Reflections on Apache Drill

Abhishek Tiwari

Citation: *A. Tiwari*, "Reflections on Apache Drill", Abhishek Tiwari, 2015.
doi:10.59350/q22bq-ekh76

Published on: September 29, 2015

I have been playing with Apache Drill for quite some time now. In layman's terms, Apache Drill is SQL query engine which can perform queries against any type of data store - in particular any non-relational data store. Apache Drill's ability to query against raw data files stored on the cloud storage platforms such as Amazon S3, Azure Blob Storage, Google Cloud Storage makes it a really attractive and cost effective query engine. Using Apache Drill one can get faster insights without requiring up-front schema knowledge or without loading the data in the relational data store.

Self-describing data formats

Apache Drill is particularly useful for the self-describing data formats such as JSON and Parquet. With self-describing data formats, data itself implies its schema or structure. It is important to note self-describing data structure can constantly evolve such as dynamic or nested JSON. So one can frequently add and remove fields to data set without affecting Drill's ability to query against the data.

Apache Parquet

A single query in Drill can join data from multiple data stores. For example, you can join a user profile data in MongoDB with a directory of web activity logs stored on Amazon S3. Then you can write joined data as Parquet file to an Amazon S3 or a local file system.

Parquet is a self-describing columnar data format. Parquet helps Apache Drill to optimize query performance and minimize I/O by enabling the column storage, data compression, data encoding and data distribution (related values in close proximity). When a read of Parquet data occurs, Drill loads only the necessary columns of data, which reduces I/O. To perform efficient querying on a large number of files, latest version of Drill can take advantage of the metadata cache feature in for the query planning. Apache Drill will generate and save a metadata cache in each directory in nested directories.

Let's take a test ride

We are going to perform SQL queries on data in Amazon S3 storage using Apache Drill.

- First download the latest version of Apache Drill and install it. You can install Drill in either embedded mode (standalone instance) or distributed mode (clustered Hadoop environment). For now we are going to install Drill in standalone mode as described here. You can start Drill in embedded mode by navigating to the Drill `$DRILL_HOME/bin` directory and then issuing `drill-embedded` command to start the Drill shell.
- Next install the JetS3t which is a free, open-source Java toolkit to interact with Amazon S3,

```
wget http://bitbucket.org/jmurty/jets3t/downloads/jets3t-0.9.3.zip
unizp jets3t-0.9.3.zip
cp jets3t-0.9.2/jars/jets3t-0.9.3.jar $DRILL_HOME/jars/3rdparty
```

Let's enable the JetS3t plugin by editing the `$DRILL_HOME/bin/hadoop-excludes.txt` file and removing the line `jets3t`. Please also remove `parquet` from this file as we are going to use Parquet module.

- Now add a `core-site.xml` file in the `$DRILL_HOME/conf/` with the following configuration. Please replace placeholder values in the configuration file with appropriate AWS API keys. This configuration is required by JetS3t to interact with the Amazon S3 buckets to read and write the stored files.

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
<property>
  <name>fs.s3.awsAccessKeyId</name>
  <value>ACCESS KEY</value>
</property>

<property>
  <name>fs.s3.awsSecretAccessKey</name>
  <value>SECRET KEY</value>
</property>

<property>
  <name>fs.s3n.awsAccessKeyId</name>
  <value>ACCESS KEY</value>
</property>

<property>
  <name>fs.s3n.awsSecretAccessKey</name>
  <value>SECRET KEY</value>
</property>
</configuration>
```

- For Amazon S3 bucket, you should add a storage plugin configuration by visiting the Apache Drill admin interface on `http://localhost:8047/storage`. I am using the following configuration for my S3 setup. If you are using below configuration, please ensure that you update the placeholder field `bucket-name` along with workspaces `root` and `out`. Please note that workspace `out` is the folder in the bucket where I would write the results because `root` is not writable as described in the configuration.

```
{
  "type": "file",
  "enabled": true,
```

```
  "connection": "s3n://<bucket-name>",
  "workspaces": {
    "root": {
      "location": "/",
      "writable": false,
      "defaultInputFormat": null
    },
    "out": {
      "location": "/out",
      "writable": true,
      "defaultInputFormat": null
    }
  },
  "formats": {
    "psv": {
      "type": "text",
      "extensions": [
        "tbl"
      ],
      "skipFirstLine": true,
      "delimiter": "|"
    },
    "csv": {
      "type": "text",
      "extensions": [
        "csv"
      ],
      "skipFirstLine": true,
      "delimiter": ","
    },
    "tsv": {
      "type": "text",
      "extensions": [
        "tsv"
      ],
      "skipFirstLine": true,
      "delimiter": "\t"
    },
    "tlsv": {
      "type": "text",
      "extensions": [
        "tilda"
      ],
      "skipFirstLine": true,
      "delimiter": "~"
    },
    "parquet": {
      "type": "parquet"
    },
    "json": {
      "type": "json"
```

```
    },
    "avro": {
      "type": "avro"
    }
  }
}
```

- Perform queries against files in your Amazon S3 bucket. Drill supports the ANSI standard for SQL.

```
SELECT * from `s3`.`customer.csv` limit 10;
```

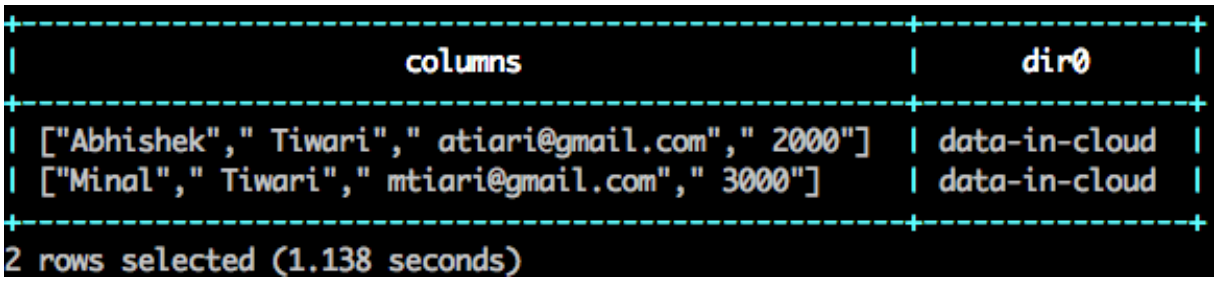


Figure 1: Apache Drill sample query-1

```
SELECT columns[0] as FirstName, columns[1] as LastName, columns[2] as
    Email  FROM `s3`.`customer.csv`;
```

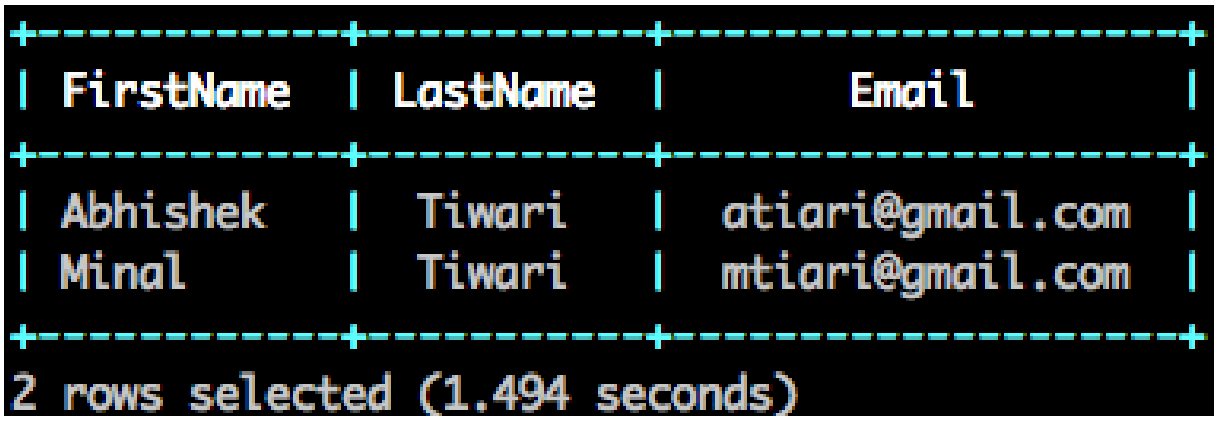


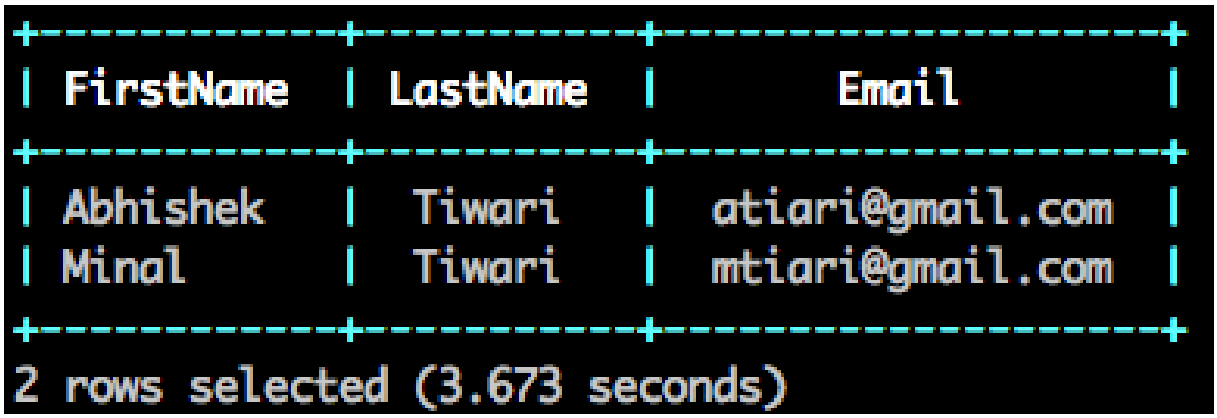
Figure 2: Apache Drill sample query-2

Now, write the data which you queried from Amazon S3 files as Parquet files back to your bucket but this time in a different folder `out`,

```
ALTER SESSION SET `store.format` = 'parquet';
```

```
CREATE TABLE `s3.out`.`customer_as_parquet` AS (SELECT columns[0] as
   FirstName, columns[1] as LastName, columns[2] as Email  FROM `s3`.`
   customer.csv`);

SELECT * FROM `s3.out`.`customer_as_parquet`;
```



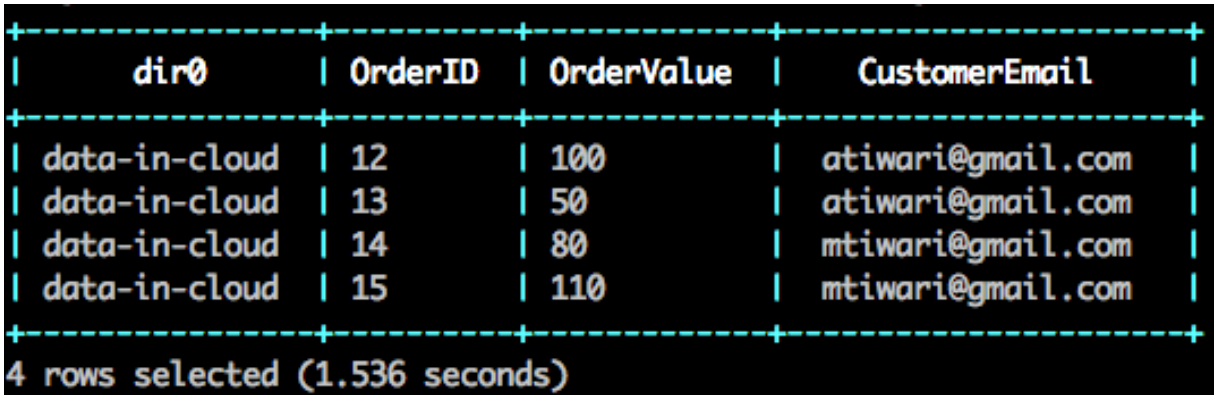
FirstName	LastName	Email
Abhishek	Tiwari	atiari@gmail.com
Minal	Tiwari	mtiari@gmail.com

2 rows selected (3.673 seconds)

Figure 3: Apache Drill sample query-3

Let's join the customer details with order data which is stored in JSON format.

```
SELECT * FROM `s3`.`order.json`;
```



dir0	OrderID	OrderValue	CustomerEmail
data-in-cloud	12	100	atiwari@gmail.com
data-in-cloud	13	50	atiwari@gmail.com
data-in-cloud	14	80	mtiwari@gmail.com
data-in-cloud	15	110	mtiwari@gmail.com

4 rows selected (1.536 seconds)

Figure 4: Apache Drill sample query-4

And here you have it - a Drill query running aginst two different data stores - one JSON another Parquet.

```
select customer.FirstName, customer.LastName, customer_order.OrderID,
   customer_order.OrderValue
FROM `s3.out`.`customer_as_parquet` as customer
```

```
INNER join `s3`.`order.json` as customer_order
on customer.Email=customer_order.CustomerEmail;
```

Conclusion

As you can see how easy it is to perform queries against the raw data files in the cloud storage buckets such as Amazon S3. Drill enabled us to treat our data like a table even when it's not.