Khaled El Tannir, M: Hello! Good evening. Sam Wanis: Good evening. Thanos Michailopoulos: Hello! Cristal Cortez: Tiffany. Hello! Khaled El Tannir, M: How are you today. Isaac Abouganem Stephens: Hello! Good morning. Oh, good evening. Sorry! Khaled El Tannir, M: See? How are you today? Do you have any question, any news? Okay? So today we will start, let me let me switch this. Okay. okay? So today we'll start by just giving the case for your homework. So almost all of you did very good. It was somehow easy. But again you did a good job, so I will show you my case. My approach to this, to this homework. I will not post the the Zeppelin note. If you need to review it, just go back to the recording, and then you compare with your submission. Okay, so do you remember it was about me open. My. okay, so this was the instruction very simple. We have 2 2 2 Csv files, and this Csv file about inspection returns. It is a real data set. It is not auto generated. It was, I think, if I'm not wrong, was collected from Kaggle and so what we have to do is to to upload this 2 files to Hdfs, then load this 2 Csv file into 2 spark data sets and then just do very, very basic analysis explore and exploration 2 or 3 query using the data set Api and 3 queries using the SQL Api. So here, what I provided to you is only the name of the column in the Csv files, but again very simple to understand and to infer. So we can understand all this. The Restaurant Csv. Are almost all of them are strings. Maybe cuisine type is not a string could be integral or longer or numerical. Colon. And here in the restaurant inspection. Csv file. Yes. May we have the violation code? We have the grade, and this might be numerical column, but the I would say the important column is inspection date. I didn't ask you to convert or to use this column, because it is a little bit tricky to to start working with date. This will come later. You will be working and dealing with date later. But for this homework I didn't want to complexify for you this part. So now let's ex. Let me explain my my case, my my solutions. Let me open this. And this should appear. Yes. okay. So the 1st part, 1st step or 1st task very simple. We create a directory on Hdfs and we upload these files on this directory. As you are loading these 2 files individually from spark, we don't need to separate them into different directories. So here we can put both file in the same directory. If I had to load this with hive and create a schema or a table for each one of this file. I could not put this in the same directory because this file has different. The file have different schemas. So to create a directory. You're already familiar with this syntax. Hdfs, Dfs make dear hyphen P. To create from the parent all the Directory, if it is not created and sub directories in one command. Otherwise you need to create. Make dear, for example, homework, and then make dear homework h. 1, or whatever the name you give to the subdirectory. then we use the put command, so the put command, take 2 arguments. Some of you use it. Only one argument. but put command takes 2 arguments. The 1st one is the local source and the target. Where we want to put this on Hdfs. If you do not specify the target. the put command will upload the files to the current directory of the current user. The current user is Zeppelin. The current Directory is user Zeppelin. So if you don't specify the target. this will be uploaded to the current directory of the current user. So here I'm providing the location on the local file system which is on your training data restaurant and the name of the file and the target is homework or the name of the directory you you created for this homework. Then I asked you to import some spark libraries, so we can put this here or later. But we are running spark now. So say, I, I prefer to separate all this all different parts. For example. I could put this spark to the 1st cell. It doesn't matter. It's exactly the same result. But just I wanted to put it after I uploaded my file to Hdfs because I'm starting using spark from here, but it doesn't matter wherever you put this before running your spark code. So here I asked you to import this because I wanted just to you to point that I wanted you to use the sequel date and time. So it was not just because we didn't use it in in, in the data analysis or the analysis itself. We didn't use the date, because, as I said, I wanted to simplify for you this homework. So but here, what I wanted you to just to point, and at least use it one for to to declare in your case, class the timestamp, or the date for this inspection, date or other. It doesn't matter if you put it in string. But I just wanted to you to use this data type. So I was flexible. If you didn't do it. I didn't penalize you so much great point for this, but just pay attention for the next homework. So we need to use a spark data set. What this mean. This means, it is a typed data frame. So this means I need to create a case class to handle the object. otherwise it will be untyped. So here the type is restaurant and inspection. So I need to create 2 case classes. The 1st one is to hold the object restaurant, and the second one is to hold the inspection object. Okay. So now, here, just I enumerated the colon we are using. For the restaurant. I have id name, borrow whatever, and for inspection I have id restaurant, inspection, date, and all the other columns. So to create the case class. Nothing more simple. Just use case class, give it a name and enumerate your column, name or attribute name, because here we are talking about class and the data type. So Id might be integral. The name is a string borrow, a string type, string, and so on. The syntax is very simple. Column, name or attribute name Colon, the 2 point followed by the data type. Now for the inspection. The same. So we only are using for inspection date, the data type data type date. Okay? Now I, this is the type. So whenever you will load one row from your Csv file restaurant or inspection spark will parse this row and infer the the attribute and map. All this attribute to your case. Class. Attribute the columns from the Csv. And the case class column. It will be mapped by the compiler directly. You don't need to do this explicitly. but our Csv files has any restaurant and Csv. Have no header. So if you load the Csv file, as is, and you ask Spark to infer the schema, it will infer the schema, but it will give you Colons name, I would say, assign it explicitly, oh, sorry, implicitly. So. It will be Colon 0, Colon one underscore Colon 0 underscore Colon one, and so on. So if you do this later, you need to rename your colon, using Colon Rename so to rename your colon. So what we can do to simplify and make this more clear. You create this structure of your schema and provided to be, use it when you load your Csv file. So as we have 2 files. So we need to create 2 schema script type using the script type. So here I'm just enumerating my column names, id names, borrow building them, and so and so on and defining the data type. So here, this syntax mean. I want to create struct type. This is the data type accepted by the schema class to be inferred. So it will be inferred. So you create a new struct type and you add the columns. So column name colon data type and true true means it is nullable means it accept null. Okay? So just to say, we don't know if this column or this row has a new in this column. So put the true, and this will simplify a lot We do the same for the restaurant and the inspection. So I created my schema, my custom schema and my inspection. Schema. Okay, now, we need to load these files using spark. Nothing complicated. Here, spark. This is our entry point from the driver, and I just call the read function. So read function, the read function, take some options. So the 1st option header is false. We do not have any header here. If you put it true. this mean spark will remove the 1st line even it is not the header, so your result will be truncated, so it will not be correct. Okay, so there is no header, and I don't want that spark infer the schema because I am providing my own structure. So if you do, or if you put inference schema true, it will read the the column, and it will assign Colon 0 Colon one. Just this name, and later. You need to rename this column. So so here I just use the option to set. There is no header, and I don't want the schema to be inferred. So I provide my schema. I provide my schema for each of these input file. And I provide the format Csv and the location where this Csv is located on Hdfs. So here Spark is installed on your it's a run on runs on your, so this means by default. It will read from Hdfs you can put, though here you can provide the full scheme. Hdfs. The name of the sandbox, the port number so we can provide it. But it is, it is not needed because we are running spark in a distributed environment. and it is running over yarn. So it. The default location is Adfs. So here. All we need to provide is the file name and the path to read this file as we want to read this and to load this as a data set spark data set. So we need to provide the object type. So here it is given by the As and the name of your case class. So as restaurant. This is our case. Class is if you just scroll up a little bit. You see, my case class name is restaurant, for the second is inspection. So whenever you will execute this function, spark will load, will provide the schema and the, As restaurant will map all the column from your Csv. To your attribute in your case class, and it will create a new instance for you. Everything is given implicitly. You see, the code is really simplified. This is because Scala, now just for performance, we can add the cash. So this mean the We. We want to persist this data set into memory until we finish our work from the from this homework, so it will be loaded in memory, so it will be a little bit faster. Same for the second file read exactly the same. The only thing you need to modify here is the name of the of the file input file and the type. So read no, header, I don't want to infer the schema and use my own schema. I provided to you and map as case class name. So now we can just before doing all the task here asked by the in the instruction. Okay, so you already defined the case class and we use the We load the data set and we use the cache. It is optional, but we it will work, and no problem. Even the we have enough memory to load this. But just I wanted to use more and more function from and be more familiar with spar because you maybe you need it later. Okay, so here we can just display and show the count from from the that each data set. So here we can just printed as row. So restore the data set dot count. So to return the count, you can just use print. Ln, put some some sentences, and in terms of it is free. It is what I wanted to see here that you are using the count function from your data set spar data set. That's all. So okay. now, to show the 1st or 11st 5 or top 5 or 1st 11st row from the data set. Use the show command or show function. The show function will show you or list the number you specify 10 in this example. False means. I don't want to truncate means, because if you remove the false it will be truncated. So if you can see here and let me just check if someone is waiting. No. So what is this? Okay? So if I run this, as you can see. So here it is. The name Colon is truncated. you see. So if we put false, it is not truncated great. So the same for the inspection. So I'll just repeating the the same operation. Now the 1st say, query, report the top 10 cuisine type. This is nothing but a group by account and account. So select cuisine type. group by cuisine type and count order by is optional. So here this syntax is a little bit maybe little verify. You can use the call, which means use the colon, or you if you don't use it here, as you can see, I didn't put any prefix. I didn't use dollar sign. I didn't use call to to specify the column name. But here, in order by you need to specify if you are using disk this descending order. If you are using ascending order, you don't need to specify the column, it will be used as a string and desk operator cannot be applied to a string. So this is why here you need to specify the column. Otherwise you can put this like this. and this is exactly the same syntax exactly the same. So here, if you do count desk, this will result. Erase an error, because desk cannot be applied to a string, so this can be applied only to a column. So you can say, I want to use the column, count, or you can say, I want to use the colon count again. This is only to be specified when you use descending order, otherwise you don't need to do it. Just use the string column name as a string. Okay. The second report, the 10 violation codes is exactly the same query. We all. We change it only the column name. So it is a group by and count. But here, instead, using cuisine type, we are using violation code. Okay. okay, now report the 10 top 10 violation codes. So here you can just output the result as a table which is exactly the same. Again, it is exactly the same. exactly the same violation code group by. And if you report it as a graph or a chart. It is accepted, of course, here. The result of this query is in text mode, because I am using the select and show from spark. And here I am using the Zeppelin show, which will convert the text. return it by, or the data frame, return it by spark. The show command and convert it into a Zipline table. So here, if you write. you run this way, it is correct, and if you want to be able to show the chart in Zeppelin. You can write. Just use the Zipline show command, and run exactly your your your query. But instead, here the show here is not from spark. It is from Zeppelin. This is how you can just convert this text output to a table grid in Zeppe. Okay. now to run the SQL Api. So here we already covered this 3 or 4 query, basic exploration. Now for this basic analysis, using the spark. Sequel to to run the spark SQL. You need to create a virtual table or virtual view in memory. This will convert your or create in memory table, and you can run the SQL statement directly. So if you create a table. it is correct. You can create or replace temp table. create or replace temp view. The only difference. Tables that you create are readable and writable. If you create a view, this means it is only read only the temp means whenever we close the session it will be discarded from the memory to to free memory whenever we finish our work. So this is better because we don't need to persist this in memory. Sorry. So here for each of the input file or the spark data set we created, we can create this temp view in memory, and we give it a name. So this name you give you can use it as a table name in your SQL. Statement. If you give any name here, my table one and my table 2. It will work no problem. And when you run your SQL statement you need to use this table name you gave to be able to query using, SQL, okay, so if you want to check, you can just print the schemas. This is for the column just to remember what columns we have. We have Id, we have name, and all of this. So this is a kind of refresher print schema. This will print every all the column name and the type. Okay, now, to run the SQL. Using the SQL. Statement, I don't want to use Scala. I don't want to use data frame Api or data set. Api, I want to use pure SQL, so this was the objective of this task. So you can run this in 2 ways in Zeppelin. So the 1st you can say, I want to use spark SQL. Or just SQL. This will work also, because this is alias for this, but I do recommend to use always spark SQL. It is more readable for for us, so we know that we are running a SQL. Query from spark using spark. Okay? So using this interpreter, it will allows you to run or write directly, or SQL. Statement. So select id restaurant, join by whatever, so it will return directly the table in Zeppelin. Now you have different other ways, some alternative syntax we can use. So here this is the the the 1st query, select column name, and I just do a join to get the name and the borrow and the question type from the second table. So this is why, here we need to use a join. Okay? So alternative syntax, you can use spark. Okay? And you can say, spark, sequel and provide your query exactly the same. But here it is more readable. When you run as and using spot sequel, you need to put it as a single string. so we can put it on more than one line. But we need to to manage all of this, so it is more readable to to use the spark, SQL. Interpreter directly and just write your SQL. Statement. So so this is the 1st alternative, so it will return, as you can see the result in a text mode. If you want to see a table in Zeppelin, just use z show Z. Show, and exactly the same commands. Part SQL. Of course we remove this final show. Okay, so let me show you here, as you can see, it is this ZZ show, and it will return this table, then you can create chart if if it is needed, or for here I didn't specify anything. So we need to to set the current. If we need to create a chart. Okay? So the second query, find the top 5 cuisine type associated with the highest number of violation. Again, here in this 3 query. We have a join to get the name, or whatever the colon we need from this from the join it table. But this is extremely simple as you can see. Select cuisine type. Count. Join on the Id restaurant to get the cuisine cuisine type and order by violation code. Okay? The 3rd query, is. Here again we have the join, and we calculate the average based on the score. Okay of each borrow. So group by an average. So this no, this is not what's for you. No, no, this was not for you. Okay, so from here this is everything you can. Just when you get here, you can just stop your spark. spark context, if you want to free, the memory can say, Okay, Spark, let's see, stop. and this will stop your context. So here, as if you just notice I didn't create the context, even it is needed when you run a spark command. So when you run a spark, command you, you need to create the context. But this context is created for you by Zeppelin. If you recreate the context, this mean you will use more. You need to use more memory from here. Let me show you from here. So here you have the memory used by. Let me this little bit bigger. as you can see. So here I have the memory actually use it in this queue. And you can see this is Zeppelin are using. And this is for spark, because this is spark is running here. If you recreate or create a new context, so this queue will be overloaded, and you will. Maybe you will not be able to run anything because you have the memory reserved by or allocated by, the context created by spark and the one you created, and as spark is configured to run on the yarn queue. this queue will be overloaded, so it will not. Maybe you'll get a crash. It depends on how it will behave. I don't know, but for sure you will get maybe a crash. You'll get a crash, or the queue will be overloaded, and I think will run correctly. So this was everything about your 1st homework. So do you have any question about this before we jump to our topic today again, if you need to review it or to you, just go back to the recording and then compare with your with your submission. And here, why? I do not submit, because with the recording, you have my explanation, and I don't want you just to get the the case and just copy paste. I just want you to understand why I did it this way, and compare to your submission and see wherever you can improve, or you can change, or you can. You can learn how to do it. Thanos Michailopoulos: Quick, quick question. If you have a second. Khaled El Tannir, M: Yeah, of course. Thanos Michailopoulos: If working with a data frame or a data set, is there any difference in in setting up a temporary table? Or is it all the exact same. Khaled El Tannir, M: It's exactly the same. Thanos Michailopoulos: Okay. Khaled El Tannir, M: Because the SQL Api use the same Api, the SQL. Api is unifying the data frame and data set. Api, the only things to need you need to do is to create this temporary table of overview in memory. Thanos Michailopoulos: Great. Thank you. Khaled El Tannir, M: Look up any other question. okay, great. So let me switch this. And now we can. Okay, so today, we will continue our 3rd class in our okay, in our data analysis part. And today we'll be covering how to choose the best file format for now you learned how to upload your data to the distributed storage. So in our case it is Hdfs. You learned how to load it. Using spark, you learned how to partition it and create a schema using hive, doing some query interactively, using Trino. creating some dashboard by connecting superset to trino or or hive. We can also connect superset to spark sequel. But this need to enable the spark, spark, thrift server. But I disable it to the sandbox because it needs a lot of memory. So, and as we are in very limited environment and resources, I didn't want to use or waste resources using for just for to use the SQL. A thrift server, but it is doable. So now until now you only work it and we only work it with a Csv file or text file. So today we'll be covering a new binary format and explain what are the advantage. What are the cons, what are the pro using this format? And when to use each of these format, because Csv file or text file are not suitable when you go to production and you want to run a distributed processing and improve performance. So if you are looking for performance in a distributed environment, don't use the extract. Okay. So today, I will explain to you what is a good file format how to understand and how to to, to, to say, this is a good file format. So, and I will explain the main 3 format compatible with hadoop, and almost all the other tools from the same from the hadoop and spark, ecosystems and oversee. There is many more others, and many more. But, for instance, we are focusing on this 3 file format. I will explain to you how to create and read this Avro parquet and olc format. Do you have any or someone have any already? Have an experience with this file format? Do you know about this? Or just tell me it is the 1st time you you heard about you hear about this file? No, okay. I have no. Okay. 1st time book it great. So you will learn something new, although great. Okay. great. So the question today for you is how to optimize files or data set size. Remember. in our 1st class, in the introduction, we talked about these different file size and so on. And we we know that the size and the data transfer or the scan are correlate, scan time to scan or or look up into a very large file, and the size are correlated. So if you have a very small file, let's say you have 100 product. So 100 product. Let's say it takes 2 kB on your storage device. And you want to just to explore this 100 products. Maybe it takes less than one second to read all these hundreds products and return some aggregation, or or enumeration, or whatever. But if you have 1 billion. this is at the same time you need more time to scan or to look up to retrieve the data you want, so more rows or more size. Large is the size, more time you need. You need more time to explore and look up your data. So the 1st thing we are looking to to do or to to improve is the size. So how can I reduce the size of my data set? But again, by reducing the size, I need also to keep the same data quality. I want to improve the performance. So I have many factors. That's only reducing size. It's still be readable, because I can use a very simple way to reduce the size. I can compress my data. I can use Zip Archive. or a Tar, or a Gz, whatever. It will reduce the size, but it is no more readable, no more readable, to be processed. You need to extract it before, and then you need to reprocess. So the problem is the same. It is not solved. Even you are compressing the your input. Data. It is not solved, because when you need to process, you need to unflat, and then reprocess, and as you re. you will get again the same size as the original file, so we didn't do anything. So so here we are looking, how to reduce the size and keep this queryable. how to optimize querying time? Because, okay it is. I found the way to reduce the size. But maybe it is not optimizing the querying time. I reduce the size by compressing the input data. But it is not optimizing the querying time, because when I need to query. I need to extract from the archive and then reprocess. So it is not solving the problem. So if I had to just enumerate some file format. Maybe you know, maybe you don't know. So you you already for sure know about Csv files or Tsv, you know about row text, so it can be also rotext without any header whatever. G. 7. Format protobuf Protobuff is more protocol to exchange files. The thrift also is the protocol to exchange files is used by hive. Whenever you need, for example, to connect a tableau server, or you need to connect any external application to to your hive Metastore. The data will be, or the metadata will be transferred using this protocol. The Thrift Protocol. So sequence file was the 1st contain binary container created by hadoop to store the small files generated by the mapreduce algorithm, remember that mapreduce. we have the input phase, everything is K value and the map output intermediate result. And this also in K value format. This intermediate result are transferred, merged, sorted, and shuffled, and sort and transfer it to the reducer via network. And here the sequence file was created to group as a content in a single container all these small files generated by the intermediate mapper. So here it is. First, st I would say, Yeah, binary container to group, because Hdfs does not support small files. So sequence file was a solution for this also. Many others have repart K format. So files can be organized in based on the complexity. We can say, this file is the complex by nature or complex by structure, complex by letters, for example, audio file, an audio file, an image video recording these are complex. Binator complex by structure could be, for example, a Json file could be an Xml file. So and you classify this, the data is classified, structured, semi-structured, or non-structured, and the files are classified based on their complexity by nature or by structure. And also we can classify these files regarding their performance. For example, if you take a Json or or you take a Csv, even, we can put the same information, but the representation of the structure is not the same. So in the Csv file you can take one. Let's say, one row from your product table and put it in one line in your Csv file. But if you want to represent this information into G 7, you need to create the structure of the G. 7. You need to create, to give the attribute name followed by the attribute value. So g. 7 represent the same information, but it is more, it is a larger, larger. So we have more characters, more, it is more verbals. So we need to put more characters. So the file is larger. So you see here it maybe it is not a good way to. If you want to improve the performance, to analyze the Json or or a Csv instead of a Csv file. So now, for these reasons. So we have to optimize the size, we keep it queryable. We optimize for querying. 3 file format was created. 1st was created by Hadoop Avro, then by parquet, and Orc is used in in mainly in in hive. Okay? So what is a good file format. In the illustration. You can see 6 6 features or 6 element. I have the left side. and I have the right side. So the in the left side I can see. Feature 1, 2, and 3, and on the right side I can see 4, 5, and 6. Okay, 1, 2, 3. These features are for human. 4, 5, 6 are machine side. Okay? So for the human. a good file format can be interpreted as it should be well defined. I will explain what this in a moment should be expressive and should be simple. From the machine side the good file format should be. should use. An optimized binary encoding should be compressed natively. I don't want to use or to add any new tools to compress or uncompress, so which need to be native. So compress it natively. As we are using this in hadoop environment or on Hdfs because you are using the block approach to store our data should be splittable. so a good file format should have. All these characteristics should be well defined, should be expressive, simple should use very optimized binary encoding should be compressed natively, and as you are using this on hadoop, express ecosystems should be splittable if you are not using an hadoop, and it is splittable. No problem. But again, this should be splittable if we need to use to use this file format on hadoop ecosystem. Okay, so what is a well-defined file format. When we say a file is format is well defined. This mean the reading or analyzing a file, so the reading operation should not be interrupted by a missing or miscoded value. The reading operation should not be interrupted. So let's take this example. I have a Csv file in this Csv file. I have the 1st line, which is our, the header. and I have one line. So I have movie, id and title. I have my movie, Id, 5, 5, 2, 6, 9. I have the title Dargenic Limited Comma, the we as a human. We understand that the word the is part of the title, the Darjeeling Limited. But the machine say, Okay, I have a header with 2 columns, but I have one line or one row with 3 columns. What should I do. it will raise an error. It will stop reading so, or it will ignore. I don't know. It depends. But here most of the case. You will get an error because you have a colon, and we don't know what to do with. So here the reading operation was interrupted because it was miscoded. I. This is why many many operating, not operating, many administrator choose to enclose this into double code. because they want to use the comma, but they don't want to change the file format. So they use. Or they put double code to say, Okay, I don't. I want to use this literally. This is not a comma you can interpret. This is literally. But if do you think using this double quote, or enclosing this, this title into double quote is a good something good. I don't think so. I will. I will explain why, in in a moment so well defined means reading operation should not be interrupted by a missing or miscoded value. Now let's take expressive so expressive file format means the reading should be as easy as possible. This mean, if I can read or perform the reading operation in a single pass. This is efficient if I need more than one operation or more than one task. to read or to get the data, it is not efficient. So let's discuss this example. So here we have the movie title, the movie Id, the title and the genres. So we have the movie Id. We have the title of the movie and the genres. We have here a kind of a collection separated by a pipe. So we have comedy, drama, romance. To parse this is correct. No problem. This is, it is well defined because there is no miscoded or missing values. But here we need to operation at least to parse and get the collection, array the generous array, the generous collection. So I need to read comedy, drama, and romance and split you this column, the value using the pipe so it will return for me. The 3 element of this collection in this example. So this mean to get the genres. I use it to operation reading and parsing. while if I use it, a Json format in a Json, it is performance in a single task, because this is a native Jason array in Json generates. We just read in one operation, I get my array. I get my. The element of my collection. So in this example. it is not expressive. So back to my question of the previous slide. do we consider enclosing in double quote good practice? No, because it will make the file format not expressive, because to parse, to get the full title, I need to read. using the double code and perform a task to remove the double code. So including or enclosing into double quote. Yes, it works the miss. There is no miscoded or missing values, but the file is no more express. The file format is no more expressive. So here, in this example, you can see that g. 7 is well defined and reason is expressive, because G. 7, there is no miscoded or missing values the and it is expressive, because in a single reading operation you can get the information. Even this is a collection. So the 3rd characteristics is to be simple. Now, Xml is well format, well defined, but it is not simple. Why? Because when you create an Xml many, many times, you need to define elements, you need to define namespaces. You need to define the tags you need to define the schema to validate the many. Many information you need to include into an Xml file. So Xml is well defined, is expressive, but not simple, is well defined is expressive, simple. There is no tags, there is no namespace to define, there is no validation sheet to to define. So G. 7 is. for now is the winner. But this is from the human side. Now let's take a look to the machine side. Okay? So the 1st characteristic, the file should be or should using should use, optimize it. Encoding file format. What this mean? This mean when you store on the storage device, this should use the smallest space possible. Okay, so everything should be encoded and written in Byte, of course, to reduce storage. But now we are not looking to be readable by the human. This should be readable by your machine. Json is readable by a human. So if you take a Json file, you can understand exactly what it is. The content of this Json file. So let's take the example. So I want to encode or store this number 1, 1, 0 3, 4. So if I use the Ascii characters. So Ascii, I would say, the 1st habit is just the very, very basic Ascii. It will use 5 Byte, (313) 133-3032, 34. So 32 is the space. And yeah, it is 33. I don't exactly remember exactly what the encoding, but it will use one bite on the storage device. So 5 characters encoding this in Ascii will use 5 Byte. But now most of the representation, the operating system. Use Utf Utf because we have different languages, we have different characters. And so so it use. Utf is 32 Byte. So it use up to 4 Byte on the storage system storage device. So Ascii is not optimized. And this is what you are using when you you store your Csv in text file, using Utf 8 encoding characters encoding. so let's say, in this simplest format, it will use at least 5 Byte. No Ibm in, say, early in the 70. Maybe they created the big Indian and little Indian encoding system. So this was created to represent numbers numbers with decimal floating point, and so on. So it was a way to reduce the number of bytes. So instead storing this in Ascii, 1, 1 0, 3, 4 in Big India, I will not go into the detail of this encoding system just to let you understand that this encoding format used 4 Byte on storage system instead of 5. So here I just save it. 1 Byte. Okay, great. But it is not enough. Another encoding system called zigzag encoding. so the exact encoding will use only 2 Byte to represent the same value, 1, 1 0, 3, 4, and the value of these 2 Byte is 9 a. And 5 6. So here I save it 3 Byte. So if I have a very large file here I am saving more than the half of this. I reduce the size by half. which is a very good. So using a very efficient and optimized encoding file format, it is very important. So again, this is from the machine side. You don't want to go deep into all these details, but just to understand how this file format works and how they are behaving with your data to store, to reduce the size and to represent the data in efficient way, and so on. The second characteristic for the machine side should be compressed natively. we all know about compression. Compression is used to reduce the storage size. So it is a fact. But this might increase the processing time, because you need to extract or uncompress. then process. And maybe you need to recompress later. Okay, so compressing and compressing, this will increase processing time. And we don't. We are looking how and searching how to optimizing this querying time and processing time. And also you cannot use any compression format. You cannot, because this should be splittable again, splittable means compatible with Hadoop Hdfs. It could be able to store this and process this on Hdfs, so the compression will reduce the size. But here we need to to improve the processing time and ensure that this will improve, the process will not decrease. The processing time should be splitful. What this means splittable in in real world, I would say so. This mean, you will be able to process the part of the file independently, because we need to process in parallel. So if I have my file started on Hdfs in blocks, I want to be able to read these blocks in parallel individually, and be able to process this data individually, and do not wait until loading all the blocks into memory and uncompress into memory process and recompress. This is not the way we want to do it so if it is compressed natively so, this should be splittable. This means all the blocks stored on Hdfs are compressed, but I am able to load independently any blocks and process it as I do be with a file, not compress it natively. Okay, so this a low parallelization. This is why it is very important to risk, to, to, to, to fill this characteristic be splittable. Okay. Now, the best file format. If I summarize what I said, this should be well defined, expressive, simple use. An optimized binary encoding should be compressed natively and split one. All the files. Your format, you already know, does not mean this expectations. Csv. Do not, does not meet this expectation. Text xml. Json, HTML, no, no. One. So this is why main file format was created. So avro parquet and Orc Orc is a mix between Avro and and parquet. But again. each of this file format has pro and cons, and cannot be used for any case or for any use. Case no, we need to use it for dedicated use case, and you will understand in a moment. So this is just a table to summarize what I just explain it. Psv is not well defined, is not ex is well defined because I can see say colons, and it is separated, and so on. But it is not expressive. It is simple, but it doesn't use binary encoding system, optimize it. It is not comprehensive natively, and it is not splittable until now. I didn't provide you a file size that exceed the size of a block on Hdfs. You work it with files that all the file can be stored in a single block. But I didn't provide you any larger file than the block size on Hdfs. So this is why you can work with this easily, but when it is on the block and you want to read, you will meet some error because your data is split on the edge of the block on Hdfs. So file the best Avro. It is well defined, expressive, simple. It use a binary encoding, optimized binary encoding. We'll see this in a moment. It it is compressed natively, and it is splittable, parquet and oversea the same. It is well defined. It is expressive, simple, very simple. for not not for all I would say to understand. Not very simple. Maybe the 1st time. But when you have to to deal with you will be very familiar, and you understand Markdown is not cannot be used just. I would say it was derived from HTML. So Markdown cannot be used as a file format to store and process your data at scale. So binary the parquet and oversea use an extremely efficient binary storage format, extremely efficient to give you an idea, and you will see this in the workshop. We will be covering this after the break. If, let's say, if we have a file size, let's say, 10 GB, 10 GB in Avro. It will be reduced to let's say, 4 4 GB from 10 to 4, so you save 6 GB. If you use parquet, it will be reduced to 1.2 or 1.5 GB, from 10 to 1.5, with parquet. parking and Orc almost the same. So very small differences. But we are the same range. But you see, from reading 10 GB, of course, and for sure you need more time to read 10 GB than 1 GB. So for sure, parquet is extremely fast and faster than Csv. And this is, you will see this in in the workshop, in in the after we cover this after the break. So it use, as I said, extremely, very good optimized binary system on the source system. It is comprehensively, and it is splittable. So it meets all the characteristics we are looking for. Okay, so now let's go into details for each of these file format of row, parquet and oversea. So Avro Avro is a good file format. Of course. Avro is a data serialization system. It is role oriented and need a schema. and it has a very flexible model and can be implemented in almost all the programming language. You can use this from any programming language. So data serialization system, what this mean? This means, it is designed to collect your data in real time. This is a data serialization system. So you send. or you want to to insert the data into your Avro container. It is serialized. So when we collect data in motion. the data is coming over and it is coming, coming, coming, coming. So you need to store this somewhere. So Csv file, no. So Avro, here is very good file formats. When you are collecting your data in motion. data is moving. You collect data in AV, you are transferring data from so location source location to a target destination. It was Avro. Because this is a data serialization system. It is row oriented will. This mean data is organized in rows. It is very, very similar to your table in your rdbms. So you have a container in this container. One row follow the other. One row followed. Serialize it. So this mean they are stored one after the other. So when you create an Avro container, or if you want to use an Avro file, you need a schema. So the schema here is to provide as you did in your spark data set, you provide a schema, and this will map your internal data to this schema. So you need to provide an adjacent format. the schema which is the enumeration of your column name Colon type, and if it is if this column accept noon or not. Okay. And this also, the schema is mandatory. You cannot create an Avro container or Avro file. If you don't provide a schema, it is not possible. Just understand this as a table. So the schema here is the table, column, name, and type. But instead, here in the Avro, we are providing this in Json format. Okay, I will show you the details in a moment. Okay, so, Avro, provide us a high compatibility with Hado, because it is splittable, natively compressed, and so on. It is. It provides a rich data structure because we need to provide a schema and the schema. You can use a nested object. So we can. This is very rich structure. It is compact fast, and the data is natively stored in a binary format. and it can be used from many, many different programming languages. So there is no limitation, and for sure you can use this from dynamic language, such as such as python, for example, you can use. So Avro benefits are, write and forget, and especially for data in motion. Write and forget what this mean. This mean. You connect to your data source. It will be stored on your storage device in the Avro container. And and when you say, forget this mean, you don't need to worry about the data in case of failure in case of failure, your data is saved on disk. There is no data loses in memory. Nothing is, everything is stored on disk, and as it stored, everything is on disk. Avro has very, very low memory usage. Everything is on disk. When you collect your data. Everything is on disk. Nothing is stored in memory. So and again, it will ensure the data is written, and especially in case of failure. This is what we understand by write and forget. So this is safe format, and especially for data in motion. Now, if you take a look to the table, we are the logical table. We here have 3 columns and 4 rows. So the column name and type. You provide this as a separated schema. When you create the container, and now you insert all your rows, and you can see the storage pattern. So we have the 1st row, followed by the second row, followed by the 3, rd and so on. So this is what we mean by row oriented every every row follow the the other. So just as a queue. you see, overall is very simple, very safe, and it is used especially when you collect data in motion or when you transfer data between systems. Now for parquet. It is, of course, very good file format, but here it is. Column oriented. Avro is row oriented while parquet is colon oriented. So what this mean? This mean in Avro or in your let's say in Standard Table you have one row, and in one row you have the information about let's say an entity. You are the representing a contact. You are representing a product you are representing a person. This is one entity, so one row, an overall, or one row in your table represent one value, one entity, one value. In parquet, one row encode a group of value instead, one value. So, for example, let's take the product example I have in in my table. My the the product I have in my warehouse. So I say, Okay, I have a product, one product, 2, product 3, and so on. So each row represent one product. In my warehouse, in parquet. one row represent a group of value. I can say in this row. I have all the product from this category. or all the product in this department. or all the products sold this month. See? So here I am encoding a group of value, or representing a group of value is instead one value. So one row in parquet encode a group of value rather than one value. So overc is, use it almost only with hype. It is used by hype, but, unlike Orc. parquet and Avro are commonly used outside hive. Many, many applications use parquet to represent their data, and especially when you have to process or to query your data in the cloud, because you pay for the computation. So you load. If you load a Csv file and your Csv file is, let's say, 1 GB, and you need to process this in memory. So you need to load 1 GB in memory, and all the time you are trying to query, you pay for this on the cloud. This is the compute part. You are charged for this for. So if you have this in a parquet, and instead, the 1 GB you loaded, let's say 100, kB, you pay less. and you will save money because you need less time to process it to query it. So parquet is used, especially when you are running this in the cloud or with the distributed environment. As Avro, we need a schema. So parquet also need a schema. We used to provide the same schema. If you create the schema for the other container, you can use exactly the same without changing anything. When you create your parquet container. So you create your schema once. and this schema can be used either for parquet or for for hour, and could be also used it with the or see. But we'll see the little later how how to do it. Okay? So parking was initially created by Google. So it is was published on Google Dreaml paper was academic paper, and explained how we store efficiently in column oriented. File a binary file. Okay? So the benefits of using parquet. So parquet is extremely fast, very, very fast, while reading extremely fast, because in a single operation you load a group of values instead of one value. So it is extremely fast, and you can read selective columns in one row, one row can. Maybe we can have maybe 10 million row. No problem, 10 million columns, no problem. You can read selective column. No problem. There's no restriction at this this level. So it use an extremely good encoding storage. So the data is very small. I just gave you the example. 10 GB on Csv. Will be reduced to 1.2 or 1.3 GB so extremely efficient, and to again, we can support different compression and encoding scheme. We'll talk about this in the workshop to better to understand. But here, let's take a look how it is organized, and what we say, how to understand what is a colon oriented. Remember, in Avro we have our table, 3 column, 4 rows. We take the 4 1st row storage the container, we read, the second row, storage the container, and so on. So here, in this illustration or in this example, I have the table. 3 columns, 4 columns and 4 rows. Okay, the 3 rows. Sorry. So I in the parquet forma. I start by reading the 1st column. and I will put or insert all the values from this column until the end of the table, so it needs to read all the rows in this column before starting, storing the second column, so you can see here, you can just say, Take it to the blue column, the Id columns, the blue. It reads all the values. then start storing all the values from for this column, from the second column, then the yellow, then the red, you see. So this is how the data is organized inside the parquet file. So but here, maybe we is need to wait. Parquet need to wait all the rows you see here. Sorry here, for example, for the Id column. we need to read all the rows and then start storing the second column. So this is what we call need to wait all the rows to finalize a column. So this means all these rows are grouped into in memory, so parquet will create groups groups for this column in memory. So parquet has very, very, very high memory usage very high when you encode, not when you read, when you encode, when you encode. it needs to load everything in memory and from memory it creates a group memory, and then it will create a column. So it needs to wait all the numbers. But if we have not know 1 billion row how this works. parquet will flash on disk, so it will load on memory what we call a page. Physical. Page 256. Megabyte. Once it is full. it will flush on disk and continue under the latest row in the table. You have 1 billion, you need to wait until it's finished. But the problem with parquet is the memory. So you need a lot of memory when you encode. And in case of failure, data is not guaranteed to be written because it is everything in memory. So now you see the difference between Avro and parquet. When you transfer data, we use Avro because it is secure when we process data. Now that we use parquet. But when we encode the parquet, we need to to pray. Do not have any failure, because in case of failure, you need to restart the encoding. But it is, if very, very, very efficient. Again. When you have a small table or small file, it is doable, verified, relatively fast. But if you have very large table you need to wait. You need to wait until it reads all the rows and then start all the columns and so on. Now Orc is a color oriented. It is a mix. it is a mix, so it use the best of Avro and the best of parquet. and combine these 2 best characteristic from each file format into a single file format called OS optimized row columnar. So it but it is considered as column oriented, not row oriented, column, oriented and similar to to parquet. It will encode a group of values rather than one value. But Orc includes metadata. So it. While it creates this group of Colon in a single row. It will include what we call the bloom filter. So the bloom filter is here metadata in the header, and will, for example, when you perform a count. The count needs to go and scan all the rows in your table and do this aggregation. Remember, the I just explained in the high class to use the analyze compute statistics. So this function, we'll we'll covering this today because we'll be using this in the workshop. But the idea of this function high function is to do this. the filling of this bloom filter. So it will do the count once. It will do many, many other information and collect many information and statistics, and in store this as a header for the row. So now, when you perform account. oversee and hive does not need to rescan your table. Just go to the just bloom filter, and from there, from there it will read the information and provide it to you. So performing a count will be extremely fast. So this is a way how to use oversee. And again, it needs a schema so overc it is mainly created to be used in hive in hive. Of course Trino can read Orc and is compatible. Spark can read Orc, because we are talking about all this ecosystem in hadoop and spark so they can read it. And from spark you can also create and read and write orc format. So also we need a schema. So if you create this from outside hive. You provide the same schema we created for Avro parquet and can be used in the Rc. If you create this from hive hive is smart enough to say, to consider your schema table as overc schema. So, for example, you say in hive, create table products, id description, unit price, and remember in hive, we say, store it as text file. So here you say, store it as overc by using this storage as Overc. So hive will convert your table schema into your Orc schema, and it will use it as the schema for this container. So the oversee benefits are almost the same as Avro and parquet. So it is suitable for read and heavy workload. Very long queries. Remember, hive was designed to not be interactive. It was not designed as an interactive tool. It was here to process and query your data even. It takes a long time from oversea. You can read selective column because it is colon oriented. And so here we by default, there is different compression library we'll be using. You see this. You'll see this in the workshop, and by default. I've used the Deflate Compression Library and the Deflate Library has a best compression ratio than others. But again, I will explain this part, because this is very important. Part. I will explain this in the workshop, because better to understand, because you will see the size, the numbers, and everything. and it is, of course, natively supported by hype. Now, how to create any of these containers? I want to create an Avro parquet. I want to create an Avro container or parquet container or overc container. The 1st step is to create your schema. I will show you how to do it. Create the schema when you create your schema. If it is not in hive as a table we use. This is just a convention. You we we used to use the exit file, and we use the extension Avsc avro schema. The same is used for Avro parquet and Overc outside half. So we start by creating the schema similar to when you create a table. You want to read a table data in hive, you create your schema in hive, and then you load the data or populate the table. With this data. it is exactly the same process. We create the schema. Now, we. when you create the schema, you use it to create the empty container to create a empty avro per k container. Oversee it will, as it is in hive. When you run the create table Hive will infer the schema versus schema from your table schema and create this empty container for you. The 3rd step is to serialize or to import the data into your container. Why you need this because it is natively compressed. Natively compressed means. When you insert the data into the container, it will be compressed. So if you, if you create a table in half. and you say my contact customer id customer id 1st name, last name, and you say store it as parquet external storage as per K, and you give the location, and you have a text file. It will not be converted into parquet. The table will be still be written in text format because it is not enough to create the table and specify the format. You need to populate explicitly the data. So how can you do it? You can use. Insert into, you can use, create, select a create table as select. So you need to to populate this explicitly, otherwise it will not work. And here it is very, very important when you create your schema, to always always use lowercase, always because use it is case sensitive. The the schema is case sensitive. So always think and use lowercase always. Once you create the schema, you create the empty container you insert into or populate the data your computer with your data. Now you can use it, and you can read from any any Api provider such as in your python, your spark in hive in Trino. Whatever so many, many application able to read your parquet will be able to read your or Avro or Rc. Will be able to read it. So let's take a look to the schema. So the schema you provide nothing but the column, name and column data type. So the Avro scheme or parquet or overc, provide you the primitive type. provide you Union records, and provide you a way to to create collection, map and list. So the primitives are Boolean, long, double, sorry string. So all this primitive type union is used, especially when we want to use null. Remember, when you created the schema in spark, we say, add struct type, add column, name, data type. followed by nullible or not, true or false. So union in the Avro parquet schema is used to specify this characteristic. So why? Union? Because I say, okay, it accept new and it is string type. It accept neural. And it is int type. This is what we mean by union. It is not the same concept, as you already know, in your table or Rdbm. Sd, you take 2 table and merge them into one output. No, the union here is to draw, not to join, to, to let you use more than one type. noon, or the data type nothing but so record it will. It is used to define the type of the schema you you want to express. So always, we are using this to represent a kind of a table. So always you're using the same. So don't Don't worry about this part. Just use as is the complex type, and it will be tool works. So now let let me show you an example. So this is very simple. It is a Json file. So we start by 3, attribute the type, the namespace. the the name, and the collection of the fields, since the schema. So the type is a record, so it will shows always. This is always use, always the record. Don't don't. don't waste your time to search other type. Just always use record. So it shows the type of the document. Okay, generally, document. Because here we are using more than one column. Okay, the namespace here is similar to the database, because in the Avro support multiple, our support multiple schema. And it supports schema evolution. Oh, this me. Today, you have a product. You want to store the table product in your Avro container. Okay? You say, I have. I product, id description, unity price, and many other colors. Later, you want to add a new column to your product team. let's say quantity in in the warehouse quantity available quantity or discount, or whatever. So now the data in your container is mapped to a particular schema, and your applications are using this data based on this schema from the other container you can add a new schema and add new data with the new column and all the applications still be able to read. If you hit the version one schema, the data will be returned as version one without the new colon. If you hit the second version, so you can the new application or say version 2 can read the version, 2 from the container. So Avro, support this schema evolution and virtual version. So the namespace here is just kind of database just and understand. It belongs to this namespace. Okay, for example, the name of our class, Ycbs. 2, 5, 7. The name oh, here, what is Rep. What are you are representing in this schema? When you create a table, you give it a name, create table, a customer, create table products, create table grades, whatever. So the name here is the name of the the table if you want in the upper container. But again, you will still be able to read your data without parsing this information. But you need to provide this when you create the the schema. Okay? And followed by the fields. So type, namespace, name, and field. This is the 4 attributes you need to to provide in in your schema. So the fields here is a Json array. and you provide the column name and data type. So here you say, name the column name is name capital. N. Remember, this is case sensitive. So when you need to read later, you should provide the same casing. This is why I recommend always to use lowercase. So no problem with this. You don't need to remember if it was capital. N. Capital P. Whatever. So just use small case always the type of this colon is a string. The second column name is age. The type of this column is int. And this is the schema provided, and when you want to create the container, this is how you provide the scheme. Later in the workshop you will learn how to do it, and I will show you how we will be automating this later. Okay, now, this is the summary how to create so hive support, creating table storage as Avro stored as parquet, store it as Overc. But again, even you create a table. And if you say, Okay, create external table product or restaurant. create external table restaurant. You provide the color, you say, store it as parquet. It is external. Use, location and point to the Csv file. Your data will still in Csv format, it will not be converted. You need to create a table and do insert into or select create table as select. This is the only way to do it, and the most common Library Compression library are snappy, zip, Zlib, and deflate. These are the most common. I will explain. and you will see the compression factor for each of one of all, each of these compression libraries. And you, you understand when to use which one. So I keep this part for the the workshop after the break. And so this is you. You can be used with any of this file format can be avro snappy parquet, snappy by default. Spark. When you say I want to persist on, disk, the output format by default is parquet snappy by default. So if now you go back and re-explore the case. Study, for example, the h. 1 b data analysis. When you output the data partition data by spark, it will be stored on parquet and snappy. Now you know how to. You will be able to use this data as is as it is stored by spar. And again. All this depends on the one we want using their pro and cons. And you will see this in the workshop. So to create a hive table. Very simple, you need to say, create table table name, external or not external internal. Store it as Avro parquet or C, and you need to provide the table properties. If you are using a particular library table property, for example, you say compression. Snappy Avro is compressed with snappy or parquet is Zip or olc Zlib. so you can define any of the supported Compression Library. And then, as I said, you populate the table using load or insert into or create table as select. So this is the way you create and populate tables in in hype. So look at the example you have create table data, have row. you provide the schema. Okay, store it as Avro table property avro compress snappy. So here I am using snappy to compress internally my avro container. And now I populate this table, using the insert into so insert into the table name that I've select from data. This is another table. Okay, this is the only way. Now I can provide you. I can provide you a a container already populated in this case, if you'd say, create external table and point to this already. prepare it. Container. This will work. We don't need to do any insert into or create a table. As so, this is only if the format you want to create from scratch, but tomorrow or or or later you will see. I will provide you already prepared. File Avro parquet, or C, and you will be able just by creating your hive. External table point to the location, and it is readable because it is. It was already encoded. So the data is already stored in the container. So this table is just to refresh our discussion about this format. So schema evolution overall support. Very good. Faïçal Sawadogo: Sorry for interrupting. Khaled El Tannir, M: Yeah, of course. Faïçal Sawadogo: But I had a question in the chat, but regarding the the well formatted files like is Yaman is is is considered as a well format. No. Khaled El Tannir, M: No, because it is a text format, and it is not optimal if we to be a good file for good file format, for to be used to be processed to query. You got your. Faïçal Sawadogo: Okay. Thank you. Khaled El Tannir, M: With them. So Avro is best, if you need to do some schema evolution. But in our class we will not do it, because this is considered as intermediate to advance it. Just keep things simple. Parquet is good, not very good to support schema evolution or see better. Okay, compression split split ability, row or column column, oriented or row oriented, read and write, so have row best for right. Don't this mean when you ingest your data, and especially in motion. Parquet is best for read, and especially when you process your data when you need to query whatever you want to do, and especially when you are in the cloud, and Orc is optimized to to read, to to be written. So read operation so the compression codec, the most common Z. Lib. Gz, bz, Lz. 0, Lz. 4, and snappy. We will be working most commonly with the snappy, because the easy of use of snappy, and it is also because of its compression factor. Again, I will explain this in in the workshop. I better. I want you to see the result better than just talking about numbers and no prepared. You will see by yourself. So data in motion, data at rest. data at rest can be written. Use parquet or overc hive Trino spark, SQL. Convert you. You can use this to convert all this Csv text into parquet or overc data addressed that are in motion. You can use spark streaming. 9, 5. Flume 9, 5 is our next topic. Okay? So we we start by covering I 5, we start, we learn how to start automating by, for now you are doing this manually. So now, starting from the next class, you will learn how to automate how to build your pipe. and you will be able to ingest data in motion with spark streaming. So, data in motion. Our, for example, I want to import a table from my sequel to Hdfs. How can I do it? I will not export this in Csv. I will not export this in Json. Use Avro, if your rdbms does not support Avro because it is export only in Json. You need to convert, you need to convert, and then you can transfer it to another way to another destination. But all this will be covered in the coming weeks. I would say. Okay, now, how to manage this? This containers. So we have the Avro tools. Avro tools is a command line tool which will let you extract the schema, extract the data so it will to do many, many operation. You provide the Avro container, and you can extract the metadata. You can extract the sample. You can extract whatever you want. You can do whatever you want with. So Avro tools is, use it to manage your Avro container. but it is not used to create. I'm not sure if you can create with it. But no, I'm not sure you can create with it, but you can manage. You can extract, or whatever parquet tools. This is the name of the tool, parquet, tool parquet, hyphen tools, avro, hyphen tools. the same. We have the same, but not the same keyword. So, for example, if I want to get the schema from the other tool, I will say, get schema. You can see this, maybe 6 or 7 command in the list. Get Schema to get the schema of the container from the container, and for the parquet it should be Meta something or or schema, just schema. So again, this will be more friendly for you in the workshop, and we have the oversee so oversea tool again, it is here to let us extract the information, Meta information from the oversea container. So here again, we can just extract the schema. We can extract the sample. We can see what is the compression library, use it, and so on. So this is everything I wanted to share with you about this file format. After the break we we go more into details. I will explain to you and show you by in practice how we can do all of this, and you will see by yourself all the differences between this format, size, computation, time, and work on everything. Do you have any question? Okay? So if there is no question, so let's go to the break, let's say 10 or 15 min, and after the break we'll continue with the workshop. Rajesh Kamaraj: Tanil. I I don't have a question but I just I just wanted to talk to you privately on a feedback that you gave me. Khaled El Tannir, M: We can. We can. We can do this after the class. Rajesh Kamaraj: Yeah, sure. Sure. Yeah, sure. Khaled El Tannir, M: Okay. Great. Reem Hoteit: Yes. Khaled El Tannir, M: Great. Okay? So and the workshop. So the you have here, you have 2 tutorials to to one more generally about all the file format and one ticket to spark with Avro and parquet. and in this particular case study. You have also, the usage of parquet from Spark. So let me show you. So the 1st is, let's say in in sequel, hive. So this is the file format from hive, but is not good only for hype. But again, so here you have all the explanation, and how to. I would say, to create the table and import and populate this from hmm. other table, other data source. Okay, so here, what we are doing again, we are collecting this data. This is a large file. We create our external table, and we provide a location. And then we compute statistics. I will show you this in a moment in the result. Okay, and then we have here to create the arrow format advantage. And so this is the structure of the overall container. So you can see each each row or each block in the in the overall container. It has a header in the header. We can see the schema, and we have all these records stored one after the other, and each. As you can see, each block, we have a 16 Byte marker just to link to to the blocks, to be able to retrieve everything. So this is an example of a schema. You can see here we are using 3 column name, type and string. And this is how we use the neural type. You see, for example, this column age accept int and accept null. You see. So namespace, tutorial type record. The name is user. So user column, name, column, age and column gender. Okay, so the column name is a string. It doesn't accept new. The Colon age is type in and accept new. The colon gender name type is string, and it accept new. If you don't want to to accept new in this column, just make it as the 1st declaration here. So this mean, it is mandatory to provide a value. If it not, if the value is missing, you will get an error when you store. When you want to insert the data and said to say, it doesn't accept new. So you need to provide a value. Okay, so I will say so by default, by default. The compression output is in hive is disabled. So I disabled this because I want to save resources. We will enable it only for this workshop. Okay? So here you can see. I can define, for example, the compress. Yes, I want hive to compress the output, and I want to use the snappy. Okay, create table, if not exist. This, the database and table name stored as avro location. This is internal table create table. So this is a hive manager table, but I can overread the location provided created by default. So this way, in this exercise I wanted to put everything together in the same location. So you will be able to explore this easily on Hdfs rather than back and forth different directories. And so here you can provide a location. Even it is an internal table, and once you will populate the table. So here create table. If not exist. store it as avro location, so that will be stored at this location as select from tutorial. So, as you can see. I am populating explicitly the Avro table. So this mean this part. So select star from the Csv table. So this mean, it will read all the rows and insert these rows into the Avro table while inserting it will be all the data will be converted, compressed, natively using the snappy kodak. Now, the analyze table computer statistics again, this is here to collect metadata about the data row size, the compressed size. How many partition, the partition size, how many physical file, all this information can be collected? So when, whenever you need to retrieve it, it will be extremely fast again. So this is for the Avro, so very different way to use the other tools. We can use it directly, or you can use it through the hadoop, so I'll show you the different alternative in a moment. But here I am what I am doing, Avro tools. This is our tool to manage. Our Avro container. Get meta, so this will return all the metadata from the container. This is where the physical file, as you remember here in our table create as stored as Avro. So now in this location, okay. So now. from this location, I've created this file, this physical file. So I am using hadoop jar avro tools get meta, so it will be able to read from Hdfs this file and extract the metadata from this alpha container, and the result is here. We will see this in the zip in a moment for the parquet the same. So here is the structure of the parquet file, so the size by default in Avro is 128 MB in parquet is 256 MB. The parquet does not use the 16 Byte marker to link this chunk on the storage device. And by removing this 16 Byte maker, it will save a lot of space. Okay? So when you create the other container, you provide the same schema, same format. But when you retrieve the data, the schema from the container, it will be a different format. So here for this is an example of a schema extracted from the parquet container. So it will say, Okay, optional means it's exceptional int. 32 binary and name etf. 8 binary. This is a string utf 8 double salary in 32 the age and compression you can use snappy Jz, Lz and Lz O again. So the idea here to let you see the difference and appreciate this storage size and everything the idea here is to take to create a baseline. using a Csv file. Show statistics from this file, then create a Lavro container. gather statistics and compare it to the baseline. do the same with the parquet, create a parquet, get this, collect the statistics, and compare it to the baseline and compare it to other. This is what this tutorial is trying to to, to to teach you. So we are repeating the same process, but with different file format. Okay? And the parquet tool the same here when you do hadoop jar, parquet tools, Meta, to get the Meta information from the outputted file. You see here it is a little bit different. So here you can see we have the schema. And, for example, here you can see the Meta information file Hdfs tutorial, the physical location, the Creator. Here we have the schema user, id movie id rating and timestamp. It is optional. This means it is accepting null int 32 is the data type and float at X, int. 64. So here we can see. No, it is not compressive. No, it is not. It is not compressive. because we should be able to see the maybe it is not in this screen capture. Okay for oversea. We are repeating the process. So we create the table we populate. So this is the structure of the container. It is a mix between Avro and parquet. And here, yeah, this is an example. Again, I am asking explicitly. I have to compress the output, so I can compare, and I am populating the rating oversee table from the rating Csv table. I compute this statistic. I collect this statistics and show the table properties, so I can see the number of rows. Number of rows should be the same, and all the table, but the size might be different, should be different. So here hadoop, jar, orc tools. Meta, this is the function to get the Meta data, as you can see here, you can see the compression it is using. Z. Lib. And, as you can see, we have the stripe. Here we have. This is the bloom filter. You can see, for example, Byte on disk. 1st Max and other main Max, and many other Meta information needed by by Hiva, and then we have the schema with the color. and, as you can see, it is a little bit different, because it is a mix between Avro and parquet. Okay, so this is comparison table. You have let me. So you have this. Oh, you have the second tutorial with spark. So here you have park, spark avro and parquet, the usage of parquet and spark and parquet in spark again. So some don't use the same file, and things don't use the same, maybe the same. Okay, so here, how to output and use this from spot. So here we have the input file, which is a Csv a, Csv file. Okay, we load this on Adfs, we load the Csv using the read into a data frame here, I'm not using the data set. I mean, just spark data frame and type it data set and convert to parquet. Just write options. Okay for compression. Of course, here we don't use header. There is no header in parquet, because you provide the schema formal parquet and safe. So this 2 options for option and for my are by default. So if you don't specify these 2 option. I would say 2 2 function, option, compression, snappy and format rk by default. Spark will output in this format if you need to change the compression from snappy, for example, to Gzip. To use Gzip in this case, you need to specify this option. If you don't want to change parquet you want to to store as avro just use format avro. So here, so you can easily, with the spark convert between format. If you read. You read using the Csv form in Csv format. Just write in the output desired format, arrow or parquet or oversea. We'll see because it is compatible with hive spark can read hive tables. So this is why you can write or output data and oversee. And now to read a parquet spark, read, you can provide a schema, but it is not as Avro and Parket con already has the schema embedded. It can read it no problem. but you can provide it if you want, but usually we don't. We don't. And read parquet. Just provide the parquet location for file location, and it will read. We can now work with it as exactly any data frame or any data set loading. Avro, it's exactly the same process. Spark, read format Avro, and it will read from Avro. Okay, so you see, in Spark, it is somehow easy. Now let me show you from Zeppelin. Okay, so the second part of your tutorial hands-on workshop. This is a guided workshop. So here, this is a multi-part. We start by creating the baseline. Then we create Avro per K or C with different compression ratio, and we compare all of these files. So here I provide to you. So it is a guided hands-on exercise. So the data preparation. We read the file. This is this all the instruction. So if you you can copy or you can get this Zeppelin note, but we need to do it in this order, A, BCD, because we start by creating the data. So let me run this. I don't know if I already did this. So we are reading this page, count File, and to get the size on Hdfs. The size on Hdfs is almost 324 MB. So you see it is a large file, and it exceed the size of one block on Hdfs. So dfs hyphen du disk usage hyphen H. Give it in readable format. This is why I have the megabyte, otherwise it will be shown in byte. So Megabyte is more readable for me. Okay, so okay, let's create the database. So I'm not sure if already created, but not the problem. So it's create the database. So in case of you want to repeat the experience, you need to drop the table. It is from now empty. And let's create our 1st table. This is our baseline. So create external table database, file format speech count. This table has 4 column project code page, name page views, and by this is a log from Wikipedia. So it is a row. Limited fields. The separate character is this space line terminated by new line. This is the format of the input file store it as a text file and location file format leaves us. You can. From here we can explore virtual file format. And this is our lab, learn data. And this is our file. And you can see 324 MB. Okay, so let's create the table. And now it is an external table, so I can just load and select, so I can see the content of this tape. Now, this is the important part. Analyze table file formats, page count, compute statistics. So now it will collect all the statistics for this stable, and to show the statistics we need to use the described formatted so you can use, describe, extended. but it will show you the result where in one column it is not readable. So instead, we'll be using this in the described formatted. So this will show as formatted output. So we can see here we have the column name column data type. and you can see the database file format the location. This is the physical location where it is the type. It is an external table, and we can see here the number of files physical file. We have only one file. If you go to Hdfs and we browse we have only one physical file. Okay, number of rows. This is the number of rows, the row size, the total size. And so this information can be retrieved from the statistics computed. So you can see here number of rows. and we can see total size. See total total size. How to retrieve. You can say, show table properties, total size. Okay, so to return the number you see on the right and for the page, count number of rows. You can do the same. Show table property from the page count, name rows. and this will return the number here. This cannot be done. If you didn't compute the statistics. If you to to get the row, count, you need to run. Select Count Star from this table, so this will take time to to process and return the count, while, if you already computed the statistics, it is already here. But in in hive there is a glitch if the data, especially for parquet, if it is not created and populated by hive. This information is not available. Even you run the statistics complete statistics. but I think it was fixed in the version. 4 of hive. I'm not sure I just didn't check it yet. I'm processing and studying and testing this and validating this. Okay, so this is now our baseline and the next step we will create the arrow and compare to this result. So the total size should be different. Number of rows should be the stay the same, because we import the same number of rows. But the size should be different. Okay, so this is our 1st step. and the second, you have all the steps in the provided document. I have everything, all the steps. So let's run the second one. So now I want to create the upro container in case of we need to re do it again. Just drop the table and restart the so by default. Again, hive will not compress the output by default. So here I want to ensure that it is set to false no compression, because I want you to show something. I want to show you something. So here I'm creating the Avro table. Store it as Avro, you see, create table, store it as Avro. and I want to overread the default location created by hive. I want you to output to this location and the create table as select. So this will retrieve all the rows from our base table and import into avo. So this will be run, and the output is not compressed. So I am disabling the native compression of Avu. Okay? And now, so this maybe take 30 seconds or 20 seconds. It is not not very large. So okay, finish. You see, it took 20 seconds. And now let's compute the statistics because we need to compare the same condition. So let's come on and compute the statistics and compare the result. What I can do, I can keep this open. So we can compare. okay. okay, complete statistics took less. You see, 14 seconds. And in the Csv file complete statistics took 21 seconds on my machine. Okay? So here it was little bit faster. because the size, maybe is different, or the structure is different. Many things. So now let's show the information we are interested in. So here you can see number of files 7. It was one. Now it is 7. Number of rows should stay the same the total size. Oh, it is little bit bigger. Okay. so I will explain to you. Why did you? Do you guess why is bigger? You see here, the data initial data size is 340 MB, or 324. Megabyte. Okay, here. 366. So this is larger. Why? Because I will use 16 Byte marker to share or to link blocks. And this is the size. The additional size is for this 16 Byte block marker. Why, it is bigger, because it is natively compressed, and I disabled compression. So now it is raw. But again, now I will repeat the experience. And this time I say, Okay, keep natively compresses the overall and this time use snappy. Okay, so let's do it. So now, it will import the data using the from the Csv. this is the original table file format page count. So from the Csv or the text format compress import into the Avro and compress using snappy. Okay, it took 19 seconds on my machine. So let's now compute the statistics to be able to compare. And okay, you see, it took 6 seconds. Now let's show this Meta information. So here, number of file it is 7. The number of rows the same, and the total size. Oh, shrink it to 1, 47, MB. Great good news! It is now almost 145 MB. Okay, so not that. No, I almost 50%, almost, because originally I have 324 or 340 MB. And now I have 147, almost the same. They have okay, great. And I have 7 file instead of one single file. Remember, in the original. I had one single file. Okay. So now, if I go to Hdfs, I have my arrow Avro, which is here no compression. I have my 7 files. You can see from 0 to 6 7 7 physical files. Avro snappy. I have my 7 files from 0 to 7, and it is so here? Why, I have 7. Because this is how many process in parallel we run to to compress, and maybe if you run this on your side, you should have the same output number of files. But maybe if you run this in a production environment with more resources and more CPU core, you get maybe different number of output. So the output depends on the resource you are allocating to process this, to perform this task great. So now I can. I need to repeat the information, but this time I will use the deflate, so deflate is the default, compression codec used by hype hive, use, oversee, deflate. This is by default. Spark, use parquet snappy. Okay, so let's see the output. So again, we repeat. the operation created a new table. Save it at this location. So I can compare the same directory. I can navigate on the sync in one directory instead, go and different directory. And now so take 20 second on my machine compute statistics. This is why it is important to run the computer statistics, but because it will enhance and improve a lot. Information. Okay, so now we have 7 output 5. The number of rows is the same, and the total size is smaller. Okay? So you see, it is almost 94. So smaller. The original size was 340 now, with deflate, arrow and deflate 94 size. So you see more. You shrink or you downsize best your performance you are. You're reducing the size so best will be your performance. Okay? So now here we can just explore on Hdfs. Hdfs hyphen Ls in recursive show the snappy file. Okay, show the deflate files. You can show the summary here. We have 7 files, so I can use hyphen du hyphen S. To create a summary to summarize. So here I can run this on Hdfs and it will say 140 MB. Okay? And the size on Hdfs. For the deflate is almost 90. So from 324 MB, it was. Downsize it to 90 MB. So almost we keep. We save it almost 60%, 60, almost 60%. Now, to show the information from the Avro. So now the Avro, I can now share this file with you and you can say, create external table and hive. Store it as Avro, and you can read it. No need to populate wherever because it is already populated. But when you create a new empty, you need to populate it explicitly. But when it is already populated. you don't need to repeat this operation. Okay, so now let's run and get the Avro data. Oh, I have. Here. let I will check this after. And this need there is here may be okay. I know there is a because I updated this arrow tool in in this operation. And this is not the right Java version. I I will, I will provide you the fix how to do it. Okay. So here, we will not be able to run this. Okay, because our we have here a version. Yeah, I will provide you the fix. How we can read fix from from. But again, here we can extract the schema. You can see the schema from the Avro container. There is different way to do it. This is only the once we need to do it. From using this, we'll be doing this from different tools later. Let me see if the parquet tools. Now this is for the Omc. For the parking, for parking are the same, so doing the same. Because I asked in this version of the sandbox, I installed the newer version of these tools. and I didn't check the Java version needed. This is why we got this error. But I will provide you the fix to do this. Okay, so from the parquet, let me run this, and now you can see in the parquet we have the row size. So without any compression. the row size is smaller than the average size. Why? Because parquet do not use the 16 Byte marker. So now it is row storage. No compression, as is it is stored as is okay. Now I can, I will repeat, using the snappy. So snappy is not the with with the snappy. We are not looking to the best compression factor with the snappy. We are looking to balance between performance and size. This is why Snappy will not compress a lot. It will. It will try to balance between speed and size. This is why using or deflate, deflate, use a higher compression ratio. zip or Zealand use a higher compression ratio, because we are looking to make the data smaller to to downsize the original size. Snappy. No, we are looking to balance. So this is why, if you are. if you show an hexadecimal, the content of a snappy. It is still readable. while if you show or show the content of the parquet compressed with the zip, deflate, or zip, it is not readable because we are trying, or we are looking to small, to downsize this, the this actual size. And we are not trying to balance between size and this processing speed. So here I am repeating the the process, using snappy Gz. so in a single shot, I'm doing this. Okay. So now let's take a look to the content, output it. So I have 7 file. Okay, number of role the same. The parquet compression is a snappy and the total size is 129, you see. And in using the zip. so here's your. The size is almost 82 MB, because we are not looking to balance between the size and the processing time. We are looking to reduce the size only. So if you are on the cloud, I understand many of my clients choose to use Gz, when they are in the cloud, because they want to reduce the charging the billing. And but yeah, when you have a lot of lot of data it make, it can make a difference. So here showed, the total size is 129, and originally it was 3, 340, and 1, 47, or 45 in Avro. So here is smaller than Avro. If you use Gz, it is smaller. 82 MB. Okay, so this is to show the content. So you can see here on the parquet the size row 310 MB, and this is snappy. This is for 3 g. Zip, and the size for snappy. 123 MB, and the size for the gzip, which is 78. Megabyte. Great! I don't know. I think. I wish I would get the same error here. Oh, no, okay. It is only for the hour. Okay, so here for the parquet, as you can see, let me do this little bigger. so you can see the the schema. So here we have the schema. the project. This is column, name and column type. So project, code, page, name, page, view bytes. You can notice that this is exactly the same column name when I created my hive table. Okay, this is optional. So this means it support it support null operation binary string. And this is the type. So column, name, string, type, string and support new. Now we can see. Here it is. We have more information. We have the snappy, so we can read, or we can know what is the compression codec used. So I know this is parquet, and we use the snappy to Comp as compression library, and, as you can see, parquet include many filters. As you can see, we have minimum. So the 1st row is starting by Aa, the latest row in this block or latest column in this block has the en number of nulls 0. There is no null. So you see, these are some information added, when Hi, when you run the compute statistics function. So. and in Orc it is a little bit more detailed. But here in parquet we can see. This is what we call the bloom filter from here. It is not everything, but this is part of the bloom filter, but what it is important for us when you run your parquet tools to get the Meta from a parquet file already encoded. Parquet file you get the schema. and you can see here it was created by hive. There's a hive schema. Okay, this is our schema. And you can see the compression type. which is in this case snappy. So we can say for the gzip, let's see this, is it? Okay? Exactly the same. But here, instead, the snappy, we get the z-zip, which is our encoding library. Okay, if you want to extract the schema, only just run parquet tool with schema. and it will return the schema. So message it is a hive schema created by how this is the name and the column name and type. You can see it is by a project called the String, and it's support. New. Okay? So the same, you can do the same for this number. That is okay. So so good to know the parquet works fine. So this is only for Avro. Okay, Avratul, I will fix this, and we'll send you the fixing data so, or C, I will just want to show you the same, the same approach, we do the same. So we create a new table from the exist and populating from the existing one. We change different compression algorithm and use snappy. We use Zlib. okay, okay, so here we are using. And now to analyze the table. So here I am analyzing the table over Cuc to compute the statistics. Hive as it is its native tool format. It took one second to do this. Statics, statistics computation. while it took almost 20, second for 21 second on my machine to compute the Csv when when the data was in Csv file format. Okay? So if we show the describe. So here we have the different sizes. Okay. the same. So now all the sizes has been reported here for Overc, with the snappy overc by default with the snappy and with Zenib we can see. So it is almost the same size with for parquet and Gzip almost the same size between deflate and Zip and Gzip snappy. Same again it. We are not looking to downsize the original size. We are looking to balance between size and speed processing speed. So if we want to do. Yeah, let me show this. Okay for Overc, we can show this Meta information in 2 different ways. The 1st one I can use my Orc tool. Okay? Or I can use nothing is here. or I can use my let's say it is an alternative way. I can use hive, but as it is directly use it internally by hive, so I can use my hive here and this, see, hive, oversee, file, dump. just provide the Olc location on Hdfs. And it works, and here it will return all the information. Let me do this little bit bigger. so you can see, compression is the lib the schema. So here we have the stripe, this stored into a different step. And where is my schema? Where is my, so here you can see the bloom filters. Okay, this is the scheme. It is here, the type struct project. So this is the schema. Remember, when we created the schema table, it was converted to the schema oversea schema by hive. and this is how high it converted. But when we create outside hive. We can provide the same schema as Avro and parquet. Okay, we can run also Hadoop jar, oversea tools and provide this. And this will work. Okay. I don't see if it is here already. No, it is. Let me do it for you. We can save, would you, job or see tools? And I think I don't remember. Maybe it is from the oversea. I don't remember exactly. It is maybe Meta. Let me. Maybe I did put it here. I think it is. I don't remember exactly that. Me I will try. I don't remember. It is Meta, or get Meta. I don't remember. or schema. Oh, Meta, okay, it works. You see, we do hadoop, jar, orc tools, Meta, to extract all the metadata, and we got exactly the same result as the hive oversea dump file. We can do this because it is natively the the native format in internal format and hive. and we can use the hatob jar, orc tools for Meta Meta to extract the metadata from this oversea or a default. So if you go to your browser and here file format collapse. So you see, we have here all the experience. We have the parquet with the snappy, with Gz, row over C, with Z. Over C, with snappy or C row the original data and Avro row with using deflate and using snappy. So you can see you can. You can see if you go with Snappy. Let me show you. Let's say head you can see here it we can still be able to read. You see. it is still readable. We still have some Ascii character readable, and always the Avro schema. You can see it is here, on the, on the header Avro schema type recorded name. I will show you how we can extract this from another tool. We will be doing this later, because I need to give you the fix to use this Avro tools. Fix with the Java version. Okay, so this is very important part of our course, because it it is the foundation to understand how this file format works which one to choose. When you are transferring data, when you want to compute or to do some querying processing and proceed almost starting for now from the next class until the end. We almost will be using one of these file format mostly parquet. We'll be using parquet almost. Maybe we have one or 2 more experience, or a workshop with the Csv. Or G. 7, because it is better to understand. But mostly we'll be working with parquet. And this Avro format. So in the next class you have, we have, we are week 7. So you have your second homework, individual homework. So be prepared because this second homework will include everything. almost everything. But again, I would not. Don't want to complexify this for you, but it will be something you we already covered. Nothing special here. You will be expected to build end to end solution. So almost everything we covered. So we got data from the local system put on Hdfs. This is the part, the common part. I don't want to change it, to keep it as simple as is. and do some transformation with the data and create maybe 3, or 5, 3, or 4 query and create that. So this will be your next homework, and it will be available after the second, or after the class as usual. So this is everything for today. Do you have any question before? You go sleep? Yes, I think. King Ohene-Mante: The homework will be available after today's class or next week's class. Khaled El Tannir, M: After the next class. Excellent. King Ohene-Mante: Oh, okay, thank you. Khaled El Tannir, M: There we go. Any other question. Okay, thank you. So for us very much for attending this class and see you the next class, and I will send you the fix for the Avro tools.