

---

# Apache Zeppelin

MSDSF23M037 Talha Hussain

---

# Agenda

- Introduction to Apache Zeppelin
- History
- Key Features
- Architecture
- Zeppelin Interpreter
- Dynamic Form
- Customize homepage

- Use Cases
- Installation and Configuration
- Advantages and Limitations
- Future Outlook
- Conclusion
- Q&A

# Introduction to Apache Zeppelin

## Definition:

Apache Zeppelin is a web-based notebook for data analysis, allowing users to create and share live code, visualizations, and text.



## Purpose and Importance:

- Interactive Data Exploration: Execute code in multiple languages and visualize results instantly.
- Collaboration: Share insights and work together in real-time.
- Integration: Works with various data processing tools like Apache Spark.
- Visualization: Provides rich, interactive visualizations.



## **Brief History:**

### **Development and Contributions:**

- Developed by Moon Soo Lee in 2014.
- Enhanced by a global community of contributors.

### **Apache Software Foundation:**

- Became an official ASF project in 2015, ensuring open-source standards and community-driven development.

---

# Key Features

## 1- Multiple Language Backend

Apache Zeppelin interpreter concept allows any Language / data-processing-backend to be plugged into Zeppelin. Currently Apache Zeppelin supports many interpreters such as Apache Spark, Apache Flink, Python, R, JDBC, Markdown and Shell.



```
val s = "Scala with built-in Apache Spark Integration"
s: String = Scala with built-in Apache Spark Integration
```

Took 0 seconds

```
%pyspark
print "Python with built-in Apache Spark Integration"
```

Python with built-in Apache Spark Integration

```
%sql -- built-in SparkSQL Support
select * from RDD
```

```
%md Markdown
```

Markdown

```
%sh echo "Shell"
```

Shell

Took 0 seconds

# 1- Multiple Language Backend

Apache Zeppelin a valuable tool for data scientists, analysts, and developers, as it provides a unified interface for working with different languages and data processing frameworks. Additionally, Zeppelin's collaborative features, such as notebook sharing and interactive visualization, further enhance its capabilities for team collaboration and data exploration.

---

# Key Features

## 2- Integrated Data Processing

Apache Zeppelin provides built-in Apache Spark integration. You don't need to build a separate module, plugin or library for it.



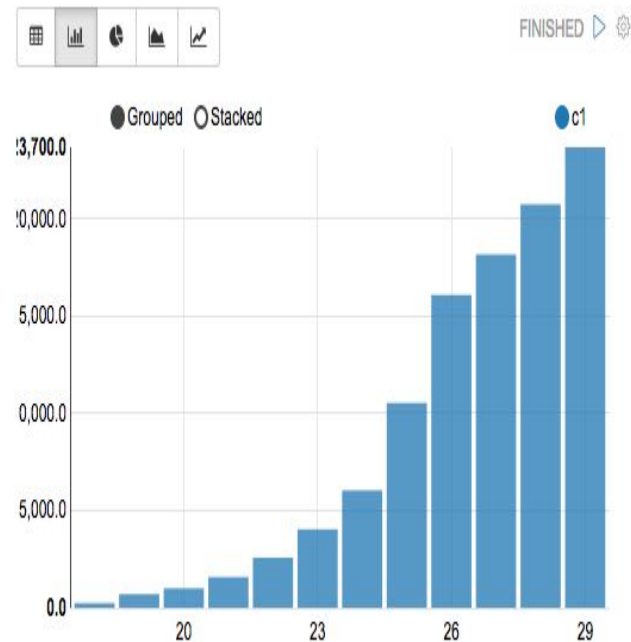
- Automatic SparkContext and SQLContext injection
  - Runtime jar dependency loading from local filesystem or maven repository.  
Learn more about dependency loader.
  - Canceling job and displaying its progress
-

---

# Key Features

## 3- Data visualization

- Offers dynamic and interactive visualizations including graphs, charts, and forms that update in real-time.
- Some basic charts are already included in Zeppelin. Visualizations are not limited to Spark SQL query, any output from any language backend can be recognized and visualized.

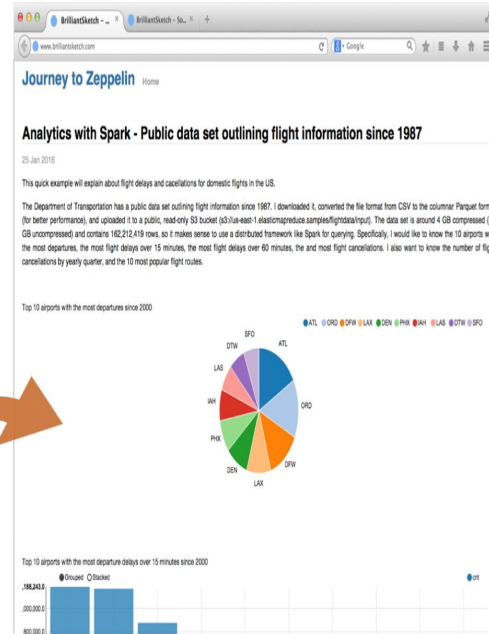
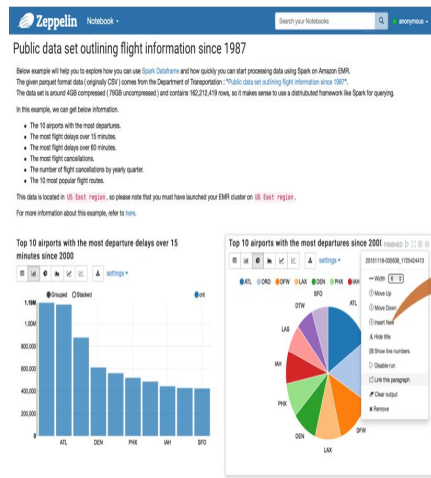




# Key Features

## 4- 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.
- Apache Zeppelin provides an URL to display the result only



---

# Key Features

## 5- Dynamic forms

```
%md Hello ${name=sun}
```

FINISHED ▶ ⌵ ⌶ ⚙

name

Hello moon

```
%spark
```

FINISHED ▶ ⌵ ⌶ ⚙

```
println("Today is "+z.select("day", Seq(("Monday", "1"),  
    ("Tuesday", "2"),  
    ("Wednesday", "3"),  
    ("Thursday", "4"),  
    ("Friday", "5"),  
    ("Saturday", "6"),  
    ("Sunday", "7"))))
```

day

Today is Friday

---

# Architecture

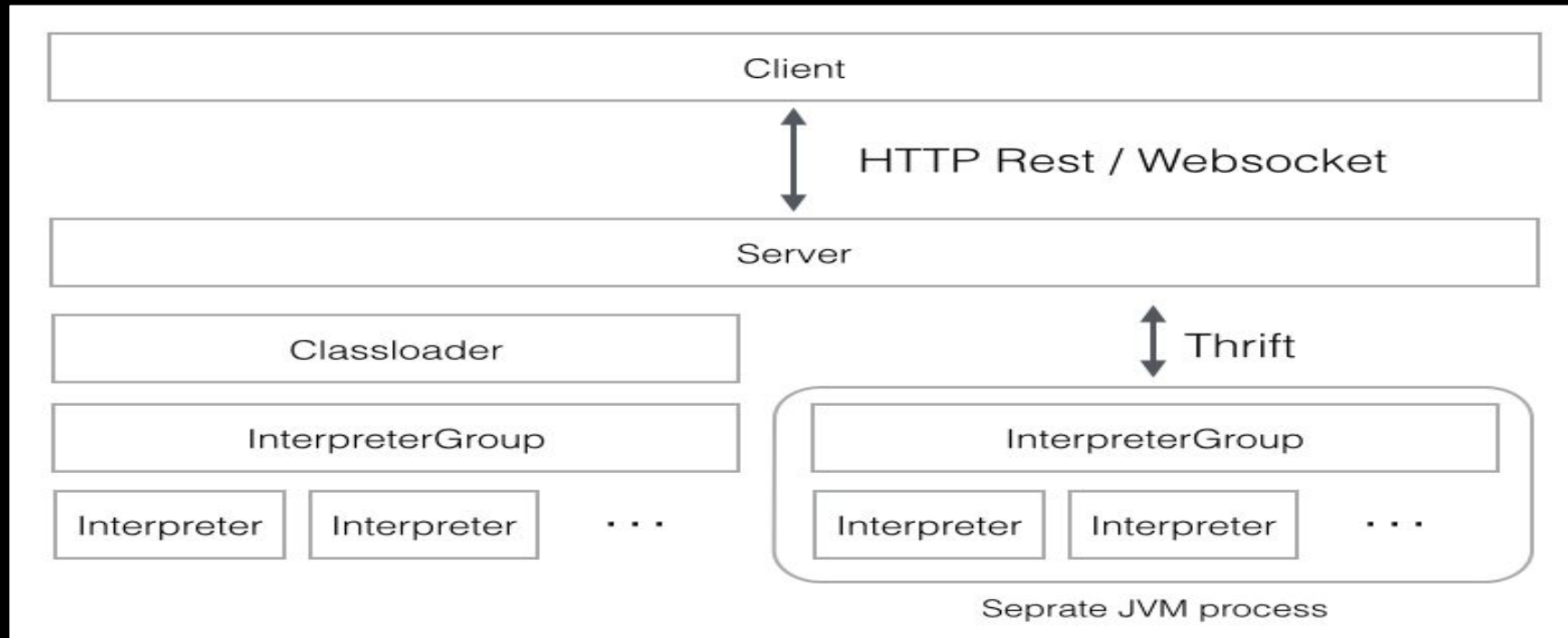
(Core Components)

**1- Zeppelin Server:**  
Acts as the central hub, managing communication between users, interpreters, and notebooks.

**2 - Interpreter**  
**Process:** Responsible for executing code within notebooks. Interpreter correspond to specific language or technology.

**3 - Notebook Storage:**  
Stores notebooks, which contain code, text, and visualizations. This is where users create and edit their work.

# Architecture



# How it Works:

## Interaction between Components

Zeppelin Server coordinates requests from users, directing them to the appropriate interpreter process. It also manages the storage and retrieval of notebooks.

## Workflow of executing a note

- User interacts with Zeppelin Server through the web interface.
- Zeppelin Server communicates with the appropriate interpreter process based on the user's request.
- Interpreter process executes the code within the notebook.
- Results are sent back to the Zeppelin Server and displayed to the user in real-time.

---

## What is Zeppelin Interpreter ?

A Zeppelin interpreter is a plug-in which enables Zeppelin users to use a specific language/data-processing-backend. For example, to use Scala code in Zeppelin, you would use the `%spark` interpreter.

---

# Create New interpreter

## Create new interpreter

Name

Interpreter

Properties

name

Save

Cancel

When you click the `+Create` button on the interpreter page, the interpreter drop-down list box will show all the available interpreters on your server.

You can create multiple interpreters for the same engine with different interpreter setting. e.g. You can create `spark2` for Spark 2.x and create `spark1` for Spark 1.x.

For each paragraph you write in Zeppelin, you need to specify its interpreter first via `%interpreter_group.interpreter_name`. e.g.  
`%spark.pyspark, %spark.r`.

If you specify interpreter, you can also pass local properties to it (if it needs them). This is done by providing a set of key/value pairs, separated by comma, inside the round brackets right after the interpreter name. If key or value contain characters like `=`, or `,`, then you can either escape them with `\` character, or wrap the whole value inside the double quotes For example:

```
%cassandra(outputFormat=cql, dateFormat="E, d MMM yy",  
timeFormat=E\, d MMM yy)
```

---

## What are the Interpreter Settings?

The interpreter settings are the configuration of a given interpreter on the Zeppelin server. For example, certain properties need to be set for the Apache Hive JDBC interpreter to connect to the Hive server.



The screenshot shows the configuration for the 'hive' interpreter. At the top, it says 'hive %hql'. Below this is a section titled 'Properties' which contains a table with two columns: 'name' and 'value'.

name	value
hive.hiveserver2.password	
hive.hiveserver2.url	jdbc:hive2://localhost:10000
hive.hiveserver2.user	hive

Properties are exported as environment variables on the system if the property name consists of upper-case characters, numbers or underscores ([A-Z\_0-9]). Otherwise, the property is set as a common interpreter property. e.g. You can define `SPARK_HOME` and `HADOOP_CONF_DIR` in spark's interpreter setting, they are be passed to Spark interpreter process as environment variable which is used by Spark.

---



# Dynamic Form

- Using form Templates
  - Text input form
  - Select form
  - Checkbox form
- Creates Programmatically
  - Text input form
  - Text input form with default value
  - Select 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, SparkSql language backend uses it.

## Text input form

To create text input form, use `${formName}` templates.

for example

%md Hello \${name=sun}

FINISHED ▶ ⚙

name

Hello moon

Also you can provide default value, using `${formName=defaultValue}`.

%md Hello \${name=sun}

FINISHED ▶ ⚙

name

Hello sun

## Select form

To create select form, use `${formName=defaultValue,option1|option2...}`

for example

`%md This is ${day=mon,mon|tue|wed|thu|fri|sat|sun}` FINISHED ▶ ⚙

**day**

mon

This is mon

Also you can separate option's display name and value, using

`${formName=defaultValue,option1(DisplayName)|option2(DisplayName)...}`

`%md This is ${day=mon,1(mon)|2(tue)|3(wed)|4(thu)|5(fri)|6(sat)|7(sun)}` FINISHED ▶ ⚙

**day**

mon

This is 1

## Checkbox form

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:

```
%md select ${checkbox:fields=name|age,name|age|salary|gender} from employees
```

FINISHED ▶ ⌵ ⌶ ⚙

**fields**

☒ name ☒ age ☐ salary ☐ gender

select name,age from employees

Took 0 seconds (outdated)

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

```
%md ${checkbox( and ):fruit=apple,apple|banana|orange}
```

FINISHED ▶ ⌵ ⌶ ⚙

**fruit**

☒ apple ☒ banana ☒ orange

apple and banana and orange

Took 0 seconds

# Creates Programmatically

Some language backend uses programmatic way to create form. For example [ZeppelinContext](#) provides form creation API

Here're some examples.

## Text input form

Scala

Python

```
%spark
println("Hello "+z.input("name"))
```

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

FINISHED ▶ ⚙

name

Hello moon

## Text input form with default value

Scala

Python

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

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

FINISHED ► ⚙

name

Hello sun

## Select form

Scala

Python

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

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

day

mon

Hello 1

## Checkbox form

Scala

Python

```
%spark
val options = Seq(("apple","Apple"), ("banana","Banana"), ("orange","Orange"))
println("Hello "+z.checkbox("fruit", options).mkString(" and "))
```

%spark

FINISHED ▶ ⌵ 📖 ⚙️

```
val options = Seq(("apple","Apple"), ("banana","Banana"), ("orange","Orange"))
println("Hello "+z.checkbox("fruit", options).mkString(" and "))
```

fruit

☒ Apple ☒ Banana ☒ Orange

```
options: Seq[(String, String)] = List((apple,Apple), (banana,Banana), (orange,Orange))
```

Hello apple and banana and orange

Took 2 seconds



## Checkbox form

Scala

Python

```
%pyspark
options = [("apple","Apple"), ("banana","Banana"), ("orange","Orange")]
print("Hello "+ " and ".join(z.checkbox("fruit", options, ["apple"])))
```

%spark

FINISHED ▶ ⌂ 📖 ⚙

```
val options = Seq(("apple","Apple"), ("banana","Banana"), ("orange","Orange"))
println("Hello "+z.checkbox("fruit", options).mkString(" and "))
```

fruit

☒ Apple ☒ Banana ☒ Orange

```
options: Seq[(String, String)] = List((apple,Apple), (banana,Banana), (orange,Orange))
```

Hello apple and banana and orange

Took 2 seconds

# Customize Apache Zeppelin homepage

- How to set a notebook as your Zeppelin homepage
  - Create a notebook using Zeppelin
  - Set the notebook id in the config file
  - Restart Zeppelin
- Show notebooks list in your custom homepage

Apache Zeppelin allows you to use one of the notebooks you create as your Zeppelin Homepage. With that you can brand your Zeppelin installation, adjust the instruction to your users needs and even translate to other languages.

## How to set a notebook as your Zeppelin homepage

The process for creating your homepage is very simple as shown below:

1. Create a notebook using Zeppelin
2. Set the notebook id in the config file
3. Restart Zeppelin

## Create a notebook using Zeppelin

Create a new notebook using Zeppelin, you can use `%md` interpreter for markdown content or any other interpreter you like. You can also use the display system to generate text, html, table or Angular (backend API, frontend API).

Run (shift+Enter) the notebook and see the output. Optionally, change the notebook view to report to hide the code sections.

## Set the notebook id in the config file

To set the notebook id in the config file, you should copy it from the last word in the notebook url. For example,

Set the notebook id to the `ZEPPELIN_NOTEBOOK_HOMESCREEN` environment variable or `zeppelin.notebook.homescreen` property.

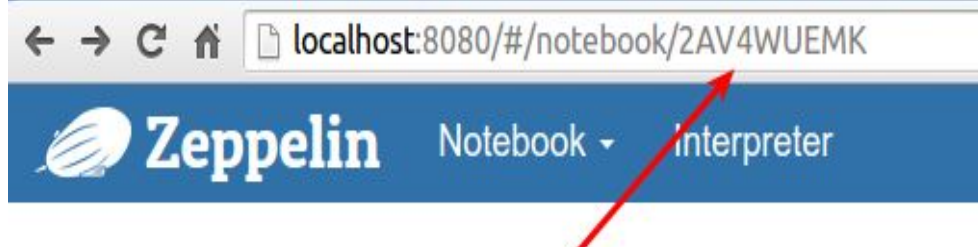
You can also set the `ZEPPELIN_NOTEBOOK_HOMESCREEN_HIDE` environment variable or `zeppelin.notebook.homescreen.hide` property to hide the new notebook from the notebook list.

## Restart Zeppelin

Restart your Zeppelin server

```
./bin/zeppelin-daemon stop  
./bin/zeppelin-daemon start
```

That's it! Open your browser and navigate to Apache Zeppelin and see your customized homepage.



# Show notebooks list in your custom homepage

If you want to display the list of notebooks on your custom Apache Zeppelin homepage all you need to do is use our %angular support.

Add the following code to a paragraph in you home page and run it... walla! you have your notebooks list.

```
println(  
  """"%angular  
    <div class="col-md-4" ng-controller="HomeCtrl as home ">  
      <h4>Notebooks</h4>  
      <div>  
        <h5><a href="" data-toggle="modal" data-target="#noteNameModal" style="text-decoration:  
none; ">  
          <i style="font-size: 15px;" class="icon-notebook"></i> Create new note</a></h5>  
          <ul style="list-style-type: none; ">  
            <li ng-repeat="note in home.notes.list track by $index "><i style="font-size: 10px;"  
class="icon-doc"></i>  
              <a style="text-decoration: none;" href="#/notebook/"></a>  
            </li>  
          </ul>  
        </div>  
      </div>  
    """)
```

# Homepage



```
println("""%angular
<div class="col-md-4" ng-controller="HomeCtrl as home">
  <h4>Notebooks</h4>

  <div>
    <h5><a href="" data-toggle="modal" data-target="#noteNameModal" style="text-decoration: none;">
      <i style="font-size: 15px;" class="icon-notebook"></i> Create new note</a></h5>
    <ul style="list-style-type: none;">
      <li ng-repeat="note in home.notes.list track by $index"><i style="font-size: 10px;" class="icon-doc"></i>
        <a style="text-decoration: none;" href="#/notebook/{{note.id}}">{{note.name || 'Note ' + note.id}}</a>
      </li>
    </ul>
  </div>
</div>
""")
```

Took 0 seconds

## Notebooks

 Create new note

-  Homepage
-  Note YG2RMG98W
-  Zeppelin Tutorial
-  python

The main trick here relays in linking the `<div>` to the controller:

```
<div class="col-md-4" ng-controller="HomeCtrl as home">
```

Once we have `home` as our controller variable in our `<div></div>` we can use `home.notes.list` to get access to the notebook list.

- Why did the **Apache Zeppelin** refuse to attend the data science party? **Because** it heard there would be too many **'NaN's** and it didn't want to be left feeling 'null' and void of fun!



# Installation

## Requirements

- Minimum of 4GB RAM
- At least 10GB of free disk space
- Java Development Kit (JDK) 1.8 or higher
- Apache Maven (for building from source)
- Git (if cloning from the repository)

Name	Value
OpenJDK or Oracle JDK	1.8 (151+) (set <code>JAVA_HOME</code> )
OS	Mac OSX  Ubuntu 18.04  Ubuntu 20.04

---

## Apache Zeppelin Configuration

**You can configure Apache Zeppelin with both environment variables in `conf/zeppelin-env.sh` (`conf\zeppelin-env.cmd` for Windows) and Java properties in `conf/zeppelin-site.xml`. If both are defined, then the environment variables will take priority.**

---



---

## Downloading Binary Package

Two binary packages are available .Only difference between these two binaries is whether all the interpreters are included in the package file.

- **all interpreter package**: unpack it in a directory of your choice and you're ready to go.
- **net-install interpreter package**: only spark, python, markdown and shell interpreter included. Unpack and follow install additional interpreters to install other interpreters. If you're unsure, just run `./bin/install-interpreter.sh --all` and install all interpreters.

## Building Zeppelin from source

Follow the instructions How to Build, If you want to build from source instead of using binary package.

---

# Starting Apache Zeppelin

## Starting Apache Zeppelin from the Command Line

On all unix like platforms:

```
bin/zeppelin-daemon.sh start
```

After Zeppelin has started successfully, go to <http://localhost:8080> with your web browser.

By default Zeppelin is listening at `127.0.0.1:8080`, so you can't access it when it is deployed on another remote machine. To access a remote Zeppelin, you need to change

```
zeppelin.server.addr to 0.0.0.0 in conf/zeppelin-site.xml
```

Check log file at `ZEPPELIN_HOME/logs/zeppelin-server-*.log` if you can not open Zeppelin.

## Stopping Zeppelin

```
bin/zeppelin-daemon.sh stop
```

## Using the official docker image

Make sure that `docker` is installed in your local machine.

Use this command to launch Apache Zeppelin in a container.

```
docker run -p 8080:8080 --rm --name zeppelin apache/zeppelin:0.11.1
```

To persist `logs` and `notebook` directories, use the `volume` option for docker container.

```
docker run -u $(id -u) -p 8080:8080 --rm -v $PWD/logs:/logs -v $PWD/notebook:/notebook \
-e ZEPPELIN_LOG_DIR='/logs' -e ZEPPELIN_NOTEBOOK_DIR='/notebook' \
--name zeppelin apache/zeppelin:0.11.1
```

`-u $(id -u)` is to make sure you have the permission to write logs and notebooks.

For many interpreters, they require other dependencies, e.g. Spark interpreter requires Spark binary distribution and Flink interpreter requires Flink binary distribution. You can also mount them via docker volume. e.g.

```
docker run -u $(id -u) -p 8080:8080 --rm -v /mnt/disk1/notebook:/notebook \
-v /usr/lib/spark-current:/opt/spark -v /mnt/disk1/flink-1.12.2:/opt/flink -e FLINK_HOME=/opt/flink \
-e SPARK_HOME=/opt/spark -e ZEPPELIN_NOTEBOOK_DIR='/notebook' --name zeppelin apache/zeppelin:0.11.1
```

# Start Apache Zeppelin with a service manager

Apache Zeppelin can be auto-started as a service with an init script, using a service manager like **upstart**.

This is an example upstart script saved as `/etc/init/zeppelin.conf` This allows the service to be managed with commands such as

```
sudo service zeppelin start
sudo service zeppelin stop
sudo service zeppelin restart
```

Other service managers could use a similar approach with the `upstart` argument passed to the `zeppelin-daemon.sh` script.

```
bin/zeppelin-daemon.sh upstart
```

# zeppelin.conf

```
description "zeppelin"
start on (local-filesystems and net-device-up IFACE!=lo)
stop on shutdown
# Respawn the process on unexpected termination
respawn
# respawn the job up to 7 times within a 5 second period.
# If the job exceeds these values, it will be stopped and marked as
failed.
respawn limit 7 5
# zeppelin was installed in /usr/share/zeppelin in this example
chdir /usr/share/zeppelin

exec bin/zeppelin-daemon.sh upstart
```

# Next Steps

Congratulations, you have installed Apache Zeppelin! Here are a few steps you might find useful:

## **New to Apache Zeppelin...**

- For an in-depth overview, head to Explore Zeppelin UI.
- And then, try run Tutorial Notebooks shipped with your Zeppelin distribution.
- And see how to change configurations like port number, etc.

## **Spark, Flink, SQL, Python, R and more**

- Spark support in Zeppelin, to know more about deep integration with Apache Spark.
- Flink support in Zeppelin, to know more about deep integration with Apache Flink.
- SQL support in Zeppelin for SQL support
- Python support in Zeppelin, for Matplotlib, Pandas, Conda/Docker integration.
- R support in Zeppelin
- All Available Interpreters

# Zeppelin Tutorial

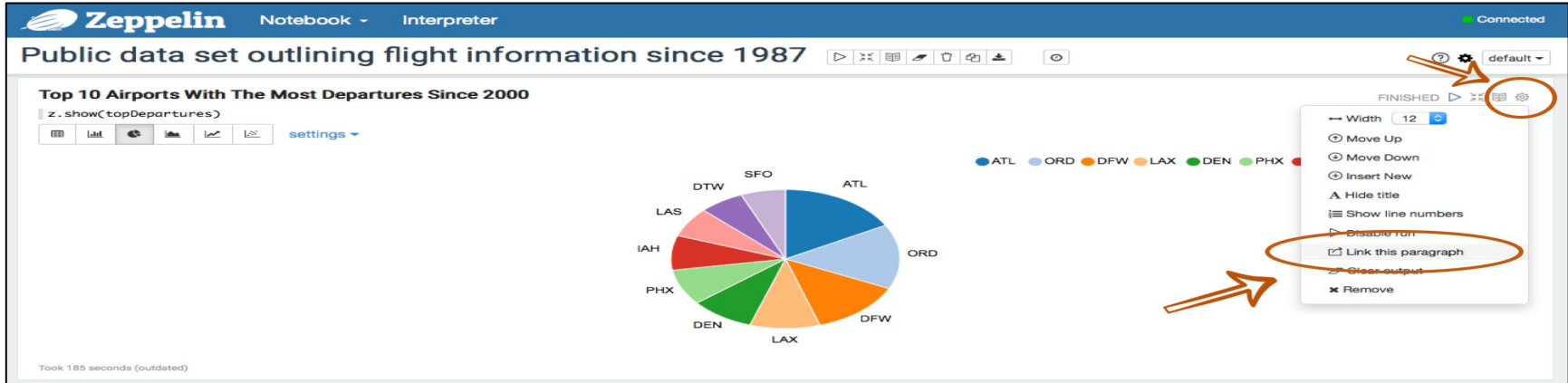
- Tutorial with Local File
  - Data Refine
  - Data Retrieval
- Tutorial with Streaming Data
  - Data Refine
  - Data Retrieval

Current main backend processing engine of Zeppelin is Apache Spark. If you're new to this system, you might want to start by getting an idea of how it processes data to get the most out of Zeppelin.

# How can you publish your paragraph ?

- Copy a Paragraph Link
- Embed the Paragraph to Your Website

Apache Zeppelin provides a feature for publishing your notebook paragraph results. Using this feature, you can show Zeppelin notebook paragraph results in your own website. It's very straightforward. Just use `<iframe>` tag in your page.



The screenshot displays the Apache Zeppelin web interface. At the top, the header shows 'Zeppelin', 'Notebook', and 'Interpreter' tabs, along with a 'Connected' status indicator. The main content area is titled 'Public data set outlining flight information since 1987'. Below this, a paragraph titled 'Top 10 Airports With The Most Departures Since 2000' is shown. The paragraph contains a pie chart and a legend for the following airports: ATL, ORD, DFW, LAX, DEN, PHX, SFO, DTW, LAS, and IAH. A context menu is open over the paragraph, with the 'Link this paragraph' option highlighted. The menu also includes options for 'Width', 'Move Up', 'Move Down', 'Insert New', 'Hide title', 'Show line numbers', 'Disable run', 'Clear output', and 'Remove'. An arrow points to the 'Link this paragraph' option, and another arrow points to the 'default' dropdown menu in the top right corner.

Top 10 Airports With The Most Departures Since 2000

z.show(topDepartures)

ATL, ORD, DFW, LAX, DEN, PHX, SFO, DTW, LAS, IAH

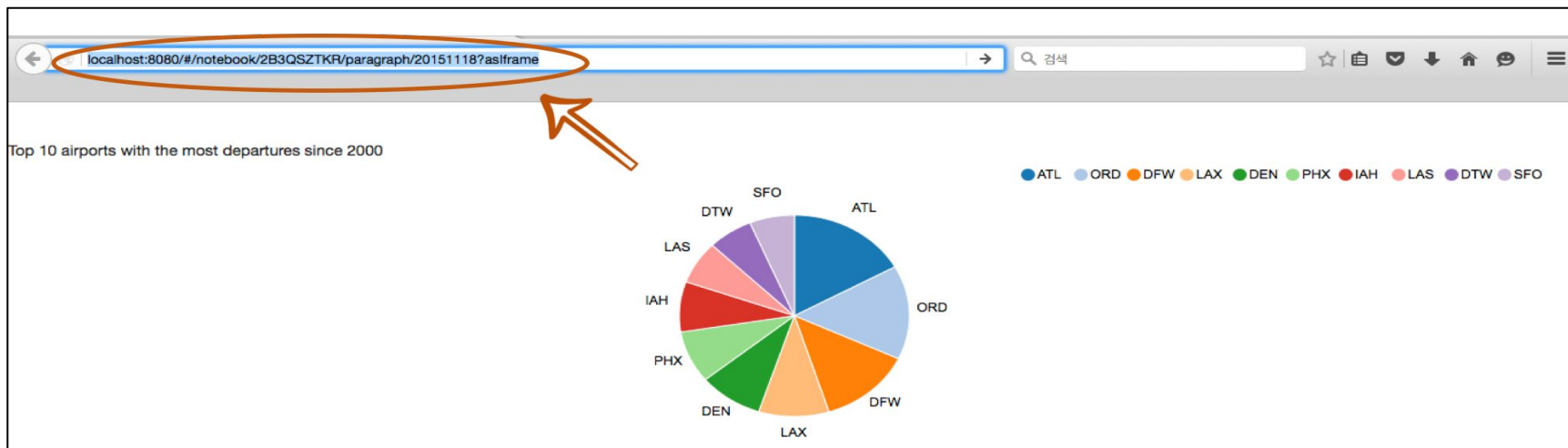
Link this paragraph



# Copy a Paragraph Link

A first step to publish your paragraph result is **Copy a Paragraph Link**.

- After running a paragraph in your Zeppelin notebook, click a gear button located on the right side. Then, click **Link this Paragraph** menu like below image.



# Embed the Paragraph to Your Website

For publishing the copied paragraph, you may use `<iframe>` tag in your website page. For example,

```
<iframe src="http://< ip-address >:< port  
>/#/notebook/2B3QSZTKR/paragraph/...?asIframe" height="" width="" ></iframe>
```

Finally, you can show off your beautiful visualization results in your website.

## Journey to Zeppelin [Home](#)

### Analytics with Spark - Public data set outlining flight information since 1987

25 Jan 2016

This quick example will explain about flight delays and cancellations for domestic flights in the US.

The Department of Transportation has a public data set outlining flight information since 1987. I downloaded it, converted the file format from CSV to the columnar Parquet format (for better performance), and uploaded it to a public, read-only S3 bucket (`s3://us-east-1.elasticmapreduce.samples/flightdata/input`). The data set is around 4 GB compressed (79 GB uncompressed) and contains 162,212,419 rows, so it makes sense to use a distributed framework like Spark for querying. Specifically, I would like to know the 10 airports with the most departures, the most flight delays over 15 minutes, the most flight delays over 60 minutes, the and most flight cancellations. I also want to know the number of flight cancellations by yearly quarter, and the 10 most popular flight routes.

Top 10 airports with the most departures since 2000



Top 10 airports with the most departure delays over 15 minutes since 2000



---

# Use case

- Data Exploration and Visualization:
    - Interactive exploration of datasets
    - Visualizing data trends and patterns
    - Creating insightful visualizations for analysis and presentation
  - Data Science and Machine Learning:
    - Prototyping and testing machine learning models
    - Performing data preprocessing and feature engineering
    - Training and evaluating machine learning algorithms
    - Tuning model parameters and experimenting with different algorithms
  - Big Data Processing:
    - Integrating with Spark for large-scale data processing
    - Analyzing and processing massive datasets efficiently
    - Leveraging distributed computing capabilities for faster processing
  - Collaborative Analytics:
    - Facilitating team-based data analysis projects
    - Sharing notebooks and insights with team members
    - Collaborating in real-time on data exploration and analysis
    - Tracking changes and contributions from multiple users
-

---

## Advantages

- Zeppelin is its versatility, supporting multiple programming languages through various interpreters. This flexibility allows users to write code in languages like Python, Scala, SQL, and more within the same environment, making it a powerful tool for diverse data analysis tasks.
  - Zeppelin boasts an interactive and user-friendly interface, offering an intuitive web-based platform for creating, sharing, and visualizing notebooks. This interface simplifies collaboration among data scientists, analysts, and other stakeholders.
  - Zeppelin seamlessly integrates with popular big data tools such as Apache Spark and Hadoop, enhancing its capability to process and analyze large-scale data efficiently.
-

---

## Limitations

- Zeppelin does have some limitations. One notable drawback is its performance issues when handling very large datasets, which can lead to slower response times and reduced efficiency. This can be a significant concern for organizations dealing with massive volumes of data.
  - there is a learning curve associated with mastering Zeppelin, especially for users who are new to big data tools. This learning curve can be a barrier to quick adoption and effective use.
  - Zeppelin's dependency on other big data tools for full functionality means that it cannot operate entirely independently. Users must integrate it with external frameworks and platforms to leverage its full potential, which can add complexity to its deployment and use.
-

---

## Conclusion

- Recap of key points:
    - Versatility with multiple programming languages
    - Interactive and user-friendly interface
    - Seamless integration with big data tools like Apache Spark
    - Useful for data exploration, visualization, and collaborative analytics
  - Importance of Apache Zeppelin in modern data analytics:
    - Facilitates comprehensive data analysis and visualization
    - Enhances collaboration among data scientists and analysts
    - Addresses diverse data processing needs efficiently
-

---

## Future Outlook:

- Potential developments:
    - Improvements in performance and scalability
    - Enhanced support for real-time data processing
    - Advanced machine learning integrations
  - Emerging trends in interactive data analysis:
    - Growing demand for powerful, collaborative data tools
    - Increasing importance of real-time analytics and machine learning
-

---

# Q&A

---



---

# Thanks

---

---

**END**

---