Quick Start Guide

Community 1.5.1 release

Quick Start Guide will walk you through various key features of Jumbune and help you to get started with Jumbune. It assumes that you have already obtained and installed Jumbune on your machine.

For more information on Jumbune installation, refer to **Installation Guide**.

To review additional details about this release before you start using the product, refer to **Release Notes.**

Table of Contents

Introduction

Jumbune is an open-source product built for accelerating Hadoop based solutions. It's a Linux based framework to assist analytic solution development, quality testing of data and efficient cluster utilization.

Jumbune is vendor neutral, being developed to work with all the major Hadoop distributions. Jumbune supports both **Yarn and non-Yarn** Hadoop clusters hence supports all the 3 active branches of Apache Distribution of Hadoop as well as other distributions such as

- Apache Hadoop 2.x, 0.23.x, 1.x
- CDH5
- MapR 3.x

Jumbune - Key Components

Jumbune provides a spectrum of components to assist Hadoop solutions enabling to accelerate solutions at different level of lifecycle and hence various user types. Key components of Jumbune are as follows:

Cluster Monitor

Prominent users: Hadoop Devops and Admins

Prominent Environments: Staging, Production

Cluster Monitor helps admins to monitor cluster with node system-level fine grained statistics. They have the option to mark frequently monitored stats as favorites and fetch refreshed results at specific interval. Selection of trends for specific stat is also available.

The cluster monitoring features can be summarized as follows:

- Node level cluster view to monitor system and Hadoop parameters.
- Data load partition to monitor the data load distribution among the various nodes of the cluster.
- Replica management view to show the data blocks replications in HDFS.

Uniqueness of Jumbune's Cluster Monitor

- Ability to give Rack aware Cluster wide Heat maps even on Apache Distribution of Hadoop
- Customizations:
  - Turn on/off monitoring from UI dashboard,
  - Change monitoring interval from dashboard,
  - Customize health criteria from dashboard over multiple health metrics
- No daemon deployment on worker nodes, easy to manage

MapReduce Job Profiler

Prominent users: Hadoop MapReduce developers, Devops and Admins.

Prominent Environments: Development, Staging, Production

Job Profiler profiles MapReduce jobs in a cluster and gives insights into CPU and heap dumps of Hadoop job. It also provides an in-depth graphical view of the MapReduce phases – Map, Reduce, Sort, Shuffle, Setup, and Cleanup.

The graphical view provides a correlation of parameters that include execution time, CPU consumption, memory usage, and data flow rate during these phases. It enables developers and devops to quickly figure out bottlenecks in Job, MapReduce phases that are consuming more time than expected, and those require optimization to increase execution efficiency.

Uniqueness of Jumbune's Job Profiler

- Correlation of resource, throughput and phase of a given Job helps to figure out,
  - Inefficient CPU and Memory utilization by the job or phases,
  - Low performing phases and respective throughput
- Works with all Apache Hadoop distributions

MapReduce Job Flow Debugger

Prominent users: MapReduce Developers, Quality Engineers

Prominent Environments: Development, Quality Assurance

Debugger provides code level control flow statistics of MapReduce job execution. User may apply regex validations or user defined validation classes. As per the applied validation, Flow Debugger checks the flow of input data tuples essentially <key,value> pair data for each mapper and reducer in the submitted job.

Jumbune provides a comprehensive table/chart view depicting the flow of input records through the job. The flow is displayed at job level, MapReduce level, and instance level. Unmatched keys/values represent the number of unexpected flow of key/value data through the job. Debugger drills down into the code to examine the flow of data for various counters like loops and if-conditions, else-if, etc.

Uniqueness of Jumbune's Flow Debugger

- Unique offering to analyse execution flow at the cluster level.
- Hierarchical drilling of flow both course grained (at Job level) to fine grained (Code level).
- Tears down hours/days of logic debugging into minutes.

Data Validation

Prominent users: MapReduce Developers, Quality Engineers, Devops

Prominent Environments: Development, Quality Assurance, Production

Data Validator enables user to generically specify data quality requirements. Jumbune's Data Validator finds the anomalies in the data lake. User just needs to specify data quality requirements and the data location to the Data Validator.

Jumbune in the current release supports HDFS as the data lake storage.

User may check for data violations under various categories like data type, null values, or regular expression values. As data validation report, user is provided with a detailed view of categorical anomalies found in the data. The results displayed at the specified HDFS path details violations for each record, which makes it easier to track corrupted records.

Uniqueness of Jumbune’s Data Validator

- Unique offering to generically specify data quality requirements according to user’s data lake.
- In depth record level data violation reports, can be drilled to line and field level.
- Makes impossible looking quality checks on Big Data Lake possible.

Data Quality Timeline

Prominent users: Quality Engineers, Devops, Admins

Prominent Environments: Quality Assurance, Staging, Production

When it’s the Environment of daily data proliferation, Data Quality Timeline (DQT) enables user to keep an eye on the quality of data that is continuously ingested. Through DQT, user can schedule jobs for analyzing data. Jobs can be scheduled using custom cron expression or through Jumbune’s scheduling interface.

DQT produces reports in the form of graph, based on different kind of checks like Null check, Regex check, Data type check etc. It also enables user to view reports of historical jobs.

Uniqueness of Jumbune’s Data Quality Timeline

- Daily Quality analysis and detailed reports.
- Produces reports in chronological order (i.e. Timeline) making interpretation of facts user friendly.
- Flexibility in job scheduling (Through Jumbune’s interface and through custom Cron Expressions).

Data Profiling

Prominent users: Quality Engineers, Devops, Admins

Prominent Environments: Quality Assurance, Staging, Production

Data Profiling facilitates a user to classify and categorize data according to his/her requirements. Jumbune provides two variants of data profiling viz ‘Without Criteria’ and ‘Criteria Based’.

When data profiling opted is of “Without Criteria” type, reports are produced on the basis of pattern discovery, consequently top five patterns are listed in the result.

In Criteria Based Data Profiling, custom criteria can be specified by the user based on requirements. User can specify numeric constraints on different fields. Comparison statistics can be seen in the report of Data Profiling.

Uniqueness of Jumbune’s Data Profiler

- Flexibility to do automatic profiling as well as specifying criteria wherever necessary.
- Probes into HDFS files and checks each individual record, there by producing immaculate reports.
- Powers user to assess risk involved in integrating data with other applications.

Starting Jumbune container

Jumbune can be started by performing following simple steps:

1. Start Jumbune by executing **bin/startWeb** script
2. Navigate to Jumbune home page at **'localhost:8080'** (if deployed on the same machine) or **<Jumbune machine IP>:8080** (if deployed on a different machine)

Building Jumbune Image From Docker

Jumbune image can be built by docker automatically by reading the instructions from the DockerFile.

"docker build", in the terminal will build the image executing the instructions step by step.

Usage

After checking out Jumbune, call docker build with the path of your source repository as the argument

```
$ sudo docker build .
```

Unless you are experimenting with docker you should always pass the -t option to docker build so that the resulting image is tagged. A simple human readable tag will help you manage what each image was created for.

Building Image

At the directory location of Dockerfile,

```
$ sudo docker build –t = "jumbune/pseudo-distributed:1.5.0"   .
```

Alternatively, image can also be built directly by passing Jumbune's repository location to the above command,

```
$ sudo docker build –t = " jumbune/pseudo-distributed:1.5.0"    github.com/impetus-opensource/jumbune
```

Alternatively, you can get an automated build from docker registry at location,
https://registry.hub.docker.com/u/jumbune/jumbune/

Running Image

```
$ docker run -d --name="jumbune" -h "jumbune-docker" -p 8042 -p 8088:8088 -p 50070:50070 -p 50075:50075 -p 50090:50090 -p 8080:8080 -p 5555 jumbune/pseudo-distributed:1.5.0
```

After few seconds, open http://localhost:8080 to see Jumbune home page

Note: - Currently Jumbune Image through Docker can be created for Apache Hadoop-2.4.1 distribution only.

Running a Jumbune Component

On the Jumbune home page, choose one of the following possible options:

a. **New**: Create a JSON file
b. **Open**: Upload a JSON from file system
c. **Select**: Choose an existing historical JSON file from Jumbune repository

To create a JSON file, perform the following steps:

1. Select the component you wish to execute
2. Fill up the Basic cluster and component specific details like Name node, Data node, and job jar details
3. Click **Validate** to validate the filled details.
4. Once your JSON is successfully validated, click **Run**

   Result for the requested component(s) will be displayed in form of graphs and details. The reports are self-navigable and intuitive to assist user.

Running a shipped example

To execute Jumbune sample examples (shipped along with the distribution), perform the following steps:

1. Navigate to the example folder you are wishing to run, go through the readme.txt

2. Upload the JSON from **$JUMBUNE_HOME/examples/resources/sample_json/** directory from the **Open** option on the Jumbune home page.

3. The sample job jars are found in the **$JUMBUNE_HOME/examples/example-distribution/** directory.

Running the Word Count example:

For running the word count example, execute the following steps:

1. Upload sample input file in HDFS using the following command:

   ```
   bin/hadoop fs -put $JUMBUNE_HOME
   /examples/resources/data/PREPROCESSED/data1 </Jumbune/data/data1 or any
   HDFS path>
   ```

   Note: Ensure that path <HDFS path> is not present on HDFS and user has appropriate permission to put data file on HDFS.

2. Upload sample wordcount JSON **$JUMBUNE_HOME/examples/resources/sample_json/WordCountSample.json**).

3. Edit Name-node and Data-node information.

4. In 'M/R Jobs' tab select the WordCount sample jar, either by mentioning the path on the Jumbune machine or by uploading from local machine.

5. **Validate** and **Run** the job.

For Running Movie Rating example (for Profiling):

For movie rating, perform the following steps:

1. Upload sample input file in HDFS by using the following command:

   bin/hadoop fs -put *$JUMBUNE_HOME* /examples/resources/data/u.data </Jumbune/examples/regex or any HDFS path>

   Note: Ensure that path is not present on HDFS and user has appropriate permission to put data file on HDFS.

2. Upload sample JSON (**$JUMBUNE_HOME/examples/resources/sample JSON/MovieRatingSample.json).**

3. Edit the Name-node and Data-node information.

4. In the **'M/R Jobs'** tab select movie rating sample jar, either by mentioning the path on the Jumbune machine or by uploading from local machine.

5. Validate and run the job.

For Running Bank Defaulters example (for Debugging):

For bank defaulters, perform the following steps:

1. Upload sample input file in HDFS by using the following command

   bin/hadoop fs -put *$JUMBUNE_HOME* examples/resources/data/defaulterlistdata.txt </Jumbune/examples/defaulter or any HDFS path>

   Note: Ensure that path is not present on HDFS and user has appropriate permission to put data file on HDFS.

2. Upload sample JSON **(<Jumbune home>/examples/resources/sample JSON/BankDefaultersSample.json**).

3. In **'M/R Jobs'** tab select the bank defaulters sample jar, either by mentioning the path on the Jumbune machine or by uploading from local machine.

4. Edit the Name-node and Data-node information.

5. **Validate** and **Run** the job.

For Running US Region Port Out example (for Debugging):

For US Region port out, perform the following steps:

1. Upload sample input file in HDFS by using the following command:

   bin/hadoop fs -put *$JUMBUNE_HOME*/examples/resources/data/PREPROCESSED/data1 /Jumbune/Demo/input/PREPROCESSED/data1
   bin/hadoop fs -put *$JUMBUNE_HOME*/examples/resources/data/PREPROCESSED/data2 / Jumbune/Demo/input/PREPROCESSED/data2

   Note: Ensure that path is not present on HDFS and user has appropriate permission to put data file on HDFS.

2. Upload sample JSON **(<Jumbune home>/examples/resources/sample JSON/USRegionPortOutSample.json**).
3. In the **'M/R Jobs'** tab select the US region portout sample jar, either by mentioning the path on the Jumbune machine or by uploading from the local machine.
4. Edit the Name-node and Data-node information.
5. **Validate** and **Run** the job.

For Running Clickstream Analysis example (for Debugging):

1. Upload sample input file in HDFS by using the following command:

   bin/hadoop fs -put *$JUMBUNE_HOME* /examples/resources/data/clickstream.tsv /Jumbune/clickstreamdata

   Note: Ensure that path is not present on HDFS and user has appropriate permission to put data file on HDFS.

2. Upload sample JSON **(<Jumbune home>/examples/resources/sample JSON/ClickstreamSample.json**).
3. In **'M/R Jobs'** tab select the clickstream sample jar, either by mentioning the path on the Jumbune machine or by uploading from the local machine.
4. Edit Name-node and Data-node information.
5. **Validate** and **Run** the job.

For Running Sensor data example (for HDFS Validation):

For sensor data, perform the following steps:

1. Upload sample input file in HDFS by using the following command

   bin/hadoop fs -put *$JUMBUNE_HOME* /examples/resources/data/sensor_data/Jumbune/sensordata

   Note: Ensure that path is not present on HDFS and user has appropriate permission to put data file on HDFS.

2. Upload sample JSON ***$JUMBUNE_HOME*****/examples/resources/sample JSON/SensorDataSample.json**).

3. Edit Name-node and Data-node information.

4. **Validate** and **Run** the job.

NOTE:

- We have used *GenericOptionsParser* in our examples, so do not provide class name information, just select **'Job Class defined in the Jar Manifest'** option on **'M/R Jobs'** tab on Jumbune UI Wizard.

- Ensure that output path provided in JSON file must not exist on HDFS previously.