**Apache Zeppelin**



Contents

[1. Installation 4](#_Toc482861346)

[1.1. Installation of Apache Spark 4](#_Toc482861347)

[1.2. Installation of Scala 6](#_Toc482861348)

[1.3. Installation of Apache Zeppelin 7](#_Toc482861349)

[1.4. Apache Zeppelin Configuration 9](#_Toc482861350)

[2. Apache Zeppelin History 12](#_Toc482861351)

[3. Apache Zeppelin Architecture 14](#_Toc482861352)

[4. Apache Zeppelin Interpreter 14](#_Toc482861353)

[4.1. Interpreter Architecture 14](#_Toc482861354)

[4.2. Interpreter Processing Lifecycle 14](#_Toc482861355)

[4.3. Core Interpreters 15](#_Toc482861356)

[4.4. Third-parties Interpreters 15](#_Toc482861357)

[4.5. Create Your Own Interpreter 15](#_Toc482861358)

[5. What Is Apache Zeppelin? 16](#_Toc482861359)

[5.1. Multi-purpose Notebook 16](#_Toc482861360)

[5.2. Multiple Language Backend 17](#_Toc482861361)

[5.3. Data Visualization 17](#_Toc482861362)

[5.4. Dynamic forms 18](#_Toc482861363)

[5.5. Collaborate by sharing your Notebook & Paragraph 18](#_Toc482861364)

[5.6. 100% Opensource 19](#_Toc482861365)

[6. Apache Zeppelin UI 19](#_Toc482861366)

[6.1. Main home 19](#_Toc482861367)

[6.2. Menus 20](#_Toc482861368)

[6.2.1. Notebook 20](#_Toc482861369)

[6.2.2. Settings 20](#_Toc482861370)

[6.3. Note Layout 22](#_Toc482861371)

[7. MongoDB Interpreter for Apache Zeppelin 25](#_Toc482861372)

[7.1. Download 25](#_Toc482861373)

[7.2. Deployment 25](#_Toc482861374)

[7.3. Configuration 25](#_Toc482861375)

[7.4. MongoDB Installation Verification 25](#_Toc482861376)

[7.5. Create MongoDB Interpreter 26](#_Toc482861377)

[7.6. Load Sample Data into MongoDB 27](#_Toc482861378)

[7.7. Generate Sample Reports 27](#_Toc482861379)

[7.7.1. Display a Table 27](#_Toc482861380)

[7.7.2. Display a Bar Chart 28](#_Toc482861381)

[7.7.3. Display a Pie Chart 28](#_Toc482861382)

[8. Generic JDBC Interpreter for Apache Zeppelin 29](#_Toc482861383)

[8.1. Overview 29](#_Toc482861384)

[8.2. Create a new JDBC Interpreter 29](#_Toc482861385)

[8.3. More Properties 31](#_Toc482861386)

[8.4. Binding JDBC Interpreter to Notebook 32](#_Toc482861387)

[8.5. Test Database and Zeppelin Connection 33](#_Toc482861388)

[Run the paragraph with JDBC interpreter 33](#_Toc482861389)

[8.6. Load Test Data into Hive 33](#_Toc482861390)

[8.7. Generate Sample Reports 34](#_Toc482861391)

[8.7.1. Display Data in Various Format 34](#_Toc482861392)

[8.7.2. Print Report 35](#_Toc482861393)

[9. Apache Zeppelin Dynamic Form 36](#_Toc482861394)

[9.1. Using form Templates 36](#_Toc482861395)

[9.2. Creates Programmatically 38](#_Toc482861396)

[9.3. MongoDB Dynamic Form Example 40](#_Toc482861397)

# Installation

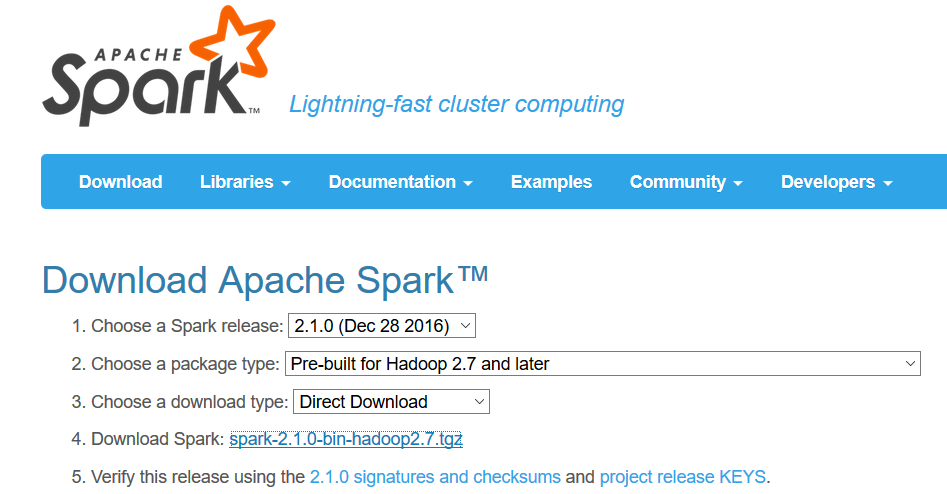
Installation including install following application

* Apache Spark
* Apache Scala
* Apache Zeppelin

## Installation of Apache Spark

**Step 1: Download Spark**

Apache Spark can be downloaded from this address: <http://spark.apache.org/downloads.html>.



Download Spark binary file spark-2.1.0-bin-hadoop2.7.tgz and save it to a local directory.

**Step 2: Transfer Spark to VirtualBox**

Transfer the downloaded Spark binary file to the /opt folder on the virtual machine, and extract the content using the below command:

$tar xvf spark-2.1.0-bin-hadoop2.7.tgz

A new folder will be added as /opt/spark-2.1.0-bin-hadoop2.7.

**Step 3: Add SPARK\_HOME to PATH**

Change to directory /opt, create a soft link to spark folder

ln -s /opt/spark-2.1.0-bin-hadoop2.7 spark

Edit the .profile file in folder /home/vagrant by adding following lines

export SPARK\_HOME=/opt/spark

PATH="/opt/spark/bin:$PATH"

**Step 4: Set IP Address for Spark**

Edit the file $SPARK\_HOME/conf/spark\_env.sh by adding following two lines:

export SPARK\_MASTER\_IP=127.0.0.1

export SPARK\_LOCAL\_IP=127.0.0.1

**Step 5: Verify Spark Installation**

Change to Spark home bin directory:

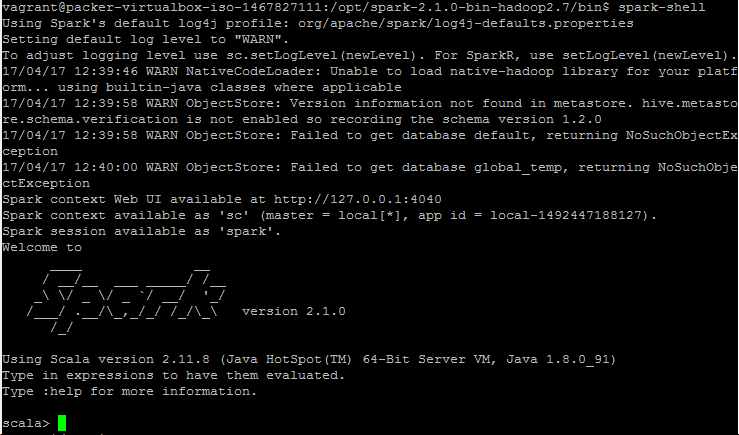
$cd $SPARK\_HOME/bin

Change the access permissions of files under this folder:

$chmod +x \*

Run spark-shell command:

$spark-shell



**Reference**: Install Apache Spark 2.0 - Quick Setup video: <https://www.youtube.com/watch?v=NVr5Dc-b4C4>

## Installation of Scala

**Step 1: Download Spark**

Download Scala 2.11.8 (scala-2.11.8.tgz) from this address <https://www.scala-lang.org/download/2.11.8.html>

**Step 2: Transfer Scala to VirtualBox**

Transfer the downloaded Scala binary file to the /opt folder on the virtual machine, and extract the content using the below command:

$tar xvf scala-2.11.8.tgz

A new folder will be added as /opt/scala-2.11.8.

**Step 3: Add SCALA\_HOME to PATH**

Change to directory /opt, create a soft link to spark folder

ln -s /opt/scala-2.11.8 scala

Edit the .profile file in folder /home/vagrant by adding following lines

export SCALA\_HOME=/opt/scala

PATH="/opt/scala/bin:$PATH"

**Step 4: Verify Scala Installation**

Change to Scala home bin directory:

$cd $SCALA\_HOME/bin

Change the access permissions of files under this folder:

$chmod +x \*

Run spark-shell command:

$scala -vesion

****

## Installation of Apache Zeppelin

**Step 1: Download Apache Zeppelin**

Apache Zeppelin (version 0.7.1) can be downloaded from this address: <https://zeppelin.apache.org/download.html>.



Download Zeppelin binary file zeppelin-0.7.1-bin-all.tgz and save it to a local directory.

**Step 2: Transfer Zeppelin to VirtualBox**

Transfer the downloaded Zeppelin binary file to the /opt folder on the virtual machine, and extract the content using the below command:

$tar xvf zeppelin-0.7.1-bin-all.tgz

A new folder will be added as /opt/zeppelin-0.7.1-bin-all.

**Step 3: Verify SPAKE\_HOME and JAVA\_HOME**

Ensure SPARK\_HOME and JAVA\_HOME are set in .bashrc file.

export SCALA\_HOME=/opt/scala-2.11.8

export SPARK\_HOME=/opt/spark-2.1.0-bin-hadoop2.7

export JAVA\_HOME="/usr/lib/jvm/java-8-oracle"

**Step 4: Update Zeppelin Configuration Files**

Edit the file /opt/zeppelin-0.7.1-bin-all/conf/zeppelin-env.sh by adding following two lines:

export SPARK\_HOME=/opt/spark

export JAVA\_HOME="/usr/lib/jvm/java-8-oracle"

Edit the file /opt/zeppelin-0.7.1-bin-all/conf/zeppelin-site.xml by changing the port number to 8090 following two lines:

<property>

<name>zeppelin.server.port</name>

<value>8090</value>

<description>Server port.</description>

</property>

**Note:** Remember to add 8090 port forwarding in Virtual Machine.

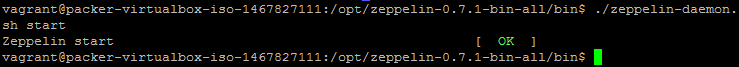
**Step 4: Verify Zeppelin Installation**

Change to Zeppelin bin directory, and change the access permissions of files under this folder:

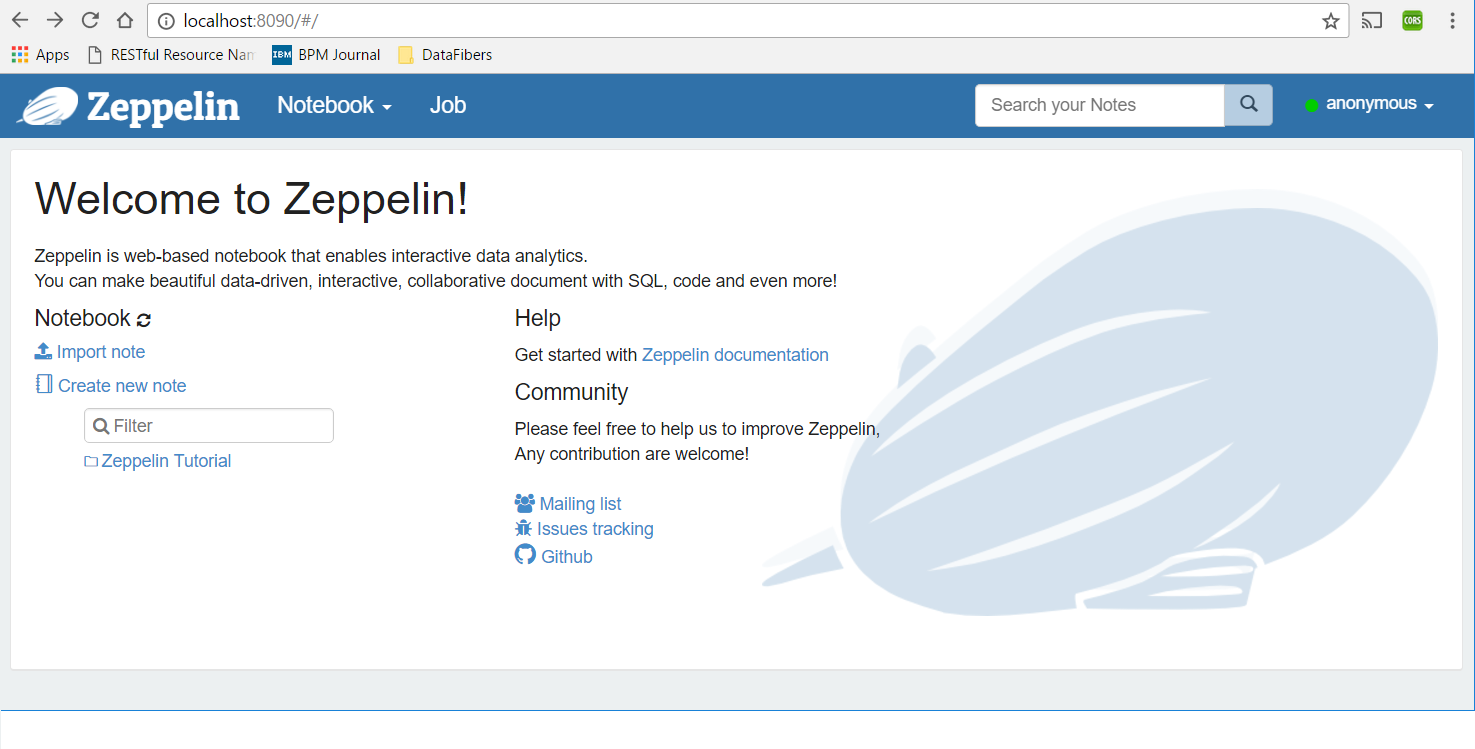
$chmod +x \*

Run Zeppelin by issuing command:

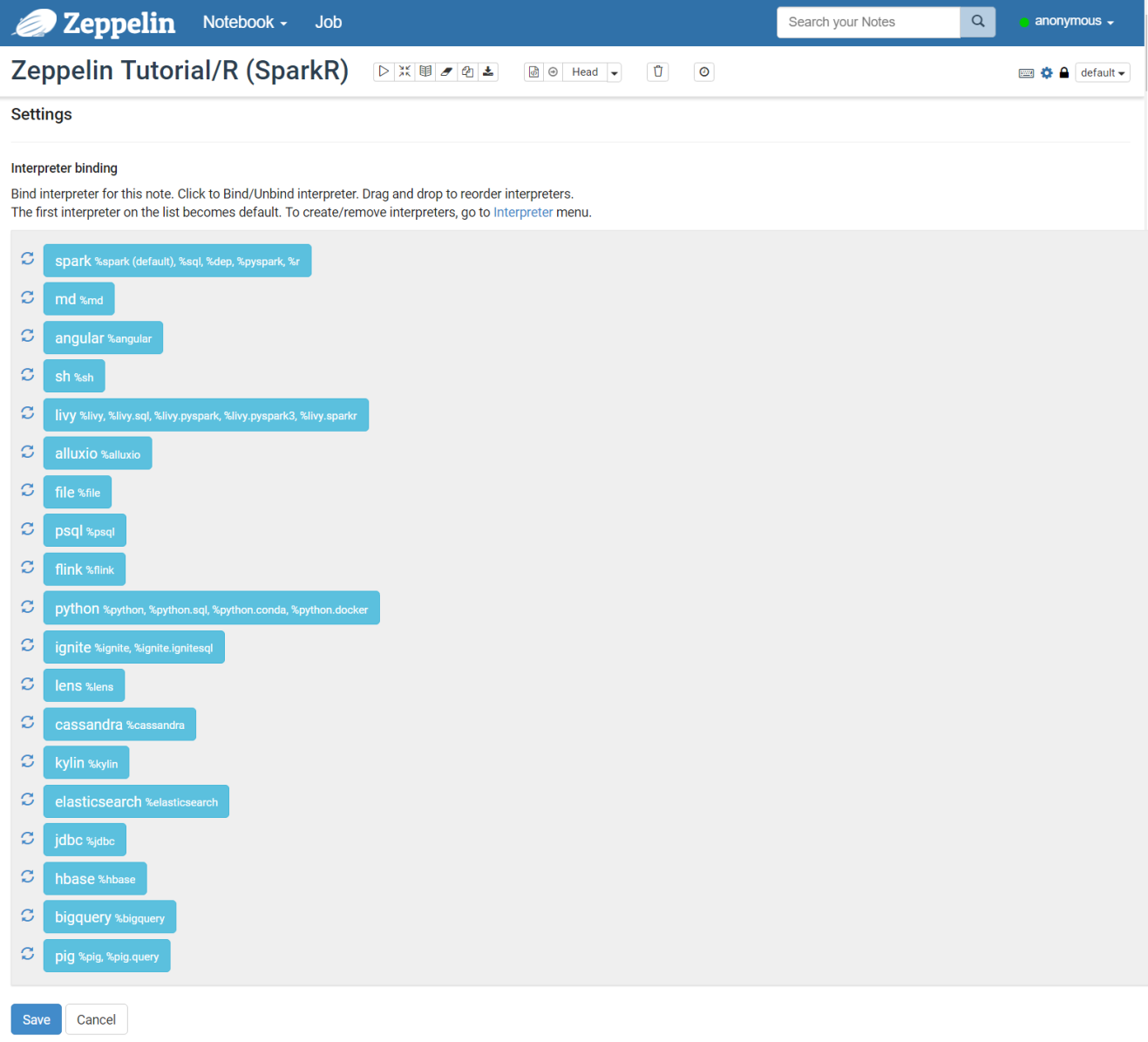
$./zeppelin-daemon.sh start



Enter <http://localhost:8090> into browser address, following Zeppelin home page should show up.



Expand the link **Zeppelin Tutorial**, click on any tutorial, e.g. **Basic Features (Spark)**, the below page will show up. Click the Save button to save the interpreter bindings.

****

**Reference**: Install Apache Zeppelin video: <https://www.youtube.com/watch?v=9cZldDfG9s0>

**Zeppelin Tutorial**: <https://hortonworks.com/hadoop-tutorial/getting-started-apache-zeppelin/>

## Apache Zeppelin Configuration

Configuration can be done by both environment variable(conf/zeppelin-env.sh) and java properties(conf/zeppelin-site.xml). If both defined, environment variable is used.

|  |  |  |  |
| --- | --- | --- | --- |
| **zepplin-env.sh** | **zepplin-site.xml** | **Default value** | **Description** |
| ZEPPELIN\_PORT | zeppelin.server.port | 8080 | Zeppelin server port. |
| ZEPPELIN\_MEM | N/A | -Xmx1024m -XX:MaxPermSize=512m | JVM mem options |
| ZEPPELIN\_INTP\_MEM | N/A | ZEPPELIN\_MEM | JVM mem options for interpreter process |
| ZEPPELIN\_JAVA\_OPTS | N/A |  | JVM Options |
| ZEPPELIN\_ALLOWED\_ORIGINS | zeppelin.server.allowed.origins | \* | Allows a way to specify a ',' separated list of allowed origins for rest and websockets. i.e. http://localhost:8080 |
| ZEPPELIN\_SERVER\_CONTEXT\_PATH | zeppelin.server.context.path | / | Context Path of the Web Application |
| ZEPPELIN\_SSL | zeppelin.ssl | false |  |
| ZEPPELIN\_SSL\_CLIENT\_AUTH | zeppelin.ssl.client.auth | false |  |
| ZEPPELIN\_SSL\_KEYSTORE\_PATH | zeppelin.ssl.keystore.path | keystore |  |
| ZEPPELIN\_SSL\_KEYSTORE\_TYPE | zeppelin.ssl.keystore.type | JKS |  |
| ZEPPELIN\_SSL\_KEYSTORE\_PASSWORD | zeppelin.ssl.keystore.password |  |  |
| ZEPPELIN\_SSL\_KEY\_MANAGER\_PASSWORD | zeppelin.ssl.key.manager.password |  |  |
| ZEPPELIN\_SSL\_TRUSTSTORE\_PATH | zeppelin.ssl.truststore.path |  |  |
| ZEPPELIN\_SSL\_TRUSTSTORE\_TYPE | zeppelin.ssl.truststore.type |  |  |
| ZEPPELIN\_SSL\_TRUSTSTORE\_PASSWORD | zeppelin.ssl.truststore.password |  |  |
| ZEPPELIN\_NOTEBOOK\_HOMESCREEN | zeppelin.notebook.homescreen |  | Id of notebook to be displayed in homescreen ex) 2A94M5J1Z |
| ZEPPELIN\_NOTEBOOK\_HOMESCREEN\_HIDE | zeppelin.notebook.homescreen.hide | false | hide homescreen notebook from list when this value set to "true" |
| ZEPPELIN\_NOTEBOOK\_DIR | zeppelin.notebook.dir | notebook | Where notebook file is saved |
| ZEPPELIN\_NOTEBOOK\_S3\_BUCKET | zeppelin.notebook.s3.bucket | zeppelin | Bucket where notebook saved |
| ZEPPELIN\_NOTEBOOK\_S3\_USER | zeppelin.notebook.s3.user | user | User in bucket where notebook saved. For example bucket/user/notebook/2A94M5J1Z/note.json |
| ZEPPELIN\_NOTEBOOK\_STORAGE | zeppelin.notebook.storage | org.apache.zeppelin.notebook.repo.VFSNotebookRepo | Comma separated list of notebook storage |
| ZEPPELIN\_NOTEBOOK\_RELOAD\_FROM\_STORAGE | zeppelin.notebook.reloadAllNotesFromStorage | false | Notebook list and contents will be always loaded from repository if set true. If set false, modified notebooks or new notebooks added on file system level won't be reflected on Zeppelin till user restarts Zeppelin. |
| ZEPPELIN\_INTERPRETERS | zeppelin.interpreters | org.apache.zeppelin.spark.SparkInterpreter, org.apache.zeppelin.spark.PySparkInterpreter, org.apache.zeppelin.spark.SparkSqlInterpreter, org.apache.zeppelin.spark.DepInterpreter, org.apache.zeppelin.markdown.Markdown, org.apache.zeppelin.shell.ShellInterpreter, org.apache.zeppelin.hive.HiveInterpreter ... | Comma separated interpreter configurations [Class]. First interpreter become a default |
| ZEPPELIN\_INTERPRETER\_DIR | zeppelin.interpreter.dir | interpreter | Zeppelin interpreter directory |

# Apache Zeppelin History



**Moon Soo Lee** (이문수, 李文秀)

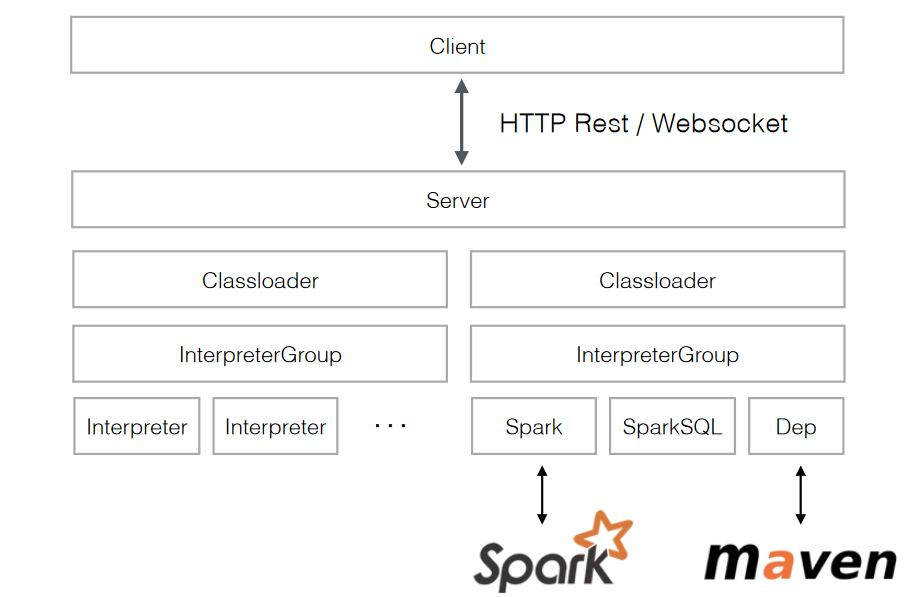
-- *co-founder and CTO of* [*NFLabs*](http://www.nflabs.com/) *and the creator for* [*Apache Zeppelin*](https://zeppelin.incubator.apache.org/)

*“For data analysis, we need tool, but couldn’t find a one I like.”*

*“Decided to make one, really good one”*

|  |  |
| --- | --- |
| **2012 – 2013** | **The First Attempt**    It’s got graphic REPL, deployment, search, import tool  But **failed**, because   * It wasn’t widely used * It wasn’t opensource |
| **2013 – 2014** | **The Second Attempt**     * Opensourced, graphic REPL from commercial product * the first version of Zeppelin * not widely used * it was slow * difficult to use, * … |
| **2014 ~** | **The Third Attempt**    Graphic REPl 🡪 Notebook  with Apache Spark integration |

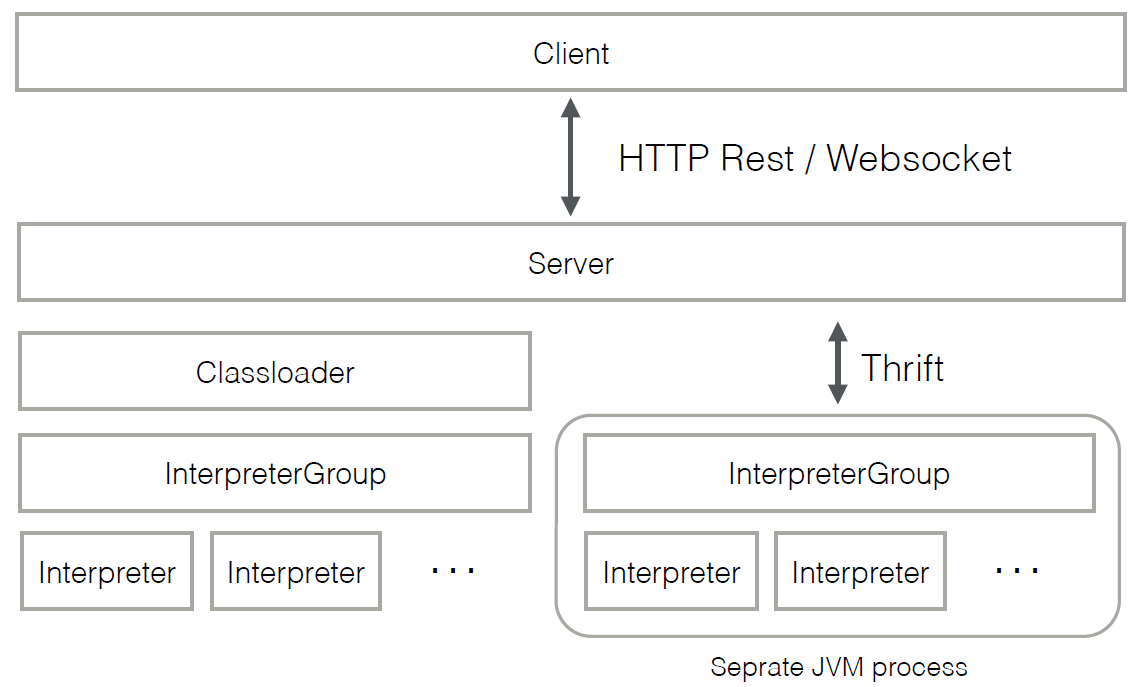
# Apache Zeppelin Architecture



# Apache Zeppelin Interpreter

## Interpreter Architecture

Pluggable interpreters system.

****

## Interpreter Processing Lifecycle

1. Receive input commands /data

* as raw text
* from form data

1. Process the input commands /data by the External back-end
2. Format the response using Zeppelin display system
3. Send response back to the Zeppelin engine

## Core Interpreters

1. Spark (Spark core, SparkSQL/DataFrame, PySpark)

* Spark core = default (or %spark)
* SparkSQL = %sql

1. Shell (%sh)
2. Markdown (%md)
3. AngularJS (%angular)

## Third-parties Interpreters

* Hive
* Phoenix
* Tajo
* Flink
* Ignite
* Lens
* Cassandra
* Geode
* PostgreSQL
* Kylin

## Create Your Own Interpreter

Take the SparkInterpreter.java as an example.

package org.apache.zeppelin.spark;

// bunch of imports

/\*\* Spark interpreter for Zeppelin. \*/

public class SparkInterpreter extends Interpreter {

public SparkInterpreter(Properties property, SparkContext sc) {...}

@Override

public void **open()**{...}

@Override

public InterpreterResult **interpret(String line, InterpreterContext context)**

{...}

@Override

public void **close()** {...}

@Override

public void **cancel(InterpreterContext context)** {...}

@Override

public int **getProgress(InterpreterContext context)** {...}

@Override

public List<InterpreterCompletion> **completion(String buf, int cursor)** {...}

@Override

public FormType **getFormType()** {...}

@Override

public Scheduler **getScheduler()** {...}

}

To register your interpreter in config file, add the FQCN of your interpreter to conf/zeppelin-site.xml file:

<property>

<name>zeppelin.interpreters</name>

<value>

**org.apache.zeppelin.spark.SparkInterpreter**,

org.apache.zeppelin.markdown.Markdown,

org.apache.zeppelin.angular.AngularInterpreter,

org.apache.zeppelin.shell.ShellInterpreter,

org.apache.zeppelin.flink.FlinkInterpreter,

……

</value>

<description>Comma separated interpreter configurations. **First interpreter**

**become a default.**

</description>

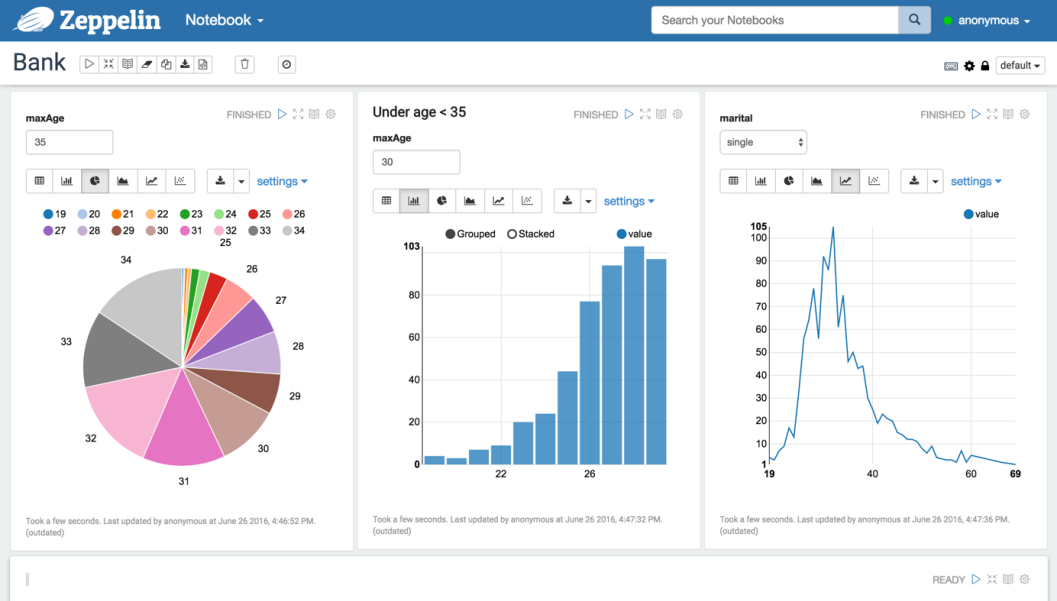
</property>

# What Is Apache Zeppelin?

## Multi-purpose Notebook

The Notebook is the place for all your needs.

* **Data Ingestion --** obtaining/importing large volumes data of diverse formats into data lake
* **Data Discovery --** using interactive reports and exploring data from multiple sources.
* **Data Analytics --** qualitative and quantitative processes to enhance productivity and business gain.
* **Data Visualization & Collaboration** **--** interactive visual representations of data, shared by more than one person with the common goal



## Multiple Language Backend

[Apache Zeppelin interpreter](https://zeppelin.apache.org/docs/latest/manual/interpreters.html) concept allows any language/data-processing-backend to be plugged into Zeppelin. Currently Apache Zeppelin supports many interpreters such as Apache Spark, Python, JDBC, Markdown and Shell.



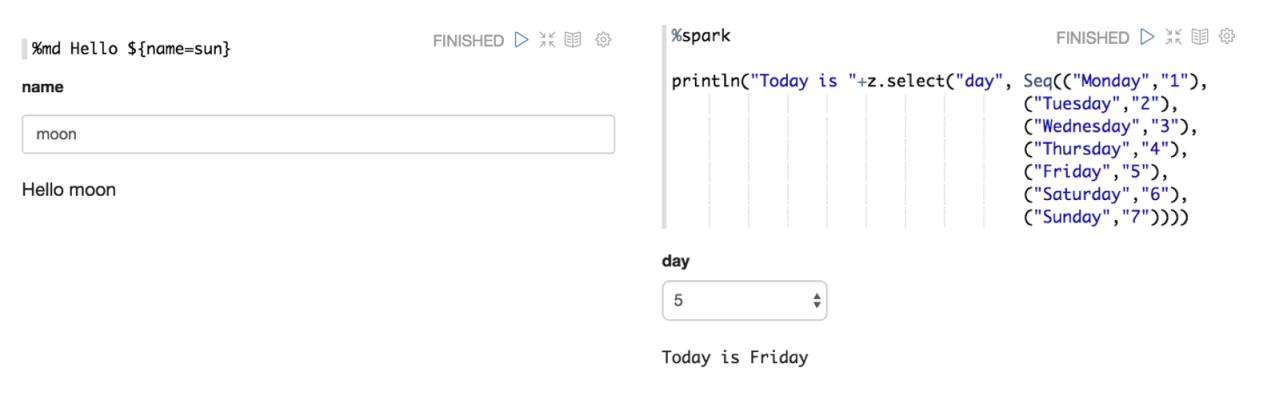
## Data Visualization

Some basic charts are already included in Apache Zeppelin. Visualizations are not limited to SparkSQL query, any output from any language backend can be recognized and visualized.

|  |  |
| --- | --- |
| https://zeppelin.apache.org/assets/themes/zeppelin/img/graph1.png | https://zeppelin.apache.org/assets/themes/zeppelin/img/graph2.png |

## Dynamic forms

Apache Zeppelin can dynamically create some input forms in your notebook.



## Collaborate by sharing your Notebook & Paragraph

Your notebook URL can be shared among collaborators. Then Apache Zeppelin will broadcast any changes in realtime, just like the collaboration in Google docs.



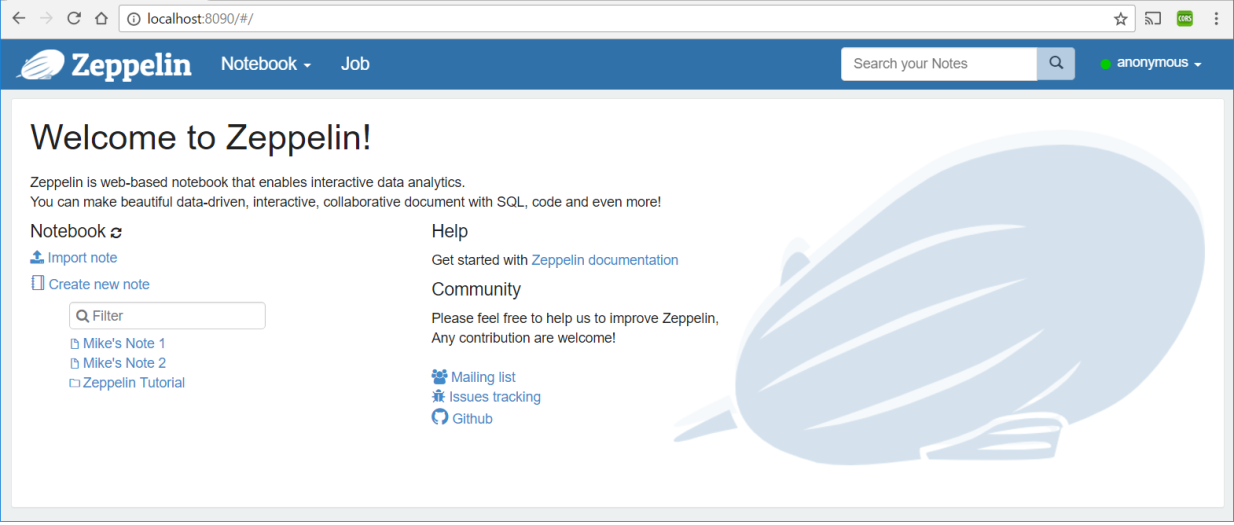
## 100% Opensource



# Apache Zeppelin UI

## Main home

The first time you connect to Zeppelin, you'll land at the main page similar to the below screen capture.



On the left of the page are listed all existing notes. Those notes are stored by default in the $ZEPPELIN\_HOME/notebook folder.

You can filter them by name using the input text form. You can also create a new note, refresh the list of existing notes (in case you manually copy them into the $ZEPPELIN\_HOME/notebook folder) and import a note.

|  |  |
| --- | --- |
| When clicking on the Import Note link, a new dialog opens. From there you can import your note from local disk or from a remote location if you provide the URL.  By default, the name of the imported note is the same as the original note but you can override it by providing a new name. |  |

## Menus

### Notebook

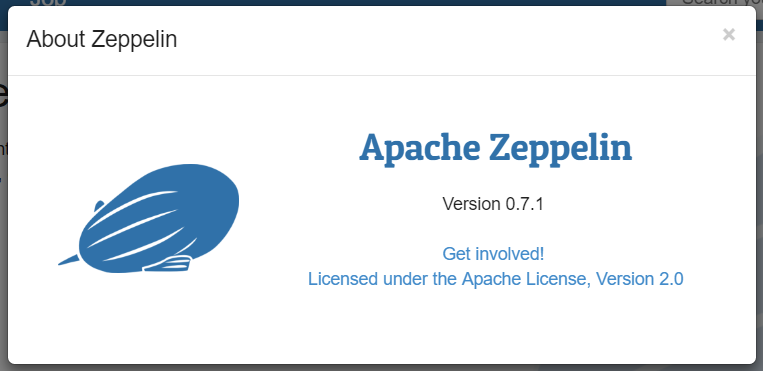
|  |  |
| --- | --- |
| The Notebook menu proposes almost the same features as the note management section in the home page. From the drop-down menu you can:   1. Open a selected note 2. Filter node by name 3. Create a new note |  |

### Settings

|  |  |
| --- | --- |
| This menu gives you access to settings and displays information about Zeppelin. User name is set to anonymous if you use default shiro configuration. If you want to set up authentification, see [Shiro authentication](https://zeppelin.apache.org/docs/0.7.1/security/shiroauthentication.html). |  |

**About Zeppelin**

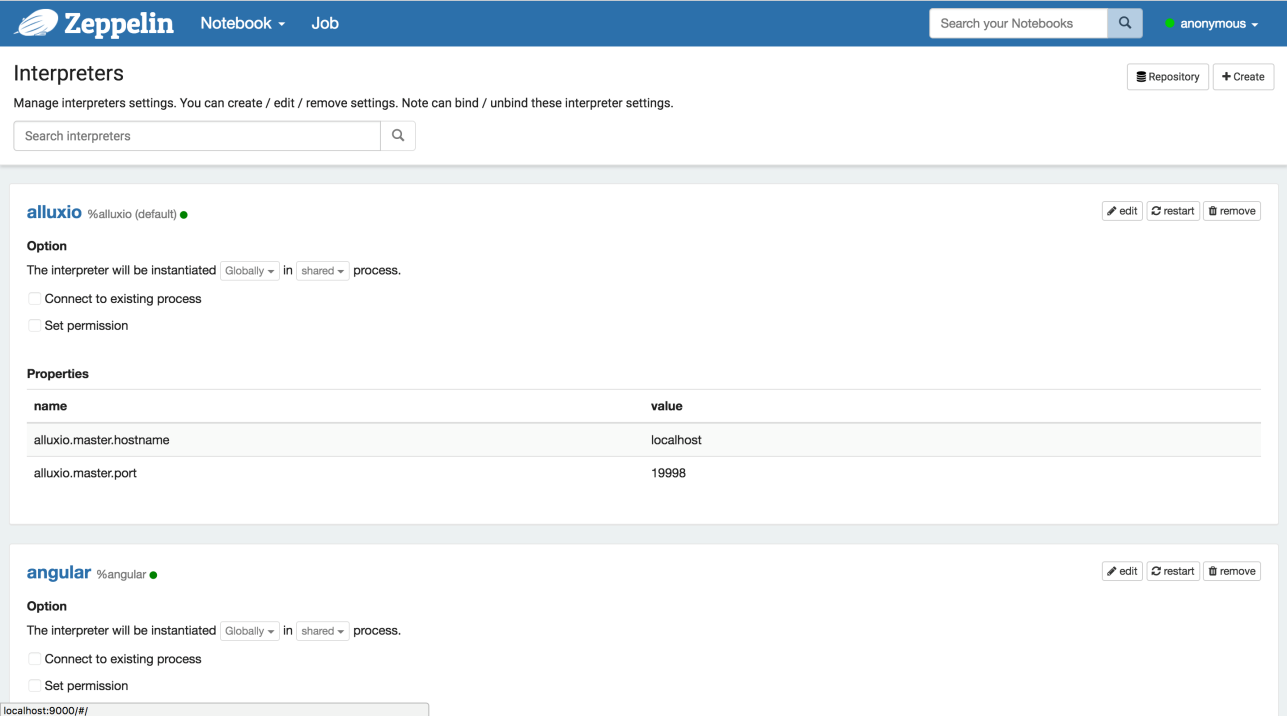
You can check Zeppelin version in this menu.



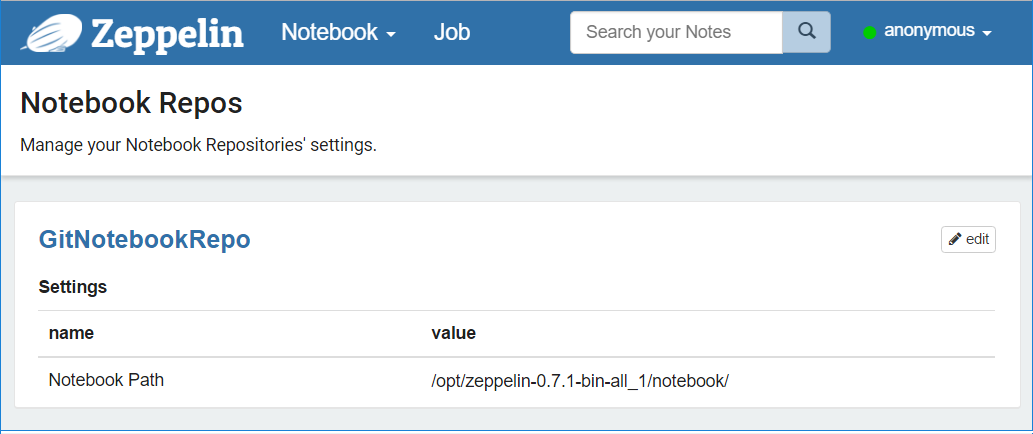
**Interpreter**

In this menu you can:

1. Configure existing **interpreter instance**
2. Add/remove **interpreter instances**

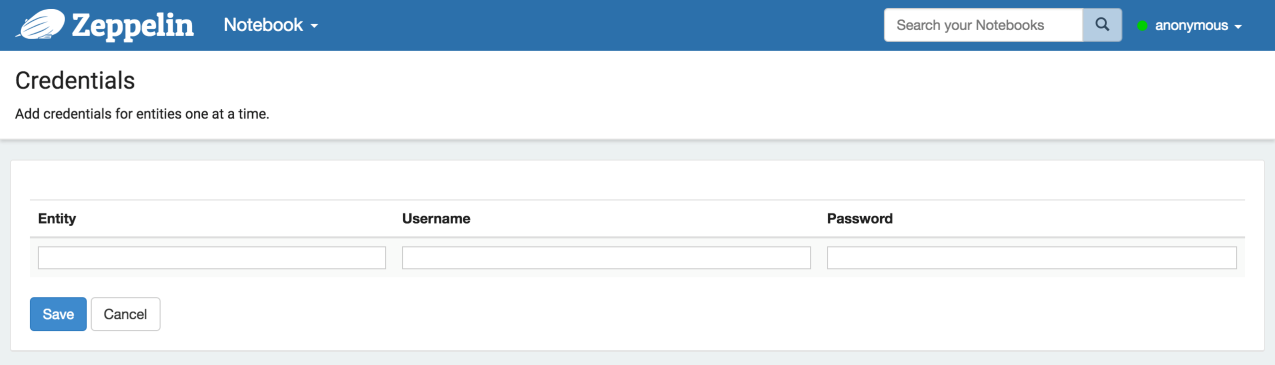


**Notebook Repos**

****

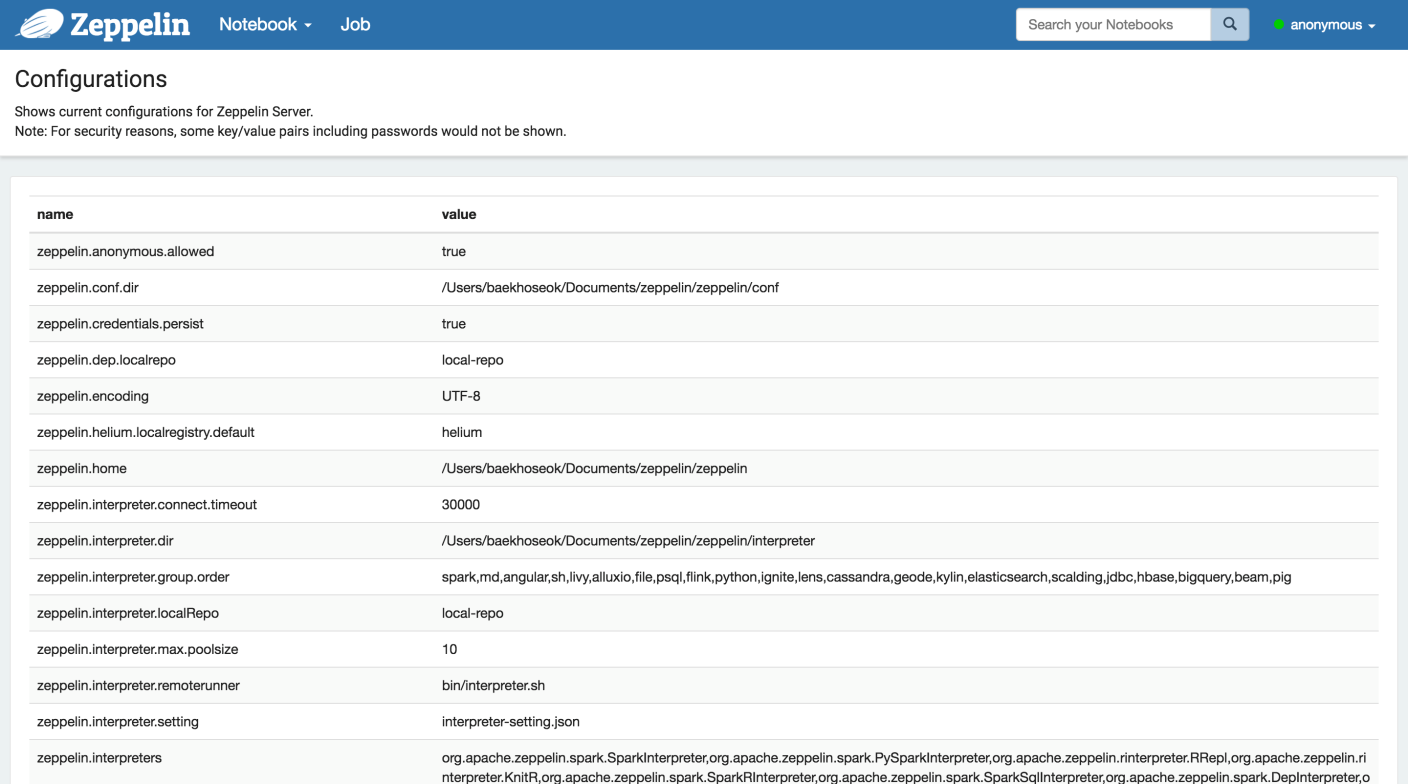
**Credential**

This menu allows you to save credentials for data sources which are passed to interpreters.



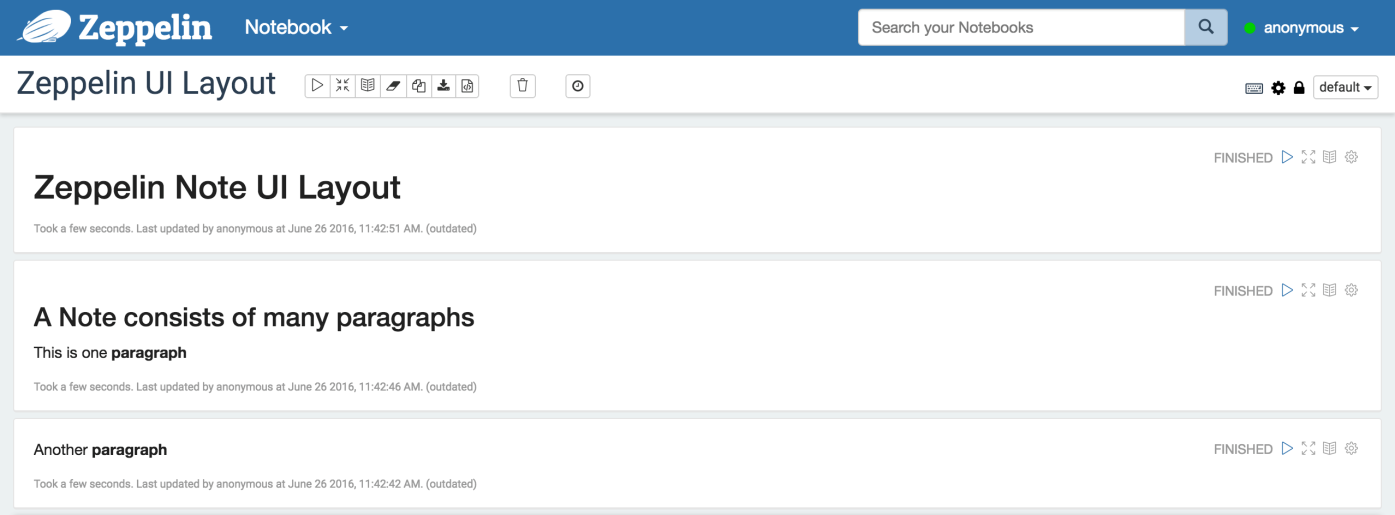
**Configuration**

This menu displays all the Zeppelin configuration that are set in the config file $ZEPPELIN\_HOME/conf/zeppelin-site.xml



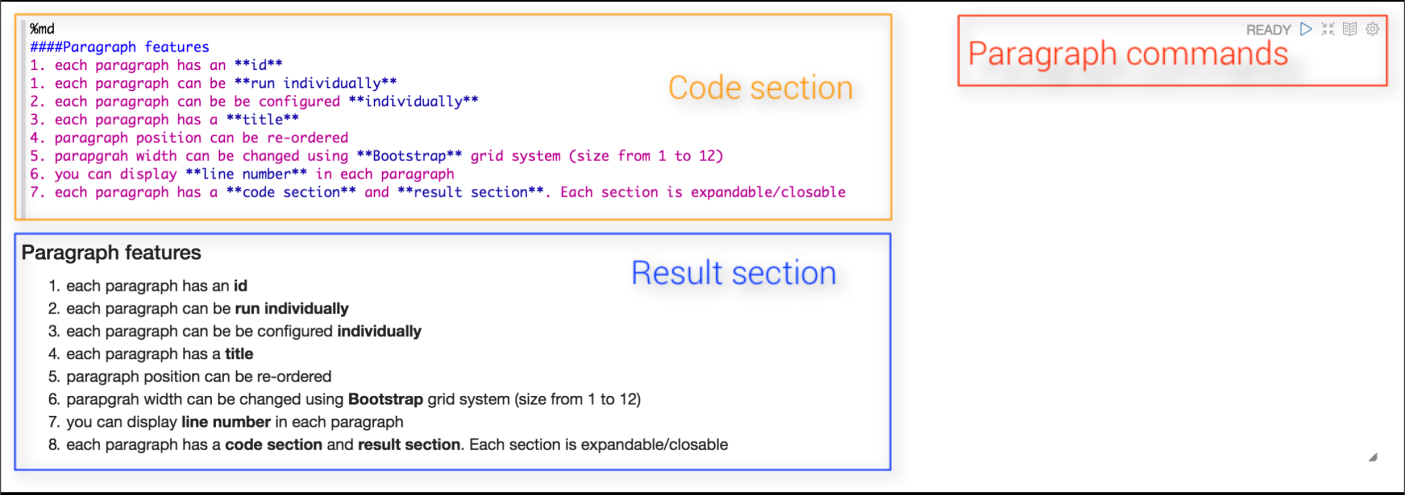
## Note Layout

Each Zeppelin note is composed of 1 .. N paragraphs. The note can be viewed as a paragraph container.



**Paragraph**

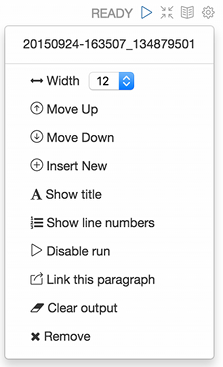
Each paragraph consists of 2 sections: code section where you put your source code and result section where you can see the result of the code execution.



On the top-right corner of each paragraph there are some commands to:

* execute the paragraph code
* hide/show code section
* hide/show result section
* configure the paragraph

To configure the paragraph, just click on the gear icon:



From this dialog, you can (in descending order):

* find the **paragraph id** ( **20150924-163507\_134879501** )
* control paragraph width. Since Zeppelin is using the grid system of **Twitter Bootstrap**, each paragraph width can be changed from 1 to 12
* move the paragraph 1 level up
* move the paragraph 1 level down
* create a new paragraph
* change paragraph title
* show/hide line number in the code section
* disable the run button for this paragraph
* export the current paragraph as an **iframe** and open the **iframe** in a new window
* clear the result section
* delete the current paragraph

**Note toolbar**

At the top of the note, you can find a toolbar which exposes command buttons as well as configuration, security and display options.



On the far right is displayed the note name, just click on it to reveal the input form and update it.

In the middle of the toolbar you can find the command buttons:

* execute all the paragraphs **sequentially**, in their display order
* hide/show code section of all paragraphs
* hide/show result section of all paragraphs
* clear the result section of all paragraphs
* clone the current note
* export the current note to a JSON file. \_Please note that the code section and result section of all paragraphs will be exported. If you have heavy data in the result section of some paragraphs, it is recommended to clean them before exporting
* commit the current node content
* delete the note
* schedule the execution of **all paragraph** using a CRON syntax

https://zeppelin.apache.org/docs/0.7.1/assets/themes/zeppelin/img/ui-img/note_commands.png

On the right of the note tool bar you can find configuration icons:

* display all the keyboard shorcuts
* configure the interpreters binding to the current note
* configure the note permissions
* switch the node display mode between default, simple and report

# MongoDB Interpreter for Apache Zeppelin

## Download

The interpreter can be downloaded from this URL:

<https://github.com/bbonnin/zeppelin-mongodb-interpreter/releases>

After downloaded, rename the file from **zeppelin-mongodb-0.7.0.zip** to **zeppelin-mongodb-0.7.0.jar.**

## Deployment

* Update $ZEPPELIN\_HOME/conf/zeppeln-site.xml

<property>

<name>zeppelin.interpreters</name>

<value>...,**org.apache.zeppelin.mongodb.MongoDbInterpreter**</value>

</property>

* Create $ZEPPELIN\_HOME/interpreter/mongodb
* Copy interpreter jar in $ZEPPELIN\_HOME/interpreter/mongodb

## Configuration

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Default value** | **Description** |
| mongo.shell.path | mongo | Mongo shell path |
| mongo.shell.command.timeout | 60000 | Mongo command timeout |
| mongo.shell.command.table.limit | 1000 | Limit of documents displayed in a table |
| mongo.server.database | test | MongDB database name |
| mongo.server.host | localhost | Host of the MongDB server |
| mongo.server.port | 27017 | Port of the MongDB server |
| mongo.server.username |  | Username for authentication |
| mongo.server.password |  | Password for authentication |
| mongo.server.authentdatabase |  | Database used for authentication |

## MongoDB Installation Verification

First check MongoDB version:

$ mongod –version

vagrant@packer-virtualbox-iso-1467827111:~$ mongod --version

db version v3.0.14

git version: 08352afcca24bfc145240a0fac9d28b978ab77f3

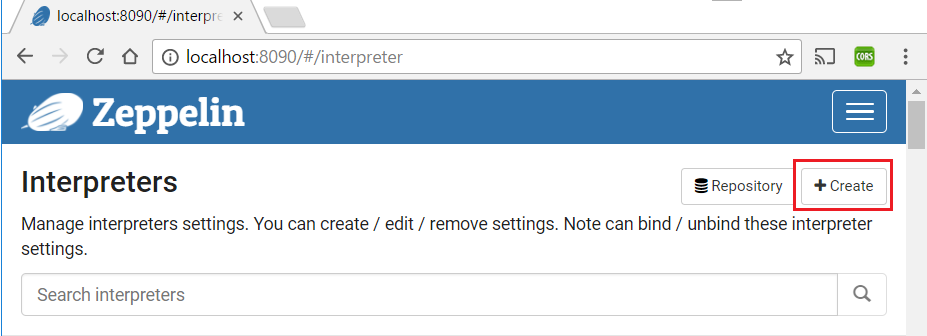
Then check if MongoDB is running:

ps -ef | grep mongod | grep -v grep | wc -l | tr -d ' '

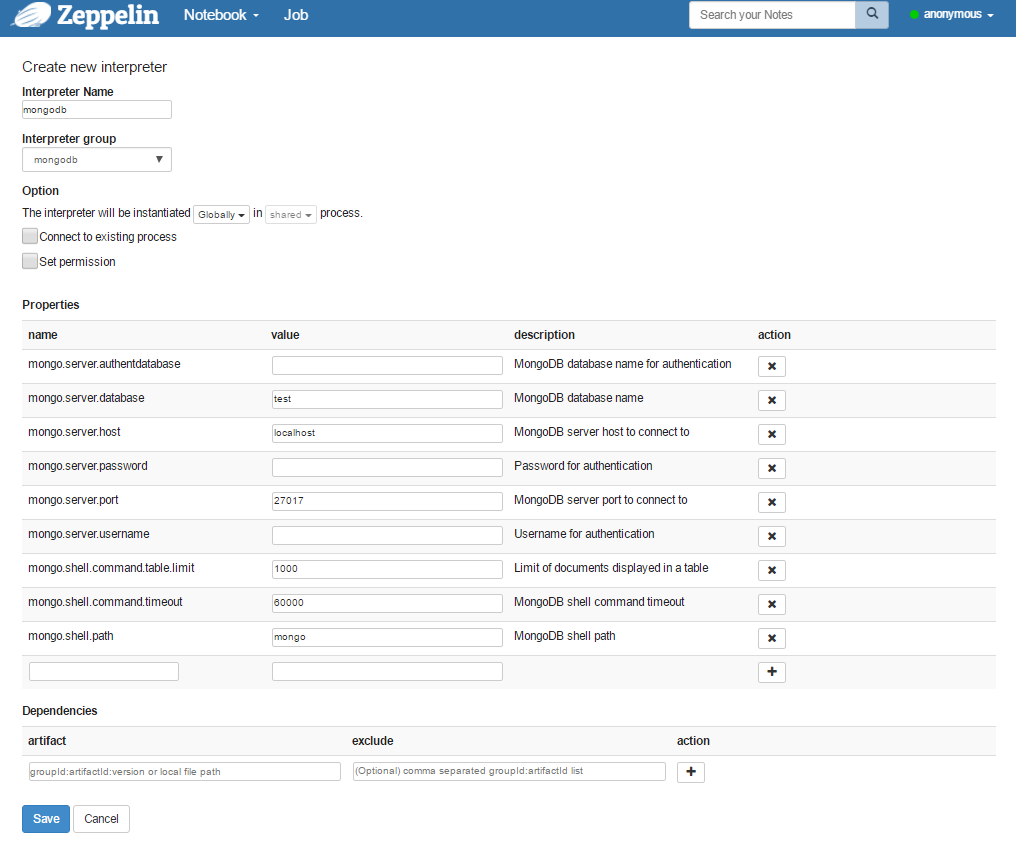
This will get you the number of MongoDB processes running, thus if it is other than 0, then you have MongoDB running on your system.

## Create MongoDB Interpreter

Open Zeppelin Interpreters page, and click the **Create** button.



Enter interpreter name **mongodb**, and select the **mongodb** from the Interpreter Group. Default property values will be populated automatically. Click the **Save** button.



## Load Sample Data into MongoDB

Import sample data zips.json into MongoDB as collection zipcodes.

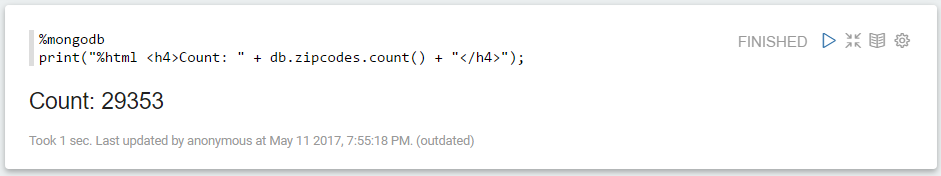
$ mongoimport --db test --collection zipcodes --file /home/vagrant/zips.json

2017-05-11T19:52:29.430-0400 connected to: localhost

2017-05-11T19:52:30.032-0400 imported 29353 documents

vagrant@packer-virtualbox-iso-1467827111:~$

After import, check the import result using Zeppelin UI:



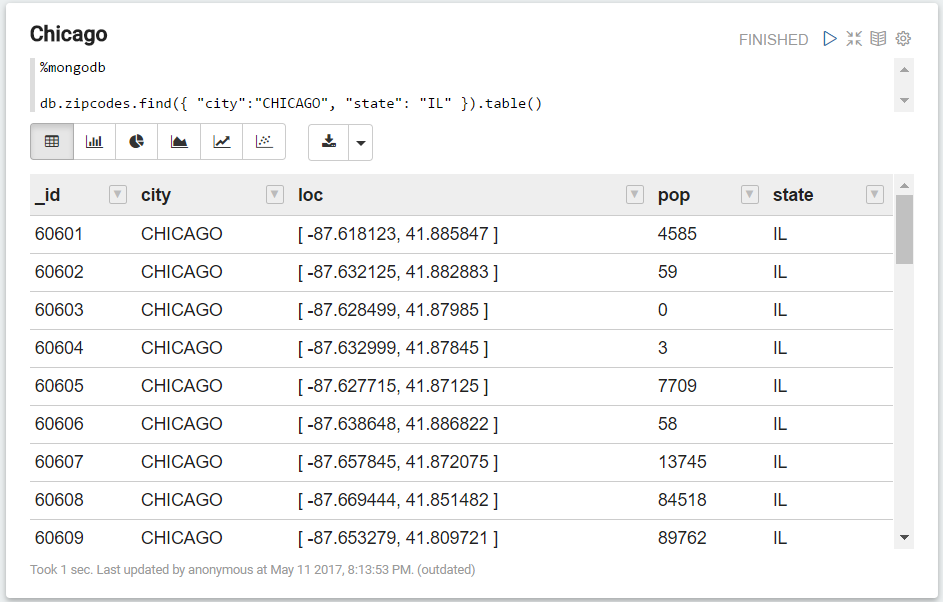
## Generate Sample Reports

Code samples are from this web site: https://github.com/bbonnin/zeppelin-mongodb-interpreter

### Display a Table

%mongodb

db.zipcodes.find({ "city":"CHICAGO", "state": "IL" }).table()



### Display a Bar Chart

Display all states with total population less than 1 million:

%mongodb

var states = db.zipcodes.aggregate( [

{ $group: { \_id: "$state", totalPop: { $sum: "$pop" } } },

{ $match: { totalPop: { $lt: 1000\*1000 } } },

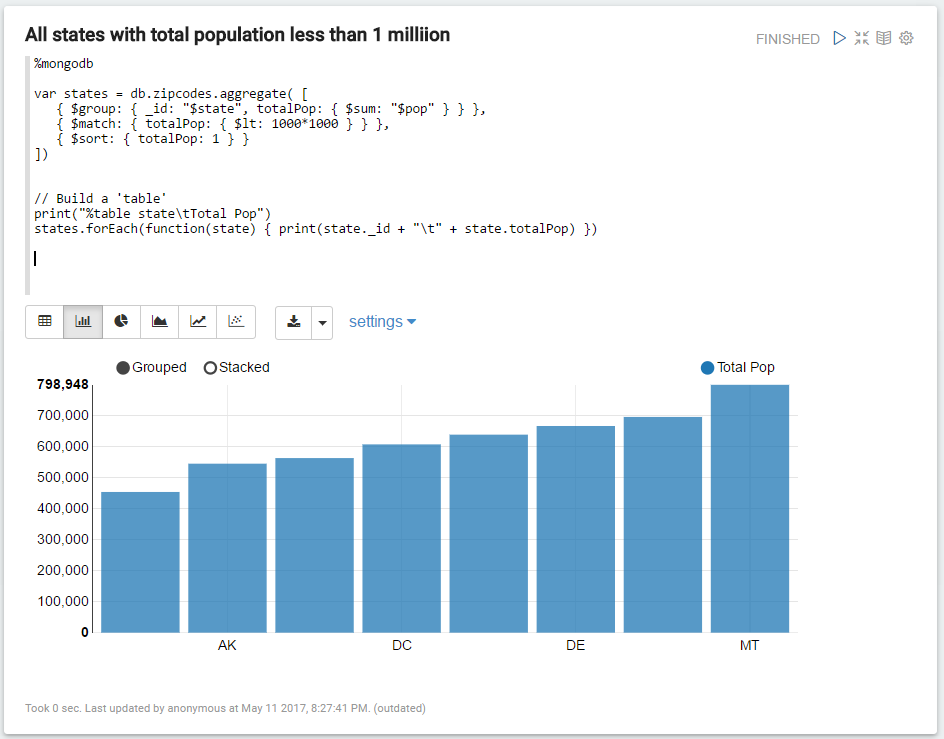
{ $sort: { totalPop: 1 } }

])

// Build a 'table'

print("%table state\tTotal Pop")

states.forEach(function(state) { print(state.\_id + "\t" + state.totalPop) })



### Display a Pie Chart

Display all states with total population greater than 10 millions:

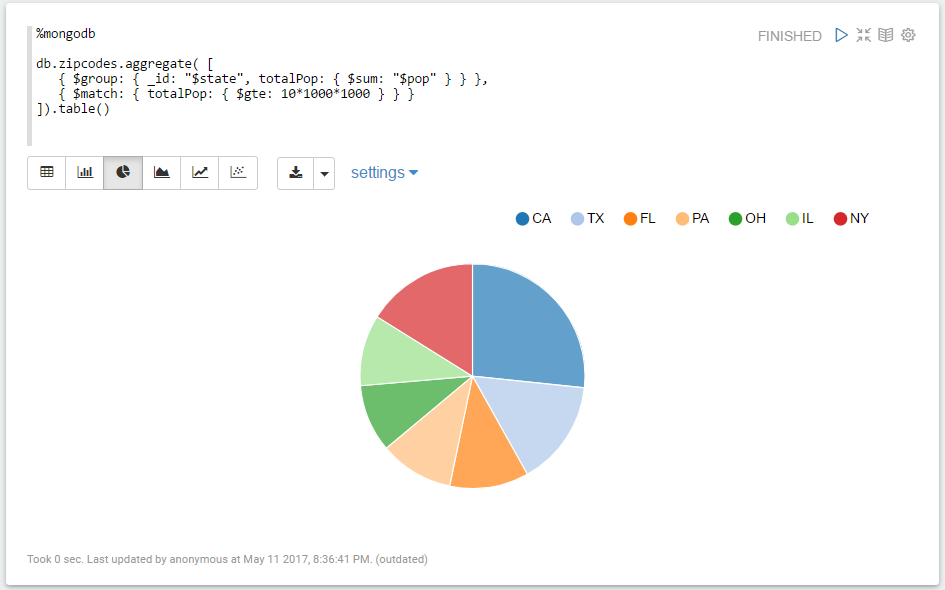
%mongodb

db.zipcodes.aggregate( [

{ $group: { \_id: "$state", totalPop: { $sum: "$pop" } } },

{ $match: { totalPop: { $gte: }\*1000\*1000 } } }

]).table()



# Generic JDBC Interpreter for Apache Zeppelin

(https://zeppelin.apache.org/docs/0.7.0/interpreter/jdbc.html#hive)

## Overview

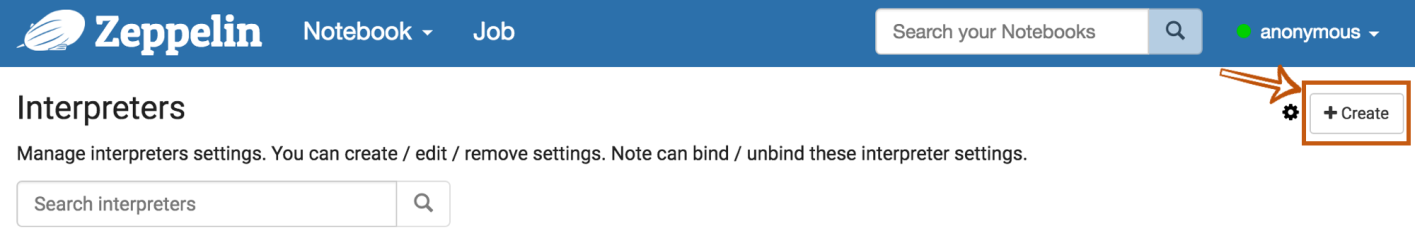
JDBC interpreter lets you create a JDBC connection to any data sources seamlessly. Inserts, Updates, and Upserts are applied immediately after running each statement. By now, it has been tested with:

|  |  |
| --- | --- |
| https://zeppelin.apache.org/docs/0.7.0/assets/themes/zeppelin/img/docs-img/tested_databases.png | [Postgresql](http://www.postgresql.org/) - [JDBC Driver](https://jdbc.postgresql.org/)   [Mysql](https://www.mysql.com/) - [JDBC Driver](https://dev.mysql.com/downloads/connector/j/)   [MariaDB](https://mariadb.org/) - [JDBC Driver](https://mariadb.com/kb/en/mariadb/about-mariadb-connector-j/)   [Redshift](https://aws.amazon.com/documentation/redshift/) - [JDBC Driver](https://docs.aws.amazon.com/redshift/latest/mgmt/configure-jdbc-connection.html)   [Apache Hive](https://hive.apache.org/) - [JDBC Driver](https://cwiki.apache.org/confluence/display/Hive/HiveServer2+Clients#HiveServer2Clients-JDBC)   [Apache Phoenix](https://phoenix.apache.org/) itself is a JDBC driver   [Apache Drill](https://drill.apache.org/) - [JDBC Driver](https://drill.apache.org/docs/using-the-jdbc-driver)   [Apache Tajo](http://tajo.apache.org/) - [JDBC Driver](https://tajo.apache.org/docs/current/jdbc_driver.html) |

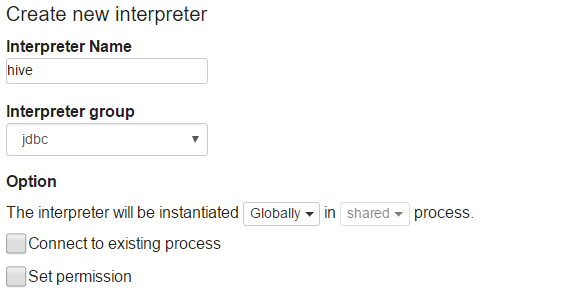
If you are using other databases not in the above list, please feel free to share your use case. It would be helpful to improve the functionality of JDBC interpreter.

## Create a new JDBC Interpreter

First, click + Create button at the top-right corner in the interpreter setting page.



Fill Interpreter name field with whatever you want to use as the alias(e.g. mysql, mysql2, hive, redshift, and etc..). Please note that this alias will be used as %interpreter\_name to call the interpreter in the paragraph. Then select jdbc as an Interpreter group.

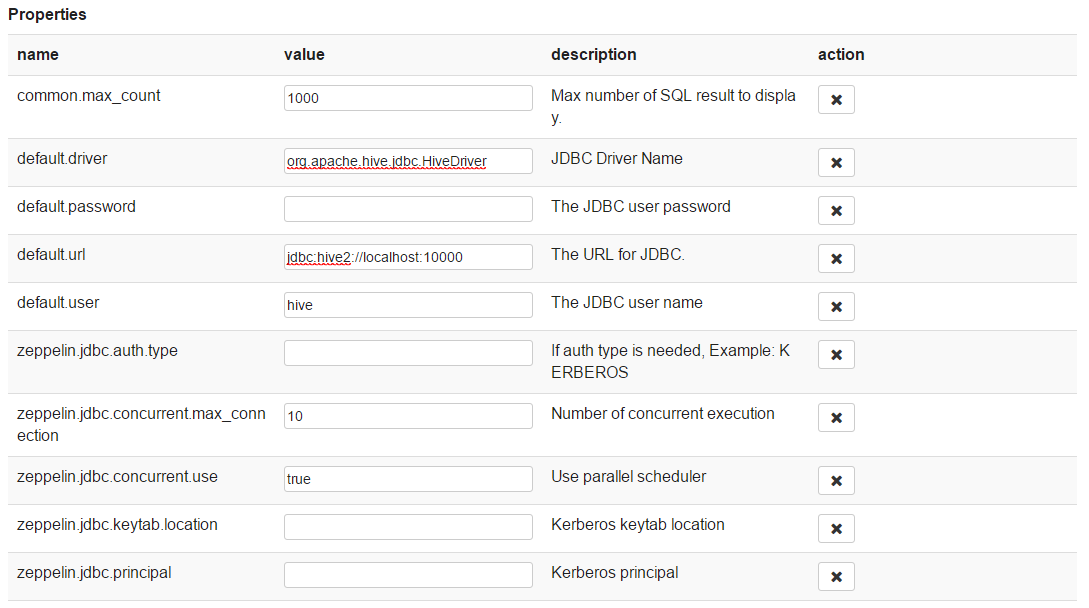


The default driver of JDBC interpreter is set as PostgreSQL. It means Zeppelin includes PostgreSQL driver jar in itself. So you don't need to add any dependencies (e.g. the artifact name or path for PostgreSQL driver jar) for PostgreSQL connection. The JDBC interpreter properties are defined by default like below.

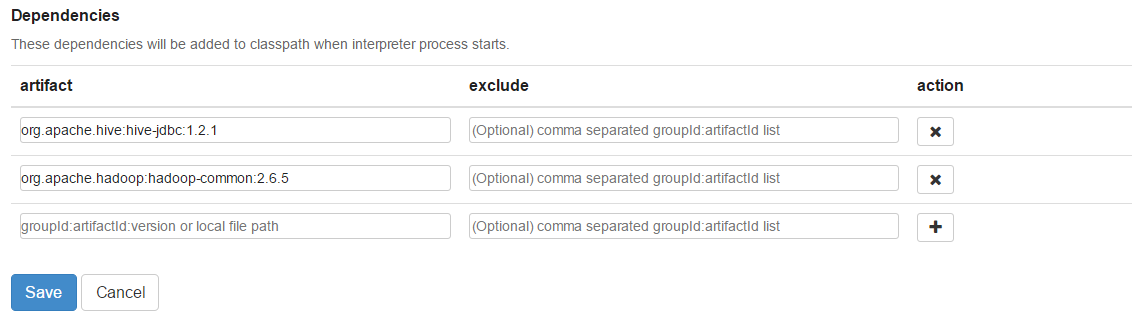
|  |  |  |
| --- | --- | --- |
| **Name** | **Default Value** | **Description** |
| common.max\_count | 1000 | The maximun number of SQL result to display |
| default.driver | org.postgresql.Driver | JDBC Driver Name |
| default.password |  | The JDBC user password |
| default.url | jdbc:postgresql://localhost:5432/ | The URL for JDBC |
| default.user | gpadmin | The JDBC user name |

If you want to connect other databases such as Mysql, Redshift and Hive, you need to edit the property values. You can also use [Credential](https://zeppelin.apache.org/docs/0.7.0/security/datasource_authorization.html) for JDBC authentication. If default.user and default.password properties are deleted(using X button) for database connection in the interpreter setting page, the JDBC interpreter will get the account information from [Credential](https://zeppelin.apache.org/docs/0.7.0/security/datasource_authorization.html).

The below example is for Hive connection.



The last step is **Dependency Setting**. Since Zeppelin only includes PostgreSQL driver jar by default, you need to add each driver's maven coordinates or JDBC driver's jar file path for the other databases.



That's it. You can find more JDBC connection setting examples ([Mysql](https://zeppelin.apache.org/docs/0.7.0/interpreter/jdbc.html" \l "mysql), [MariaDB](https://zeppelin.apache.org/docs/0.7.0/interpreter/jdbc.html#mariadb), [Redshift](https://zeppelin.apache.org/docs/0.7.0/interpreter/jdbc.html#redshift), [Apache Hive](https://zeppelin.apache.org/docs/0.7.0/interpreter/jdbc.html#apache-hive), [Apache Phoenix](https://zeppelin.apache.org/docs/0.7.0/interpreter/jdbc.html#apache-phoenix), and [Apache Tajo](https://zeppelin.apache.org/docs/0.7.0/interpreter/jdbc.html#apache-tajo)) in [this section](https://zeppelin.apache.org/docs/0.7.0/interpreter/jdbc.html#examples).

## More Properties

There are more JDBC interpreter properties you can specify like below.

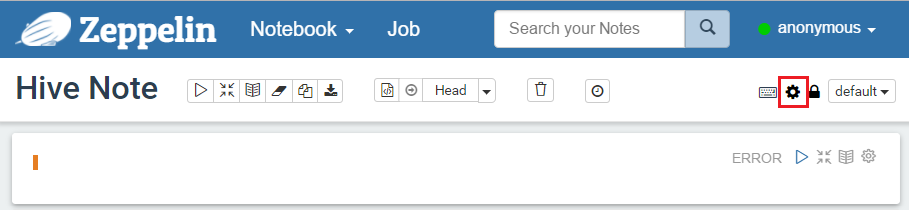
|  |  |
| --- | --- |
| **Property Name** | **Description** |
| common.max\_result | Max number of SQL result to display to prevent the browser overload. This is common properties for all connections |
| zeppelin.jdbc.auth.type | Types of authentications' methods supported are SIMPLE, and KERBEROS |
| zeppelin.jdbc.principal | The principal name to load from the keytab |
| zeppelin.jdbc.keytab.location | The path to the keytab file |
| default.jceks.file | jceks store path (e.g: jceks://file/tmp/zeppelin.jceks) |
| default.jceks.credentialKey | jceks credential key |

You can also add more properties by using this [method](http://docs.oracle.com/javase/7/docs/api/java/sql/DriverManager.html#getConnection%28java.lang.String,%20java.util.Properties%29). For example, if a connection needs a schema parameter, it would have to add the property as follows:

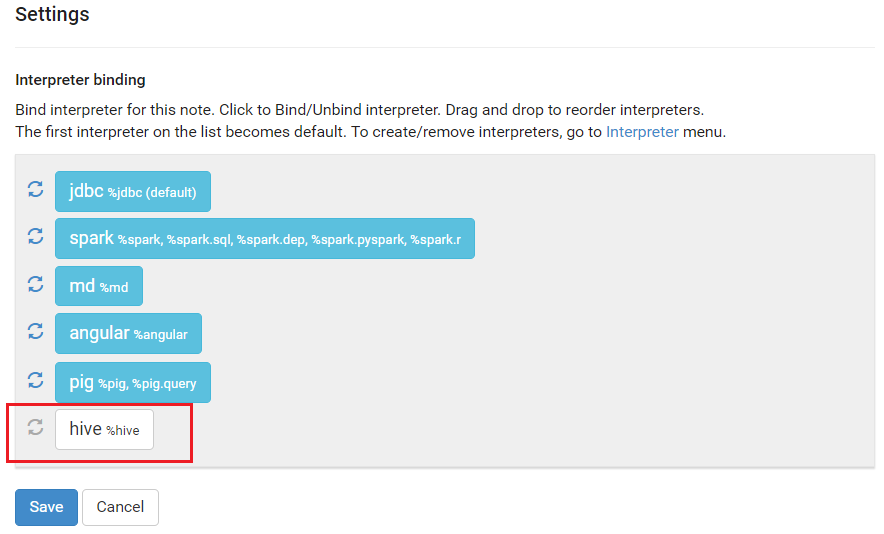
|  |  |
| --- | --- |
| **name** | **value** |
| default.schema | schema\_name |

## Binding JDBC Interpreter to Notebook

To bind the interpreters created in the interpreter setting page, click the gear icon at the top-right corner.



Select (blue) or deselect (white) the interpreter buttons depending on your use cases. If you need to use more than one interpreter in the notebook, activate several buttons. Don't forget to click Save button, or you will face Interpreter \*\*\* is not found error.



## Test Database and Zeppelin Connection

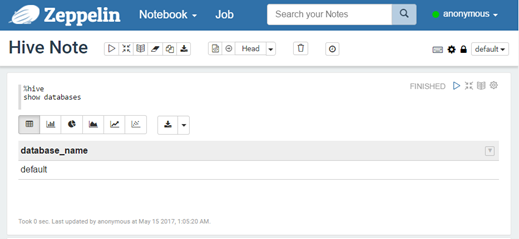
### Run the paragraph with JDBC interpreter

To test whether your databases and Zeppelin are successfully connected or not, type %jdbc\_interpreter\_name (e.g. %hive) at the top of the paragraph and run show databases.

%hive

show databases

If the paragraph is FINISHED without any errors, a new paragraph will be automatically added after the previous one with %hive. So you don't need to type this prefix in every paragraph’s header.



## Load Test Data into Hive

Open the Hive shell and enter the following script to create the **weblogs** table.

create table weblogs (

client\_ip string,

full\_request\_date string,

day string,

month string,

month\_num int,

year string,

hour string,

minute string,

second string,

timezone string,

http\_verb string,

uri string,

http\_status\_code string,

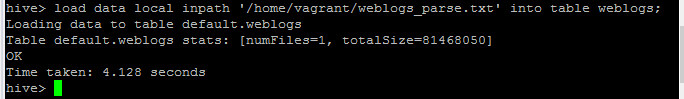
bytes\_returned string,

referrer string,

user\_agent string

) row format delimited fields terminated by '\t';

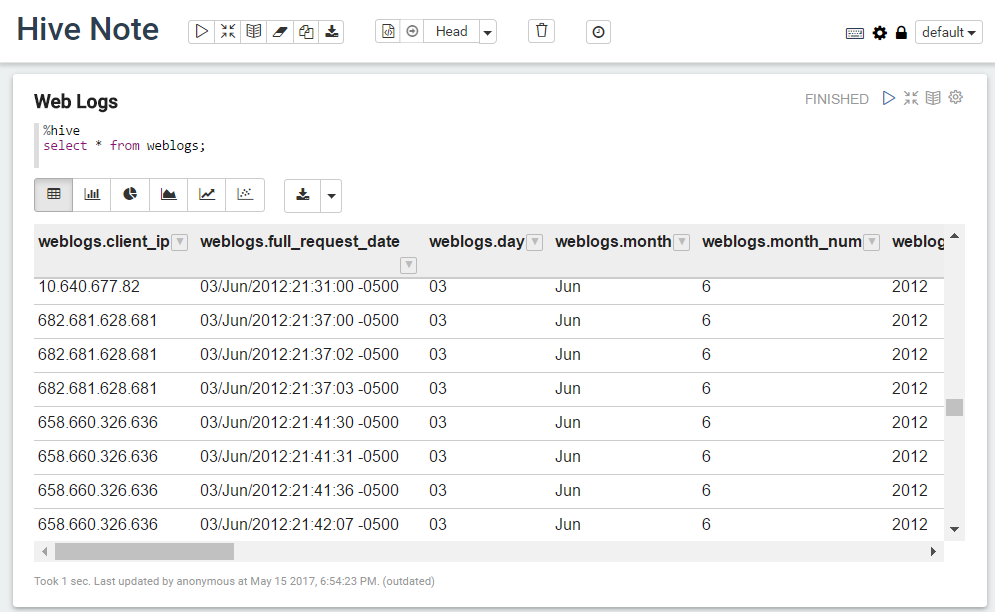
Load test data into Hive table:

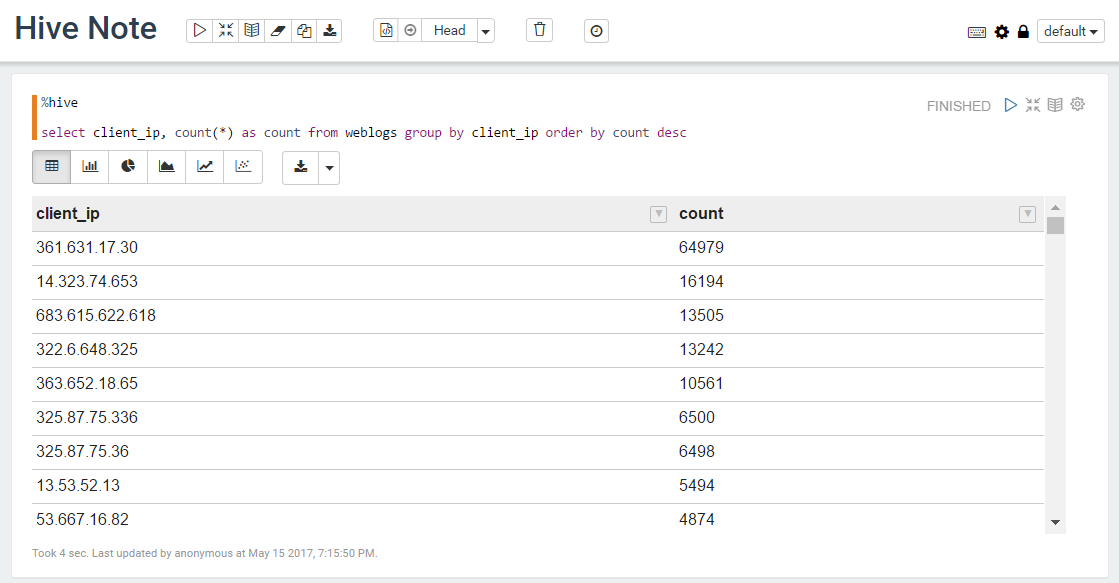


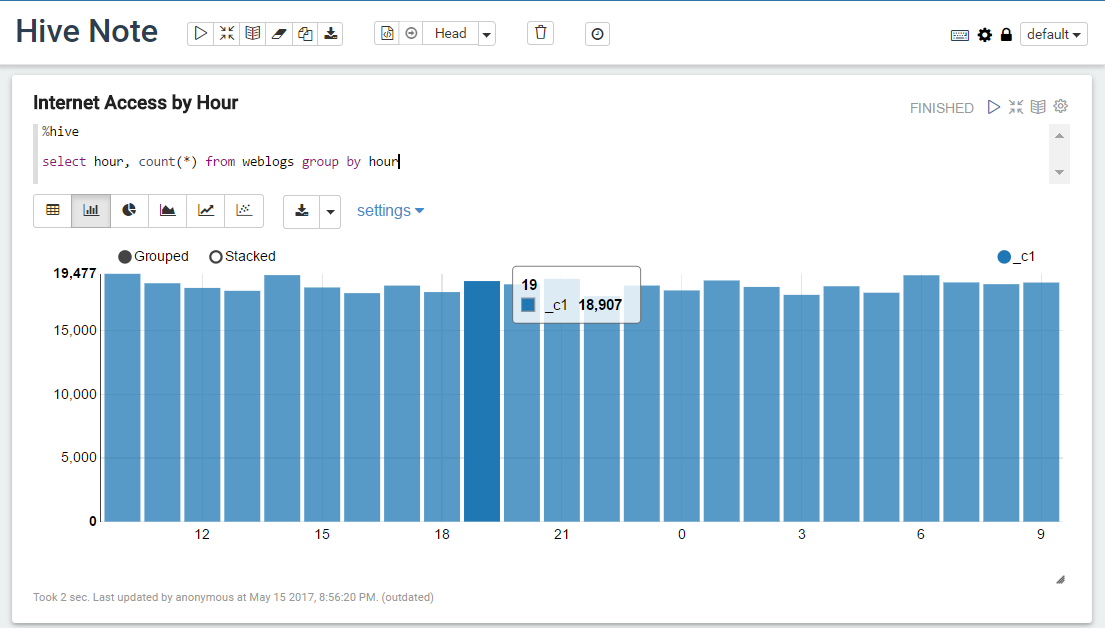
Here are some examples you can refer to

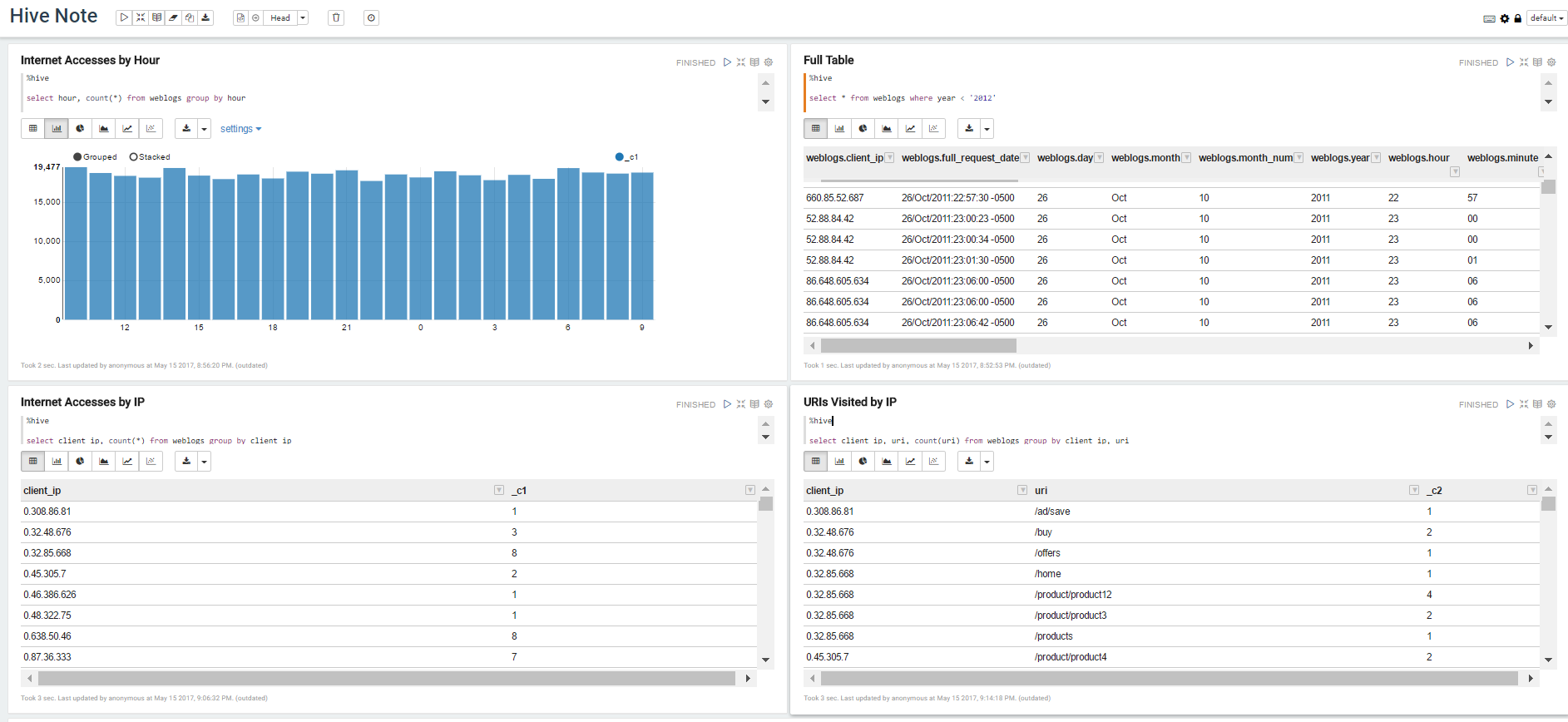
## Generate Sample Reports

### Display Data in Various Format

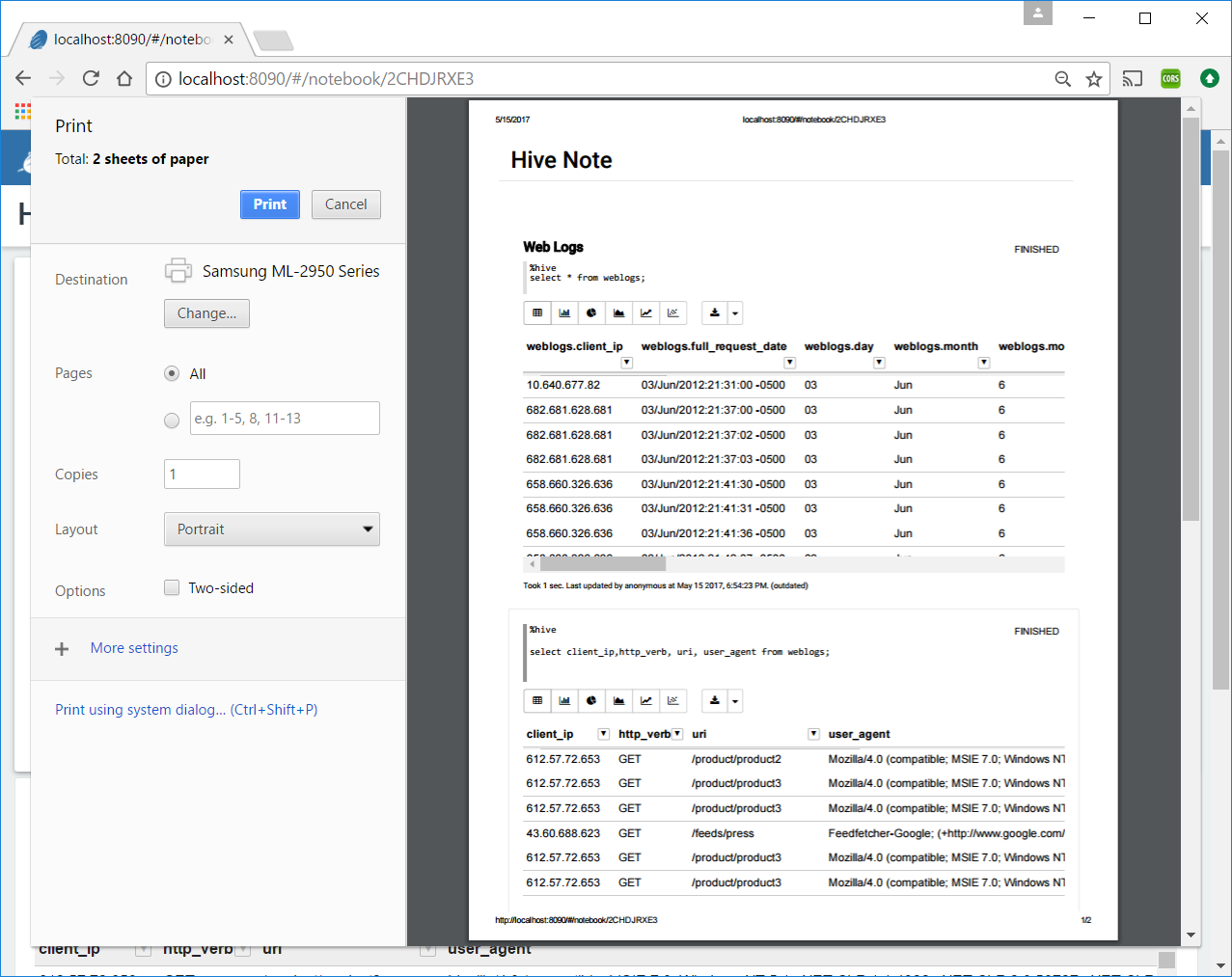








### Print Report



# Apache Zeppelin Dynamic Form

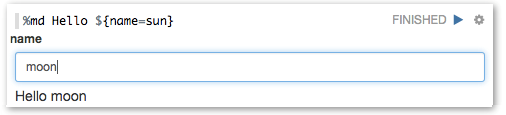
Apache Zeppelin dynamically creates input forms. Depending on language backend, there're two different ways to create dynamic form. Custom language backend can select which type of form creation it wants to use.

## Using form Templates

This mode creates form using simple template language. It's simple and easy to use. For example Markdown, Shell, Spark SQL language backend uses it.

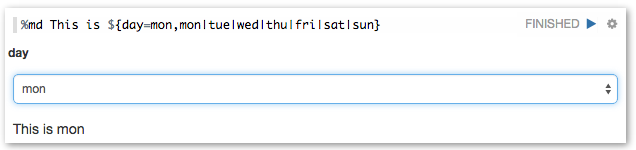
**Text Input**

To create text input field, use ${fieldName} templates. For example,

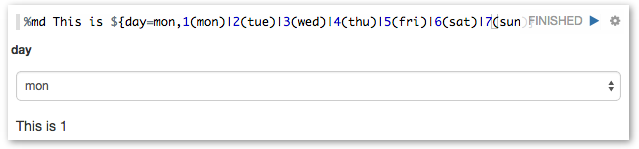


**Drop-down Combo Box**

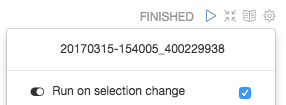
To create select drop-down list, use ${formName=defaultValue,option1|option2...}. For example,



Also you can separate option's display name and value, using ${formName=defaultValue,option1(DisplayName)|option2(DisplayName)...}

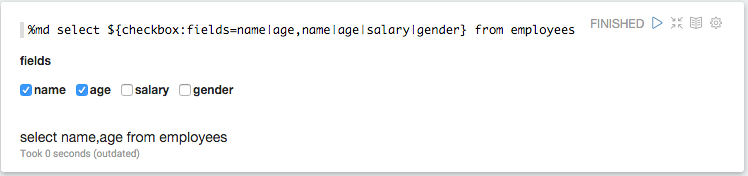


The paragraph will be automatically run after you change your selection by default. But in case you have multiple types dynamic form in one paragraph, you might want to run the paragraph after changing all the selections. You can control this by unchecking the below **Run on selection change** option in the setting menu. Even if you uncheck this option, still you can run it by pressing Enter.

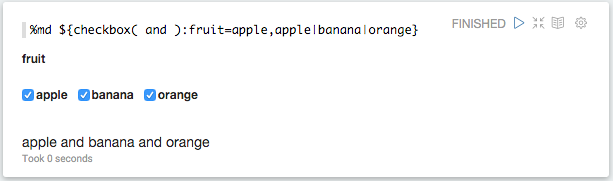


**Checkbox**

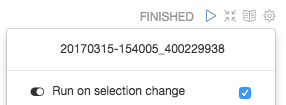
For multi-selection, you can create a checkbox form using ${checkbox:formName=defaultValue1|defaultValue2...,option1|option2...}. The variable will be substituted by a comma-separated string based on the selected items. For example:



Besides, you can specify the delimiter using ${checkbox(delimiter):formName=...}:



Like [select form](https://zeppelin.apache.org/docs/latest/manual/dynamicform.html#select-form), the paragraph will be automatically run after you change your selection by default. But in case you have multiple types of dynamic form in one paragraph, you might want to run the paragraph after changing all the selections. You can control this by unchecking the below **Run on selection change** option in the setting menu. Even if you uncheck this option, still you can run it by pressing Enter.



## Creates Programmatically

Some language backend uses programmatic way to create form. For example [ZeppelinContext](https://zeppelin.apache.org/docs/latest/interpreter/spark.html#zeppelincontext) provides form creation API

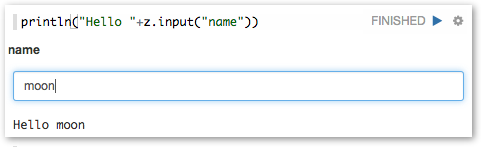
Here're some examples.

**Text Input**

* [**Scala**](https://zeppelin.apache.org/docs/latest/manual/dynamicform.html#tab_scala_0)
* [**Python**](https://zeppelin.apache.org/docs/latest/manual/dynamicform.html#tab_python_0)

%spark

println("Hello "+z.input("name"))

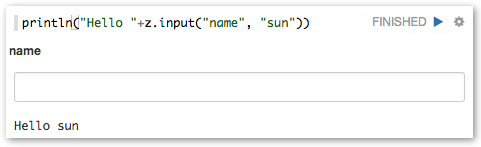


**Text input form with default value**

* [**Scala**](https://zeppelin.apache.org/docs/latest/manual/dynamicform.html#tab_scala_1)
* [**Python**](https://zeppelin.apache.org/docs/latest/manual/dynamicform.html#tab_python_1)

%spark

println("Hello "+z.input("name", "sun"))

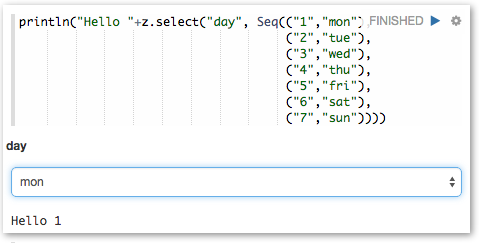


**Drop-down Combo Box**

* [**Scala**](https://zeppelin.apache.org/docs/latest/manual/dynamicform.html#tab_scala_2)
* [**Python**](https://zeppelin.apache.org/docs/latest/manual/dynamicform.html#tab_python_2)

%spark

println("Hello "+z.select("day", Seq(("1","mon"),("2","tue"),("3","wed"), ("4","thurs"), ("5","fri"),("6","sat"), ("7","sun"))))



**Checkbox**

* [**Scala**](https://zeppelin.apache.org/docs/latest/manual/dynamicform.html#tab_scala_3)
* [**Python**](https://zeppelin.apache.org/docs/latest/manual/dynamicform.html#tab_python_3)

%spark

val options = Seq(("apple","Apple"), ("banana","Banana"), "orange","Orange"))

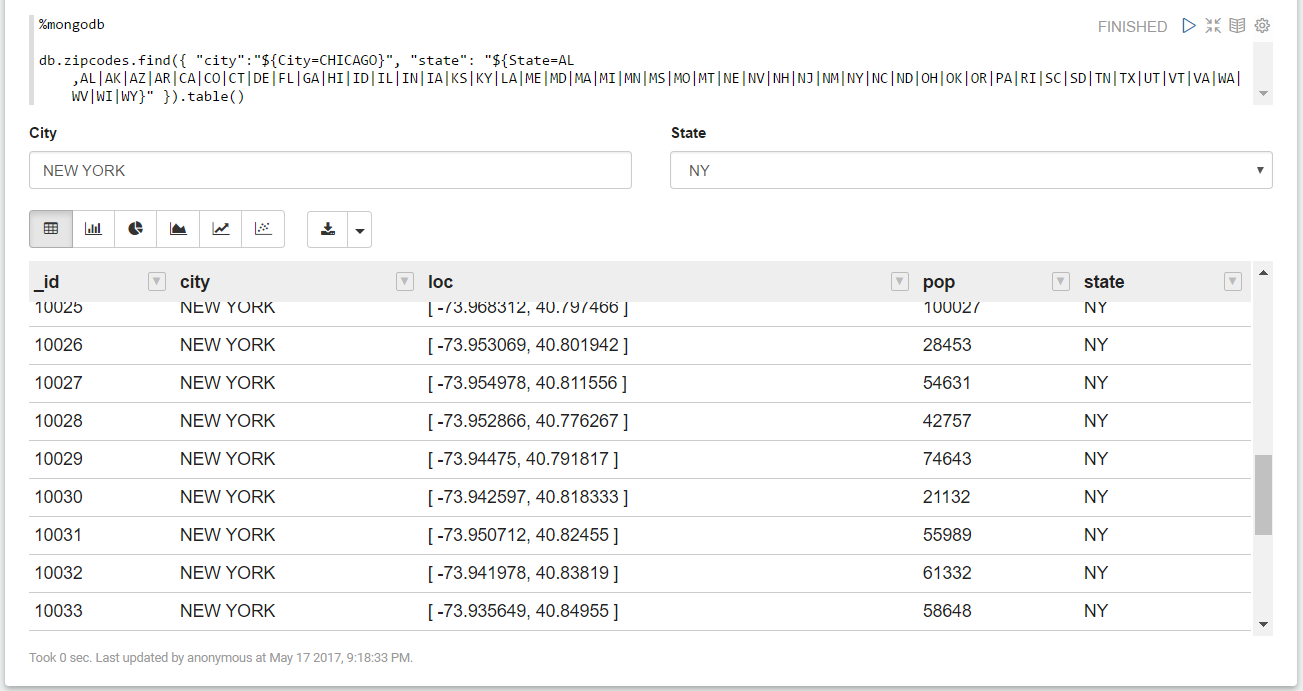
println("Hello "+z.checkbox("fruit", options).mkString(" and "))



## MongoDB Dynamic Form Example

%mongodb

db.zipcodes.find({ "city":"${City=CHICAGO}", "state": "${State=AL,AL|AK|AZ|AR|CA|CO|CT|DE|FL|GA|HI|ID|IL|IN|IA|KS|KY|LA|ME|MD|MA|MI|MN|MS|MO|MT|NE|NV|NH|NJ|NM|NY|NC|ND|OH|OK|OR|PA|RI|SC|SD|TN|TX|UT|VT|VA|WA|WV|WI|WY}" }).table()



%mongodb

db.zipcodes.find({ "city":"${City=CHICAGO}", "state": "${State=AL(Alabama),AL(Alabama)|AK(Alaska)|AZ(Arizona)|AR(Arkansas)|CA(California)|CO(Colorado)|CT(Connecticut)|DE(Delaware)|FL(Florida)|GA(Georgia)|HI(Hawaii)|ID(Idaho)|IL(Illinois)|IN(Indiana)|IA(Iowa)|KS(Kansas)|KY(Kentucky)|LA(Louisiana)|ME(Maine)|MD(Maryland)|MA(Massachusetts)|MI(Michigan)|MN(Minnesota)|MS(Mississippi)|MO(Missouri)|MT(Montana)|NE(Nebraska)|NV(Nevada)|NH(New Hampshire)|NJ(New Jersey)|NM(New Mexico)|NY(New York)|NC(North Carolina)|ND(North Dakota)|OH(Ohio)|OK(Oklahoma)|OR(Oregon)|PA(Pennsylvania)|RI(Rhode Island)|SC(South Carolina)|SD(South Dakota)|TN(Tennessee)|TX(Texas)|UT(Utah)|VT(Vermont)|VA(Virginia)|WA(Washington)|WV(West Virginia)|WI(Wisconsin)|WY(Wyoming)}" }).table()

