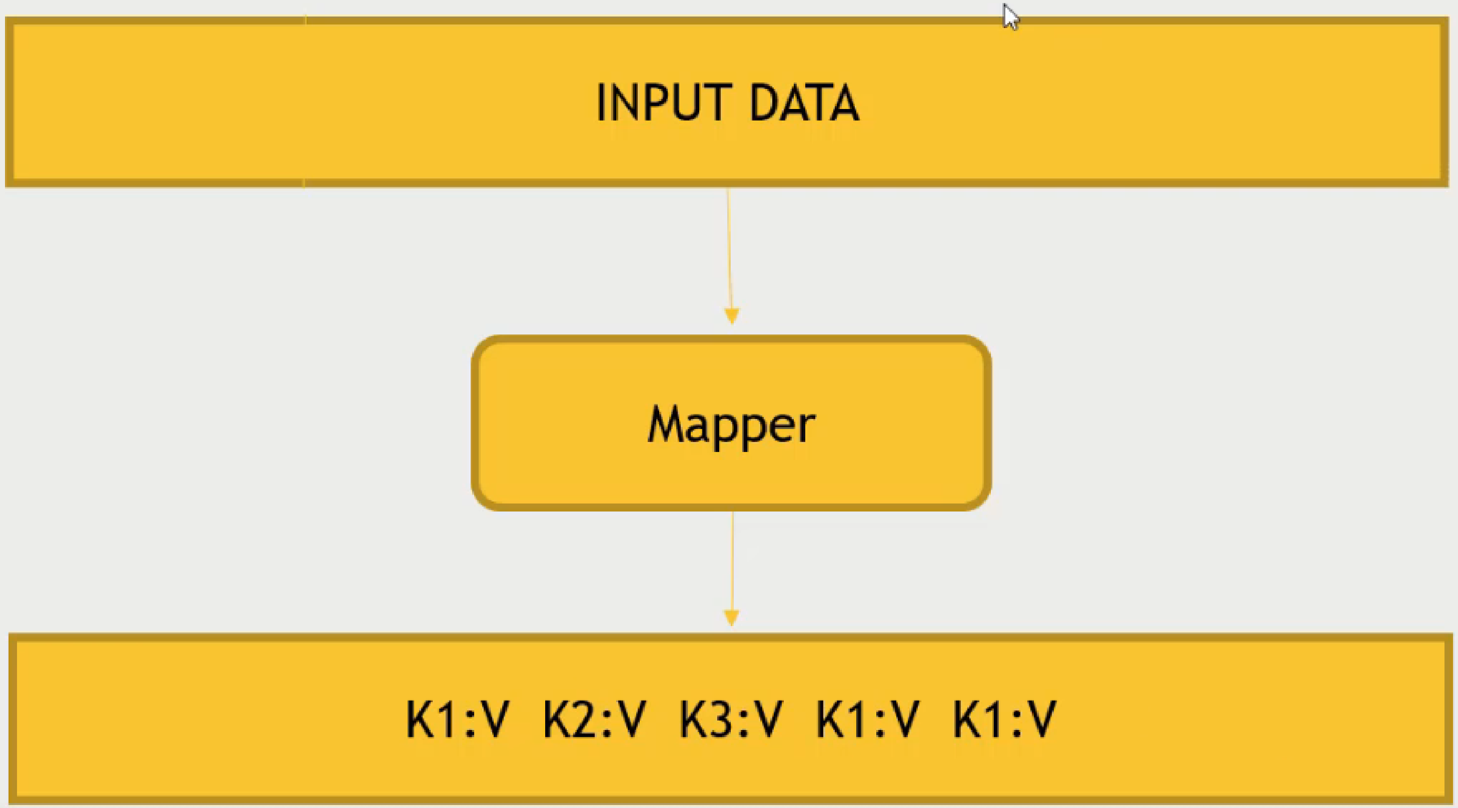
Practical hadoop Map Reduce

Movie Rating Example

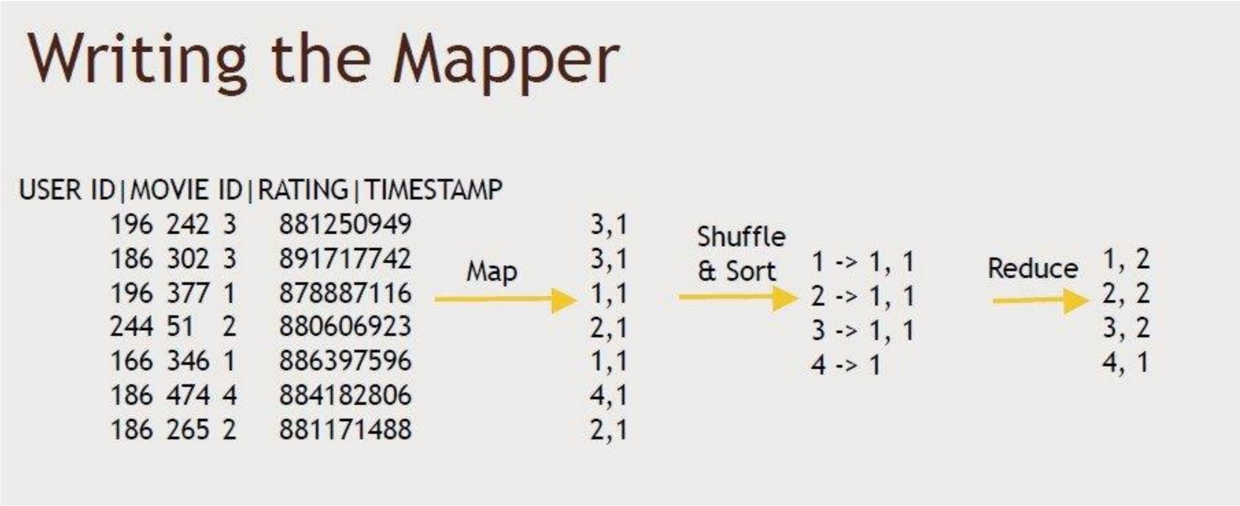
0 50 5 881250949
0 172 5 881250949
0 133 1 881250949
196 242 3 881250949
186 302 3 891717742
22 377 1 878887116
244 51 2 880606923
166 346 1 886397596
298 474 4 884182806
115 265 2 881171488
253 465 5 891628467
305 451 3 886324817

Mapping Phase

The MAPPER converts raw source data into **key/value** pairs



Map Reduce Movie Count Example




```python
from mrjob.job import MRJob
from mrjob.step import MRStep

class RatingsBreakdown(MRJob):
    def steps(self):
        return [
            MRStep(mapper=self.mapper_get_ratings,
                   reducer=self.reducer_count_ratings)        ]

    def mapper_get_ratings(self, _, line):
        (userID, movieID, rating, timestamp) = line.split('\t')
        yield rating, 1

    def reducer_count_ratings(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    RatingsBreakdown.run()
```

Workflow Scenario from the Client to NameNode/DataNode

1. Client Submission:

- The client submits a MapReduce job to the Hadoop cluster. The job includes the Mapper and Reducer code, and configuration parameters. The data is already stored in the Hadoop Distributed File System (HDFS)

2. ResourceManager and Job Scheduling:

- The ResourceManager receives the job submission and delegates the job to the ApplicationsManager.
- The ApplicationsManager allocates a new ApplicationMaster (AppMaster) instance for the job.

3. ApplicationMaster Initialization:

- The ApplicationMaster negotiates resources from the ResourceManager and requests containers (execution environments) on various NodeManagers (Data Nodes).

4. Data Splitting:

- The ApplicationMaster communicates with the NameNode to locate the input data blocks stored across the DataNodes.
- The input data is split into chunks, and the splits are assigned to the containers on the DataNodes where the data is located.

5. Mapper Execution:

- The NodeManagers launch Mapper tasks in the allocated containers.
- Each Mapper processes its assigned data chunk, producing intermediate key-value pairs.

6. Shuffling and Sorting:

The intermediate key-value pairs are shuffled and sorted by key. This involves transferring data across the network to group all values for the same key together.

7. Reducer Execution:

The NodeManagers launch Reducer tasks in containers.

Each Reducer processes its assigned key group, aggregating the values to produce the final output.

8. Writing the Output:

The Reducers write the final output to HDFS, which is managed by the NameNode.

The ApplicationMaster monitors the job's progress and updates the ResourceManager.

9. Job Completion:

Once all Mapper and Reducer tasks are complete, the ApplicationMaster informs the ResourceManager.

The client is notified of the job's completion, and the final results are available in HDFS

Practical

Hive Practical Example

Access Hive Console in Ambari

- In Ambari (left sidebar), click **Hive**.
- Look for a link under **Quick Links** called **Hive View 2.0**.
- Click it — this will open a web-based Hive editor (SQL interface).

Key Insights

- Hive works well for **batch queries** on huge datasets.
- SQL-like syntax (HiveQL) makes it easy for analysts to use.
- Not suited for real-time access — that's where HBase fits.

Step-by-Step Example in Hive

1. Create an External Table

```sql
CREATE EXTERNAL TABLE sales (
  transaction_id STRING,
  customer_id STRING,
  product_id STRING,
  amount DOUBLE,
  transaction_date STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE
LOCATION '/user/hive/sales_data/';
```

---

2. Sample Queries

a. Total sales per product

```sql
SELECT product_id, SUM(amount) AS total_sales
FROM sales
GROUP BY product_id;
```

b. Top 5 customers by total spend

```sql
SELECT customer_id, SUM(amount) AS total_spent
FROM sales
GROUP BY customer_id
ORDER BY total_spent DESC
LIMIT 5;
```

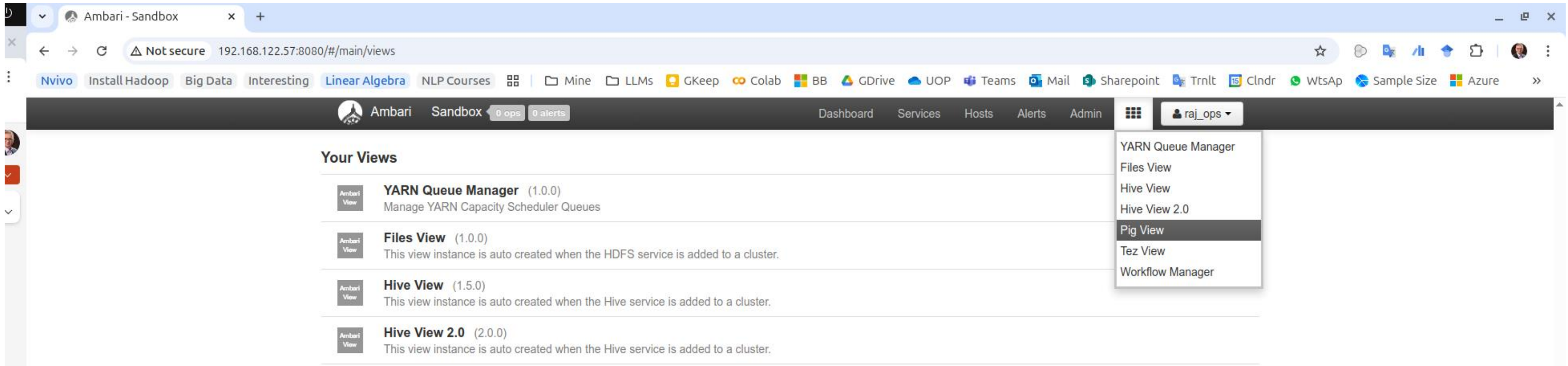
Practical Apache Pig

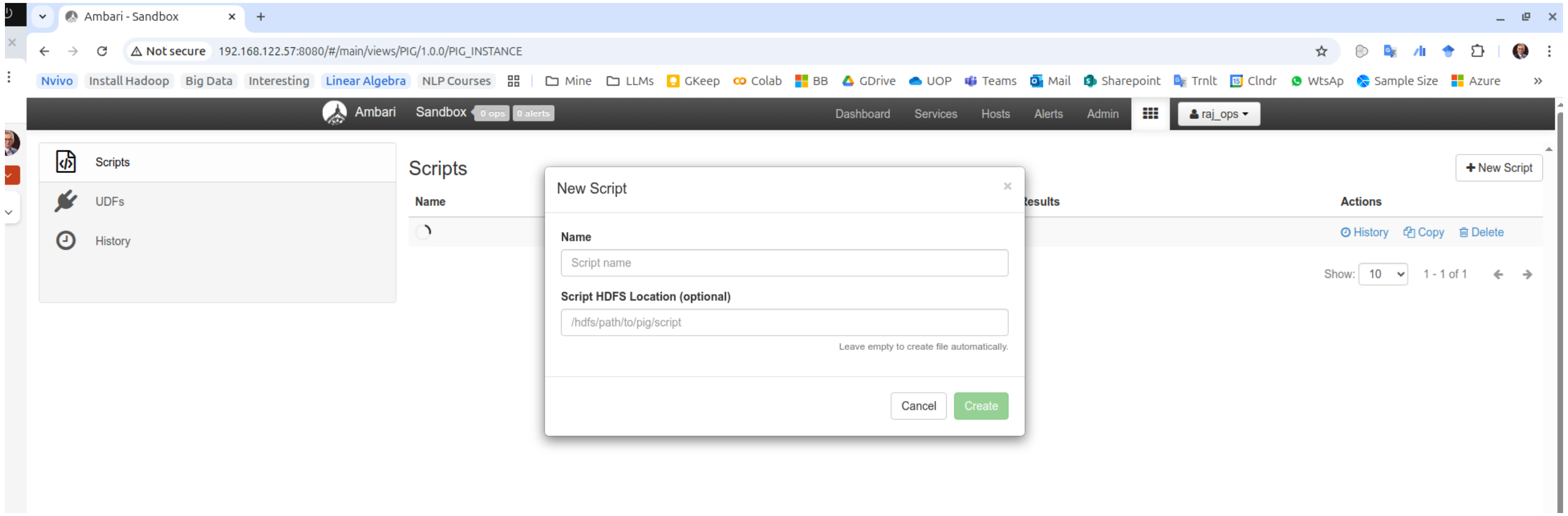
BUSINESS INTELLIGENCE AND DATA ANALYTICS

Institute of Analytics
Recognized Programme

University of Petra

Open Pig View and Create a New Script





Scenario: Analyze Website Logs with Apache Pig

Objective:
Extract insights from a web server log file (e.g., count hits per IP address, find most visited pages).

Key Highlights

- Demonstrates Pig's ability to process **semi-structured text**.
- Shows how to write a **data pipeline** using LOAD, GROUP, and FOREACH.
- Ideal for **log analysis, ETL pipelines**, or **preprocessing** before Hive or HBase.

Step 1: Sample Log Data (upload to HDFS)
Save the following as access_log.txt and upload to HDFS (e.g., /user/pig/logs/):

```
192.168.1.1 /index.html
192.168.1.2 /contact.html
192.168.1.1 /index.html
192.168.1.3 /about.html
192.168.1.2 /index.html
```

Step 2: Pig Script to Count Hits per Page

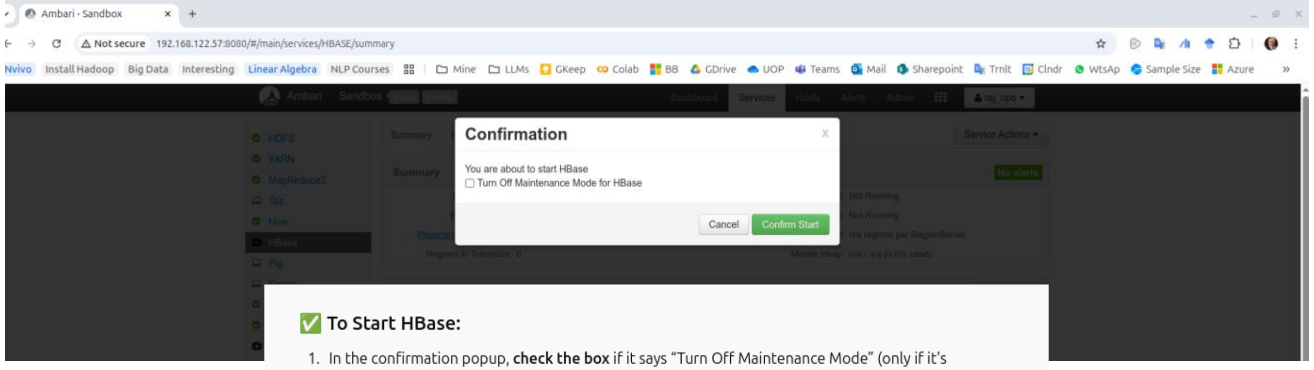
```
logs = LOAD '/user/pig/logs/access_log.txt' USING PigStorage(' ') AS (ip:chararray, page:chararray);

grouped_pages = GROUP logs BY page;

page_counts = FOREACH grouped_pages GENERATE group AS page, COUNT(logs) AS hits;

DUMP page_counts;
```

Expected Output

```
(/about.html,1)
(/contact.html,1)
(/index.html,3)
```

Practical

Practical HBase



✅ To Start HBase:

1. In the confirmation popup, **check the box** if it says "Turn Off Maintenance Mode" (only if it's checked).
2. Click **Confirm Start**.

Ambari will now:

- Start the **HBase Master**
- Start the **RegionServer**
- Connect with **ZooKeeper**

Give it 1–2 minutes. You'll see the status turn green when it's ready.

▶️ Next Steps After It Starts:

Once it's running, open a terminal in the sandbox and type:

```bash
hbase shell
```

Then you can begin testing with this:

```bash
create 'test', 'cf'
put 'test', 'row1', 'cf:name', 'Test Value'
scan 'test'
```

✅ Step 1: Start HBase from Ambari

1. Go to the **Ambari Dashboard**.
2. Click on **HBase** in the left panel.
3. Click the **Service Actions** dropdown on the top-right.
4. Select **Start** → This will start the HBase Master and RegionServer(s).
5. Wait for all green checkmarks and no alerts.

✅ Step 2: Access the HBase Shell

Once HBase is running:

1. Open **your Sandbox terminal** (you can SSH into it or use the web terminal).
2. Run:

```bash
hbase shell
```

You'll see the `hbase(main):001:0>` prompt, where you can start testing commands.

✅ Step 3: Run Sample Commands

Try these basic ones to verify:

```bash
create 'test', 'cf'
put 'test', 'row1', 'cf:name', 'Hello HBase'
get 'test', 'row1'
scan 'test'
```

Practical kafka

Practical Scenario: Real-Time Sensor Data Streaming with Apache Kafka

- **Scenario:** A company is monitoring temperature sensors in a factory. Each sensor sends data every few seconds.
- **The goal is to:**
  - Stream the sensor readings in real time.
  - Store them in HDFS for long-term analysis.
  - Optionally process them live using Spark or Storm.
- **Components Involved:**
  - **Kafka** → for streaming data
  - **HDFS** → for storing data
  - **Optional**: Flume or Spark Streaming for processing

Practical Scenario: Real-Time Sensor Data Streaming with Apache Kafka

Step-by-Step Workflow

Step 1: Start Kafka Services, ensure ZooKeeper and Kafka are running in Ambari.

Step 2: Create a Kafka Topic

- `kafka-topics --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic sensor_topic`
-

Step 3: Simulate a Sensor Producer (Terminal)

- `kafka-console-producer --broker-list localhost:6667 --topic sensor_topic`
- Type some simulated sensor data:
- `sensor1,2025-03-23T09:00,22.5`
- `sensor2,2025-03-23T09:01,21.8`
- `sensor1,2025-03-23T09:02,22.7`

Step 4: Start a Kafka Consumer (Another Terminal)

- `kafka-console-consumer --bootstrap-server localhost:6667 --topic sensor_topic --from-beginning`

Expected Output: You'll see the same messages printed in real-time — demonstrating **decoupled data streaming**.

- **Optional: Save Kafka Stream to HDFS**
- Use **Flume** or **Kafka Connect HDFS sink** to write streamed data into HDFS.

Key Takeaways:

- Kafka is ideal for **real-time ingestion**.
- Decouples **data producers and consumers**.
- Topics enable scalable, fault-tolerant messaging across multiple services.
- Would you like a follow-up scenario showing how Hive or Spark can consume data from Kafka?

Apache Oozie workflows

What is Apache Oozie?

Oozie is a **workflow scheduler system** for managing Hadoop jobs (MapReduce, Hive, Pig, Sqoop, etc.) as **directed acyclic graphs (DAGs)**.

What You Can Do with Workflow Manager:

- Create and visualize **Oozie workflows** using a graphical interface.
- Schedule workflows to run at a specific time or in response to events.
- Chain multiple jobs (e.g., run Pig → then Hive → then Email Notification).
- Define decision points, forks, and joins in the workflow logic.

Example Scenario: A student wants to automate the following daily:

- Run a Pig job to process logs.
- Run a Hive job to summarize the results.
- Archive the data using HDFS commands.
- With Workflow Manager, they can design and monitor this process without writing XML manually.

Practical Scenario: Daily Data Pipeline Automation with Oozie

Scenario: A company wants to automate a daily pipeline that:

- Runs a **Pig job** to clean raw log data.
- Runs a **Hive query** to generate a summary report.
- Moves the output to an **archive folder** in HDFS.

Workflow Steps in Oozie

- **Start node** – begins the workflow.
- **Pig action** – cleans the data (e.g., filters malformed records).
- **Hive action** – aggregates metrics (e.g., page views per user).
- **FS action** – moves the final result to `/data/archive/YYYY-MM-DD/`.
- **End node** – completes the workflow.

How to Implement in Hortonworks

- Open **Workflow Manager** in Ambari (as shown in your screenshot).
- Create a new workflow.
- Drag and drop actions:
  - **Pig**: run `clean_logs.pig`
  - **Hive**: run `summary_query.hql`
  - **FS**: define move operation
- Add a **coordinator** to schedule it daily at midnight.
- Deploy and run the workflow.

- **Example Hive Query (Step 2)**
- `SELECT user_id, COUNT(*) AS pageviews`
- `FROM cleaned_logs`
- `GROUP BY user_id;`

Key Highlights

- Automates multi-step workflows with scheduling.
- Centralized control, retry on failure, alerting.
- Supports Pig, Hive, Spark, Sqoop, Shell, Java, and more.

Oozie Workflow Manager (GUI) – Visual Design in Ambari

Steps to Create the Workflow:

- Open **Ambari → Workflow Manager → Create Workflow**.
- Name it: `daily-log-workflow`.
- Drag and drop the following components onto the canvas:
  - **Start**
  - **Pig Action**
    - Script: `/user/oozie/scripts/clean_logs.pig`
    - Input path: `/data/logs/raw/`
    - Output path: `/data/logs/cleaned/`
  - **Hive Action**
    - Script: `/user/oozie/scripts/summary_query.hql`
  - **FS Action**
    - Move `/data/logs/summary/` to `/data/logs/archive/${YEAR}-${MONTH}-${DAY}/`
  - **End**
- Link them in sequence: **Start → Pig → Hive → FS → End**
- Click **Save**, then go to the **Coordinator tab** to schedule it daily at midnight.

Daily Log Workflow – XML Version

```xml
<workflow-app name="daily-log-workflow" xmlns="uri:oozie:workflow:0.5">
    <start to="clean-logs"/>

    <action name="clean-logs">
        <pig>
            <script>clean_logs.pig</script>
            <param>input=/data/logs/raw/</param>
            <param>output=/data/logs/cleaned/</param>
        </pig>
        <ok to="summarize"/>
        <error to="fail"/>
    </action>

    <action name="summarize">
        <hive xmlns="uri:oozie:hive-action:0.5">
            <script>summary_query.hql</script>
        </hive>
        <ok to="archive"/>
        <error to="fail"/>
    </action>

    <action name="archive">
        <fs>
            <move source="/data/logs/summary/" target="/data/logs/archive/${YEAR}-${MONTH}-${DAY}/"/>
        </fs>
        <ok to="end"/>
        <error to="fail"/>
    </action>

    <kill name="fail">
        <message>Workflow failed</message>
    </kill>

    <end name="end"/>
</workflow-app>
```

Optional: Coordinator XML (to schedule it daily)

```
<coordinator-app name="daily-log-coordinator"
    xmlns="uri:oozie:coordinator:0.4" frequency="1"
    start="2025-03-24T00:00Z" end="2025-12-31T00:00Z" timezone="UTC">

    <action>
        <workflow>
            <app-path>/user/oozie/workflows/daily-log-workflow</app-path>
            <configuration>
                <property>
                    <name>YEAR</name>
                    <value>${coord:formatTime(coord:actualTime(), 'yyyy')}</value>
                </property>
                <property>
                    <name>MONTH</name>
                    <value>${coord:formatTime(coord:actualTime(), 'MM')}</value>
                </property>
                <property>
                    <name>DAY</name>
                    <value>${coord:formatTime(coord:actualTime(), 'dd')}</value>
                </property>
            </configuration>
        </workflow>
    </action>
</coordinator-app>
```