Spark Architecture
===================

Spark on YARN

Hadoop -

HDFS - NameNode , DataNode (Storage)
MAPREDUCE - Processing/computation
YARN - Resource Manager

YARN -
Resource Manager (Master)
Node Manager (slave/worker)

you are invoking a hadoop job from client machine

hadoop jar <>

what will happen now?

The request goes to the Resource manager

Resource manager will cordinate with one of the Node Managers and create a container in that worker node..

inside this container it will start a service called as Application Master.

this application master will act as a local manager to manage this application..

this application master is now responsible to get more resources for the application, it will request the resource manager for more resources..

3 containers

2 containers 2 gb ram, 1 core, worker node 1
1 container 1 gb ram, 1 core, worker node 2

we got containers on worker node 1 & node 2

node manager will come in...

node manager manages the containers/executors which are running on worker nodes.

this application master will interact with name node to understand where the blocks of file is kept in hdfs..

uber mode - the job is so small that it can run in the container in which application master is running, it does not need other containers.

application master is called as Driver..

every spark job has one driver

YARN

Spark Architecture

Application master is your driver

interactive mode - Notebooks, pyspark shell

submit the job - spark-submit

client mode - Notebook, spark shell - Interactive

cluster mode - spark-submit

client mode - the spark driver runs outside the spark cluster. it runs on the gateway node/edge node

cluster mode - the spark driver runs in the spark cluster. even if the gateway node crashes, or even when we logout from gateway node the application will still run..

==========

1 driver

5 executors

YARN - Yet another resource negotiator

Resource Manager - Master

Node Manager - slave

http://m02.itversity.com:19088/cluster

Total Memory - 151 GB (Memory)

Total Vcores - 90 (CPU)

3 worker nodes

each node configuration
=========================
12 CPU cores - 36 vcores - 30 vcores..
64 GB RAM - 50.33 GB is used as part executor memory

3 machines

90 vcores
151 GB RAM

1 driver - gateway node / worker node

2 executors

1 CPU core
2 GB RAM

memory - GB

storage memory
execution memory

min
<memory:1024, vCores:1>

max
<memory:8192, vCores:4>

Yarn is like a operating system, and it manages he resources

capacity scheduler

100% resources

60% sales

40% marketing

various ways to access a column in pyspark

column string "cust_id"

column object col("cust_id")

column expression expr()

```
orders_df.select("order_id",orders_df.order_date,orders_df['order_date'],column('cust_id'),col('cust_id'),expr("order_status")).show()
```

"order_id" - column string

orders_df.order_date

column('cust_id') - column object

expr("order_status") - column expression

you are joining 2 dataframes..

both these dataframes have a column which is cust_id

orders dataframe

customer dataframe

cust_id

```
orders_df.select("order_id","cust_id", expr("cust_id + 1 as new_cust_id")).show()
```

```
orders_df.select("order_id",orders_df.order_date,orders_df['order_date'],column('cust_id'),col('cust_id'),expr("order_status")).where(col('order_status').like('PENDING%')).show()
```

```
orders_df.select("order_id",orders_df.order_date,orders_df['order_date'],column('cust_id'),col('cust_id'),expr("order_status")).where("order_status like 'PENDING%'").show()
```

column string - "order_id"

column expression - expr("order_id")

column object - column, col("order_id")

prefix the dataframe name before the column name

=============

Aggregate functions
=====================

/public/trendytech/datasets/order_data.csv

1. simple aggregations
- will give only one output row
count the total number of records, you want to find the sum of quantitites

count the total number of records, count number of distinct invoice ids, sum of quantities, avg unit price

programatic style
===================

```
orders_df.select(count("*").alias("row_count"),countDistinct("invoiceno").alias("unique_invoice"),sum("quantity").alias("total_quantity"),avg("unitprice").alias("avg_price")).show()
```

```
orders_df.selectExpr("count(*) as row_count","count(distinct(invoiceno)) as unique_invoice","sum(quantity) as total_quantity", "avg(unitprice) as avg_price").show()
```

```
spark.sql("select count(*) as row_count, count(distinct(invoiceno)) as unique_invoice, sum(quantity) as total_quantity, avg(unitprice) as avg_price from orders").show()
```

column expression

spark sql

2. grouping aggregations
- we will do a group by

3. windowing aggregations

===============

grouping aggregations
=======================

programmatic

```
summary_df = orders_df \
.groupBy("country","invoiceno") \
.agg(sum("quantity").alias("total_quantity"),sum(expr("quantity * 
unitprice")).alias("invoice_value")).sort("invoiceno")
```

select expression

```
summary_df1 = orders_df \
.groupBy("country","invoiceno") \
.agg(expr("sum(quantity) as total_quantity"),expr("sum(quantity * unitprice) as 
invoice_value")).sort("invoiceno")
```

spark sql

```
orders_df.createOrReplaceTempView("orders")
```

```
spark.sql(""" select country, invoiceno, sum(quantity) as total_quantity, 
sum(quantity * unitprice) as invoice_value from orders group by country, 
invoiceno order by invoiceno""").show()
```

=========

windowing aggregations
=======================

/public/trendytech/datasets/windowdata.csv

```
  Germany|    48|         11|        1795|     3309.75| 3309
  Germany|    49|         12|        1852|     4521.39| 7800
      Germany|    50|         15|        1973|     5065.79| 12850
  Germany|    51|          5|        1103|     1665.91| 14200
```

1. partition by based on country
2. sort based on week num
3. the window size

```
mywindow = Window.partitionBy("country") \
.orderBy("weeknum") \
.rowsBetween(Window.unboundedPreceding,Window.currentRow)

result_df =
orders_df.withColumn("running_total",sum("invoicevalue").over(mywindow))

result_df.show()
```

Windowing functions
====================

rank
dense_rank
row_number

lead
lag

windowdata.csv

windowdatamodified.csv

/public/trendytech/datasets/windowdatamodified.csv

```
hadoop fs -cat /public/trendytech/datasets/windowdatamodified.csv
```

when calculating a running total

1. partition column
2. sorting column
3. window size

ankur - 100 Rank - 1 Denserank - 1 rownum - 1

satish - 100 Rank - 1 Denserank - 1 rownum - 2

Kapil - 100 Rank - 1   Denserank - 1 rownum - 3

kaushik - 99 Rank - 4 Denserank - 2 rownum - 4

Ram - 99   Rank - 4 Denserank - 2 rownum - 5

rohit - 98   Rank - 6 Denserank - 3 rownum - 6

100 seats...

rank function

2 people - 100 points - 1st rank

1 person - 99 points - 2nd rank

1 person - 98 points - 3rd rank

gold medals

silver medals

bronze medals

==================

lead, lag

========

```
mywindow = Window.partitionBy("country")

orders_df.withColumn("total_invoice_value",sum("invoicevalue").over(mywindow))
```

=========

logdata file stored in hdfs

```
INFO,2015-8-8 20:49:22
WARN,2015-1-14 20:05:00
INFO,2017-6-14 00:08:35
INFO,2016-1-18 11:50:14
DEBUG,2017-7-1 12:55:02
```

hdfs path is : /public/trendytech/datasets/logdata1m.csv

1 million records

we need to analyse these logs and find some inference

january error 10000
december info 20000

12 months

5 different log levels

60 output rows

```
logs_data = [("DEBUG","2014-6-22 21:30:49"),
("WARN","2013-12-6 17:54:15"),
("DEBUG","2017-1-12 10:47:02"),
("DEBUG","2016-6-25 11:06:42"),
("ERROR","2015-6-28 19:25:05"),
("DEBUG","2012-6-24 01:06:37"),
("INFO","2014-12-9 09:53:54"),
("DEBUG","2015-11-8 19:20:08"),
("INFO","2017-12-21 18:34:18")]
```

now we want to apply some aggregations

```
spark.sql("select loglevel, date_format(logtime, 'MMMM') as month, count(*)
as total_occurence from serverlogs group by loglevel, month").show()
```

======

```
from pyspark.sql import SparkSession
import getpass
username = getpass.getuser()
spark = SparkSession. \
   builder. \
   config("spark.sql.warehouse.dir", f"/user/{username}/warehouse"). \
   enableHiveSupport(). \
   master('yarn'). \
   getOrCreate()
```

```
logs_data = [("DEBUG","2014-6-22 21:30:49"),
("WARN","2013-12-6 17:54:15"),
("DEBUG","2017-1-12 10:47:02"),
("DEBUG","2016-6-25 11:06:42"),
("ERROR","2015-6-28 19:25:05"),
("DEBUG","2012-6-24 01:06:37"),
("INFO","2014-12-9 09:53:54"),
("DEBUG","2015-11-8 19:20:08"),
("INFO","2017-12-21 18:34:18")]

log_df = spark.createDataFrame(logs_data).toDF('loglevel','logtime')

log_df.show()

log_df.printSchema()

from pyspark.sql.functions import *

new_log_df = log_df.withColumn("logtime", to_timestamp("logtime"))

new_log_df.show()

new_log_df.printSchema()

new_log_df.createOrReplaceTempView("serverlogs")

spark.sql("select * from serverlogs").show()

spark.sql("select loglevel, date_format(logtime, 'MMMM') as month from
serverlogs").show()

spark.sql("select loglevel, date_format(logtime, 'MMMM') as month, count(*)
as total_occurence from serverlogs group by loglevel, month").show()

logschema = "loglevel string, logtime timestamp"

log_df = spark.read \
.format("csv") \
.schema(logschema) \
.load("/public/trendytech/datasets/logdata1m.csv")

log_df.show()

log_df.count()

log_df.createOrReplaceTempView("serverlogs")

spark.sql("select * from serverlogs").show()

spark.sql("select loglevel, date_format(logtime, 'MMMM') as month from
serverlogs").show()

spark.sql("""select loglevel, date_format(logtime, 'MMMM') as month,
count(*) as total_occurences
from serverlogs
group by loglevel, month""").show()

spark.sql("""select loglevel, date_format(logtime, 'MMMM') as month,
count(*) as total_occurences
from serverlogs
group by loglevel, month order by month""").show()

spark.sql("""select loglevel, date_format(logtime, 'MMMM') as month,
date_format(logtime, 'M') as month_num,
count(*) as total_occurences
from serverlogs
group by loglevel, month, month_num order by month_num""").show(60)

spark.sql("""select loglevel, date_format(logtime, 'MMMM') as month,
int(date_format(logtime, 'M')) as month_num,
count(*) as total_occurences
from serverlogs
group by loglevel, month, month_num order by month_num""").show(60)

spark.sql("""select loglevel, date_format(logtime, 'MMMM') as month,
date_format(logtime, 'MM') as month_num,
count(*) as total_occurences
from serverlogs
group by loglevel, month, month_num order by month_num""").show(60)

spark.sql("""select loglevel, date_format(logtime, 'MMMM') as month,
first(date_format(logtime, 'MM')) as month_num,
count(*) as total_occurences
from serverlogs
group by loglevel, month order by month_num""").show(60)

result_df = spark.sql("""select loglevel, date_format(logtime, 'MMMM') as
month, first(date_format(logtime, 'MM')) as month_num,
count(*) as total_occurences
from serverlogs
group by loglevel, month order by month_num""")

result_df.show()

final_df = result_df.drop("month_num")

final_df.show()
```

```
spark.sql("select loglevel, date_format(logtime, 'MMMM') as month from
serverlogs").show()

spark.sql("select loglevel, date_format(logtime, 'MMMM') as month from
serverlogs").groupBy('loglevel').pivot('month').count().show()

spark.sql("select loglevel, date_format(logtime, 'MM') as month from
serverlogs").groupBy('loglevel').pivot('month').count().show()

month_list = ['January','February','March','April','May','June','July', 'August',
'September', 'October', 'November', 'December']

spark.sql("select loglevel, date_format(logtime, 'MMMM') as month from
serverlogs").groupBy('loglevel').pivot('month',month_list).count().show()

month_list = ['Jan','February','March','April','May','June','July', 'August',
'September', 'October', 'November', 'December']

spark.sql("select loglevel, date_format(logtime, 'MMMM') as month from
serverlogs").groupBy('loglevel').pivot('month',month_list).count().show()
```