AVD Group Off\_Feb\_2022:

𝐀𝐯𝐫𝐨 𝐒𝐜𝐡𝐞𝐦𝐚 𝐄𝐯𝐨𝐥𝐮𝐭𝐢𝐨𝐧

---------------------------

When you work with data, schema evolution plays a very crucial role.

For instance, while working on Kafka with Avro data you might have observed many exceptions that the producer/consumer clients face wrt the schema compatibility. It's important to understand the concept of schema evolution.

What is Avro?

~ In order to transfer data over network you need data to be serialized (data conversion to binary format) and Avro is one such system or tool that helps you achieve it.

~ Avro depends on schema and you can also think of it as a JSON format with the schema attached to it. And that's the main reason why Avro is preferred over json, because you can't enforce schema with json.

~ Avro is fast & compact.

~ Data is fully typed and named.

A simple Avro schema can look like -

{

"type": "record",

"name": "Employee",

"fields" : [

{"name":"emp\_name", "type":"string"},

{"name":"emp\_id", "type":"long"},

{"name":"department", "type":"string"}

}

Schema Evolution -

In order to evolve an avro schema you need to keep some important things in mind in order to make the changes compatible.

Backward Compatibility -

Your producer application is producing messages/data using an old schema and your consumer application can read the data using a newly evolved schema.

Let's use the above schema to understand it.

Suppose the Producer app created a record using the following schema -

{

"type": "record",

"name": "Employee",

"fields" : [

{"name":"emp\_name", "type":"string"},

{"name":"emp\_id", "type":"long"},

{"name":"department", "type":"string"}

}

(so the record has emp\_name, emp\_id and department)

Now, the Consumer app on the other side reads this record using a new evolved schema that doesn't contains the field department.

{

"type": "record",

"name": "Employee",

"fields" : [

{"name":"emp\_name", "type":"string"},

{"name":"emp\_id", "type":"long"}

}

But the consumer is still able to read the record and the data would just have emp\_name and emp\_id (department is silently ignored).

Forward Compatibility -

Your producer app uses a new schema to write messages and your consumer app can read the messages using an old schema.

~ There could exist a combination of both backward and forward compatible schema as well, a fully compatible schema.

Some of the rules that I personally found useful for creating compatible schemas -

- You can easily add a field with a default value in new schema.

Now suppose the producer is writing using an old schema and consumer uses this new schema, as we have a default value associated with the newly added field we don't need to worry about this field missing in the producer schema and the field would get default value on consumer side.

- You can easily remove a field having default value in new schema.

- You can't rename a field but you can use aliases.

- You can't change the data type.

Why Hadoop 1 is single point of failure ..Improvements in Hadoop 2

Hadoop 1:-

--------------

It contains MapReduce(For Processing) and HDFS(For Storage).

If Name Node Fails there is no access for metadata...No access to cluster.

Hadoop 2:

-------------

The improvement in Hadoop2 was YARN(Yet Another Resource Negotiator).

In Hadoop 2 two things help Downtime.

1.Meta Files -->FSImage(Snapshot of in memory filesystem at a given Time)

-->Edit Logs(All transactions happen after snapshot will store in here)

2.Secondary Name Node

Secondary Name Node Helps in Merging FSImage and Edit Logs to Create a new FSImage.

Difference between --split-by and --boundry-queries?

While doing the sqoop import data is splitted between mapper by using primary key.

sqoop will normally throw an error message, when there is no primary key in our table, In that case we can use --split-by to know the sqoop another column to be used to compute min and max to compute the split size.

During sqoop import, when number of mapper is greater than 1, A bounding vals query will run internally for creating the splits.

It select the min(), max() from table and divide it with total number of mapper( by default it's 4) to get the split size.

we can also customized the bounding vals query by using --bounding-query.

Hadoop – Different Modes of Operation

Hadoop Mainly works on 3 different Modes:

Standalone Mode

Pseudo-distributed Mode

Fully-Distributed Mode

1. Standalone Mode

In Standalone Mode none of the Daemon will run i.e. Namenode, Datanode, Secondary Name node, Job Tracker, and Task Tracker. Standalone Mode also means that we are installing Hadoop only in a single system. By default, Hadoop is made to run in this Standalone Mode or we can also call it as the Local mode. We mainly use Hadoop in this Mode for the Purpose of Learning, testing, and debugging.

Hadoop works very much Fastest in this mode among all of these 3 modes. As we all know HDFS (Hadoop distributed file system) is not utilized in this mode. When your Hadoop works in this mode there is no need to configure the files – hdfs-site.xml, mapred-site.xml, core-site.xml for Hadoop environment. In this Mode, all of your Processes will run on a single JVM (Java Virtual Machine) and this mode can only be used for small development purposes.

2. Pseudo Distributed Mode (Single Node Cluster)

In Pseudo-distributed Mode we also use only a single node, but the main thing is that the cluster is simulated, which means that all the processes inside the cluster will run independently to each other. All the daemons that are Namenode, Datanode, Secondary Name node, Resource Manager, Node Manager, etc. will be running as a separate process on separate JVM(Java Virtual Machine) or we can say run on different java processes that is why it is called a Pseudo-distributed.

One thing we should remember that as we are using only the single node set up so all the Master and Slave processes are handled by the single system. Namenode and Resource Manager are used as Master and Datanode and Node Manager is used as a slave. A secondary name node is also used as a Master. The purpose of the Secondary Name node is to just keep the hourly based backup of the Name node.

3. Fully Distributed Mode (Multi-Node Cluster)

This is the most important one in which multiple nodes are used few of them run the Master Daemon’s that are Namenode and Resource Manager and the rest of them run the Slave Daemon’s that are DataNode and Node Manager. Here Hadoop will run on the clusters of Machine or nodes. Here the data that is used is distributed across different nodes. This is actually the Production Mode of Hadoop let’s clarify or understand this Mode in a better way in Physical Terminology.

* Once you download the Hadoop in a tar file format or zip file format then you install it in your system and you run all the processes in a single system but here in the fully distributed mode we are extracting this tar or zip file to each of the nodes in the Hadoop cluster and then we are using a particular node for a particular process. Once you distribute the process among the nodes then you’ll define which nodes are working as a master or which one of them is working as a slave.