DharmeshGajera_week8_assignment

Question 1.
What is UBER mode?

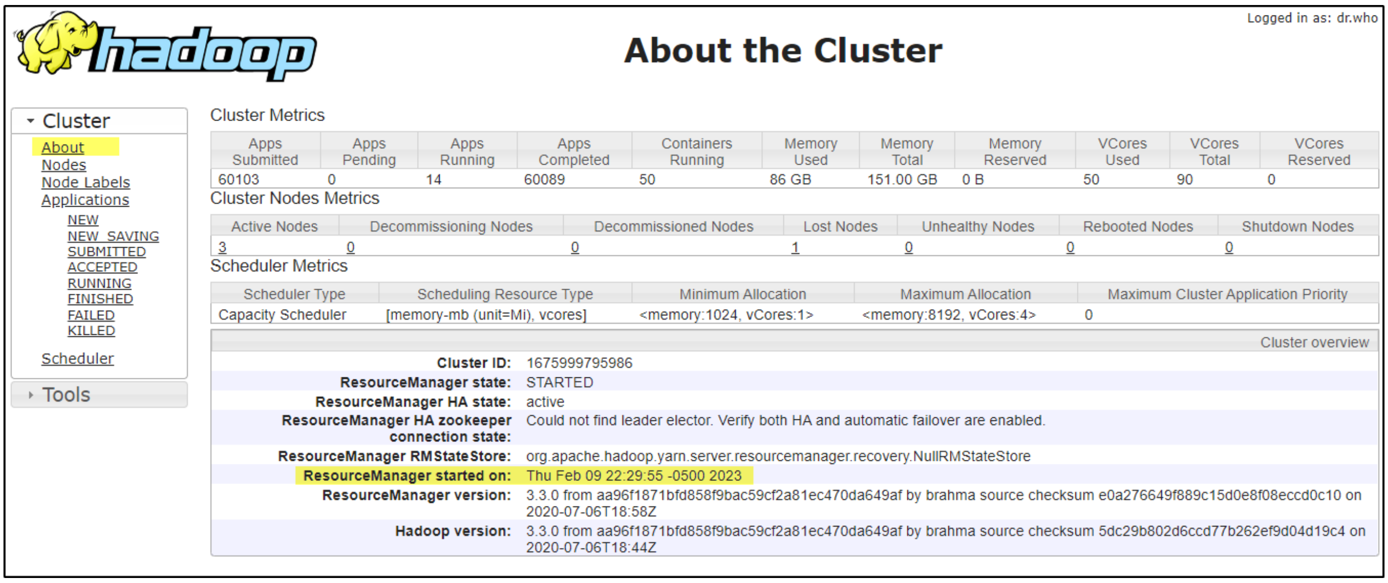
Ans: - Normally mappers and reducers will run by Resource Manager (RM), RM will create separate container for mapper and reducer. Uber configuration, will allow to run mapper and reducers in the same process as the Application Master (AM).

If you have a small dataset or you want to run MapReduce on small amount of data, Uber configuration will help you out, by reducing additional time that MapReduce normally spends in mapper and reducers phase.

Uber jobs are jobs that are executed within the MapReduce Application Master. Rather than communicate with RM to create the mapper and reducer containers. The AM runs the map and reduce tasks within its own process and avoided the overhead of launching and communicate with remote containers.

Question 2.
Multi-node cluster
Ans: -

1. In About section it shows when Resource Manager Started and other details like Resource Manager version and Hadoop version.



Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved
60103	0	14	60089	50	86 GB	151.00 GB	0 B	50	90	0

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes	Shutdown Nodes
3	0	0	1	0	0	0

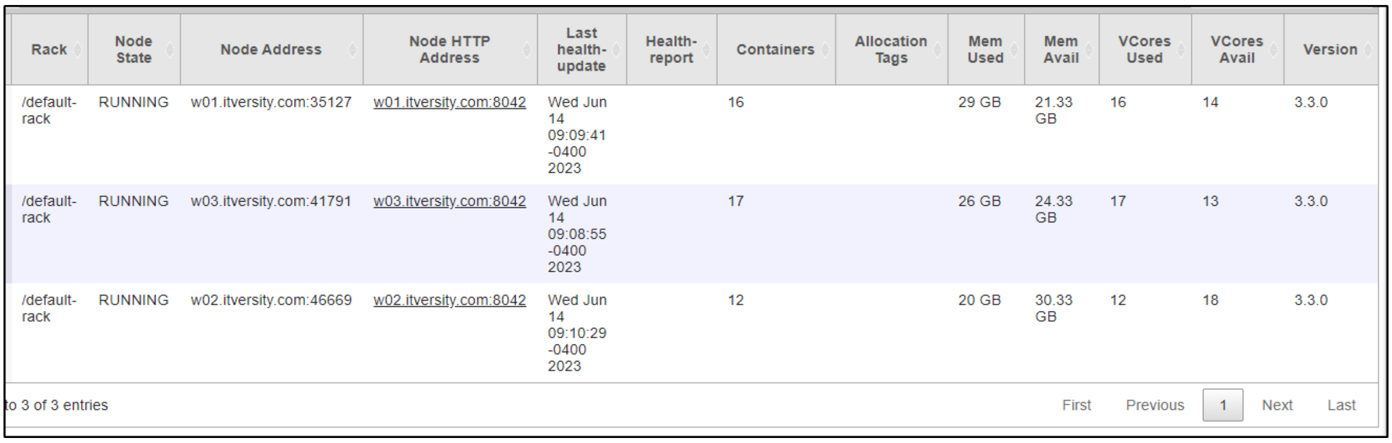
Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation	Maximum Cluster Application Priority
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>	0

Cluster overview

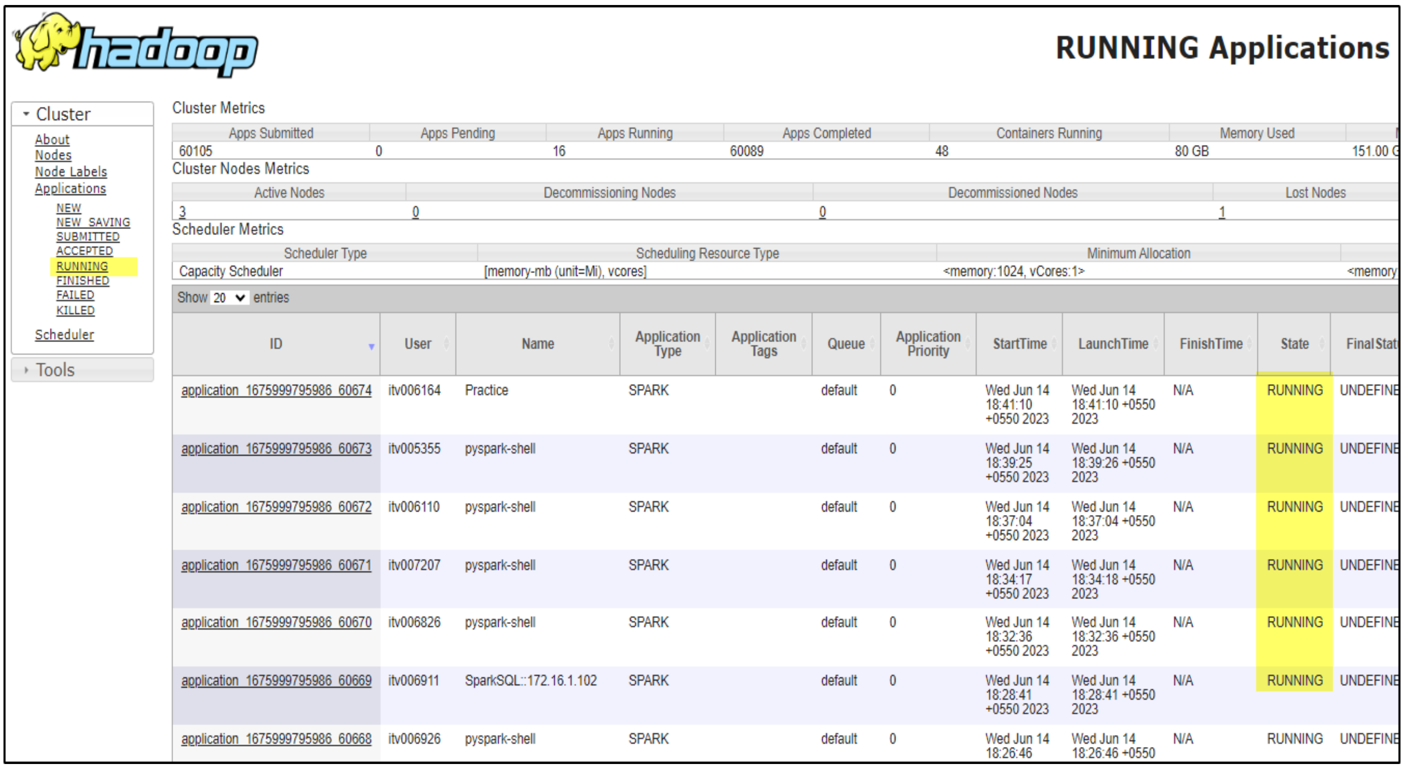
Cluster ID:	1675999795986
ResourceManager state:	STARTED
ResourceManager HA state:	active
ResourceManager HA zookeeper connection state:	Could not find leader elector. Verify both HA and automatic failover are enabled.
ResourceManager RMStateStore:	org.apache.hadoop.yarn.server.resourcemanager.recovery.NullRMStateStore
ResourceManager started on:	Thu Feb 09 22:29:55 -0500 2023
ResourceManager version:	3.3.0 from aa96f1871bfd858f9bac59cf2a81ec470da649af by brahma source checksum e0a276649f889c15d0e8f08eccd0c10 on 2020-07-06T18:58Z
Hadoop version:	3.3.0 from aa96f1871bfd858f9bac59cf2a81ec470da649af by brahma source checksum 5dc29b802d6ccd77b262ef9d04d19c4 on 2020-07-06T18:44Z

2. In the Nodes tab we can see the all nodes of the cluster with details of Containers, Total Used Memory, Total available Memory, Total Used VCores, Total Available VCores.



Rack	Node State	Node Address	Node HTTP Address	Last health-update	Health-report	Containers	Allocation Tags	Mem Used	Mem Avail	VCores Used	VCores Avail	Version
/default-rack	RUNNING	w01.itversity.com:35127	w01.itversity.com:8042	Wed Jun 14 09:09:41 -0400 2023		16		29 GB	21.33 GB	16	14	3.3.0
/default-rack	RUNNING	w03.itversity.com:41791	w03.itversity.com:8042	Wed Jun 14 09:08:55 -0400 2023		17		26 GB	24.33 GB	17	13	3.3.0
/default-rack	RUNNING	w02.itversity.com:46669	w02.itversity.com:8042	Wed Jun 14 09:10:29 -0400 2023		12		20 GB	30.33 GB	12	18	3.3.0

3. In the Application there is RUNNING tab which shows only RUNNING applications.



RUNNING Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	
60105	0	16	60089	48	80 GB	151.00 G

Cluster Nodes Metrics

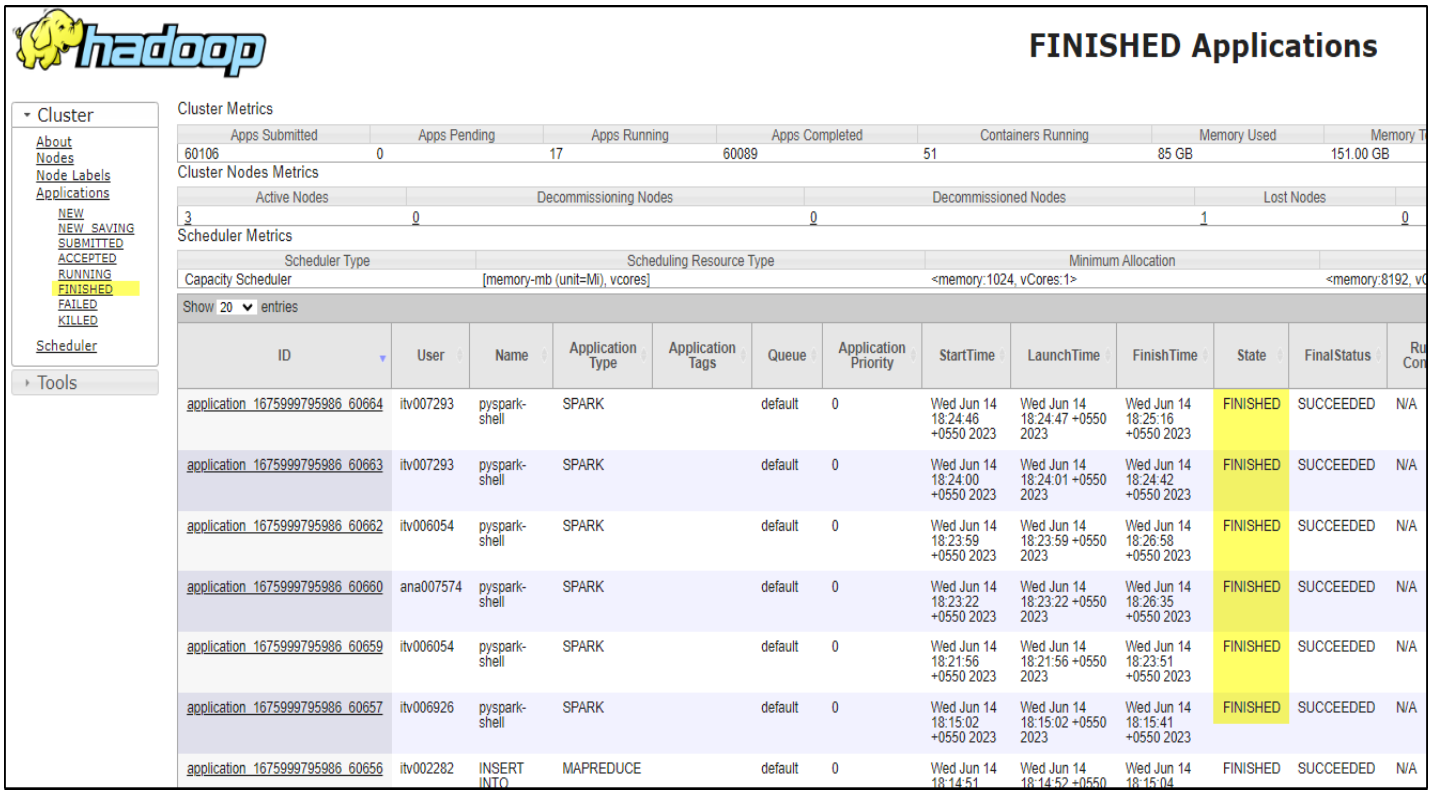
Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes
3	0	0	1

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	Final Stat
application_1675999795986_60674	itv006164	Practice	SPARK		default	0	Wed Jun 14 18:41:10 +0550 2023	Wed Jun 14 18:41:10 +0550 2023	N/A	RUNNING	UNDEFINE
application_1675999795986_60673	itv005355	pyspark-shell	SPARK		default	0	Wed Jun 14 18:39:25 +0550 2023	Wed Jun 14 18:39:26 +0550 2023	N/A	RUNNING	UNDEFINE
application_1675999795986_60672	itv006110	pyspark-shell	SPARK		default	0	Wed Jun 14 18:37:04 +0550 2023	Wed Jun 14 18:37:04 +0550 2023	N/A	RUNNING	UNDEFINE
application_1675999795986_60671	itv007207	pyspark-shell	SPARK		default	0	Wed Jun 14 18:34:17 +0550 2023	Wed Jun 14 18:34:18 +0550 2023	N/A	RUNNING	UNDEFINE
application_1675999795986_60670	itv006826	pyspark-shell	SPARK		default	0	Wed Jun 14 18:32:36 +0550 2023	Wed Jun 14 18:32:36 +0550 2023	N/A	RUNNING	UNDEFINE
application_1675999795986_60669	itv006911	SparkSQL::172.16.1.102	SPARK		default	0	Wed Jun 14 18:28:41 +0550 2023	Wed Jun 14 18:28:41 +0550 2023	N/A	RUNNING	UNDEFINE
application_1675999795986_60668	itv006926	pyspark-shell	SPARK		default	0	Wed Jun 14 18:26:46	Wed Jun 14 18:26:46 +0550	N/A	RUNNING	UNDEFINE

4. In the Application there is FINISHED tab which shows only FINISHED applications.



FINISHED Applications

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory T
60106	0	17	60089	51	85 GB	151.00 GB

Cluster Nodes Metrics

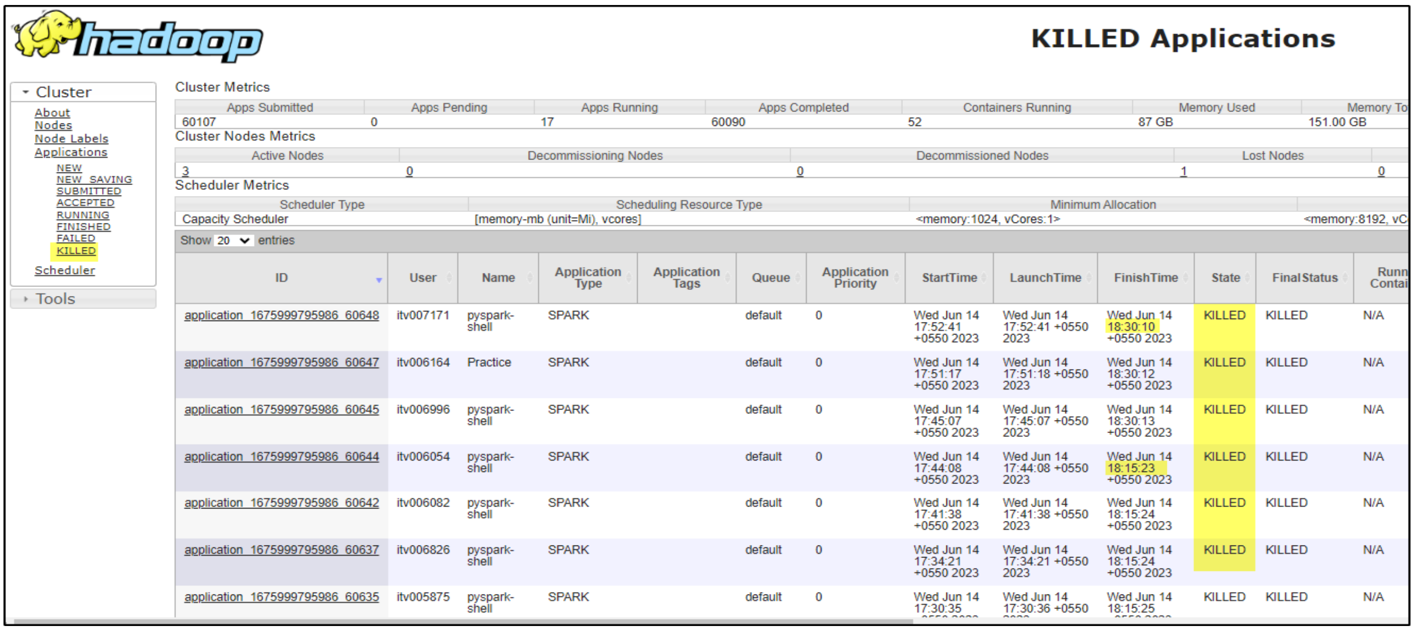
Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	
3	0	0	1	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vC

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Ru Con
application_1675999795986_60664	itv007293	pyspark-shell	SPARK		default	0	Wed Jun 14 18:24:46 +0550 2023	Wed Jun 14 18:24:47 +0550 2023	Wed Jun 14 18:25:16 +0550 2023	FINISHED	SUCCEEDED	N/A
application_1675999795986_60663	itv007293	pyspark-shell	SPARK		default	0	Wed Jun 14 18:24:00 +0550 2023	Wed Jun 14 18:24:01 +0550 2023	Wed Jun 14 18:24:42 +0550 2023	FINISHED	SUCCEEDED	N/A
application_1675999795986_60662	itv006054	pyspark-shell	SPARK		default	0	Wed Jun 14 18:23:59 +0550 2023	Wed Jun 14 18:23:59 +0550 2023	Wed Jun 14 18:26:58 +0550 2023	FINISHED	SUCCEEDED	N/A
application_1675999795986_60660	ana007574	pyspark-shell	SPARK		default	0	Wed Jun 14 18:23:22 +0550 2023	Wed Jun 14 18:23:22 +0550 2023	Wed Jun 14 18:26:35 +0550 2023	FINISHED	SUCCEEDED	N/A
application_1675999795986_60659	itv006054	pyspark-shell	SPARK		default	0	Wed Jun 14 18:21:56 +0550 2023	Wed Jun 14 18:21:56 +0550 2023	Wed Jun 14 18:23:51 +0550 2023	FINISHED	SUCCEEDED	N/A
application_1675999795986_60657	itv006926	pyspark-shell	SPARK		default	0	Wed Jun 14 18:15:02 +0550 2023	Wed Jun 14 18:15:02 +0550 2023	Wed Jun 14 18:15:41 +0550 2023	FINISHED	SUCCEEDED	N/A
application_1675999795986_60656	itv002282	INSERT INTO	MAPREDUCE		default	0	Wed Jun 14 18:14:51	Wed Jun 14 18:14:52 +0550	Wed Jun 14 18:15:04	FINISHED	SUCCEEDED	N/A

5. In the Application there is KILLED tab which shows only KILLED applications.



Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory To
60107	0	17	60090	52	87 GB	151.00 GB

Cluster Nodes Metrics

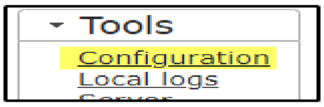
Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	
3	0	0	1	0

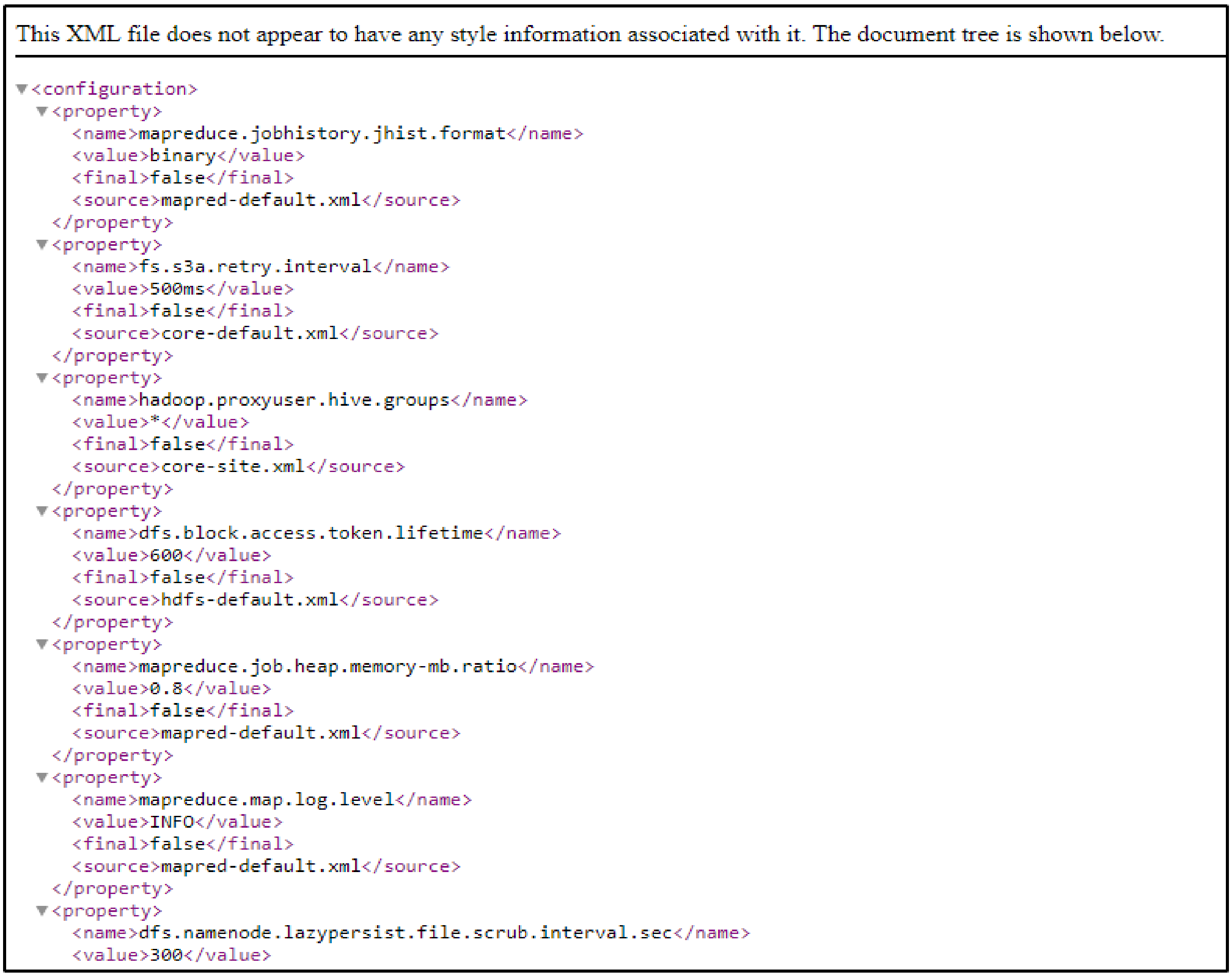
Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vC

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Runn Contai
application_1675999795986_60648	itv007171	pyspark-shell	SPARK		default	0	Wed Jun 14 17:52:41 +0550 2023	Wed Jun 14 17:52:41 +0550 2023	Wed Jun 14 18:30:10 +0550 2023	KILLED	KILLED	N/A
application_1675999795986_60647	itv006164	Practice	SPARK		default	0	Wed Jun 14 17:51:17 +0550 2023	Wed Jun 14 17:51:18 +0550 2023	Wed Jun 14 18:30:12 +0550 2023	KILLED	KILLED	N/A
application_1675999795986_60645	itv006996	pyspark-shell	SPARK		default	0	Wed Jun 14 17:45:07 +0550 2023	Wed Jun 14 17:45:07 +0550 2023	Wed Jun 14 18:30:13 +0550 2023	KILLED	KILLED	N/A
application_1675999795986_60644	itv006054	pyspark-shell	SPARK		default	0	Wed Jun 14 17:44:08 +0550 2023	Wed Jun 14 17:44:08 +0550 2023	Wed Jun 14 18:15:23 +0550 2023	KILLED	KILLED	N/A
application_1675999795986_60642	itv006082	pyspark-shell	SPARK		default	0	Wed Jun 14 17:41:38 +0550 2023	Wed Jun 14 17:41:38 +0550 2023	Wed Jun 14 18:15:24 +0550 2023	KILLED	KILLED	N/A
application_1675999795986_60637	itv006826	pyspark-shell	SPARK		default	0	Wed Jun 14 17:34:21 +0550 2023	Wed Jun 14 17:34:21 +0550 2023	Wed Jun 14 18:15:24 +0550 2023	KILLED	KILLED	N/A
application_1675999795986_60635	itv005875	pyspark-shell	SPARK		default	0	Wed Jun 14 17:30:35	Wed Jun 14 17:30:36 +0550	Wed Jun 14 18:15:25	KILLED	KILLED	N/A

6. In Tools there is Configuration tab which shows all the configuration of the cluster.

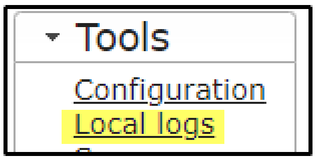


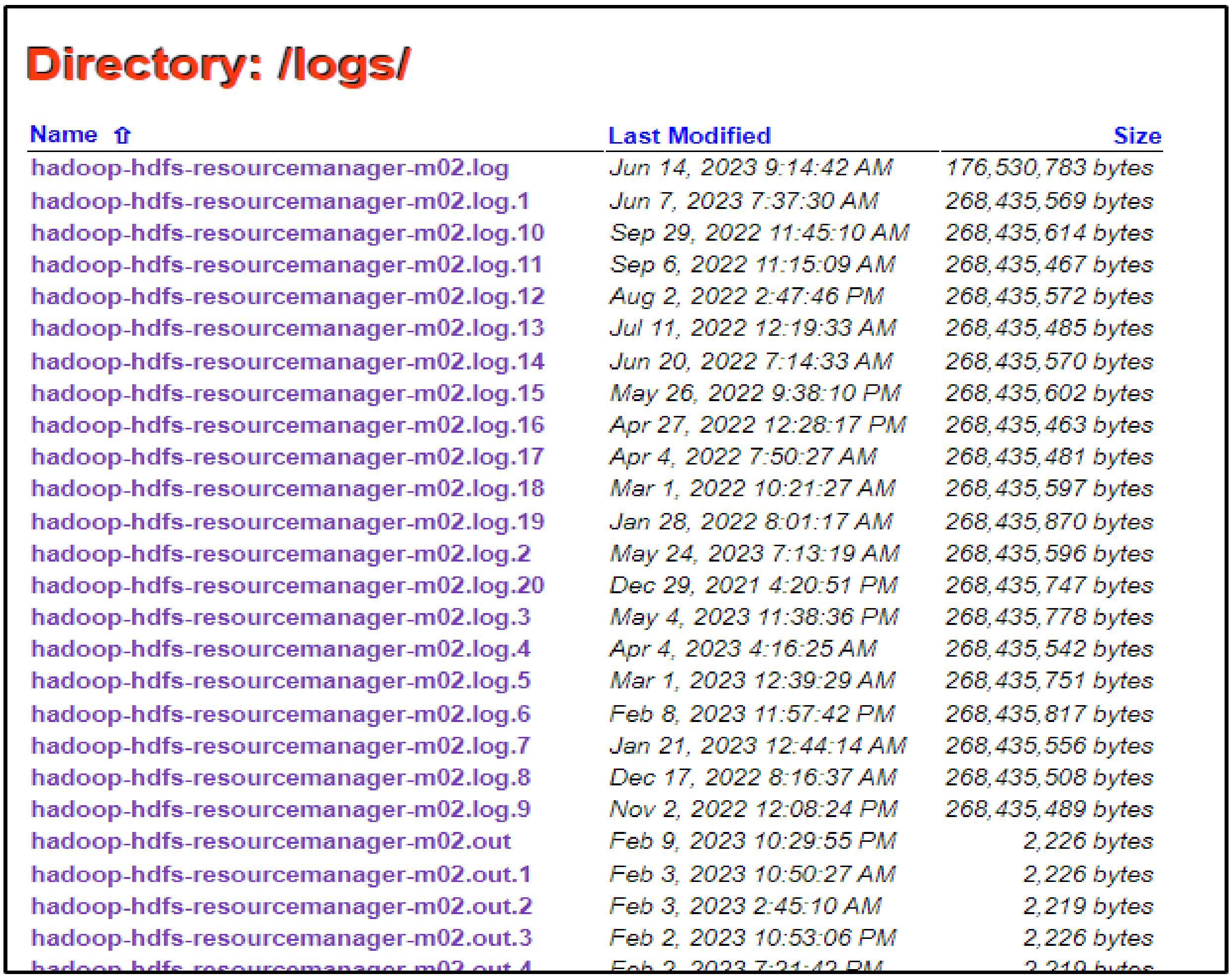


This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<configuration>
  ▼<property>
     <name>mapreduce.jobhistory.jhist.format</name>
     <value>binary</value>
     <final>false</final>
     <source>mapred-default.xml</source>
   </property>
  ▼<property>
     <name>fs.s3a.retry.interval</name>
     <value>500ms</value>
     <final>false</final>
     <source>core-default.xml</source>
   </property>
  ▼<property>
     <name>hadoop.proxyuser.hive.groups</name>
     <value>*</value>
     <final>false</final>
     <source>core-site.xml</source>
   </property>
  ▼<property>
     <name>dfs.block.access.token.lifetime</name>
     <value>600</value>
     <final>false</final>
     <source>hdfs-default.xml</source>
   </property>
  ▼<property>
     <name>mapreduce.job.heap.memory-mb.ratio</name>
     <value>0.8</value>
     <final>false</final>
     <source>mapred-default.xml</source>
   </property>
  ▼<property>
     <name>mapreduce.map.log.level</name>
     <value>INFO</value>
     <final>false</final>
     <source>mapred-default.xml</source>
   </property>
  ▼<property>
     <name>dfs.namenode.lazypersist.file.scrub.interval.sec</name>
     <value>300</value>
```

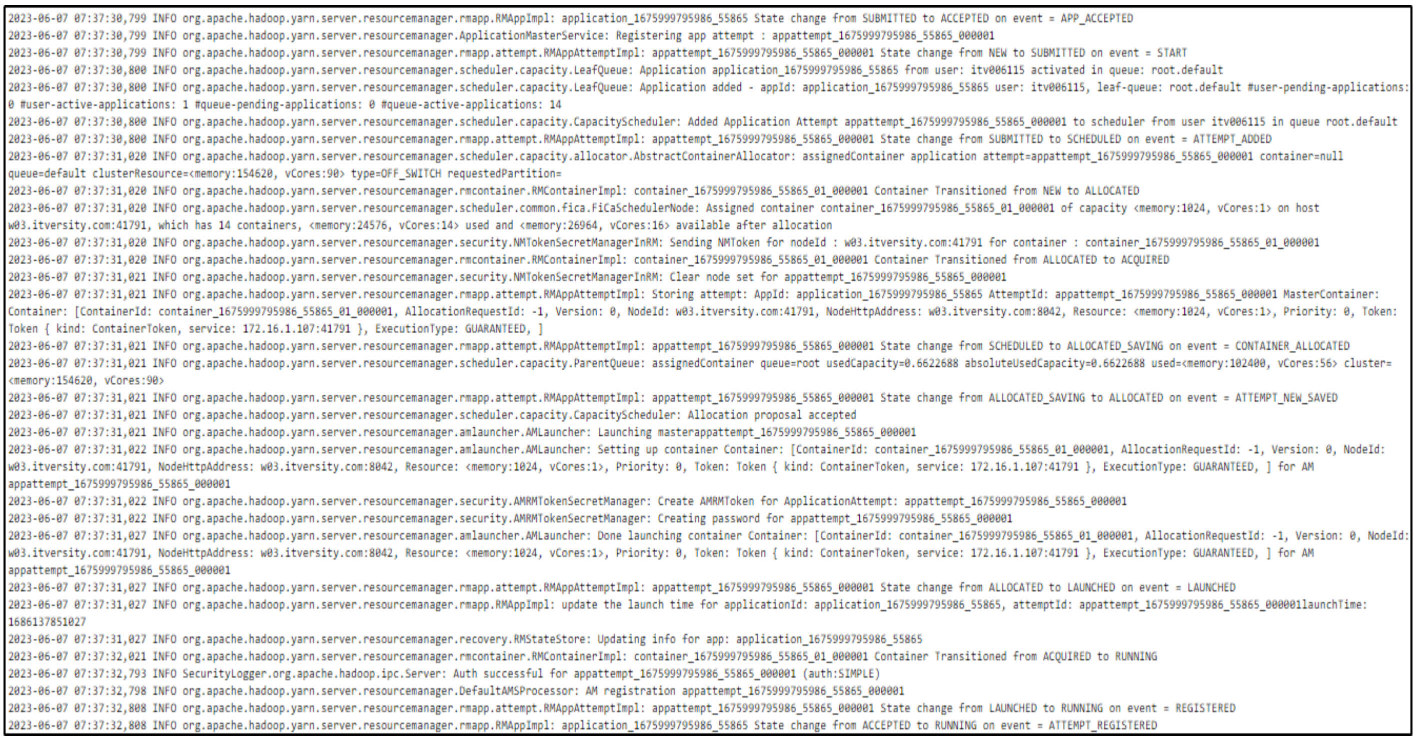
7. In Tools there is Local logs tab which shows all the logs of the cluster.





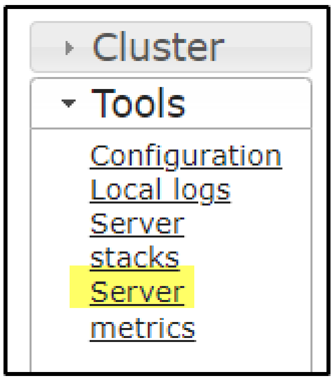
Directory: /logs/

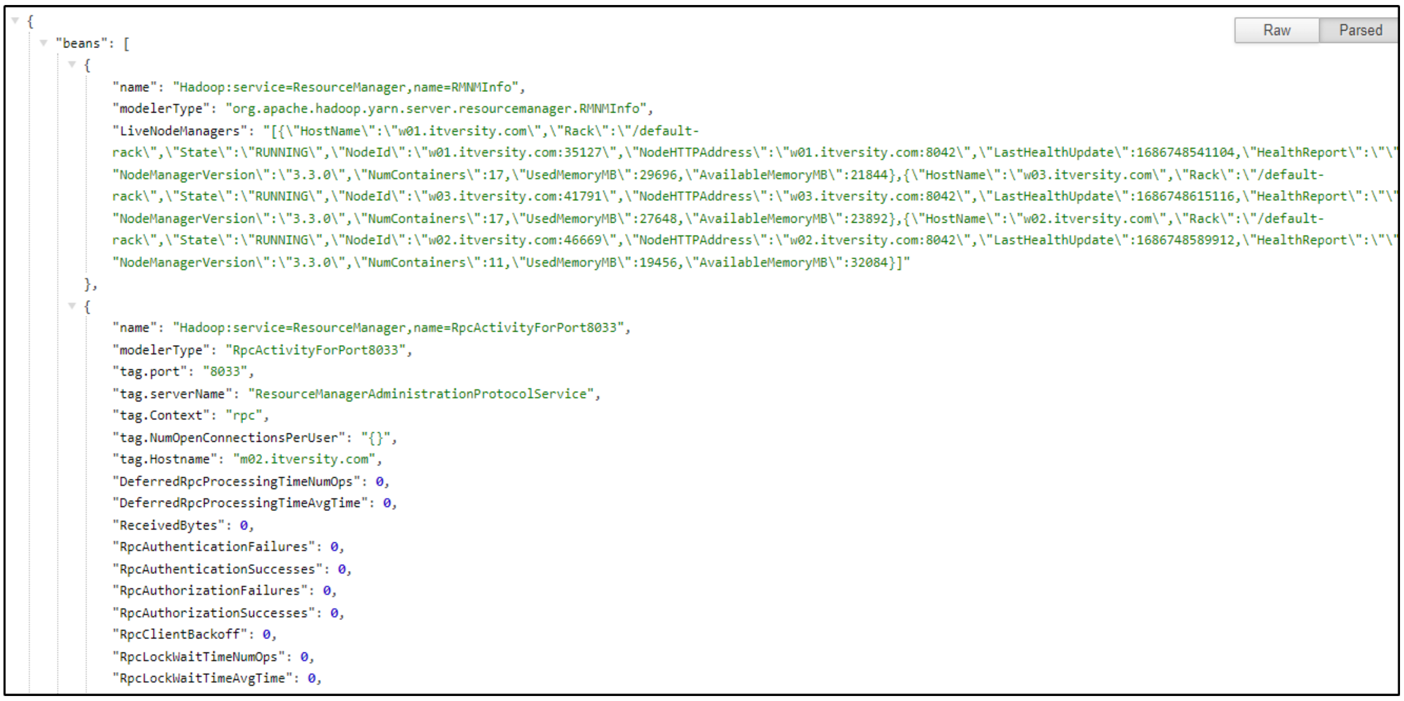
Name ⇧	Last Modified	Size
hadoop-hdfs-resourcemanager-m02.log	Jun 14, 2023 9:14:42 AM	176,530,783 bytes
hadoop-hdfs-resourcemanager-m02.log.1	Jun 7, 2023 7:37:30 AM	268,435,569 bytes
hadoop-hdfs-resourcemanager-m02.log.10	Sep 29, 2022 11:45:10 AM	268,435,614 bytes
hadoop-hdfs-resourcemanager-m02.log.11	Sep 6, 2022 11:15:09 AM	268,435,467 bytes
hadoop-hdfs-resourcemanager-m02.log.12	Aug 2, 2022 2:47:46 PM	268,435,572 bytes
hadoop-hdfs-resourcemanager-m02.log.13	Jul 11, 2022 12:19:33 AM	268,435,485 bytes
hadoop-hdfs-resourcemanager-m02.log.14	Jun 20, 2022 7:14:33 AM	268,435,570 bytes
hadoop-hdfs-resourcemanager-m02.log.15	May 26, 2022 9:38:10 PM	268,435,602 bytes
hadoop-hdfs-resourcemanager-m02.log.16	Apr 27, 2022 12:28:17 PM	268,435,463 bytes
hadoop-hdfs-resourcemanager-m02.log.17	Apr 4, 2022 7:50:27 AM	268,435,481 bytes
hadoop-hdfs-resourcemanager-m02.log.18	Mar 1, 2022 10:21:27 AM	268,435,597 bytes
hadoop-hdfs-resourcemanager-m02.log.19	Jan 28, 2022 8:01:17 AM	268,435,870 bytes
hadoop-hdfs-resourcemanager-m02.log.2	May 24, 2023 7:13:19 AM	268,435,596 bytes
hadoop-hdfs-resourcemanager-m02.log.20	Dec 29, 2021 4:20:51 PM	268,435,747 bytes
hadoop-hdfs-resourcemanager-m02.log.3	May 4, 2023 11:38:36 PM	268,435,778 bytes
hadoop-hdfs-resourcemanager-m02.log.4	Apr 4, 2023 4:16:25 AM	268,435,542 bytes
hadoop-hdfs-resourcemanager-m02.log.5	Mar 1, 2023 12:39:29 AM	268,435,751 bytes
hadoop-hdfs-resourcemanager-m02.log.6	Feb 8, 2023 11:57:42 PM	268,435,817 bytes
hadoop-hdfs-resourcemanager-m02.log.7	Jan 21, 2023 12:44:14 AM	268,435,556 bytes
hadoop-hdfs-resourcemanager-m02.log.8	Dec 17, 2022 8:16:37 AM	268,435,508 bytes
hadoop-hdfs-resourcemanager-m02.log.9	Nov 2, 2022 12:08:24 PM	268,435,489 bytes
hadoop-hdfs-resourcemanager-m02.out	Feb 9, 2023 10:29:55 PM	2,226 bytes
hadoop-hdfs-resourcemanager-m02.out.1	Feb 3, 2023 10:50:27 AM	2,226 bytes
hadoop-hdfs-resourcemanager-m02.out.2	Feb 3, 2023 2:45:10 AM	2,219 bytes
hadoop-hdfs-resourcemanager-m02.out.3	Feb 2, 2023 10:53:06 PM	2,226 bytes
hadoop-hdfs-resourcemanager-m02.out.4	Feb 2, 2023 7:21:42 PM	2,219 bytes



```
2023-06-07 07:37:30,799 INFO org.apache.hadoop.yarn.server.resourcemanager.rmapp.RMAppImpl: application_1675999795986_55865 State change from SUBMITTED to ACCEPTED on event = APP_ACCEPTED
2023-06-07 07:37:30,799 INFO org.apache.hadoop.yarn.server.resourcemanager.ApplicationMasterService: Registering app attempt : appattempt_1675999795986_55865_000001
2023-06-07 07:37:30,799 INFO org.apache.hadoop.yarn.server.resourcemanager.rmapp.attempt.RMAppAttemptImpl: appattempt_1675999795986_55865_000001 State change from NEW to SUBMITTED on event = START
2023-06-07 07:37:30,800 INFO org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.LeafQueue: Application application_1675999795986_55865 from user: itv006115 activated in queue: root.default
2023-06-07 07:37:30,800 INFO org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.LeafQueue: Application added - appId: application_1675999795986_55865 user: itv006115, leaf-queue: root.default #user-pending-applications:
0 #user-active-applications: 1 #queue-pending-applications: 0 #queue-active-applications: 14
2023-06-07 07:37:30,800 INFO org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler: Added Application Attempt appattempt_1675999795986_55865_000001 to scheduler from user itv006115 in queue root.default
2023-06-07 07:37:30,800 INFO org.apache.hadoop.yarn.server.resourcemanager.rmapp.attempt.RMAppAttemptImpl: appattempt_1675999795986_55865_000001 State change from SUBMITTED to SCHEDULED on event = ATTEMPT_ADDED
2023-06-07 07:37:31,020 INFO org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.allocator.AbstractContainerAllocator: assignedContainer application attempt=appattempt_1675999795986_55865_000001 container=null
queue=default clusterResource=<memory:154620, vCores:90> type=OFF_SWITCH requestedPartition=
2023-06-07 07:37:31,020 INFO org.apache.hadoop.yarn.server.resourcemanager.rmcontainer.RMContainerImpl: container_1675999795986_55865_01_000001 Container Transitioned from NEW to ALLOCATED
2023-06-07 07:37:31,020 INFO org.apache.hadoop.yarn.server.resourcemanager.scheduler.common.fica.FiCaSchedulerNode: Assigned container container_1675999795986_55865_01_000001 of capacity <memory:1024, vCores:1> on host
w03.itversity.com:41791, which has 14 containers, <memory:24576, vCores:14> used and <memory:26964, vCores:16> available after allocation
2023-06-07 07:37:31,020 INFO org.apache.hadoop.yarn.server.resourcemanager.security.NMTokenSecretManagerInRM: Sending NMToken for nodeId : w03.itversity.com:41791 for container : container_1675999795986_55865_01_000001
2023-06-07 07:37:31,020 INFO org.apache.hadoop.yarn.server.resourcemanager.rmcontainer.RMContainerImpl: container_1675999795986_55865_01_000001 Container Transitioned from ALLOCATED to ACQUIRED
2023-06-07 07:37:31,021 INFO org.apache.hadoop.yarn.server.resourcemanager.security.NMTokenSecretManagerInRM: Clear node set for appattempt_1675999795986_55865_000001
2023-06-07 07:37:31,021 INFO org.apache.hadoop.yarn.server.resourcemanager.rmapp.attempt.RMAppAttemptImpl: Storing attempt: AppId: application_1675999795986_55865 AttemptId: appattempt_1675999795986_55865_000001 MasterContainer:
Container: [ContainerId: container_1675999795986_55865_01_000001, AllocationRequestId: -1, Version: 0, NodeId: w03.itversity.com:41791, NodeHttpAddress: w03.itversity.com:8042, Resource: <memory:1024, vCores:1>, Priority: 0, Token:
Token { kind: ContainerToken, service: 172.16.1.107:41791 }, ExecutionType: GUARANTEED, ]
2023-06-07 07:37:31,021 INFO org.apache.hadoop.yarn.server.resourcemanager.rmapp.attempt.RMAppAttemptImpl: appattempt_1675999795986_55865_000001 State change from SCHEDULED to ALLOCATED_SAVING on event = CONTAINER_ALLOCATED
2023-06-07 07:37:31,021 INFO org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.ParentQueue: assignedContainer queue=root usedCapacity=0.6622688 absoluteUsedCapacity=0.6622688 used=<memory:102400, vCores:56> cluster=
<memory:154620, vCores:90>
2023-06-07 07:37:31,021 INFO org.apache.hadoop.yarn.server.resourcemanager.rmapp.attempt.RMAppAttemptImpl: appattempt_1675999795986_55865_000001 State change from ALLOCATED_SAVING to ALLOCATED on event = ATTEMPT_NEW_SAVED
2023-06-07 07:37:31,021 INFO org.apache.hadoop.yarn.server.resourcemanager.scheduler.capacity.CapacityScheduler: Allocation proposal accepted
2023-06-07 07:37:31,021 INFO org.apache.hadoop.yarn.server.resourcemanager.amlauncher.AMLauncher: Launching masterappattempt_1675999795986_55865_000001
2023-06-07 07:37:31,022 INFO org.apache.hadoop.yarn.server.resourcemanager.amlauncher.AMLauncher: Setting up container Container: [ContainerId: container_1675999795986_55865_01_000001, AllocationRequestId: -1, Version: 0, NodeId:
w03.itversity.com:41791, NodeHttpAddress: w03.itversity.com:8042, Resource: <memory:1024, vCores:1>, Priority: 0, Token: Token { kind: ContainerToken, service: 172.16.1.107:41791 }, ExecutionType: GUARANTEED, ] for AM
appattempt_1675999795986_55865_000001
2023-06-07 07:37:31,022 INFO org.apache.hadoop.yarn.server.resourcemanager.security.AMRMTokenSecretManager: Create AMRMToken for ApplicationAttempt: appattempt_1675999795986_55865_000001
2023-06-07 07:37:31,022 INFO org.apache.hadoop.yarn.server.resourcemanager.security.AMRMTokenSecretManager: Creating password for appattempt_1675999795986_55865_000001
2023-06-07 07:37:31,027 INFO org.apache.hadoop.yarn.server.resourcemanager.amlauncher.AMLauncher: Done launching container Container: [ContainerId: container_1675999795986_55865_01_000001, AllocationRequestId: -1, Version: 0, NodeId:
w03.itversity.com:41791, NodeHttpAddress: w03.itversity.com:8042, Resource: <memory:1024, vCores:1>, Priority: 0, Token: Token { kind: ContainerToken, service: 172.16.1.107:41791 }, ExecutionType: GUARANTEED, ] for AM
appattempt_1675999795986_55865_000001
2023-06-07 07:37:31,027 INFO org.apache.hadoop.yarn.server.resourcemanager.rmapp.attempt.RMAppAttemptImpl: appattempt_1675999795986_55865_000001 State change from ALLOCATED to LAUNCHED on event = LAUNCHED
2023-06-07 07:37:31,027 INFO org.apache.hadoop.yarn.server.resourcemanager.rmapp.RMAppImpl: update the launch time for applicationId: application_1675999795986_55865, attemptId: appattempt_1675999795986_55865_000001launchTime:
1686137851027
2023-06-07 07:37:31,027 INFO org.apache.hadoop.yarn.server.resourcemanager.recovery.RMStateStore: Updating info for app: application_1675999795986_55865
2023-06-07 07:37:32,021 INFO org.apache.hadoop.yarn.server.resourcemanager.rmcontainer.RMContainerImpl: container_1675999795986_55865_01_000001 Container Transitioned from ACQUIRED to RUNNING
2023-06-07 07:37:32,793 INFO SecurityLogger.org.apache.hadoop.ipc.Server: Auth successful for appattempt_1675999795986_55865_000001 (auth:SIMPLE)
2023-06-07 07:37:32,798 INFO org.apache.hadoop.yarn.server.resourcemanager.DefaultAMSProcessor: AM registration appattempt_1675999795986_55865_000001
2023-06-07 07:37:32,808 INFO org.apache.hadoop.yarn.server.resourcemanager.rmapp.attempt.RMAppAttemptImpl: appattempt_1675999795986_55865_000001 State change from LAUNCHED to RUNNING on event = REGISTERED
2023-06-07 07:37:32,808 INFO org.apache.hadoop.yarn.server.resourcemanager.rmapp.RMAppImpl: application_1675999795986_55865 State change from ACCEPTED to RUNNING on event = ATTEMPT_REGISTERED
```

8. In Tools there is Server tab which shows server details of the cluster in json format.



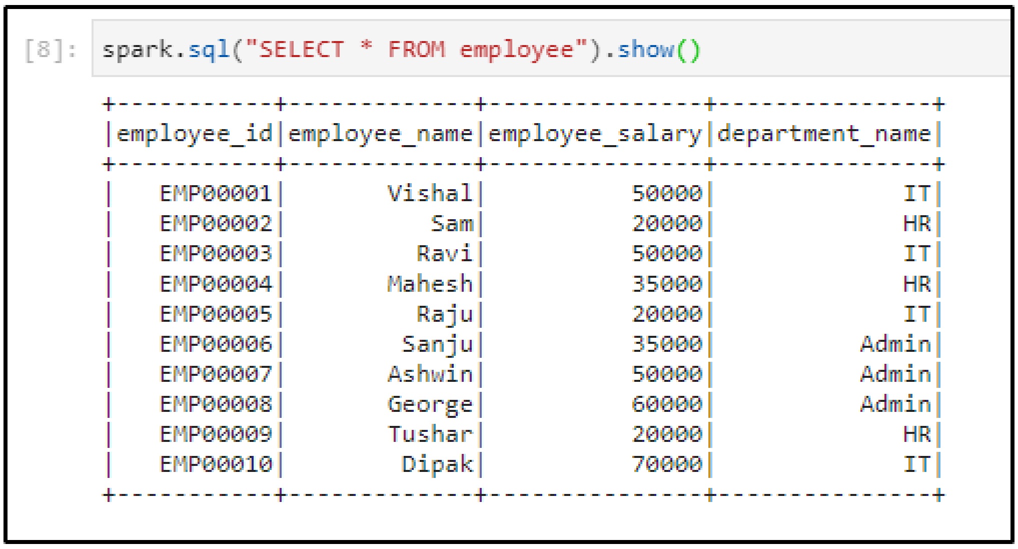


```
{
  "beans": [
    {
      "name": "Hadoop:service=ResourceManager,name=RMNMInfo",
      "modelerType": "org.apache.hadoop.yarn.server.resourcemanager.RMNMInfo",
      "LiveNodeManagers": "[{\"HostName\":\"w01.itversity.com\",\"Rack\":\"/default-
      rack\",\"State\":\"RUNNING\",\"NodeId\":\"w01.itversity.com:35127\",\"NodeHTTPAddress\":\"w01.itversity.com:8042\",\"LastHealthUpdate\":1686748541104,\"HealthReport\":\"\"
      "NodeManagerVersion\":\"3.3.0\",\"NumContainers\":17,\"UsedMemoryMB\":29696,\"AvailableMemoryMB\":21844},{\"HostName\":\"w03.itversity.com\",\"Rack\":\"/default-
      rack\",\"State\":\"RUNNING\",\"NodeId\":\"w03.itversity.com:41791\",\"NodeHTTPAddress\":\"w03.itversity.com:8042\",\"LastHealthUpdate\":1686748615116,\"HealthReport\":\"\"
      "NodeManagerVersion\":\"3.3.0\",\"NumContainers\":17,\"UsedMemoryMB\":27648,\"AvailableMemoryMB\":23892},{\"HostName\":\"w02.itversity.com\",\"Rack\":\"/default-
      rack\",\"State\":\"RUNNING\",\"NodeId\":\"w02.itversity.com:46669\",\"NodeHTTPAddress\":\"w02.itversity.com:8042\",\"LastHealthUpdate\":1686748589912,\"HealthReport\":\"\"
      "NodeManagerVersion\":\"3.3.0\",\"NumContainers\":11,\"UsedMemoryMB\":19456,\"AvailableMemoryMB\":32084}]"
    },
    {
      "name": "Hadoop:service=ResourceManager,name=RpcActivityForPort8033",
      "modelerType": "RpcActivityForPort8033",
      "tag.port": "8033",
      "tag.serverName": "ResourceManagerAdministrationProtocolService",
      "tag.Context": "rpc",
      "tag.NumOpenConnectionsPerUser": "{}",
      "tag.Hostname": "m02.itversity.com",
      "DeferredRpcProcessingTimeNumOps": 0,
      "DeferredRpcProcessingTimeAvgTime": 0,
      "ReceivedBytes": 0,
      "RpcAuthenticationFailures": 0,
      "RpcAuthenticationSuccesses": 0,
      "RpcAuthorizationFailures": 0,
      "RpcAuthorizationSuccesses": 0,
      "RpcClientBackoff": 0,
      "RpcLockWaitTimeNumOps": 0,
      "RpcLockWaitTimeAvgTime": 0,
```

Question 3.

Perform various queries

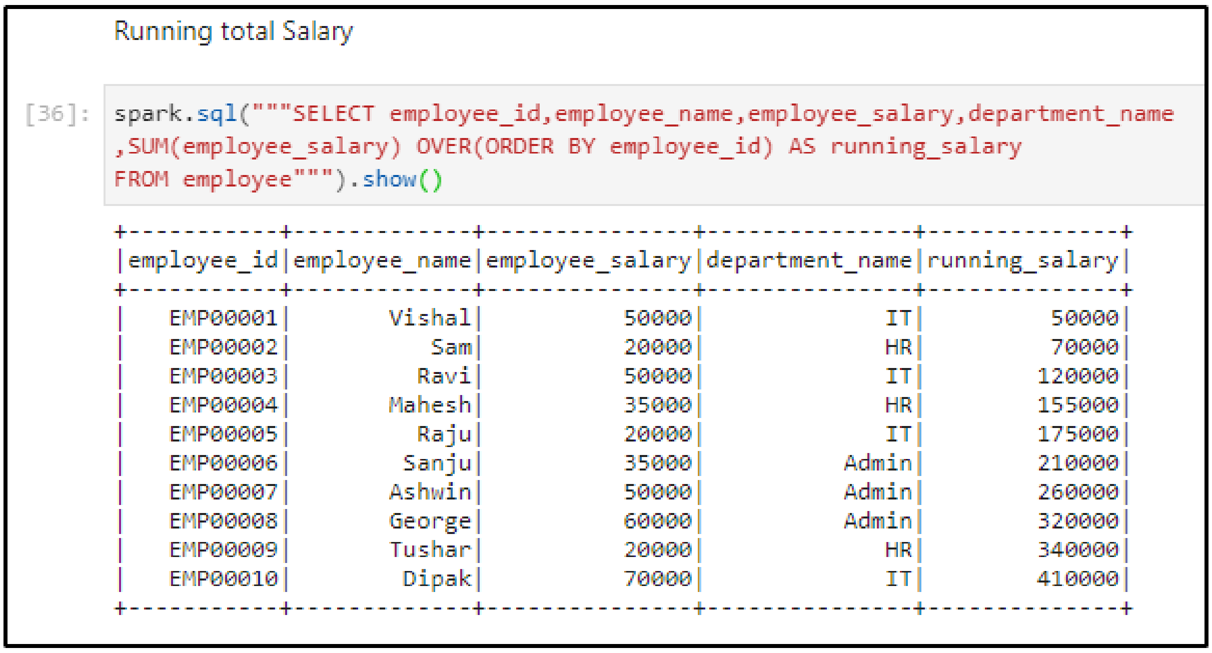
Ans: - I have choose below employee dataset



```
[8]: spark.sql("SELECT * FROM employee").show()

+-----------+-------------+---------------+---------------+
|employee_id|employee_name|employee_salary|department_name|
+-----------+-------------+---------------+---------------+
|   EMP00001|       Vishal|          50000|             IT|
|   EMP00002|          Sam|          20000|             HR|
|   EMP00003|         Ravi|          50000|             IT|
|   EMP00004|       Mahesh|          35000|             HR|
|   EMP00005|         Raju|          20000|             IT|
|   EMP00006|        Sanju|          35000|          Admin|
|   EMP00007|       Ashwin|          50000|          Admin|
|   EMP00008|       George|          60000|          Admin|
|   EMP00009|       Tushar|          20000|             HR|
|   EMP00010|        Dipak|          70000|             IT|
+-----------+-------------+---------------+---------------+
```

1. Running salary of employee

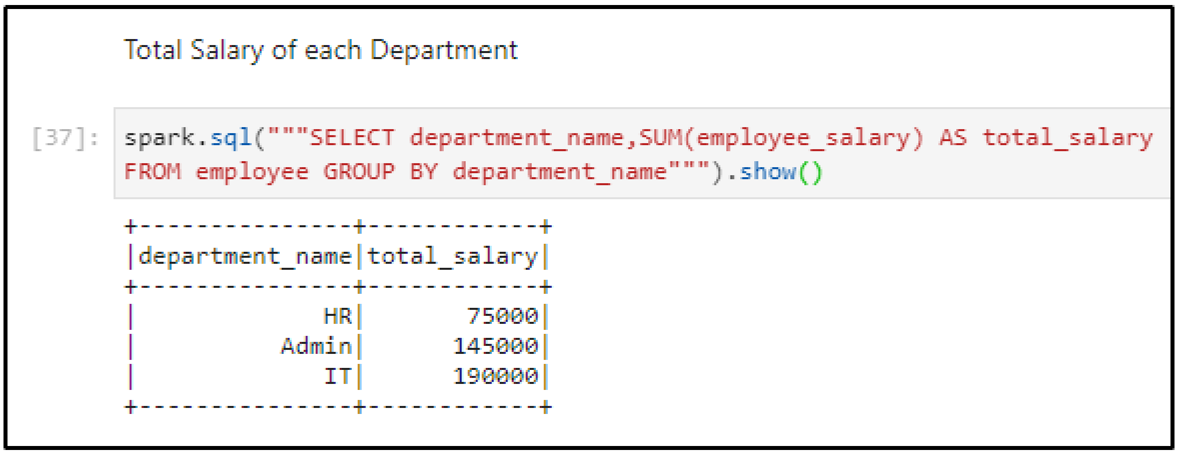


Running total Salary

[36]:
```
spark.sql("""SELECT employee_id,employee_name,employee_salary,department_name
,SUM(employee_salary) OVER(ORDER BY employee_id) AS running_salary
FROM employee""").show()
```

```
+-----------+-------------+---------------+---------------+--------------+
|employee_id|employee_name|employee_salary|department_name|running_salary|
+-----------+-------------+---------------+---------------+--------------+
|   EMP00001|       Vishal|          50000|             IT|         50000|
|   EMP00002|          Sam|          20000|             HR|         70000|
|   EMP00003|         Ravi|          50000|             IT|        120000|
|   EMP00004|       Mahesh|          35000|             HR|        155000|
|   EMP00005|         Raju|          20000|             IT|        175000|
|   EMP00006|        Sanju|          35000|          Admin|        210000|
|   EMP00007|       Ashwin|          50000|          Admin|        260000|
|   EMP00008|       George|          60000|          Admin|        320000|
|   EMP00009|       Tushar|          20000|             HR|        340000|
|   EMP00010|        Dipak|          70000|             IT|        410000|
+-----------+-------------+---------------+---------------+--------------+
```

2. Total salary of each department

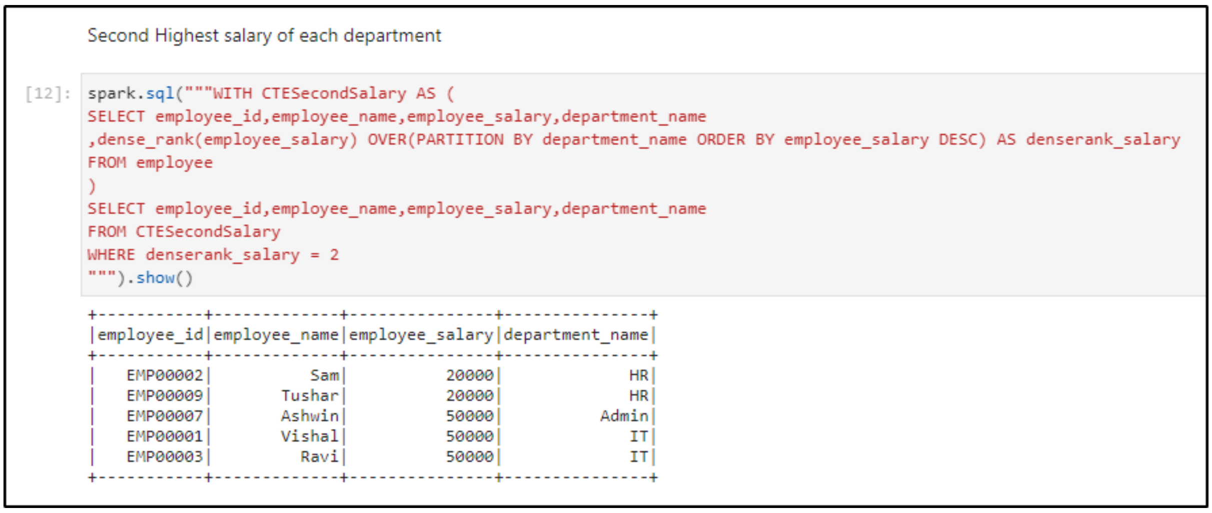


Total Salary of each Department

[37]:
```
spark.sql("""SELECT department_name,SUM(employee_salary) AS total_salary
FROM employee GROUP BY department_name""").show()
```

```
+---------------+------------+
|department_name|total_salary|
+---------------+------------+
|             HR|       75000|
|          Admin|      145000|
|             IT|      190000|
+---------------+------------+
```

3. Second Highest salary of each department

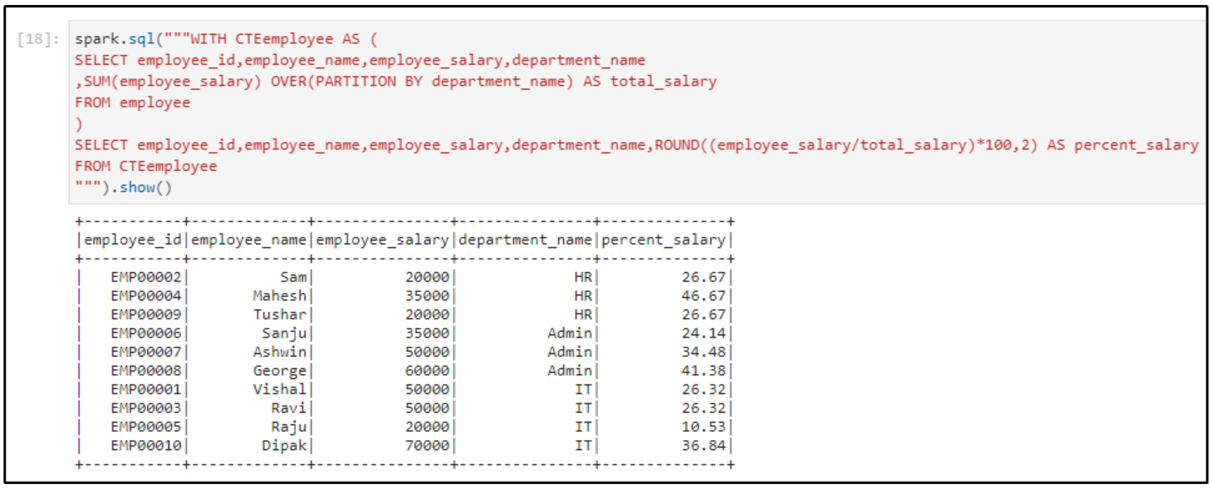


Second Highest salary of each department

[12]:
```
spark.sql("""WITH CTESecondSalary AS (
SELECT employee_id,employee_name,employee_salary,department_name
,dense_rank(employee_salary) OVER(PARTITION BY department_name ORDER BY employee_salary DESC) AS denserank_salary
FROM employee
)
SELECT employee_id,employee_name,employee_salary,department_name
FROM CTESecondSalary
WHERE denserank_salary = 2
""").show()
```

```
+-----------+-------------+---------------+---------------+
|employee_id|employee_name|employee_salary|department_name|
+-----------+-------------+---------------+---------------+
|   EMP00002|          Sam|          20000|             HR|
|   EMP00009|       Tushar|          20000|             HR|
|   EMP00007|       Ashwin|          50000|          Admin|
|   EMP00001|       Vishal|          50000|             IT|
|   EMP00003|         Ravi|          50000|             IT|
+-----------+-------------+---------------+---------------+
```

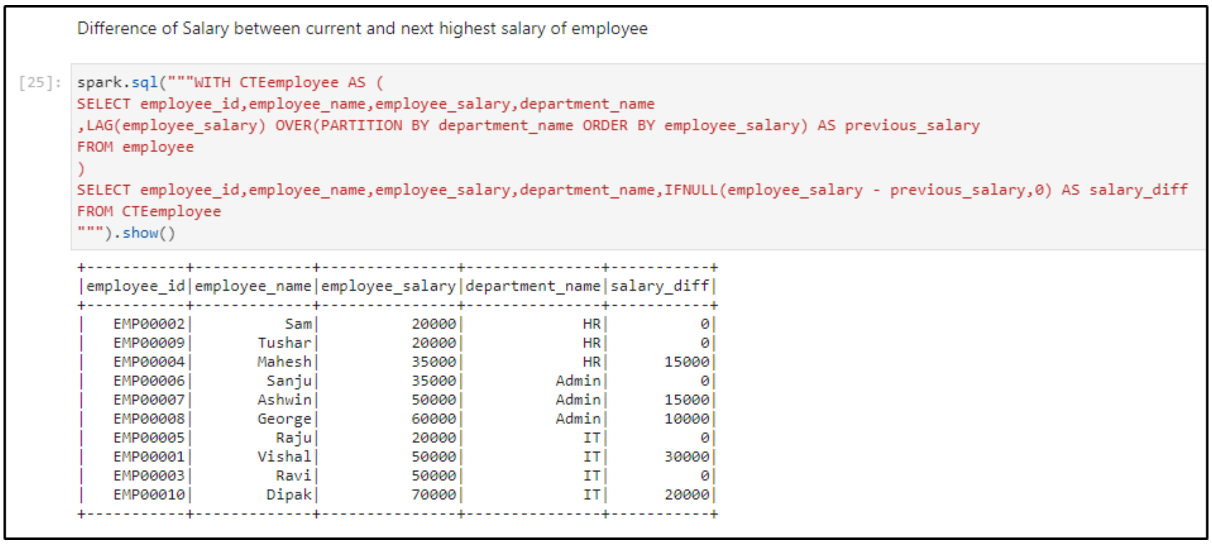
4. Percentage salary of each employee by department



```
[18]: spark.sql("""WITH CTEemployee AS (
SELECT employee_id,employee_name,employee_salary,department_name
,SUM(employee_salary) OVER(PARTITION BY department_name) AS total_salary
FROM employee
)
SELECT employee_id,employee_name,employee_salary,department_name,ROUND((employee_salary/total_salary)*100,2) AS percent_salary
FROM CTEemployee
""").show()
```

```
+-----------+-------------+---------------+---------------+--------------+
|employee_id|employee_name|employee_salary|department_name|percent_salary|
+-----------+-------------+---------------+---------------+--------------+
|   EMP00002|          Sam|          20000|             HR|         26.67|
|   EMP00004|       Mahesh|          35000|             HR|         46.67|
|   EMP00009|       Tushar|          20000|             HR|         26.67|
|   EMP00006|        Sanju|          35000|          Admin|         24.14|
|   EMP00007|       Ashwin|          50000|          Admin|         34.48|
|   EMP00008|       George|          60000|          Admin|         41.38|
|   EMP00001|       Vishal|          50000|             IT|         26.32|
|   EMP00003|         Ravi|          50000|             IT|         26.32|
|   EMP00005|         Raju|          20000|             IT|         10.53|
|   EMP00010|        Dipak|          70000|             IT|         36.84|
+-----------+-------------+---------------+---------------+--------------+
```

5. Difference of Salary between current and next highest salary of employee

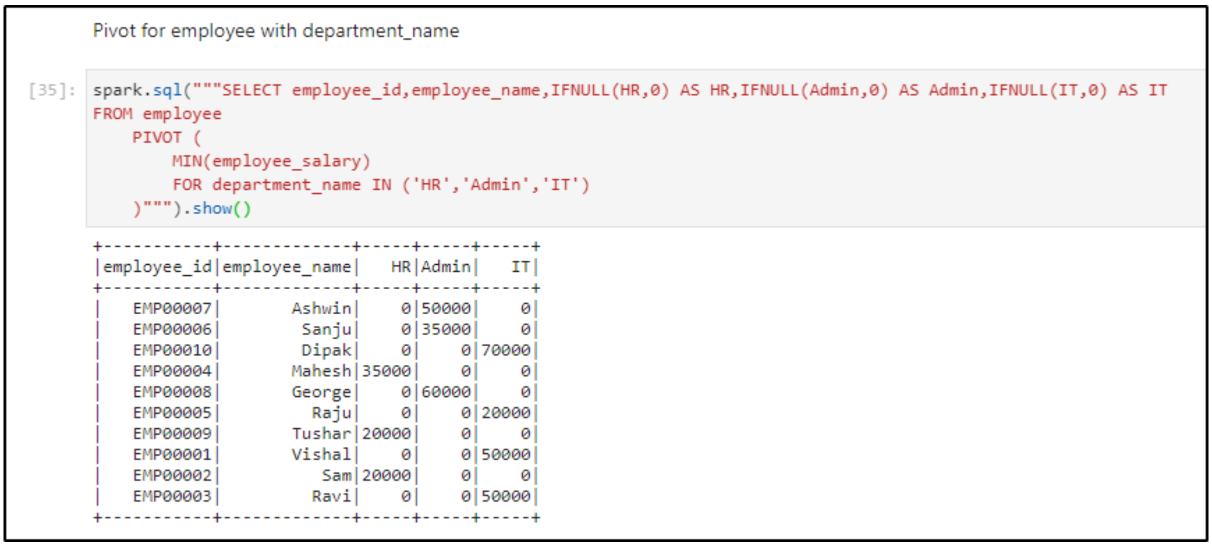


Difference of Salary between current and next highest salary of employee

```
[25]: spark.sql("""WITH CTEemployee AS (
SELECT employee_id,employee_name,employee_salary,department_name
,LAG(employee_salary) OVER(PARTITION BY department_name ORDER BY employee_salary) AS previous_salary
FROM employee
)
SELECT employee_id,employee_name,employee_salary,department_name,IFNULL(employee_salary - previous_salary,0) AS salary_diff
FROM CTEemployee
""").show()
```

```
+-----------+-------------+---------------+---------------+-----------+
|employee_id|employee_name|employee_salary|department_name|salary_diff|
+-----------+-------------+---------------+---------------+-----------+
|   EMP00002|          Sam|          20000|             HR|          0|
|   EMP00009|       Tushar|          20000|             HR|          0|
|   EMP00004|       Mahesh|          35000|             HR|      15000|
|   EMP00006|        Sanju|          35000|          Admin|          0|
|   EMP00007|       Ashwin|          50000|          Admin|      15000|
|   EMP00008|       George|          60000|          Admin|      10000|
|   EMP00005|         Raju|          20000|             IT|          0|
|   EMP00001|       Vishal|          50000|             IT|      30000|
|   EMP00003|         Ravi|          50000|             IT|          0|
|   EMP00010|        Dipak|          70000|             IT|      20000|
+-----------+-------------+---------------+---------------+-----------+
```

6. Pivot for employee with department



Pivot for employee with department_name

```
[35]: spark.sql("""SELECT employee_id,employee_name,IFNULL(HR,0) AS HR,IFNULL(Admin,0) AS Admin,IFNULL(IT,0) AS IT
FROM employee
    PIVOT (
        MIN(employee_salary)
        FOR department_name IN ('HR','Admin','IT')
    )""").show()
```

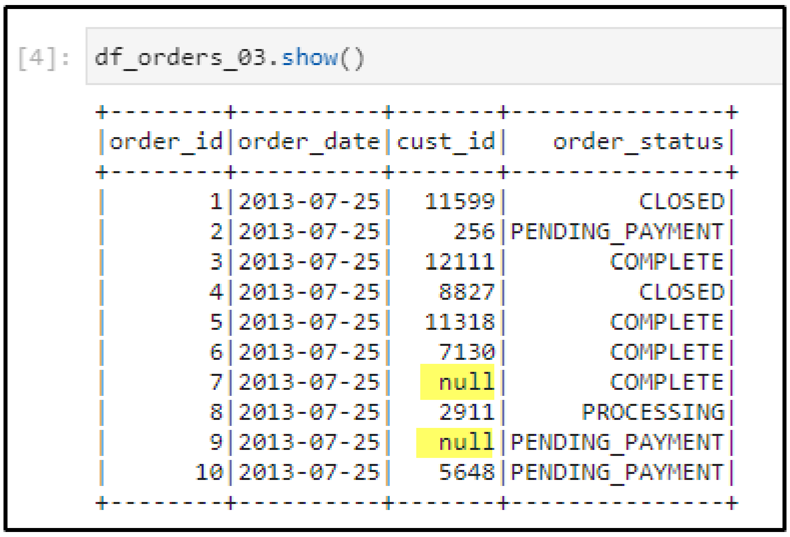
```
+-----------+-------------+-----+-----+-----+
|employee_id|employee_name|   HR|Admin|   IT|
+-----------+-------------+-----+-----+-----+
|   EMP00007|       Ashwin|    0|50000|    0|
|   EMP00006|        Sanju|    0|35000|    0|
|   EMP00010|        Dipak|    0|    0|70000|
|   EMP00004|       Mahesh|35000|    0|    0|
|   EMP00008|       George|    0|60000|    0|
|   EMP00005|         Raju|    0|    0|20000|
|   EMP00009|       Tushar|20000|    0|    0|
|   EMP00001|       Vishal|    0|    0|50000|
|   EMP00002|          Sam|20000|    0|    0|
|   EMP00003|         Ravi|    0|    0|50000|
+-----------+-------------+-----+-----+-----+
```

Question 4.

How to deal with nulls in Apache Spark?

Ans: -

1. In Spark, if there is datatype mismatch then due to default mode permissive. it will display as NULL.

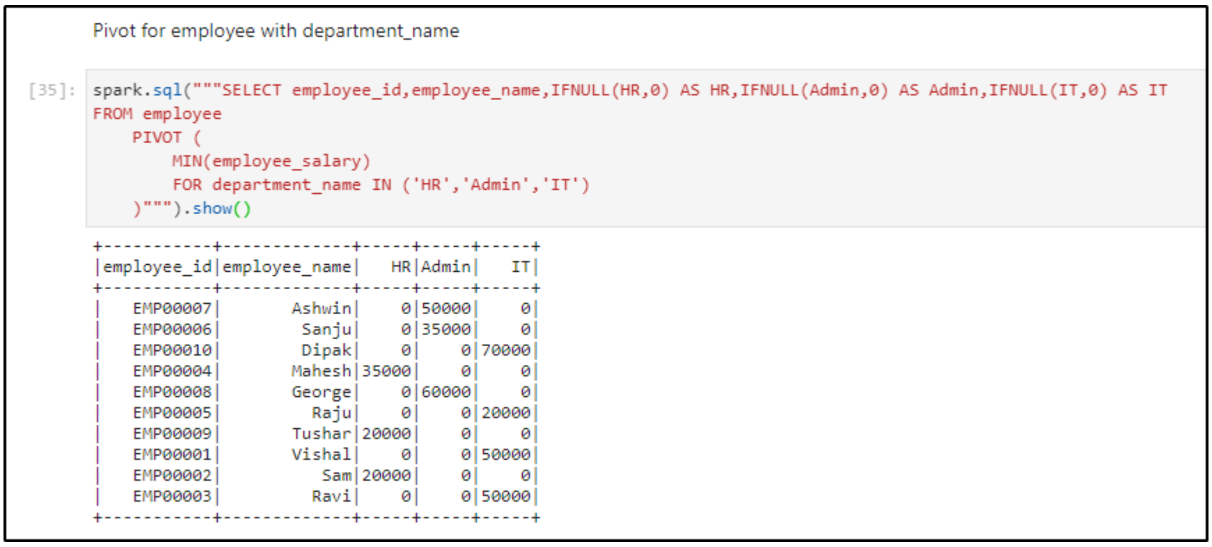


```
[4]: df_orders_03.show()
```

```
+--------+----------+-------+---------------+
|order_id|order_date|cust_id|   order_status|
+--------+----------+-------+---------------+
|       1|2013-07-25|  11599|         CLOSED|
|       2|2013-07-25|    256|PENDING_PAYMENT|
|       3|2013-07-25|  12111|       COMPLETE|
|       4|2013-07-25|   8827|         CLOSED|
|       5|2013-07-25|  11318|       COMPLETE|
|       6|2013-07-25|   7130|       COMPLETE|
|       7|2013-07-25|   null|       COMPLETE|
|       8|2013-07-25|   2911|     PROCESSING|
|       9|2013-07-25|   null|PENDING_PAYMENT|
|      10|2013-07-25|   5648|PENDING_PAYMENT|
+--------+----------+-------+---------------+
```

We can change this behaviour by changing the mode to failfast or dropmalformed.

2. While defining schema we can set the column as not allow NULL then if we try to insert NULL in that column it will fail.
3. In User define function if we encounter NULL and due to that if code will fail then we can filter the NULL.
4. We can replace NULL values with fill() function in dataframe.
5. For integer column we can replace it with 0



Pivot for employee with department_name

```
[35]: spark.sql("""SELECT employee_id,employee_name,IFNULL(HR,0) AS HR,IFNULL(Admin,0) AS Admin,IFNULL(IT,0) AS IT
      FROM employee
          PIVOT (
              MIN(employee_salary)
              FOR department_name IN ('HR','Admin','IT')
          )""").show()
```

```
+-----------+-------------+-----+-----+-----+
|employee_id|employee_name|   HR|Admin|   IT|
+-----------+-------------+-----+-----+-----+
|   EMP00007|       Ashwin|    0|50000|    0|
|   EMP00006|        Sanju|    0|35000|    0|
|   EMP00010|        Dipak|    0|    0|70000|
|   EMP00004|       Mahesh|35000|    0|    0|
|   EMP00008|       George|    0|60000|    0|
|   EMP00005|         Raju|    0|    0|20000|
|   EMP00009|       Tushar|20000|    0|    0|
|   EMP00001|       Vishal|    0|    0|50000|
|   EMP00002|          Sam|20000|    0|    0|
|   EMP00003|         Ravi|    0|    0|50000|
+-----------+-------------+-----+-----+-----+
```