Khaled El Tannir, M: Resume. Thank you. Sam Wanis: Yeah, thank you. I'm sorry for that. Khaled El Tannir, M: No, no problem. Okay, so let me, for example. let's do it here because it is more readable. Okay? So when we run, for example. Hdfs. Dfs hyphen ls, so this will run a list command. So running the list command like this is different. Then running the Hdfs Dfs hyphen Ls. and this command, these 2 commands are different. Okay, this is from Linux. So, Linux, you are running under an account. You are using an account and accessing all the resources. So the 1st thing here, when I run Hdfs Dfs hyphen ls, I didn't provide the entry point from where we should start looking to to list the content. If I didn't provide the entry point. Remember, Hdfs need entry point. So if you didn't provide the entry point, it will look into current directory of the current user. The current connected user when you are using Zeppelin is Zeppelin. So if you run this command. so this is the 1st time I'm running. So it should take maybe 2 or 3 seconds more than this. Okay, you see, it is listing something so and remember in I don't know if you already know. But if all the Linux files when you start with dot, this means this is a hidden file. but here we can see even it is hidden. It is listed. Okay, so these are temporary file. Just for I would say, Meta, file to, to, to indicate, to, to the, to fling which we are not using and spark staging. This steps is completed. Okay, so, but where can we find this? So now let's go back to our browser and in our browser, if you go to the user. this is the directory of users on Hdfs. So here you have training and Zeppelin. So if you go to Zeppelin you should see the same. So running the command, Hdfs, Dfs hyphen Ls. Without any entry point. It will list the content of the current directory of the current user. So the current user is Zeppelin. So it is listing the Zeppelin Command Directory. Now, if I provide the entry point. So this mean? I want to read from the root. from the main point. So here I am getting. So if I go back to the root. so it is listing what whatever we see on the root from Gss space. So we can see all these directories apps each base solar whatever. So here we are listing this entry point from the entry points you can see all these are directories. Now let's say I want to upload a file. So let's say, Hdfs, Dfs hyphen put. So the put is reading a file or a group of file or a directory from the local system. The local system is on your virtual machine. It is not on your host machine. the local system. So in Linux we need to start from the root to indicate a path. So home training. Because when you connect using Ssh, you are connected under the training account. So training data. And here you have all the file you can use. So let's say an example. I want to take this one. Okay. But now where I should put it on my destination, on my target, so I can say, for example, let's say under user. And Zeppelin, okay. So now it will read from local. And now, if I again, if I rerun my Ls without any entry point. So you can see now the input file. The file I uploaded from my local system to my Zeppelin Directory is now on Hdfs. So this is very important to understand the difference. Now, if I want to get out Hdfs. so here, where maybe I can be facing the access denied, or permission issue. And this is true for all the hadoop, or whatever you are running under Linux, when you are, whenever you perform a right operation. or execute operation. The account under you are using to execute or to perform this operation should should have this access execution, for example, or write, or or read, and whatever. So let's say, now, I want to do Hdfs, Dfs hyphen get get means. I read from Hdfs. And I write out to local system. Okay, where is my input directory, my input directory? Let's say it is here. What is the name of the file? This name of the file is here and now I want to downloaded to my local machines. So I will say. downloads. Okay, I can give a name another name, so I can, or I keep it. If I keep it without anything. it will be copied with the same name. If I want to change the name I just can follow here the new name. Okay. So now let's run this command. And here you may be okay. You can see we get a permission denied which is normal. It is not an error of the sandbox or whatever. No, it is a behavior expected behavior. So why? It is delighted because you are running under the Zeppelin account. and we are trying to write to the home training, to the training. personal or private directory. So here we can. What what I used to do. Or here we can change this permission. We can change this settings to let you are because we are here under learning or practicing. So we can change this permission. But if you are in production environment, the administrator should give you permission or grant you permission to access to the turkey. So the 1st work around, I will say we can take the same command. But instead. putting this into the Home Training Directory, we can try, maybe to put it into them time. We know that temporary directory is here, and it is open. There is no restriction. So this should work. So this command should work. Now what you see, there is no restriction now, what I recommend to change this access or to grant access to the Zeppelin account. and maybe later spark and whatever to our download, for example, download directory. So this will be a workaround. But it is. It is not a problem, because you are just in our sandbox environment, and we are here for learning. But you cannot do this from Zeppelin. because you need to enter a password. and Zeppelin is not able to handle until you enter the password and continue performing the command. So to do this, you need to connect to your To your sandbox, using mobile, external or any Ssh client. So let me connecting this. Okay, you can see it now. Okay, so we have our okay. Okay. So now I am connecting under training. You can see here the account. The current account is training. Okay? So what I can do here I can do pseudo, which is super user do. It's just a kind of per you are running as administrator. This command you can do. Change mode ch mode. Let's say hyphen R. Recursive. The new permission is, oh, so this is for Linux. So 7, 5, 7 means I can grant. Actually, it is not allow, it is read only for others. It will read and write for me, and read only for others. so I will grant right permission for others. So this is given by this code 7, 5, 7, and I would say, starting from home and training, so later, you will not. Okay, it asks you the password. The password is training, and now it is done. So if I go back to my command. get user. And now I rerun this again. This should work, because now we grant the right permission to the Zeppelin account. And now I can run my Ls command, which is Linux command. I do not need Hdfs here, because this is on my local system. So I can just run Ls. for whom? Training and down. Sorry downloads. Okay? And you see, I have my file there. So this didn't go over in the same order. But these are, the say, the summary of the question I got from in in mail, by by email, and another one was about a permission denied no, not permission. That file already exists in the second. So let me get this out. So again, this is normal. You are here. You are learning, which is completely normal. And it is very good to to face this problem. Because let me, okay, so let me this one. Okay, so here. You see, we get when you run this mapreduce application, it will run no problem. So, as you can see, we start by moving or uploading. We create a directory. Okay, so this directory is on Hdfs. So because we are starting, Hdfs make dear mapreduce, I run so I can run the command. Okay. So now it is creating a directory on Hdfs, so I can browse it. Let me refresh this. And if you go to workshop. And now you can see, mapreduce. Okay, whether data it is created and it is empty, which is normal. The second command is the put command. So the put command. We are running the put to upload this file to this location on Hdfs, so source file source path and target. So path. So okay, around this. okay, so if I go and refresh the explorer, I can see now my file. It is uploaded, so it's everything is fine. Now I can run my hadoop jar. So you can see here hadoop jar. Okay, which is the hadoop command to run mapreduce application the jar because I want to run already. Packaged application. So I provide the pass and the full name of this Java Archive. So the Java Archive is Count station, dot jar. and the physical path or location is located in this path. Home training hadoop mapreduce 0 3, and the name of the count station and the name of the sharp. So this is the input where the application is packaged and physically stored. Now, inside this package, I want to run a particular application because one package can handle more than one application. You can package more than one application in a single physical archive. So the name of this application I want to run is Count stations. Okay? And now, this application needs input data and output directory. So the input data can be found or the input directory can be found here. So this is the location workshop mapreduce weather. So this is the one we just created. Because mapreduce in hadoop reads only data from Hdfs cannot read data from local system. Okay? And now it will output the result to this location. Okay, this is the output tongue station. Also, let's run this. So it will take. Maybe I don't know. Okay, so we have also here a permission denied, you see. But this is normal. This is normal. So here it is in the hadoop node, so we can grant this to to this no, that. Give me so once, but this is not on this is on the Hdfs side. It is not on the local systems. Of course, the map reduce here it is writing the intermediate result to this temporary location. Okay? So just here again, we need to to grant this access. But the the command is a little bit different. Just give me one second, and I will show it to you where it did this. Okay. okay, so I should be somewhere here. where it is, where it is, where it is where it is, where it is would be okay. Just give me one second what it is. because the command is a little bit different than the one. It is almost the same work it easily while changing the account. Yeah, you can to it depends. Because, I am working. Yeah, maybe you didn't get this error because the one the sandbox you had we? I tried to provide all. I will grant all the access, but as I work differently with the one is installed on my machine. Maybe it is a little bit different. but you should not get getting this error from your side. but I wanted to show you, because this might happen. But and the command is a little bit different. So I will just show you a little bit where it should be somewhere here. Yeah, this is okay, this is, okay. Okay. So let's go back to our what it is is here. Okay, that's me. speaker. And again, so here, what we are doing again we need to run under. You cannot run this command from Zeppelin because it needs, maybe to enter a password. So here we already entered the password. So it's, I think, should not ask again. So pseudo super user do hyphen. You use the Hdfs account because the Hdfs account is the Admin for the Hdfs space and use or run this command Hdfs Dfs ch. Mode, the same as we use it to change the or to grant access in our local system. But this time on Hdfs recursive everything inside, and we provide the All provide all, grant all permission. So this should run. Okay. And now, if I rerun this again. where it is, it is here. And if a rerun this again should works fine. Okay? So it is running. Okay, okay? So now it is running, and we can see running while it is doing all the computation. So now it is finished it. And now I can go to the output and I run the output. Okay, I can list the content of the output. So I have my output file, which is fine. Now, if I rerun this again. if I rerun this again now you will be facing a problem. You see, it is not executing it is. Get you get an error. So this error, said the output directory. You see, output Directory already exist, which is normal because the hadoop tools hadoop. Spark, hive, whatever. They will not overwrite anything, any existing directory, or any existing file. It is up to the user or the administrator to set this implicitly sorry explicitly, so you should say overwrite, you should say, delete, replace whatever. But the user should do this explicitly. So now how can I turn around or get around this issue just I can change the output directory. You can give a new name or just just change the name. So here, now, if I rerun this again. This should run without any problem. You see, this is right. So this is very common situation, and this is how you can solve them this answer all your questions. Just no. Are you here? Yes, okay. Great. Okay. Great. Okay. So now, this was what we covered the last week. And and now we'll be moving into a new class. Oh, okay. I wanted also to show you something else. because I got also a question. Just give me one second, please. Okay. So here you can see the the result of the survey you you just entered for your development skills. So here you can see. So I can see that you have. You have, or you already a little bit familiar with programming language. maybe most of you, maybe python and SQL, so which is fine, so SQL. Will be very helpful for you and the starting from the next class, starting from the next class, we'll start today a little bit with the spark sequel. But starting from the next class, we have 2 dedicated classes for querying structured and unstructured data at scale. So your knowledge and skills and experience in SQL. Will be very helpful for you, because it is almost about querying. But we need to query in a different, not a different way. But we need to think differently. Okay. And as you already know, most of you know, python, which is good. So you have already the how perform the algorithm how to write a little bit the language. But again, here in our class, we will not be using python at all. We will be using Scala, and starting from today, we will be covering a little bit. Of course, in the live class I cannot cover everything. This is why, as you are not. This is something new for you. You have this tutorial, I provide you, and you need to work a little bit, maybe at home to just be more familiar with the language and create because all the code you will be writing. Not very complex code, maybe 5 or 6 line of code, something like that. But we need to to do a little bit coding, but very, very simple. And again, as now, I know a little bit about the profile, about the skills. I can adapt the 1st homework because your 1st homework is starting from next class the next week. we week 4 the next week, and it is about Spark and Scala. Okay, so here I have one question, what it is. Okay. So the question here or the comment here, I would like to receive a list of most important skills. Okay, so remember the our. I organized the course in 3 part, the 1st part, we have 2 classes about hadoop and spark. In this part, we'll focus mainly on how we can move or manipulate data from local system to our distribute storage system. So the Hdfs and so on. So here mainly, it is more close to the Linux command line. So here we need to focus on this command. Linux, command, understand the structure, the syntax, and so on. Starting from, I would say, after the break today, by before the end of the class and the next classes. you will be analyzing data. So you need to prepare your data, you need to clean your data. You need to organize your data. So we'll be doing this using different tools spark mainly, and others tool, such as Hive and Trino dedicated to querying. So spark is about Scala, so Spark and Scala. So you need here to focus on the Scala language until the end of the course and focus on my sequel. Not my sequel. Sorry on SQL. Language, because it is the fundamentals of our data analysis will be using SQL, not dedicated to my sequel. But we'll be using SQL, okay, and later on, after this, starting from week 7, we will be changing a little bit, so we'll be doing more graphically. Build your pipeline, and in week 9, if I'm not wrong. Week, 10 week we go back to spark and processing real time processing with spark streaming. So to answer this question, now, this part. Focus a little bit on Linux command, basic Linux command. Hdfs command the Scala language and my and SQL. Language later on, maybe a little bit Cassandra, because it is a SQL. But dedicated to Cassandra. And this is what you need to focus on. So every part has the the the languages and tool to focus to focus on. Is this answer your question. Yes, no. Okay, okay. Great. Thank you. Okay, great. So now let's move this back and and see about 2. Okay. let me stop sharing this and share not this one. But I want to share why, it is sharing here. You want to share this one. Why, it is not sharing the the right screen. It is not sharing the right screen. Okay, never mind. Let's do this better and go here. Okay. now it can. You can see the screen share the screen. I don't know why it is not sharing it, because I have more than one screen. Okay. it is not. Oh, oh, the technology. No, no, it's not. It is not changing. Bizarre. Okay, never mind. Now, can you see this? Okay, no, it is not on the screen. Hey, woof! And now. Okay, never mind, I will. I will show why this it is doing this. Hammed Akande: Karen, and share again. I think that should serve it. Khaled El Tannir, M: Sorry. Hammed Akande: You can stop sharing, and then you share again. Then you should. Khaled El Tannir, M: Okay, let me try this. And now no, because it is taking to another. don't know why it is not showing on the on on the right screen. I have. It is showing the I have 3 screens. It is not showing on the one I want to to do the okay, no problem. Let me. Hammed Akande: Maybe plug that one to the right screen. Khaled El Tannir, M: Sorry. Hammed Akande: You can drag the court, the one showing on your side to the right screen. Now. Khaled El Tannir, M: I'm trying to do this, but it isn't working. I don't know why. Okay, I don't want to waste my time with this. I will check this later. Just, I want to be sure that it is. And now, okay, you can see this here. Okay, no problem. So let's be let let's move this. Okay, okay, okay, okay, okay, okay. So we will be focusing on the spar part. And of course. spark. Here is our tool to process a large data set. And in memory, you will understand what I am saying when I say in in memory. okay, remember, in big data or data at scale, we focus to solve 2 problems. The 1st is the storage, the second is the processing hadoop provide 3 main component. The 1st component is the distributed file system to store at scale data at scale. The second component is to process data at scale mapreduce, and the 3rd component is to manage the cluster resources, which is yar. It is not covered by the course, but it will be manipulating. And directly. Yeah, I will show you this today how we can just monitor and get an idea about what it is allocated, and so on. And now spark is a tool which is not part of the hadoop ecosystem. It is a completely separated ecosystem, but we can, I would say, not to say mix, but we can use the Hadoop Hdfs layer and hadoop comp cluster manager layer with to to make spark work and run, so we can mix these 2 ecosystem in a single platform. And here it is. The result is a very powerful platform. So spark is independent than Hadoop. But we can make this work together because spark use and needs to use all these components provided by the hadoop ecosystem. So now, until now, the last class, what we did just we, you learn it to work or manage or collect files from your local system and put them on the Hdfs in Hdfs. So this is the collect phase. So today, with the spark, we introduce a new stage in our data pipe line model. This new stage is about how to organize your data. You can filter, you can clean, you can sort, you can aggregate, do aggregation, you can partition, you can do whatever you want. So now, this is the second main stage or phase in our in our pipe pipeline. Okay? So later you can. I would say, prepare your data using spark, and later it will be ready to be analyzed and create dashboard, using Zeppelin or sharp using Zeppelin or dashboard using superset. Okay? So what is Apache spark spark is again, is a tool, independent ecosystem. It is in memory analytics. Open source framework. It runs 10 times faster than on disk than hadoop, and 100 times faster in memory. Why? Because everything is in memory, and when everything is in memory memory process. Everything in memory is faster than process on disk, because disk you have to read and write operation. You need to load from disk, and this mechanical stuff are not fast as the memory components, so memory are extremely fast. So this is why the 1st factor why spark is faster than hadoop. It is highly compatible with hadoop storage. Api Hdfs spark can connect to any any data source, and as it can connect to any data, source. It is highly compatible with the hadoop storage layer, which is Hdfs. So it was designed also to run. execute multi staging and complex. Remember what I said in the last class regarding hadoop mapreduce, you cannot run perceptron or recursive algorithm or recursive application in mapreduce, because when you run this recursive application or perceptron, whatever the name node or the hadoop orchestrator will think that this task is blocked, or is freezing whatever so, and it will disconnected, or it will relocate it, or it will re rerun it. So this is why, when you run recursive application in mapreduce application, so you could perceptron or recursive loop or function in a mapreduce application. This will not work, and there is dedicated library for this, which is my How in Hadoop. But spark was designed to run this multi stages and complex recursive iteration and application. So the developer also can write the application spark application, using 4 different languages, can write application in Java, in Scala and Python, and in R. So Java, Scala, Python, and R. We will be using Scala. I will explain why so initially spark itself was written in Scala, okay? And it started under Berkeley today. The company behind Spark is data bricks. Mate Zaharia, the the guy who developed spark which in was in doctorate thesis. And now spark is handled by databricks and databricks is the company who which maintains spark provide us all the new features, and so on. So spark provide for model the spark core itself. Of course it provide the spark sequel, module. the spark streaming model, the spark ml, model and spark graph frame model. So the names are a little bit different from the one it was introduced to the market and act the actual naming convention in spark. So spark, core, and spark sequel are the same spark and spark. SQL. This, the name of this model didn't didn't change. Spark streaming doesn't change. Now the Spark, Ml. Lib. Was the 1st name and the actual name the 1st name with the 1st version in Spark and Graphx. and the actual name are spark, Ml. And spark graph freight. Why? Because the 1st or earliest or legacy version of spark use a different stricter in memory to handle data than the current version. I will talk about this in more in detail in a moment. So just to let you understand. the Ml. Lib. Is based on the spark Rdds, which is the structure of data in memory. And the actual version is based on data frame little bit, different structure in memory. the graph X, the same. It was based on the Rdds, which is the legacy structure of data in memory. And the current version is based on the graph frame. So this is why the name is graph frame. I would take about this in more in detail in a moment. So if you go to the Internet and check whatever documentation. And you see, Mlb. And graphics. this is very old documentation. You don't look at that. You need to to look at more recent versions. So why do I need spark now? If if we go back to our exercise or a tutorial in Mapreduce, you can let me. It is still open, I think should be open. So let me show you. So you will understand better. Okay, so here, if you remember. Do you see the same here. Yes, you can see. Let me do this little bigger. Oh, too much bigger. Okay, so here. as you can see. we read the data from okay, let me just finish it. Okay? So we can read the data from Hdfs and then submit to the cluster. And then we perform all the maps, and until the end of all the mappers, then we start our reduce phase. Once the reduce is complete, the job is completed successfully, for in this example, so we start by reading data, and we do the 1st iteration we perform the aggregation, and the output is stored on Hdfs. If I have more than one stage, or I have a lot of data. I need to do this by waves or by iteration. This is why I read from Hdfs. I perform the processing in memory and go back to store the result on Hdfs. The next iteration do the same. Read from Hdfs. Perform and generate intermediate data and aggregation on Hdfs. So reading and writing operation is time consuming. Read the operation from Hdfs. Write operation on Hdfs is time consuming. So this is why we call hadoop mapreduce, processing on disk processing, while if we remove this operation, write, read, write, read, write, so we'll we'll gain. and we will enhance our processing, so we will reduce the processing time. And this is exactly what spark is doing. So Spark say, okay, you need to process your data. Let's read this from the input, source. It's not necessarily only Hdfs can be any source. So let's read this from the input, the data from the the source load everything in memory. If we need to perform more than one iteration. We read this from memory. We don't want to go back to disk. to write the intermediate data to disk and rerun or reread from the disk again. So everything is stored in memory. So this is why we re spark, inspire all this operation, read and write will be removed. So we have better performance. And you can see on the illustration this logistic regression benchmark. This is a real benchmark. This is not. This is what the benchmark provided by spark when it was launched in 2,02020, yeah, 29. If I'm not wrong, 2,009 if I'm not wrong. And in this performance it was yeah. The hadoop cluster was 2 2,100 nodes and the spark cluster, where, if I'm not wrong around 200. So 10, 10 times less nodes to perform the same the same job, and it was faster, almost 10 times. I think I provided you the real benchmark by the end of the annex. Okay, so spark handle data in memory in a completely different way. This is a completely different approach, completely different algorithm to process your data in memory. And this is here you can see the these 2 approach Mapreduce, read from Hdfs, distribute the function to perform on the no, on different physical nodes, then run, shuffle, and sort from this different node, to collect the intermediate data and do the aggregation to provide the final result. While spark do this completely different in a different way. This is different. Approach it, connect to the source. load the data in memory create whatever we need structure whatever we need. All the metadata needed everything still in memory until even it is partitioning, because it is in distributed manner until you as a user or developer, decide to persist the data. So the data will remain in memory until you decide. So as you can see, this is a completely different approach to solve how to process data faster, of course, and improve the performance. And this is what we want, because we we here, we are dealing with very, very large data set. So we should do this very, very fast or as fast as possible. So now the spark ecosystem again, spark can connect to any data source. So this is an overview about the architecture on the bottom. you can see the data sources. So of course, this is not exhaustive. We can have any you can connect to any data. But this, this, an example of the most common data source you can connect to some of these data source are already embedded, others you need to provide the the library to be able to connect to this data source. But there is no restriction in this part. So on the bottom, you can see the data source you can connect to Hdfs. You cannot connect to Cassandra. You will learn how to do it with Cassandra. Week 11 Hive. This is a hive is the topic of the next week, each base. Maybe I will show you an example. Later, as you can see, you can read from Csv file, you can read from Json, file from parquet, from from whatever. So there is no restriction. If you have already the data source embedded, you can connect directly. If it is not already embedded. You need to provide the package, and it is all on top of the data source we have the spark core. The spark core is the main libraries of the spark itself. On top of the spark core, you can see the Rdd Api. So this is the Api to access the structure of the data in memory. Again. spark. Use a dedicated structure in memory, to handle the data, to handle failure, to to handle partition, to handle dependencies, everything. everything is structured, and this structure in memory to handle the data and metadata, and everything is called Rdds. I will explain this more in detail. Just wait a minute and to access and manipulate this structure in memory spark. Provide the Rdd. Api. No. On top of the Rd. Api. We have the languages. Api, so you can access this Rdd. Api from any language from any language supported by spark, so it can be Scala, Java, Python, or R. So any of this language can be can handle or can access. The Rdd Api. With the spark core. Remember, we have 4 module provided by the spark ecosystem spark sequel, spark streaming Ml. And graphics, and all this module axis have access to the Rdd. Api. So this was the 1st version is sued by spark data. Bricks does did not exist at the moment. So just spark later, 3, maybe in 2011 or 2013, something like that. They introduce it, the new spark, SQL. Api. So they introduced the data frame, the sequel and the data set. Apis, 3 Apis data frame Api, SQL Api and data set. Api. So this is, if you have, as you are already familiar with Python, you for sure you are familiar with the python data frame pandas data frame. So the spark data frame spark data set are inspired from the Python pandas data frame. But here it was designed to support a very, very large data set and 1 billion of rows, while the Pandas data frame has some limitation. When you load very large data set. no data frame was introduced in 2011 or 2013, something like that data frames and SQL. Api in 2015 was introducing the data set. Api. So the data set Abi is the most efficient way to structure your data and handle your data in spark. It is secure, it is compiled. So there is no no error possible. I will explain you why so? But the data set Api is not accessible from the dynamic language, dynamic language, Python and R. There is no, it is not accessible from data set Api as not is not accessible. You can access the data set Api only from Java or Scala. while the data frame Api and the SQL. Api. These 2 Apis are accessible from all these languages. Java, Scala, Python and R. This is why you will. You can find, maybe more often people will be using the data frame, then represent the data sets because it is most common. But again, internally. spark. Use this data set. Api, it does not use the data frame Api. Now, on top of this data, set data frame and SQL, Api, you can connect any application you you you want. So you can connect hive. You can connect pig. You can connect each to those. This is for deep learning and and machine learning, and so on. You can connect Zeppelin. That we will be doing. Scoop is I remove this pig and scoop, I remove them from the Course pig is provided to process unstructured language and structured data, but we can do the same with the spark. This is why I removed pig from the course scope is, use it to import and export data from the hadoop. and we will be using another tool more efficient than than it is, which is nice than our scope. So this is why I removed this from the course. Now to run spark, run spark needs. As you can see, spark does not provide the storage layer. So it needs a storage layer and it needs an environment execution. Environment so initially, spark was create. When spark was created it was created to run. and on the using the mesos framework misos is very similar. Yarn is very similar to missus, which is a cluster manager, so it can handle the memory. CPU. Wherever everything is orchestrated, and yarn is very, very similar to to missus and yarn spark is compatible with yarn and Hdfs. So we can connect. So we need execution environment to execute spark. This can be docker if it is a standalone, can run this on Amazon, using Ec. 2 mesos openstack. If you are running in Kubernetes whatever. So this is the spark environment. Do not hesitate. If you have any question, do not hesitate. Now, just to very, we'll go more in detail. So the spark sequel. the spark sequel again, is the model that let you run SQL language. So SQL. Language is here to run. For example, select star or select column name from my table. So this is the structure. So spark can run your command and support this SQL. Language, and also support the hive. Query language. So the hive query, language is very similar to SQL. It is a subsystem sub sequel. It's not completely SQL. It is subsystem of the SQL. Language, and add some new dedicated or case or operation dedicated to to hive. So spark, SQL. Support the hive, query, language, and other. So we have. After the break. We can talk more in details about the spark sequel, which is part of the course spark streaming we have, as the name suggests, it is dedicated to collect data in real time or in near real time we'll explain this, and we have a dedicated class to to manipulate and to learn how to create application, using spark streaming to ingest data in real time. So we have dedicated class for this. Again, the 1st version use Rdds, but we will not use Rdd anymore. It is very complex. And you really really need to be a kind of developer, I'd say, to go inside. And it is a little bit complicated. So we'll be using the data frame. And again, we have the data class for this. So spark, graph frame. It is not part of the course, neither, nor the spark Mlb. But and the latest class. I will explain a little bit to you how you can use this graph with examples and some examples, and the spark Ml. Lib. How to use how to create a pipe, how to create prediction. If you are a bit curious, you can. It is provided as 2 different tutorials in the sandbox. You can go over, and in the latest class I will explain more in detail these 2 tutorials. Now, just to give you the comparison between hadoop and spark, and just a summary. So the hadoop core the spark core, we have the same. So I do provide mapreduce to process data spark, spark or the spark itself. It is here to process your data in memory. Hadoop provide yarn while Spark does not provide any cluster manager, but it use missus initially, but it can use. Any other cluster manager can use. Docker can use kubernetes eks whatever, but it should be compatible with the same yarn, and it is, we say, cluster manager. the hadoop provide the storage layer. Hdfs. Initially, spark was created to run, and provided the Tachyon interface. The Tachyon interface is just an abstraction layer to let you connect to any data source later the chain, the name, change it, and now it is a Lucio. So Aluccio will operate as an abstraction layer to connect to another data source. And as a cache. so it can improve the performance. If you have a very large data set in production, I I used to to use it in in different situation. Not all the situation, but it can improve the the performance, because we it use the data, it create a space in memory as a cache, so it will improve the the performance. The hadoop provide an external tool hive to query your data. Now we are analyzing, we are in the application level while spark provides spark sequel. Hadoop, provide the mahout library as machine learning library spark, provide Ml, spark. Ml. Machine learning storm is the legacy tool to process data in real time in hadoop ecosystem. Now, storm is a little bit old. We say old. It was developed by Twitter. And now we have a new tool called Apache Heron. Apache hero is a real time to process your data and collect and process data in real time the spark provide the spark streaming. But spark streaming does not process your data. In real time. It process data. In near real time we will see the difference in the dedicated class. But to give you an idea, it is about deadline and timing. If you are, you need to process your data inside dedicated data frame, and it is titled to the time titled to a deadline. It is near real time. So these are the difference different component between hadoop and spark. So the question now do. Still, I need hadoop. The question is, yes, I still need hadoop. Why? Because Hdfs and yar Hdfs is needed as a storage layer, as yar does not provide this, and yarn is needed to to run spark. As it is not, it doesn't provide any cluster manager. Again, we are. If you download the package spark from the spark website and you run this locally, it is not running in a distributed manner. It is running in standalone. and in this case you do not need any distribution or any distributed file system or any distributed memory. This is why, just pay attention to this and the sandbox. Your spark is running it in a distributed manner, and it is running over yarn. It is running. It is not running in standalone mode. so if we stop the yarn or the hadoop, your spark in the sandbox will not run because it rely on the yarn component to get to allocate memory, to allocate all the resources needed to execute the spark application. Also in the hadoop ecosystem we can find many other tools not provided by spark. So you can find, for example, Ni-fi hbase Kafka. All these are database, for example, very high velocity, database. Hbase is a very, very high velocity is dedicated to collect data at very high speed. Ni-fi is here to create your data pipe. You will be using this starting from week 7. So we have more than only spark sequels. Spark, Ml. Spark, graph frame. Yes, these components are very useful, but it is not enough. and or in a company or in a real world environment. also in different situation. You can connect spark to Amazon, a 3 to to. You can connect this to a Microsoft azure. You can use different storage layer, and you can change. So it is compatible. based on the configuration you have on premise. If you are using, missus, you can use spark if you are using in the cloud, for example, on s. 3 on Amazon, you can use spark to run your process your data. So it is versatile. And so it depends. Again, it depends on the situation. I need to provide storage layer, and I need to provide a cluster management if I need to run this in a distributed environment. Now, how spark works the spark? Architecture is very simple. So again, it is a kind of master slave. But here the name is a little bit different. The name or the terminology is different. So here we talk about driver and worker. So the driver is your entry point. Without the driver you cannot access to or run any spark application, and you will not get any result. So your entry point is the driver. So the driver is little bit. We say more for the developers, the terminology. So we users use the spark context. So we need. as far context to be able to run and get the result from spark application. So your spark context is your entry point. Now, the spark context. When you submit your application or you write your code and you are waiting to get there is evaluating and get the result. The driver will interact with the cluster manager and the cluster manager will allocate resources. And these resources are assigned to workers. So you you will interact only as a user. You will interact only with the spark context and the spark context will take in charge everything, and it will communicate to the with the cluster manager and get the resources to to allocated to your application, and then it will trigger the execution, for in the worker and wait to get back the result. so the cluster measure again can be. You can run this and standalone if you don't want a distributed manner. But we need, for example, yarn or Kubernetes docker wherever or or missiles, of course. So this is just a summary. The master connect to the cluster manager to allocate resources, the acquire the executor and the executor get an application Id or application code, and then the tasks are triggered. So this mean, if you submit an application, and the driver did not got, did not get from the cluster manager the resources. Your application is pending this impending model. You still wait until you get the resource from the cluster. Okay, so this is why it is very important to ensure, when you run your spark application to have enough memory in your cluster and stand alone. You do not have this problem. When you are running this in a distributed environment on a cluster. you need to ensure that your cluster has enough memory or enough resources to allocate your application. Otherwise you're still in pending mode pending until we got the resources. Okay? So now, this 1st structure created by spark to handle data in memory was the Rdd so Rdd. Stand for resilient distributed data set so resilient because able to compute missing or damaged partition due to node failure. Remember, mapreduce is fault. Tolerant. Hdfs is fault tolerant in case of failure. You do not lose your job. and if on Hdfs you have, you still have access to your data, even in node failure in spark Rds was designed to be fault tolerant. But this fault tolerant mechanism is different than Hadoop and the Rdd. In the metadata of the Rdd. We create the execution tree. We create the logical plan of execution in case of failure. We just read again this execution tree, and we can recompute everything. So no data lose and no need to re rarely reaccess the data from the source. Really, rarely it can. It can happen, but really rarely so distributed because it is distributed over your cluster over different physical nodes, but in minimally, not on disk and data set. This is the collection of partitioned data. So you can see in the illustration how it is structured. So you have the Rdg. It is a collection of partition. The partition here. If, for example, remember, each machine has limited physical data, sorry physical memory space, then you can allocate. If your data does not fit into this space, you need to partition. This is where the management of partitioning, Rdd, partitioning come in handle. So partition is here to partition your. You can also decide even it is on single machine. You can decide to partition your data, your your data again, you can now allocate more than one core per partition to perform to to enhance performance. But again, this is more for administrator, not for us. We stay at the data engineer level. We don't go to the infrastructure level. So you have all the compute function. So every, all your, the code you write to do computation, to to call your, to clean, to, to sort, to aggregate your data. And we have different metadata information about creation, about access or whatever. So this is the structure of the Rdd. now, this is just high say, high level, how using the Rdd can be distributed over different physical machines or different physical nodes. And how can we transfer data in parallel? Okay? Now, this is the Rdg. how can we create an Rdd spark can create automatically an Rdd for you form a fart. When you read a file you read a Json file. You read a Csv file. Wherever you need to read, you perform this read operation. Spark will convert this while reading to an Rdd. And the output of the read function is an Rdd. so create from file. No, you, you have data in memory. For example, it is a list of color, red, blue, white. This is just a collection in memory. You can convert or create an Rdd. From this list of, or sequence of element. So from a file you can create an identity is created for you direct directly, automatically from data in memory. you have a sequence of element, an array. If you if you want or a list. and you can convert or you can create an Rdd. From this list. The 3rd way to create an Rdd is transformation. The Rdds are immutable. You, once you load data into an Rdd, you cannot change, you cannot modify, you cannot Update. So any transformation will create a new Rv, for example, even filter. I want, for example, I load data into my Rdd, I want to filter my Rdd filtering. The filter operation will create a new rd. so here we are creating the tree transformation, the execution, all the steps you perform it to clean, to prepare to to do whatever to your data stored in the Rdd. And after each operation or each transformation it will generate a new Rdd. this is how spark in case of failure, just need to recompute, and it got everything. So it is a different way than hadoop no. A spark. Also provide a particular type of arteries which we call the Rdg of pairs. This is very special. Are the very special data representation. This is only to be compatible with the mapreduce algorithm, remember. and when we covered Mapreduce in the last class. I just told you this important to understand how it works, because we will be using this from spark, and remember that everything processed by hadoop internally, and the output is always in the K value format. It's it's a tuple of K value. The input should be K value and mapreduce will output a K value pair always so. The Rbd of pairs is provided by spark to be compatible with the mapreduce algorithm. You will not run the Hadoop mapreduce application, you will run. You will execute a map and reduce function in spark. But again, this is a way. For example, if you want to count or to perform or to filter. If you want to filter an Rdd or data, you have data, you have lines in the text, and you will. You want to filter what you can do? Just run a map function. The map function will do as will perform as an iterator. so it will do or perform the same function over all the items. This is how map function function works. So it will perform this computation on each element. It will, so it will behave as a loop or as an iteration, so to be compatible. Spark. Provide the Rdd of pairs, as you can see as a tuple of 2 elements. and in the illustration you can see this is a word, Count. You can see, we have the word. which is the K, followed by the value which is the count. This is just basic example, and the Rdd of a string, which is, for example, this is, if you read a text file from your data source and it is loaded into an Rdd. let's say, here we have one character or one word per per item in the Rdd. you can have different format. Different. It depends on how we load our data into this. Rdd, but this is in this example. In this illustration you have one word per item in the Rdd. And as you can see, it is distributed, so the the collection of items of the Rdd of pair is distributed over physical machines. So this is how it looks like. okay. Now, the in spark, the user or developer can perform only 3 type of action. 3 operation to be more accurate it. The user can do transformation. The user can do action and the user can do persistence. No transformation. This mean, you can. Here you can do. For example, filter map. You can do all this computation. So we are not doing this, the aggregation. We are not doing aggregation here we are doing everything, for example, filtering, doing this tank doing union. We are so. All this operation or all this transformation, will generate a new output. Rdd. the Rdd itself is immutable, cannot be modified. but everything still lazy. Nothing is executed. So you can. For example, you can say, read text, read my input, file. The command is executed. You see, the compiler accepted it, but behind the scene the file is not loaded, but Spark created the 1st step in the computation tree. Now you say, okay, I want to do a filter. Okay, I want to remove all this unwanted words, or whatever. Yes, it is. You see that the compiler accepted this operation or this function. but the result is not yet here, because nothing is executed but spark is building the execution tree. It is only executed when you perform an action only when you perform an action. Now the execution is triggered. For example, you say, read text, or read, file from you, provide a path and provide a filing. You will not see here at this moment if any typo and the file name or the path execution correct. When you run the action. For example, you say, give me the count of the element in this file. Give me show me only the 1st 3 or 5 lines. Now you trigger the execution, and in case, for example, here, if there is a typo in the path or the file name here, it will be throw, raise it, the error will be raised, not before. So with the spark. When you execute your code, you will not see any error. If it is a syntax error, it is different, but it is not executed. It is analyzed for the syntax, but it is not executed until you perform an action? No. The 3rd type of operation is persistence. You can persist data in memory. You can persist data on disk. You can persist data in a cache. for example, we are doing a word, count. So for the word count, you load your file. you run or perform the map and reduce application, and you want to output the result, to disk. So at this moment you decide to persist the result of the word count to the disk. So you say, okay, for example, save as a text file. and you per you specify the location can be on local system, can be on Sdfs can be in a in a database wherever. So you decide where to persist. You can decide to persist in cache, in memory cache. so you can persist to create a structure in memory. you can decide to persist in a cache memory cache or on disk. So this is how you can output the result of your computation. Basically when you are using the Rdds. So this is just a refresher about the what I said from from the spark framework evolution and 2011 integrated the different function into 2013 introduces the data frame and the data set was introduced in 2015. So the data set type that data set type are. suppose it to be type safe and faster. So now, how can you code? So this is just an example. So spark accept different, styling, coding style. If you are the Python developer, if you are a Java developer, if you are Javascript developer or Scala developer, you can write in different ways and spark support all of these different way. So here we have different style. So, okay, I will not say this is better, or this is not good. It depends how you are more comfortable with use it. So now for the 1st line we can see Rdd filter underscore age greater than 21. This is the Rdd Rdd. Style. So Rdd is the name of our variable, our variable. That contains the data filter is the function I want to perform. and the parentheses we can see we have underscore dot H. Underscore me. This is Scala and this, say, any this mean any element, any item. read any item in the Rdd. And apply the filter function. This underscore means any. So start in the 1st item, enter the last item in the Rd. The age which is a property or a variable in. So in the inside the Rdd. We have a property 8, and should be the value of this column, if you want which would be greater than 21. The second style is the data frame style. SQL, style can say, Okay, my data frame filter data frame name use so here I am, using the double code to provide the expression column name greater than value, age greater than 21 sit will return the same result. But the coding style is different. You can say, I want also to use a different coding size. So here, data frame filter data frame colon ages should be a greater than Gt. Greater than 21. So this is another style. All this style are accepted data set the same. The data. Rdds, data frame and data sets support all these styles. The data sets filter underscore age. So this mean any element. And the data set apply the filter function and compare to 21 and should be less than 21. So this will return. As a result, data set the Rdd filter function, return an Rdd. the data frame filter will return a data frame, filter it. So you see, this is how you can. You are more comfortable with different with one of this style. Maybe we'll be using the data frame sky some sometime. I will be using the expression style. So it depends on the use case depends on on how I want to be sure that it is more readable and more maintainable, because very important. When you come back and you read, you understand, and you not waste your time just to try to understand what is said. So if you are very comfortable with the scala, which is maybe not the case of, or anyone in the company. But maybe use style that let you, if you come back, maybe one month or 2 month or 3 months later to be easily understandable for you. Okay, so this now, just why, SQL and data frame SQL, Api data frame and data set Api how they behave in the the spark, environment and execution environment. So if you run a SQL. SQL. Command, so you are you applying, for example, you say, SQL. Select select column name from my table. So this is SQL. SQL. Syntax, SQL. Statement. So the syntax error are detected only in runtime. and if the syntax is good. But, for example, you have a typo in the table name, or a typo in in a column name. This is detected only when you run the query. So this is at analysis level. So sequel. I will say it is not the best way, because errors can be during the execute analysis stage. So the data frame. Any syntax error are detected while you, when you combine. When you write, when you compile, it is detected syntax error. But again. the SQL. As we are running here we prepare for maybe you you choose a column name. It doesn't exist, or you have a typo or something bad in this. So it is detected only on runtime. Now the data sets while they are more safe because it it is comp syntax, error are detected at compile, time and analysis, error are detected at compile time. So when you compile and you create your package, you are sure 100%. It is error free. No errors when you submit or you create a data or use a data set data sets part. This is a type data type. Okay, so now depends on the use case sometime. It is easier to use the data frame because we don't need to create some structure and so on. But again, sometime data sets are a little bit slower, so it depends on the use case. But ensure that spark ensure us that data set type are very safe and are fast access acceptable speed. Now everything is created in Scala. Okay, it is almost. Let's take 10 min before going to to the Scala language, because we'll talk about this until the end of the class today. It sounds good for you. Yes, no. Are you here? Great? So let's take a moment with the break. and after the break. We'll talk about more, Scala, and we'll continue with spark sequel, because this is our 1st model. Today. It's very simple spark sequel. But we need to to start working with it. or, if you want, we can. No, let me do this. The break now, and after the break we can talk about Scala and the for your 1st Scala code. I will show you how you can write your 1st Scala code. Okay, I will try to create. I will try to fix this. Now. did you have the chance to little bit over this tutorial of Scala, or not at all? Yes, no, no, okay. Rajesh Kamaraj: No, I I, personally, did not get time to go over them. Khaled El Tannir, M: Okay, okay, so okay, so again, in the class, I cannot cover everything. It is very, very large topic. And it is a programming language. So we cannot just. I'm saying, say everything in half an hour. This is why it is very important to go after the class, or within the week to go to to this and take these tutorials, because your next class after the next class, your 1st homework, is to code 6, 5 or 6 line of scala in spark. So you have this. We say the understanding scalar variables is very important. The scala syntax is very important. So this at least these 2 tutorials. If you have time. and you want to go a little bit beyond just the very basic syntax. And what are the variable type? You can go over this string manipulation to to learn a little bit about how to manipulate string in Scala. And if you want really to go one step beyond. Go to this particular insight, regular expression so, but at least at least the 1st one. The 1st tutorial I recommend you is scala syntax, to understand how to write this command or the function, and so on, and understanding understanding the Scala variable type. Okay, so so, Skylar. we will be learning Scala, or doing everything in Scala, because this is the language spark was written in Scala. and Scala was designed and created to support the distributed processing. So Scala is pure object, oriented language. Everything is object, even. There is no primitive, everything is object, even numbers are objects. So Scala combine programming, oriented language and functional language. Do you have an idea about the what is the difference between object oriented programming model and functional oriented programming model, or not at all. for sure. Do you know, Orient object, oriented programming model? Who can tell me what it is? You don't? Yes, sir, go ahead. Sam Wanis: Yeah. So basically, the object oriented programming is mainly for encapsulation or having a classes and objects to represent everything in objects. It has many features to use classes and objects like inheritance, or I mean, as in Java, it's like it's it's basically to represent everything in objects and the classes. And this is provides encapsulation of data and and maybe making it not accessible. Yeah. Khaled El Tannir, M: When you want to represent something, you create a template. This template is called a class or an object, and when you run in memory. You create an instance of this object or this template, and you can embed the attribute you can attribute. You can embed function, and you can apply inheritance, and so on. So this is object oriented programming model. And what about functional programming model? Any idea? So the functional oriented programming models or functional language wants or aims to represent everything as mathematical function. So, for example, in a functional language, I don't have a a loop. while, for example, in your Python language or C language or Java language. You can have a loop while do while, or for next or so you can apply a loop, for you can apply, start, index and index and apply an operation. These are loops or iteration in functional language that doesn't exist. A loop or an iteration is nothing but a recursive function. But this recursive function has different constraint to exit. So we are trying to solve everything and mathematical approach. So what is the benefit doing this, the benefit is to reduce the code you need to write when you are creating your model and object oriented programming approach. You need to create your template or the class. You need to to write all the attribute, getter and setter, because you need to to add the function you need to add different variable. So you need to write a lot of code. You need to write. For example, in Java you start by curly braces the code, and you the block you, you finish or terminate with closet, curly braces, and so on. You need to put semicolon at the end of each instruction. So you have a lot of type things to type. So by combining these 2 model programming model object, oriented and functional model, you reduce a lot, the code you need to write. So this is the 1st benefit writing Scala skylar. So when you write less code, you do less error. And this is very important. When you come to maintain when you come to, what? To, to. to refactor, maybe, and so on. So scanner is strongly, statically typed. What this means, everything is object. So everything is object, and even numbers, numbers can have function. So scanner is derived from derived from Java. So this is why. when you run Scala, it come run into Gvm onto Java virtual machine and compiles into byte code. So it is derived from Java. Scala is derived from Java. But Scala programming. Typically you write less code, as again you can. You write a lot of thing less, you see by by yourself, and there is. It is very concise. There is no boilerplate if you are comfortable to with Java, for example, and you want to put semicolour at the end of each function or instruction, you can do it, but it is optional. You are the the new. the K word new. When you create a new instance of your object of your class. This is implicit. You don't need to do it. Scala will do it for you. The compiler will do it for you. And so so we write less code. So we need less time to write, and we need less time to maintain. And we do less error. Okay, so Scala, as I said, does not have any primitive, only object. Numbers are object also. So means number has methods. Okay, and functions also are objects. So you can pass them everywhere as an object also. What is this? Sorry? What is also very, very interesting. In in Scala you can chain any number of function you need to perform. You can change this 1, 2, 3, 5, whatever. You can change any number of folks which is very, very interesting. So you can in a single line. You can wrote, you can write a lot of code. Okay, so, and everything behind scene is converted to primitive is necessary, and there is no loss of efficiency. Again, Scala was designed to be very efficient, very fast and support distributed execution environment. Okay, so now let's take a look to the variable. So you have the possibility to create a class. It is very similar to the object oriented programming languages and support to create a class. So you can create a class. But very important. You do not need. When you create your code integrate in particular this particular situation. You need maybe to specify the type. But Scala can infer the type of your variable for you. You don't need to specify the type. It is implicitly inferred by by the compiler. But we need to understand the main 2 variable type. So we have the Var and the Val very important to understand the difference. So the var means variable, so means this variable is readable and writable. So this, if you create a class, the scala will generate automatically for you the getter and setter, so getter and setter are the way to access to read the value from this variable. This is the getter and the setter are here to set the value or update the value for this particular variable or attribute. So, as you can see in the example, you can see class account, I said, just Var and Val. So in the background, or automatically base it on Var or Val. The compiler will understand. This variable is read and write or read only. So you just need to write this bar and Val. So Val means value which is read only this means once it is instantiated, or you initialize the 1st time the value. You cannot modify its value again, which is very good, because when you are creating program or application, you will be sure that the value in this variable will not change. Still, as is, nothing can happen, nothing. You cannot have an extra or hidden function that rewrites the value in this variable does not happen not possible so far and Var, I would say 99% in our application and our code. You will be using the Val. The read only the value. You associate this in the 1st run. and once the value is loaded. you don't need to modify the value again. Okay, we we create a new Val, which is more secure when you execute your code. So this is very important to understand this difference. And again, you have all the details in the tutorial dedicated to this part is understanding the variable type of scala var and difference between Var and Var. And you have many, many examples. Okay. Now, as its support, the object oriented programming model. So you can create class. A class is nothing. A class is nothing but a template, just a template to represent your entity. In Scala we have another type of classes dedicated for manipulating data. So you can create a class to represent your data. You can create getters and setters, add, attribute, add function to do everything. For example, you need to compare. For let's say, a person or a car. You create the object class for to represent a car. Now, you need to compare car. You need to clone car, you need to copy. So all this manipulation of the object itself as a data. Scala provide a new type or a dedicated type for this data manipulation. And this type is called case class. So case class are here exactly the same to represent data in a class, but it provide more features and function to manipulate easily your data which will reduce a lot of your code. So your code will be really reduced. So the case class are used to create or represent data that will be manipulated. So you you need just it is not just only the structure, but you can also manipulate that you can clone. You can test or compare this object together. For example, if you want to compare the object car. So in the 1st instance, I load the attribute. For, for example, a Bmw. In the second instance, I loaded the attribute for a Mercedes. Now you need to compare. How can you compare this object together? You can decide to create a function that compare names, compares, colors, compares, motor horsepower. And so you need to to to provide a way to compare how to can you compare these entities? So the case class will provide this by default. You don't need to implement this kind of features. Everything is already implemented. So and also the when you use the create, the case class, the in programming Orient object oriented programs. When you need to create the physical instance in memory from the class you use the new keyword. you see, the new keyword is used to create a new instance, to concretize your object in memory. The new keyword in Scala is optional. It is implicitly run and created by the compiler. So so you write less lot, really less code. So your code is more readable. Okay, so let's see the 1st example. This is very, very, very basic example. On the left. I have my code on the right. I have the execution of the code. So let's start with the case class. So here I am representing a person. Okay. So case class person, scala is case sensitive. So person, capital. P is different than person small. P. Starting by this with a small peak. So case class person 2 attributes. First, st name, last. Name. Okay? So here, as we, the compiler doesn't know anything about the attributes you are providing in this case class, you provide the type. Okay, it is not inferred at this level because it doesn't know how what to represent. But when you execute function, it can infer. For example, you can say, read this file, okay, it will return a string. For example, tell me the number of lines. Tell me the row counts account is numeric, so it will be an integral or a long something like that, so it can infer. But here you are defining the type, so it cannot and be inferred. The type cannot be inferred. So you need to provide the type. So 1st name is type of string. Last name is type of string great. So now let's see, Val, I'm creating a read only variable, Val. Me me is the name of the variable equal person, and you give a name. So you see, just by writing the case class name providing the value the compiler created for you, the object, the instance in memory, using the new keyword. Everything is done by the compiler. You don't need to write all this code. Just read the first, st I would say, affect a, or assign your variable and all you need to do so. Now the variable Me contains an object type of person, and this object contains these 2 value for 1st name and last name, you can access this, using the same notation as the object oriented programming language. So class, name, or instance, name period the attribute name so to extract or to get the 1st name and the last name just the name of the variable, followed by the name of the attribute. So here I am extracting and put and reading this into 2 different, variable, 1st and last. Okay, now we can compare, can use the compare. If me equal person, and you can provide 1st name and last name, and this will compare you. See, you don't need the compilery will compare. attribute, each attribute, so the last name to the provided last last name and 1st name to the provided last 1st name. So you see, it will prove it will do the comparison on all this attribute? Nothing. The developer wrote nothing, just only what we want to to check, if me equal person. This is all. So you see, we are writing less code, so I will explain in a moment what you can see on the output. so you can see the 1st execution defined the class person. This mean, the compiler accepted. analyzes the syntax. Everything is fine. So now we have defined our class. Once you run Val me equal person and provide the last name and 1st name. The compiler, as you can see, is getting an output. Give you an output. The output is me person. So what this means? This mean, I, the compiler, created a new variable. The variable name is me, and the type of this variable is person. Okay, this is the output same for the variable. 1st and last this is can be. You will see this dynamically on the screen when you type. Scala is able to run this dynamically. This is what we call the repel mode. This mean, you do not need to compile your code. It will act as a dynamic language as a python. You just run the command, you will see the execution or the result of the execution. So so this is very, very basic example, I have another one. I will can explain more in in details. So yes, about the spark session. This is very important to create or to access a spark. You need again the spark context. Without the spark context or the spark session, you cannot access your spark core. So if you write any program any application. You should start on the top of your application. You you mainly start by spark sessions, create builder, run on the master wherever so everything should be done this way. But in our environment, in the sandbox you don't need to do it, because it is already. I have already do this for you when you run. Zipline Zeppelin is configured to create this, the spark context for you. So you don't need to do it, and if you do it, if you run this code it will works, but it will use more memory. and we have very limited resources. So your if you, if you use more memory than allocated. This might crash. Okay? So again, for the spark session or spark context, you don't need to create it because I did everything for you. And when you load your spark interpreter in Zeppelin it is already created for you. So this will simplify a lot that works for you. You have only to use the spark variable. I will show you this in a second in in on Zeppelin, and everything is fine. Okay, this is important. the same for this spark sequel. So let's see this example, which is very, very basic example. And this example use the Rdd. okay, so I start by typing Val, text file equal A C dot text file. Read me. So while we understand, we need to create a read. Only variable text file is the name of my variable. The result should be. The content of this variable is sc, which stands for spark context. dot text file. So I want to perform or to run the text file function. And this text file function will read from the current directory. They read me, okay, the compiler will output will evaluate and output a result something on the screen. So it says, text file, which is type of spark Rdd string. and it is located at this location. In memory we are not interested where it is located. We are interested by the type. You can see, the compiler inferred automatically the type and created for us an Rdd, so I read a text file from from the disk or Look current directory. The output is now an Rdd. Of string, so it was able to infer the type of the file, which is a string, and all the data or the content of the file was loaded into an Rdd. Anyway. Now I can perform, for example, account. So the count here I run account, and now it is executed. This will trigger the execution and the result. You can see. Rs. 0, Rs. One. This is nothing but the result. They just say an index of the output. It doesn't matter if you rerun again it will change, so don't pay any attention to this number doesn't name just result number and whatever. So now, the result of the count function is long type. This is the type of the data. and the value is 126. I can call the 1st function. 1st function will show the 1st item in the Rdd. 1st item in the Rdd is string type. So res one result, one is type of string, and it contains the content of this item. The 1st item in the Rdd is the hash sign Apache spark. So this was loaded from the readme file. Okay, now, I can apply, for example, a filter. I can say, line with spark, text file filter. So here I am applying what we call inline function. This is very interesting feature of scala. So here I what I am doing. So we know that text file is an Rdd. Because the 1st line say spark is an Rd. so now I want to apply or to filter the content. So I want to exclude or include something. The items from the Rdd so filter. What I want to do is to check if the any item and the Rdds and the input Rdd contains the word spark with capital. S case, sensitive. Okay, so what here? What I can do? I can say, Okay, create inline function, this function. Take one parameter. And this function, you will get one item from the Rdd, as an, input, you take this item, and you check the content. Okay? So checking the content is contains spark the word spark, which is a string. Now the filter we can see line equal sign and greater. So the word line here is our function. But you can give any name you want. I gave line because it is more readable. But you can say, XYZ. Whatever. So I have only one element here line. So this mean the filter function can take one item from the in. Rdd, okay. now with with this item, apply the following, this is the meaning of equal greater sign. So lines means we get only one element from any element. from the input from the input Rdd and apply on this element. The function contains. You see, in this the syntax is maybe a little bit tricky for now. But you will be more familiar with this. This is what we call anonymous function. Okay? And now the result is an Rdd. Remember, Rdd. Are immutable. So the result is a new Rdd. Filtered with only items contained that contains the spark with capital S. I can do. I can shame this with count. so I can see lines contain spark dot count. This is another way to chain and get directly the result of the count. For example, you see. So this is just a very, very basic example. So now, if we take a look how we can solve a word, count word, count in Java. We need maybe 50 lines of code to create a word word count in Java. If you take the word, count source code of the mapreduce hadoop, you have almost 20 or not. 20, sorry, 50 to 60 line of code. We can do the same. And Skylar, with one line of code or 3 line of code. Okay, so first, st we need to load our file from the input, source. Okay, spark context. text file read, input, so this is for Rdsa. okay. Now, I want to do or perform the count. Val, our count, take the input. Here I am shading 3 functions. The 1st function is flat map. the result of the flat map function is transferred to the map function. The result of the map function is transferred to the reduced by K function. The result of the reduced by K is stored into the counts variable, which is read only. So let's read this so flat map, this mean, take any any array and explore this array into individual elements. Okay, so I take any element from the input line, equal, greater sign. This mean, take any element from the input and perform or run the split function on this element. So I have one line in a text, so it will read line by line, and for each line it will split, based on the space character. The split function will return an array, an array or collection of words. This collection of word is flooded, not in it by the flat map function. So the flat map function output is one column and each item. And this column contains one word. you see, in a single line. I did a lot. For for instance, I took the input as from text. I split it. using the split function and base it on the space character. This will return an array. This array is flattened into individual element, using the flat map function. No, this column, unique column is transferred or shaded to the map function. Now the map function will again. The word is just a variable name. You can give any name you want. But again, I use variable name, that it is readable for me, and I can read and better understand what is going on. So now take any element in this single column. any item, or for each item and generate a K value pair in the K take the word itself. the content of the item itself and the value put one. Remember this is how the word count works always, and when we generate Tappan K value pair always from the map for the word count algorithm. The value is one. So this will generate all the tuple for all the word with the value one. Now this new intermediate result is transferred to the reduced by K function. So the reduced by K function takes elements and apply an operator. The operator here is plus plus. You can apply any other operator. But but we are doing a word, Count. The word count is nothing but the aggregation of the number of occurrence. So we are using the plus operator to add this variable. This values so it will take 2 elements, because the reduced by the map generated a couple a Tuple K value pair. So we are taking oh, at at least all the K group it by K here reduced by K. So everything is grouped by K, and the reduced by K will take 2 by 2 the elements and apply the sum operator. If there is only one element it will add to 0. So think the underscore underscore means. Take any 2 elements and apply the plus operator. The result is stored in the counts variable. And now the counts contains the output of mapreduce is K value pair. So I have a collection of K value pairs in the count variable. You see, in a single function I was able to shave and did everything. And now I want to save or to persist this on disk. So I can say count, save as a text file and give the name. So give here the name of the output file. you see in a single one. Okay, so you say, 1st to read and last line of code to write out the result. I just use it. Only one single line of code with a 3 shaded function to perform a word count. And this is own. You see, writing this. Okay, maybe this is your 1st time. So it is maybe a little bit difficult to understand or to follow. But this is why you need to work a little bit at home. but you can see in only 3 function. I was able to to create a word, count, and this word count needs almost to 50 to 60 line of code in Java to perform the same. So now I I will show you in in the workshop what you have to do, how to run this and so on. So after this spark sequel. So spark sequel is the model. I would say the most used model in spark. Because spark SQL is here to let you manipulate and process your structured data. The data is structured. SQL, remember, SQL, stand for structured query. Language strictly just add small words to treat data, structural data, query language. It is not only the language itself is structured. The data is structured. Okay, so it it is here to to let you query, your data. Do aggregation. Do do all this analysis? Okay? So again, the 1st version introduced by Spark was based on the Rdds. And now and we'll not be working with Rdd anymore because we have the higher abstraction, the data frame and data set. But when it was introduced it needs to. We need to create the schema Rdd. and using this schema, you're still able to provide the schema. You can choose to infer the schema from the input. Data. But sometime it doesn't work 100% as you wish, or as you wanted to do to be so, maybe sometime. It depends on the case, on the data. Maybe you need to provide the schema yourself. But in in the workshops and the tutorial. You will see both case when we infer, and when we need to provide the schema explicitly. So you need to define the data source and the data source in the in the spark sequel will let here. This is where you define your source. You say I want to read from a text file. I want to read from my sequel. I want to read from Hdfs. I want to read. So this spark, SQL. Architecture is open and will let you connect to any source. And using, of course, any languages is no restriction at this level. So what is a data frame? As I said, if you are already comfortable with pandas data frame in Python this is exactly the same. So it is a collection, a distributed collection of data organized into name and column. so that a frame is a connection. It is very similar to a table. So you have to define column, name, column type, and you insert data base. It rows based on this column type. So the spar data frames can handle terabyte and petabyte of data and it support different storage format. You can support for hive. We'll talk about this in the next week, so we can support different file format also. So again, this will be covered later. So. But again, the data frame are accessible in Scala, Python, and Java and app. All these languages can access data frame very simple, very simple structure. Just define column name. It can be defined explicitly, or it can be inferred from your data. And this will create a data frame. The spark data set, which is a different type. So this is same structure, it. But it is typed. This mean, you cannot create a data set without providing a type. A type is, for example, I want to load my car or my contacts file. Okay, in the data frame just load contacts. Okay? So every row is untyped. While in data set, you need to create the type. and each row in the data set will be an object. This typed object is not just untyped object. And again, data set type is not accessible. From Pyth, dynamic language, Python and R is only accessible in Scala, or of course, Java. and of course it is much faster than the Rdd implementation. Huh? Okay, so just to give you an idea about the memory representation. So in the Rdd is a row. nothing. It is up to you to load and each item split. And we say, yes, process each item individually in the data frame. It is looks like a table. So it is organized into columns and rows and the data set. It is a type. So each row has a type. It is a new car, for example, in this example. So this is why today we are using data frames in the office and data set. But I say, it is what I see in in many, many project is, people are using more data frames than data set. But it is fine. I do. It depends on the use case sometime. Okay, we can keep this as a data frame. But sometime we need to go to data set type. So this is an idea to give you an idea about the performance. So you can see the difference between data data frame in Python and Scala. It is almost similar when we're processing data frame in Python or in Scala, while when we processing Rdd. In Python and Scala, Rdd. Is slower when you process data in Rdd using the Rdd Api, it's just slower. No. the, as I said internally, spark use data set type and not data frame, because everything, all the data the data frame you will be using is nothing but an untyped data set. nothing. But it is type of row and is not type of your car, person or type that you provide. Okay, so this is the only difference. And the data frame type is nothing but an alias in the spark. SQL. Api. But this is transparent for you now with data frame you can perform, of course, read operation and write operation. You can read from any format you can read from Csv. From Json, from parquet. You will learn about parquet in week parquet, and all other file formats in week 6, and also you can choose to write in any format you want. So this is very easy to convert between format. So, for example, you load data into a Json format. and you want to convert into parquet or into Csv or to another format. Just load this into a data frame, as you can see, spark, read, format, provide the path and the format type. and you write output the output. You can choose. Okay, the mode, overwrite, append, and whatever, and the format. So it is very easy to convert between format from Json to Csv. From Json to Parquet, Csv. To Avro, whatever. So it is very, very, very easy now how to use the Api data. Frame. Api is extremely simple to use. So you define the you, you, the spark. Here you can see spark is your the spark sequel context. the spark core has a context spark sequel has a context. So the spark core. SQL, sc spark context, the spark sequel context is spark. So spark. Here is our context. Read, perform the read operation. I want to read Json, format. Format, Json, or Json are the same just different ways to say the same thing. So if you go back here you can see the 1st syntax spark, read, format. and spark. Read, G. 7 are compatible, the same so. But I sometime I do prefer to use this style coding, it is more readable. But if you are more comfortable you can use the second. No problem. Okay. And now you can, for example, convert from Jason to parquet you can use. Okay, my data, frame flights, write mode, overwrite and output per K, and the name of the file output or the output is flights. Now we have different data source already embedded. Other should be added. So what we need in our course already provided to you. You do not need to add anything. All will be reading different file formats. Json, Csv. Parquet, Avro. You connect to Mysql, connect to Cassandra. Whatever all these are already provided and studied, and nothing to do from your side. so don't start installing anything. Everything is already installed so, but by default not everything is installed by default or provided by default. By default. You can read Json. You can connect to Hdfs. You can read from Hive it, read parquet. We'll learn about parquet more in details in week. 6. Because this is how spark! This is the default format of spark parquet. And if you need to work with Cassandra, with Avro, which is base all, or Csv. All these need to be added as external package and integrated to be able to use. But again. this is outside our sandbox sandbox have already has already everything, installed, everything you need already installed. So you don't need to do anything else, but by default not everything is provided. Now, how to use very simple as the 1st example. So now the 1st part is the data frame. Api, this mean, I will access data, query, data, aggregate data using the data frame Api, so I will use function and chain function from the data frame. Api, so read Path Json to load the file, and then we can do a group by. I can do aggregate, count whatever. So everything is function. So this is the data frame Api. While the SQL. Api. In this case I will use the SQL. Language, select star or column name from my table name and will return a result. So the data frame Api will be using function and method from Scala. The SQL. Api will be using the SQL. Language. Select star from a column name from my table. But to be able to use the SQL. Api, you need to create a virtual table or virtual view in memory. So you load your data. And you, once you have already your data frame loaded. you need to create this virtual view or table in memory. The the difference between the view and the table. The table is read, write, the view is read only so to be able to run a SQL. Command. You need to run, the create, or replace temp, view or temp table, and you give a name. This name is the name of this virtual table, or virtual view in memory. No, you will be able to perform the SQL. Query. So here is just data. Frame Api flight, select colon, origin, destination, departure, delay. filter the column. We'll talk about this dollar sign. The Dollar sign is here to specify the color name in the current data frame dollar, sign departure, delay, so filter departure, delay greater than 15 and show select is a transformation show is an action. So show will trigger the execution of the select function and limit the result to 5. The exact query, using the SQL. Language, select origin, destination, departure, delay from flights, which is the name of the virtual table, or virtual view in memory. where depth, delay limit 50 greater than 15 and limit to 5. Both will return the same result both. Now the data set, how do you create a data set? You need to create first, st the type which is the first, st the case class, and then you can create data set. Otherwise you cannot create data set. So here you have a very, very basic example from data, from memory. But again, you have more details and more examples on on the sandbox tutorial. So here we create the case, class book, name and cost. So here I create in memory a sequence. This sequence is a book you see here, I'm creating directly the instance. Okay, so I create a 1st instance book 3, instance, Scala Spark and Kafka and I convert this to the data set to Ds or 2 data frame compared to data frame book Ds show this week. So the parentheses are optional. If you are comfortable to to write the parenthesis, you can do it. If you are, don't want to put it, don't put it. This is the parentheses are optional in Scala, and the result is shown here in the tutorial, you have different example and more. But here again, in this slide, we cannot cover everything. Okay, another example. So I create here a case class for user email, foot, size and name. Here I am providing. So you can see here, I'm creating the sequence in memory. the sick sequence, the array, if you want is a type already typed is a collection of instance book. In this example I just provide the type to the data set, and it is loaded directly and infer it directly when I load the file. So I create the case class prior to my to read the the input data. And as you can see here. Val. okay, this is my variable read, only user data set spark. Read from Json. this is the location file location and path. And as user. This will create a data set. So the output of this command is a data set. Just, I gave the type, and while loading spark it will create the instance, the new and load each row as object as a user object in the data set. And now, once is loaded, you can do. You can use any style coding style. So here, using map user Ds data, set, map, underscore name. This mean, take any element in the data set and take only the variable name, so it will return the name from the for the users. But map is transformation will not output any result. To to trigger the execution we need to perform an action collect is an action so Collect will do this, trigger the execution, and return the result. So the result is nothing, but all the names of our user from the input now we can do the same with filter, filter, foot size, greater than 38, or whatever, and collect. This will return a result, because collect is an action. So this is everything. Again, I know this is new for you, but you need to just practice this at home. And again I will go over to tutorial and this exercise and workshop. So I provided to you here also. Very, very basic. I know we'll not be doing any Rdds. But it is. I think it is good to know how this works to give an idea. So here you have some examples for filter function, map, flat map, and all these examples I provided to you as Zeppelin. Note. So just you can run this and try to understand how this works. And here, yes, this is about the context. I will show you the in Zeppelin. Yeah, this is about the spark performance terra sort. So it was created on. This is a real benchmark, and the the real hadoop took 72 min to process the the logistic regression and spark took 23 min. and it was a number of node. You see, 2,100 for hadoop and 26, 2,006 for spark. Okay, if you want to go more in detail. So this is our reference books. So now let me switch to my. So in in the oops. So you have. This workshop has a 3 parts. I just separated to more be let you perform part the Rdds, the data frame and the data set each part separated. So it is. Give you time to to practice, and so on. So so the again the Rdd. We will not be doing any Rdds, but I think again. It is a good thing to know how this works, especially you are new. and you will be performing this simple exercise. and you have on the on the tutorial, the sandbox. You have the basic of Apache spark. Rdd, you have everything we need to know as beginner, how this is structured, how we can call. Just go over it. Don't don't pay. Don't waste, I will say wasting time wasting, but just give it. 1 h or 2 will be fine. and here you have an example I provided to you an exercise. This is about, how are these words when you read a file, and how you perform, you can perform very, very small operation. This is nothing but word, count, but this word count is a little bit different, because here we are counting data from the input. And we are doing something a little bit more more fancy. So this is step by step. Sure. the separate note will give you everything you need in this part. So. But where I want you to go more in detail and pay more attention to details is the data frame and data set. So data frame. So here you have this 3 tutorials. So you can have an idea about the theoretical part and practical part. So the hands-on tutorial will give you many, many examples how to load data, how to filter everything, so that aggregation is dedicated to aggregate your data and exploring data frame with summary and describe. So these are 2 function in spark. Let me open the tutorial. Maybe it will give you a better idea. So in spark where it is. So the basic of Rd, this is our 1st one. So it will explain to you about the Rdd of pairs how to parallelize from that in, in memory. How can you create? Everything is here. Okay for the second one. It is which one? It is a spark. Let me open spark. Yeah. Spark. Also, we have. Yeah, this one. The flight is very important. The data aggregation is important. Spark sequel is important. And we can say, for data set, okay. this data the hands-on tutorial. So this will let you use based on this file. So how we remember here we are doing the data collection part. We put the data on Hdfs and from Hdfs we perform filtering and processing and output to Hdfs. So this will give you an idea how to do it. So this is the spark context, and how you can rename column, how you can. So there's a lot of things to learn from these tutorials. Okay? The second one about the same. But for the data set type. So how can you create the case class to start with the case class. what? Let me show you? Yeah. here we can from memory, using the sequence, a sequence. The input file here from Hdfs and how you create a case class. and then how you can define your own type. Schema. This is how we can define your own schema and how you can read. You see, this is Spark. Read Spark our context. and you can explore by show using all the comments. Now, the use case is very important. Here the airline flight data analysis. This will give you all you need to know about data aggregation joining. So this is very close to what you will be doing in your homework. Okay? So so go over it and take your time X. Try to understand every step we are doing. There is a lot of information here. In in this case study. There is a lot of information. Oh. it will give you the way to practice, because here you will practice, join how to join data set or data frames. How can you aggregate using the data frame Api, and so on. So you have almost everything. Now let me show you and Zeppelin where it is. My zip. So why, zip so so you can create what I say. I know. Okay, here, this one. Let me go delete this one and create a new lot. So okay, let's name it a spark. And by default, by default, Zeppelin will choose the spark interpreter. Okay, by default. So let's skip it. And now again, I recommend always to put the name of the interpreter in the in the sense this will let you understand what you are doing, what command you are running. and especially this because spark will always try to run a spark, command even you are running shell, command, or Markdown command, whatever it will start, it will try to run a spark. Command. So to prevent this, just always define the name of the interpreter. So Spark is a spark. The interpreter name is a spark. So here you can see for the spark core. We have the context, sc. And for the spark sequel is spark. So let's say now, sc. Dot version. So to to to show you that the spark context is already created, for you don't need to create it. So the 1st run is always long, and it might take up to 1 min. 2 min depends on your machine. And remember, this is not the case when you run this in production, because this is assume you assume we assume that it's running always. So once the context is created, don't need to create each time. But here we are stopping our context each time and restarting the sandbox. So it will need to recreate the spark context. Okay? So now we got the result. As you can see, my result is the version of my spark version. Now let me show you the cluster. So the cluster. When you go to the hadoop cluster. it will open this. So here you can see all the application we are running from Zeppelin. So you can see here. I just run this spark context. So it is now listed as application. But if you go to the Scheduler the scheduler is how yarn is scheduling your application. So here you can see you have in this the upper part. You have the number of core. You have your number of memory, the memory size allocated. So you have all the information, but it is so large, so I need to make it very small, but it is not readable. So and here you need to scroll. You need to scroll to see. For example, you can see here we have 256 MB memory, and one virtual core, and so on. But what it is important is the scheduler. So this, remember, in our sandbox spark is running in a distributed environment. It is not running in standalone. It is running over yarn. So to to simulate the production environment. Okay, so here, let me. So if I stop the context. okay, so this will free the memory and everything should be loaded again, so I can refresh. And as you can see it is now free. Okay, so here in the scheduler you can check the memory usage by yarn. Okay, so let's run our spark. So spark. Now it will load. Okay, now I want to restart because it is not. It is in memory. I want to to clean up the memory and ensure that it will be loaded again. Okay, so now now it is running. You should see and have the same behavior on your machine. Now, if you go here so, as you can see. Now, Spark is using the queue of in yar, and we can see here we have the plane, the memory resources allocated. How many users, how many calls? So you have all the information here. if it is overloaded, whatever. So you can see here. But this is not auto refresh, you need to refresh. And you see, whenever it is loaded. So here you have the size of the queue, you can see it is, use it 26%. Now. of course, when you run your pro application so this will be will grow up. So now where is my Zip? It is here. So if, as you can see, it is executed and on my machine, it took 20 second. So the 1st time it will take maybe 1 min. If your machine is, I would say, very powerful, you have lot of power should should be low. This power contact should be loaded in less than 1 min. If it take 1, 2 min doesn't matter, because once it is created it is great. It is here. And now you can perform any anything you, you, you, you you want in spar. And when you finish. when you finish, you can run this park. Stop context. Okay? So this will free the memory. So if I go back to my cluster and I refresh. You see, the queue has been flushed, and nothing in in the queue more so now, if I take, let's take a real example. Okay, so let's say, this is our spark data frame. This is an one of the exercise you have. So we create a directory and load this to, I'll take just one or 2 min, and then I will finish this part. Okay, so the file already exists. So if you want to again, if you want to force the overwrite, you can use hyphen F, but by default it will not override the file. So now it, the file will be overridden. Okay? And now, you see, I use spark. This is spark context. Read format. Csv, the option. We have a header in our input file, infer the schema. I want you to infer the schema. Use the delimiter. And this is where the file is located on Hdfs. Okay, so now it will run. as you can see here, it will need to load the context. and it is low as I just stopping it. It would. It need to reload it, and when it is loaded. So here Spark Zephyrin will create the spark context and we'll perform reading the file. So this is why, might be a little bit longer. But we can separate this. Okay, okay, when you see spark job here, this means start running the this block. Okay. Now, you can see that the data type is data frame to see, the data type is data frame. And we have this. Now, we can show the schema. This is the schema of the data frame. So I have all these columns and the but the type is inferred by the compiler. I didn't provide any type. And now we can perform, for example, the count. So this will return the count of the number of element in the data frame I am using here the data frame. Api. the perform the select. As you can see, this is the result. So go over the tutorial, and this exercise is very important. Prepare yourself for your next class. The next class will start the second part of the course. The data analysis. You'll be introducing Hive and the hive query, language, and your homework is after the class will be available for you, so you can start after the class or whatever you want, working on your 1st home. and here we can see the result returned by spark, as in text. In the tutorials you have the way how to convert this into Zeppelin grid. So you can show chart and use different widget to to show your data, for example. Yeah, as here can see. So here I am using the spark sequel. Okay? So I created the temporary view in memory. Okay? And now we can see instead using spark only as an interpreter. I use spark dot SQL. And I just enter my SQL. Statement. Okay. so this way, you can see. And you have. If you go over the tutorial Zeppelin and all the others. You can see how we can convert from the text result into a grid results. So we can show statistics. So of course, you didn't select the statistics, the to for the graph. But you can check this out. So this is everything. Sorry I exceeded that the last time our time. But if you have any question, do not hesitate. You can drop me an email. You can ask. Now, if you have a question I am here to answer and help you. No question. Okay? So I will wish you a very good night and see you the next week. Thanos Michailopoulos: Thank you very much. Isaac Abouganem Stephens: Thank you very much. Have a good one. Khaled El Tannir, M: Yeah. Hammed Akande: Thank you. Have a good night. Khaled El Tannir, M: Good night. Cristal Cortez: Bye. Thank you.