Integrate Apache Zeppelin with DSE (pre 6.0)

1. Overview

Apache Zeppelin is a web-based notebook that enables interactive data ingestion, exploration, and analytics. Through its "Interpreter" concept, it allows data-processing-backend systems and programming languages to be plugged in. Apache Cassandra, Apache Spark, Google BigQuery, Python, Elasticsearch, and etc. are some popular plugins that are supported by Apache Zeppelin out of the box. For a complete interpreter list, please check Zeppelin document page at here. You can also add your own interpreter by extending Zeppelin's Interpreter API.

DSE is an integrated data platform that covers DSE Core (Apache Cassandra), DSE Search (Apache Solr), and DSE Analytics (Apache Spark). Due to its native support for Cassandra and Spark integeration, Apache Zeppelin makes itself a good fit as an interactive front-end UI for DSE data exploration and analytics. It is also possible to extend its capability to serve DSE Graph (Apache Tinkerpop + Gremlin) analysis through 3rd interpreter like: https://github.com/shikeio/gremzeppelin.

In this blog post, I'll explore some examples of using Apache Zeppelin to explore DSE data through CQL, Solr query, and Spark RDD/DataFrame. I may add examples for Graph exploration in later posts.

1.1. DataStax Studio and Apache Zeppelin

Since DSE version 5.0, DataStax has also offered a notebook tool, called DataStax Studio, that provides similar web-based, interactive data exploration and analysis functionality. The initial functionality (version 1.0) is only limited to support DSE graph (Gremlin). Since DSE version 5.1, DataStax studio version 2.0 added support for DSE Core (CQL); but the support for DSE Analytics (Spark) is still missing. With the next major release DSE 6.0 and DataStax studio 6.0, the support for all DSE product components will be available through DataStax Studio 6.0.

NOTE :

When DSE 6.0 and DataStax Studio 6.0 are available, it is highly recommended to use DataStax Studio 6.0, instead of Apache Zeppelin, to interact with DSE 6.0 due to its close integration with the product suite.

The exploration of integrating Apache Zeppelin with DSE in this blog post is mostly for people who want to explore Zeppelin's interactive data analysis capability with DSE pre 6.0 version, especially for the case when DSE Analytics (Spark) is needed.

2. Testing Environment and Scenario Setup

2.1. DSE Environment Setup

DSE Cluster

- Version 5.1.7
- Singled-DC, 3-node
- All workloads enabled: Search, Analytics , and Graph

Apache Zeppelin

- Download latest Apache zeppelin release (version 0.7.3 as of writing) binary package with all interpreters:https://zeppelin.apache.org/download.html
- Unzip the binary package on one of the DSE nodes.

Note: Technically this can be put on any host as long as the host is able to communicate with DSE cluster successfully (as a DSE client). Also, in order for Spark interpreter to work properly with DSE, the host should have DSE binary available.

2.2. Example Scenario Setup

The example scenario that I use for my testing is DataStax's Wikipedia demo for DSE SearchAnalytics. The steps to set up this example scenario can be found at:

https://docs.datastax.com/en/dse/5.1/dse-admin/datastax_enterprise/spark/dseSearchAnalyticsWikipediaDemo.html

We'll use Apache Zeppelin to explore the example scenario data via CQL (both standard Cassandra query and Solr query) and Spark processing (through DataStax's Cassandra-Spark connector).

3. Configure Apache Zeppelin Integration with DSE

3.1. Start Apache Zeppelin server in "DSE way"

The regular way of starting Zeppelin server is by exeucing the following command-line script.

> *<ZEPPELIN_HOME>/bin/zeppelin-daemon.sh start*

Starting this way has some issues when connecting to the DSE Analytics/Spark DC. If you try to run some Spark code from Zeppelin this way, most likely you'll see an error message in DSE system log like "Unknown application type when trying to register Zeppelin". This error (for most cases) means that the Spark client job (Zeppelin in this case) isn't submitted with proper DSE libraries.

The proper way of launching Zeppelin to connect to DSE is through "dse exec" utility, as described in DataStax document:

> https://docs.datastax.com/en/dse/5.1/dse-admin/datastax_enterprise/spark/thirdPartyToolsIntegrationSpark.html?hl=dse%2Cexec

Below is how I did:

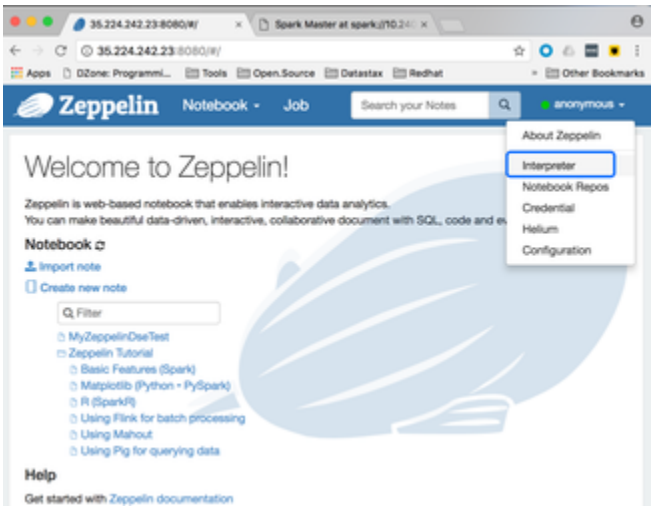
- Create a wrapper shell script (e.g. startZeppelin.sh):

> *$ cat startZeppelin.sh*
> *<ZEPPELIN_HOME>/bin/zeppelin-daemon.sh start*

- Launch Zeppelin server with proper DSE libraries

> *dse exec ./startZeppelin.sh*

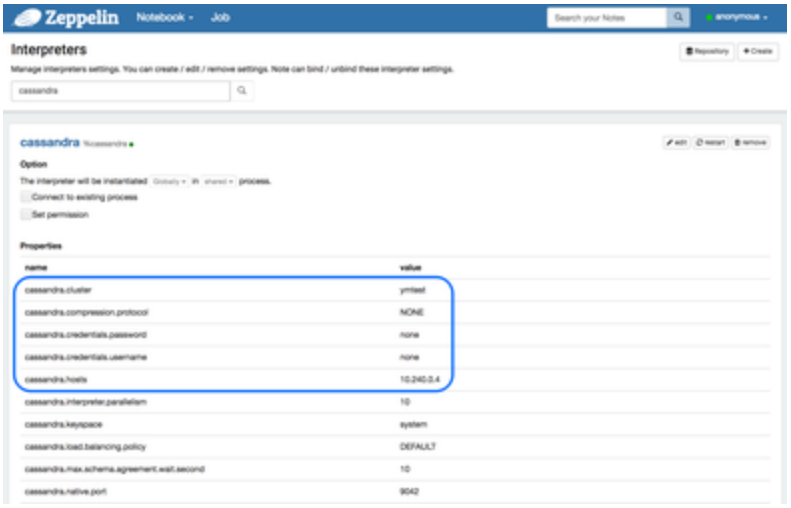
Once started successfully, you can access Zeppelin web portal at port 8080 on the host where Zeppelin server is installed, as below:



3.2. Configure Zeppelin Cassandra Interpreter

From the main port, click "anonymous Interpreter" and then search for "cassandra" to open Zeppelin Cassandra Interpreter configuration page, as below.

There is a long list of items that can be configured. The core one is to specify the Cassandra connection host, username/password (if authentication is enabled), etc. For a DSE cluster, this can be the hostname of IP of any DSE node.

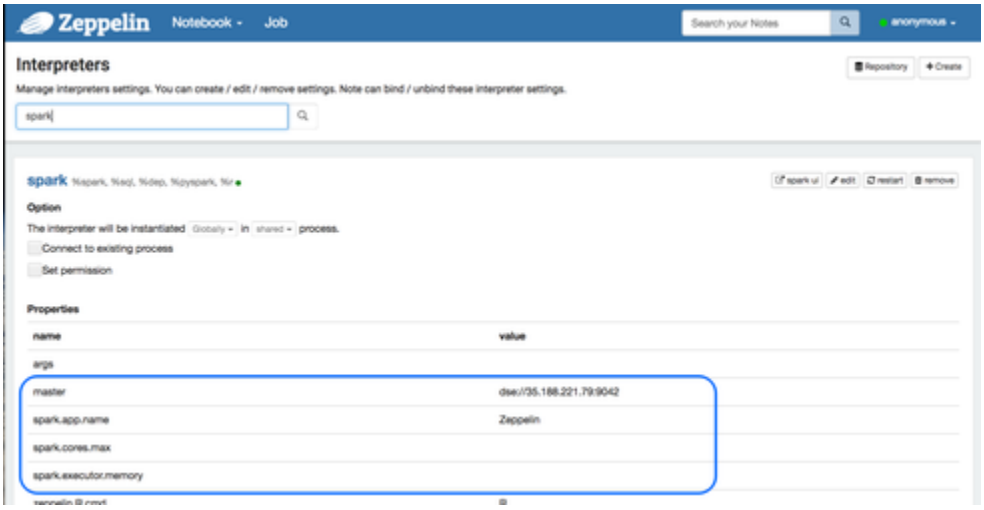


3.3. Configure Zeppelin Spark Interpreter

Similarly, bring up Zeppelin Spark Interpreter configuration page. The core spark configuration items are the following ones.

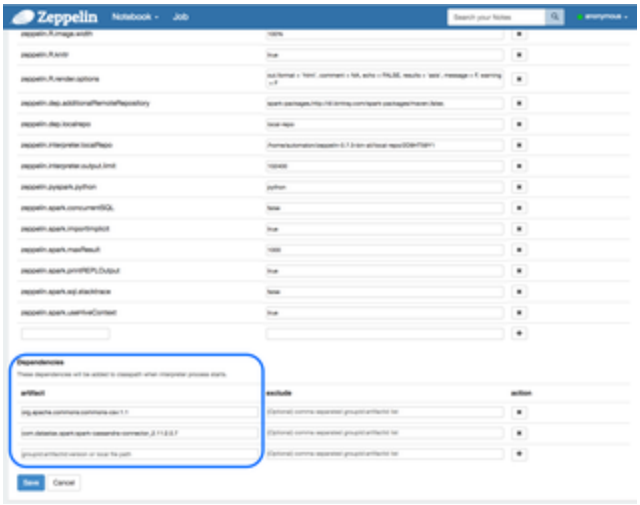
- master
- spark.cores.max
- spark.executor.memory

Unlike an OSS Spark cluster, the spark master-addres has to be specified as the format of "dse://<dse_master-node_address>:<cassandra_conection_port>". An example is in the screenshot below as "dse://35.188.221.79:9042". A simple way to get this address is from DSE command-line utility: **dse client-tool spark master-address**



In order to use DataStax's Cassandra-Spark Connector, you also need to add the dependencies to it. In the example below, I added 2 dependencies, one for CSV processing and one for Cassandra-Spark connector:

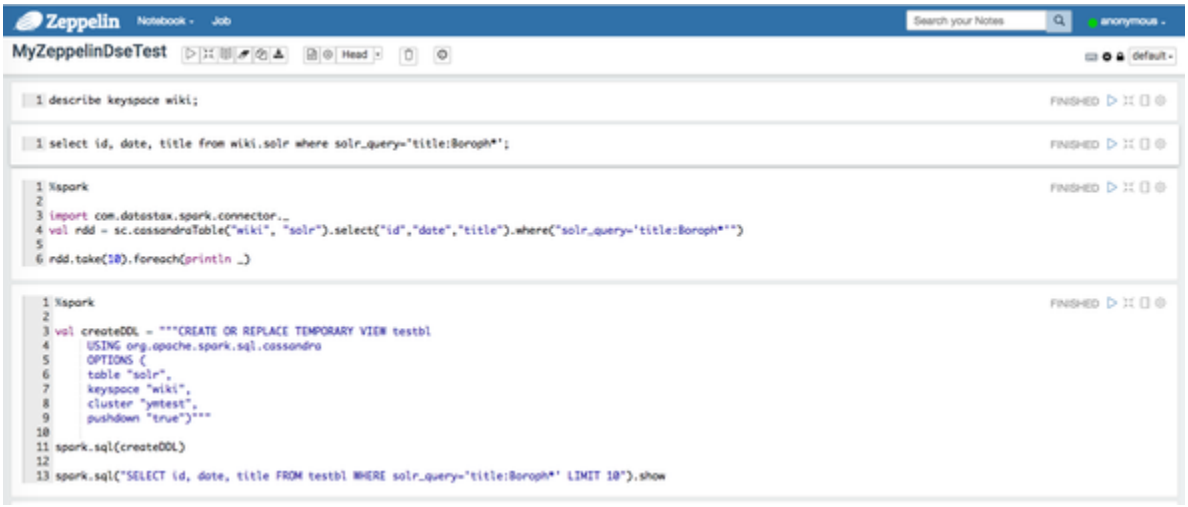
- org.apache.commons:commons-csv:1.1
- com.datastax.spark:spark-cassandra-connector_2.11:2.0.7 (the right version format is: *com.datastax.spark:spark-cassandra-connector_<scala_version>:<connector_version>*)



4. Test DSE Example Scenario Through Zeppelin

After configuring the Cassandra and Spark interpreters and restarting Zeppelin server, we're ready to test Wikipedia example scenario through Zeppelin notebook. The screenshot below is an example notebook created with Cassandra as the default interpreter. It has 4 sessions that does the following things correpsondingly. The note in "()" at the end marks the DSE component(s) that each session touches.

- A regular CQL statement to describe the example Cassandra keyspace "wiki" (note: DSE Core).
- A DSE Solr query to get documents that satisfy certain conditions (note: DSE Search).
- A Spark RDD based code snippet to get data from Cassandra via Cassandra-Spark connector (note: DSE Search and Analytics)
- A Spark SQL and DataFrame based code snippet to get data from Cassandra via Cassandra-Spark connector (note: DSE Search and Analytics)



Note that save the space for the blog post, I don't put all the session results here, I put the result of last session (Spark SQL + Spark Data Frame + Solr Query) for demonstration purpose:

```
createDDL: String =
    CREATE OR REPLACE TEMPORARY VIEW testbl
    USING org.apache.spark.sql.cassandra
    OPTIONS (
    table "solr",
    keyspace "wiki",
    cluster "ymtest",
    pushdown "true")
res5: org.apache.spark.sql.DataFrame = []
+--------+--------------------+--------------------+
| id| date| title|
+--------+--------------------+--------------------+
|23730195|13-APR-2011 15:37...| Borophagus dudleyi|
```

```
|23731282|13-APR-2011 15:34...| Borophagus orc|
|23732450|13-APR-2011 15:33...| Borophagus secundus|
|23729958|13-APR-2011 15:38...| Borophagus parvus|
|23730974|07-AUG-2011 18:21...|Borophagus littor...|
|23730528|13-APR-2011 15:36...| Borophagus hilli|
|23731616|13-APR-2011 15:34...| Borophagus pugnator|
|23730810|13-APR-2011 15:36...|Borophagus divers...|
+--------+--------------------+--------------------+
```

Meanwhile, you can verify the Spark application (Zeppelin in this case) running status from Spark Master Web UI.