



Azure Roadmap Elevating Your Cloud Skills

Project 1: Data Processing and Transformation in Hive using Azure VM

Christian Asiema

Disclaimer

The following documents are a combination of information and sources from online and project workflows that I have currently done. They are not meant for commercial or teaching purposes as the information has not been well cited and referenced. It's just a documented workflow that's for personal future references and sprints if similar workflows are to be carried out. How these documents work is when I am going through a project, I'd link it to the current theory and try to incorporate it into the document.

Table of Contents

Table of Contents.....	3
1 Key Criteria for Setting Up a Cloud Project	32
1.1 Business & Project Goals.....	32
1.2 Architecture & Design.....	32
1.3 Data Considerations (for Data/AI Projects).....	33
1.4 Performance & Optimization.....	33
2 Cost Management	34
2.1 Standard Budget Alert (Daily Evaluation).....	34
2.1.1 Scope	35
2.1.2 Details of Budget	36
2.1.3 Budget Amount.....	36
2.1.4 Filters (Optional).....	37
2.1.5 Alerts (Next Step).....	37
2.2 Collaboration & Documentation.....	38
2.3 Future-proofing.....	38
3 Project Introduction.....	40
3.1 Use Cases for the Proposed Pipeline	40
3.2 Data Characteristics and Suitability	41
3.3 Use case in the Project	42

3.4	Datasets Used	42
4	Project Flow	43
4.1	Azure VM Setup.....	43
4.2	Disk Configuration	43
4.3	SSH Connection Using PuTTY.....	43
4.4	Dataset Upload.....	43
4.5	Hadoop Installation.....	43
4.6	Hive Installation and Setup	43
4.7	Creating Hive Tables	44
4.8	Querying the Data	44
4.9	Partitioning and Bucketing	44
4.10	Sampling and Views	44
4.11	Optimized Hive Queries	44
5	Prerequisites.....	45
5.1	Ways to interact with the Azure services:	45
5.2	Best Practices for Cloud Accounts:	45
6	Services	47
6.1	Azure VM:	47
6.1.1	Layman's Explanation.....	47
6.1.2	Advantages of Azure VM:	47

6.1.3	Alternatives:.....	48
7	Apache Hadoop:.....	49
7.1	Layman's Explanation	49
7.1.1	Analogy: A Giant Puzzle Factory	50
7.2	Advantages of Hadoop	51
7.3	Alternatives	51
8	Apache Hive.....	52
8.1	Simple Explanation of Apache Hive:	52
8.1.1	Visual Analogy:	52
8.2	Advantages of Hive.....	53
9	PuTTY	55
9.1	Key Fields in PuTTY Configuration.....	58
9.1.1	Host Name (or IP address).....	58
9.1.2	Port	58
9.1.3	Connection type	58
9.1.4	Saved Sessions	59
9.1.5	Close Window on Exit	59
9.2	Next Steps after Configuration	59
10	Creating Azure Resource Group.....	60
10.1	Key Tags for Resource Groups (and Why They Matter)	62

10.1.1	Cost Management & Billing	62
10.2	Ownership & Accountability.....	62
10.3	Environment & Lifecycle	63
10.4	Security & Compliance	63
10.5	Operational Management	63
10.6	Summary of Must-Have Tags.....	65
11	Creating Azure VM	66
11.1	Architecture.....	66
11.2	Key Components in the Diagram	67
11.2.1	Hive Client.....	67
11.2.2	Hive Server 2.....	67
11.2.3	Beeline	67
11.2.4	Driver	67
11.2.5	Metastore	67
11.2.6	Execution Engine (MapReduce / YARN)	67
11.2.7	HDFS (Hadoop Distributed File System).....	68
11.3	Process Flow (Query Lifecycle).....	68
11.3.1	Layman's term	68
12	Create Virtual Machine	71
12.1	Azure VM Connectivity Setup	79

12.2	Disk Mounting.....	83
12.2.1	Command 1 : lsblk	85
12.2.2	Command 2: <i>sudo parted /dev/sdc</i>.....	88
12.2.3	Command 3: <i>sudo mkfs.xfs /dev/sdc1 –f</i>.....	90
12.2.4	Command 4: <i>sudo partprobe /dev/sdc1</i>.....	92
12.2.5	Command 5: <i>sudo mkdir /disk</i>	95
12.2.6	Command 6: <i>sudo mount /dev/sdc /disk</i>.....	96
12.2.7	Command 7: <i>sudo blkid</i>	98
12.2.8	Command 8: <i>sudo nano /etc/fstab.</i>.....	100
12.2.9	Command 9: <i>sudo -I chmod 777 /disk</i> Exit.....	102
13	Hadoop & Hive Installation Context	105
13.1	Step 1: Update & Install Java (Required for Hadoop + Hive).....	107
13.1.1	<i>sudo apt install openjdk-8-jdk -y</i>	107
13.1.2	<i>java -version; javac -version</i>	107
13.2	How It Fits into the Bigger Workflow.....	108
14	Step 2: Install OpenSSH	109
14.1	<i>openssh-server</i>	109
14.2	<i>openssh-client.....</i>	109
14.3	<i>-y</i>	109
14.4	Where This Fits in the Workflow.....	110

15	Step 3: Generate SSH Key & Authorize It	111
15.1	Generate a key pair	111
15.2	Add the public key to authorized keys	111
15.3	Lock down permissions	112
15.4	Where We Are in Workflow.....	113
16	Step 4: Test SSH Connection	113
16.1	Why This Is Important	114
16.2	Where We Are in Workflow	115
17	Step 5: Gain Root Access	116
17.1	Why Hadoop Setup Needs This	116
17.2	Analogy	117
17.3	Where We Are in Workflow	117
18	Step 6: Allow Permissions to /disk	118
18.1	Why Hadoop Needs This	118
18.2	Analogy	119
18.3	Where We Are in Workflow	119
19	Step 7: Create Hadoop Installation Directory.....	120
19.1	cd /disk	120
19.2	mkdir hadoop	120
19.3	ls	120

19.4	Where We Are in Workflow	121
20	Step 8: Download Hadoop Package	122
20.1	What This Does (Layman's Terms)	122
21	Step 9: Extract Hadoop	124
21.1	cd /disk/hadoop.....	124
21.2	tar xzf hadoop-3.4.1.tar.gz	124
21.3	Why This Step Matters	125
21.4	Analogy.....	125
22	Step 10: Add Hadoop + Hive Environment Variables	126
22.1	Save + Exit Nano	127
22.2	Apply Changes Immediately.....	127
22.3	Test Hadoop	127
22.4	Why This Step Matters	127
23	Step 11: Validate Java Setup.....	128
23.1	which javac.....	128
23.2	readlink -f /usr/bin/javac.....	128
23.3	Why Hadoop Needs This.....	128
23.4	Where We Are in Workflow.....	129
24	Step 12: Set JAVA_HOME in Hadoop Env	130
24.1	Why This Step Matters	130

24.2	Double Check.....	131
24.3	Fix it fast.....	132
	Install a full JDK.....	132
25	Step 13: Configure core-site.xml.....	134
25.1	What These Settings Mean.....	135
25.1.1	hadoop.tmp.dir	135
25.1.2	fs.defaultFS	135
25.2	Why This Step Matters	135
25.3	Next Step.....	136
26	Step 14: Configure hdfs-site.xml.....	137
26.1	What These Do.....	138
	dfs.name.dir	138
	dfs.data.dir	138
	dfs.replication	138
26.2	Verify Directories Exist	138
27	Step 15: Configure mapred-site.xml.....	139
	Edit the file	139
	Paste this configuration	139
27.1	What This Does	140
27.1.1	mapreduce.framework.name = yarn.....	140

27.2	Next Step.....	140
28	Step 16: Configure YARN.....	141
29	Step 17: Format the Hadoop Filesystem	144
29.1	Command:	144
29.2	Explanation:	144
29.2.1	Layman's Explanation:	144
29.3	What the Log Shows	145
29.3.1	Layman's Explanation	146
30	Step 18: Start Hadoop Services	147
30.1	Explanation:	147
30.2	Layman's Explanation:	148
31	Step 19: Verify Hadoop Services	148
31.1	Explanation:	149
31.1.1	Layman's Explanation:	149
31.2	Expected Output (example):	150
32	Step 1: Navigate to the Home Directory	152
32.1	Layman's Explanation:	152
33	Step 2: Move to the Attached Disk.....	153
33.1	Explanation:	153
33.1.1	Layman's Explanation:	153

34	Step 3: Create Hive Directory.....	154
34.1	Explanation:	154
34.2	Layman's Explanation:	154
35	Step 4: Move Into the Hive Folder.....	155
35.1	Explanation:	155
35.1.1	Layman's Explanation:	155
36	Step 5: Move Back to Parent Directory.....	156
36.1	Explanation:	156
36.2	Layman's Explanation:	156
36.3	Error Occured	157
36.4	Finding the Correct Hive Download URL	157
37	Correct Download Command (Step 6).....	157
37.1	Breakdown:	158
37.2	Why the Previous URL Failed.....	158
38	Step 6: Go Back Into the Hive Directory	159
38.1	What it does:	159
38.1.1	Layman's Explanation	159
39	Step 7: Extract the Hive Package.....	160
39.1	Layman's Explanation	161
40	Step 8: Move to Home Directory.....	162

40.1	What it does:	162
40.2	Layman's Explanation	162
41	Step 9: Hive	163
41.1	What you added:.....	164
41.2	Explanation in Layman's Terms:	164
41.3	What it does:	165
41.3.1	Layman's Terms:.....	165
42	Step 11: Configuring Hive to know where Hadoop is installed.	166
42.1	What to do inside:	166
42.2	What it does:	166
42.2.1	Layman's Terms:.....	167
43	Step 13: Fixing Hive permissions in HDFS.	168
	What it does:	168
43.1.1	Layman's Terms:.....	168
44	Step 14: Creating the Hive warehouse directory in HDFS.	169
44.1	What it does:	169
44.1.1	Layman's Terms:.....	170
45	Step 15: Setting Permissions for the Hive Warehouse Directory.	170
45.1	What it does:	170
45.1.1	Layman's Terms:.....	171

46	Step 16: Hive Configuration Directory.	171
46.1	What it does:	171
46.1.1	Layman's Terms:	172
47	Step 17: Create the Hive Config File.	173
47.1	What it does:	173
47.1.1	Layman's Terms:	173
48	STEP 18. sudo nano hive-site.xml	174
a)	Add <property> entries (like temp dirs) into the file:	174
b)	Look for the transactional tables section	175
48.1.1	Layman's Explanation:	176
49	STEP 19 initializing Hive's metastore	178
49.1	What it does	178
49.2	Why it matters	178
	Layman's terms:	179
50	STEP 20: Launching Hive	180
50.1	Launches the Hive CLI (Command Line Interface)	180
50.2	Logs and Warnings	180
50.2.1	Deprecation Warning:	181
50.3	The Prompt Appears	182
50.4	Layman's Explanation	182

51	STEP 21: Test Hive	184
51.1	Layman's Explanation.....	187
51.2	Key Components:.....	189
51.2.1	Beeline (CLI Client).....	189
51.2.2	HiveServer2	189
51.2.3	Driver	189
51.2.4	Metastore	189
51.2.5	Execution Layer (YARN + MapReduce/Tez)	189
51.2.6	HDFS.....	190
51.3	Beeline Setup in Simple Layman's Terms	190
51.4	Why Use Beeline + HiveServer2?	191
51.5	What to Set Up Next?.....	191
52	Organizing Data in Local Directories (on Azure VM)	192
52.1	Using SCP to Transfer Files to the VM.....	192
52.2	Unzipping and Decompressing Files	192
52.3	Layman's Analogy.....	192
53	STEP 1: Beeline setup	194
Meaning of These Properties:	195	
53.1	Layman's Explanation	196
53.2	Note on Security	197
54	STEP 2: Cleanly restarting your Hadoop environment	198

54.1	Why Stop First?	198
54.1.1	Layman's Analogy	199
54.2	What To Do Next (Restart Sequence)	199
54.3	Why run them after stopping?	201
54.4	Layman's Analogy	201
54.5	Next Step After Hadoop Restart	202
55	HiveServer2 startup step	203
55.1	hiveserver2	203
55.2	Why It's Necessary	203
55.2.1	Layman's Analogy	204
55.3	Next Step: Connect via Beeline	205
55.4	What the Output Means.....	206
55.5	Next Step: Connect via Beeline	206
55.6	Layman's Explanation	206
56	Beeline connection step.....	207
56.1	What Happens Here:	207
56.2	Layman's Analogy	208
56.3	To fix that.....	208
56.4	Commands Breakdown	209
57	Create tables in Hive.....	211

58	Step 1: Create Directories on Azure VM	212
58.1	cd /disk	212
58.2	cd data	212
58.3	What it does:	214
58.3.1	Layman's Explanation	214
59	Step 2: Copy Local Files to Azure VM Using SCP	215
59.1	Layman's Explanation:	215
60	Step 3: Unzip and Decompress Files on Azure VM	216
60.1	What It Does:	217
60.1.1	Layman's Explanation:	218
61	Step 4: Copy .bz2 Files (1987–2004) to csv Folder for Decompression	218
62	Step 5: Decompress All .bz2 Files into .csv Format	220
63	STEP 6: Move CSV Files (1987–2008) to HDFS as Asiema	222
1:	Create HDFS target directory	222
2:	Set write permissions on the directory	222
3:	Move all CSV files from 1987 to 2008 to HDFS	222
63.1.1	Breakdown	223
63.2	Airports Data	223
63.3	Carriers Data	224
63.4	Planes Data	224

63.5 Summary	225
64 Step 5: Create Hive Tables	226
64.1 What it does:	226
64.2 What it does:	227
64.3 Layman's Analogy:	227
65 Step 6: Create Hive External Tables.....	228
65.1 Create the flights table:	228
65.1.1 What it does:.....	229
65.1.2 Important Notes:.....	229
Layman's Analogy:.....	229
65.2 Create Hive External Table for Airports.....	230
65.2.1 What it does:	232
Layman's Analogy:.....	232
65.3 Step X: Create Hive External Table for Carriers	234
65.3.1 What it does:.....	234
Layman's Analogy:.....	235
65.4 Create Hive External Table for Planes	236
65.4.1 What it does:.....	237
Layman's Analogy.....	238
65.5 Step 7: Query Hive Tables to Validate Data.....	239

65.5.1	What it does:	239
Layman's Analogy:.....		239
65.6	Errors Returned.....	240
65.6.1	The Problem	240
65.6.2	How to Fix It	241
Layman's Explanation:.....		243
65.6.3	What They Do.....	244
65.6.4	Why This Is Important	245
65.6.5	Layman's Analogy	245
65.6.6	Queries You Ran	246
65.6.7	What They Do.....	246
65.6.8	Why This Matters.....	246
65.6.9	Layman's Analogy	247
66	Technical Explanation: Partitioning in Hive	248
66.1	Benefits	248
66.2	Commands to Enable Dynamic Partitioning	248
66.2.1	What They Do.....	248
66.2.2	Layman's Analogy	249
66.3	Command to Create External Partitioned Table	250
66.3.1	What It Does	251

66.3.2 Validation Command	252
66.3.3 Layman's Analogy	252
66.4 Insert Data Dynamically Into Partitioned Table	254
66.4.1 What It Does	255
66.4.2 Benefits	255
66.4.3 Layman's Analogy	256
66.5 Query Hive Tables to Validate Data.....	256
66.5.1 Validation Queries.....	256
66.5.2 What They Do.....	256
66.5.3 Why This Is Important	258
66.5.4 Layman's Analogy	258
66.6 Row counts.....	259
66.6.1 Queries	259
66.6.2 What They Do.....	259
66.6.3 Why This Is Important	260
66.6.4 Layman's Analogy	260
67 Partitioning	261
1. Benefits of Partitioning.....	261
67.1.1 Example (Flights Dataset)	261
67.1.2 Layman's Analogy	262

67.2	Enable Dynamic Partitioning	262
67.2.1	Commands.....	262
67.2.2	What They Do.....	262
67.2.3	Example in Your Case	263
67.2.4	Why It's Useful	264
67.3	Create an External Partitioned Table.....	264
67.3.1	Command.....	264
67.3.2	What It Does	266
67.3.3	How to Check	267
67.4	Insert Data Dynamically Into Partitioned Table	269
67.4.1	Why Dynamic Partitioning Matters	271
67.4.2	Layman's Analogy	272
67.5	View Table Details	272
67.5.1	What It Does	272
67.5.2	Why It's Important.....	273
67.5.3	Layman's Analogy	273
67.6	Upload New data to HDFS	273
67.6.1	Why It's Important	274
67.6.2	Alternative.....	274
67.6.3	Next Steps	275

Layman's Analogy:.....	275
67.7 Register new partition in Hive	275
67.7.1 What It Does	276
67.7.2 Why It's Needed	276
67.7.3 Layman's Analogy.....	277
67.7.4 Pro Tip: Add All Partitions Automatically	277
67.8 Load Data from the Main Table into the 2005 Partition.....	278
67.8.1 Command.....	278
67.8.2 What It Does	279
67.8.3 Why This Helps	280
67.8.4 Layman's Analogy	280
67.9 Compare Query on Non-partitioned vs Partitioned Table.....	281
67.9.1 The Queries	281
67.9.2 Why the Partitioned Query Wins	282
67.9.3 Layman's Analogy.....	282
67.9.4 Summary	282
68 Clustering/ Bucketing	283
Definition:	283
68.1 Differences from Partitioning:	283
68.2 How it Works:	283

68.3	Performance Benefits:.....	283
68.4	Use Cases:	284
68.5	Layman's Explanation	284
68.6	Why it's better than partitioning:	285
68.7	Execution	286
68.8	Layman's Explanation	287
68.9	Insert Data into the Bucketed Table	289
68.9.1	Technical Explanation (Expanded)	289
68.10	Breakdown:.....	291
68.11	Layman's Explanation	292
68.11.1	Layman's Explanation	295
69	Sampling.....	297
69.1	What is Sampling?.....	297
69.2	When to use Sampling:.....	297
69.3	Types of Sampling in Hive:	297
69.4	Layman's Explanation	299
69.5	Types of Sampling in Plain Terms:.....	299
70	Simple Limit-based Sampling.....	301
70.1	Query:	301
70.1.1	Why it's important:	301

70.1.2	Layman's Explanation	301
70.2	Random Sampling using RAND()	303
70.2.1	What it does:.....	303
70.2.2	Why it's important:	303
70.2.3	Layman's Explanation	303
70.3	Sampling by Number of Rows (TABLESAMPLE)	305
70.3.1	What it does:.....	305
70.3.2	Key Differences from LIMIT:.....	305
70.3.3	Why use it:	305
70.3.4	Layman's Explanation	305
70.4	Bucket Sampling from a Bucketed Table	307
70.4.1	What it does:.....	307
70.4.2	Why it's important:	307
70.4.3	Layman's Explanation	308
70.5	Sampling with Fixed Data Volume	309
70.5.1	What it does:.....	309
70.5.2	Why it's useful:	309
70.5.3	Layman's Explanation	310
70.6	Percentage Sampling	311
70.6.1	What it does:.....	311

70.6.2	Why it's useful:	311
70.6.3	Layman's Explanation	312
71	Joins	313
1.	What are Joins in Hive?	313
71.1	Why are joins expensive in Hive?	313
71.2	Optimized Joins in Hive:.....	313
71.3	Key benefit of optimized joins:.....	314
71.4	Layman's Explanation	314
72	INNER JOIN	315
72.1	What it does:	316
72.2	Internally in Hive:	316
72.3	Key Point:	317
72.4	Layman's Explanation	317
73	RIGHT OUTER JOIN	319
1.	Query:	319
73.1	What it does:	319
73.2	Internally in Hive:	319
73.3	Layman's Explanation	320
74	Auto-Configured Map Join	321
74.1	Query:	321

74.2	What it does:	321
74.3	Internally in Hive:	321
74.4	When to use:.....	322
74.5	Layman's Explanation	322
75	Bucket Map Join	323
75.1	Settings:.....	323
75.2	What is a Bucket Map Join?.....	323
75.2.1	Conditions:.....	323
75.3	Internally in Hive:	324
75.4	Summary:.....	324
75.5	Layman's Explanation	324
76	Complex Multi-Table JOIN (LEFT JOIN Example).....	325
76.1	Query:	325
76.2	What it does:	326
76.3	Internally in Hive:	326
76.4	Layman's Explanation	327
77	VIEWS.....	329
77.1	What is a View in Hive?	329
77.2	Key Characteristics:	329
77.3	Layman's Explanation.....	330

78	CREATE VIEW flightsview AS SELECT...	331
78.1.1	Layman's Explanation.....	332
78.1.2	In short:	333
79	SELECT * FROM flightsview LIMIT 10	334
1.	What it does:.....	334
79.1	Purpose:.....	334
79.2	Layman's Explanation.....	335
79.3	In short:	335
80	SET hive.cli.print.header=true	336
1.	What it does:.....	336
80.1	Example:	336
80.2	Layman's Explanation.....	336
80.3	In short:	337
81	SELECT * FROM flightsview WHERE year=2008 AND month=1 LIMIT 10	338
1.	What it does:.....	338
81.1	Purpose:.....	338
81.2	Layman's Explanation.....	339
81.3	In short:	339
82	INSERT OVERWRITE DIRECTORY	340
1.	What it does:.....	340

2.	Layman's Explanation	341
82.1	In short:	341
83	cd /disk/hive/apache-hive-3.1.2-bin/bin.....	342
1.	What it does:	342
83.1	Layman's Explanation	343
83.2	In short:	344
84	ALTER VIEW flightsview RENAME TO flights_view	344
1.	What it does:	344
84.1	Layman's Explanation	344
84.2	In short:	345
85	DROP VIEW flights_view	346
1.	What it does:	346
85.1	Layman's Explanation	346
85.2	In short:	347
86	OPTIMIZATION	348
1.	Layman's Explanation	349
86.1	In short:	349
87	COMPRESSION	350
1.	Enable Compression for Output	350
87.1	Set Compression Codec	350

87.2	Enable Compression for Intermediate Data	350
87.3	Hive Execution Engine	350
87.4	Check Current Engine	351
87.5	Set Execution Engine	351
87.5.1	Layman's Explanation.....	351
87.5.2	In short:	352
88	EXPLAIN.....	353
1.	What is EXPLAIN?.....	353
88.1	Key Use Cases:.....	353
88.1.1	Example 1: Basic Usage	353
88.1.2	Example 2: Partition Comparison.....	354
88.2	Layman's Explanation	354
88.3	In short:	355
89	Extended & Authorization Plan.....	356
90	Data Processing and Transformation in Hive Using Azure VM	358
91	Summary	365
92	Troubleshooting	367
92.1	Steps to Downgrade Hive.....	367
92.1.1	Remove/rename your current Hive	367
92.1.2	Download Hive 3.1.2	367

92.1.3	Extract Hive 3.1.2	367
92.1.4	Update your environment variables	367
92.1.5	Link Hive to Hadoop.....	368
92.1.6	Verify	369

PART 1: PROJECT SETUP

1 Key Criteria for Setting Up a Cloud Project

1.1 Business & Project Goals

- **Problem definition:** What are you solving? (e.g., data pipeline, fraud detection, migration, cost optimization).
 - **Success metrics:** How will you measure success? (e.g., SLA uptime %, query speed, reduced storage cost, improved accuracy).
 - **Stakeholder alignment:** Who are the end-users? Who approves design decisions?
-

1.2 Architecture & Design

- **Cloud model:** Public, private, hybrid, or multi-cloud?
 - **Resource planning:** VM size, storage type (Blob vs Disk vs DB), network topology.
 - **Scalability:** Can the system handle 10x traffic or data growth?
 - **Security by design:** RBAC, identity management (Azure Entra ID), encryption at rest/in transit.
 - **High availability & disaster recovery:** Load balancers, failover zones, backup strategy.
-

1.3 Data Considerations (for Data/AI Projects)

- **Data sources:** What are you ingesting (databases, IoT streams, files)?
 - **ETL/ELT pipeline:** Batch vs streaming, tools (Azure Data Factory, Databricks, Synapse).
 - **Data quality:** Validation, cleansing, schema enforcement.
 - **Governance:** Compliance (GDPR, POPIA, HIPAA), lineage, cataloging (Purview).
 - **Storage design:** Lake vs warehouse, partitioning strategy, hot/cold tiers.
-

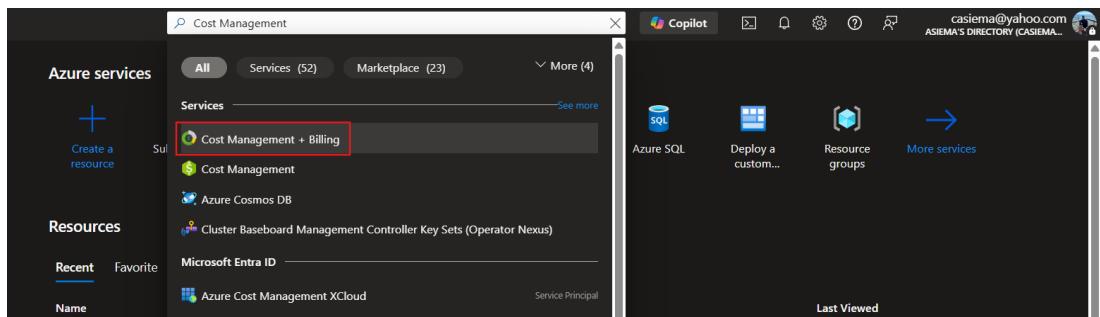
1.4 Performance & Optimization

- **Compute efficiency:** Rightsizing (don't overspend on VMs or cores).
 - **Caching & indexing:** Improve query performance.
 - **Automation:** Infrastructure as Code (Terraform, Bicep, ARM).
 - **Monitoring:** Metrics, logs, alerts (Azure Monitor, Application Insights).
-

2 Cost Management

2.1 Standard Budget Alert (Daily Evaluation)

1. Go to **Azure Portal**



2. In the Search bar, type **Cost Management + Billing → Budgets.**

A screenshot of the "Christian Asiema | Budgets" page in the Azure portal. The left sidebar shows navigation options: Overview, Access control (IAM), Billing scopes, Diagnose and solve problems, Cost management (with Cost analysis, Cost alerts, and Budgets selected and highlighted with a red box), Advisor recommendations, and Billing. The main content area displays a message: "Scope : Christian Asiema" and "Name ↑ Scope ↑ Reset period". Below this, it says "You do not have any budgets." There is also a "Search by name" input field and a "Feedback" button.

2.1.1 Scope

The screenshot shows the 'Create budget' page. At the top, there's a breadcrumb navigation: Home > Christian Asiema | Budgets >. Below it, the title 'Create budget' is displayed with a '...' button. A 'Budget' link is also present. Two buttons are at the top: '1 Create a budget' (underlined) and '2 Set alerts'. A descriptive text follows: 'Create a budget and set alerts to help you monitor your costs.' The 'Budget scoping' section includes a 'Scope' dropdown set to 'Christian Asiema' and a 'Filters' section with a 'Add filter' button. The 'Budget Details' section allows setting a name ('Enter a unique name'), reset period ('Monthly'), creation date ('2025 September 1'), and expiration date ('2027 August 21').

- This is the level at which the budget applies.
- In your screenshot, the scope is your **subscription (Christian Asiema)**.
- You could also set budgets at **management group** or **resource group** level for more granular control.

👉 For cost alerts every 24 hours, subscription-level is ideal.

2.1.2 Details of Budget

This defines the **lifecycle of your budget**.

- **Name:** A unique identifier (e.g., Monthly-Cost-Budget, DataPipeline-Budget).
- **Reset Period:**
 - **Monthly** → Resets every month. Best for ongoing cost control.
 - **Quarterly/Annually** → Good for projects or annual budgets.
- **Creation Date:** The start date for the budget (e.g., Sept 1, 2025).
- **Expiration Date:** When the budget stops being enforced (e.g., Aug 31, 2027).
- **Amount:** Your cost threshold in **USD** (or your billing currency). Example: ZAR 10,000 → If spend goes above, alerts can trigger.

👉 **Tip:** Always set an amount a little below your hard spending cap so you get notified early.

2.1.3 Budget Amount

- This is the maximum cost you want Azure to monitor.
 - Example: Enter 200 if you want to be notified when your subscription costs approach ZAR 20,000/month.
 - Alerts can be set at different **percentages of this budget** (e.g., 50%, 75%, 100%).
-

2.1.4 Filters (Optional)

- Let's you restrict the budget to certain **resource groups, services, or locations**.
 - Example: Only track **Virtual Machines in East US** or **Storage Accounts in South Africa North**.
 - If you leave blank → budget applies to the entire subscription.
-

2.1.5 Alerts (Next Step)

Once you move to **Step 2 (Set Alerts)**:

- You can configure thresholds (e.g., 50%, 80%, 100%).
- Assign **Action Groups** (email, SMS, Teams, webhook, automation).

⚠️ Important: **Azure Budgets by default check once daily.** To get alerts every **6 hours**, you'll need to add an **Azure Monitor alert rule** on top of this budget that evaluates costs every 6 hours.

✓ So, think of this screen as **defining the “budget container”** (scope + dates + amount).

The **alerts step** is where you decide who gets notified and at what thresholds.

And if you want **6-hour checks**, you extend it with Azure Monitor.

- **Budgeting:** Tagging (Department, Project, CostCenter).
- **Forecasting:** Use cost calculators before deployment.
- **Optimization:** Reserved instances, spot VMs, auto-scaling.

- **Review cadence:** Weekly/monthly cost governance meetings.
-

2.2 Collaboration & Documentation

- **Version control:** GitHub/GitLab/Azure DevOps.
 - **CI/CD pipeline:** Automate deployment (GitHub Actions, Azure DevOps Pipelines).
 - **Runbooks/manuals:** Step-by-step documentation for replication and troubleshooting.
 - **Knowledge sharing:** Wiki, PDF manual, README.
-

2.3 Future-proofing

- **Extensibility:** Can you plug in new data sources or services easily?
 - **Portability:** Vendor lock-in awareness (e.g., making your architecture Azure-centric vs. hybrid-ready).
 - **Upgrade path:** Regularly review SDKs, API versions, VM images.
 - **Skill alignment:** Team's ability to maintain/operate the solution.
-

PART 2: PROJECT SCOPE

3 Project Introduction

The main aim of the project is to demonstrate how to handle and process large volumes of structured airline data using Hive on an Azure Virtual Machine (VM). The focus is on building a scalable and efficient pipeline that can manage high-volume datasets by leveraging distributed computing frameworks in the Hadoop ecosystem.

This project will simulate real world big data scenarios by working with a comprehensive airline dataset and processing high flight records to extract meaningful insights using Hive. The project will highlight how Hive's components such as HiveQL, tables, partitions and optimization strategies can be used to analyze and query vast amounts of airline-related information efficiently.

3.1 Use Cases for the Proposed Pipeline

- **Flight Performance Monitoring:** Track flight schedules, delays, and on-time performance using flight data collected across the years.
- **Maintenance and Fleet Analysis:** Analyze aircraft-related data to observe trends based on tail number, engine type, and manufacturer to ensure timely maintenance and optimization.
- **Carrier and Route Optimization:** Evaluate the performance of different carriers and routes to suggest optimizations in scheduling, resource allocation, and demand-based routing.
- **Airport and Location Insights:** Use airport metadata to understand traffic flow across regions, helping authorities and airlines plan infrastructure and logistics.

The project addresses the challenges posed by high-velocity, high-volume airline data.

The goal is to create a unified and query-efficient data model where structured data from multiple sources can be joined, filtered, and analyzed using Hive.

This project will use the following datasets:

- **airlines.csv** - Contains fields like IATA_code, airport_name, city, state, and country
 - **carrier.csv** - Contains code and description of each airline
 - **plane-data.csv** - Contains tail_number, type, manufacturer, model, engine_type
 - **Flights data (yearly)** - Includes fields such as flight_num, departure, arrival, origin, destination, and distance
-

3.2 Data Characteristics and Suitability

The dataset consists of structured and semi-structured data in CSV format, suitable for Hive ingestion and analysis. It reflects time-series data generated by various sources such as airlines, airports, and air traffic systems. The data volume is ideal for demonstrating batch processing capabilities and query optimization techniques in Hive.

By executing this on an Azure VM, we simulate a platform, showing how cloud-based infrastructure and open-source tools like Hive can effectively process and extract actionable insights from airline industry data.

3.3 Use case in the Project

This project will encompass the use case of large-scale airline data and architect a batch-processing pipeline hosted on an **Azure Virtual Machine (VM)**. The VM will be configured with **Apache Hive** and other ecosystem tools to process, query, and analyze massive airline datasets in a scalable and efficient manner.

Processing large volumes of aviation-related data demands a robust data architecture capable of handling complex joins, filters, and aggregations. Hive, built on top of Hadoop, provides a SQL-like interface to handle this structured data in a fault-tolerant and distributed way.

3.4 Datasets Used

This project uses various CSV files representing different dimensions of airline data, including:

- **airlines.csv** - IATA_code, airport_name, city, state, country
- **carrier.csv** - code, description
- **plane-data.csv** - tail_number, type, manufacturer, model, engine_type
- **flights data (yearly)** - flight_num, departure, arrival, origin, destination, distance

These CSV files simulate real world aviation databases, covering aspects like airports, aircraft specs, flight history, and airline carriers. Each dataset will be ingested and processed through Hive tables both managed and external.

4 Project Flow

4.1 Azure VM Setup

The project starts by launching an Azure Virtual Machine that will host all big data processing components. System updates, Java installation, and environment configuration are completed at this stage.

4.2 Disk Configuration

An additional disk is attached and mounted to the VM to handle the large datasets. This ensures sufficient storage and smooth data processing operations.

4.3 SSH Connection Using PuTTY

Access to the Azure VM is established using the PuTTY application or any SSH client. This allows secure terminal-based control for all further set-up and execution.

4.4 Dataset Upload

Airlines datasets including airlines.csv, carrier.csv, plane-data.csv, and yearly flights.csv are uploaded. Files are transferred via SCP or fetched using wget and organized into folders.

4.5 Hadoop Installation

Hadoop is installed and configured with required paths and variables. This forms the core data storage and processing engine for Hive to work on top.

4.6 Hive Installation and Setup

Hive is installed after Hadoop, and the schema is initialized using Derby for metadata storage. The Hive shell is used to start writing and running queries.

4.7 Creating Hive Tables

External Hive tables are created based on each CSV file with appropriate schemas. Data is read directly from the uploaded files for immediate querying.

4.8 Querying the Data

Basic HiveQL queries are run to filter, group, and join the datasets. This helps in analyzing key metrics from the airline data.

4.9 Partitioning and Bucketing

Hive features like partitioning by year or destination and bucketing by manufacturer are applied. This improves performance by reducing scan times.

4.10 Sampling and Views

Sampling techniques are used to analyze subsets of large data quickly. Views are created to represent complex queries in a reusable format.

4.11 Optimized Hive Queries

Optimized HiveQL is executed on top of all datasets to simulate real-world big data processing. This helps us understand scalability and performance in cloud setups.

5 Prerequisites

Setup required:

- [Create an Azure Account](#)
- [Download the Code](#)
- [Download Data](#)
- [Download Installation & Execution](#)
- [Download PuTTY Application](#)

In this project, the pipeline is set up on cloud - Azure.

5.1 Ways to interact with the Azure services:

- **Console** - Using the Azure Web-based UI
 - **CLI** - Using the Azure CLI tool in Terminal/CMD
 - For this project, we will be using the UI Console of Azure to create a Data Pipeline.
-

5.2 Best Practices for Cloud Accounts:

- Protect the account by adding only authorized users/groups.
- Choose the region where services work best.
- [Create a Budget and set up an alert notification.](#)
- Keep track of unused ongoing services.

- Delete all the services after execution to cut down on charges.
- If you are using a free tier account, ensure all project services are available under the free tier; upgrade if needed. Delete unused services after execution to avoid charges.
- Follow a naming convention for resources for better understanding and name them according to the project aim.

6 Services

6.1 Azure VM:

Azure Virtual Machine (VM) is a scalable cloud-based computing service that allows users to deploy virtualized infrastructure. It provides flexibility to run a range of operating systems and applications just like a physical machine. Users can choose from a variety of VM sizes and specifications based on their workload needs. Azure VMs are accessible remotely and are ideal for large-scale data processing, development, and testing environments.

6.1.1 Layman's Explanation

An Azure Virtual Machine is like renting a computer that lives on the internet. You pick how powerful it is, turn it on when you need it, install and run software just like on a normal PC, and connect to it from anywhere. It's great for heavy data work, building and testing apps, or trying things out without buying your own hardware and you can shut it down when you're done to stop paying.

6.1.2 Advantages of Azure VM:

- Supports a wide range of operating systems, frameworks, and tools, making it easy to run anything from Linux-based big data platforms to Windows-based enterprise apps.
- Offers flexible scaling options to handle workloads of different sizes and adjust resources based on demand without service interruptions.
- Enables full customization of the environment, allowing users to set up disks, install specific tools, and configure network and security settings as needed.

- Guarantees high availability with regional redundancy and automatic backup features, reducing downtime and ensuring reliability during peak operations.

6.1.3 Alternatives:

- Amazon EC2
- Google Compute Engine
- IBM Cloud Virtual Servers DATA

7 Apache Hadoop:

Apache Hadoop is an open-source distributed data processing framework designed to store and analyze massive datasets across clusters of computers. It follows a master-slave architecture and uses HDFS for storage and MapReduce for processing. Hadoop is fault-tolerant, scalable, and capable of handling both structured and unstructured data. It is widely used in big data projects to enable parallel processing and storage.

7.1 Layman's Explanation

Apache Hadoop is like a giant teamwork system for computers. When you have huge amounts of data so big that one computer can't handle it you can use Hadoop to split that data into smaller chunks and send it to many computers working together. These computers store the data safely and process it at the same time, like a group of friends each solving a part of a big puzzle. This makes things faster, more reliable, and great for handling both neat, organized data (like spreadsheets) and messy data (like social media posts). It's a popular tool for big data projects.

7.1.1 Analogy: A Giant Puzzle Factory

Imagine you have a **huge, complex puzzle** too big for one person to solve. Now picture:

-  **Hadoop as the Puzzle Factory Manager**

It sees the big picture and organizes everything.

-  **Many Workers (Computers/Nodes)**

Each one gets a small section of the puzzle to work on. These are your “**nodes**” in Hadoop.

-  **Puzzle Boxes (HDFS – Hadoop Distributed File System)**

Each puzzle piece is stored in a special box (like HDFS), and copies of each box are made and sent to different workers just in case one worker drops theirs.

-  **Assembly Line (MapReduce)**

Each worker does their task, then passes their completed section to the next stage, which assembles everything into the final picture.

-  **Supervisor with Backup Plans (Fault-Tolerance)**

If one worker gets tired or disappears, someone else jumps in with a backup copy and keeps going without stopping the whole factory.

7.2 Advantages of Hadoop

- Supports large-scale data processing by breaking data into blocks and distributing them across multiple nodes for fast, parallel execution.
- Allows flexible storage by using HDFS, which stores data reliably even if hardware fails, making it suitable for massive unstructured datasets.
- Enables cost-effective data handling by using commodity hardware instead of expensive systems, reducing infrastructure costs significantly.
- Offers high scalability allowing organizations to add more nodes to the cluster without changing data formats or applications.

7.3 Alternatives

- Apache Spark
- Apache Flink
- Amazon EMR

8 Apache Hive

Apache Hive is a data warehouse infrastructure built on top of Hadoop that allows querying and managing large datasets using a SQL-like language called HiveQL. It translates these queries into MapReduce jobs for execution, making it easier for users familiar with SQL to work with big data. Hive is ideal for batch processing and analyzing structured data stored in HDFS.

8.1 Simple Explanation of Apache Hive:

Imagine you have a **huge Excel sheet** stored across many computers (thanks to Hadoop). Now, instead of writing complicated code to analyze this big sheet, Apache Hive lets you write **simple SQL-like queries** - just like you'd do in Excel or a database.

Hive acts like a translator: it takes your easy-to-write SQL-style commands and turns them into more complex instructions (called MapReduce) that the computers can understand and run in parallel. It's great for people who know basic SQL and want to work with big data without learning deep programming.

8.1.1 Visual Analogy:

Hive = Your Smart Assistant in a Giant Library

-  **Hadoop = Giant Library** holding endless shelves of books (data).
-  **You = SQL User**, asking questions in simple language.
-  **Hive = Smart Librarian Assistant** that understands your question and runs around the library (using MapReduce) to get the exact information you need.

-  **HDFS = Book Storage System** where all the information is stored neatly.

8.2 Advantages of Hive

- Supports SQL-like querying using HiveQL, which lowers the learning curve for analysts and developers coming from traditional RDBMS environments.
- Enables efficient batch processing of large datasets by translating queries into optimized MapReduce jobs executed across Hadoop clusters.
- Provides high compatibility with Hadoop and HDFS, ensuring seamless integration and allowing access to massive, distributed data.
- Offers schema flexibility and scalability supporting both static and dynamic partitioning for faster access and better performance.

Alternatives

- | | |
|-----------------|----------------------------|
| • Apache Impala | • Amazon Redshift Spectrum |
| • Presto | • Google BigQuery |

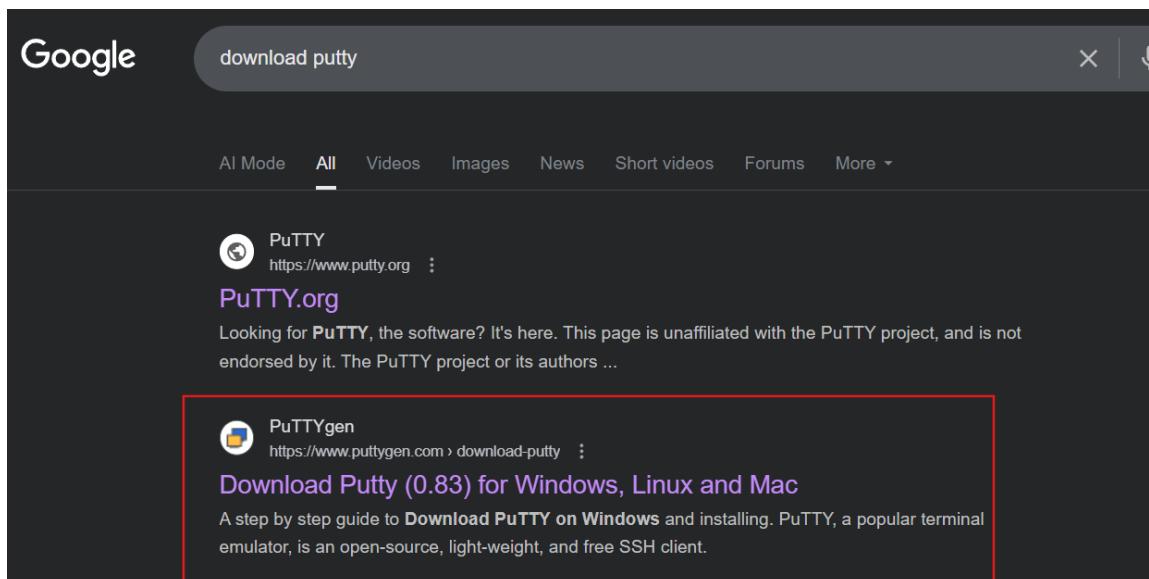
PART 3: CONFIGURE THE RESOURCES

9 PuTTY

In this project, we will install and set up Hive and Hadoop on an Azure VM to process large datasets using Hive queries. Before creating the Azure VM instance, it is essential to establish a secure connection between the Azure VM and the local machine. This ensures we can remotely configure the environment, install frameworks, and execute Hive queries directly from our system.

To achieve this, we use **PuTTY**, which is a widely used SSH client. PuTTY enables secure access to remote machines (like Azure VMs) using SSH protocol. It's lightweight, reliable, and easy to install, making it ideal for establishing terminal-based connections from a local Windows machine to Azure Linux-based VMs.

- a) For Windows users, this is the standard process to install and set up the PuTTY Application: Search for “**Download PuTTY**” on your browser and click on the first official link that appears.



- b) Click on the appropriate download link that takes you to the page containing PuTTY installers.

Download PuTTY for Windows , Linux and Mac

CONTENTS

- Download PuTTY for Windows
- Download PuTTY on Linux and Installation Guide
- Download PuTTY for Mac and Installation Guide

SHARE

[Twitter](#) [Facebook](#) [LinkedIn](#)

ENHANCED BY Google

Download PuTTY (0.83) for Windows

A step by step guide to Download PuTTY on Windows and installing.

PuTTY, a popular terminal emulator, is an open-source, light-weight, and free SSH client. It was developed by Simon Tatham in C language. Its primary function is to connect to a computer remotely while offering the facilities of transferring files, manipulating data, etc.

It offers support to a variety of network protocols like SSH, Telnet, Serial, SCP, SFTP, etc. PuTTY also comes with a command-line tool called "psftp" which can securely transfer files between computers over an SSH connection. It is compatible to use in most of the operating systems (for both 32 bit and 64 bit).

- c) On the download page, locate the MSI Windows installer and choose the correct version (preferably 64-bit) based on your system's configuration.

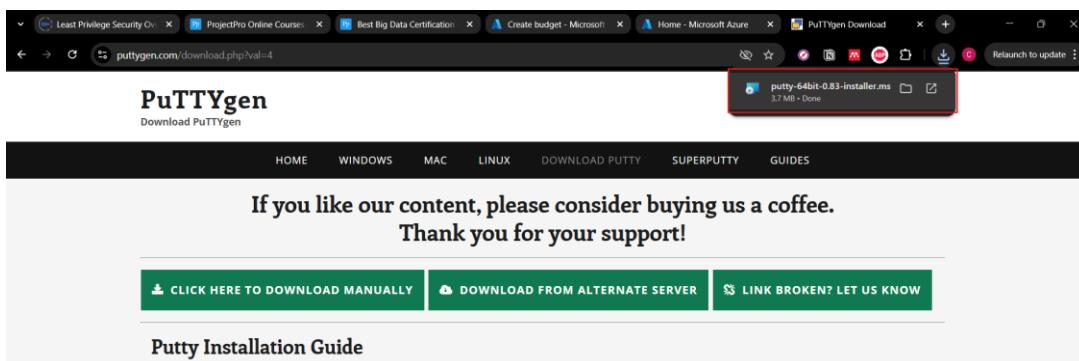
Download PuTTY for Windows Operating System (Released 27 May 2022)

Following are the steps required to download and install the latest version of PuTTY for Windows operating system:

Choose the appropriate windows installer version (32-bit or 64-bit) and click on it.

MSI ('Windows Installer for putty')			
32-bit:	putty-0.83-installer.msi	(via FTP)	(signature)
64-bit:	putty-64bit-0.83-installer.msi	(via FTP)	(signature)
64-bit x86:	putty-arm64-0.83-installer.msi	(via FTP)	(signature)
Unix source archive for putty			
.tar.gz:	putty-0.83.tar.gz	(via FTP)	(Signature)

- d) Download the .msi installer file to your local system.

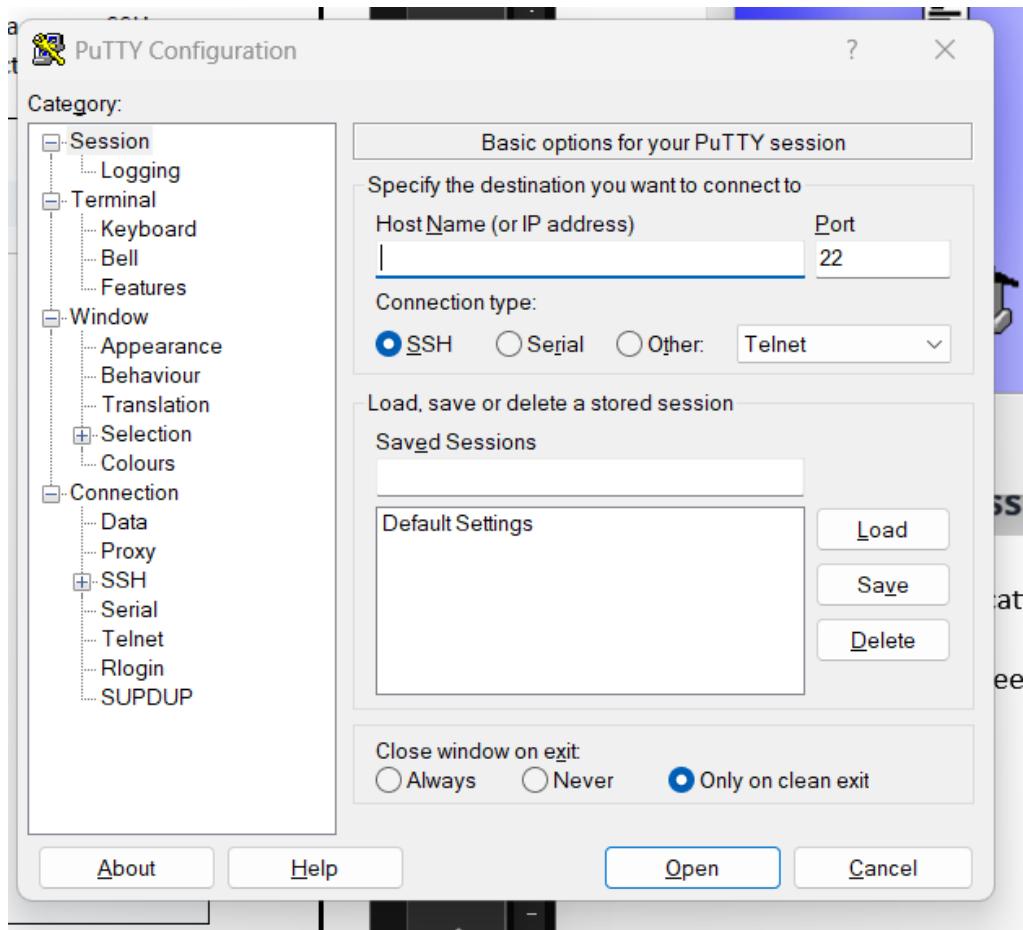


- e) Once downloaded, double-click the installer and proceed by clicking "Next" through each step.
- f) Complete the standard installation process by clicking "Finish" at the end.



Now the PuTTY application is installed and ready to be used for establishing a secure SSH connection between your local system and the Azure VM to begin the project setup.

9.1 Key Fields in PuTTY Configuration



9.1.1 Host Name (or IP address)

- Here you enter the **public IP address** of your Azure Virtual Machine.
- Example: 52.176.xxx.xxx
- You can find this in the **Azure Portal → Virtual Machines → Networking → Public IP Address**.

9.1.2 Port

- Default SSH port is 22.
- Only change this if you configured your VM with a custom SSH port.

9.1.3 Connection type

- Select **SSH** (default and secure).

- Other options like Telnet or Serial are rarely needed for Azure VMs.

9.1.4 Saved Sessions

- You can save this session by entering a name (e.g., AzureVM-Hive) and clicking **Save**.
- Next time, just select it and hit **Load → Open** instead of typing everything again.

9.1.5 Close Window on Exit

- **Only on clean exit** is selected by default (the window only closes if the session ends normally).

9.2 Next Steps after Configuration

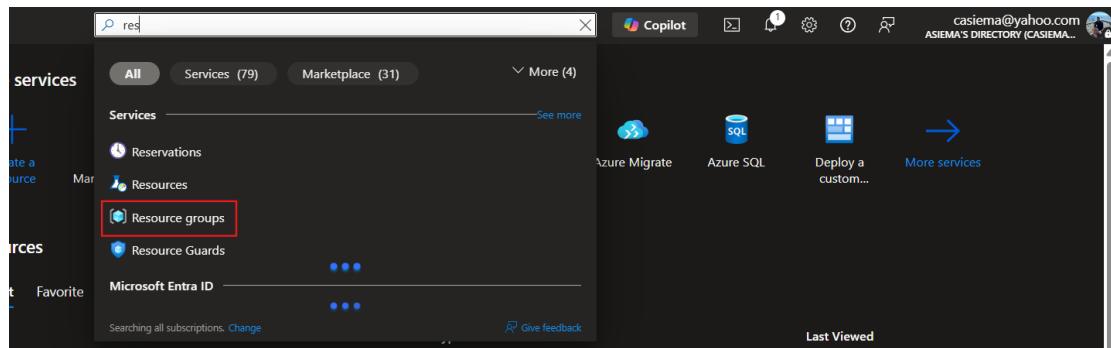
- Click **Open**.
- A terminal window will appear asking:
 - login as: → Enter your **Azure VM username** (from when you created the VM).
 - Password: → Enter the **password** (or provide SSH private key if key-based auth is enabled).
- Once logged in, you can run Linux commands, install Hadoop/Hive, and manage the VM.

10 Creating Azure Resource Group

A Resource Group is created in Azure to manage all Azure Services effectively in one place. It is easier to monitor, update, or delete all Azure resources together as a single unit.

Resource Group

- In the search bar, enter "**Resource Groups**" and select it from the suggestions.



- In the top-left corner, click on **Create**.

A screenshot of the Azure portal's 'Resource groups' page. At the top, there is a search bar with the placeholder 'Search resources, services, and docs (G+/-)'. Below the search bar, the 'Home' link is visible. The main title is 'Resource groups' with a 'Create' button highlighted with a red box. Other buttons include 'Manage view', 'Refresh', 'Export to CSV', 'Open query', and 'Assign tags'. A message at the top says 'You are viewing a new version of Browse experience. Click here to access the old experience.' Below this, there are filter options: 'Filter for any field...', 'Subscription equals all', 'Location equals all', and 'Add filter'. The main content area displays a table of resource groups. The columns are 'Name ↑' (sorted by name), 'Subscription' (with a dropdown arrow), and three ellipsis buttons. The listed groups are: 'admin' (Subscription: Data Science Portfolio), 'app-group' (Subscription: Data Science Portfolio), 'az-group-rg' (Subscription: Data Science Portfolio), and 'AzureBackupRG_southafricanorth_1' (Subscription: Data Science Portfolio).

- In the **Resource Group Name** field, enter the desired name for your resource group. Then, click on **Next: Tags**

Home > Resource groups >

Create a resource group

Basics Tags Review + create

Resource group - A container that holds related resources for an Azure solution. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization. [Learn more](#)

Subscription * ⓘ Data Science Portfolio

Resource group name * ⓘ P001

Region * ⓘ (Africa) South Africa North

- d) Under **Tags**, in the **Name** field, enter something like *developer*. In the **Value** field, enter your name.

Home > Resource groups >

Create a resource group

Basics Tags Review + create

Owner	:	Christian.Aseima	Resource group
Department	:	IT	Resource group
CostCenter	:	IT-P001	Resource group
Project	:	ProjectPro001 Hive	Resource group
Environment	:	Test	Resource group
Lifecycle	:	Pilot	Resource group
Application	:	DataAPI	Resource group
Service	:	VMCluster	Resource group
Region	:	SouthAfricaNorth	Resource group
DataClassification	:	Confidential	Resource group
Compliance	:	DevOps-OnCall	Resource group
Contact	:	DevOps-OnCall	Resource group

10.1 Key Tags for Resource Groups (and Why They Matter)

10.1.1 Cost Management & Billing

- **CostCenter** (e.g., Finance-101, IT-205)

→ Ties resources to a finance/budget code.

- **Department** (e.g., Marketing, R&D)

Lets you allocate cloud spend across departments.

- **Project** (e.g., DataPipeline2025)

Tracks cost at a project level. It is very useful if multiple projects share the same subscription.

📌 *Why?* These tags let you **break down Azure costs by department/project** in Cost Management reports.

10.2 Ownership & Accountability

- **Owner** (e.g., christian.asiema@domain.com)

Identifies who is responsible for the resource group.

- **Contact** (e.g., DevOps-OnCall-Team)

→ For support or escalation.

📌 *Why?* Helps avoid “orphaned resources” that rack up costs when no one knows who owns them.

10.3 Environment & Lifecycle

- **Environment** (e.g., Prod, Dev, Test, UAT)

Differentiates production from non-production workloads.

- **Lifecycle or Stage** (e.g., POC, Pilot, Active, Decommission)

Helps in automation (e.g., auto-delete all POC resources after 30 days).

 **Why?** Supports governance policies and helps apply different SLAs/security settings.

10.4 Security & Compliance

- **DataClassification** (e.g., Confidential, Internal, Public)

Marks the sensitivity of data inside.

- **Compliance** (e.g., GDPR, POPIA, HIPAA)

Useful for audits and regulatory alignment.

 **Why?** Ensures correct handling of resources during security reviews.

10.5 Operational Management

- **Application** (e.g., CustomerPortal, FraudDetectionAPI)

Associates the group with a business application.

- **Service** (e.g., AppService, VM-Cluster)

Clarifies the service type.

- **Region** (e.g., EastUS, SouthAfricaNorth)

Indicates deployment region.

📌 *Why?* Supports operational dashboards and **automation scripts** (e.g., shut down non-Prod VMs after 6 PM).

10.6 Summary of Must-Have Tags

Tag Name	What It Represents	Example Value
Owner	Person/team responsible for the resource group. Helps with accountability.	christian.asiema@domain.com
Department	Business department consuming the resource. Useful for chargeback/showback.	Finance
CostCenter	Financial code for billing and cost tracking.	IT-205
Project	Identifies the project the resources belong to.	DataPipeline2025
Environment	Defines stage of usage (Prod/Dev/Test/UAT).	Prod
Lifecycle/Stage	Indicates resource lifecycle (POC, Pilot, Active, Decommission).	POC
Application	The business app or service that uses the resources.	FraudDetectionAPI
Service	Type of service/workload hosted.	AppService, VMCluster
Region	Geographic deployment location.	SouthAfricaNorth
DataClassification	Sensitivity of data handled. Important for compliance/security.	Confidential
Compliance	Regulatory or legal framework applied.	GDPR, c
Contact	On-call team or support group.	DevOps-OnCall

- e) Then Click on the **Next: Review+Create** and then **Create option**, resource group will be created. Now, this resource group will be used as a central location to create Azure Service in the project.

11 Creating Azure VM

In this step, we will create an Azure Virtual Machine (VM) a cloud-hosted server rented from Microsoft Azure. This VM will serve as the core computing environment where we will install Hadoop and Hive. The purpose is to set up a processing framework capable of handling large-scale data and executing HiveQL queries efficiently. After creating the VM, we will access it remotely and configure the required software stack. This setup will allow us to simulate a real-world big data processing pipeline in a cloud environment.

11.1 Architecture



11.2 Key Components in the Diagram

11.2.1 Hive Client

- This is the interface where the user submits Hive queries.
- It could be via **Hive CLI** or through external applications that connect to Hive.

11.2.2 Hive Server 2

- Acts as the gateway between clients and Hive execution.
- Accepts queries from clients (like JDBC/ODBC tools).
- Handles authentication, query parsing, and forwarding.

11.2.3 Beeline

- A command-line interface that interacts with HiveServer2.
- More secure and flexible than the older Hive CLI.

11.2.4 Driver

- Manages query execution.
- Tasks include parsing the query, optimizing it, and generating an execution plan.
- Communicates with the **Metastore** to get schema and metadata.

11.2.5 Metastore

- Stores metadata about Hive tables (schemas, partitions, locations in HDFS).
- Critical for query execution - Hive queries won't work without it.

11.2.6 Execution Engine (MapReduce / YARN)

- **MapReduce** or **YARN** executes the actual jobs on the Hadoop cluster.

- Hive translates queries (HiveQL) into **MapReduce or Tez jobs** for distributed execution.

11.2.7 HDFS (Hadoop Distributed File System)

- The storage layer where data actually resides.
 - Hive queries ultimately read/write data from HDFS.
-

11.3 Process Flow (Query Lifecycle)

1. User submits a query from **Hive Client/Beeline**.
 2. **HiveServer2** receives it and forwards it to the **Driver**.
 3. **Driver** checks the **Metastore** for schema/table details.
 4. Driver compiles the query into an execution plan then submits to **YARN/MapReduce**.
 5. The job runs on data stored in **HDFS**.
 6. Results are sent back through HiveServer2 to the **Client**.
-

11.3.1 Layman's term

Hive in Simple Words

Imagine Hive as a **restaurant** that serves data instead of food.

1. Hive Client (The Customer)

- You (the user) walk into the restaurant and **place an order** (your query).

- The order could be: “Bring me the total sales from last month.”

2. Hive Server 2 (The Waiter)

- The waiter takes your order and passes it to the kitchen.
- This is HiveServer2 it listens to what you want and makes sure it's valid.

3. Driver (The Chef)

- The chef figures out **how to prepare the dish**.
- In Hive, the driver **breaks down your query** into steps the kitchen can follow.

4. Metastore (The Recipe Book)

- The chef checks the recipe book to see where the ingredients are kept and how they should be used.
- In Hive, the **Metastore** stores all the “recipes” about your data tables - like where they live and what they look like.

5. Execution Engine: MapReduce / YARN (The Kitchen Staff)

- The chef hands over tasks to the kitchen staff (MapReduce/YARN).
- They actually **cook the food** by processing the data in parallel.
- Each cook (worker node) handles part of the order, then everything is combined.

6. HDFS (The Storage/Fridge)

- The ingredients (data) are stored in the restaurant’s **fridge** (HDFS).
- That’s where all the raw data lives until someone needs it.

7. Result Back to Client

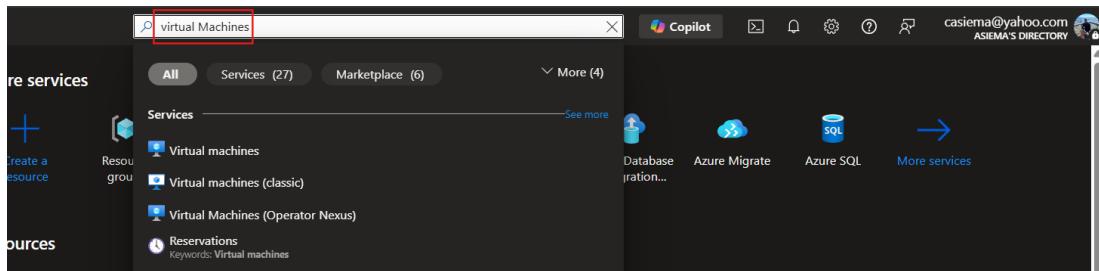
- Once the food is ready, the waiter (HiveServer2) brings it back to your table (Hive Client).
 - In Hive terms: you get the results of your query back.
-

Simple analogy:

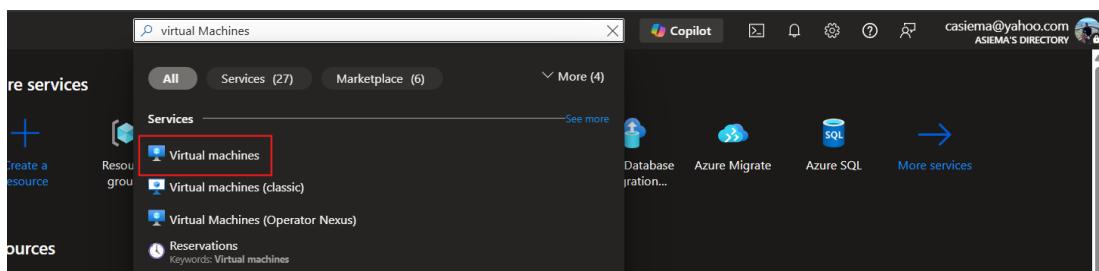
- Hive Client = Customer
- HiveServer2 = Waiter
- Driver = Chef
- Metastore = Recipe book
- MapReduce/YARN = Kitchen staff
- HDFS = Fridge/storage
- Query Results = Your served meal

12 Create Virtual Machine

- a) Navigate to the Azure Portal and search for **Virtual Machines** in the search bar.



- b) Click on the **Virtual Machines** service to open the VM dashboard.



- c) Click on **Create** and then select **Virtual Machine** to initiate the setup process.

The screenshot shows the Azure Compute Infrastructure Virtual Machines dashboard. On the left, a sidebar menu includes 'Overview', 'All resources', and 'Infrastructure' (with 'Virtual machines' selected). The main content area has a 'Create' button (highlighted with a red box) and three cards:

- Virtual machine**: Best for lower-traffic workloads, testing, or to control or highly customize apps, OS, or file system. If your workload or traffic starts to grow, a VM can later be attached to a Virtual Machine Scale Set (VMSS).
- Virtual machine scale set (VMSS)**: Built-in scaling, performance optimization, load balancing, and batch management for 1 to 1,000 VMs (no added cost). Include multiple VM sizes, zones, regions, and domains, along with discounted Spot VMs.
- Presets**: Create a pre-configured VM designed to optimize memory, capacity, or for general use. Deploy as-is, or customize as needed.

On the right, there's a message: 'No virtual machines to display'. It also shows links for 'Windows virtual machines' and 'Linux virtual machines'.

- d) Under **Resource Group**, select the one you previously created for this project.

Home > Compute infrastructure | Virtual machines >

Create a virtual machine

Help me create a VM optimized for high availability Help me choose the right VM

Basics Disks Networking Management Monitoring Advanced Tags Review + create

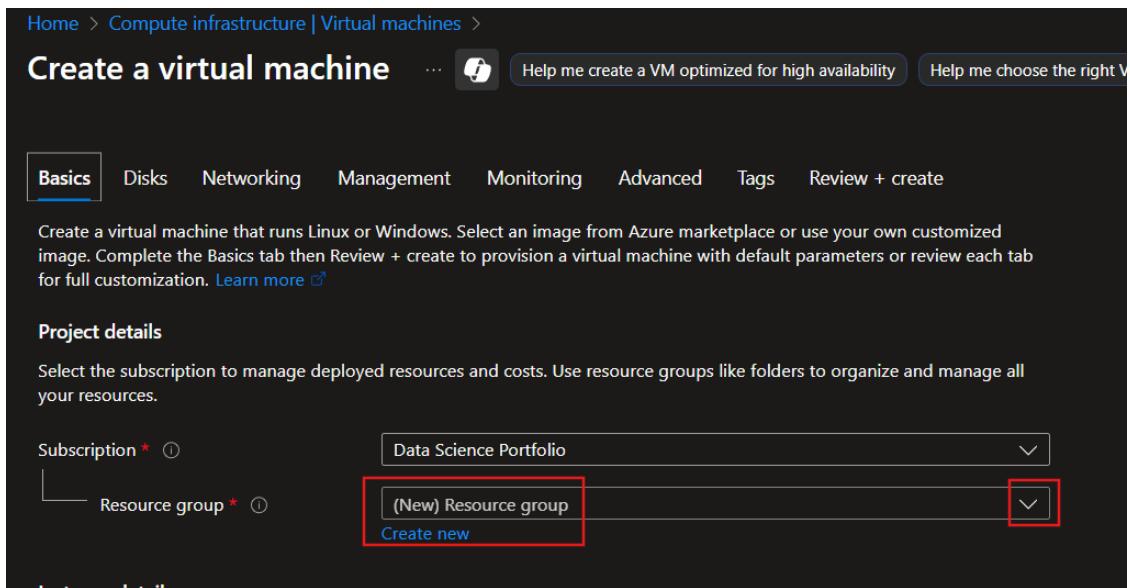
Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image. Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization. [Learn more](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ Data Science Portfolio

Resource group * ⓘ (New) Resource group Create new



- e) Specify the **name** of your Virtual Machine-this will be used to identify your VM in Azure.

Home > Compute infrastructure | Virtual machines >

Create a virtual machine

Help me create a VM optimized for high availability Help me choose the right VM

Subscription * ⓘ Data Science Portfolio

Resource group * ⓘ P001 Create new

Instance details

Virtual machine name * ⓘ ProjectPro-P001

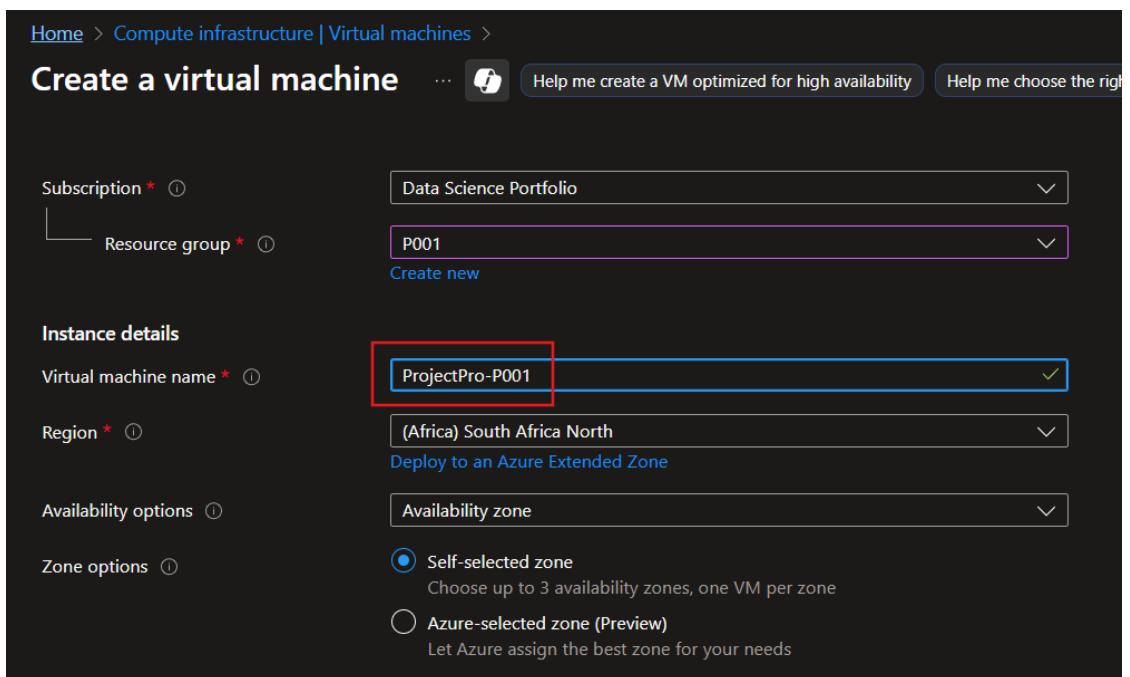
Region * ⓘ (Africa) South Africa North Deploy to an Azure Extended Zone

Availability options ⓘ Availability zone

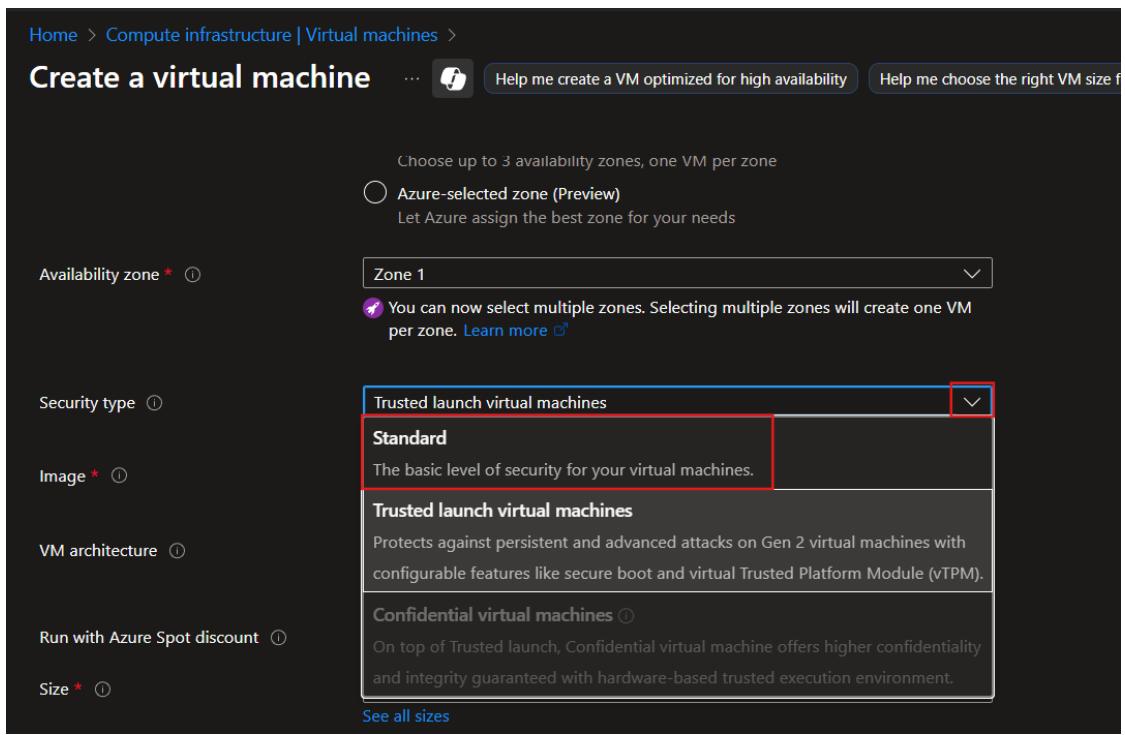
Zone options ⓘ

Self-selected zone Choose up to 3 availability zones, one VM per zone

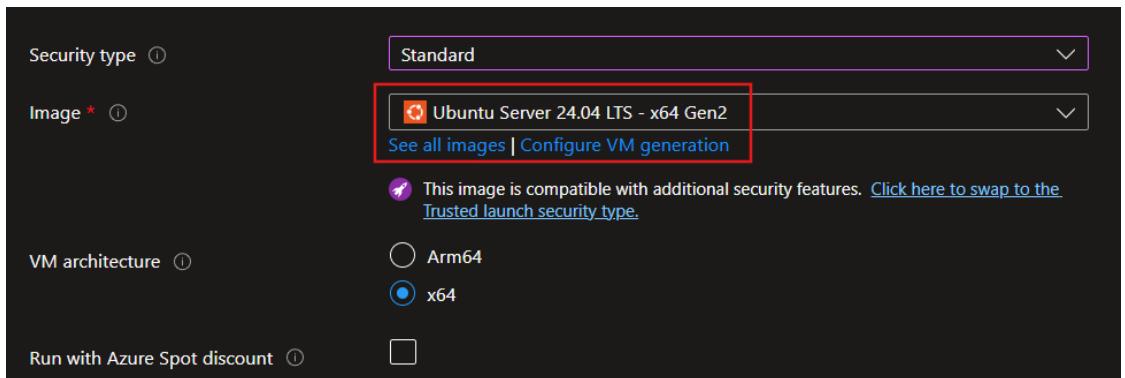
Azure-selected zone (Preview) Let Azure assign the best zone for your needs



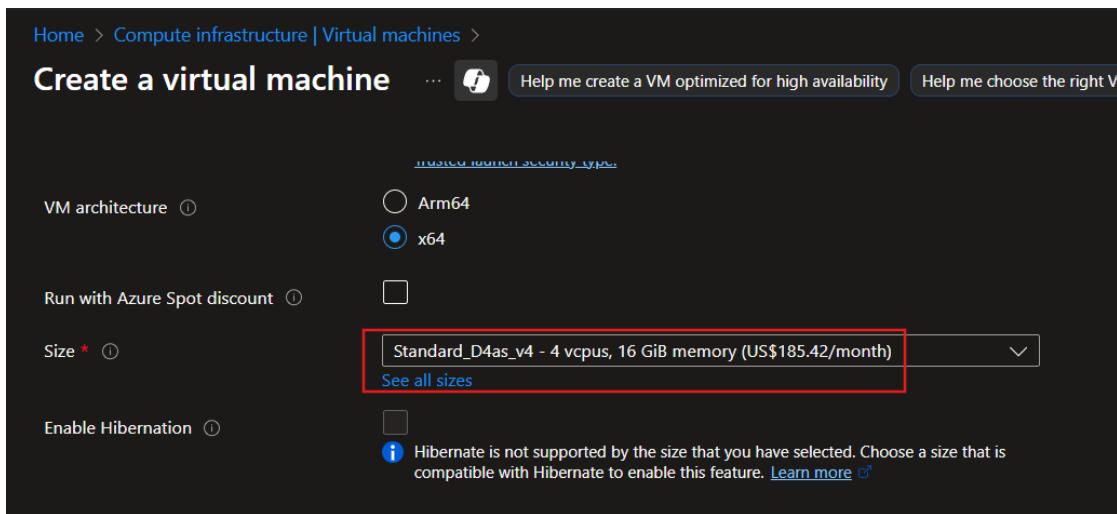
- f) From the **Security type** , select **Standard** to ensure a minimal yet efficient VM that reduces cost.



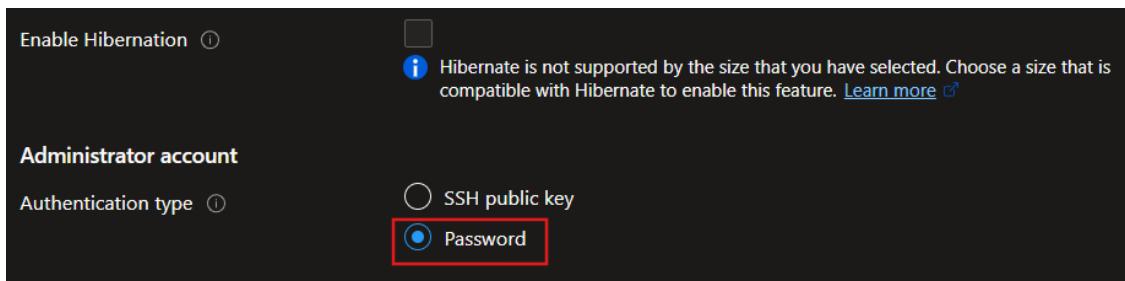
- g) In the **Image** field, select **Ubuntu** because it's open-source, lightweight, and widely used for running big data tools like Hadoop and Hive.



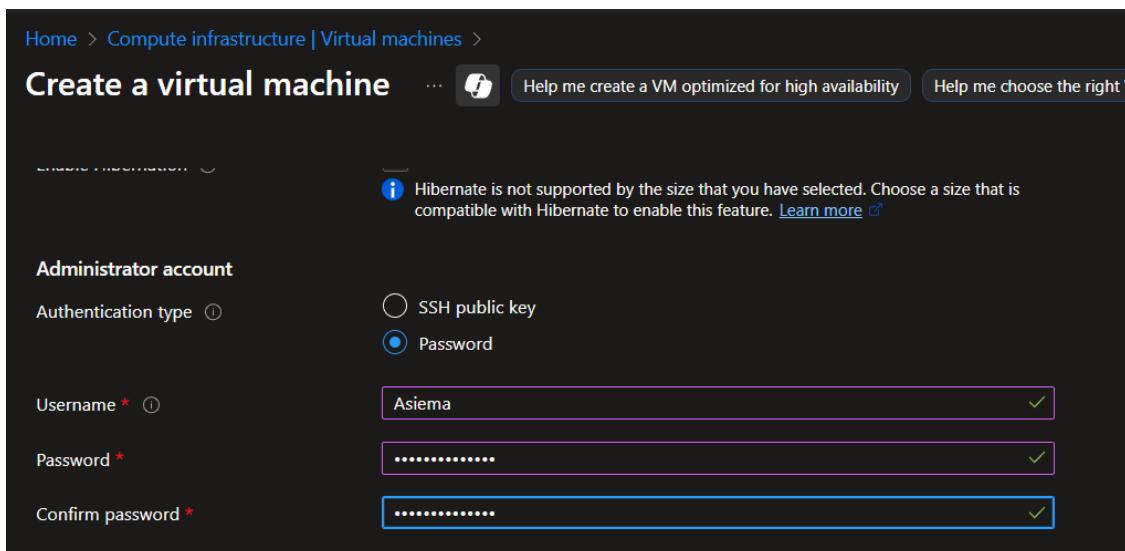
- h) Under the **Size** section, choose **D4as_v4** (4 vCPUs, 16 GiB RAM) a balanced, general-purpose instance type ideal for this use case.



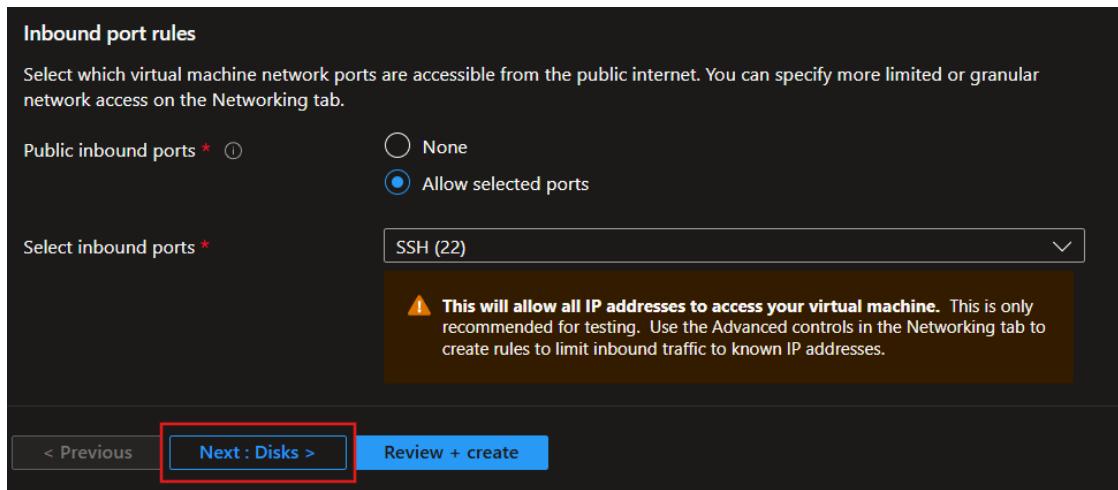
- i) For the Authentication type, make sure it is selected as a password. Since the connection to the Azure VM will be established using a password.



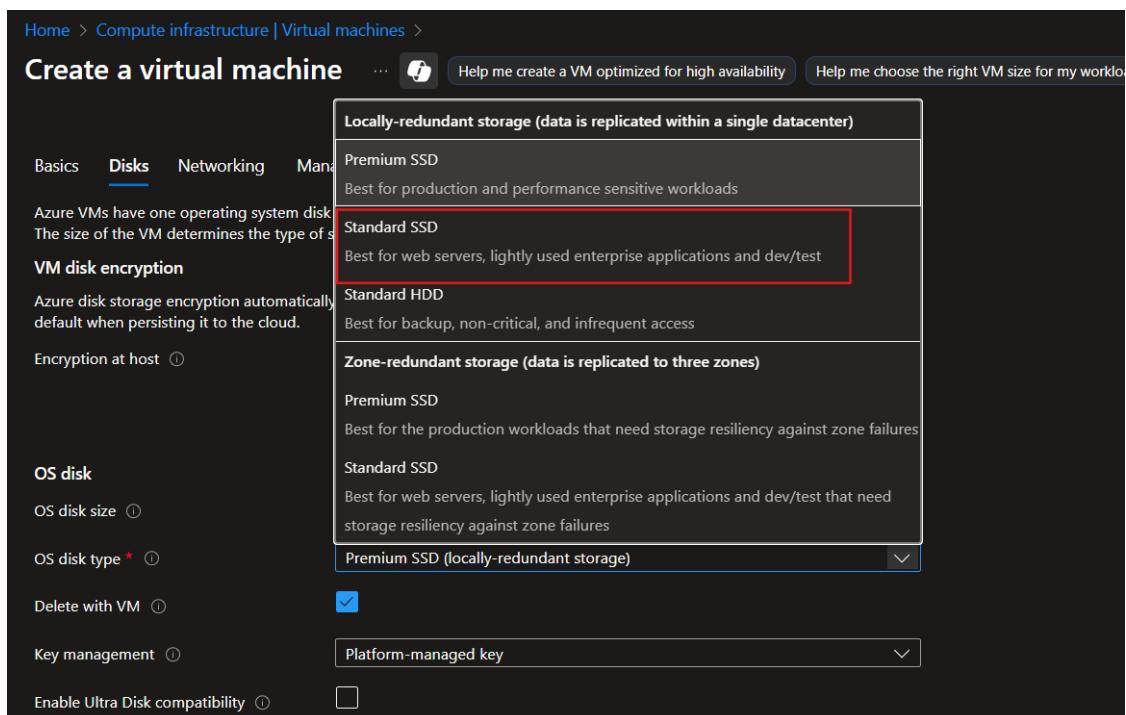
In the username and password field, specify the username and password in the prompt and save it for the connection to the Azure VM.



- j) Click **Next: Disks** to move to the storage configuration step. We need to define the disk and storage type for performance and cost-effectiveness.



- k) For **OS Disk Type** , choose **Standard SSD** .



Then under **Data Disks** , click **Create and attach a new disk** to add extra storage to your VM.

Delete with VM ⓘ	<input checked="" type="checkbox"/>				
Key management ⓘ	Platform-managed key <input checked="" type="checkbox"/>				
Enable Ultra Disk compatibility ⓘ	<input type="checkbox"/>				
Data disks for ProjectPro-P001					
You can add and configure additional data disks for your virtual machine or attach existing disks. This VM also comes with a temporary disk.					
LUN	Name	Size (GiB)	Disk type	Host caching	Delete with VM ⓘ
Create and attach a new disk		Attach an existing disk			

Home > Compute infrastructure | Virtual machines > Create a virtual machine >

Create a new disk

Create a new disk to store applications and data on your VM. Disk pricing varies based on factors including disk size, storage type, and number of transactions. [Learn more](#)

Name *	<input type="text" value="ProjectPro-P001_DataDisk_0"/>
Source type * ⓘ	<input type="text" value="None (empty disk)"/>
Size * ⓘ	<p>1024 GiB Premium SSD LRS Change size</p>
Key management ⓘ	<input type="text" value="Platform-managed key"/>
Enable shared disk	<input type="radio"/> Yes <input checked="" type="radio"/> No
Delete disk with VM	<input type="checkbox"/>

- l) For **Storage Type**, select **Standard SSD (LRS)** to optimize cost and performance.

Home > Compute infrastructure | Virtual machines > Create a virtual machine > Create a new disk >

Select a disk size

Browse available disk sizes and their features.

Storage type ⓘ

Size	Performance tier	Provisioned IOPS	Provisioned throughput	Max Shares ⓘ	Max burst IOPS ⓘ	Max burst throughput ⓘ
4 GiB	E1	500	100	3	600	150
8 GiB	E2	500	100	3	600	150
16 GiB	E3	500	100	3	600	150
32 GiB	E4	500	100	3	600	150
64 GiB	E6	500	100	3	600	150
128 GiB	E10	500	100	3	600	150
256 GiB	E15	500	100	3	600	150
512 GiB	E20	500	100	3	600	150
1024 GiB	E30	500	100	5	1000	250
2048 GiB	E40	500	100	5	-	-
4096 GiB	E50	500	100	5	-	-
8192 GiB	E60	2000	400	10	-	-
16384 GiB	E70	4000	600	10	-	-
32767 GiB	E80	6000	750	10	-	-

- m) Set the new disk size to **128 GiB** -a sufficient and cost-effective capacity for our processing needs.

Storage type ⓘ						
Size	Performance tier	Provisioned IOPS	Provisioned throughput	Max Shares ⓘ	Max burst IOPS ⓘ	Max burst throughput ⓘ
4 GiB	E1	500	100	3	600	150
8 GiB	E2	500	100	3	600	150
16 GiB	E3	500	100	3	600	150
32 GiB	E4	500	100	3	600	150
64 GiB	E6	500	100	3	600	150
128 GiB	E10	500	100	3	600	150
256 GiB	E15	500	100	3	600	150
512 GiB	E20	500	100	3	600	150
1024 GiB	E30	500	100	5	1000	250

- n) Enable the checkbox **Delete disk with VM** so that when the VM is deleted, the attached disk will also be automatically removed. Click **OK** to confirm.

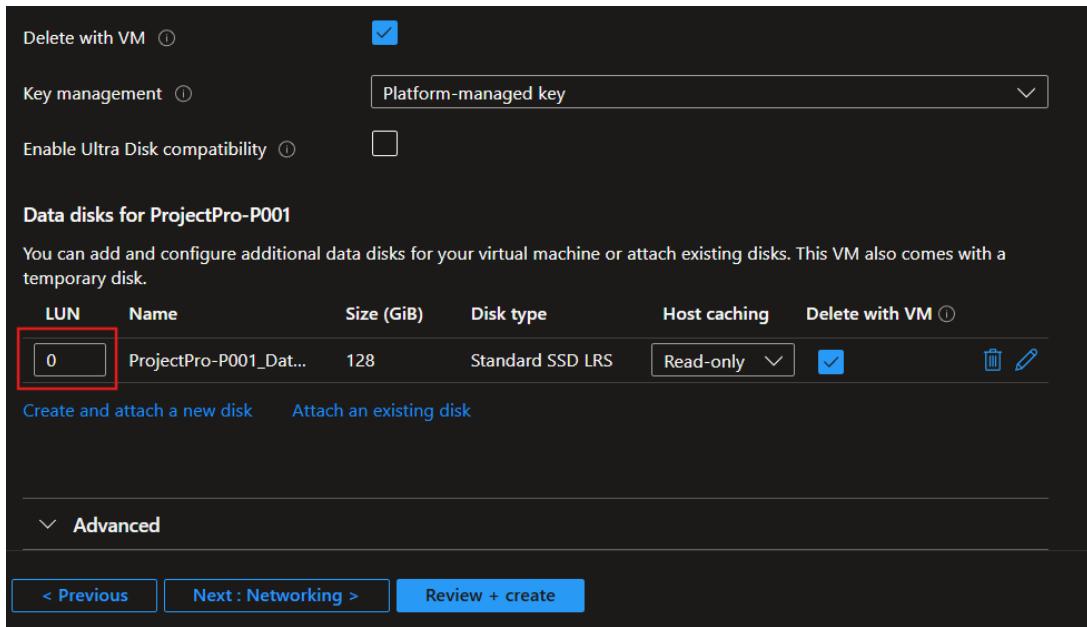
Home > Compute infrastructure | Virtual machines > Create a virtual machine >

Create a new disk

Create a new disk to store applications and data on your VM. Disk pricing varies based on factors including disk size, storage type, and number of transactions. [Learn more ↗](#)

Name *	ProjectPro-P001_DataDisk_0
Source type * ⓘ	None (empty disk)
Size * ⓘ	128 GiB Standard SSD LRS Change size
Key management ⓘ	Platform-managed key
Enable shared disk	<input type="radio"/> Yes <input checked="" type="radio"/> No
Delete disk with VM	<input checked="" type="checkbox"/>

- o) Specify the **LUN** (Logical Unit Number) as 0 . This defines the order in which disks are attached- LUN 0 means it's the first attached disk (D1) with a size of 128 GiB.



- p) Continue clicking **Next** on all subsequent pages to skip additional configuration.
- q) On the final screen, click **Next: Review + Create** and then **Create**. Your Azure VM will be provisioned and available for use under your specified resource group.

Now, the Azure Virtual Machine (VM) has been successfully created, which is essential for executing the project. All necessary components such as Hive, Hadoop, and related frameworks will be installed and set up on this VM. This environment will be used to process and analyze the **airline dataset** using Hive queries and other optimization techniques.

Note: Please make sure to monitor the Azure VM continuously to keep track of resource consumption and manage costs effectively. It is highly recommended to stop the VM when not in use and delete it once the project execution is complete. If the VM is left running, it will continue to accumulate charges, which could result in unexpectedly high costs.

12.1 Azure VM Connectivity Setup

After successfully creating the Azure Virtual Machine (Azure VM), the next step is to establish a secure connection from your local system using the PuTTY application. *PuTTY acts as the client application that allows SSH-based remote access to the Azure VM using your credentials.*

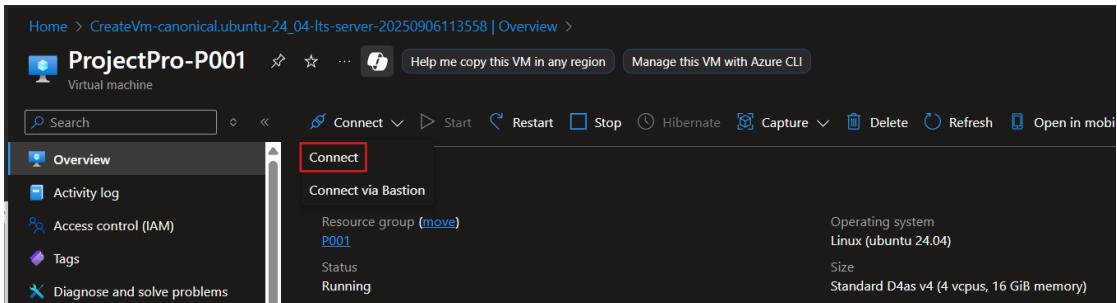
PuTTY is like a **remote-control app** for your computer in the cloud (your Azure VM).

- Your **Azure VM** is a computer sitting somewhere in Microsoft's datacenter.
- Instead of physically touching it, you need a way to “open the door” and control it from your own laptop.
- **SSH** is the “secure key” that unlocks the door safely so no one else can sneak in.

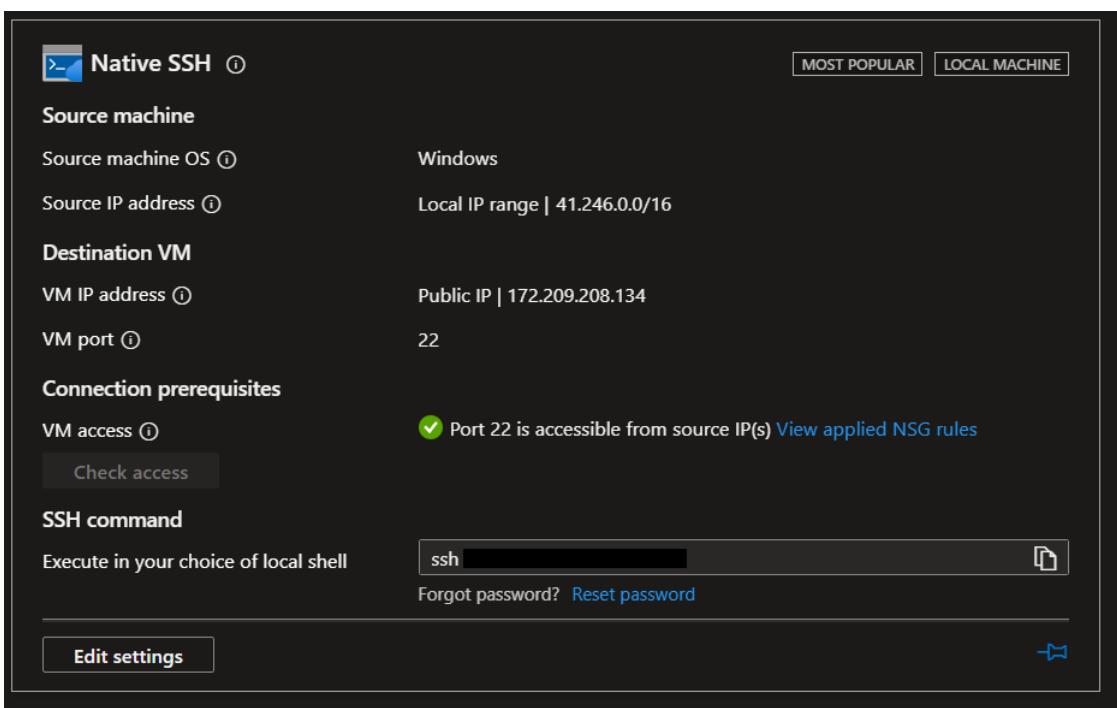
- **PuTTY** is the **remote-control handset** - it's the little program you run on your own computer that lets you type commands and control the Azure VM as if you were sitting right in front of it.

To connect, you need to follow this approach:

- a) Open the Azure Portal and navigate to the Virtual Machine that you created.

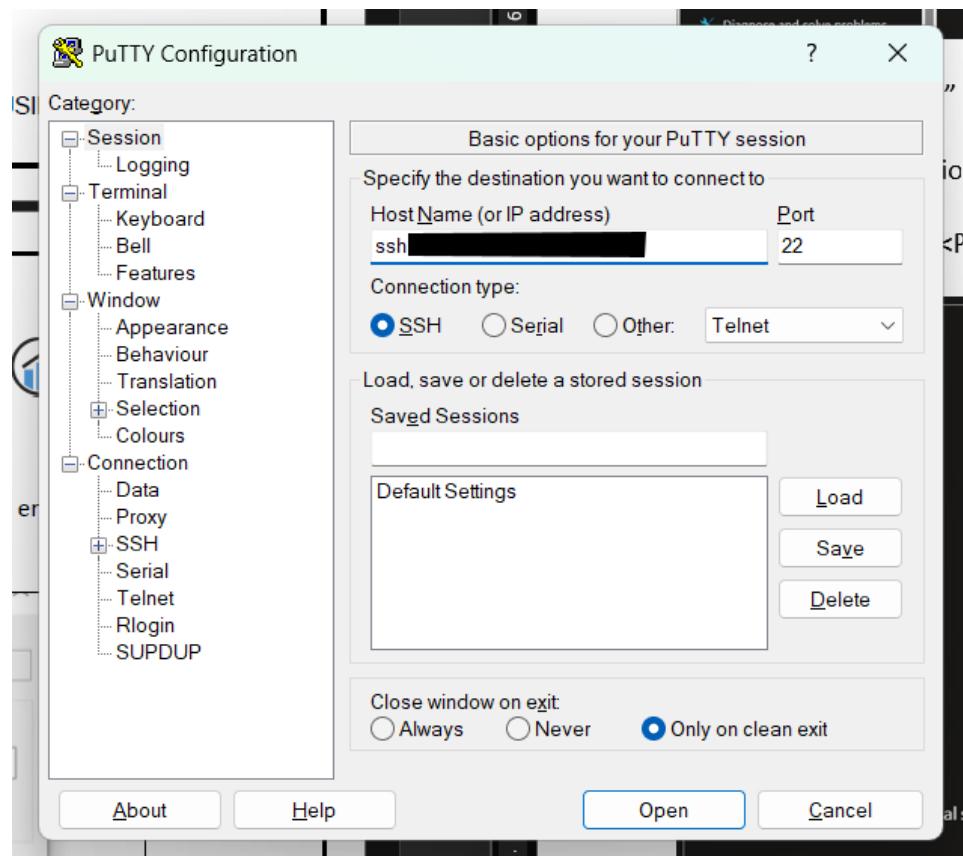


- b) Click on the “Connect” button on the top bar of the VM overview page, then choose “SSH” as the connection method. Copy the provided SSH string, which is typically in the format: ubuntu@<Public_IPv4_Address>

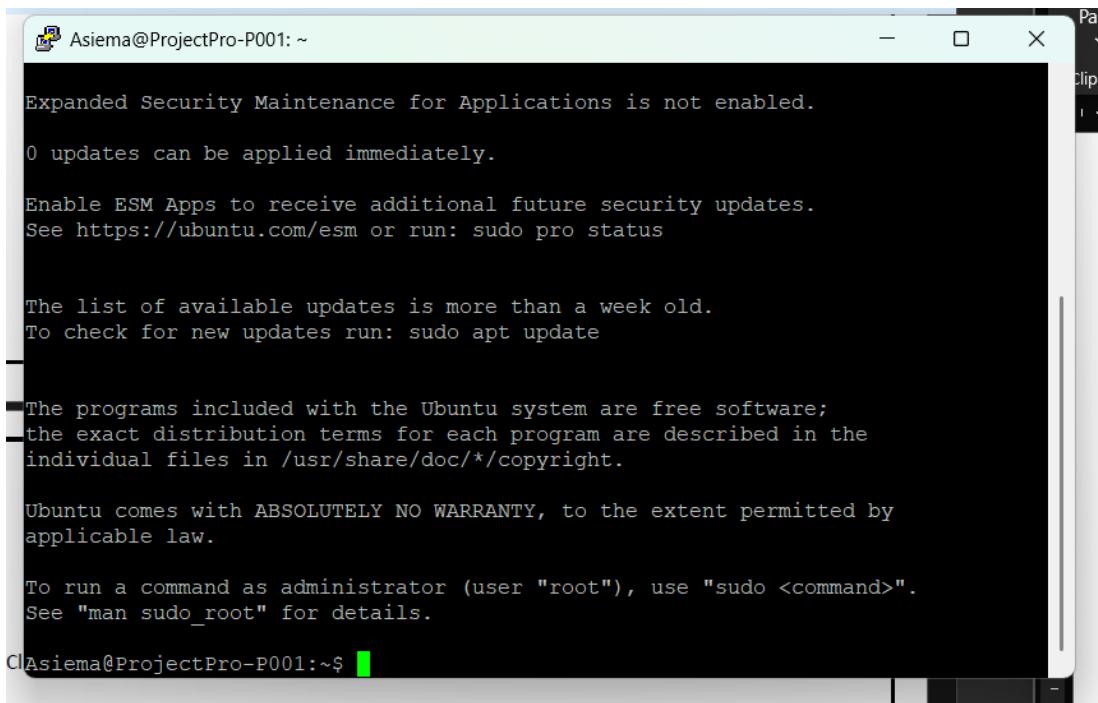


- c) Open the PuTTY application on your local system.

- d) In the Host Name (or IP address) field, paste the copied SSH string or enter the public IP address of your Azure VM.



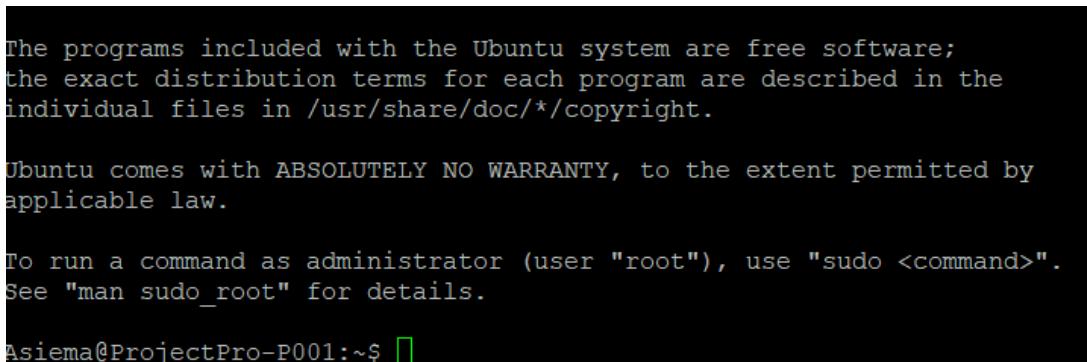
- e) Make sure the Port is set to 22 and the Connection Type is set to SSH .
- f) Click “Open” to start the connection.
- g) A terminal window will open. It may display a security alert the first time – click “Yes” to proceed.



A screenshot of a terminal window titled "Asiema@ProjectPro-P001: ~". The window displays several lines of text about system updates:

```
Expanded Security Maintenance for Applications is not enabled.  
0 updates can be applied immediately.  
Enable ESM Apps to receive additional future security updates.  
See https://ubuntu.com/esm or run: sudo pro status  
  
The list of available updates is more than a week old.  
To check for new updates run: sudo apt update  
  
The programs included with the Ubuntu system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/*copyright.  
  
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by  
applicable law.  
  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.
```

- h) When prompted, enter the password or private key (if configured) that you set during the Azure VM creation.
- i) Upon successful login, you will be connected to your Azure VM and can begin using the terminal interface to install tools or execute commands.



A screenshot of a terminal window titled "Asiema@ProjectPro-P001: ~". The window displays several lines of text about system updates:

```
The programs included with the Ubuntu system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/*copyright.  
  
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by  
applicable law.  
  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.
```

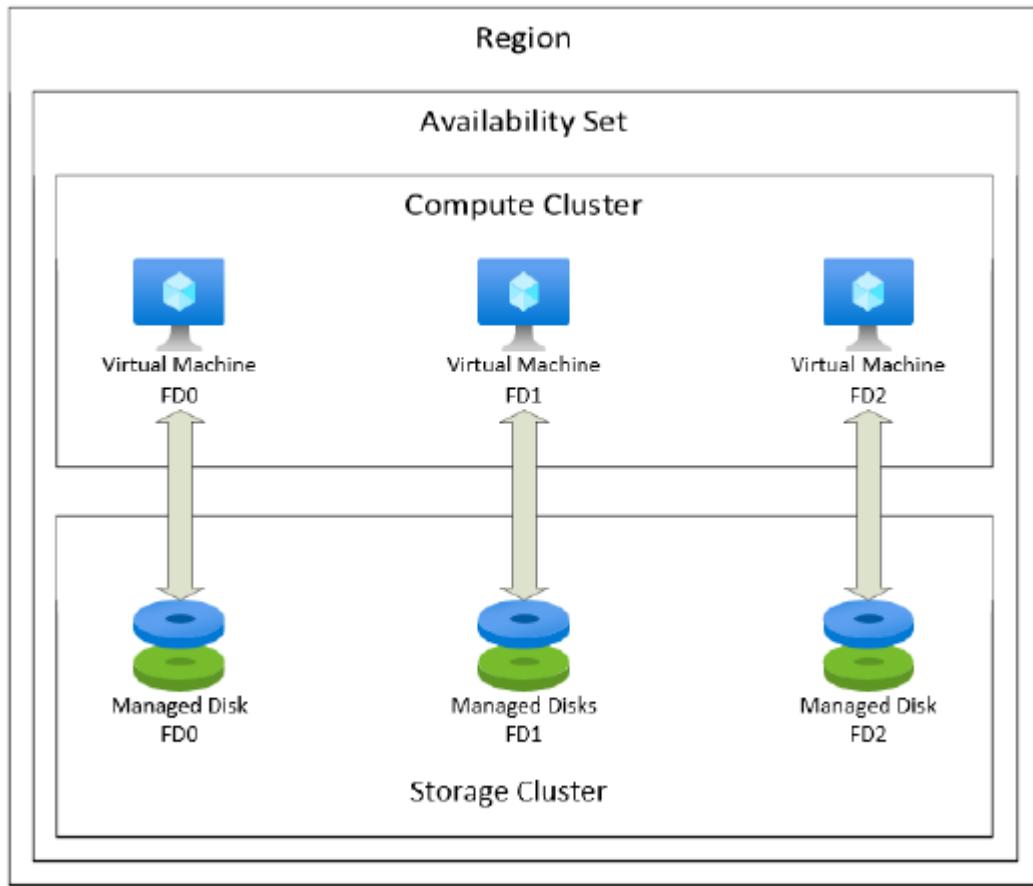
- j) You can now proceed with setting up Hadoop, Hive, or any other required services on the VM.
- k) To save costs, stop the Azure VM when it's not in use instead of letting it run continuously.

Note: If you encounter a "Network timeout" or connection refused error while using PuTTY, it might be due to restricted inbound traffic rules on the Azure VM. In such cases:

- Go to your Azure VM's Networking section in the portal.
- Under Inbound port rules, ensure there is a rule allowing SSH (Port 22) traffic.
- If missing, create a new rule: **Protocol:** TCP and **Port Range: 22**
 - Source: Any (or your IP for added security)
 - Action: Allow
- Once this is configured, try connecting again using PuTTY.

12.2 Disk Mounting

Once the connection to the Azure Virtual Machine (VM) is established using PuTTY, the next crucial step is to mount the attached disk. This must be done before proceeding with the download and setup of Hadoop and Hive frameworks on the VM. Mounting is essential because the disk acts as the storage location where all Hadoop data, Hive warehouse files, and configurations will reside. Unlike AWS or GCP, Azure requires manual disk mounting, as additional data disks attached to an Azure VM are not pre-formatted or auto mounted. In Azure, we must explicitly partition, format, and mount the disk using commands executed from within the VM. Without this step, the big data frameworks won't have a functional path to store and process data.



Mounting is essential before installing Hadoop and Hive because: Both frameworks rely on storage directories for configuration, data warehousing, and job processing. Without a mounted and accessible disk, the installation paths and data locations won't work as expected.

To mount the disk, follow the series of commands listed below. You can find these commands in the **mountingthedisks.txt** file from the code folder or run them directly in the Azure VM terminal.

Each command is explained step-by-step along with its purpose and the expected output. This is the explanation of each command to run this to attach the volume to the Azure VM:

12.2.1 Command 1 : lsblk

What does lsblk mean?

The command `lsblk` stands for "**list block devices**." Think of it like asking your computer:

"Hey, what storage drives do you have plugged in right now?"

What does it show?

It shows **all the hard drives** and storage devices that are connected to your virtual machine (like a computer in the cloud), including:

- The main drive that the system runs on (the one with Windows or Linux installed)
 - Any extra drives that you've added for more space
-

Why is it useful?

Let's say you've plugged in a new hard drive (or Azure added one for you), but you can't see or use it yet. That's because:

- It's still **blank** (has no format or folder system)
- Or it hasn't been **set up** yet

`lsblk` helps you **see those drives**, even the ones you can't use yet, so you know what needs to be prepared.

```

Asiema@ProjectPro-P001:~$ lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda      8:0    0 128G  0 disk
sdb      8:16   0 30G  0 disk
└─sdb1   8:17   0 29G  0 part /
└─sdb14  8:30   0  4M  0 part
└─sdb15  8:31   0 106M 0 part /boot/efi
└─sdb16  259:0  0 913M 0 part /boot
sdc      8:32   0 32G  0 disk
└─sdc1   8:33   0 32G  0 part /mnt
sr0     11:0   1 628K 0 rom
Asiema@ProjectPro-P001:~$ █

```

What Are You Seeing?

NAME	SIZE	MOUNTED AT	WHAT IT IS
sda	128GB	–	A full internal SSD (unused)
sdb	30GB	–	Another disk (your main one)
└ sdb1	29GB	/	Main area where Linux is installed (root)
└ sdb14	4MB	–	Small system partition (used by the OS)
└ sdb15	106MB	/boot/efi	For booting system (EFI boot files)
└ sdb16	913MB	/boot	Stores files to boot up Linux
sdc	32GB	–	Likely a USB or external flash drive
└ sdc1	32GB	/mnt	Manually mounted storage (maybe your personal files)
sr0	628KB	–	Virtual CD/DVD drive (ROM) – probably unused

What Does This All Mean?

- Your computer has **three main storage devices**:
 - sda (128 GB): seems **unused** or blank
 - sdb (30 GB): the **main disk** where **Linux OS is installed**
 - sdc (32 GB): likely a **USB flash drive**, mounted at /mnt so you can access its files
 - The partitions like /boot, /boot/efi, and / are **essential for the system to boot and run**.
 - /mnt is a **manually mounted folder**, meaning you or the system temporarily plugged in a device (like a USB drive).
 - sr0 is likely a **virtual CD-ROM drive**, usually not in use unless you're mounting an ISO.
-



Summary in Simple Terms:

- You're running Linux from a 30GB internal drive (sdb).
- Your system has all the necessary partitions to boot and run correctly.
- You have a **bigger 128GB drive (sda)** that's not being used - you could format and store data there.
- A 32GB USB/external drive is plugged in and accessible at /mnt.
- The CD-ROM (sr0) is just a default virtual drive - probably not used.

12.2.2 Command 2: *sudo parted /dev/sdc*

```
sudo parted /dev/sdc --script mklabel gpt mkpart xfspart xfs 0% 100%
```

What it Does (Simple Terms):

Imagine /dev/sdc is a **blank USB drive or external disk**, and you're getting it **ready for use**. This command does **two big things**:

1. Wipes the Disk and Sets Up a Modern System (GPT)

- Think of this like **removing all walls in a house** and building a **modern layout (GPT)**.
 - GPT is a **newer, smarter way** to organize a disk it supports big drives and avoids old limitations.
-

2. Makes a New Room (Partition) Using the Whole Disk

- The command says: "Create one big section (partition) that uses **the entire disk (0% to 100%)**".
 - It gets this section **ready to use the XFS format** - which is a strong, reliable file system.
-

Analogy:

You're:

- Demolishing old structures on a plot of land (mklabel gpt)
- Building one large, new building that covers the entire land (mkpart)

- Making sure the new building uses **XFS-style plumbing and wiring** (modern and efficient)
-

Summary:

Word	Meaning (Layman's Explanation)
sudo	Run the command with administrator (superuser) permissions - needed to make changes to disks.
parted	A tool used to manage and edit partitions on a hard drive or USB stick.
/dev/sdc	The device name for the disk you're working on (in this case, likely a USB or external drive).
--script	Tells parted to run automatically without asking questions (non-interactive mode).
mklabel	Stands for “ make label ” - it creates a new partition table , which is like the blueprint for organizing the drive.
gpt	The type of partition table to create - GPT (GUID Partition Table) is a modern layout that supports large drives and many partitions.
mkpart	Stands for “ make partition ” - creates a new usable section (partition) on the drive.
xfspart	This is the name or label for the partition. It’s optional, but here it’s called xfspart.
xfs	The file system type to use - xfs is a modern format that’s fast, reliable, and good for large files.
0%	The start point of the partition - means begin at the very beginning of the drive (0%).

100%	The end point of the partition - means use all available space up to the end of the drive (100%).
-------------	--

12.2.3 Command 3: *sudo mkfs.xfs /dev/sdc1 –f*

sudo mkfs.xfs /dev/sdc1 –f

What This Command Does (in Simple Terms):

This command **formats the entire disk /dev/sdc** using the **XFS file system** and **forces the format** even if something is already on it.

⚠ However: This is **not recommended**.

You should **never format /dev/sdc directly**, only the **partition** like /dev/sdc1.

Otherwise, it may damage the partition table.

Explanation (from the image):

- mkfs.xfs: Tool that creates a new **XFS filesystem**
- -f: **Force overwrite**, even if the disk already has data
- XFS is great for **big data, scalability, and high performance**, e.g., Hadoop or Hive environments

Table: Each Part of the Command Explained

Command Part	Meaning
sudo	Run the command with admin privileges (needed to format disks)
mkfs.xfs	Make a new XFS file system on the target device
/dev/sdc	Refers to the whole disk (⚠️ not recommended - you should use <code>/dev/sdc1</code>)
-f	Force the formatting, even if something is already on the disk

12.2.4 **Command 4: sudo partprobe /dev/sdc1**

sudo partprobe /dev/sdc1

What It Means (Layman's Terms):

- When you create or change partitions on a disk, the **operating system doesn't always notice right away**. This command tells Linux:

"Hey, I've changed the map of this disk - please **refresh** your view of it so you know about the new partitions."

- That way, you don't have to **reboot** your computer to make the new partition usable.

Table: Each Part of the Command Explained

Command Part	Meaning (Layman's Terms)
sudo	Run as administrator (needed for disk operations)
partprobe	A tool that tells Linux to re-read the partition table (refreshes the disk layout)
/dev/sdc	The specific disk you're refreshing (in this case, your USB/external drive)

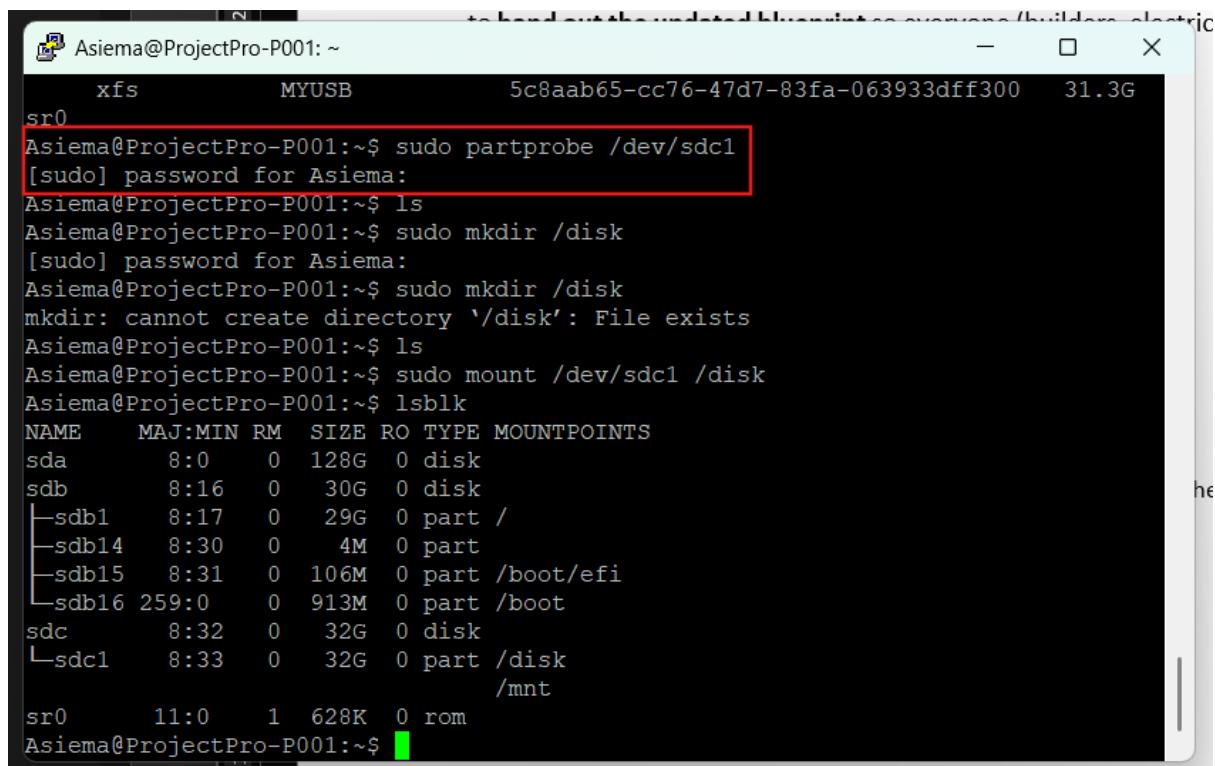
Why It's Important

- Without this, sometimes the system **still thinks the old partitions exist**.
- partprobe ensures that the **kernel** (the core of Linux) and all tools are on the same page about the **current partition layout**.
- It avoids the need for a **reboot** after making changes with parted or fdisk.

Analogy

It's like **redrawing the floor plan** of a house. After you move walls (make partitions), you need to **hand out the updated blueprint** so everyone (builders, electricians, plumbers = Linux tools) knows the new layout.

```
sudo partprobe /dev/sdc1
```



```
Asiema@ProjectPro-P001: ~
sr0
Asiema@ProjectPro-P001:~$ sudo partprobe /dev/sdc1
[sudo] password for Asiema:
Asiema@ProjectPro-P001:~$ ls
Asiema@ProjectPro-P001:~$ sudo mkdir /disk
[sudo] password for Asiema:
Asiema@ProjectPro-P001:~$ sudo mkdir /disk
mkdir: cannot create directory '/disk': File exists
Asiema@ProjectPro-P001:~$ ls
Asiema@ProjectPro-P001:~$ sudo mount /dev/sdc1 /disk
Asiema@ProjectPro-P001:~$ lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda     8:0    0 128G  0 disk
sdb     8:16   0  30G  0 disk
└─sdb1   8:17   0   29G  0 part /
└─sdb14  8:30   0    4M  0 part
└─sdb15  8:31   0 106M  0 part /boot/efi
└─sdb16 259:0   0  913M  0 part /boot
sdc     8:32   0   32G  0 disk
└─sdc1   8:33   0   32G  0 part /disk
                           /mnt
sr0     11:0   1  628K  0 rom
Asiema@ProjectPro-P001:~$
```

What It Means (Layman's Terms):

This command **creates a new empty folder** called /disk.

That folder will act as a “**mount point**” - the place in your system where the contents of the USB/disk will appear once mounted.

Think of it like creating a **doorway**:

- The folder /disk is the doorway.

- Mounting the disk attaches the **drive's contents** to that doorway so you can walk through and see the files.

Table: Each Part of the Command Explained

Command Part	Meaning (Layman's Terms)
sudo	Run as administrator, because creating system-wide folders usually requires extra permission
mkdir	Stands for make directory - creates a new folder
/disk	The path and name of the new folder being created (here it's /disk)

Why It's Important

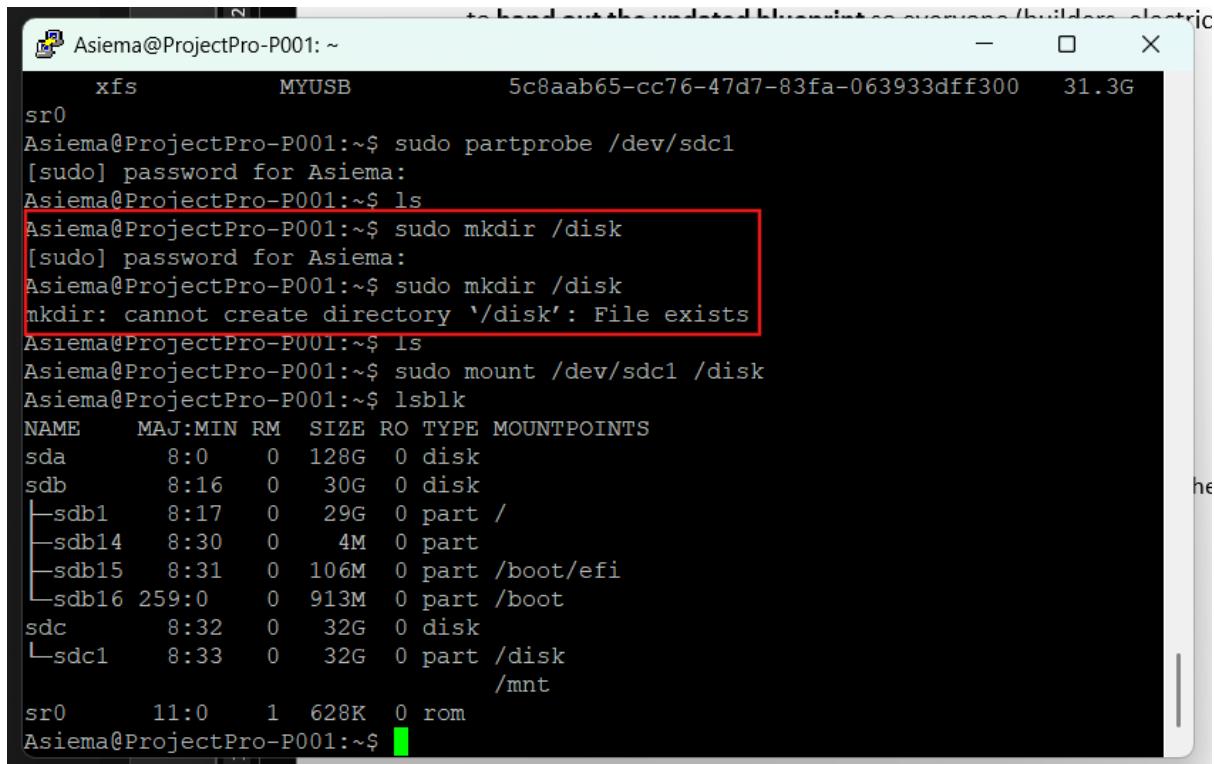
- You **must have a mount point** (an empty folder) before you can attach a disk to it.
 - Without it, Linux won't know **where to show the contents** of the drive.
 - You can name it anything, like /mnt/usb or /media/mydrive, but /disk is short and clear.
-

Analogy

Imagine you bought a new wardrobe (USB drive). Before putting it in your room, you **clear a space (make a folder)** so it has a place to stand. That space is the **mount point**.

12.2.5 Command 5: sudo mkdir /disk

```
sudo mkdir /disk
```



The screenshot shows a terminal window with the following session:

```
Asiema@ProjectPro-P001: ~
xfs          MYUSB           5c8aab65-cc76-47d7-83fa-063933dff300  31.3G
sr0
Asiema@ProjectPro-P001:~$ sudo partprobe /dev/sdc1
[sudo] password for Asiema:
Asiema@ProjectPro-P001:~$ ls
Asiema@ProjectPro-P001:~$ sudo mkdir /disk
[sudo] password for Asiema:
Asiema@ProjectPro-P001:~$ sudo mkdir /disk
mkdir: cannot create directory '/disk': File exists
Asiema@ProjectPro-P001:~$ ls
Asiema@ProjectPro-P001:~$ sudo mount /dev/sdc1 /disk
Asiema@ProjectPro-P001:~$ lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda     8:0    0 128G  0 disk
sdb     8:16   0  30G  0 disk
└─sdb1   8:17   0  29G  0 part /
└─sdb14  8:30   0   4M  0 part
└─sdb15  8:31   0 106M  0 part /boot/efi
└─sdb16 259:0   0 913M  0 part /boot
sdc     8:32   0   32G  0 disk
└─sdc1   8:33   0   32G  0 part /disk
                           /mnt
sr0     11:0    1  628K  0 rom
Asiema@ProjectPro-P001:~$
```

What It Does (Layman's Terms):

- This creates a **new empty folder** named /disk.
- That folder is important because it will serve as the **mount point** - the place where your USB drive or external disk will be attached so you can see its files.
- Without this folder, Linux won't know **where to show the contents** of the drive when you mount it.

Table: Each Word Explained

Word	Meaning (Layman's Terms)
sudo	Run as administrator , because making folders in system-level places requires permission
mkdir	Short for make directory → creates a new folder
/disk	The name and location of the new folder → here it creates a folder called disk directly under / (the root of your system)

Why It's Important

- A **mount point** must exist before you can mount a disk.
 - Once you mount /dev/sdc1, all the files on that drive will show up **inside /disk**.
 - You could call it something else, like /mnt/myusb or /media/storage, but /disk is short and simple.
-

Analogy

Think of this as **making an empty shelf** before you put a new box (your USB drive) on it.

The shelf (/disk) must exist first, or else you have nowhere to place the box.

12.2.6 Command 6: sudo mount /dev /sdc /disk

sudo mount /dev/sdc /disk

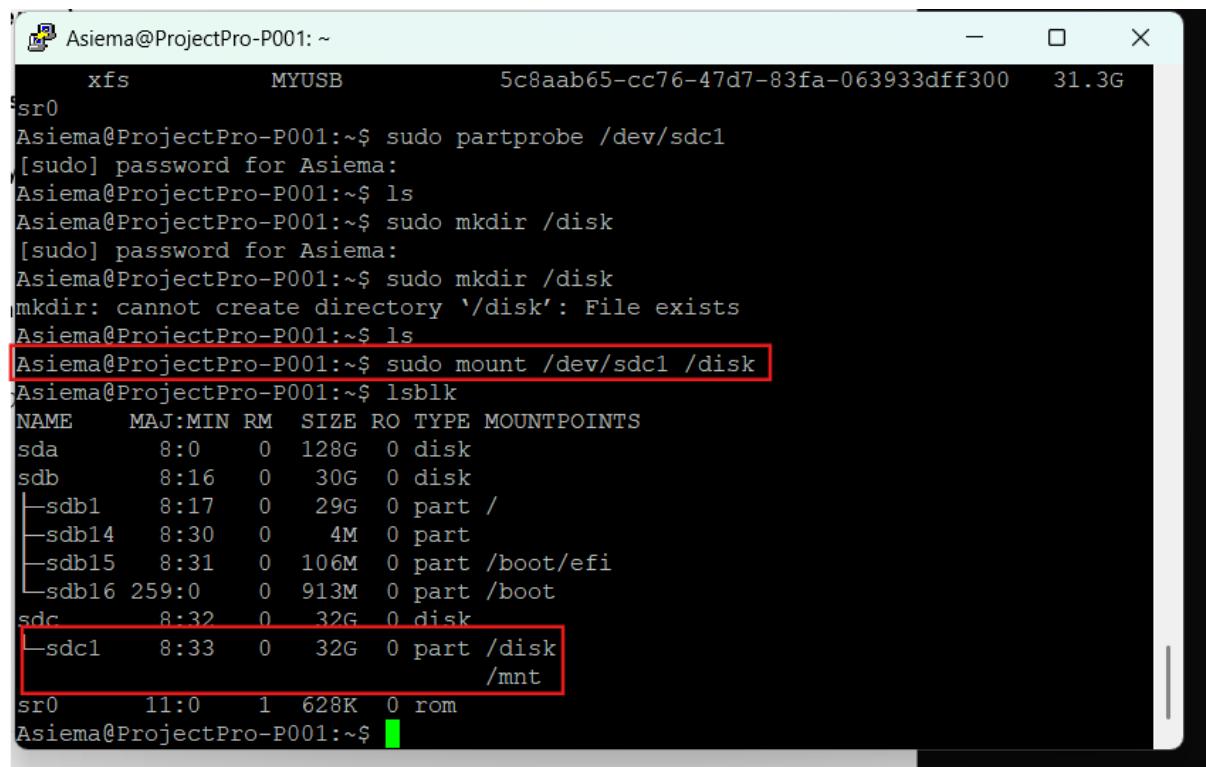
What It Does (Layman's Terms):

- This command **attaches your disk** (/dev/sdc) to the folder /disk. After this, anything you put in /disk will actually be stored on the USB/external drive, not your computer's main hard drive.
- It's like plugging in a flash drive and then saying:

“Okay, show its contents inside this folder.”

Table: Each Word Explained

Word	Meaning (Layman's Terms)
sudo	Run as administrator , needed for mounting disks
mount	The command that links a disk or partition to a folder so you can use it
/dev/sdc1	The partition on your USB/external disk that you formatted (with XFS earlier)
/disk	The mount point folder where the disk will show up (you created it with mkdir /disk)



```

Asiema@ProjectPro-P001: ~
xfs          MYUSB           5c8aab65-cc76-47d7-83fa-063933dff300  31.3G
sr0
Asiema@ProjectPro-P001:~$ sudo partprobe /dev/sdc1
[sudo] password for Asiema:
Asiema@ProjectPro-P001:~$ ls
Asiema@ProjectPro-P001:~$ sudo mkdir /disk
[sudo] password for Asiema:
Asiema@ProjectPro-P001:~$ sudo mkdir /disk
mkdir: cannot create directory '/disk': File exists
Asiema@ProjectPro-P001:~$ ls
Asiema@ProjectPro-P001:~$ sudo mount /dev/sdc1 /disk
Asiema@ProjectPro-P001:~$ lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda     8:0    0 128G  0 disk
sdb     8:16   0  30G  0 disk
└─sdb1  8:17   0   29G  0 part /
  └─sdb14 8:30   0   4M  0 part
  └─sdb15 8:31   0 106M  0 part /boot/efi
  └─sdb16 259:0  0 913M  0 part /boot
sdc     8:32   0   32G  0 disk
└─sdc1  8:33   0   32G  0 part /disk
                                         /mnt
sr0     11:0   1   628K  0 rom
Asiema@ProjectPro-P001:~$ 

```

Why It's Important

- Without mounting, the disk exists but is **invisible** - you can't access files on it.
 - Mounting tells Linux: "Display this drive's contents in this folder."
 - From then on, **anything you copy into /disk is actually saved on the USB drive.**
-

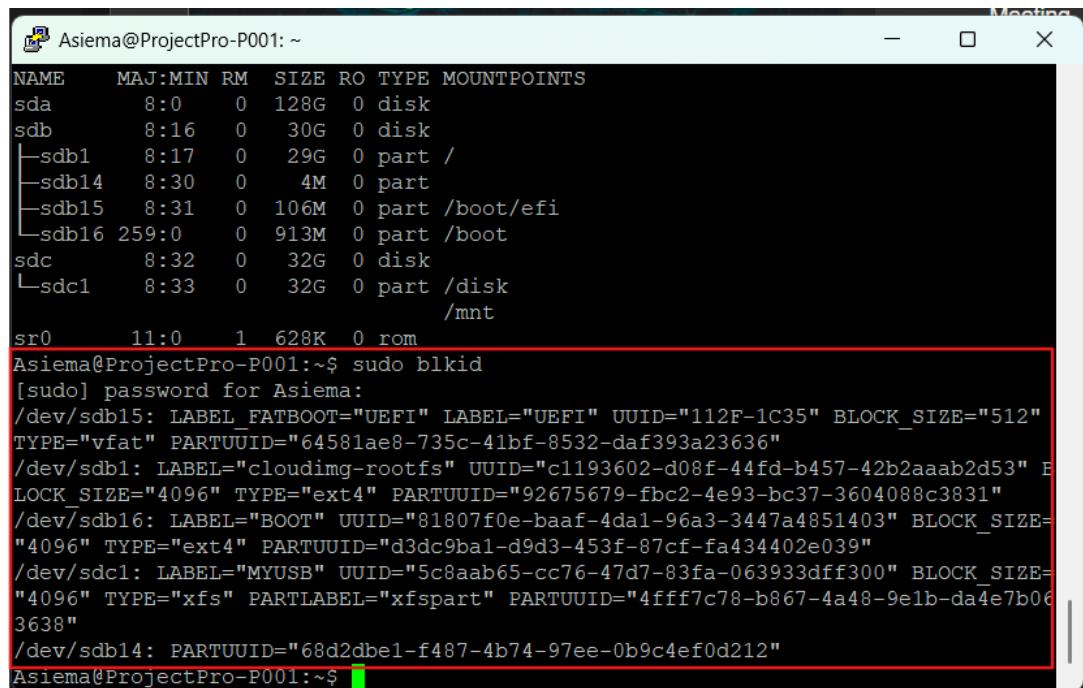
Analogy

It's like plugging in a USB stick and deciding **which drawer in your desk it should go in.**

Once you say, "this drawer is for the USB", all papers (files) you put in that drawer actually go to the USB.

12.2.7 Command 7: sudo blkid

sudo blkid



```
Asiema@ProjectPro-P001: ~
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda      8:0    0 128G  0 disk
sdb      8:16   0 30G   0 disk
└─sdb1   8:17   0 29G   0 part /
└─sdb14  8:30   0  4M   0 part
└─sdb15  8:31   0 106M  0 part /boot/efi
└─sdb16  259:0  0 913M  0 part /boot
sdc      8:32   0 32G   0 disk
└─sdc1   8:33   0 32G   0 part /disk
                           /mnt
sr0     11:0   1 628K  0 rom
Asiema@ProjectPro-P001:~$ sudo blkid
[sudo] password for Asiema:
/dev/sdb15: LABEL="UEFI" UUID="112F-1C35" BLOCK_SIZE="512"
TYPE="vfat" PARTUUID="64581ae8-735c-41bf-8532-daf393a23636"
/dev/sdb1: LABEL="cloudimg-rootfs" UUID="c1193602-d08f-44fd-b457-42b2aaab2d53" E
LOCK_SIZE="4096" TYPE="ext4" PARTUUID="92675679-fbc2-4e93-bc37-3604088c3831"
/dev/sdb16: LABEL="BOOT" UUID="81807f0e-baaf-4da1-96a3-3447a4851403" BLOCK_SIZE=
"4096" TYPE="ext4" PARTUUID="d3dc9ba1-d9d3-453f-87cf-fa434402e039"
/dev/sdc1: LABEL="MYUSB" UUID="5c8aab65-cc76-47d7-83fa-063933dff300" BLOCK_SIZE=
"4096" TYPE="xfs" PARTLABEL="xfspart" PARTUUID="4fff7c78-b867-4a48-9e1b-da4e7b06
3638"
/dev/sdb14: PARTUUID="68d2dbe1-f487-4b74-97ee-0b9c4ef0d212"
Asiema@ProjectPro-P001:~$
```

What It Does (Layman's Terms):

This command **lists all the storage devices** on your system and shows details about them:

- **UUID** → a unique permanent ID for each partition
- **TYPE** → the filesystem (ext4, xfs, swap, etc.)
- **LABEL** → the name you gave it (e.g., MYUSB)

This is important because **device names like /dev/sdc1 can change** if you plug in more drives. But the **UUID never changes**, so Linux can always recognize the same drive.

Table: Each Word Explained

Word	Meaning (Layman's Terms)
sudo	Run with administrator privileges (needed to read system-level device info)
blkid	“ Block ID ” → shows all storage devices with their UUIDs, labels, and filesystem types

Why It's Important

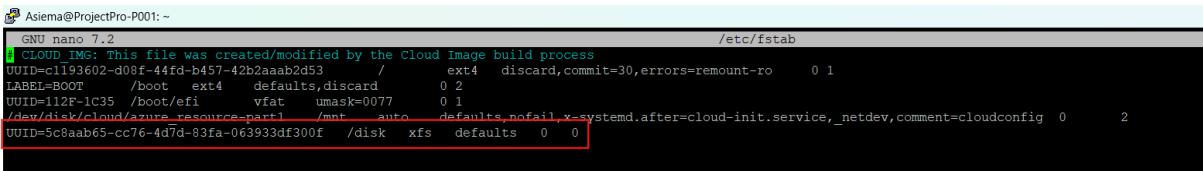
- If you want the disk to **auto-mount every time the system boots**, you'll need its **UUID**.
- You use this UUID in the /etc/fstab file instead of /dev/sdc1, because /dev/sdc1 might change to /dev/sdd1 later, but UUID will always remain the same.

Analogy

Imagine you park your car in different parking spots each day (`/dev/sdc1`, `/dev/sdd1`, etc.). The **UUID** is like your car's license plate - it always identifies the same car, no matter which spot it's in.

12.2.8 Command 8: sudo nano /etc/fstab

sudo nano /etc/fstab



```
Asiema@ProjectPro-P001: ~
GNU nano 7.2
/etc/fstab
CLOUD_IMG: This file was created/modified by the Cloud Image build process
UUID=c1193602-d08f-44fd-b457-42b2aab2d53    /      ext4  discard,commit=30,errors=remount-ro  0  1
LABEL=BOOT  /boot  ext4  defaults,discard  0  2
UUID=112F-1c35  /boot/efi  vfat  umask=0077  0  1
/dev/disk/cloud/azure_resource-part1  /mnt  auto  defaults,nofail,_v-sysmd.after=cloud-init.service,_netdev,comment=cloudconfig  0      2
UUID=5c8aab65-cc76-4d7d-83fa-063933df300f  /disk  xfs  defaults  0  0
```

What It Does (Layman's Terms):

- This opens the system configuration file `/etc/fstab` in the **nano text editor**. The file **lists all drives and tells Linux where to mount them at startup**.
- By adding your disk's **UUID** here, you're telling the system:

"Every time I restart, automatically mount this drive into /disk."

Table: Each Word Explained

Word	Meaning (Layman's Terms)
sudo	Run as administrator (editing system files requires root permissions)
nano	A simple terminal-based text editor
/etc/fstab	The file that controls which disks are mounted where during system boot

Analogy

Think of /etc/fstab like a **to-do list for Linux when it starts**. Each line says: “Mount this drive here, using this format, with these rules.” If you don’t add your USB here, you’ll have to **mount it manually every time** with sudo mount.

Breaking Down the Entry

Field	Example	Meaning
UUID=37755c54-...	The permanent ID of your partition (from blkid)	Always points to your specific drive
/disk	The mount point folder you created	Where the disk will show up
xfs	Filesystem type (XFS in your case, could be ext4, etc.)	Ensures correct driver is used
defaults	Default options (read/write, auto-mount, etc.)	Good for most cases
0	Dump (backup) option, usually 0	Tells system not to auto-backup
0	FSCK check option, usually 0 for secondary drives	Skips filesystem check on boot

12.2.9 Command 9: sudo -i chmod 777 /disk Exit

sudo – i chmod 777 /disk Exit

```
[sudo] password for Asiemka:  
chmod: cannot access 'Exit': No such file or directory  
Asiemka@ProjectPro-P001:~$ sudo nano /etc/fstab  
Asiemka@ProjectPro-P001:~$ sudo -i chmod 777 /disk Exit  
chmod: cannot access 'Exit': No such file or directory  
Asiemka@ProjectPro-P001:~$ sudo -i  
root@ProjectPro-P001:~# chmod 777 /disk  
root@ProjectPro-P001:~# Exit  
Exit: command not found  
root@ProjectPro-P001:~# exit  
logout  
Asiemka@ProjectPro-P001:~$
```

What Each Does (Layman's Terms)

sudo -i

- Opens a root shell (acts like logging in as the super administrator).
- You need this because changing permissions in system folders requires root privileges.

chmod 777 /disk

- Changes permissions on the /disk folder.
- 777 means:
 - **7 = Read + Write + Execute**
 - Given to **everyone** (owner, group, others).
- In simple terms → anyone can create, delete, and run files inside /disk.
- This is useful for shared environments (like Hadoop/Hive clusters, or multi-user systems).

exit

- Leaves the root shell and returns you to your normal user shell.

Table: Each Word Explained

Command	Meaning (Layman's Terms)
sudo -i	Switch into root mode so you can run commands without repeating sudo every time
chmod	Command to change permissions of files or folders
777	Gives full control (read, write, execute) to everyone
/disk	The mount point folder where your USB/external drive is mounted
exit	Go back to your normal user session from root mode

Security Note

Using chmod 777 is **very open** (anyone can do anything).

Safer alternatives could be:

- chmod 755 /disk → everyone can read, only owner can write
 - chown <your-username>:<your-group> /disk → give full control to you only
-

Why It's Useful

- In big data environments like **Hadoop** or **Hive**, multiple services need to read/write freely.
- This avoids permission errors when different users or apps try to access the mounted disk.

PART 4: INSTALLATION

13 Hadoop & Hive Installation Context

Key Idea

- **Hive** looks like SQL (you write HiveQL queries).
- But Hive cannot run on its own - it **depends on Hadoop**.

Why?

- Hive queries get **translated into MapReduce (or YARN) jobs**.
- Hadoop provides:
 - **HDFS (Hadoop Distributed File System)** → where Hive stores its tables/data
 - **MapReduce/YARN engine** → executes the queries as distributed jobs

So, without Hadoop installed and running, Hive is useless.

1. **What You Must Do Before Running Hive**
2. **Install Hadoop** (this is the base layer).
3. **Configure Hadoop properly** so Hive can connect.

Configuration files that matter:

- **.bashrc** → sets environment variables so Hadoop & Hive commands work in your terminal.
- **hadoop-env.sh** → configures Java environment for Hadoop.
- **core-site.xml** → defines core Hadoop settings like default filesystem (hdfs://).
- **hdfs-site.xml** → configures storage replication, block size, etc.

- **mapred-site.xml** → configures MapReduce jobs (e.g., run in YARN).
 - **yarn-site.xml** → configures YARN resource manager and node managers.
-

Summary in Layman's Terms

Think of it like building a **house**:

- Hadoop = the **foundation and walls** (data storage + job engine).
- Hive = the **furniture** (lets you sit down and write SQL instead of handling raw Hadoop commands).

Without the foundation (Hadoop), Hive has nowhere to sit.

Why This Connects to Your Disk Setup

Earlier, you prepared and mounted a disk at /disk. This is perfect for **HDFS storage**.

You can point Hadoop's hdfs-site.xml to use /disk/hdfs as its storage directory.

That way, Hive queries will store data inside the disk you just prepared.



13.1 Step 1: Update & Install Java (Required for Hadoop + Hive)

Commands:

```
sudo apt update
```

```
sudo apt install openjdk-8-jdk -y
```

```
java -version; javac -version
```

What These Do (Layman's Terms)

sudo apt update

- Refreshes your package list (like refreshing the “app store” on your Linux system).
- Ensures you’ll get the latest available versions of software.

13.1.1 **sudo apt install openjdk-8-jdk -y**

- Installs **OpenJDK 8**, which is the Java Development Kit.
- Hadoop (and Hive) are written in Java, so they **cannot run without Java**.
- The **-y** flag just auto-confirms "yes" to installation prompts.

13.1.2 **java -version; javac -version**

- Verifies that both **Java runtime (java)** and **Java compiler (javac)** are installed and working.
- You should see something like:

Why Java First?

- Hadoop's entire ecosystem (HDFS, YARN, MapReduce, Hive, Spark, etc.) is built on **Java**. Without Java, Hadoop daemons (NameNode, DataNode, etc.) simply won't start.
 - That's why installing **OpenJDK 8** is always step 1 in any Hadoop/Hive setup.
-

13.2 How It Fits into the Bigger Workflow

1. **Java** → Required Runtime for Hadoop and Hive
2. **Hadoop** → Core engine: HDFS (storage) + YARN/MapReduce (execution)
3. **Hive** → SQL-like layer on top of Hadoop

14 Step 2: Install OpenSSH

Command:

```
sudo apt install openssh-server openssh-client -y
```

What This Does (Layman's Terms)

14.1 openssh-server

- Let's other machines (or even your own) connect into this system securely via SSH. Think of it as setting up a **doorway for remote control**.

14.2 openssh-client

- This system allows us to connect **out** to other systems using SSH. Like having the **keys** to enter someone else's system.

14.3 -y

- Auto-confirms the installation without asking "Are you sure?"
-

Why Hadoop Needs This

- Hadoop is designed to run on a **cluster of nodes** (multiple machines).
 - Even in **pseudo-distributed mode** (all daemons running on one machine), Hadoop services (like **NameNode** and **DataNode**) still talk to each other **over SSH**. Without SSH, Hadoop components can't coordinate.
-

Analogy

Imagine Hadoop is a company with multiple departments:

- **NameNode** = Headquarters
- **DataNodes** = Branch offices

They need a **secure phone line (SSH)** to talk to each other - even if all the “offices” are in the same building (your single machine).

14.4 Where This Fits in the Workflow

- **Step 1 (Java):** Install Java (runtime for Hadoop/Hive)
- **Step 2 (OpenSSH):** Enable secure communication between Hadoop services
- **Step 3 (Next):** Download and configure Hadoop

```
Processing triggers for libc-bin (2.39-0ubuntu8.5) ...
Processing triggers for libgdk-pixbuf-2.0-0:amd64 (2.42.10+dfsg-3ubuntu3.2) ...
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (gemu) binaries on this host.
Asiema@ProjectPro-P001:~$ sudo apt install openssh-server openssh-client -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
openssh-server is already the newest version (1:9.6p1-3ubuntu13.13).
openssh-client is already the newest version (1:9.6p1-3ubuntu13.13).
openssh-client set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 6 not upgraded.
Asiema@ProjectPro-P001:~$
```

15 Step 3: Generate SSH Key & Authorize It

Commands:

```
ssh-keygen -t rsa -P "" -f ~/.ssh/id_rsa  
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys  
chmod 0600 ~/.ssh/authorized_keys
```

What These Do (Layman's Terms)

15.1 Generate a key pair

- ```
ssh-keygen -t rsa -P "" -f ~/.ssh/id_rsa
```
- -t rsa → creates an RSA encryption key
  - -P "" → no password (empty passphrase)
  - -f ~/.ssh/id\_rsa → saves the key as id\_rsa inside the .ssh folder

This gives you:

- A **private key** (id\_rsa)
- A **public key** (id\_rsa.pub)

#### 15.2 Add the public key to authorized keys

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

- Appends your own public key into the list of keys allowed to log in
- It means you can now SSH into yourself **without a password**

### 15.3 Lock down permissions

```
chmod 0600 ~/.ssh/authorized_keys
```

- Sets the file so **only you can read/write it**
  - SSH won't work if this file is world-readable
- 

### Why Hadoop Needs This

- Hadoop starts multiple daemons (**NameNode**, **DataNode**, **ResourceManager**, **NodeManager**)
  - These daemons **SSH into localhost** to coordinate with each other
  - If passwords were required, Hadoop would hang waiting for manual input
  - Passwordless SSH ensures everything runs automatically
- 

### Analogy

Think of Hadoop as a manager that needs to move between different offices in the same building. Instead of asking for an **ID card every time** (password), you give him a **master pass** (SSH key) so he can move freely.

---

## 15.4 Where We Are in Workflow

- **Step 1** → Install Java
- **Step 2** → Install OpenSSH
- **Step 3** → Set up Passwordless SSH
- **Step 4 (Next)** → Download & extract Hadoop, then set up environment variables

## 16 Step 4: Test SSH Connection

**Command:**

```
ssh localhost
```

### What This Does (Layman's Terms)

- Tries to **SSH into your own machine** using localhost (which always points to your computer itself). If Passwordless SSH was set up correctly (Step 3), this should log you in **without asking for a password**.
  - You'll see a system info banner (like in your screenshot) → confirming SSH is working.
-

```

Asiema@ProjectPro-P001:~$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
Asiema@ProjectPro-P001:~$ chmod 0600 ~/.ssh/authorized_keys
Asiema@ProjectPro-P001:~$ ssh localhost
The authenticity of host 'localhost (127.0.0.1)' can't be established.
ED25519 key fingerprint is SHA256:mZjdBwvTrKeHwKKQ1DEQLiveBwtyKOTyOpvLrFm0pgEg
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'localhost' (ED25519) to the list of known hosts.
Welcome to Ubuntu 24.04.3 LTS (GNU/Linux 6.11.0-1018-azure) -

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro

System information as of Sat Sep 6 22:15:39 UTC 2025

System load: 0.0 Processes: 160
Usage of /: 8.3% of 28.02GB Users logged in: 1
Memory usage: 2% IPv4 address for eth0: 10.1.0.4
Swap usage: 0%

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s just raised the bar for easy, resilient and secure K8s cluster deployment.

 https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

6 updates can be applied immediately.
4 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Sat Sep 6 from 41.246.130.117
Asiema@ProjectPro-P001:~$

```

## 16.1 Why This Is Important

- Hadoop daemons (**NameNode**, **DataNode**, **ResourceManager**, etc.) constantly SSH into **localhost** to manage themselves. If this doesn't work, Hadoop startup will fail because it will get stuck waiting for a password.
- By testing here, you confirm that Hadoop can run smoothly.

## Analogy

Think of it like testing your **master pass key** before giving it to a building manager (Hadoop). If the key works on the front door (SSH localhost), it will work on all the internal doors when Hadoop runs.

---

## 16.2 Where We Are in Workflow

- **Step 1** → Install Java
- **Step 2** → Install OpenSSH
- **Step 3** → Generate Passwordless SSH keys
- **Step 4** → Test SSH connection (  must succeed)
- **Step 5 (Next)** → Download Hadoop and set up environment variables (.bashrc)

## 17 Step 5: Gain Root Access

**Command:**

```
sudo -i
```

### What This Does (Layman's Terms)

- Switches you into a **root shell** (super administrator mode).
- Once you're root, every command you type runs with **full system privileges**.
- Needed for tasks like:
  - Modifying system directories (e.g., /disk, /usr/local/hadoop)
  - Editing global config files (/etc/hosts, /etc/fstab, etc.)
  - Installing software across the system

---

### 17.1 Why Hadoop Setup Needs This

- Hadoop installation often involves:
  - Copying files to /usr/local
  - Changing permissions for Hadoop folders
  - Editing system-wide files

All of these require **root permissions**.

By running sudo -i, you don't need to prefix every single command with sudo.

---

## 17.2 Analogy

Think of it like switching from a **regular office employee** (normal user) to the **building administrator** (root). Now you can access every locked room without needing to ask for permission each time.

---

## 17.3 Where We Are in Workflow

- **Step 1** → Installed Java
- **Step 2** → Installed OpenSSH
- **Step 3** → Set up Passwordless SSH
- **Step 4** → Tested SSH connection
- **Step 5** → Gained root access (so you can install Hadoop system-wide)
- **Step 6 (Next)** → Download & Extract Hadoop into /usr/local/hadoop

## 18 Step 6: Allow Permissions to /disk

**Command:**

```
chmod 777 /disk
```

---

```
Last login: Sat Sep 6 10:43:35 2025 from 41.246.130.117
Asiema@ProjectPro-P001:~$ sudo -i
[sudo] password for Asiema:
root@ProjectPro-P001:~# chmod 777 /disk
root@ProjectPro-P001:~# cd / disk
ls
```

### What This Does (Layman's Terms)

- chmod changes permissions for files/folders.
- 777 means:
  - **Read + Write + Execute** access for:
    - Owner
    - Group
    - Everyone else

So, anyone (including Hadoop processes) can freely **create, delete, and run files** inside /disk.

---

### 18.1 Why Hadoop Needs This

- Hadoop will use /disk for:
  - **Binaries** (executables it needs to run)
  - **HDFS storage directories** (where your data blocks are stored)

If the permissions are too strict, Hadoop daemons (NameNode, DataNode, etc.) may fail to write into this directory. By giving 777, you make sure Hadoop won't run into permission errors.

 **Note:** This is fine for **development/test** environments (like your Azure VM).

For production, you'd assign Hadoop to a dedicated user and give ownership instead of 777.

---

## 18.2 Analogy

- Think of /disk as a **shared warehouse**. By setting chmod 777, you're saying:  
*"Everyone has a key - anyone can walk in, store boxes, move them around, or take them out."*
  - That way, Hadoop doesn't get blocked at the door.
- 

## 18.3 Where We Are in Workflow

- **Step 1** → Installed Java
- **Step 2** → Installed OpenSSH
- **Step 3** → Set up Passwordless SSH
- **Step 4** → Tested SSH connection
- **Step 5** → Gained root access
- **Step 6** → Allowed permissions to /disk 

- **Step 7 (Next)** → Download and extract Hadoop into /usr/local/hadoop, and configure environment variables

## 19 Step 7: Create Hadoop Installation Directory

### Commands:

```
cd /disk
```

```
mkdir hadoop
```

```
ls
```

---

### What These Do (Layman's Terms)

#### 19.1 cd /disk

- Moves you into the /disk folder (the mount point we prepared earlier).

#### 19.2 mkdir hadoop

- Creates a new folder named hadoop inside /disk. This folder will be used to **download and store Hadoop installation files** (like the tarball we extract).

#### 19.3 ls

- Lists the contents of the current directory. Confirms that the new hadoop folder was successfully created.
-

## Why This Step Matters

- Hadoop needs a **dedicated workspace** for binaries and data.
  - By creating /disk/hadoop, you ensure:
    - Downloads don't clutter other folders
    - You know exactly where Hadoop was installed
    - HDFS storage paths can later be pointed here
- 

## Analogy

Think of /disk/hadoop as **building a storage room** in your warehouse (/disk). This is where you'll put all the **Hadoop tools and storage racks** so they're organized in one place.

---

## 19.4 Where We Are in Workflow

- **Step 1** → Installed Java
- **Step 2** → Installed OpenSSH
- **Step 3** → Generated passwordless SSH keys
- **Step 4** → Tested SSH
- **Step 5** → Gained root access
- **Step 6** → Allowed permissions to /disk
- **Step 7** → Created Hadoop directory 

- **Step 8 (Next)** → Download Hadoop tarball into /disk/hadoop and extract it

## 20 Step 8: Download Hadoop Package

### Command:

```
wget -P /disk/hadoop
https://dlcdn.apache.org/hadoop/common/hadoop-3.4.1/hadoop-
3.4.1.tar.gz
```

---

```
Resolving dlcdn.apache.org (dlcdn.apache.org) ... 151.101.2.132, 2a04:4e42::644
Connecting to dlcdn.apache.org (dlcdn.apache.org)|151.101.2.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 974002355 (929M) [application/x-gzip]
/disk/hadoop/hadoop-3.4.1.tar.gz: Permission denied

Cannot write to '/disk/hadoop/hadoop-3.4.1.tar.gz' (Success).
Asiema@ProjectPro-P001:/disk$ sudo wget https://dlcdn.apache.org/hadoop/common/hadoop-3.4.1/hadoop-3.4.1.tar.gz -P /disk/hadoop
[sudo] password for Asiema:
--2025-09-07 10:24:19-- https://dlcdn.apache.org/hadoop/common/hadoop-3.4.1/hadoop-3.4.1.tar.gz
Resolving dlcdn.apache.org (dlcdn.apache.org) ... 151.101.2.132, 2a04:4e42::644
Connecting to dlcdn.apache.org (dlcdn.apache.org)|151.101.2.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 974002355 (929M) [application/x-gzip]
Saving to: '/disk/hadoop/hadoop-3.4.1.tar.gz'

hadoop-3.4.1.tar.gz 100%[=====] 928.88M 25.4MB/s in 38s

2025-09-07 10:24:57 (24.8 MB/s) - '/disk/hadoop/hadoop-3.4.1.tar.gz' saved [974002355/974002355]
Asiema@ProjectPro-P001:/disk$
```

### 20.1 What This Does (Layman's Terms)

- **wget** → downloads a file from the internet.
- **-P /disk/hadoop** → saves the file into /disk/hadoop.
- **URL** → points to the official Apache mirror hosting Hadoop **v3.4.1**.
- The downloaded file will be:
  - /disk/hadoop/hadoop-3.4.1.tar.gz

---

## Why This Step Matters

- This pulls the **official Hadoop release tarball** from Apache.
- You'll extract this archive to /usr/local so the system recognizes Hadoop as a software package.
- Keeping the tarball in /disk/hadoop ensures all Hadoop-related files are organized in one place.

## 21 Step 9: Extract Hadoop

**The guide's commands:**

```
cd /disk/hadoop
```

```
tar xzf hadoop-3.4.1.tar.gz
```

---

```
tar: Exiting with failure status due to previous errors
Asiema@ProjectPro-P001:/disk/hadoop$ ls -lh /disk/hadoop/hadoop-3.4.1.tar.gz
-rw-r--r-- 1 root root 929M Oct 10 2024 /disk/hadoop/hadoop-3.4.1.tar.gz
Asiema@ProjectPro-P001:/disk/hadoop$ sudo tar xzf hadoop-3.4.1.tar.gz
[sudo] password for Asiema:
Asiema@ProjectPro-P001:/disk/hadoop$ sudo tar xzf hadoop-3.4.1.tar.gz
Asiema@ProjectPro-P001:/disk/hadoop$ ls
hadoop-3.4.1 hadoop-3.4.1.tar.gz
Asiema@ProjectPro-P001:/disk/hadoop$ ls hadoop-3.4.1
LICENSE-binary NOTICE-binary README.txt etc lib licenses-binary share
LICENSE.txt NOTICE.txt bin include libexec sbin
Asiema@ProjectPro-P001:/disk/hadoop$ █
```

### What This Does (Layman's Terms)

#### 21.1 cd /disk/hadoop

- where you downloaded the Hadoop tarball.

#### 21.2 tar xzf hadoop-3.4.1.tar.gz

- tar → the Linux archiving tool
- x → extract
- z → decompress gzip (.gz) files
- f → use file

 This unpacks the Hadoop archive into a folder named:

/disk/hadoop/hadoop-3.4.1

Inside it, you'll find directories like bin, etc, lib, sbin, share.

---

## 21.3 Why This Step Matters

- This makes Hadoop's **binaries and configuration files** accessible. Once extracted, you can point environment variables to this folder so you can run hadoop, hdfs, yarn commands globally.

## 21.4 Analogy

Think of the tarball (hadoop-3.4.1.tar.gz) as a **sealed box** of tools.

- Extracting it unpacks the tools into a workshop (/usr/local/hadoop).
- The **symlink** (/usr/local/hadoop → /usr/local/hadoop-3.4.1) is like putting a permanent sign on the workshop door so you don't need to remember the exact version number.

## 22 Step 10: Add Hadoop + Hive Environment Variables

Open .bashrc:

```
cd ~
nano ~/.bashrc
```

---

Add this block at the end of the file:

```
1. # Java
2. export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
3.
4. # Hadoop
5. export HADOOP_HOME=/usr/local/hadoop
6. export HADOOP_INSTALL=$HADOOP_HOME
7. export HADOOP_MAPRED_HOME=$HADOOP_HOME
8. export HADOOP_COMMON_HOME=$HADOOP_HOME
9. export HADOOP_HDFS_HOME=$HADOOP_HOME
10. export YARN_HOME=$HADOOP_HOME
11. export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
12. export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
13. export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
14.
15. # Hive (you'll set this once Hive is installed)
16. # export HIVE_HOME=/usr/local/hive
17. # export PATH=$PATH:$HIVE_HOME/bin
18.
```

---

## 22.1 Save + Exit Nano

- Press CTRL + O, then Enter to save.
  - Press CTRL + X to exit.
- 

## 22.2 Apply Changes Immediately

1. source ~/ .bashrc
  - 2.
- 

## 22.3 Test Hadoop

1. hadoop version
  - 2.
- 

You should see something like:

1. Hadoop 3.4.1
  2. Source code repository ...
  3. Compiled by ...
  - 4.
- 

## 22.4 Why This Step Matters

- Without this, Hadoop commands would only work if you navigated into the bin folder. With these variables, commands like hdfs namenode -format or start-dfs.sh will work globally.
- When you add Hive later, the same pattern applies (HIVE\_HOME + PATH update).

## 23 Step 11: Validate Java Setup

### Commands:

```
1. which javac
2. readlink -f /usr/bin/javac
3.
```

---

### 23.1 which javac

- Shows the path of the Java compiler (javac) binary.
- Usually points to /usr/bin/javac.

### 23.2 readlink -f /usr/bin/javac

- Follows the symbolic links to the **real Java compiler path**.
  - On your system, this should return something like:
  - /usr/lib/jvm/java-8-openjdk-amd64/bin/javac
- 

### 23.3 Why Hadoop Needs This

- Hadoop requires **both the runtime (java) and compiler (javac)**. JAVA\_HOME must point to the **parent directory** containing both bin/java and bin/javac.
- That's why we set:

```
1. export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
2.
```

Because inside this directory:

bin/java ✓

bin/javac ✓

---

## 23.4 Where We Are in Workflow

- Java runtime validated ✓
- Hadoop recognizes version ✓
- Now we've confirmed the compiler path ✓

## 24 Step 12: Set JAVA\_HOME in Hadoop Env

### Command:

```
1. sudo nano $HADOOP_HOME/etc/hadoop/hadoop-env.sh
2.
```

### What to Do in the File

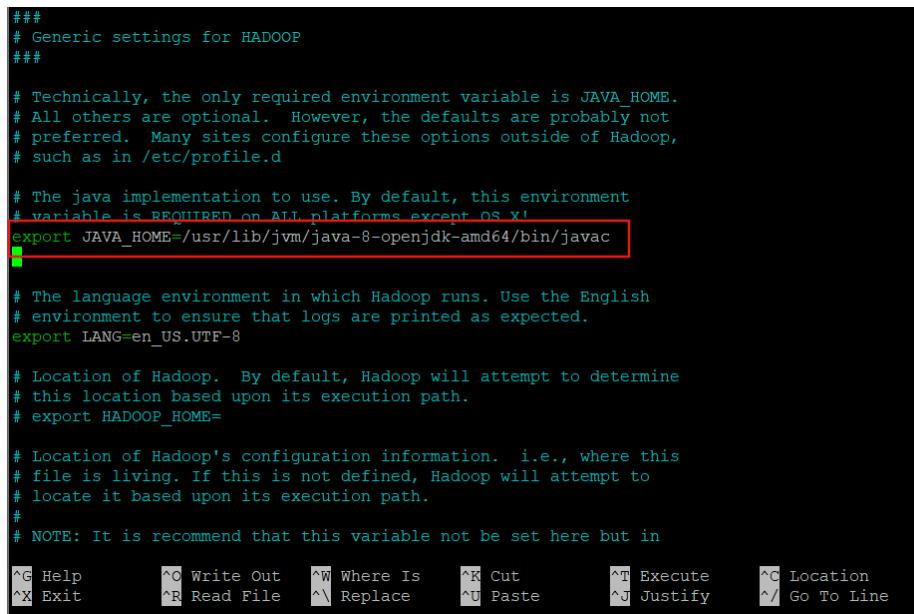
- Scroll down until you find this line:

```
1. # export JAVA_HOME=
2.
```

(It may be commented out with a #.)

- Replace it with:

```
1. export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
2.
```



```

Generic settings for HADOOP

Technically, the only required environment variable is JAVA_HOME.
All others are optional. However, the defaults are probably not
preferred. Many sites configure these options outside of Hadoop,
such as in /etc/profile.d

The java implementation to use. By default, this environment
variable is REQUIRED on ALL platforms except OS X!
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/bin/javac

The language environment in which Hadoop runs. Use the English
environment to ensure that logs are printed as expected.
export LANG=en_US.UTF-8

Location of Hadoop. By default, Hadoop will attempt to determine
this location based upon its execution path.
export HADOOP_HOME=

Location of Hadoop's configuration information. i.e., where this
file is living. If this is not defined, Hadoop will attempt to
locate it based upon its execution path.

NOTE: It is recommend that this variable not be set here but in
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^Y Replace ^U Paste ^J Justify ^/ Go To Line
```

### 24.1 Why This Step Matters

- The .bashrc export works for your **user shell**, but Hadoop daemons (NameNode, DataNode, etc.) also rely on **their own env file**. By setting JAVA\_HOME in hadoop-

env.sh, you ensure all Hadoop processes (started via start-dfs.sh and start-yarn.sh) know where Java is.

---

## 24.2 Double Check

- After editing, reload and verify:

```
1. echo $JAVA_HOME
2. hadoop version
3.
```

- You should consistently see Hadoop **3.4.1** without the JAVA\_HOME not set error anymore. 
- 

## Analogy

Think of .bashrc as setting up your **personal shortcuts**, while hadoop-env.sh is like **teaching the entire Hadoop system** where Java lives so all daemons can run properly.

## Error occurred

```
Asiema@ProjectPro-P001:~$ echo $JAVA_HOME
/usr/lib/jvm/java-8-openjdk-amd64
Asiema@ProjectPro-P001:~$ hadoop version
ERROR: JAVA_HOME /usr/lib/jvm/java-8-openjdk-amd64/bin/javac does not exist.
Asiema@ProjectPro-P001:~$ █
```

- The error says Hadoop can't find **bin/javac** under your JAVA\_HOME.

**Right now:**

```
1. echo $JAVA_HOME
2. /usr/lib/jvm/java-8-openjdk-amd64
3.
```

but Hadoop complains:

```
1. /usr/lib/jvm/java-8-openjdk-amd64/bin/javac does not exist
2.
```

- That means you are likely to have only the **JRE** (runtime) or a partial install.

Hadoop's scripts expect the **JDK** (which includes javac).

## 24.3 Fix it fast

### Install a full JDK

- Try Java 8 JDK first; if it's not available on your repo, use Java 11 (Hadoop 3.4.x works with 8 or 11).

```
1. sudo apt update
2. sudo apt install -y openjdk-8-jdk || sudo apt install -y openjdk-11-jdk
3.
```

### Confirm javac exists and note its path

```
1. which javac
2. readlink -f "$(which javac)"
3.
```

```
Asiema@ProjectPro-P001:~$ sudo sed -i 's|^#\? *export JAVA_HOME *=.*|export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64|' \
/disk/hadoop/hadoop-3.4.1/etc/hadoop/hadoop-env.sh
Asiema@ProjectPro-P001:~$ echo 'export HADOOP_HOME=/disk/hadoop/hadoop-3.4.1' >> ~/.bashrc
Asiema@ProjectPro-P001:~$ echo 'export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH' >> ~/.bashrc
Asiema@ProjectPro-P001:~$ echo 'export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64' >> ~/.bashrc
Asiema@ProjectPro-P001:~$ source ~/.bashrc
Asiema@ProjectPro-P001:~$ echo $HADOOP_HOME
/disk/hadoop/hadoop-3.4.1
Asiema@ProjectPro-P001:~$ echo $JAVA_HOME
/usr/lib/jvm/java-8-openjdk-amd64
Asiema@ProjectPro-P001:~$ hadoop version
Hadoop 3.4.1
Source code repository https://github.com/apache/hadoop.git -r 4d7825309348956336b8f06a08322b78422849b1
Compiled by mthakur on 2024-10-09T14:57Z
Compiled on platform linux-x86_64
Compiled with protoc 3.23.4
From source with checksum 7292fe9dba5e2e44e3a9f763fce3e680
This command was run using /disk/hadoop/hadoop-3.4.1/share/hadoop/common/hadoop-common-3.4.1.jar
Asiema@ProjectPro-P001:~$
```

You should see something like:

- `/usr/lib/jvm/java-8-openjdk-amd64/bin/javac` or
- `/usr/lib/jvm/java-11-openjdk-amd64/bin/javac`

## 25 Step 13: Configure core-site.xml

**Open the file:**

1. nano \$HADOOP\_HOME/etc/hadoop/core-site.xml
  - 2.
- 

**Paste this configuration:**

1. <configuration>
  - 2.
  3. <property>
  4. <name>hadoop.tmp.dir</name>
  5. <value>/disk/hadoop/tmpdata</value>
  6. </property>
  - 7.
  8. <property>
  9. <name>fs.defaultFS</name>
  10. <value>hdfs://127.0.0.1:9000</value>
  11. </property>
  - 12.
  13. </configuration>
  - 14.
-

## 25.1 What These Settings Mean

### 25.1.1 hadoop.tmp.dir

- A scratch directory Hadoop uses for temp files, logs, etc.
- Here it points to /disk/hadoop/tmpdata.
- Make sure that directory exists:
- `mkdir -p /disk/hadoop/tmpdata`

### 25.1.2 fs.defaultFS

- Defines the default filesystem URI.
  - `hdfs://127.0.0.1:9000` means your NameNode will run locally and listen on port **9000**.
  - All HDFS commands (like `hdfs dfs -ls /`) will go here by default.
- 

## 25.2 Why This Step Matters

- Without `fs.defaultFS`, Hadoop doesn't know where HDFS lives.
  - Without `hadoop.tmp.dir`, Hadoop will complain about missing temp directories.
-

## 25.3 Next Step

After this, you'll configure:

- **hdfs-site.xml** → sets replication and storage directories.
- **mapred-site.xml** → tells MapReduce to run on YARN.
- **yarn-site.xml** → configures YARN for resource management.

## 26 Step 14: Configure hdfs-site.xml

**Open the file:**

1. nano \$HADOOP\_HOME/etc/hadoop/hdfs-site.xml
- 2.

**Paste this config:**

```
1. <configuration>
2. <property>
3. <name>dfs.name.dir</name>
4. <value>/disk/hadoop/dfsdata/namenode</value>
5. </property>
6. <property>
7. <name>dfs.data.dir</name>
8. <value>/disk/ hadoop /dfsdata/datanode</value>
9. </property>
10. <property>
11. <name>dfs.replication</name>
12. <value>1</value>
13. </property>
14. </configuration>
15.
```

## 26.1 What These Do

### dfs.name.dir

Points to where the **NameNode** stores its metadata (like the “index” of all HDFS files).

### dfs.data.dir

Points to where the **DataNode** stores actual file blocks.

### dfs.replication

Defines how many copies of each block are stored.

→ For single-node clusters, this must be set to **1** (otherwise Hadoop will complain it can't replicate).

---

## 26.2 Verify Directories Exist

Before starting Hadoop, make sure the directories are in place:

1. `ls -ld /disk/ hadoop /dfsdata/namenode`
2. `ls -ld /disk/ hadoop /dfsdata/datanode`
- 3.

If not, create them again:

1. `mkdir -p /disk/ hadoop /dfsdata/namenode`
2. `mkdir -p /disk/ hadoop /dfsdata/datanode`
- 3.

```
2546 directories, 20240 files
Asiema@ProjectPro-P001:~$ sudo nano $HADOOP_HOME/etc/hadoop/core-site.xml
Asiema@ProjectPro-P001:~$ sudo nano $HADOOP_HOME/etc/hadoop/hdfs-site.xml
Asiema@ProjectPro-P001:~$ ls -ld /disk/hadoop/tmpdata
drwxr-xr-x 2 root root 6 Sep 7 14:10 /disk/hadoop/tmpdata
Asiema@ProjectPro-P001:~$ ls -ld /disk/hadoop/dfsdata/namenode
drwxr-xr-x 2 root root 6 Sep 7 14:11 /disk/hadoop/dfsdata/namenode
Asiema@ProjectPro-P001:~$ ls -ld /disk/hadoop/dfsdata/datanode
drwxr-xr-x 2 root root 6 Sep 7 14:11 /disk/hadoop/dfsdata/datanode
Asiema@ProjectPro-P001:~$ █
```

## 27 Step 15: Configure mapred-site.xml

### Open the template file

By default, Hadoop ships with mapred-site.xml.template. First copy it to the correct name:

```
1. cp $HADOOP_HOME/etc/hadoop/mapred-site.xml.template
$HADOOP_HOME/etc/hadoop/mapred-site.xml
2.
```

Edit the file

```
1. nano $HADOOP_HOME/etc/hadoop/mapred-site.xml
2.
```

Paste this configuration

```
1. <configuration>
2. <property>
3. <name>mapreduce.framework.name</name>
4. <value>yarn</value>
5. </property>
6. </configuration>
7.
```

```
Asiema@ProjectPro-P001: ~
GNU nano 7.2 /disk/hadoop/hadoop-3.4.1/etc/hadoop/mapred-site.xml
?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<!--
Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
<property>
 <name>mapreduce.framework.name</name>
 <value>yarn</value>
</property>
</configuration>
```

## 27.1 What This Does

### 27.1.1 mapreduce.framework.name = yarn

- Ensures MapReduce jobs are submitted to the **YARN ResourceManager**.
- Without this, jobs would try to run in “local mode” (not distributed).

---

## 27.2 Next Step

- Once this is set, the next file to configure is:

**Step 16: yarn-site.xml** → where you specify how YARN manages NodeManager and ResourceManager.

## 28 Step 16: Configure YARN

### Command:

1. sudo nano \$HADOOP\_HOME/etc/hadoop/yarn-site.xml
- 2.

### Paste this inside:

```
1. <configuration>
2. <property>
3. <name>yarn.nodemanager.aux-services</name>
4. <value>mapreduce_shuffle</value>
5. </property>
6. <property>
7. <name>yarn.nodemanager.aux-
services.mapreduce.shuffle.class</name>
8. <value>org.apache.hadoop.mapred.ShuffleHandler</value>
9. </property>
10. <property>
11. <name>yarn.resourcemanager.hostname</name>
12. <value>127.0.0.1</value>
13. </property>
14. <property>
15. <name>yarn.acl.enable</name>
16. <value>0</value>
17. </property>
18. <property>
19. <name>yarn.nodemanager.env-whitelist</name>
20.
<value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_DIR
,HADOOP_YARN_HOME,HADOOP_MAPRED_HOME</value>
21. </property>
22. </configuration>
```

```

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->

<configuration>
 <property>
 <name>yarn.nodemanager.aux-services</name>
 <value>mapreduce_shuffle</value>
 </property>

 <property>
 <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
 <value>org.apache.hadoop.mapred.ShuffleHandler</value>
 </property>

 <property>
 <name>yarn.resourcemanager.hostname</name>
 <value>127.0.0.1</value>
 </property>

 <property>
 <name>yarn.acl.enable</name>
 <value>0</value>
 </property>

 <property>
 <name>yarn.nodemanager.env-whitelist</name>
 <value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_DIR,HADOOP_YARN_HOME,HADOOP_MAPRED_HOME</value>
 </property>
</configuration>

```

## Explanation:

This configuration sets up **YARN (Yet Another Resource Negotiator)**, which is Hadoop's cluster manager responsible for scheduling and running jobs.

- **yarn.nodemanager.aux-services = mapreduce\_shuffle** → enables the service that moves intermediate data between Map and Reduce jobs.
- **mapreduce.shuffle.class = ShuffleHandler** → tells Hadoop which program handles that shuffle step.
- **yarn.resourcemanager.hostname = 127.0.0.1** → points the ResourceManager to the local machine (since this is single-node).

- **yarn.acl.enable = 0** → turns off strict access control lists (ACLs), so permissions don't block your test setup.
  - **env-whitelist** → allows YARN to use important environment variables like JAVA\_HOME and Hadoop's paths.
- 

## Layman's Explanation:

Think of Hadoop jobs like **students doing a group project**:

- YARN is the **teacher** assigning who does what.
- The **shuffle service** is like passing notes between students so they can share work.
- The **ResourceManager** (teacher's desk) is at your own computer (127.0.0.1).
- ACLs are “permission slips” - here we disable them, so no one gets blocked.
- The whitelist is just the **allowed school supplies** (Java and Hadoop tools) that students can use.

## 29 Step 17: Format the Hadoop Filesystem

### 29.1 Command:

```
hdfs namenode -format
```

---

### 29.2 Explanation:

- This command **initializes the Hadoop Distributed File System (HDFS)** by creating the filesystem structure and necessary metadata.
  - It sets up the **NameNode directories** (which you defined earlier in hdfs-site.xml) so they're ready to store files and track blocks.
  - You only need to run this **once**, right after installation. Running it again later would erase existing HDFS data.
- 

#### 29.2.1 Layman's Explanation:

Think of the **NameNode** as a brand-new **library**.

- Running hdfs namenode -format is like **installing the bookshelves and catalog system** for the first time.
- It doesn't add any books (data) yet, but it prepares the space so books can be stored and organized.
- You must do this setup step once - if you did it again later, it would be like **wiping the library clean** and starting over.

```

= false
2025-09-07 14:58:49,535 INFO snapshot.SnapshotManager: SkipList is disabled
2025-09-07 14:58:49,538 INFO util.GSet: Computing capacity for map cachedBlocks
2025-09-07 14:58:49,538 INFO util.GSet: VM type = 64-bit
2025-09-07 14:58:49,538 INFO util.GSet: 0.25% max memory 3.5 GB = 8.9 MB
2025-09-07 14:58:49,539 INFO util.GSet: capacity = 2^20 = 1048576 entries
2025-09-07 14:58:49,544 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.window.num.buckets = 10
2025-09-07 14:58:49,544 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.num.users = 10
2025-09-07 14:58:49,544 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.windows.minutes = 1,5,25
2025-09-07 14:58:49,547 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
2025-09-07 14:58:49,547 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap and retry cache entry expiry time is 600000 millis
2025-09-07 14:58:49,548 INFO util.GSet: Computing capacity for map NameNodeRetryCache
2025-09-07 14:58:49,548 INFO util.GSet: VM type = 64-bit
2025-09-07 14:58:49,548 INFO util.GSet: 0.029999999329447746% max memory 3.5 GB = 1.1 MB
2025-09-07 14:58:49,548 INFO util.GSet: capacity = 2^17 = 131072 entries
2025-09-07 14:58:49,629 INFO namenode.FSImage: Allocated new BlockPoolId: BP-1024503607-10.1.0.4-1757257129571
2025-09-07 14:58:49,662 INFO common.Storage: Storage directory /disk/hadoop/dfsdata/namenode has been successfully formatted.
2025-09-07 14:58:49,688 INFO namenode.FSImageFormatProtobuf: Saving image file /disk/hadoop/dfsdata/namenode/current/fsimage.ckpt_00000000000000000000 using no compression
2025-09-07 14:58:49,765 INFO namenode.FSImageFormatProtobuf: Image file /disk/hadoop/dfsdata/namenode/current/fsimage.ckpt_00000000000000000000 of size 401 bytes saved in 0 seconds .
2025-09-07 14:58:49,770 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2025-09-07 14:58:49,774 INFO blockmanagement.DatanodeManager: Slow peers collection thread shutdown
2025-09-07 14:58:49,791 INFO namenode.FSNamesystem: Stopping services started for active state
2025-09-07 14:58:49,791 INFO namenode.FSNamesystem: Stopping services started for standby state
2025-09-07 14:58:49,794 INFO namenode.FSImage: FSImageSaver clean checkpoint: txid=0 when meet shutdown.
2025-09-07 14:58:49,794 INFO namenode.NameNode: SHUTDOWN_MSG:
*****SHUTDOWN_MSG: Shutting down NameNode at ProjectPro-P001/10.1.0.4*****
Asiema@ProjectPro-P001:~$ █

```

## 29.3 What the Log Shows

- **Storage directory /disk/hadoop/dfsdata/namenode has been successfully formatted.**

This means HDFS directories were created and initialized. ✓

- **Image file /disk/hadoop/dfsdata/namenode/current/fsimage.ckpt... saved**

A metadata file was saved - this is Hadoop's way of tracking filesystem state.

- **INFO namenode.NameNode: SHUTDOWN MSG:** Hadoop shuts down the NameNode after formatting - that's normal. You don't keep the format process running, it's just a one-time setup.

---

### 29.3.1 Layman's Explanation

Think of this like you **built and labeled all the shelves in your library** (NameNode). The shelves are empty now, but they're ready to store books (your data).

- Hadoop created the structure (/dfsdata/namenode) and a **catalog file** (fsimage) that will help it track files.
  - Once the setup is done, Hadoop **shuts down the builder process** - you only need to start the real library services next.
- 

 So your **format step is successful**.

## 30 Step 18: Start Hadoop Services

**Command:**

```
1. cd /disk/hadoop/hadoop-3.4.1/sbin/
2. ./start-dfs.sh
3. ./start-yarn.sh
4.
```

```
Asiema@ProjectPro-P001:~$ cd /disk/hadoop/hadoop-3.4.1/sbin/
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$./start-dfs.sh ./start-yarn.sh
Usage: start-dfs.sh [-upgrade|-rollback] [-clusterId]
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$./start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [ProjectPro-P001]
ProjectPro-P001: Warning: Permanently added 'projectpro-p001' (ED25519) to the list of known
hosts.
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$./start-yarn.sh
Starting resourcemanager
Starting nodemanagers
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$ █
```

### 30.1 Explanation:

- **./start-dfs.sh** → Starts the **HDFS daemons**:
  - **NameNode** → The master that manages metadata (file locations).
  - **DataNode(s)** → The workers that store actual data blocks.
  - **Secondary NameNode** → A helper that periodically merges metadata.
- **./start-yarn.sh** → Starts the **YARN daemons**:
  - **ResourceManager** → Allocates cluster resources for jobs.
  - **NodeManager(s)** → Manage resources on each worker node.

After running both scripts, your **Hadoop cluster is live** and ready to store files and run jobs.

---

## 30.2 Layman's Explanation:

Think of this like **turning on the power in a new factory**:

- `start-dfs.sh` → switches on the **storage system** (like activating the warehouse, shelves, and catalog system).
- `start-yarn.sh` → switches on the **job manager** (like supervisors and workers who handle tasks inside the warehouse).

Now, your Hadoop “factory” is fully running. From here, you can start putting files into HDFS, run MapReduce jobs, or connect Hive.

---

### Next Steps:

To confirm everything is running fine, you should check the Hadoop daemons with:

1. `jps`
  - 2.

This will list Java processes like NameNode, DataNode, ResourceManager, and NodeManager.

## 31 Step 19: Verify Hadoop Services

**Command:**

```
1. jps
2.
```

---

```
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$./start-yarn.sh
Starting resourcemanager
Starting nodemanagers
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$ jps
17185 NodeManager
16657 DataNode
17044 ResourceManager
16855 SecondaryNameNode
17544 Jps
16461 NameNode
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$ █
```

### 31.1 Explanation:

- jps (Java Process Status) lists all Java processes currently running on your machine.
- After starting Hadoop, you should see the following daemons:
  - **NameNode** → Manages metadata and the HDFS namespace.
  - **DataNode** → Stores actual data blocks on disk.
  - **SecondaryNameNode** → Helps checkpoint the NameNode metadata.
  - **ResourceManager** → Manages resource allocation for jobs.
  - **NodeManager** → Runs tasks on worker nodes.

---

#### 31.1.1 Layman's Explanation:

Think of jps like a **checklist of workers in a factory**:

- It shows you which staff (Hadoop daemons) have clocked in.

- If one of them is missing (e.g., DataNode didn't start), you know something's wrong with that part of the system.
  - Seeing all 4–5 main daemons running means your “Hadoop factory” is fully staffed and ready to work.
- 

### 31.2 Expected Output (example):

1. 12345 NameNode
2. 12346 DataNode
3. 12347 SecondaryNameNode
4. 12348 ResourceManager
5. 12349 NodeManager
- 6.

# HIVE INSTALLATION

Now we will install and set up **Hive** using the commands from the **hiveinstallation.txt** file located in the code folder. After configuring **Hadoop** on the Azure VM, the next step is to install Hive, which is the **main component** that allows us to run **SQL-like Hive queries** on large datasets like the airline data stored in HDFS. These commands will be executed **one by one** on the Azure VM to complete the Hive setup. You can get commands for Hive Installation (HiveInstallation.txt) from code folder.



## 32 Step 1: Navigate to the Home Directory

### Command:

```
1. cd ~
2.
```

---

### Explanation:

- `cd ~` takes you to your **home directory**.
  - The `~` symbol is shorthand for your user's home path (e.g., `/home/Asiema`).
  - This ensures you start from a **clean, well-known location** before downloading or setting up Hive.
- 

### 32.1 Layman's Explanation:

Think of `cd ~` as **going back to your room before starting a new project**.

- Instead of working from a random hallway (some other folder), you return to your home base.
- From here, it's easier to organize downloads, installations, and configurations without getting lost.

## 33 Step 2: Move to the Attached Disk

### Command:

```
1. cd /disk
2.
```

---

```
17544 Jps
16461 NameNode
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$ cd ~
Asiema@ProjectPro-P001:~$ cd /disk
Asiema@ProjectPro-P001:/disk$ █
```

### 33.1 Explanation:

- `cd /disk` moves into the **attached disk** we prepared earlier for Hadoop.
  - This ensures **Hive is stored in the same place** as Hadoop, keeping all big-data tools organized.
  - Helps with consistency and makes it easier to manage configurations later.
- 

#### 33.1.1 Layman's Explanation:

- Think of this like saying:

👉 “Instead of scattering my tools in different rooms, I’m putting both Hadoop and Hive in the same storage room (`/disk`), so I always know where to find them.”
- It keeps things tidy and avoids confusion later when Hive needs to talk to Hadoop.

---

## 34 Step 3: Create Hive Directory

**Command:**

mkdir hive

---

```
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$ cd ~
Asiema@ProjectPro-P001:~$ cd /disk
Asiema@ProjectPro-P001:/disk$ ls
hadoop
Asiema@ProjectPro-P001:/disk$ mkdir hive
Asiema@ProjectPro-P001:/disk$ cd hive
```

### 34.1 Explanation:

- mkdir hive creates a **new folder named hive** inside /disk.
  - This serves as a **dedicated space for Hive files** separate from Hadoop's folder.
  - Keeps installations tidy and prevents conflicts.
- 

### 34.2 Layman's Explanation:

Imagine you're /disk is a **bookshelf**. You already have one shelf labeled **Hadoop**. Now, you're adding another labeled **Hive**.

👉 This way, Hadoop books and Hive books don't get mixed up, but they still live on the same bookshelf for easy access.

## 35 Step 4: Move Into the Hive Folder

**Command:**

cd hive

---

```
Asiema@ProjectPro-P001:/disk$ ls
hadoop
Asiema@ProjectPro-P001:/disk$ mkdir hive
Asiema@ProjectPro-P001:/disk$ cd hive
Asiema@ProjectPro-P001:/disk/hive$ █
```

### 35.1 Explanation:

- The command `cd hive` changes the current working directory into the **hive** folder you created earlier.
  - This ensures that when you download Hive, all files will be stored **inside** `/disk/hive`, keeping the installation tidy and organized.
- 

#### 35.1.1 Layman's Explanation:

Imagine your computer's storage is like a **house with many rooms**:

- You just **built a new room** called **Hive** inside the house.
- With `cd hive`, you're literally **walking into that room**.
- Now, when the Hive software arrives (downloaded), it will be placed neatly **inside this room**, instead of being dropped randomly around the house.

## 36 Step 5: Move Back to Parent Directory

**Command:**

cd ..

---

```
Asiema@ProjectPro-P001:/disk$ cd hive
Asiema@ProjectPro-P001:/disk/hive$ cd ..
Asiema@ProjectPro-P001:/disk$ █
```

### 36.1 Explanation:

- The cd .. command moves you **one level up** in the folder hierarchy.
  - Since you were inside /disk/hive, this command takes you back to /disk.
  - This is needed so you can clearly specify the **full path** (/disk/hive) in the next download command when fetching Hive.
- 

### 36.2 Layman's Explanation:

Think of your computer folders like **nested boxes**:

- You were inside the **Hive box** (inside the **Disk box**).
- The command cd .. means **step out of the Hive box** and go back into the **Disk box**.
- This makes it easier to tell the system, “Hey, when I download Hive, put it inside that Hive box inside Disk.”

### 36.3 Error Occured

```
Asiema@ProjectPro-P001:/disk$ wget https://archive.apache.org/... -P /hive
--2025-09-07 16:59:52-- https://archive.apache.org/...
Resolving archive.apache.org (archive.apache.org) ... 65.108.204.189, 2a01:4f9:1a:a084::2
Connecting to archive.apache.org (archive.apache.org)|65.108.204.189|:443... connected.
HTTP request sent, awaiting response... 404 Not Found
2025-09-07 16:59:53 ERROR 404: Not Found.
```

URL you're using is outdated or incorrect.

### 36.4 Finding the Correct Hive Download URL

- The latest **stable release** of Hive is **4.1.0**, released on **July 30, 2025**.  
<https://hive.apache.org+1>
- You can find it in the official directory listing:

<https://downloads.apache.org/hive/hive-4.1.0/downloads.apache.org>

---

## 37 Correct Download Command (Step 6)

Here's how to properly download the Hive binary into your /disk/hive directory:

```
1. cd /disk/hive
2. wget -P . https://downloads.apache.org/hive/hive-4.1.0/apache-hive-4.1.0-bin.tar.gz
3.
```

```
Asiema@ProjectPro-P001:/disk$ cd /disk/hive
wget -P . https://downloads.apache.org/hive/hive-4.1.0/apache-hive-4.1.0-bin.tar.gz
--2025-09-07 17:07:24-- https://downloads.apache.org/hive/hive-4.1.0/apache-hive-4.1.0-bin.t
ar.gz
Resolving downloads.apache.org (downloads.apache.org) ... 135.181.214.104, 88.99.208.237, 2a01
:4f9:3a:2c57::2, ...
Connecting to downloads.apache.org (downloads.apache.org)|135.181.214.104|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 491971556 (469M) [application/x-gzip]
Saving to: './apache-hive-4.1.0-bin.tar.gz'

apache-hive-4.1.0-bin.t 100%[=====] 469.18M 15.4MB/s in 32s

2025-09-07 17:07:58 (14.5 MB/s) - './apache-hive-4.1.0-bin.tar.gz' saved [491971556/491971556]
```

```
Asiema@ProjectPro-P001:/disk/hive$
```

### 37.1 Breakdown:

- **cd /disk/hive** - Ensures the download goes into your Hive folder.
- **wget -P . URL** - Downloads directly into the current folder (./), so your Hive files will land neatly inside /disk/hive.
- The URL points to the **binary distribution** of Hive 4.1.0.

Alternatively, if you run into mirror issues, you can use:

```
1. wget -P . https://archive.apache.org/dist/hive/hive-4.1.0/apache-hive-4.1.0-bin.tar.gz
2.
```

This fallback uses the Apache archive site, which mirrors all releases.[archive.apache.org](https://archive.apache.org)

---

### 37.2 Why the Previous URL Failed

- Your earlier command likely didn't match the exact Hive release path or included typos. For example, missing the correct folder (e.g., hive-4.1.0) or using an incorrect filename (.tar.gz instead of -bin.tar.gz) will lead to 404 errors.

## 38 Step 6: Go Back Into the Hive Directory

**Command:**

```
cd /disk/hive
```

```
2025-09-07 17:07:58 (14.5 MB/s) - `./apache
]

Asiema@ProjectPro-P001:/disk/hive$ ls
apache-hive-4.1.0-bin.tar.gz
Asiema@ProjectPro-P001:/disk/hive$ █
```

### 38.1 What it does:

- Moves you back into the **Hive folder** where you downloaded the Hive archive.
  - This ensures you're in the correct working directory before running the extraction command.
- 

#### 38.1.1 Layman's Explanation

Think of /disk/hive as a dedicated cabinet where you're storing all Hive-related files. You've just placed a **big, zipped box (Hive tar.gz file)** into it. Now, before opening the box, you need to **stand inside that cabinet** so that when you unpack the box, everything lands neatly inside Hive's space instead of being scattered elsewhere.

## 39 Step 7: Extract the Hive Package

### Command:

```
tar xzf apache-hive-4.1.2-bin.tar.gz
```

### What it does:

- **tar** → the command to extract compressed archives.
- **xzf** →
  - x: extract files
  - z: use gzip (since it's a .tar.gz file)
  - f: specify the filename that follows
- **apache-hive-4.1.2-bin.tar.gz** → the Hive archive file you downloaded.

When you run this, it unpacks Hive into a folder named **apache-hive-4.1.2-bin** inside /disk/hive.

---

```
2025-09-07 17:07:58 (14.5 MB/s) - './apache-hive-4.1.0-bin.tar.gz' saved
]

Asiema@ProjectPro-P001:/disk/hive$ ls
apache-hive-4.1.0-bin.tar.gz
Asiema@ProjectPro-P001:/disk/hive$ ^C
Asiema@ProjectPro-P001:/disk/hive$ tar xzf apache-hive-4.1.0-bin.tar.gz
Asiema@ProjectPro-P001:/disk/hive$ █
```

## 39.1 Layman's Explanation

- Think of the Hive .tar.gz file as a **zip file** or a **sealed box** that contains all the parts of Hive (programs, scripts, and libraries). Running this command is like **opening the box and neatly arranging all its contents into a folder**, so you can start using Hive right away.
  - Without this, Hive would still be locked away in the compressed file and unusable.
- 

 After running this, you should see a new directory called:

apache-hive-4.1.2-bin

inside /disk/hive.

## 40 Step 8: Move to Home Directory

**Command:**

```
cd ~
```

### 40.1 What it does:

- **cd** → changes the current working directory.
- **~** → represents your home directory (for example /home/Asiema).

So, this command moves you out of /disk/hive and brings you back to your **home folder**, where files like .bashrc are located.

---

### 40.2 Layman's Explanation

- Think of your **home directory** like your **personal workspace or desk**. Before you start editing your system's personal notebook (.bashrc), you return to your desk so you can open and update it from the right place.
- It's a simple reset to a familiar location before making important changes.

## 41 Step 9: Hive

### Command:

```
sudo nano .bashrc
```

- Opens the .bashrc file in the nano text editor.
- This is where you configure **environment variables** so that Hive commands can run globally.

```
enable programmable completion features (you don't need to enable
this, if it's already enabled in /etc/bash.bashrc and /etc/profile
sources /etc/bash.bashrc).
if ! shopt -oq posix; then
 if [-f /usr/share/bash-completion/bash_completion]; then
 . /usr/share/bash-completion/bash_completion
 elif [-f /etc/bash_completion]; then
 . /etc/bash_completion
 fi
fi

export HADOOP_HOME=/disk/hadoop/hadoop-3.4.1
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
export PATH=$JAVA_HOME/bin:$PATH
export HADOOP_HOME=/disk/hadoop/hadoop-3.4.1
export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64

export HIVE_HOME="/disk/hive/apache-hive-4.1.0-bin"
export PATH=$PATH:$HIVE_HOME/bin
```

## 41.1 What you added:

```
1. export HIVE_HOME="/disk/hive/apache-hive-4.1.0-bin"
2. export PATH=$PATH:$HIVE_HOME/bin
3.
```

- **HIVE\_HOME** → tells the system where Hive is installed (/disk/hive/apache-hive-4.1.0-bin).
  - **PATH** → adds Hive's bin folder to your PATH so you can run Hive commands (hive, beeline) from anywhere.
- 

## 41.2 Explanation in Layman's Terms:

Think of .bashrc like your computer's **notebook of shortcuts**.

- You just wrote down: "*Whenever I type hive, go to this folder and run it.*"
- Without this, your system wouldn't know where Hive is installed.
- It's like giving directions so your OS can always find Hive, no matter where you are in the terminal.

**Command:**

```
source ~/.bashrc
```

---

### 41.3 What it does:

- **Reloads .bashrc** → applies all the new environment variable changes you just made.
  - **Makes Hive available right away** → no need to log out or restart your terminal.
- 

#### 41.3.1 Layman's Terms:

Think of .bashrc like your **settings file**. When you update it, your system doesn't automatically know about the changes.

- Running source ~/.bashrc is like **clicking “refresh”** so the computer remembers your new Hive shortcuts.
- Without it, typing hive would still give a “command not found” error until you restarted.

## 42 Step 11: Configuring Hive to know where Hadoop is installed.

### Command:

```
sudo nano $HIVE_HOME/bin/hive-config.sh
```

---

```
Asiema@ProjectPro-P001:~$ sudo nano .bashrc
Asiema@ProjectPro-P001:~$ source ~/.bashrc
Asiema@ProjectPro-P001:~$ sudo nano $HIVE_HOME/bin/hive-config.sh
Asiema@ProjectPro-P001:~$ hdfs dfs -mkdir /tmp
Asiema@ProjectPro-P001:~$ █
```

### 42.1 What to do inside:

At the **bottom** of this file, add the following line:

```
export HADOOP_HOME=/disk/hadoop/hadoop-3.4.1
```

---

Then save and exit (Ctrl+O, Enter, Ctrl+X in nano).

---

### 42.2 What it does:

- Opens Hive's **config script** (hive-config.sh).
  - By adding HADOOP\_HOME, you are telling Hive **where to find Hadoop** so the two can work together.
-

#### 42.2.1 Layman's Terms:

Think of Hive as a **car driver** and Hadoop as the **car engine**.

- Hive knows how to drive but doesn't know **where the engine is located**.
- By setting `HADOOP_HOME`, you're pointing Hive to Hadoop's garage (`/disk/hadoop/hadoop-3.4.1`) so it can actually run queries on HDFS.

## 43 Step 13: Fixing Hive permissions in HDFS.

**Command:**

---

```
hdfs dfs -chmod g+w /tmp
```

---

**What it does:**

- Gives **group write permissions** to the /tmp directory inside **HDFS** (Hadoop Distributed File System).
  - This ensures that Hive and other users/services can **write temporary files** during queries.
- 

### 43.1.1 Layman's Terms:

Think of /tmp as a **shared workspace table** where Hive puts scratch notes while solving queries.

- Without permission, only one person could write on the table, and Hive would complain.
  - By giving **group write access**, you're saying: "*Everyone in the group can use this workspace table at the same time.*"
  - This prevents Hive jobs from failing due to **permission denied errors**.
- 

 After this, Hive queries should run smoother.

👉 Do you want me to also cover the **next step where we create the Hive warehouse directory (/user/hive/warehouse)?**

## 44 Step 14: Creating the Hive warehouse directory in HDFS.

### ◆ Command:

```
hdfs dfs -mkdir -p /user/hive/warehouse
```

---

### 44.1 What it does:

- Creates the directory /user/hive/warehouse inside **HDFS**.
  - This is where **Hive stores all its table data** (both managed and external tables).
  - The -p flag ensures that if parent directories like /user/hive don't exist, they're created too.
- 

```
Asiema@ProjectPro-P001:~$ sudo nano $HIVE_HOME/bin/hive-config.sh
Asiema@ProjectPro-P001:~$ hdfs dfs -mkdir /tmp
Asiema@ProjectPro-P001:~$ hdfs dfs -chmod g+w /tmp
Asiema@ProjectPro-P001:~$ hdfs dfs -mkdir -p /user/hive/warehouse
Asiema@ProjectPro-P001:~$ █
```

#### 44.1.1 Layman's Terms:

Think of this as **building a storage warehouse**  in Hadoop's filesystem where Hive will **keep all the tables you create.**

- Every time you run something like CREATE TABLE students, Hive needs a **place to save the data files.**
- This /user/hive/warehouse directory is the **default home** for all Hive tables.

Without this step, Hive wouldn't know where to store your data, and queries would

### 45 Step 15: Setting Permissions for the Hive Warehouse Directory.

**Command:**

```
hdfs dfs -chmod g+w /user/hive/warehouse
```

---

```
Asiema@ProjectPro-P001:~$ sudo nano .bashrc
Asiema@ProjectPro-P001:~$ source ~/.bashrc
Asiema@ProjectPro-P001:~$ sudo nano $HIVE_HOME/bin/hive-config.sh
Asiema@ProjectPro-P001:~$ hdfs dfs -mkdir /tmp
Asiema@ProjectPro-P001:~$ hdfs dfs -chmod g+w /tmp
Asiema@ProjectPro-P001:~$ hdfs dfs -mkdir -p /user/hive/warehouse
Asiema@ProjectPro-P001:~$ hdfs dfs -chmod g+w /user/hive/warehouse
Asiema@ProjectPro-P001:~$ █
```

#### 45.1 What it does:

- Adds **group write permissions** to the /user/hive/warehouse folder in HDFS.
- This ensures Hive can **create, modify, and delete table files** inside the warehouse directory.

- Without this, Hive queries like CREATE TABLE or INSERT INTO may fail because Hive won't have permission to write data.
- 

#### 45.1.1 Layman's Terms:

Think of the warehouse folder  as a **shared storage room** where Hive wants to put all the tables.

- By default, the room is **locked for writing**.
- This command is like **giving your team the keys**  so Hive (and its users) can actually place and update files inside.

👉 Without this permission, Hive can look into the room but won't be able to store or update anything there.

### 46 Step 16: Hive Configuration Directory.

**Command:**

```
cd $HIVE_HOME/conf
```

---

```
Asiema@ProjectPro-P001:~$ hdfs dfs -chmod g+w /tmp
Asiema@ProjectPro-P001:~$ hdfs dfs -mkdir -p /user/hive/warehouse
Asiema@ProjectPro-P001:~$ hdfs dfs -chmod q+w /user/hive/warehouse
Asiema@ProjectPro-P001:~$ cd $HIVE_HOME/conf
Asiema@ProjectPro-P001:/disk/hive/apache-hive-4.1.0-bin/conf$
```

#### 46.1 What it does:

- Switches you into Hive's **configuration folder**.

- This folder contains the key config files Hive needs to run, such as:
    - `hive-site.xml` → defines Hive settings (metastore DB, warehouse dir, etc.)
    - `hive-default.xml.template` → a template you can copy to create a working config
    - `hive-env.sh` → environment variables for Hive runtime
- 

#### 46.1.1 Layman's Terms:

Think of this folder as Hive's **control room** .

- Inside it are all the **instruction manuals** (config files) that tell Hive how to behave.
- For example, where to store data, how to connect to Hadoop, and what environment to run in.

 By moving here, you're preparing to **customize Hive's settings**, so it works properly with your Hadoop setup.

## 47 Step 17: Create the Hive Config File.

### ◆ Command:

```
cp hive-default.xml.template hive-site.xml
```

---

### 47.1 What it does:

- Takes the **default Hive config template** (hive-default.xml.template).
  - Makes a **copy named hive-site.xml**.
  - Hive will actually **read settings from hive-site.xml**, not the template.
- 

#### 47.1.1 Layman's Terms:

Think of it like moving into a new house .

- The **template** is like a **blueprint**-it has example settings but isn't active.
- By copying it into **hive-site.xml**, you're saying:  
 “Okay Hive, here's your real instruction book. Now we can fill in the important details (like where to store tables, how to talk to Hadoop, etc.).”

## 48 STEP 18. sudo nano hive-site.xml

- Opens the Hive configuration XML file
  - Where we define Hive system properties
- 

### a) Add <property> entries (like temp dirs) into the file:

```
http://www.apache.org/licenses/LICENSE-2.0
```

```
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

-><configuration>
<!-- WARNING!!! This file is auto generated for documentation purposes ONLY! -->
<!-- WARNING!!! Any changes you make to this file will be ignored by Hive. -->
<!-- WARNING!!! You must make your changes in hive-site.xml instead. -->
<!-- Hive Execution Parameters -->

<property>
 <name>system:java.io.tmpdir</name>
 <value>/tmp/hive/java</value>
</property>

<property>
 <name>system:user.name</name>
 <value>${user.name}</value>
</property>

<property>
 <name>hive.metastore.client.cache.v2.enabled</name>
 <value>true</value>
 <description>This property enables a Caffeine Cache for Metastore client</description>
</property>
<property>
 <name>hive.metastore.client.cache.v2.maxSize</name>
 <value>1Gb</value>
 <description>
 Expects a byte size value with unit (blank for bytes, kb, mb, gb, tb, pb).
 Set the maximum size (number of bytes) of the metastore client cache (DEFAULT: 1GB). O
 </description>
</property>
<property>
 <name>hive.metastore.client.cache.v2.recordStats</name>
 <value>false</value>
```

1. <property>
2. <name>system:java.io.tmpdir</name>
3. <value>/tmp/hive/java</value>
4. </property>
- 5.
6. <property>
7. <name>system:user.name</name>
8. <value>\${user.name}</value>
9. </property>

- ◆ These tell Hive where to place **temporary files** and how to reference the **current user**.

### b) Look for the **transactional tables section**

```

</property>
<property>
 <name>hive.locks.max.partitions</name>
 <value>-1</value>
 <description>Locks the entire table if number of partition locks exceeds user-defined threshold</description>
</property>
<property>
 <name>hive.txn.xlock.iow</name>
 <value>true</value>
 <description>
 Ensures MERGE INSERT operations acquire EXCLUSIVE / EXCL_WRITE lock for transactional tables.
 If enabled, prevents duplicates when MERGE statements are executed in parallel transactions.
 </description>
</property>
<property>
 <name>hive.txn.xlock.write</name>
 <value>false</value>
 <description>
 Manages concurrency levels for ACID resources. Provides better level of query parallelism.
 </description>
</property>

```

- If you see strange/unwanted characters in the comments or properties, clean them up.
- Keep it simple, like:

1. <property>
2. <name>hive.txn.xlock.iow</name>
3. <value>true</value>
4. <description>Ensures exclusive locks for transactional tables.</description>
5. </property>
- 6.

- ◆ This ensures Hive can handle **transactional tables** properly (tables that support INSERT/UPDATE/DELETE).

### c) Look for `hive.metastore.schema.verification`

```
</property>
<property>
 <name>datanucleus.schema.autoCreateAll</name>
 <value>false</value>
 this to false, after creating it once.To enable auto create also set hive.metastore.schema.verification
</property>
<property>
 <name>hive.metastore.schema.verification</name>
 <value>false</value>
 <description>
 Enforce metastore schema version consistency.
 True: Verify that version information stored in is compatible with one from Hive jars.
 schema migration attempt. Users are required to manually migrate schema after Hive
 proper metastore schema migration. (Default)
 False: Warn if the version information stored in metastore doesn't match with one from
 </description>
```

- Change value from **true** → **false**.

```
1. <property>
2. <name>hive.metastore.schema.verification</name>
3. <value>false</value>
4. <description>
5. Skip strict schema checks so Hive can auto-create necessary
metadata.
6. </description>
7. </property>
8.
```

- ◆ Setting this to **false** makes Hive more flexible, allowing it to auto-create or adjust schemas instead of failing with version mismatches.

---

#### 48.1.1 Layman's Explanation:

Think of this as **customizing Hive's settings** so it knows:

1. Where to drop **scratch/temporary files** (like drafts while working).
2. How to deal with **transaction tables** without confusion.

3. To be **less strict about schema checks** so Hive starts smoothly instead of blocking you with version errors.

It's like adjusting the "preferences" menu before you start using a new software.

## 49 STEP 19 initializing Hive's metastore

This step is about **initializing Hive's metastore** (the database Hive uses to store metadata about your tables, partitions, and schemas).

Here's the breakdown:

### Command

```
cd $HIVE_HOME
$HIVE_HOME/bin/schematool -dbType derby -initSchema
```

### 49.1 What it does

- **cd \$HIVE\_HOME** → moves into the Hive installation folder.
- **schematool -dbType derby -initSchema** → uses Hive's built-in schema tool to create all the necessary metadata tables inside an **embedded Derby database**.

### 49.2 Why it matters

- Hive needs a **metastore** to remember table names, columns, partitions, and other metadata.
- Without this initialization, Hive won't know how to track or query data.
- Since you're using the default **Derby** backend here, it will set up a single-user database (good for testing or single-node setups).

## Layman's terms:

Think of this as giving Hive a “**notebook**” where it can write down every table and dataset you create. Without the notebook, Hive would forget everything each time you close it.

---

 Note: Derby is good for practice, but in production, people usually replace it with **MySQL** or **PostgreSQL** as the Hive metastore.

## 50 STEP 20: Launching Hive

When the user runs the command:

```
hive
```

It performs the following:

### 50.1 Launches the Hive CLI (Command Line Interface)

- Hive CLI allows the user to execute HiveQL (Hive Query Language) statements interactively.
- It connects to the Hive Metastore and initializes the necessary Hadoop and Hive configurations.

### 50.2 Logs and Warnings

- The CLI logs show binding messages from SLF4J (Simple Logging Facade for Java), indicating how logging is handled in the backend. These messages are mostly internal:
- SLF4J: Class path contains multiple SLF4J bindings.
- SLF4J: See [http://www.slf4j.org/codes.html#multiple\\_bindings](http://www.slf4j.org/codes.html#multiple_bindings) for an explanation.
- Hive notifies that it is using configuration files from specific locations:
- Logging initialized using configuration in file:/<path-to-hive-config>
- A **Hive Session ID** is generated:
- Hive Session ID = <UUID>

This identifies your interactive session and helps in logging and resource tracking.

```
Asiemea@ProjectPro-P001:/disk/hive$ hive
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/disk/hive/apache-hive-3.1.2-bin/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/disk/hadoop/hadoop-3.4.1/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = 098e4c3c-5b2d-461e-8c10-701a73da1abf

Logging initialized using configuration in jar:file:/disk/hive/apache-hive-3.1.2-bin/lib/hive-common-3.1.2.jar!/hive-log4j2.properties Async: true
Hive Session ID = 215dd687-1400-4091-a1f2-dad266230460
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
hive> █
```

### 50.2.1 Deprecation Warning:

- Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions.

This warns that the old method of executing Hive queries via MapReduce (hive.execution.engine=mr) is outdated and might not be supported in newer versions. You are advised to use **Tez** or **Spark** instead.

## 50.3 The Prompt Appears

hive>

This shows that you are now inside the Hive CLI and can type HiveQL commands like:

---

SELECT \* FROM tablename;

## 50.4 Layman's Explanation

Imagine you're opening a special type of spreadsheet editor for your big datasets but instead of clicking, you **type commands** to view and analyze the data.

Here's what happened:

1. You typed `hive` - like double-clicking to open the program.
2. The system prepared a workspace and showed some "behind-the-scenes" messages. These are like your computer saying:
  - "I'm loading all the tools you need..."
  - "Oh, I noticed a few repeated tools, but I'll pick one..."
  - "By the way, the old way of doing things might not work in the future - maybe update later!"
3.  A unique ID was created for your session - like your name tag while working.
4.  You now see `hive>` - a blinking prompt waiting for you to start typing questions like:
  5. Show me all the products.

or

What's the average sales per day?

- ✓ From here, you can **explore your big data using SQL-like commands**, directly from your keyboard - no clicking needed.

## 51 STEP 21: Test Hive

This sequence demonstrates how to **verify that Hive is working** by creating and listing a database.

```
show schemas;
```

This is **synonymous** with show databases; in Hive (both list all databases).

**Output:**

```
default
```

Meaning only the default database currently exists.

---

```
show databases;
```

Same as above - shows all available Hive databases.

**Output:**

```
default
```

---

```
create database test;
```

This creates a **new Hive database** named test.

**Output:**

```
OK
```

---

#### 4. show databases;

Now you'll see:

default

test

-  This confirms the new database test was successfully created.
- 

#### 5. show schemas;

Returns the same list again:

default

test

---

```
Logging initialized using configuration in jar:file:/disk/hive/apache-hive-3.1.2-bin/lib/hive-common-3.1.2.jar!/hive-log4j2.properties Async: true
Hive Session ID = e0fc123c-3110-46fb-ad99-a07b102d4f83
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
hive> show databases;
OK
default
Time taken: 0.458 seconds, Fetched: 1 row(s)
hive> create database test;
OK
Time taken: 0.069 seconds
hive> show databases;
OK
default
test
Time taken: 0.021 seconds, Fetched: 2 row(s)
hive> █
```

---

## **Summary of Commands Used:**

<b>Command</b>	<b>Purpose</b>
<b>show schemas;</b>	Lists all Hive databases (schemas).
<b>create database x</b>	Creates a new Hive database.
<b>show databases;</b>	Also lists all Hive databases.

---

## 51.1 Layman's Explanation

Imagine you're using a giant virtual **filing cabinet** for organizing data.

### Step-by-step like you're testing the cabinet:

1. You ask: "**What drawers (databases) do I have?**"

 Hive says: "You only have the **default** drawer."

2. You ask again, using a slightly different phrase (schemas vs. databases).

 Answer is the same: "Just **default**."

3. You say: "**Let me add a new drawer called test.**"

 Hive says: "Sure, it's added."

4. You check again: "**What drawers do I have now?**"

 Hive replies: "Now you have **default** and **test**."

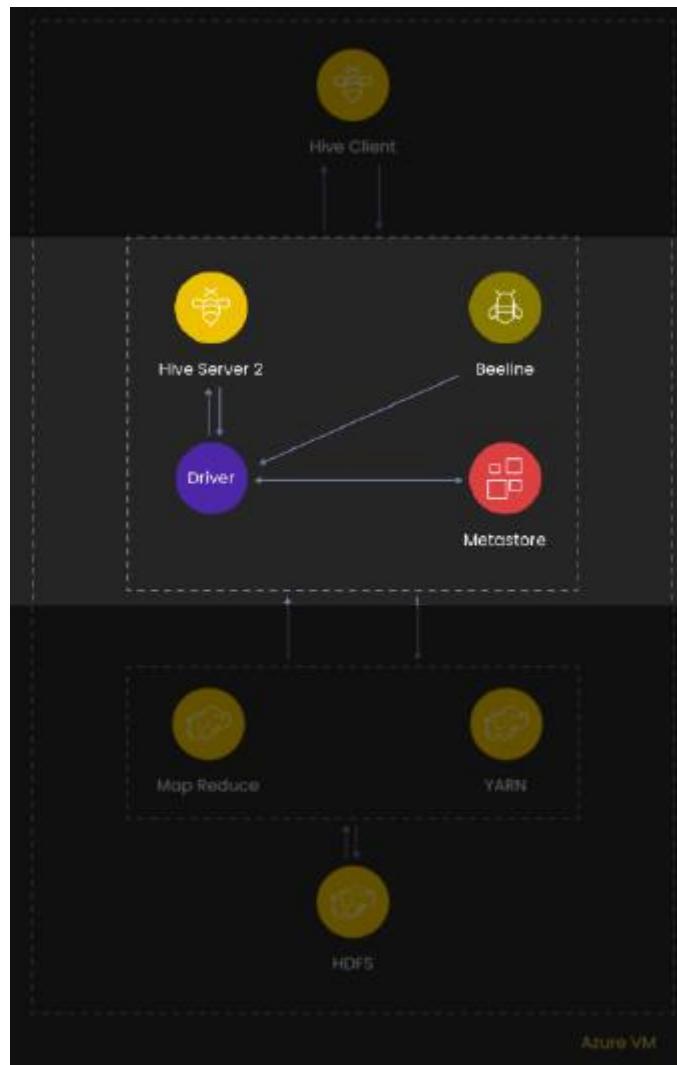
5. You double-check one more time with show schemas;.

 Yep! Both drawers (**default** and **test**) are still there.

---

# BEEHIVE SETUP

The diagram shows how **Beeline** connects to **HiveServer2**, which in turn works with several components to execute queries on HDFS-based data.



## 51.2 Key Components:

### 51.2.1 Beeline (CLI Client)

- A JDBC-based command-line interface.
- Connects to HiveServer2 (not to Hive CLI directly).
- Supports multi-user environments securely.

### 51.2.2 HiveServer2

- Accepts **JDBC/ODBC client connections** (like Beeline).
- Manages sessions, executes queries, returns results.
- Interfaces with the **Driver**, **Metastore**, and **execution engines** like MapReduce, Tez, or Spark.

### 51.2.3 Driver

- Coordinates query execution.
- Parses, compiles, and optimizes HiveQL queries.
- Interacts with:
  - **Metastore** to get schema/metadata.
  - **Execution engine** to run jobs (MapReduce, Tez, etc.).

### 51.2.4 Metastore

- Stores metadata: databases, tables, columns, partitions, file locations, etc.
- Accessible via a Thrift server (even from remote clients).

### 51.2.5 Execution Layer (YARN + MapReduce/Tez)

- Processes the actual data in HDFS.

- The execution is distributed and handled across nodes.

### 51.2.6 HDFS

- Stores your big data files.
  - Hive reads/writes from/to HDFS via execution engines.
- 

## 51.3 Beeline Setup in Simple Layman's Terms

Imagine this is your **Hive restaurant setup**:

Role	Description
 <b>You (Beeline)</b>	You are the customer placing an order - the interface that sends SQL queries
 <b>HiveServer2</b>	The waiter - takes your order, sends it to the kitchen, returns your results
 <b>Metastore</b>	The recipe book - where it checks what ingredients (tables, columns) exist
 <b>Driver</b>	The chef - plans how to cook the meal (query plan), picks tools
 <b>YARN/MapReduce</b>	The kitchen tools & appliances - perform the actual cooking (data processing)
 <b>HDFS</b>	The pantry - where all the food (data files) is stored

So, when you send a query via **Beeline**, it goes to HiveServer2, which:

1. Checks the **Metastore** for what's in the database.

2. Uses the **Driver** to figure out how to run the query.
  3. Hands it to **MapReduce or Tez** to fetch/process the data from **HDFS**.
  4. Returns the answer to you.
- 

## 51.4 Why Use Beeline + HiveServer2?

Benefit	Description
<input checked="" type="checkbox"/> <b>Multi-user</b>	Supports multiple clients, unlike old Hive CLI
<input checked="" type="checkbox"/> <b>JDBC/ODBC</b>	Works with BI tools like Tableau, Power BI
<input checked="" type="checkbox"/> <b>Secure</b>	Can enforce authentication (Kerberos, LDAP)
<input checked="" type="checkbox"/> <b>Production-grade</b>	Better for clusters and real-world workloads

---

## 51.5 What to Set Up Next?

If you're moving toward this architecture (instead of embedded Derby):

- Set up Hive with **MySQL Metastore**
- Enable and start **HiveServer2**
- Use `beeline -u jdbc:hive2://localhost:10000` to connect
- Stop using embedded Derby mode

## 52 Organizing Data in Local Directories (on Azure VM)

- You create folders like /data/flights/, /data/airports/.
- Formats: .csv, .bz2, .tsv.
- Hive is not a text editor - it assumes files are clean, consistent, and ready for ingestion.

### 52.1 Using SCP to Transfer Files to the VM

- scp (secure copy) pushes files from your **laptop → Azure VM**.
- Without this, Hive has nothing in HDFS to read.
- Typical command:
- `scp flights.csv user@vm-ip:/disk/hive/data/`

### 52.2 Unzipping and Decompressing Files

- Real datasets are often compressed (.zip, .bz2) to save space.
- Hive needs plain text (.csv, .txt) for ingestion.
- You use tar, gunzip, or bunzip2 to prepare the files.

---

### 52.3 Layman's Analogy

Think of this as **prepping groceries before cooking**:

1.  **Organizing** = Sorting veggies by type (carrots in one basket, onions in another).

2.  **Transferring (scp)** = Bringing groceries from the market (your laptop) to your kitchen (Azure VM).
3.  **Unzipping** = Taking items out of sealed packages before cooking.

Only after this can the **chef (Hive)** start cooking queries. 

## 53 STEP 1: Beeline setup

- When you use **Beeline** → **HiveServer2**, you are no longer directly accessing Hadoop/HDFS yourself.
  - HiveServer2 needs permission to **act as a proxy user** (it impersonates the actual end-user when running queries).
  - Without this, HiveServer2 cannot read/write to HDFS on behalf of Beeline users.
- 

### ◆ What You Edited:

File:

\$HADOOP\_HOME/etc/hadoop/core-site.xml

You added:

```
1. <property>
2. <name>hadoop.proxyuser.Asiema.hosts</name>
3. <value>*</value>
4. </property>
5.
6. <property>
7. <name>hadoop.proxyuser.Asiema.groups</name>
8. <value>*</value>
9. </property>
10.
```

---

```

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License. See accompanying LICENSE file.
-->

<!-- Put site-specific property overrides in this file. -->

<configuration>
 <property>
 <name>hadoop.tmp.dir</name>
 <value>/disk/hadoop/tmpdata</value>
 </property>
 <property>
 <name>fs.defaultFS</name>
 <value>hdfs://127.0.0.1:9000</value>
 </property>
</configuration>
<property>
 <name>hadoop.proxyuser.Asiema.hosts</name>
 <value>*</value>
</property>

<property>
 <name>hadoop.proxyuser.Asiema.groups</name>
 <value>*</value>
</property>

```

## Meaning of These Properties:

### 1. **hadoop.proxyuser.Asiema.hosts**

- Says from which **hosts** the user Asiema is allowed to submit jobs.
- \* means **any host is allowed**.

### 2. **hadoop.proxyuser.Asiema.groups**

- Defines which **Linux groups** the user Asiema can impersonate.
- \* means **all groups**.

So you're basically telling Hadoop:

“Hey, whenever the user Asiemka (HiveServer2) tries to act as another user, let it - no restrictions on host or group.”

---

### 53.1 Layman’s Explanation

Think of **HiveServer2** as a **secretary** (the proxy user).

- The **employees** (Beeline clients) don’t directly enter the company’s records (HDFS).
- Instead, they **hand their requests to the secretary (HiveServer2)**.
- The secretary then **acts on behalf of each employee**, filing or fetching data.

The configuration you added is basically:

- **Hosts = \*** → The secretary can accept requests from anywhere.
- **Groups = \*** → The secretary can impersonate any employee group.

Without this permission, the secretary (HiveServer2) would be stopped at the door and couldn’t do anything on behalf of the employees.

---

## 53.2 Note on Security

Using \* for both **hosts** and **groups** is fine for a test environment, but in production you

would want to **restrict** this:

1. <property>
2.     <name>hadoop.proxyuser.hive.hosts</name>
3.     <value>my-hive-server-hostname</value>
4. </property>
- 5.
6. <property>
7.     <name>hadoop.proxyuser.hive.groups</name>
8.     <value>hadoop, analysts</value>
9. </property>
- 10.

This limits impersonation to known hosts and groups.

## 54 STEP 2: Cleanly restarting your Hadoop environment

**Command:**

```
cd /disk/hadoop/hadoop-3.41/sbin/
```

```
./stop-dfs.sh & ./stop-yarn.sh
```

- **stop-dfs.sh** → Stops **HDFS services** (NameNode, DataNode, SecondaryNameNode).
- **stop-yarn.sh** → Stops **YARN services** (ResourceManager, NodeManager).

```
^C
[1]+ Done ./stop-dfs.sh
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$ cp $HADOOP_HOME/etc/hadoop/core-site.xml{,.bak}
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$ nano $HADOOP_HOME/etc/hadoop/core-site.xml
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$ cd $HADOOP_HOME/sbin
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$./stop-dfs.sh; ./stop-yarn.sh
Stopping namenodes on [localhost]
Stopping datanodes
Stopping secondary namenodes [ProjectPro-P001]
Stopping nodemanagers
Stopping resourcemanager
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$./start-dfs.sh; ./start-yarn.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [ProjectPro-P001]
Starting resourcemanager
Starting nodemanagers
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$ █
```

### 54.1 Why Stop First?

- Configuration files (like core-site.xml, hdfs-site.xml, yarn-site.xml) are read **only at service startup.**
- If you don't stop & restart, your changes won't apply.
- Stopping ensures no corrupted/half-running services are left behind before restart.

#### 54.1.1 Layman's Analogy

Imagine you're updating the rules of a game 🎮.

- If players are **already playing**, they won't follow the new rules until the game is restarted.
  - Stopping Hadoop services = **ending the old game session**.
  - Restarting later = **starting fresh with the updated rulebook** (core-site.xml with your new proxy settings).
- 

## 54.2 What To Do Next (Restart Sequence)

After stopping:

1. Start HDFS:
2. ./start-dfs.sh
3. Start YARN:
4. ./start-yarn.sh
5. Start Hive Metastore (if not embedded Derby):
6. hive --service metastore &
7. Start HiveServer2:
8. hive --service hiveserver2 &
9. Connect using Beeline:
10. beeline -u jdbc:hive2://localhost:10000 -n Asiemaa

## Technical Explanation

### Command:

```
./start-dfs.sh & ./start-yarn.sh
```

```
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$ cd $HADOOP_HOME/sbin
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$./stop-dfs.sh; ./stop-yarn.sh
Stopping namenodes on [localhost]
Stopping datanodes
Stopping secondary namenodes [ProjectPro-P001]
Stopping nodemanagers
Stopping resourcemanager
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$./start-dfs.sh; ./start-yarn.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [ProjectPro-P001]
Starting resourcemanager
Starting nodemanagers
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$ █
```

- **start-dfs.sh**
  - Starts HDFS daemons:
    - **NameNode** (master, namespace manager)
    - **SecondaryNameNode** (checkpointing)
    - **DataNodes** (store actual blocks of data)
- **start-yarn.sh**
  - Starts YARN daemons:
    - **ResourceManager** (assigns resources to jobs)
    - **NodeManagers** (run tasks on worker nodes)

### 54.3 Why run them after stopping?

- Your configuration changes (like adding proxyuser.Asiema) are only applied when services are restarted.
  - This ensures HDFS + YARN are running with the new settings, ready for HiveServer2 and Beeline to connect.
- 

### 54.4 Layman's Analogy

Think of Hadoop as a **factory**:

- **HDFS (DFS)** = The **warehouse** where raw materials (data) are stored.
- **YARN** = The **floor manager** that decides which worker uses which machine.

When you update the rules of the factory (configs), you first **shut everything down** (stop-dfs.sh, stop-yarn.sh), then **reopen the factory** (start-dfs.sh, start-yarn.sh) so all staff follow the new rules.

---

## 54.5 Next Step After Hadoop Restart

1. Verify daemons are running:
2. jps

You should see:

- NameNode
- DataNode
- SecondaryNameNode
- ResourceManager
- NodeManager

3. Start Hive services:

4. hive --service hiveserver2 &

5. Connect via Beeline:

```
beeline -u jdbc:hive2://localhost:10000 -n Asiema
```

## 55 HiveServer2 startup step

You're at the **HiveServer2 startup step**, which is essential for using **Beeline** (and any JDBC/ODBC connection to Hive).

---

### Command:

```
cd $HIVE_HOME/bin
```

### 55.1 hiveserver2

- **hiveserver2** starts the **HiveServer2 daemon**.
  - It listens on a **JDBC port (default: 10000)**.
  - It's the “middleman” between clients (Beeline, BI tools like Tableau/PowerBI, JDBC apps) and the Hive execution layer (Driver, Metastore, YARN, HDFS).
  - HiveServer2 supports **multi-user sessions**, unlike the old `hive` CLI which was single-user.
- 

### 55.2 Why It's Necessary

- Beeline cannot connect directly to HDFS or the Metastore.
- Instead, it connects over JDBC to HiveServer2.
- HiveServer2 then:
  1. Authenticates the user (with proxy-user settings you configured earlier).
  2. Sends SQL queries to the **Driver**.

3. The Driver consults the **Metastore**.
  4. Executes queries on YARN/MapReduce/Tez.
  5. Returns results back to Beeline.
- 

### 55.2.1 Layman's Analogy

Think of HiveServer2 as the **front desk receptionist**  at a company:

- Clients (Beeline, BI tools) walk in and hand their requests to the receptionist.
- The receptionist logs the request, checks the rulebook (Metastore), and assigns tasks to the right department (YARN + HDFS).
- Then it hands the finished result back to the client.

Without the receptionist (HiveServer2), employees (Beeline clients) have no one to receive and route their requests.

---

## 55.3 Next Step: Connect via Beeline

Once HiveServer2 is running, open a new terminal and run:

```
beeline -u jdbc:hive2://localhost:10000 -n Asiema
```

Inside Beeline:

```
!info
```

```
show databases;
```

This should confirm that your Beeline client is successfully talking to HiveServer2.

---

### ⚠ Important:

- Keep HiveServer2 running in the background (or in a separate terminal tab).
- If it crashes or you close that session, Beeline will fail to connect.

```
Asiema@ProjectPro-P001:/disk/hadoop/hadoop-3.4.1/sbin$ cd $HIVE_HOME/bin; hiveserver2
2025-09-08 18:42:09: Starting HiveServer2
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/disk/hive/apache-hive-3.1.2-bin/lib/log4j-slf4j-impl-2.10.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/disk/hadoop/hadoop-3.4.1/share/hadoop/common/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.apache.logging.slf4j.Log4jLoggerFactory]
Hive Session ID = 4f43f95d-24f2-46ee-89ca-d0d0cfa646b6
Hive Session ID = 7fd4d9f9-c636-4b67-b911-d73fa22a366a
Hive Session ID = ebefc8b-5612-4668-94c6-879b7f646819
Hive Session ID = 70a954db-fb45-418c-9b7b-2ff452b692fe
Hive Session ID = dc2333be-992b-4981-ae37-21f9cd7c5eb5
Hive Session ID = c24d7acc-bd71-4a4c-96c9-ffaef4a0de17f
[redacted]
```

## 55.4 What the Output Means

- **Starting HiveServer2** → The HiveServer2 daemon is up and running.
- **Hive Session ID = ...** → Every time a client (like Beeline) connects, a new session ID is generated.
  - The list of session IDs you see means HiveServer2 has already spawned multiple connection contexts.
- **SLF4J warnings** (multiple bindings) → Just logging noise, safe to ignore.

This is exactly what you want to see when HiveServer2 is healthy.

---

## 55.5 Next Step: Connect via Beeline

Open a **new terminal** (don't close the one running HiveServer2). Then run:

```
beeline -u jdbc:hive2://localhost:10000 -n Asiema
```

If you're on the same VM and running locally, localhost works fine.

If you want to connect remotely from another machine, use the VM's IP instead of localhost.

## 55.6 Layman's Explanation

You've just opened the **reception desk (HiveServer2)** for your data warehouse.

- Each **session ID** is like a customer taking a ticket at the desk.
- Next, you'll walk in as a client (Beeline) and talk to the receptionist.
- The receptionist will then process your SQL queries and return answers.

## 56 Beeline connection step

You've now reached the **Beeline connection step**.

---

### Command:

```
beeline -u jdbc:hive2://localhost:10000 -n Asiema
```

- **beeline** → A lightweight JDBC client for Hive.
  - **-u jdbc:hive2://localhost:10000** → Tells Beeline to connect to HiveServer2 running locally on port **10000**.
  - **-n Asiema** → The username passed to HiveServer2 (you can replace this with your updated Asiema user).
- 

### 56.1 What Happens Here:

1. Beeline establishes a JDBC connection to HiveServer2.
  2. HiveServer2 verifies the user (thanks to the proxy-user settings in core-site.xml).
  3. Beeline drops you into an interactive SQL shell where you can run HiveQL commands.
-

## 56.2 Layman's Analogy

Think of **Beeline** as the **phone line**  that connects you (the client) to the **receptionist** (**HiveServer2**) at your data warehouse.

- You dial the number (jdbc:hive2://localhost:10000).
  - You say your name (-n Asiema or -n Asiema).
  - The receptionist (HiveServer2) picks up and asks:
- 

### What this step is doing (in layman's terms):

When Hive starts, it uses a **metastore** (a database where Hive stores metadata like table names, schemas, partitions, etc.).

If your setup is **incomplete, corrupted, or buggy**, you might see errors like:

- “Could not open connection to metastore”
  - “Table not found”
  - “Schema version mismatch”
- 

## 56.3 To fix that

You're instructed to **delete the metastore** and **reinitialize it** using Derby (the default lightweight DB).

---

## 56.4 Commands Breakdown

```
rm -rf $HIVE_HOME/metastore_db
```

✖ Deletes the existing metastore database (like resetting Hive's memory of tables)

```
$HIVE_HOME/bin/schematool -dbType derby -initSchema
```

❖ Recreates the metadata tables Hive needs to work.

This uses **Apache Derby** as the backend database. No need to install MySQL or PostgreSQL if you're just experimenting or testing locally.

---

### 🔴 Important Note:

Only run this if:

- You're using Derby (default local metastore)
  - You haven't created production tables you care about (this **will delete them**)
  - You're stuck during Hive startup
- 

### ▣ Follow-up Steps After This:

Once you run this and see success:

1. Restart HiveServer2:

2. \$HIVE\_HOME/bin/hiveserver2

3. Try Beeline again:

```
beeline -u jdbc:hive2://localhost:10000 -n Asiema
```

## 57 Create tables in Hive

Once Apache Hive is installed and configured on the Azure VM, the next step is to set up the environment to ingest and query the airline-related datasets. These datasets—flights, airports, carriers, and plane details—will be copied from the local machine to the Azure VM.

After transferring the data, we'll move them into HDFS and create external Hive tables referencing them. This process allows Hive to manage structured queries over large-scale datasets efficiently. You can get commands from the file (**DatabaseAndTables**) from the code folder to get all the commands to create tables in the Hive.

The following outlines the process we will follow to create tables in Hive and load data into them.

- a. Download Datasets
- b. Copy the Datasets to Azure VM
- c. Copy Files to HDFS
- d. Create Tables in the Hive

## 58 Step 1: Create Directories on Azure VM

### 58.1 cd /disk

- **What it does:** Navigate to the /disk directory on the VM, which we will use as the root directory to store the dataset files locally before uploading them to HDFS.

*mkdir data*

*cd data*

---

### 58.2 cd data

- **What it does:** Creates a data directory inside /disk and changes into it. This is where all subfolders (flights, airports, carriers, planes) will be created to organize different datasets.

*cd data*

*mkdir flights*

*mkdir airports*

*cd airports*

*mkdir csv*

*cd ..*

*mkdir carriers*

*cd carriers*

*mkdir csv*

*cd ..*

---

*mkdir planes*

*cd planes*

*mkdir csv*

*cd ..*

---

```
hive> Asiema@ProjectPro-P001:/disk/hive/apache-hive-3.1.2-bin/bin$
Asiema@ProjectPro-P001:/disk/hive/apache-hive-3.1.2-bin/bin$ cd /disk
Asiema@ProjectPro-P001:/disk$ mkdir data
Asiema@ProjectPro-P001:/disk$ cd data
Asiema@ProjectPro-P001:/disk/data$ mkdir flights
Asiema@ProjectPro-P001:/disk/data$ mkdir airports
Asiema@ProjectPro-P001:/disk/data$ mkdir csv
Asiema@ProjectPro-P001:/disk/data$ ls
airports csv flights
Asiema@ProjectPro-P001:/disk/data$ cd ..
Asiema@ProjectPro-P001:/disk$ ls
data hadoop hive
Asiema@ProjectPro-P001:/disk$ cd data
Asiema@ProjectPro-P001:/disk/data$ ls
airports csv flights
Asiema@ProjectPro-P001:/disk/data$ cd airports
Asiema@ProjectPro-P001:/disk/data/airports$ mkdir csv
Asiema@ProjectPro-P001:/disk/data/airports$ cd ..
Asiema@ProjectPro-P001:/disk/data$ mkdir carriers
Asiema@ProjectPro-P001:/disk/data$ cd carriers
Asiema@ProjectPro-P001:/disk/data/carriers$ mkdir csv
Asiema@ProjectPro-P001:/disk/data/carriers$ ls
csv
Asiema@ProjectPro-P001:/disk/data/carriers$ cd ..
Asiema@ProjectPro-P001:/disk/data$ ls
airports carriers csv flights
Asiema@ProjectPro-P001:/disk/data$ mkdir planes
Asiema@ProjectPro-P001:/disk/data$ cd planes
Asiema@ProjectPro-P001:/disk/data/planes$ mkdir csv
Asiema@ProjectPro-P001:/disk/data/planes$ ls
csv
Asiema@ProjectPro-P001:/disk/data/planes$ cd ..
Asiema@ProjectPro-P001:/disk/data$ ls
airports carriers csv flights planes
Asiema@ProjectPro-P001:/disk/data$
```

### 58.3 What it does:

- `mkdir flights, airports, carriers, planes`: These create separate folders for each dataset type in the `/disk/data` folder.
- The `csv` folder inside each is meant to store the actual CSV files after decompression, making the structure neat and organized.

#### 58.3.1 Layman's Explanation

Think of your VM like a **filing cabinet**:

1. You open the cabinet to a big drawer named `/disk`.
2. Inside, you add a **main folder** called `data` to hold all your work.
3. Then, you create **subfolders** like:
  - **flights** → for all flight information
  - **airports** → for details about airports
  - **carriers** → for airline names and codes
  - **planes** → aircraft model info
4. Inside each of these, you make a **csv folder**, like a pocket, to hold the real files (after unpacking them).

### Why?

To keep everything **organized, clean, and easy to find** when you start uploading to Hadoop or querying in Hive.

## 59 Step 2: Copy Local Files to Azure VM Using SCP

From your **local machine terminal**, run:

```
scp /path/to/local/DataExpo2009.zip Asiema@azure-vm-ip:/disk/data/flights/
```

- scp: Secure copy command to transfer files via SSH.
- Replace /path/to/local/DataExpo2009.zip with your real file path.
- Replace Asiema@azure-vm-ip with your actual Azure VM credentials.
- /disk/data/flights/: Destination path on your Azure VM.

---

```
PS C:\Users\casie> scp "C:/Users/casie/OneDrive/Documents/02-Coding/05. Azure Roadmap Elevating Your Cloud Skills from N
ovice to Pro/01-Data Processing and Transformation in Hive using Azure VM [Easy]/airports.csv"
isk\data\airports\csv/
 password: 100% 239KB 716.8KB/s 00:00
airports.csv
PS C:\Users\casie> scp "C:/Users/casie/OneDrive/Documents/02-Coding/05. Azure Roadmap Elevating Your Cloud Skills from N
ovice to Pro/01-Data Processing and Transformation in Hive using Azure VM [Easy]/airports.csv"
isk\data\airports\csv/
 password: 100% 239KB 589.4KB/s 00:00
airports.csv
PS C:\Users\casie>
PS C:\Users\casie> scp "C:/Users/casie/OneDrive/Documents/02-Coding/05. Azure Roadmap Elevating Your Cloud Skills from N
ovice to Pro/01-Data Processing and Transformation in Hive using Azure VM [Easy]/carriers.csv"
isk\data\carriers\csv/
 password: 100% 43KB 192.5KB/s 00:00
carriers.csv
PS C:\Users\casie>
PS C:\Users\casie> scp "C:/Users/casie/OneDrive/Documents/02-Coding/05. Azure Roadmap Elevating Your Cloud Skills from N
ovice to Pro/01-Data Processing and Transformation in Hive using Azure VM [Easy]/plane-data.csv"
/disk/data/planes\csv/
 password: 100% 419KB 811.5KB/s 00:00
plane-data.csv
PS C:\Users\casie> |
```

### 59.1 Layman's Explanation:

Think of this like **emailing yourself a zipped folder** of data - but instead of email, you're using a secure copy command (scp).

- You tell your computer:

“Hey, send this zipped file with flight data to the special ‘flights’ folder inside the Azure virtual machine.”
- It’s like uploading a ZIP file into the correct folder in your cloud workspace.

Once copied, you’ll later unzip it and extract the actual spreadsheet files into the folders you made earlier.

```
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
Asiema@ProjectPro-P001:/disk/data$ tree
.
├── airports
│ └── csv
│ └── airports.csv
├── carriers
│ └── csv
│ └── carriers.csv
├── csv
└── flights
 └── planes
 └── csv
 └── plane-data.csv

9 directories, 3 files
Asiema@ProjectPro-P001:/disk/data$ █
```

## 60 Step 3: Unzip and Decompress Files on Azure VM

### Technical Breakdown:

```
cd /disk/data/flights # Move to the flights folder

sudo apt-get install unzip # Install unzip utility (if missing)

unzip DataExpo2009.zip # Extract contents of the ZIP archive
```

## 60.1 What It Does:

- Changes the current directory to /disk/data/flights
- Installs the unzip utility (needed to extract .zip files)
- Extracts DataExpo2009.zip into .bz2 files like:
  - 2005.csv.bz2
  - 2006.csv.bz2
  - and so on...

---

```
No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
Asiema@ProjectPro-P001:/disk/data/flights$ unzip Data.zip
Archive: Data.zip
 creating: Data/
 inflating: Data/1987.csv.bz2
 inflating: Data/1988.csv.bz2
 inflating: Data/1989.csv.bz2
 inflating: Data/1990.csv.bz2
 inflating: Data/1991.csv.bz2
 inflating: Data/1992.csv.bz2
 inflating: Data/1993.csv.bz2
 inflating: Data/1994.csv.bz2
 inflating: Data/1995.csv.bz2
 inflating: Data/1996.csv.bz2
 inflating: Data/1997.csv.bz2
 inflating: Data/1998.csv.bz2
 inflating: Data/1999.csv.bz2
 inflating: Data/2000.csv.bz2
 inflating: Data/2001.csv.bz2
 inflating: Data/2002.csv.bz2
 inflating: Data/2003.csv.bz2
 inflating: Data/2004.csv.bz2
 inflating: Data/2005.csv.bz2
 inflating: Data/2006.csv.bz2
 inflating: Data/2007.csv.bz2
 inflating: Data/2008.csv.bz2
 inflating: Data/2009. Data expo.. ASA Statistics Computing and Graphics.html
 creating: Data/2009. Data expo.. ASA Statistics Computing and Graphics_files/
 inflating: Data/2009. Data expo.. ASA Statistics Computing and Graphics_files/seal1.gif
 inflating: Data/2009. Data expo.. ASA Statistics Computing and Graphics_files/style.css
 inflating: Data/2009. Data expo.. ASA Statistics Computing and Graphics_files/urchin.js
 inflating: Data/The data. Data expo 09. ASA Statistics Computing and Graphics.html
 creating: Data/The data. Data expo 09. ASA Statistics Computing and Graphics_files/
 inflating: Data/The data. Data expo 09. ASA Statistics Computing and Graphics_files/seal1.g
if
 inflating: Data/The data. Data expo 09. ASA Statistics Computing and Graphics_files/style.c
ss
 inflating: Data/The data. Data expo 09. ASA Statistics Computing and Graphics_files/urchin.
js
Asiema@ProjectPro-P001:/disk/data/flights$
```

### 60.1.1 Layman's Explanation:

Imagine you moved a **Ziploc bag** called DataExpo2009.zip into the **flights drawer**.

This step:

1. **Opens that drawer** (cd /disk/data/flights)
2. **Makes sure you have scissors** (sudo apt-get install unzip)
3. **Unzips the bag** to reveal many small bags (like 2005.csv.bz2, 2006.csv.bz2, etc.) containing the actual data.

## 61 Step 4: Copy .bz2 Files (1987–2004) to csv Folder for Decompression

```
mkdir csv
```

```
cd Data/
```

```
cp 2004.csv.bz2 ..//csv/
cp 2003.csv.bz2 ..//csv/
cp 2002.csv.bz2 ..//csv/
cp 2001.csv.bz2 ..//csv/
cp 2000.csv.bz2 ..//csv/
cp 1999.csv.bz2 ..//csv/
cp 1998.csv.bz2 ..//csv/
cp 1997.csv.bz2 ..//csv/
cp 1996.csv.bz2 ..//csv/
cp 1995.csv.bz2 ..//csv/
cp 1994.csv.bz2 ..//csv/
cp 1993.csv.bz2 ..//csv/
cp 1992.csv.bz2 ..//csv/
cp 1991.csv.bz2 ..//csv/
cp 1990.csv.bz2 ..//csv/
cp 1989.csv.bz2 ..//csv/
cp 1988.csv.bz2 ..//csv/
cp 1987.csv.bz2 ..//csv/
```

---

```
Asiema@ProjectPro-P001:/disk/data/flights/csv$ bzip2 -d 2008.csv.bz2
Asiema@ProjectPro-P001:/disk/data/flights/csv$ tree
.
├── 1987.csv.bz2
├── 1988.csv.bz2
├── 1989.csv.bz2
├── 1990.csv.bz2
├── 1991.csv.bz2
├── 1992.csv.bz2
├── 1993.csv.bz2
├── 1994.csv.bz2
├── 1995.csv.bz2
├── 1996.csv.bz2
├── 1997.csv.bz2
├── 1998.csv.bz2
├── 1999.csv.bz2
├── 2000.csv.bz2
├── 2001.csv.bz2
├── 2002.csv.bz2
├── 2003.csv.bz2
└── 2004.csv.bz2
2 directories
```

### ● What it does:

- Creates a csv directory to organize the flight .bz2 files
- Copies all .bz2 files from **1987 to 2004** from DataExpo2009/ to the csv/ folder for easy decompression

## 62 Step 5: Decompress All .bz2 Files into .csv Format

```
cd ../csv/
```

```
sudo apt install bzip2
```

```
bzip2 -d 1987.csv.bz2
```

```
bzip2 -d 1998.csv.bz2
```

```
bzip2 -d 1988.csv.bz2
```

```
bzip2 -d 1999.csv.bz2
```

```
bzip2 -d 1989.csv.bz2
```

```
bzip2 -d 2000.csv.bz2
```

```
bzip2 -d 1990.csv.bz2
```

```
bzip2 -d 2001.csv.bz2
```

```
bzip2 -d 1991.csv.bz2
```

```
bzip2 -d 2002.csv.bz2
```

```
bzip2 -d 1992.csv.bz2
```

```
bzip2 -d 2003.csv.bz2
```

```
bzip2 -d 1993.csv.bz2
```

```
bzip2 -d 2004.csv.bz2
```

```
bzip2 -d 1994.csv.bz2
```

```
bzip2 -d 2005.csv.bz2
```

```
bzip2 -d 1995.csv.bz2
```

```
bzip2 -d 2006.csv.bz2
```

```
bzip2 -d 1996.csv.bz2
```

```
bzip2 -d 2007.csv.bz2
```

---

```
bzip2 -d 1997.csv.bz2
```

```
bzip2 -d 2008.csv.bz2
```

```
Asiema@ProjectPro-P001:/disk/data/flights/csv$ tree
.
├── 1987.csv
├── 1988.csv
├── 1989.csv
├── 1990.csv
├── 1991.csv
├── 1992.csv
├── 1993.csv
├── 1994.csv
├── 1995.csv
├── 1996.csv
├── 1997.csv
├── 1998.csv
├── 1999.csv
├── 2000.csv
├── 2001.csv
├── 2002.csv
├── 2003.csv
├── 2004.csv
├── 2005.csv
├── 2006.csv
├── 2007.csv
└── 2008.csv
```

● **What it does:**

- ○ Installs the bzip2 decompression tool if not already installed.
- ○ Decompresses all .bz2 files into .csv format, so Hive can read them later.

## 63 STEP 6: Move CSV Files (1987–2008) to HDFS as Asiema

### 1: Create HDFS target directory

```
hdfs dfs -mkdir -p /user/Asiema/data/flights/csv
```

### 2: Set write permissions on the directory

```
hdfs dfs -chmod g+w /user/Asiema/data/flights/csv
```

### 3: Move all CSV files from 1987 to 2008 to HDFS

```
for year in {1987..2008}; do
```

```
 hdfs dfs -moveFromLocal /disk/data/flights/csv/${year}.csv
 /user/Asiema/data/flights/csv
```

```
done
```

---

```
Asiema@ProjectPro-P001:/disk/data/flights/csv$ cd Data
-bash: cd: Data: No such file or directory
Asiema@ProjectPro-P001:/disk/data/flights/csv$ cd ..
Asiema@ProjectPro-P001:/disk/data/flights$ ls
Data Data.zip csv
Asiema@ProjectPro-P001:/disk/data/flights$ cd data
-bash: cd: data: No such file or directory
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -mkdir -p /user/Asiema/data/flights/csv
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -chmod g+w /user/Asiema/data/flights/csv
Asiema@ProjectPro-P001:/disk/data/flights$ for year in {1987..2008}; do
 hdfs dfs -moveFromLocal /disk/data/flights/csv/${year}.csv /user/Asiema/data/flights/csv
done
Asiema@ProjectPro-P001:/disk/data/flights$
```

### 63.1.1 Breakdown

Command	Purpose
<code>hdfs dfs -mkdir -p</code>	Creates the full directory path for the CSV files in HDFS (including any parent folders).
<code>hdfs dfs -chmod g+w</code>	Grants write permissions to the group, which helps Hive or other users modify the data.
<code>hdfs dfs -moveFromLocal</code>	Moves each year's CSV file from local to HDFS and <b>removes the original</b> after upload.

## 63.2 Airports Data

1. `hdfs dfs -mkdir -p /user/Asiema/data/airports/csv`
2. `hdfs dfs -chmod g+w /user/Asiema/data/airports/csv`
3. `hdfs dfs -moveFromLocal /disk/data/airports/csv/airports.csv`  
`/user/Asiema/data/airports/csv`
- 4.

```
Asiema@ProjectPro-P001:/disk/data/airports$ hdfs dfs -mkdir -p /user/Asiema/data/airports/csv
Asiema@ProjectPro-P001:/disk/data/airports$ hdfs dfs -chmod g+w /user/Asiema/data/airports/csv
Asiema@ProjectPro-P001:/disk/data/airports$ for year in {1987..2008}; do
 hdfs dfs -moveFromLocal /disk/data/airports/csv/${year}.csv /user/Asiema/data/airports/csv
done
Asiema@ProjectPro-P001:/disk/data/airports$ hdfs dfs -mkdir -p /user/Asiema/data/airports/csv
Asiema@ProjectPro-P001:/disk/data/airports$ hdfs dfs -chmod g+w /user/Asiema/data/airports/csv
Asiema@ProjectPro-P001:/disk/data/airports$ hdfs dfs -moveFromLocal /disk/data/airports/csv/airports.csv /user/Asiema/data/airports/csv
Asiema@ProjectPro-P001:/disk/data/airports$
```

### 63.3 Carriers Data

```
1. hdfs dfs -mkdir -p /user/Asiema/data/carriers/csv
2. hdfs dfs -chmod g+w /user/Asiema/data/carriers/csv
3. hdfs dfs -moveFromLocal /disk/data/carriers/csv/carriers.csv
/user/Asiema/data/carriers/csv
4.
done
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -mkdir -p /user/Asiema/data/airports/csv
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -chmod g+w /user/Asiema/data/airports/csv
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -moveFromLocal /disk/data/airports/csv/ai
rports.csv /user/Asiema/data/airports/csv
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -mkdir -p /user/Asiema/data/carriers/csv
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -chmod g+w /user/Asiema/data/carriers/csv
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -moveFromLocal /disk/data/carriers/csv/ca
rriers.csv /user/Asiema/data/carriers/csv
Asiema@ProjectPro-P001:/disk/data/flights$
```

---

### 63.4 Planes Data

hdfs dfs -mkdir -p /user/Asiema/data/planes/csv

hdfs dfs -chmod g+w /user/Asiema/data/planes/csv

hdfs dfs -moveFromLocal /disk/data/planes/csv/planedata.csv /user/Asiema/data/planes/csv

```
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -mkdir -p /user/Asiema/data/airports/csv
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -chmod g+w /user/Asiema/data/airports/csv
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -moveFromLocal /disk/data/airports/csv/ai
rports.csv /user/Asiema/data/airports/csv
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -mkdir -p /user/Asiema/data/carriers/csv
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -chmod g+w /user/Asiema/data/carriers/csv
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -moveFromLocal /disk/data/carriers/csv/ca
rriers.csv /user/Asiema/data/carriers/csv
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -mkdir -p /user/Asiema/data/planes/csv
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -chmod g+w /user/Asiema/data/planes/csv
Asiema@ProjectPro-P001:/disk/data/flights$ hdfs dfs -moveFromLocal /disk/data/planes/csv/plan
e-data.csv /user/Asiema/data/planes/csv
Asiema@ProjectPro-P001:/disk/data/flights$
```

---

## 63.5 Summary

Dataset	HDFS Target Directory	Local Source
Airports	/user/Asiema/data/airports/csv	/disk/data/airports/csv/airports.csv
Carriers	/user/Asiema/data/carriers/csv	/disk/data/carriers/csv/carriers.csv
Planes	/user/Asiema/data/planes/csv	/disk/data/planes/csv/plane-data.csv

## 64 Step 5: Create Hive Tables

From your Hive installation folder, run:

```
cd $HIVE_HOME/bin
```

```
hive
```

### 64.1 What it does:

- **cd \$HIVE\_HOME/bin** = Navigates to the Hive bin directory, where the Hive executable files live.
  - **hive** = Launches the Hive Command Line Interface (CLI) so you can run HiveQL commands.
- 

Now, create a database and start using it:

```
create database airline location
'/user/Asiema/hive/warehouse/airline.db';

use airline;
```

```
Logging initialized using configuration in jar:file:/disk/hive/apache-hive-3.1.2-bin/lib/hive
-common-3.1.2.jar!/hive-log4j2.properties Async: true
Hive Session ID = e8e95d2d-201f-4c7f-a96b-923e25cd4d0b
Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider
using a different execution engine (i.e. spark, tez) or using Hive 1.X releases.
hive> create database airline location '/user/Asiema/hive/warehouse/airline.db';
OK
Time taken: 0.542 seconds
hive> █
```

## 64.2 What it does:

- Create database **airline** location ...; → Creates a Hive database named **airline**, with its data stored in the specified HDFS path (/user/Asiema/hive/warehouse/airline.db).
- **use airline;** = Switches the current working database to **airline**, so any new tables or queries will apply here.

```
hive> create database airline location '/user/Asiema/hive/warehouse/airline.db';
OK
Time taken: 0.542 seconds
hive> use airline
 > ;
OK
Time taken: 0.025 seconds
hive>
```

---

## 64.3 Layman's Analogy:

- **cd \$HIVE\_HOME/bin** = Walking into the Hive control room.
- **hive** = Sitting down at the Hive operator's desk, ready to give instructions.
- **Create database **airline** ...** = Setting up a new section in the big data library called **airline** and telling Hive where to keep its books.
- **use **airline**;** = Moving into that section so every book (table) you add now goes into **airline**.

## 65 Step 6: Create Hive External Tables

### 65.1 Create the flights table:

```
1. create external table flights (year smallint,month tinyint,dayofmonth
tinyint,dayofweek tinyint, deptime smallint, crsdeptime smallint, arrtime
smallint, crsarrrtime smallint, uniquecarrier string, flightnum string, tailnum
string, actualelapsedtime smallint,crselapsedtime smallint, airtime smallint,
arrdelay smallint, depdelay smallint, origin string, dest string, distance
smallint, taxiin string, taxiout string,cancelled string, cancellationcode string,
diverted string, carrierdelay smallint,weatherdelay smallint, nasdelay smallint,
securitydelay smallint, lateaircraftdelay smallint)
2. row format delimited fields terminated by ','
3. location '/user/Asiema/data/flights/csv';
4.
```

```
43) at java.lang.reflect.Method.invoke(Method.java:498)
 at org.apache.hadoop.util.RunJar.run(RunJar.java:330)
 at org.apache.hadoop.util.RunJar.main(RunJar.java:245)
FAILED: ParseException line 1:31 cannot recognize input near ')' 'create' 'external' in column
n name or constraint
hive> CREATE EXTERNAL TABLE flights (
 > year SMALLINT,
 > month TINYINT,
 > dayofmonth TINYINT,
 > dayofweek TINYINT,
 > deptime SMALLINT,
 > crsdeptime SMALLINT,
 > arrtime SMALLINT,
 > crsarrrtime SMALLINT,
 > uniquecarrier STRING,
 > flightnum STRING,
 > tailnum STRING,
 > actualelapsedtime SMALLINT,
 > crselapsedtime SMALLINT,
 > airtime SMALLINT,
 > arrdelay SMALLINT,
 > depdelay SMALLINT,
 > origin STRING,
 > dest STRING,
 > distance SMALLINT,
 > taxiin SMALLINT,
 > taxiout SMALLINT,
 > cancelled STRING,
 > cancellationcode STRING,
 > diverted STRING,
 > carrierdelay SMALLINT,
 > weatherdelay SMALLINT,
 > nasdelay SMALLINT,
 > securitydelay SMALLINT,
 > lateaircraftdelay SMALLINT
 >)
 > ROW FORMAT DELIMITED
 > FIELDS TERMINATED BY ','
 > STORED AS TEXTFILE
 > LOCATION '/user/Asiema/data/flights/csv';
OK
Time taken: 0.362 seconds
hive>
```

### 65.1.1 What it does:

- create external table flights (...) → Creates a table called **flights**, but as an **external table**. This means Hive only points to the data - it doesn't own or manage the files.
  - row format delimited fields terminated by ',' → Tells Hive the file format is CSV-like, with each field separated by a comma.
  - location '/user/Asiema/data/flights/csv'; → Specifies the folder in HDFS where the flight CSV files are stored. Hive just maps to this data.
- 

### 65.1.2 Important Notes:

- External tables are useful when your data already exists in HDFS. Dropping the table won't delete the data files.
  - Hive uses **OpenCSVSerde** to correctly parse CSV files (including quoted strings like "New York").
- 

### Layman's Analogy:

Think of Hive as a librarian again:

- create external table flights (...) = Instead of moving the books into a new shelf, you just give the librarian a catalog of where the books already are in the warehouse.
- row format ... = You tell the librarian how the books are organized (e.g., chapters separated by commas).

- location ... = You give the librarian the exact aisle where those books are kept.

So Hive doesn't "own" these books - it just knows how to find and read them.

## 65.2 Create Hive External Table for Airports

```
CREATE EXTERNAL TABLE airports (
 airport_id INT,
 name STRING,
 city STRING,
 country STRING,
 iata STRING,
 icao STRING,
 latitude DOUBLE,
 longitude DOUBLE,
 altitude INT,
 timezone STRING,
 dst STRING,
 tz_database_timezone STRING
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
 "separatorChar" = ",",
 "quoteChar" = "\",
 "escapeChar" = "\\"
)
STORED AS TEXTFILE
LOCATION '/user/Asiema/data/airports/csv';
```

```
> year INT
>)
> ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
> WITH SERDEPROPERTIES (
> "separatorChar" = ",",
> "quoteChar" = "\"",
> "escapeChar" = "\\"
>)
> STORED AS TEXTFILE
> LOCATION '/user/Asiema/data/planes/csv';
OK
Time taken: 0.081 seconds
hive> DROP TABLE IF EXISTS airports;
OK
Time taken: 0.409 seconds
hive> CREATE EXTERNAL TABLE airports (
> airport_id INT,
> name STRING,
> city STRING,
> country STRING,
> iata STRING,
> icao STRING,
> latitude DOUBLE,
> longitude DOUBLE,
> altitude INT,
> timezone STRING,
> dst STRING,
> tz_database_timezone STRING
>)
> ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
> WITH SERDEPROPERTIES (
> "separatorChar" = ",",
> "quoteChar" = "\"",
> "escapeChar" = "\\"
>)
> STORED AS TEXTFILE
> LOCATION '/user/Asiema/data/airports/csv';
OK
Time taken: 0.099 seconds
hive>
```

### 65.2.1      **What it does:**

- CREATE EXTERNAL TABLE airports (...) → Creates a table that maps to your airports CSV data.
  - ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' → Tells Hive to use **OpenCSVSerde**, a special parser that can handle complex CSVs with **quotes** (e.g. "New York, NY") and escape characters.
  - WITH SERDEPROPERTIES (...) → Defines how the CSV is structured:
    - Separator = , (fields are separated by commas).
    - Quote character = " (quoted values allowed).
    - Escape character = \ (handles special symbols safely).
  - STORED AS TEXTFILE → Treats the file as plain text CSV.
  - LOCATION '/user/Asiema/data/airports/csv' → Points Hive to where the airports dataset is stored in HDFS.
- 

### Layman's Analogy:

Imagine your CSV is **messy** - some airport names might have commas (like "Los Angeles, Intl"). A normal librarian (Hive) would get confused by those commas.

So, you hire a **special librarian assistant (OpenCSVSerde)** who understands quotes and escapes.

- quoteChar = tells the assistant: "Anything inside quotes belongs together."

- `escapeChar =` says: “If you see a backslash, don’t get confused as it means the next character is special.”

This way, even tricky values like "San José, Costa Rica" won’t break your table.

### 65.3 Step X: Create Hive External Table for Carriers

```
1. CREATE EXTERNAL TABLE carriers (
2. code STRING,
3. description STRING
4.)
5. ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
6. WITH SERDEPROPERTIES (
7. "separatorChar" = ",",
8. "quoteChar" = "\"",
9. "escapeChar" = "\\"
10.)
11. STORED AS TEXTFILE
12. LOCATION '/user/Asiema/data/carriers/csv';
13.
```

```
> dst STRING,
> tz_database_timezone STRING
>)
> ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
> WITH SERDEPROPERTIES (
> "separatorChar" = ",",
> "quoteChar" = "\"",
> "escapeChar" = "\\"
>)
> STORED AS TEXTFILE
> LOCATION '/user/ubuntuhive/data/airports/csv';
OK
Time taken: 0.097 seconds
hive> CREATE EXTERNAL TABLE carriers (
> code STRING,
> description STRING
>)
> ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
> WITH SERDEPROPERTIES (
> "separatorChar" = ",",
> "quoteChar" = "\"",
> "escapeChar" = "\\"
>)
> STORED AS TEXTFILE
> LOCATION '/user/Asiema/data/carriers/csv';
OK
Time taken: 0.087 seconds
hive> []
```

#### 65.3.1 What it does:

- CREATE EXTERNAL TABLE carriers (...) → Creates an external table called **carriers** that points to your CSV file.

- code STRING → Stores the airline carrier code (e.g., "AA" for American Airlines).
  - description STRING → Stores the full airline name (e.g., "American Airlines").
  - ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde' → Uses **OpenCSVSerde** to properly handle quoted strings and special characters.
  - WITH SERDEPROPERTIES (...) → Ensures Hive reads CSV files correctly:
    - Commas separate values.
    - Quotes handle names with commas inside.
    - Backslashes act as escape characters.
  - STORED AS TEXTFILE → Treats the CSV file as plain text.
  - LOCATION '/user/Asiema/data/carriers/csv' → Tells Hive where the carriers dataset is stored in HDFS.
- 

### Layman's Analogy:

Think of the carriers file as a **glossary**:

- Each entry has a short code (AA) and its meaning (American Airlines).
- Hive is just creating a **dictionary index** for this glossary, so whenever you ask, “What’s UA?”, Hive can look it up without changing the original file.

## 65.4 Create Hive External Table for Planes

```
1. CREATE EXTERNAL TABLE planes (
2. tailnum STRING,
3. type STRING,
4. manufacturer STRING,
5. issue_date STRING,
6. model STRING,
7. status STRING,
8. aircraft_type STRING,
9. engine_type STRING,
10. year INT
11.)
12. ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
13. WITH SERDEPROPERTIES (
14. "separatorChar" = ",",
15. "quoteChar" = "\"",
16. "escapeChar" = "\\"
17.)
18. STORED AS TEXTFILE
19. LOCATION '/user/Asiema/data/planes/csv';
20.
```

```

OK
Time taken: 0.097 seconds
hive> CREATE EXTERNAL TABLE carriers (
 > code STRING,
 > description STRING
 >)
 > ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
 > WITH SERDEPROPERTIES (
 > "separatorChar" = ",",
 > "quoteChar" = "\"",
 > "escapeChar" = "\\"
 >)
 > STORED AS TEXTFILE
 > LOCATION '/user/Asiema/data/carriers/csv';
OK
Time taken: 0.087 seconds
hive> CREATE EXTERNAL TABLE planes (
 > tailnum STRING,
 > type STRING,
 > manufacturer STRING,
 > issue_date STRING,
 > model STRING,
 > status STRING,
 > aircraft_type STRING,
 > engine_type STRING,
 > year INT
 >)
 > ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
 > WITH SERDEPROPERTIES (
 > "separatorChar" = ",",
 > "quoteChar" = "\"",
 > "escapeChar" = "\\"
 >)
 > STORED AS TEXTFILE
 > LOCATION '/user/Asiema/data/planes/csv';
OK
Time taken: 0.081 seconds
hive>
```

#### 65.4.1 What it does:

- `CREATE EXTERNAL TABLE planes (...)` → Creates an external Hive table that maps to the **planes dataset**.
- The columns (`tailnum`, `type`, `manufacturer`, etc.) match the CSV file's structure.
- `ROW FORMAT SERDE '...OpenCSVSerde'` → Uses OpenCSVSerde to correctly parse CSV data, including special characters and quoted values.
- `WITH SERDEPROPERTIES (...)` → Ensures proper CSV parsing (commas, quotes, escapes).

- STORED AS TEXTFILE → Reads the CSV as plain text.
  - LOCATION '/user/Asiema/data/planes/csv' → Points Hive to where your planes CSV data is stored in HDFS.
- 

### Layman's Analogy:

Think of this as building a **catalog for aircrafts**:

- Each row in the CSV is a plane.
- Columns are details about the plane: its tail number (like a license plate), manufacturer (Boeing, Airbus), model, engine type, and year built.
- Instead of copying this list into Hive, you simply **point Hive to the existing file**, and Hive learns how to read it.

## 65.5 Step 7: Query Hive Tables to Validate Data

Run the following queries in Hive:

```
1. SELECT * FROM flights LIMIT 5;
2.
3. SELECT * FROM airports LIMIT 5;
4.
5. SELECT * FROM carriers LIMIT 5;
6.
7. SELECT * FROM planes LIMIT 5;
8.
```

### 65.5.1 What it does:

- Each query retrieves the **first 5 rows** from the respective table.
- Confirms that:
  - The table schema matches the CSV structure.
  - Hive is correctly parsing the data (e.g., numbers in numeric columns, strings in text columns).
  - The external tables are pointing to the correct HDFS locations.

---

### Layman's Analogy:

Imagine you just set up 4 new **catalogs** in a library (flights, airports, carriers, planes).

Before trusting them, you pull out the **first 5 sample books** from each catalog just to make sure:

- The catalog is pointing to the right shelf.

- The books are readable and properly labeled.
- Nothing is missing or corrupted.

## 65.6 Errors Returned

```
hive> SELECT * FROM flights LIMIT 5;
OK
NULL NULL NULL NULL NULL NULL NULL NULL UniqueCarrier FlightNum T
ailNum NULL NULL NULL NULL NULL Origin Dest NULL NULL NULL NULL Cance
lled CancellationCode Diverted NULL NULL NULL NULL NULL NULL NULL NULL NULL
1987 10 14 3 741 730 912 849 PS 1451 NA 91 7
9 NULL 23 11 SAN SFO 447 NULL NULL 0 NA 0 N
ULL NULL NULL
1987 10 15 4 729 730 903 849 PS 1451 NA 94 7
9 NULL 14 -1 SAN SFO 447 NULL NULL 0 NA 0 N
ULL NULL NULL
1987 10 17 6 741 730 918 849 PS 1451 NA 97 7
9 NULL 29 11 SAN SFO 447 NULL NULL 0 NA 0 N
ULL NULL NULL
1987 10 18 7 729 730 847 849 PS 1451 NA 78 7
9 NULL -2 -1 SAN SFO 447 NULL NULL 0 NA 0 N
ULL NULL NULL
Time taken: 1.109 seconds, Fetched: 5 row(s)
```

### 65.6.1 The Problem

- Many columns are coming back as NULL.
- Only some values (year, month, dayofmonth, dayofweek, etc.) appear correctly.
- This usually happens when:
  1. **CSV column order doesn't match your Hive schema** meaning Hive maps by column position, not by name.
  2. The **CSV has headers** (first row = column names), and Hive tries to read them as data.
  3. Some columns have values that don't match the declared datatype (e.g., NA in a numeric column).

## 65.6.2 How to Fix It

### 1. Check your CSV structure

- Look at the **first few lines** of /user/Asiema/data/flights/csv/1987.csv.
- Does it have a **header row** (column names)? If yes → you'll need to tell Hive to skip it.

**Example (first line of the CSV):**

Year,Month,DayofMonth,DayOfWeek,DepTime,CRSDepTime,ArrTi  
me,...

### 2. Update Hive table to handle headers (if present)

Add this property to your table definition:

### 3. TBLPROPERTIES ("skip.header.line.count"="1");

**Example fix for flights table:**

```
1. CREATE EXTERNAL TABLE flights (
2. year SMALLINT,
3. month TINYINT,
4. dayofmonth TINYINT,
5. dayofweek TINYINT,
6. deptime STRING,
7. crsdeptime STRING,
8. arrtime STRING,
9. crsarrrtime STRING,
10. uniquecarrier STRING,
11. flightnum STRING,
12. tailnum STRING,
13. actualelapsedtime STRING,
14. crselapsedtime STRING,
15. airtime STRING,
16. arrdelay STRING,
17. depdelay STRING,
```

```
18. origin STRING,
19. dest STRING,
20. distance STRING,
21. taxiin STRING,
22. taxiout STRING,
23. cancelled STRING,
24. cancellationcode STRING,
25. diverted STRING,
26. carrierdelay STRING,
27. weatherdelay STRING,
28. nasdelay STRING,
29. securitydelay STRING,
30. lateaircraftdelay STRING
31.)
32. ROW FORMAT DELIMITED
33. FIELDS TERMINATED BY ','
34. STORED AS TEXTFILE
35. LOCATION '/user/Asiema/data/flights/csv'
36. TBLPROPERTIES ("skip.header.line.count"="1");
37.
```

Notice I changed numeric columns like deptime, arrtime, and delay columns to STRING this prevents Hive from choking on NA values. Later you can cast them to numbers when needed.

#### 4. Re-run validation

```
1. SELECT year, month, dayofmonth, uniquecarrier, flightnum, origin, dest
2. FROM flights
3. LIMIT 10;
4.
```

This will show whether data is now correctly parsed.

```
> SELECT * FROM flights;
OK
1987 10 14 PS 1451 SAN SFO
1987 10 15 PS 1451 SAN SFO
1987 10 17 PS 1451 SAN SFO
1987 10 18 PS 1451 SAN SFO
1987 10 19 PS 1451 SAN SFO
1987 10 21 PS 1451 SAN SFO
1987 10 22 PS 1451 SAN SFO
1987 10 23 PS 1451 SAN SFO
1987 10 24 PS 1451 SAN SFO
1987 10 25 PS 1451 SAN SFO
Time taken: 0.182 seconds, Fetched: 10 row(s)
hive> █
```

---

### Layman's Explanation:

Right now Hive is reading your CSV **out of sync** with the schema:

- It's expecting numbers where the file has text (NA).
- It's reading the header row as if it were data.

That's like trying to read a table of contents as if it were a chapter - the pages don't line up.

By skipping the header and being more flexible with datatypes (STRING), Hive will align everything properly.

### Queries You Ran

`SELECT count(*) FROM flights;`

`SELECT count(*) FROM flights WHERE year = 2008;`

---

```

2025-09-09 23:31:49,766 Stage-1 map = 17%, reduce = 0%, Cumulative CPU 39.8 sec
2025-09-09 23:31:59,131 Stage-1 map = 21%, reduce = 0%, Cumulative CPU 50.18 sec
2025-09-09 23:32:02,272 Stage-1 map = 26%, reduce = 0%, Cumulative CPU 61.06 sec
2025-09-09 23:32:03,300 Stage-1 map = 29%, reduce = 0%, Cumulative CPU 65.93 sec
2025-09-09 23:32:04,334 Stage-1 map = 31%, reduce = 0%, Cumulative CPU 70.59 sec
2025-09-09 23:32:11,587 Stage-1 map = 33%, reduce = 0%, Cumulative CPU 75.49 sec
2025-09-09 23:32:15,729 Stage-1 map = 36%, reduce = 0%, Cumulative CPU 80.43 sec
2025-09-09 23:32:16,770 Stage-1 map = 38%, reduce = 12%, Cumulative CPU 85.78 sec
2025-09-09 23:32:17,802 Stage-1 map = 43%, reduce = 12%, Cumulative CPU 95.86 sec
2025-09-09 23:32:20,905 Stage-1 map = 45%, reduce = 12%, Cumulative CPU 100.15 sec
2025-09-09 23:32:22,980 Stage-1 map = 45%, reduce = 15%, Cumulative CPU 100.22 sec
2025-09-09 23:32:27,090 Stage-1 map = 48%, reduce = 15%, Cumulative CPU 104.28 sec
2025-09-09 23:32:29,170 Stage-1 map = 50%, reduce = 16%, Cumulative CPU 109.02 sec
2025-09-09 23:32:30,209 Stage-1 map = 55%, reduce = 16%, Cumulative CPU 119.16 sec
2025-09-09 23:32:32,291 Stage-1 map = 57%, reduce = 16%, Cumulative CPU 124.26 sec
2025-09-09 23:32:35,392 Stage-1 map = 57%, reduce = 19%, Cumulative CPU 124.32 sec
2025-09-09 23:32:37,447 Stage-1 map = 60%, reduce = 19%, Cumulative CPU 129.16 sec
2025-09-09 23:32:40,535 Stage-1 map = 62%, reduce = 19%, Cumulative CPU 133.9 sec
2025-09-09 23:32:41,565 Stage-1 map = 62%, reduce = 20%, Cumulative CPU 133.93 sec
2025-09-09 23:32:42,594 Stage-1 map = 64%, reduce = 20%, Cumulative CPU 138.82 sec
2025-09-09 23:32:43,624 Stage-1 map = 69%, reduce = 20%, Cumulative CPU 148.6 sec
2025-09-09 23:32:47,757 Stage-1 map = 69%, reduce = 23%, Cumulative CPU 148.65 sec
2025-09-09 23:32:48,785 Stage-1 map = 71%, reduce = 23%, Cumulative CPU 153.19 sec
2025-09-09 23:32:51,870 Stage-1 map = 74%, reduce = 23%, Cumulative CPU 157.73 sec
2025-09-09 23:32:53,940 Stage-1 map = 74%, reduce = 25%, Cumulative CPU 157.79 sec
2025-09-09 23:32:54,967 Stage-1 map = 76%, reduce = 25%, Cumulative CPU 162.88 sec
2025-09-09 23:32:55,993 Stage-1 map = 81%, reduce = 25%, Cumulative CPU 173.01 sec
2025-09-09 23:32:59,081 Stage-1 map = 81%, reduce = 27%, Cumulative CPU 173.08 sec
2025-09-09 23:33:00,126 Stage-1 map = 83%, reduce = 27%, Cumulative CPU 177.54 sec
2025-09-09 23:33:03,217 Stage-1 map = 86%, reduce = 27%, Cumulative CPU 182.12 sec
2025-09-09 23:33:05,280 Stage-1 map = 86%, reduce = 29%, Cumulative CPU 182.21 sec
2025-09-09 23:33:06,305 Stage-1 map = 88%, reduce = 29%, Cumulative CPU 186.85 sec
2025-09-09 23:33:08,362 Stage-1 map = 93%, reduce = 29%, Cumulative CPU 196.42 sec
2025-09-09 23:33:10,415 Stage-1 map = 95%, reduce = 29%, Cumulative CPU 201.04 sec
2025-09-09 23:33:11,447 Stage-1 map = 95%, reduce = 32%, Cumulative CPU 201.13 sec
2025-09-09 23:33:12,467 Stage-1 map = 98%, reduce = 32%, Cumulative CPU 205.86 sec
2025-09-09 23:33:13,486 Stage-1 map = 100%, reduce = 32%, Cumulative CPU 209.44 sec
2025-09-09 23:33:14,505 Stage-1 map = 100%, reduce = 100%, Cumulative CPU 211.26 sec
MapReduce Total cumulative CPU time: 3 minutes 31 seconds 260 msec
Ended Job = job_1757411143708_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 42 Reduce: 1 Cumulative CPU: 211.26 sec HDFS Read: 12029977856 HDFS
Write: 107 SUCCESS
Total MapReduce CPU Time Spent: 3 minutes 31 seconds 260 msec
OK
7009728
Time taken: 115.788 seconds, Fetched: 1 row(s)
hive> █

```

### 65.6.3 What They Do

1. **SELECT count(\*) FROM flights;**
  - o Returns the **total number of rows** in the entire flights table (all years).
  - o Confirms that Hive successfully ingested **all records across 1987–2008**.
  
2. **SELECT count(\*) FROM flights WHERE year = 2008;**
  - o Filters the dataset to just **year = 2008**.

- Returns the total flights recorded in that year.
  - Helps you check **yearly completeness** - for example, you can compare row counts with known dataset statistics.
- 

#### 65.6.4 Why This Is Important

- **Total count check** → ensures your raw ingestion didn't lose rows.
  - **Yearly count check** → ensures no missing partitions (like a year's worth of data skipped).
  - Helps verify your Hive schema matches the source files.
  - Useful for **data quality audits** before you build analysis/ETL pipelines.
- 

#### 65.6.5 Layman's Analogy

Think of your dataset like **airline logbooks** from 1987–2008.

- First query = “*How many pages do I have in total?*”
- Second query = “*How many pages belong to 2008?*”

This way, you can confirm you didn't lose any books (total check) or misplaced a specific year's book (yearly check).

## 65.6.6 Queries You Ran

```
SELECT count(*) FROM airports;
```

```
SELECT count(*) FROM carriers;
```

```
SELECT count(*) FROM planes;
```

---

## 65.6.7 What They Do

- **SELECT count(\*) FROM airports;** → Counts all rows in the **airports** table (you got 317 in your screenshot).
- **SELECT count(\*) FROM carriers;** → Counts all rows in the **carriers** table.
- **SELECT count(\*) FROM planes;** → Counts all rows in the **planes** table.

These queries confirm that your **dimension tables** were successfully created, and Hive can read them.

---

## 65.6.8 Why This Matters

- Dimension tables (airports, carriers, planes) are **lookup tables** that enrich your fact table (flights).
  - If the counts are correct, you know Hive can later join them to the flights table.
  - Example joins:
    - flights.dest ↔ airports.iata → find destination airport name/city.
    - flights.uniquecarrier ↔ carriers.code → find airline full name.
    - flights.tailnum ↔ planes.tailnum → find plane manufacturer/model.

Without validating row counts here, you might hit errors or missing data when running joins.

---

### 65.6.9 Layman's Analogy

Think of your data model as a **giant airline diary**:

- **Flights table** = the diary of every flight (big table).
- **Airports table** = glossary of all airports.
- **Carriers table** = dictionary of all airlines.
- **Planes table** = registry of all aircraft.

Checking row counts in airports/carriers/planes is like making sure your **glossaries are complete**, so when you look up codes (ORD, UA, N12345), you'll always find the right details.

## 66 Technical Explanation: Partitioning in Hive

- **Problem:** Right now, when you query flights WHERE year=2008, Hive scans the **entire flights dataset** (1987–2008). That's billions of rows scanned, even if you only need one year.
- **Solution: Partitioning** divides a Hive table into smaller, logical pieces based on column values (e.g., year, month).

### 66.1 Benefits

1. **Minimize scan size** → Hive only reads the partition that matches your filter (year=2008).
2. **Improve query performance** → Faster because Hive skips irrelevant data.
3. **Reduce cost on cloud platforms** → Less I/O and compute = lower bill.

you're now looking at **dynamic partitioning in Hive**, which is the next step after creating partitioned tables. Let's break it down clearly.

---

### 66.2 Commands to Enable Dynamic Partitioning

```
SET hive.exec.dynamic.partition = true;
SET hive.exec.dynamic.partition.mode = nonstrict;
```

---

#### 66.2.1 What They Do

- **SET hive.exec.dynamic.partition = true;**
  - Tells Hive: “*Go ahead and create partitions automatically at runtime.*”

- Without this, Hive won't generate new partitions dynamically.
- **SET hive.exec.dynamic.partition.mode = nonstrict;**
  - By default, Hive only allows **static partitions** (you must manually define them with ALTER TABLE ... ADD PARTITION).
  - Setting this to nonstrict means Hive can look at your data, detect partition values (e.g., years from the flights dataset), and create those partitions **on the fly**.

### 66.2.2 Layman's Analogy

Think of your flights data like a big pile of documents:

- **Static partitioning** = You manually create folders for each year (1987, 1988, ...) and then sort documents into them.
- **Dynamic partitioning** = You dump all documents on Hive's desk, and Hive automatically creates folders for each year and sorts them for you.

---

⚡ This saves you time when you have **many partitions** (like year + month, or even daily data).

### 66.3 Command to Create External Partitioned Table

```
1. CREATE EXTERNAL TABLE flights_part (
2. month TINYINT,
3. dayofmonth TINYINT,
4. dayofweek TINYINT,
5. deptime STRING,
6. crsdeptime STRING,
7. arrtime STRING,
8. crsarrtime STRING,
9. uniquecarrier STRING,
10. flightnum STRING,
11. tailnum STRING,
12. actualelapsedtime STRING,
13. crselapsedtime STRING,
14. airtime STRING,
15. arrdelay STRING,
16. depdelay STRING,
17. origin STRING,
18. dest STRING,
19. distance STRING,
20. taxiin STRING,
21. taxiout STRING,
22. cancelled STRING,
23. cancellationcode STRING,
24. diverted STRING,
25. carrierdelay STRING,
26. weatherdelay STRING,
27. nasdelay STRING,
28. securitydelay STRING,
29. lateaircraftdelay STRING
30.)
31. PARTITIONED BY (year INT)
32. ROW FORMAT DELIMITED
33. FIELDS TERMINATED BY ',';
34.
```

```

hive> CREATE EXTERNAL TABLE flights_part (
 > month TINYINT,
 > dayofmonth TINYINT,
 > dayofweek TINYINT,
 > deptime STRING,
 > crsdeptime STRING,
 > arrtime STRING,
 > crsarrrtime STRING,
 > uniquecarrier STRING,
 > flightnum STRING,
 > tailnum STRING,
 > actualedapsedtime STRING,
 > crselapsedtime STRING,
 > airtime STRING,
 > arrdelay STRING,
 > depdelay STRING,
 > origin STRING,
 > dest STRING,
 > distance STRING,
 > taxiin STRING,
 > taxiout STRING,
 > cancelled STRING,
 > cancellationcode STRING,
 > diverted STRING,
 > carrierdelay STRING,
 > weatherdelay STRING,
 > nasdelay STRING,
 > securitydelay STRING,
 > lateaircraftdelay STRING
 >)
 > PARTITIONED BY (year INT)
 > ROW FORMAT DELIMITED
 > FIELDS TERMINATED BY ',';
OK
Time taken: 0.093 seconds

```

### 66.3.1 What It Does

- Creates a new **external table** called flights\_part.
- Stores the same flight details as before (month, origin, delays, etc.).
- Partitioned by year** → Hive organizes data in subdirectories by year.
  - Example: /flights\_part/year=2004/

- Example: /flights\_part/year=2005/
  - The data is assumed to be in **CSV format** (comma-separated).
  - Being **external**, Hive only points to the data in HDFS - dropping the table won't delete the actual files.
- 

### 66.3.2 Validation Command

After creating the table, check its structure:

```
DESCRIBE EXTENDED flights_part;
```

This shows:

- Schema of the table.
  - Partition information (year).
  - Location in HDFS.
- 

### 66.3.3 Layman's Analogy

Think of your **flights\_part** table as a **filing cabinet**:

- Each drawer (partition) is labeled with a **year**.
- Inside the drawer, you keep all the flight logs for that year.
- This way, if you only want 2008 flights, Hive only opens the **2008 drawer**, instead of searching the entire cabinet.

```

Time taken: 0.093 seconds
hive> DESCRIBE EXTENDED flights_part;
OK
month tinyint
dayofmonth tinyint
dayofweek tinyint
deptime string
crsdeptime string
arrrtime string
crsarrtime string
uniquecarrier string
flightnum string
tailnum string
actualelapsedtime string
crselapsedtime string
airtime string
arrdelay string
depdelay string
origin string
dest string
distance string
taxiin string
taxiout string
cancelled string
cancellationcode string
diverted string
carrierdelay string
weatherdelay string
nasdelay string
securitydelay string
lateaircraftdelay string
year int

Partition Information
col_name data_type comment
year int

Detailed Table Information Table(tableName:flights_part, dbName:default, owner:Asiema, c
reateTime:1757462142, lastAccessTime:0, retention:0, sd:StorageDescriptor(cols:[FieldSchema(n
ame:month, type:tinyint, comment:null), FieldSchema(name:dayofmonth, type:tinyint, comment:n
ull), FieldSchema(name:dayofweek, type:tinyint, comment:null), FieldSchema(name:deptime, type:
string, comment:null), FieldSchema(name:crsdeptime, type:string, comment:null), FieldSchema(n
ame:arrrtime, type:string, comment:null), FieldSchema(name:crsarrtime, type:string, comment:n
ull), FieldSchema(name:uniquecarrier, type:string, comment:null), FieldSchema(name:flightnum,

```

## 66.4 Insert Data Dynamically Into Partitioned Table

Now you're at the step where you actually **populate the partitioned table dynamically.**

---

### Command

```
1. INSERT INTO flights_part PARTITION (year)
2. SELECT
3. month,
4. dayofmonth,
5. dayofweek,
6. deptime,
7. crsdeptime,
8. arrttime,
9. crsarrttime,
10. uniquecarrier,
11. flightnum,
12. tailnum,
13. actualelapsedtime,
14. crselapsedtime,
15. airtime,
16. arrdelay,
17. depdelay,
18. origin,
19. dest,
20. distance,
21. taxiin,
22. taxiout,
23. cancelled,
24. cancellationcode,
25. diverted,
26. carrierdelay,
27. weatherdelay,
28. nasdelay,
29. securitydelay,
30. lateaircraftdelay,
31. year
32. FROM flights;
33.
34.
```

#### 66.4.1 What It Does

- Takes data from your existing **flights table**.
- Inserts it into **flights\_part**, the **partitioned version** of the table.
- Uses the year column at the end to decide which partition each row belongs to.
- Thanks to:
  - SET hive.exec.dynamic.partition = true;
  - SET hive.exec.dynamic.partition.mode = nonstrict;

Hive will **automatically create partitions for each year (1987–2008)**.

---

#### 66.4.2 Benefits

- Queries like:

```
SELECT * FROM flights_part WHERE year=2008;
```

will now **only scan the 2008 partition**, instead of the full dataset.

- Huge performance improvements and cost savings if running in the cloud.
-

### 66.4.3 Layman's Analogy

Think of it like **re-sorting your flight logbooks**:

- You had one giant pile (flights).
- Now you're filing them into **yearly drawers** (flights\_part).
- Whenever you need 2008, Hive only opens the 2008 drawer instead of flipping through every log since 1987.

## 66.5 Query Hive Tables to Validate Data

This step is about **validating your Hive tables** after creation. Let's go through it clearly.

---

### 66.5.1 Validation Queries

```
SELECT * FROM flights LIMIT 5;
```

```
SELECT * FROM airports LIMIT 5;
```

```
SELECT * FROM carriers LIMIT 5;
```

```
SELECT * FROM planes LIMIT 5;
```

---

### 66.5.2 What They Do

- **SELECT \* FROM flights LIMIT 5;**

→ Shows the first 5 rows of your flight fact table. Confirms that the CSV ingestion (and partitions, if applied) worked correctly.

```

Time taken: 107.106 seconds, Fetched: 1 row(s)
hive> SELECT * FROM flights LIMIT 5;
OK
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| year | month | dayofmonth | dayofyear | origin | dest | distance | hour | tailnum | originAirportID | destAirportID |Cancelled |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1987 | 10 | 14 | 3 | 741 | SFO | 730 | 912 | 849 | PS | 1451 | NA | 91 | 7 |
| 9 | NA | 23 | 11 | SAN | SAN | 447 | NA | NA | 0 | NA | 0 | N |
| A | NA | 0 | NA | 0 | N |
| 1987 | 10 | 15 | 4 | 729 | SAN | 730 | 903 | 849 | PS | 1451 | NA | 94 | 7 |
| 9 | NA | 14 | -1 | SAN | SFO | 447 | NA | NA | 0 | NA | 0 | N |
| A | NA | 0 | NA | 0 | N |
| 1987 | 10 | 17 | 6 | 741 | SAN | 730 | 918 | 849 | PS | 1451 | NA | 97 | 7 |
| 9 | NA | 29 | 11 | SAN | SFO | 447 | NA | NA | 0 | NA | 0 | N |
| A | NA | 0 | NA | 0 | N |
| 1987 | 10 | 18 | 7 | 729 | SAN | 730 | 847 | 849 | PS | 1451 | NA | 78 | 7 |
| 9 | NA | -2 | -1 | SAN | SFO | 447 | NA | NA | 0 | NA | 0 | N |
| A | NA | 0 | NA | 0 | N |
| 1987 | 10 | 19 | 1 | 749 | SFO | 730 | 922 | 849 | PS | 1451 | NA | 93 | 7 |
| 9 | NA | 33 | 19 | SAN | SFO | 447 | NA | NA | 0 | NA | 0 | N |
| A | NA | 0 | NA | 0 | N |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
Time taken: 0.108 seconds, Fetched: 5 row(s)
hive>

```

- **SELECT \* FROM airports LIMIT 5;**

→ Displays a sample of the airports lookup table. Ensures airport codes map correctly.

```

hive> SELECT * FROM airports LIMIT 5;
OK
+-----+-----+-----+-----+-----+-----+
|iata | airport | city | state | country | lat | long |
+-----+-----+-----+-----+-----+-----+
| 00M | Thigpen | Bay Springs | MS | USA | 31.95376472 | -89.23450472 |
| 00R | Livingston Municipal | Livingston | TX | USA | 30.68586111 | -95.01792778 |
| 00V | Meadow Lake | Colorado Springs | CO | USA | 38.94574889 | -104.5698933 |
| 01G | Perry-Warsaw | Perry | NY | USA | 42.74134667 | -78.05208056 |
+-----+-----+-----+-----+-----+-----+
Time taken: 0.09 seconds, Fetched: 5 row(s)
hive>

```

- **SELECT \* FROM carriers LIMIT 5;**

→ Displays a few airline carrier codes with descriptions. Checks carrier data ingestion.

```

hive> SELECT * FROM carriers LIMIT 5;
OK
+-----+-----+
| Code | Description |
+-----+-----+
| 02Q | Titan Airways |
| 04Q | Tradewind Aviation |
| 05Q | Comlux Aviation, AG |
| 06Q | Master Top Linhas Aereas Ltd. |
+-----+-----+
Time taken: 0.083 seconds, Fetched: 5 row(s)
hive>

```

- **SELECT \* FROM planes LIMIT 5;**

→ Displays a few rows from the planes registry. Confirms plane metadata is available.

```
Time taken: 0.083 seconds, Fetched: 5 row(s)
hive> SELECT * FROM planes LIMIT 5;
OK
tailnum type manufacturer issue_date model status aircraft_type engine_type
N050AA NULL NULL NULL NULL NULL NULL
N051AA NULL NULL NULL NULL NULL NULL
N052AA NULL NULL NULL NULL NULL NULL
N054AA NULL NULL NULL NULL NULL NULL
Time taken: 0.099 seconds, Fetched: 5 row(s)
hive>
```

---

#### 66.5.3 Why This Is Important

- Confirms **data ingestion worked** (tables aren't empty).
- Ensures your schema matches the data (columns align properly).
- Quick way to **catch parsing issues** (like NULL values, wrong delimiters, missing headers).

---

#### 66.5.4 Layman's Analogy

Think of it like flipping open the first 5 pages of each logbook (flights, airports, carriers, planes). If the first pages look clean and readable, chances are the rest of the book was written correctly too.

## 66.6 Row counts

This step is about validating your dataset **row counts** in Hive to make sure ingestion worked correctly.

---

### 66.6.1 Queries

```
SELECT COUNT(*) FROM flights;
```

```
SELECT COUNT(*) FROM flights WHERE year = 2008;
```

---

```
MapReduce Total cumulative CPU time: 2 minutes 52 seconds 50 msec
Ended Job = job_1757411143708_0009
MapReduce Jobs Launched:
Stage-Stage-1: Map: 42 Reduce: 1 Cumulative CPU: 172.05 sec HDFS Read: 12029948742 HDFS
Write: 109 SUCCESS
Total MapReduce CPU Time Spent: 2 minutes 52 seconds 50 msec
OK
123534969
Time taken: 107.301 seconds, Fetched: 1 row(s)
hive> █
```

### 66.6.2 What They Do

1. **SELECT COUNT(\*) FROM flights;**
  - Returns the **total number of rows** across the entire flights dataset (1987–2008).
  - Confirms whether all flight records were successfully loaded.
2. **SELECT COUNT(\*) FROM flights WHERE year=2008;**
  - Filters only flights for 2008.
  - Returns the **row count for that specific year**.
  - Ensures partitioning and data ingestion for 2008 worked properly.

---

### 66.6.3 Why This Is Important

- Validates that the **flights table isn't empty**.
  - Confirms ingestion across **all years** (total count).
  - Confirms ingestion of **specific year(s)** (yearly count).
  - Helps you check if data matches expected source counts (if you know them from documentation).
- 

### 66.6.4 Layman's Analogy

It's like checking a bookshelf after moving books:

- First count → “*Do I still have all 22 books from 1987–2008?*”
- Second count → “*Is the 2008 book fully there, or did I lose pages?*”

## 67 Partitioning

When you query large datasets (gigabytes or terabytes), Hive by default **scans the entire table**. That's slow and expensive. Partitioning helps by breaking the data into smaller, logical pieces.

---

### 1. Benefits of Partitioning

1. **Minimize scan size** → Only read the relevant slice of data.
  2. **Faster queries** → Skip irrelevant data.
  3. **Lower costs** → On cloud platforms, less compute and storage I/O.
- 

#### 67.1.1 Example (Flights Dataset)

- Suppose your dataset covers years 2005–2008.
- Without partitioning:
- `SELECT * FROM flights WHERE year = 2008;`

Hive still scans **all 4 years** (2005, 2006, 2007, 2008).

- With partitioning by year:
    - Hive will only read the partition folder `/flights_part/year=2008/`.
    - It **skips all other years**, saving time and resources.
-

### 67.1.2 Layman's Analogy

Think of your dataset like a filing cabinet:

- **Without partitioning:** All papers from 2005–2008 are dumped in one drawer. If you want 2008, you must flip through everything.
- **With partitioning:** Each year has its own labeled drawer. To find 2008, you just open the “2008” drawer - much faster.

## 67.2 Enable Dynamic Partitioning

This step is about enabling **dynamic partitioning in Hive**, which saves you from manually creating partitions.

---

### 67.2.1 Commands

```
SET hive.exec.dynamic.partition = true;
```

```
SET hive.exec.dynamic.partition.mode = nonstrict;
```

---

### 67.2.2 What They Do

- **SET hive.exec.dynamic.partition = true;**  
→ Allows Hive to create partitions automatically at runtime. Without this, Hive won't generate new partitions dynamically.
- **SET hive.exec.dynamic.partition.mode = nonstrict;**  
→ By default, Hive only allows **static partitions** (you must specify them manually).

→ Setting it to **nonstrict** enables **fully dynamic partitioning**, where partition values (e.g., year) are pulled directly from the data itself.

---

### 67.2.3 Example in Your Case

If you run:

```
INSERT OVERWRITE TABLE flights PARTITION (year)
SELECT
 month, dayofmonth, dayofweek, deptime, crsdeptime, arrtime,
 crsarrrtime, uniquecarrier, flightnum, tailnum, actualelapsedtime,
 crselapsedtime, airtime, arrdelay, depdelay, origin, dest, distance,
 taxiin, taxiout, cancelled, cancellationcode, diverted, carrierdelay,
 weatherdelay, nasdelay, securitydelay, lateaircraftdelay, year

FROM flights_raw;
```

👉 Hive will look at the year column and **automatically create partitions** like:

/flights/year=1987/

/flights/year=1988/

/flights/year=1989/

/flights/year=.../

#### 67.2.4 Why It's Useful

- No need to manually ALTER TABLE ... ADD PARTITION for every year.
  - Handles large datasets (e.g., daily/hourly partitions) with ease.
  - Makes queries much faster (WHERE year=2008 → Hive only scans /year=2008/).
- 

 **Layman's Analogy:** Instead of you manually labeling every folder by year and filing the documents, you just dump everything on Hive's desk - Hive reads the year from each page and creates the correct folders automatically.

### 67.3 Create an External Partitioned Table

You're now at the “**Create External Partitioned Table**” step. Let's break it down.

---

#### 67.3.1 Command

*CREATE EXTERNAL TABLE flights\_part (*

*month TINYINT,*

*dayofmonth TINYINT,*

*dayofweek TINYINT,*

*deptime STRING,*

*crsdeptime STRING,*

*arrtime STRING,*

*crsarrtime STRING,*

*uniquecarrier STRING,*

*flightnum STRING,*  
*tailnum STRING,*  
*actualelapsedtime STRING,*  
*crselapsedtime STRING,*  
*airtime STRING,*  
*arrdelay STRING,*  
*depdelay STRING,*  
*origin STRING,*  
*dest STRING,*  
*distance STRING,*  
*taxiin STRING,*  
*taxiout STRING,*  
*cancelled STRING,*  
*cancellationcode STRING,*  
*diverted STRING,*  
*carrierdelay STRING,*  
*weatherdelay STRING,*  
*nasdelay STRING,*  
*securitydelay STRING,*  
*lateaircraftdelay STRING*

)

*PARTITIONED BY (year SMALLINT)*

*ROW FORMAT DELIMITED*

*FIELDS TERMINATED BY ',';*

```
hive> set hive.exec.dynamic.partition.mode=nonstrict;
hive> CREATE EXTERNAL TABLE flights_part (
 > month TINYINT,
 > dayofmonth TINYINT,
 > dayofweek TINYINT,
 > deptime STRING,
 > crsdeptime STRING,
 > arrtime STRING,
 > crsarrrtime STRING,
 > uniquecarrier STRING,
 > flightnum STRING,
 > tailnum STRING,
 > actualelapsedtime STRING,
 > crselapsedtime STRING,
 > airtime STRING,
 > arrdelay STRING,
 > depdelay STRING,
 > origin STRING,
 > dest STRING,
 > distance STRING,
 > taxiin STRING,
 > taxiout STRING,
 > cancelled STRING,
 > cancellationcode STRING,
 > diverted STRING,
 > carrierdelay STRING,
 > weatherdelay STRING,
 > nasdelay STRING,
 > securitydelay STRING,
 > lateaircraftdelay STRING
 >)
 > PARTITIONED BY (year SMALLINT)
 > ROW FORMAT DELIMITED
 > FIELDS TERMINATED BY ',';
OK
Time taken: 0.095 seconds
hive> █
```

### 67.3.2 What It Does

- Creates an **external table** in Hive called flights\_part.

- Table schema includes flight-related columns (month, delays, origin, destination, etc.).
  - It is **partitioned by the year column** - Hive will create subfolders for each year (1987 ... 2008).
  - The data is assumed to be in **CSV format**, fields separated by commas.
  - Because it's **external**, Hive only points to the data in HDFS - dropping the table won't delete the files.
- 

### 67.3.3 How to Check

After creating the table, run:

```
DESCRIBE EXTENDED flights_part;
```

You'll see:

- The schema (columns and types).
  - Partition information (year).
  - The HDFS location where data will be stored.
-

```
Detailed Table Information Table(tableName:flights_part, dbName:default, owner:Asiema, c
reateTime:1757511976, lastAccessTime:0, retention:0, sd:StorageDescriptor(cols:[FieldSchema(n
ame:month, type:tinyint, comment:null), FieldSchema(name:dayofmonth, type:tinyint, comment:n
ull), FieldSchema(name:dayofweek, type:tinyint, comment:null), FieldSchema(name:deptime, type:
string, comment:null), FieldSchema(name:crsdeptime, type:string, comment:null), FieldSchema(n
ame:arrtime, type:string, comment:null), FieldSchema(name:crsarrtime, type:string, comment:n
ull), FieldSchema(name:uniquecarrier, type:string, comment:null), FieldSchema(name:flightnum,
type:string, comment:null), FieldSchema(name:tailnum, type:string, comment:null), FieldSchema
(name:actualelapsedtime, type:string, comment:null), FieldSchema(name:crselapsedtime, type:st
ring, comment:null), FieldSchema(name:airtime, type:string, comment:null), FieldSchema(name:a
rrdelay, type:string, comment:null), FieldSchema(name:depdelay, type:string, comment:null), F
ieldSchema(name:origin, type:string, comment:null), FieldSchema(name:dest, type:string, comme
nt:null), FieldSchema(name:distance, type:string, comment:null), FieldSchema(name:taxiin, typ
e:string, comment:null), FieldSchema(name:taxiout, type:string, comment:null), FieldSchema(na
me:cancelled, type:string, comment:null), FieldSchema(name:cancellationcode, type:string, com
ment:null), FieldSchema(name:diverted, type:string, comment:null), FieldSchema(name:carrierde
lay, type:string, comment:null), FieldSchema(name:weatherdelay, type:string, comment:null), F
ieldSchema(name:nasdelay, type:string, comment:null), FieldSchema(name:securitydelay, type:st
ring, comment:null), FieldSchema(name:lateaircraftdelay, type:string, comment:null), FieldSch
ema(name:year, type:smallint, comment:null)], location:hdfs://127.0.0.1:9000/user/hive/wareho
use/flights_part, inputFormat:org.apache.hadoop.mapred.TextInputFormat, outputFormat:org.apac
he.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat, compressed:false, numBuckets:-1, serdeInf
o:SerDeInfo(name:null, serializationLib:org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe, p
arameters:{serialization.format=, field.delim=,}), bucketCols:[], sortCols:[], parameters:{}
, skewedInfo:SkewedInfo(skewedColNames:[], skewedColValues:[], skewedColValueLocationMaps:{}),
storedAsSubDirectories:false, partitionKeys:[FieldSchema(name:year, type:smallint, comment:
null)], parameters:{totalSize=0, EXTERNAL=TRUE, numRows=0, rawDataSize=0, COLUMN_STATS_ACCUR
ATE={"BASIC_STATS":"true"}, numFiles=0, numPartitions=0, transient_lastDdlTime=1757511976
, bucketing_version=2}, viewOriginalText:null, viewExpandedText:null, tableType:EXTERNAL_TABLE
, rewriteEnabled:false, catName:hive, ownerType:USER)
Time taken: 0.115 seconds, Fetched: 35 row(s)
```

## Layman's Analogy

Think of flights\_part as a **filing cabinet where each drawer = one year**.

- You've defined what each folder inside contains (columns like origin, destination, delay).
- Hive won't move data into the cabinet yet - it just knows **how to organize** it once you insert records.

## 67.4 Insert Data Dynamically Into Partitioned Table

You're now at the **Insert Data Dynamically Into Partitioned Table** step. Let's break this down clearly.

---

### Command

*INSERT INTO flights\_part PARTITION (year)*

*SELECT*

*month,*

*dayofmonth,*

*dayofweek,*

*deptime,*

*crsdeptime,*

*arrtime,*

*crsarrtime,*

*uniquecarrier,*

*flightnum,*

*tailnum,*

*actualelapsedtime,*

*crselapsedtime,*

*airtime,*

*arrdelay,*

```
depdelay,
origin,
dest,
distance,
taxiin,
taxiout,
cancelled,
cancellationcode,
diverted,
carrierdelay,
weatherdelay,
nasdelay,
securitydelay,
lateaircraftdelay,
year
FROM flights;
```

---

## What It Does

- Copies all rows from your **flights** table (raw, non-partitioned).
- Inserts them into **flights\_part**, which is partitioned by year.

- Hive automatically creates subfolders in HDFS:
  - /flights\_part/year=1987/
  - /flights\_part/year=1988/
  - ...
  - /flights\_part/year=2008/
  - Each year's data goes into its respective folder.
- 

#### 67.4.1 Why Dynamic Partitioning Matters

Earlier, you enabled:

```
SET hive.exec.dynamic.partition = true;
```

```
SET hive.exec.dynamic.partition.mode = nonstrict;
```

👉 This allows Hive to read the year column values from the data and create partitions on-the-fly, instead of you manually specifying them.

---

## 67.4.2 Layman's Analogy

Think of it like taking one giant box of flight papers (flights table) and telling Hive: "Sort these into drawers by year." Hive automatically files 1987 flights into the 1987 drawer, 1988 flights into the 1988 drawer, etc.

## 67.5 View Table Details

This is the "**View Table Details**" step for your partitioned table.

---

### Command

```
DESCRIBE EXTENDED flights_part;
```

---

## 67.5.1 What It Does

- Displays **detailed metadata** about the table.
  - Information includes:
    - **Schema** → all column names and types.
    - **Partition Info** → confirms year is used as a partition.
    - **Table Type** → should say *EXTERNAL*.
    - **Location** → the exact HDFS directory where data is stored.
    - **Serde info** → how Hive reads the data (CSV, OpenCSVSerde, etc.).
-

### 67.5.2 Why It's Important

- Confirms your **partition setup is correct**.
  - Ensures Hive is reading from the right **HDFS location**.
  - Helps debug issues like empty tables (if Hive is pointing to the wrong path).
- 

### 67.5.3 Layman's Analogy

It's like pulling the **label card** from your filing cabinet drawer. It tells you:

- What type of drawer it is (external).
- What kind of files are inside (schema).
- How the files are organized (partitioning).
- Where the drawer physically sits (HDFS location).

## 67.6 Upload New data to HDFS

This step explains how to **upload new data into HDFS** so Hive can access it.

---

### Command

```
hdfs dfs -moveFromLocal /disk/data/flights/csv/2005.csv
/user/Asiema/data/flights/csv
```

---

## What It Does

- **hdfs dfs -moveFromLocal** → Moves a file from your local Linux filesystem into HDFS.
  - **/disk/data/flights/csv/2005.csv** → Source file on local disk.
  - **/user/Asiema/data/flights/csv** → Destination folder in HDFS where Hive will read it.
  - After moving, the **local copy is deleted** (only the HDFS copy remains).
- 

### 67.6.1 Why It's Important

- Hive external tables depend on **HDFS paths**.
  - By moving 2005.csv into /user/Asiema/data/flights/csv, Hive will automatically pick it up when you query (if your table points to that directory).
- 

### 67.6.2 Alternative

If you don't want to delete the local file, use:

```
hdfs dfs -copyFromLocal /disk/data/flights/csv/2005.csv
/user/Asiema/data/flights/csv
```

---

### 67.6.3 Next Steps

1. Run this move command for any **new year files** (2006.csv, 2007.csv, etc.).
  2. Check that the files exist in HDFS:
  3. `hdfs dfs -ls /user/Asiema/data/flights/csv`
  4. Validate Hive sees the new data:
  5. `SELECT COUNT(*) FROM flights_part WHERE year=2005;`
- 

#### Layman's Analogy:

Think of HDFS as your “official cabinet” and local disk as your “desk.”

- `moveFromLocal` = pick the paper from your desk, file it in the cabinet, and throw away the desk copy.
- `copyFromLocal` = make a photocopy for the cabinet but keep the desk version.

### 67.7 Register new partition in Hive

This step is about **registering a new partition in Hive** after uploading a new year’s CSV file to HDFS.

---

## Command

```
ALTER TABLE flights_part
ADD PARTITION (year=2005)
LOCATION
'/user/hive/warehouse/airlines.db/flights_part/year=2005';
```

---

### 67.7.1 What It Does

- **ALTER TABLE flights\_part** → Tells Hive you're updating the partition metadata for this table.
  - **ADD PARTITION (year=2005)** → Creates a partition for the year 2005.
  - **LOCATION '...'** → Points Hive to the exact HDFS directory where 2005's data is stored.
- 

### 67.7.2 Why It's Needed

- When you move or copy new year files (like 2005.csv) into HDFS manually, Hive won't automatically know they exist.
- This command “registers” the folder so Hive can query it.
- Without it, queries like:
- `SELECT COUNT(*) FROM flights_part WHERE year=2005;`

would return **0 rows**, even though the data is physically in HDFS.

---

### 67.7.3 Layman's Analogy

Think of it like putting a new drawer (2005) into your filing cabinet. You placed the files inside HDFS, but Hive doesn't know about the drawer yet - this command **labels and registers it** so Hive can open and read it.

---

### 67.7.4 Pro Tip: Add All Partitions Automatically

Instead of adding one year at a time, you can refresh all partitions with:

```
MSCK REPAIR TABLE flights_part;
```

or in modern Hive:

```
ALTER TABLE flights_part RECOVER PARTITIONS;
```

This scans the HDFS path (/flights\_part/) and registers **all missing year partitions** automatically.

## 67.8 Load Data from the Main Table into the 2005 Partition

Now you're looking at the **manual per-year load into a partition** method.

---

### 67.8.1 Command

`INSERT INTO flights_part PARTITION (year=2005)`

`SELECT`

`month,`

`dayofmonth,`

`dayofweek,`

`deptime,`

`crsdeptime,`

`arrtime,`

`crsarrtime,`

`uniquecarrier,`

`flightnum,`

`tailnum,`

`actualelapsedtime,`

`crselapsedtime,`

`airtime,`

`arrdelay,`

`depdelay,`

```
origin,
dest,
distance,
taxiin,
taxiout,
cancelled,
cancellationcode,
diverted,
carrierdelay,
weatherdelay,
nasdelay,
securitydelay,
lateaircraftdelay

FROM flights

WHERE year=2005;
```

---

### 67.8.2 What It Does

- Takes **only 2005 records** from the raw flights table.
- Inserts them into the **partition for year=2005** in your flights\_part table.
- Unlike dynamic partitioning, here you **manually specify** the partition year.

---

### 67.8.3 Why This Helps

- If your dynamic partition job (1987–2008 all at once) is too slow, this method **breaks it into smaller pieces.**
  - Easier to debug → if the 2005 load works, you can repeat for 2006, 2007, etc.
  - Avoids reducer bottlenecks caused by a single massive shuffle.
- 

### 67.8.4 Layman's Analogy

Instead of asking Hive to **sort all flight records (1987–2008) into 22 drawers in one go**, you say:

👉 “Just take the 2005 papers and put them into the 2005 drawer.”

It's slower if you repeat manually, but each step is faster, cleaner, and less error-prone.

## 67.9 Compare Query on Non-partitioned vs Partitioned Table

This is a **perfect illustration of why partitioning matters in Hive** 

---

### 67.9.1 The Queries

#### 67.9.1.1 Non-Partitioned

```
SELECT DISTINCT month FROM flights WHERE year=2008;
```

- Runs on the **non-partitioned table** (flights).
  - Hive has to scan the **entire dataset (all years 1987–2008)**.
  - Filtering by year=2008 happens **after scanning everything**, so it wastes compute.
- 

#### 67.9.1.2 Partitioned

```
SELECT DISTINCT month FROM flights_part WHERE year=2008;
```

- Runs on the **partitioned table** (flights\_part).
  - Hive directly jumps into the year=2008 partition folder in HDFS.
  - Only scans **data for 2008**, skipping all other years.
  - Much faster (usually less than 50% of the time).
-

### 67.9.2 Why the Partitioned Query Wins

- **Data scanned:**
    - Non-partitioned → all years, millions of rows.
    - Partitioned → only 2008 data, thousands of rows.
  - **I/O:** Hive reads fewer blocks from HDFS.
  - **CPU:** Less unnecessary filtering work.
  - **Cost:** On cloud clusters (Azure, AWS EMR, etc.), fewer resources = lower cost.
- 

### 67.9.3 Layman's Analogy

Imagine you have a **giant filing cabinet** (1987–2008).

- Non-partitioned = You open the **whole cabinet** and read every paper to find 2008.
  - Partitioned = You go **directly to the 2008 drawer** and only read that.
- 

### 67.9.4 Summary

- Both queries return the same result (months with flights in 2008).
- The partitioned query is **faster and cheaper** because Hive only scans what's needed.
- Partitioning is one of the **most effective optimizations** for big data queries.

# 68 Clustering/ Bucketing

Definition:

Bucketing is a technique in Hive that improves query efficiency by splitting data into a fixed number of buckets (files), based on the **hash value** of a chosen column.

## 68.1 Differences from Partitioning:

- **Partitioning:** Splits data based on distinct values of a column (e.g., each year gets its own folder). If you have 20 years of data, you get 20 partitions.
- **Bucketing:** Divides data into a **fixed number** of buckets (e.g., 10), regardless of how many distinct values exist. Records are distributed into buckets using a hashing algorithm on the chosen column.

## 68.2 How it Works:

- You select a column (say student\_id).
- Hive computes a **hash value** for each row's student\_id.
- That hash value determines which bucket the row goes into.
- The number of buckets is fixed (e.g., 10).
- As a result, data is evenly distributed.

## 68.3 Performance Benefits:

- **Faster lookups:** When you query a specific student\_id, Hive only checks the relevant bucket instead of scanning the entire dataset.
- **Efficient joins:** If two tables are bucketed on the same column and number of buckets, Hive can **map each bucket directly** during joins instead of shuffling all the data.

- **Better parallelism:** Since data is evenly spread, Hive can process queries in parallel across buckets.

## 68.4 Use Cases:

- Repeated queries on certain columns (student\_id, customer\_id, etc.).
  - Large datasets where scanning everything would be costly.
  - Pre-optimizing joins between big tables.
- 

## 68.5 Layman's Explanation

Think of **bucketing like organizing books in a library**:

- **Without bucketing:**

All books are dumped into one giant pile. If you want to find one book (say a math book by “Student ID 25”), you must search through the entire pile. That’s slow.

- **With bucketing:**

The librarian decides: “I’ll have 10 shelves (buckets), and I’ll place each book on a shelf depending on the last digit of the student’s ID.”

- If student ID = 25 → goes to shelf 5
- If student ID = 42 → goes to shelf 2

Now, when someone asks for Student ID 25’s record, the librarian goes straight to shelf 5 instead of scanning all the shelves. Much faster!

## 68.6 Why it's better than partitioning:

- Partitioning is like making a separate room for every subject. If you have 500 subjects, you'll end up with 500 rooms - too many to manage.
- Bucketing is like deciding you'll only ever have 10 shelves, and no matter how many subjects there are, you spread them across those 10 shelves. Easier to handle, more balanced.
- **Benefit in simple terms:**

Bucketing means **less searching, faster results**. Instead of looking through the entire dataset, Hive knows *exactly which bucket to check*.

---

⚡ In short:

- **Techy way:** Bucketing uses hashing to distribute rows into a fixed number of buckets, improving joins and queries.
- **Every day:** Bucketing is like splitting a huge pile into smaller piles (buckets), so you only search where you need to - saving time and effort.

## 68.7 Execution

When working with **bucketing and partitioning** in Hive, you must configure these three settings first:

1. **set hive.enforce.bucketing = true;**
  - Ensures Hive **applies bucketing rules** when inserting data into a bucketed table.
  - Without this, Hive might not distribute rows properly into the predefined buckets.
  - It essentially says: “Don’t ignore the bucket definition - follow it strictly.”
2. **set hive.exec.dynamic.partition = true;**
  - Enables **dynamic partitioning**.
  - Normally, partitions must be specified manually when inserting. Dynamic partitioning allows Hive to **create partitions on the fly** based on the data.
  - Example: If inserting student data by year, Hive automatically creates new partitions like year=2022, year=2023 when such values appear in the data.
3. **set hive.exec.dynamic.partition.mode = nonstrict;**
  - Hive has two modes for dynamic partitioning:
    - **Strict:** At least one static partition column must be provided (e.g., year=2023 while month is dynamic).
    - **Nonstrict:** Allows **all partition columns to be dynamic**. Hive figures everything out automatically.

- This is useful when you don't want to hardcode any partition values at insert time.

#### Together:

- Bucketing is enforced (data distributed into buckets properly).
  - Partitions can be created dynamically.
  - Dynamic partitions can be fully automated (no need for any static value upfront).
- 

## 68.8 Layman's Explanation

Think of this as setting the **rules for organizing your library** before you start shelving books.

### 1. **hive.enforce.bucketing = true;**

- Like telling the librarian:

*“Make sure you always follow the rule of putting books on the correct shelves (buckets), don’t just toss them randomly.”*

### 2. **hive.exec.dynamic.partition = true;**

- Like saying:

*“If a new subject comes in that doesn’t have a section yet, go ahead and create a new section on the spot.”*

### 3. **hive.exec.dynamic.partition.mode = nonstrict;**

- Like telling the librarian:

*“You don’t need me to tell you which subjects to create. Just create*

*whatever sections are needed as new subjects arrive.”*

---

 In short:

- **Technical side:** These settings enforce bucketing, allow dynamic partitions, and permit fully dynamic (nonstrict) partition creation.
- **Layman side:** They’re like rules you give a librarian: always shelve correctly, create new sections automatically, and don’t wait for me to predefine them.

## 68.9 Insert Data into the Bucketed Table

This is the **actual Hive table creation statement** for a **bucketed and partitioned table**.

Let's expand and then simplify.

---

### 68.9.1 Technical Explanation (Expanded)

The command:

```
create external table flight_bkt (
 month tinyint,
 dayofmonth tinyint,
 dayofweek tinyint,
 deptime smallint,
 crsdeptime smallint,
 arrtime smallint,
 crsarrrtime smallint,
 uniquecarrier string,
 flightnum string,
 tailnum string,
 actualelapsedtime smallint,
 crselapsedtime smallint,
 airtime smallint,
 arrdelay smallint,
```

```
depdelay smallint,
origin string,
dest string,
distance smallint,
taxiin string,
taxiout string,
cancelled string,
cancellationcode string,
diverted string,
carrierdelay smallint,
weatherdelay smallint,
nasdelay smallint,
securitydelay smallint,
lateaircraftdelay smallint
)
partitioned by (year smallint)
clustered by (month)
sorted by (month, dayofmonth)
into 4 buckets
row format delimited
fields terminated by ',';
```

## 68.10 Breakdown:

### 1. External Table (create external table)

- Table **flight\_bkt** is external, meaning Hive only stores metadata.
- The data itself remains in its original location (say HDFS path). If the table is dropped, the data is not deleted.

### 2. Partitioning (partitioned by (year))

- Data is divided into **directories based on year** (e.g., /year=2006/, /year=2007/).
- This helps when queries filter by year → only that year's folder is scanned.

### 3. Bucketing (clustered by (month) into 4 buckets)

- Within each year, data is split into **4 buckets** using the hash of month.
- This evenly distributes rows across 4 files, improving query parallelism and joins.

### 4. Sorting (sorted by (month, dayofmonth))

- Data inside each bucket is **sorted by month, then dayofmonth**.
- Makes range queries (e.g., "all flights in March, 15th–20th") much faster because Hive doesn't scan unnecessary rows.

### 5. Storage format (row format delimited fields terminated by ',')

- Defines the file format (CSV-like, separated by commas).

## 68.11 Layman's Explanation

Imagine we're organizing **flight records** in a big filing system:

### 1. External Table

- Hive is just the **catalog**.
- It knows *where the files live*, but it doesn't own them. If you throw away the catalog, the actual files are still safe in storage.

### 2. Partition by Year

- First, you create a **separate cabinet for each year** (2006, 2007, etc.).
- So, if you only need 2007 flights, you don't waste time looking in other cabinets.

### 3. Bucket by Month (4 Buckets)

- Inside each year's cabinet, you have **4 drawers (buckets)**.
- Every flight record is dropped into one of those 4 drawers based on its **month**.
- This keeps drawers evenly filled and makes searching quicker.

### 4. Sort by Month and Day

- Within each drawer, records are neatly **sorted in date order** (month → day).
- So, if you ask for “flights on March 10th,” you can flip straight to that page instead of shuffling through random papers.

### 5. CSV Format

- The records are written like **comma-separated lists**.
  - Easy to read, share, or process with other tools.
- 

 **In simple terms:**

This table is like a well-organized filing system:

- **Cabinets** = years (partitions)
- **Drawers** = 4 evenly shared buckets inside each year
- **Neatly ordered pages** = sorted by month/day
- **Catalog** = Hive external table keeps track of where everything is

This makes finding, filtering, and joining flight data **way faster** than scanning one giant messy file.

**Technical Explanation (Expanded)**

**Query Used:**

```
select month, count(1)
from <table_name>
where year = 2008 and month in (2,3)
group by month;
```

**Results:**

1. **Bucketed table (flight\_bkt)**

- **Time taken:** ~172 seconds (but efficient CPU use, smaller shuffle).
- Query only scans **the relevant buckets for months 2 and 3** inside year 2008.
- Since data is pre-distributed by month, Hive avoids scanning unnecessary buckets.

## 2. Partitioned table (flights\_part)

- **Time taken:** ~35 seconds.
- Partitioning by year helps eliminate irrelevant years.
- But within year 2008, Hive still needs to scan **all months** because partitioning wasn't done on month, only year.
- This is faster than flat but slower than bucketed when filtering on month.

## 3. Flat table (flights)

- **Time taken:** ~2+ minutes.
- No partitioning, no bucketing. Hive scans the **entire dataset**, filtering rows at runtime.
- Very inefficient for large datasets.

### **Key Insight:**

- **Partitioning helps skip big chunks (e.g., years).**
- **Bucketing helps optimize within partitions (e.g., months).**
- **Flat tables are slow** because every row must be checked.

👉 For **range-based queries** (like filtering on specific months), **bucketing is most efficient** because the data is **pre-sorted into smaller, manageable files**.

---

### 68.11.1 Layman's Explanation

Imagine you're trying to **count flights in February and March 2008**.

1. **Flat Table (flights)** – Like a **giant box with all flight papers mixed together**.
    - You must flip through every page from all years and months just to find Feb and March 2008.
    - Takes a lot of time.
  2. **Partitioned Table (flights\_part)** – Like having a **cabinet for each year**.
    - You open the 2008 cabinet.
    - But inside, all months are still in one messy drawer. You still have to scan through the whole year's data.
    - Faster than the giant box, but still time-consuming.
  3. **Bucketed Table (flight\_bkt)** – Like having a **cabinet for each year, and inside it drawers for months**.
    - You open the 2008 cabinet, then go directly to the drawers for February and March.
    - Much faster, because you skip everything else.
-

**In simple terms:**

- Flat table = search everywhere.
- Partitioned table = search only in the right year, but still scan inside.
- Bucketed table = search in the exact drawers you need → fastest.

## 69 Sampling

### 69.1 What is Sampling?

- Sampling is the process of querying only a **subset of data** instead of scanning the entire dataset.
- This is useful for **prototyping, debugging, or quick analysis** when full accuracy is not needed.
- It reduces **query execution time, resource usage, and cost** in large-scale datasets.

### 69.2 When to use Sampling:

- Debugging or exploring unfamiliar datasets.
- Running experiments or quick analytics where approximate results are acceptable.
- Cost control - scanning billions of rows just to test a query is inefficient.

### 69.3 Types of Sampling in Hive:

#### 1. **LIMIT**

- Example:

```
SELECT * FROM sales LIMIT 1000;
```

- Returns the first 1000 rows (not random, just the top slice).

## 2. ORDER BY RAND()

- Example:

```
SELECT * FROM sales ORDER BY RAND() LIMIT 1000;
```

- Returns 1000 random rows. Useful for unbiased samples.

## 3. TABLESAMPLE

- Example:

```
SELECT * FROM sales TABLESAMPLE(BUCKET 3 OUT OF 10);
```

```
SELECT * FROM sales TABLESAMPLE(10 PERCENT);
```

- Options:

- **Row count:** e.g., 1000 rows.
- **Percent:** e.g., 10% of rows.
- **Bucket:** directly pick data from specific buckets (if the table is bucketed).

### Practical Example (Retail Data):

If you have **millions of transactions**:

- Instead of running queries over all rows, you can:
    - Sample **1000 random transactions**.
    - Sample **10% of all rows**.
    - Sample from a **specific bucket** (if bucketing was used earlier).
- 

### 69.4 Layman's Explanation

Think of sampling as **taste-testing food**:

- If you cook a giant pot of soup, you don't drink the entire pot to check the flavor.
- Instead, you take **a spoonful sample** - that's enough to get an idea.

Similarly in Hive:

- Instead of scanning millions of records, you take a **small representative subset**.
- This saves **time, effort, and money** (since big queries cost more in compute).

### 69.5 Types of Sampling in Plain Terms:

1. **LIMIT** → Just grab the first 1000 rows (like looking at the first 1000 receipts from a pile).
2. **ORDER BY RAND()** → Pick 1000 random rows (like shuffling the pile and grabbing random receipts).

3. **TABLESAMPLE** → Choose rows by percentage, count, or bucket (like saying: “I’ll check 10% of all receipts” or “just the records in drawer 3”).

### Everyday Example:

Imagine a supermarket with **10 million receipts**:

- You don’t need to check them all to know the top-selling product.
- You can sample:
  - 1000 random receipts,
  - 10% of receipts, or
  - receipts from one drawer (bucket).

You still see the **trend** without reading every single record.

---

### In short:

- **Techy way:** Sampling lets you query subsets of data (row count, percent, or bucket) for faster prototyping and lower costs.
- **Everyday way:** Sampling is like taste-testing - you don’t need to eat the whole pot of soup to know if it’s good.

## 70 Simple Limit-based Sampling

### 70.1 Query:

```
select *
from carriers
limit 20;
```

#### What it does:

- Fetches the **first 20 rows** from the carriers table.
- Hive does not scan the **entire dataset**; it just retrieves the first set of rows.
- Useful for **previewing data** (checking structure, format, column values) without running a full scan.

#### 70.1.1 Why it's important:

- **Debugging / exploration** → Before running expensive queries on billions of rows, you can quickly peek at a small portion of the data.
- **Schema validation** → See if data types and content look correct.
- **Performance** → Extremely fast (as shown: only 0.117 seconds to fetch 20 rows).

---

#### 70.1.2 Layman's Explanation

Think of **limit-based sampling** as **flipping through the first few pages of a huge book**:

- Instead of reading all 1,000 pages, you just look at the first 20 pages.

- It gives you a **quick preview** of what the book looks like (writing style, formatting, chapter names).
- You don't need to read the whole book to get a basic idea.

In the same way:

- Hive's LIMIT 20 just shows the **first 20 records** from the table.
  - This is the fastest and simplest way to **check your data**.
- 

### In short:

- **Techy way:** LIMIT returns the first N rows, useful for debugging or previewing without a full scan.
- **Everyday:** It's like reading the first few pages of a book to get a sense of the content instead of reading the entire book.

this screenshot shows **Random Sampling with RAND()** in Hive. Let's expand technically and then simplify.

---

## 70.2 Random Sampling using RAND()

```
select * from airports order by RAND() limit 20;
```

### 70.2.1 What it does:

- The **RAND() function** generates a random number for each row in the table.
- ORDER BY RAND() **shuffles the table randomly**.
- LIMIT 20 then selects 20 rows from this shuffled order.
- Every time you run this query, you get **different random rows**.

### 70.2.2 Why it's important:

- **Unbiased sampling** → Unlike LIMIT, which always gives the first rows, this ensures randomness.
- **Broad diversity** → Useful when you want a representative subset of the dataset.
- **Applications:**
  - Data exploration with variety.
  - Creating test datasets.
  - Machine learning training/testing (random subsets are crucial).

---

### 70.2.3 Layman's Explanation

Think of this as **drawing random names out of a hat**:

- You have a giant box of slips (airport records).

- Instead of always taking the **first 20 slips from the top** (like with LIMIT), you **shake the box** and then grab 20 slips at random.
- Next time you shake the box, you'll probably pull out different slips.

This way, you don't always see the same set of records - you get a **fresh, unbiased mix** each time.

---

### In short:

- **Techy way:** ORDER BY RAND() LIMIT n gives you a random subset of rows, useful for unbiased sampling and model training/testing.
- **Everyday way:** It's like shaking a box of slips and picking 20 at random instead of just reading the first 20 slips from the top.

## 70.3 Sampling by Number of Rows (TABLESAMPLE)

```
select * from planes tablesample (30 rows);
```

### 70.3.1 What it does:

- Retrieves **30 sampled rows** from the planes table using Hive's **TABLESAMPLE clause**.
- Unlike LIMIT, this does **not always return the same 30 rows** each time you run it.
- It's designed to provide a **representative subset** of the table instead of a fixed slice.

### 70.3.2 Key Differences from LIMIT:

- **LIMIT** → Always gives the first  $N$  rows. Results are deterministic.
- **TABLESAMPLE** → Randomized selection; results vary with each execution.

### 70.3.3 Why use it:

- For **testing and debugging** with varied subsets.
- When you want **different portions of data** across multiple runs.
- Useful in **large datasets** where analyzing the entire dataset is costly.

---

### 70.3.4 Layman's Explanation

Think of it as **taking 30 spoons of soup from different spots in the pot**:

- If you use LIMIT 30, you're always scooping from the **top of the pot** → you might only taste the top layer.

- With TABLESAMPLE(30 rows), Hive scoops **30 spoons randomly from different places in the pot** → giving you a better overall taste.

So:

- LIMIT = predictable, but not diverse.
  - TABLESAMPLE = more random, better variety.
- 

### In short:

- **Techy way:** TABLESAMPLE( $n$  rows) fetches a randomized sample of rows, unlike LIMIT which always fetches the first  $N$  rows.
- **Everyday way:** It's like tasting from different parts of a soup pot instead of just sipping from the top every time.

## 70.4 Bucket Sampling from a Bucketed Table

**Query:**

```
select * from flight_bkt tablesample(bucket 2 out of 4) limit 20;
```

### 70.4.1 What it does:

- The table `flight_bkt` was earlier defined with **4 buckets**.
- This query says: “**Select only data from bucket 2 out of the 4 buckets.**”
- Hive uses the **hash function** on the bucketing column (e.g., month) to assign rows to buckets.
- Each bucket holds an approximately equal share of rows.

### 70.4.2 Why it’s important:

- **Efficiency:** Instead of scanning all buckets, Hive only reads from one bucket ( $\frac{1}{4}$  of the data in this case).
- **Consistency:** Because bucketing is deterministic (hash-based), the same row always ends up in the same bucket. That means bucket 2 always holds the same slice of data.
- **Use cases:**
  - Debugging or testing queries on smaller, representative slices.
  - Distributed workloads (different jobs can process different buckets in parallel).
  - Faster analytics when you only care about a portion of data.

### 70.4.3 Layman's Explanation

Think of a **library with 4 drawers** for storing records:

- Every book (row of data) is assigned to a drawer (bucket) based on a rule (hash of the month).
- Drawer 1 → Jan–Mar, Drawer 2 → Apr–Jun, Drawer 3 → Jul–Sep, Drawer 4 → Oct–Dec (just an example).

Now, if you only want to check Drawer 2:

- You don't need to open all 4 drawers.
- You just look in Drawer 2 and grab 20 books.

This saves time, reduces effort, and still gives you a consistent “slice” of the dataset every time.

---

### In short:

- **Techy way:** TABLESAMPLE(bucket x out of y) retrieves rows from a specific bucket in a bucketed table, reducing data scanned and enabling distributed processing.
- **Everyday way:** It's like opening only one drawer in a filing cabinet instead of checking all of them - faster and more focused.

## 70.5 Sampling with Fixed Data Volume

```
select * from flight_bkt tablesample(1M);
```

### 70.5.1 What it does:

- Retrieves approximately **1 million rows** from the flight\_bkt table.
- The 1M stands for **1 million rows**, not exactly, but Hive attempts to approximate that many rows.
- This is different from LIMIT because:
  - LIMIT always gives exactly N rows.
  - TABLESAMPLE(1M) distributes the read across blocks/buckets and then approximates 1M rows.

### 70.5.2 Why it's useful:

- For **large datasets** (billions of rows), fetching the whole table would be expensive and unnecessary.
- This gives you a **large enough subset** to run meaningful analytics without full cost.
- Strikes a **balance** between small random sampling and full dataset scans.

#### Use cases:

- Training/testing machine learning models on a **representative bulk subset**.
- Stress-testing queries without running them on the entire dataset.

- Batch exploration of large datasets.
- 

### 70.5.3 Layman's Explanation

Think of this as **taking a scoopful from a giant rice sack**:

- If the sack has **100 million grains** (rows of data), you don't want to count or look at every grain.
- But you also don't want just 20 grains (LIMIT 20), because that's too small to understand the whole sack.
- Instead, you scoop **1 million grains** → still manageable, but large enough to see patterns.

So:

- You don't waste time with the full sack.
  - You don't under sample.
  - You get a **good balance between size and efficiency**.
- 

### In short:

- **Techy way:** TABLESAMPLE(1M) retrieves ~1 million rows, giving a manageable but sizable subset for testing or analysis on massive datasets.
- **Everyday way:** It's like scooping out a cup of rice from a giant sack instead of checking every grain - enough to understand the whole sack, but not overwhelming.

## 70.6 Percentage Sampling

```
select * from carriers tablesample(1 percent);
```

### 70.6.1 What it does:

- Retrieves **1% of the entire carriers table.**
- Unlike fixed row sampling, percentage sampling always adjusts based on table size:
  - If the table has 100,000 rows → returns ~1,000 rows.
  - If the table has 10 million rows → returns ~100,000 rows.
- Ensures you always get a **proportional sample**, regardless of dataset growth.

### 70.6.2 Why it's useful:

- **Scalability:** Works well in production systems where table sizes can vary dramatically.
- **Consistency:** You always analyze a fixed *proportion*, not an arbitrary count.
- **Use cases:**
  - Trend analysis across datasets of different sizes.
  - Testing queries in environments where table sizes may grow.
  - Quick exploration when a small but proportional view is needed.

### 70.6.3 Layman's Explanation

Think of this as **taking 1% of a jar of candy**:

- If the jar has 100 candies, you pick 1.
- If the jar has 1,000 candies, you pick 10.
- If the jar has 10,000 candies, you pick 100.

No matter how the jar size changes, you're always sampling **the same proportion**.

This is very useful when you want to **compare trends** between jars (datasets) of different sizes.

---

#### In short:

- **Techy way:** TABLESAMPLE(percent) retrieves a proportional subset of data, ensuring consistency across different table sizes.
- **Everyday way:** It's like always taking 1% of a candy jar - whether the jar is small or huge, you always get a fair proportion.

# 71 Joins

## 1. What are Joins in Hive?

- A **JOIN** combines rows from two (or more) tables based on a related column (key field).
- Example: joining a flights table and a carriers table using carrier\_id.
- This is the same concept as SQL joins but adapted for Hive's distributed system.

### 71.1 Why are joins expensive in Hive?

- Hive runs on top of Hadoop (MapReduce or Tez/Spark engines).
- Joining two big tables requires **shuffling data across nodes** in the cluster so matching rows can meet.
- This **data movement (network I/O)** is costly and time-consuming.

### 71.2 Optimized Joins in Hive:

- **MapJoin (Broadcast Join):**
  - Used when one table is small enough to fit in memory.
  - The small table is broadcasted to all nodes, so Hive doesn't shuffle large data.
  - Much faster.
- **Bucket Map Join:**
  - Works when both tables are bucketed on the join key using the same number of buckets.

- Since rows with the same key are already in the same bucket, Hive only needs to join matching buckets (no global shuffle).
- Efficient for large datasets.

### 71.3 Key benefit of optimized joins:

- They **reduce data shuffling**, leading to **lower compute costs and faster execution**.
- 

### 71.4 Layman's Explanation

Think of this like **combining two lists of people**:

- You have one list of **students** and another list of their **grades**.
- A join means you're matching each student with their grade.

Now, imagine these lists are spread across **different classrooms (cluster nodes)**:

- **Normal Join:**
  - You'd ask each classroom to **send all their papers to one place**.
  - Very slow and messy.
- **MapJoin (small + big table):**
  - If one list is small (say a short list of grade codes), you just **photocopy it and give it to every classroom**.
  - Then each classroom can do the matching locally. Much faster.

- **Bucket Map Join (both tables bucketed):**
    - Imagine you already sorted both lists into drawers (buckets) by student ID.
    - To match them, you only open the **same drawer in both lists**.
    - No need to mix everything together.
- 

#### In short:

- **Techy way:** Normal joins in Hive are costly because of data shuffling, but MapJoin and Bucket Map Join reduce shuffle, saving time and resources.
- **Everyday way:** Instead of dumping two messy lists together, either **copy the small one everywhere** (MapJoin) or **organize both lists into matching drawers** (Bucket Map Join) before combining.

## 72 INNER JOIN

Query:

```
SELECT COUNT(1)
FROM carriers c
JOIN flight_bkt f
ON c.code = f.uniquecarrier
WHERE f.year = 2008
AND f.month = 1;
```

## 72.1 What it does:

- Performs an **INNER JOIN** between:
  - carriers table (alias c)
  - flight\_bkt table (alias f)
- Join condition: `c.code = f.uniquecarrier`
- Filters the joined rows to only include:
  - `year = 2008`
  - `month = 1`
- Returns a **count of matching rows** (number of flights in Jan 2008 with matching carrier codes).

## 72.2 Internally in Hive:

1. **Read data** from both tables in parallel.
2. **Identify the join key** (`code = uniquecarrier`).
3. **Partition the data** by the join key → rows with the same carrier code are grouped together.
4. **Shuffle data across cluster nodes** so that matching rows from both tables meet in the same reducer.
5. Emit **only matching rows** as output.

## 72.3 Key Point:

- **INNER JOIN** → Only includes rows that have a match in **both tables**.
  - Rows without a match are excluded.
- 

## 72.4 Layman's Explanation

Think of this like **matching students with their grades**:

- You have one list of **students (carriers)**.
- You have another list of **exam results (flights)**.
- You only want rows where there is a **match**:
  - If a student exists in both the student list *and* the exam results, you keep it.
  - If a student is missing from either list, ignore it.

So here:

- Carriers are matched with flights based on their **carrier code**.
- We only count the flights from **Jan 2008** that have a valid carrier in the carriers table.

👉 The result is the number of flights in Jan 2008 where both the **carrier code** and the **flight record** exist.

---

### In short:

- **Techy way:** An INNER JOIN returns only rows with matching keys in both tables, after shuffling and grouping by the join key across the Hive cluster.
- **Everyday way:** It's like matching students with exam results - you only keep the ones who appear in *both* lists.

## 73 RIGHT OUTER JOIN

### 1. Query:

```
SELECT COUNT(1)
FROM carriers c
RIGHT OUTER JOIN flight_bkt f
ON c.code = f.uniquecarrier
WHERE f.year = 2008
AND f.month = 1;
```

#### 73.1 What it does:

- Joins the **carriers** table (c) and **flight\_bkt** table (f) on code = uniquecarrier.
- Since it's a **RIGHT OUTER JOIN**:
  - All rows from the **right table** (flight\_bkt) are returned.
  - If there's no match in the **left table** (carriers), the carrier fields are filled with **NULL**.
- Filters rows to only **flights from Jan 2008**.
- Returns the count of these rows.

#### 73.2 Internally in Hive:

- Works similarly to an **INNER JOIN**, but during the merge phase:
  - Hive **keeps all rows from the right table (flight\_bkt)**.
  - Matching rows from the left table (carriers) are included.
  - Non-matching rows → carrier values set to **NULL**.

- Still requires **shuffling and sorting** of data across cluster nodes.
- 

### 73.3 Layman's Explanation

Imagine you're trying to **match students (carriers) with exam results (flights)**:

- **Right Outer Join rule:**
  - Keep **all exam results** (flights).
  - If the student name (carrier) exists in the student list, include it.
  - If not, leave the student info blank (**NULL**).

👉 Example:

- Flight record shows carrier "XYZ".
- If "XYZ" exists in the carriers list → include its details.
- If "XYZ" does NOT exist in the carriers list → still include the flight, but carrier info is left empty.

So with **RIGHT OUTER JOIN**:

- You **never lose any flights** (right table).
  - You only lose carrier details when they don't exist.
- 

**In short:**

- **Techy way:** RIGHT OUTER JOIN keeps all rows from the right table, adds matches from the left, and fills missing left values with NULL.

- **Everyday way:** It's like keeping all exam results, even if the student isn't on the official student list - you just leave the student details blank.

## 74 Auto-Configured Map Join

### 74.1 Query:

```

1. SELECT /*+ MAPJOIN(c) */ COUNT(1)
2. FROM flight_bkt f
3. JOIN carriers c
4. ON c.code = f.uniquecarrier
5. WHERE f.year = 2008
6. AND f.month = 1;

```

### 74.2 What it does:

- The **/\*+ MAPJOIN(c) \*/** hint tells Hive:
  - Treat the carriers table (c) as a **small table**.
  - Instead of shuffling it across the cluster, **broadcast it to all mappers**.

### 74.3 Internally in Hive:

1. Hive loads the **small table (carriers)** fully into memory on each mapper.
2. As the large table (flight\_bkt) is scanned:
  - Each mapper matches rows locally with the in-memory carriers table.
3. **No shuffle phase** is needed → avoids costly data movement.
4. The join is completed in the **map phase**, making it much faster.

#### 74.4 When to use:

- When one table is **small** (fits into memory, typically <25MB).
  - Especially useful for **dimension/reference tables** like carriers, countries, or product categories.
- 

#### 74.5 Layman's Explanation

Think of this like **a teacher handing out a class list**:

- Imagine you have **millions of exam answer sheets (flights)** spread across classrooms.
- You also have a **small list of student names (carriers)**.
- Instead of calling all classrooms to one hall to match sheets with names (**expensive shuffle**), the teacher just **photocopies the small student list and gives a copy to each classroom**.
- Now, each classroom can do the matching locally - much quicker!

So:

- **MAPJOIN saves time** by avoiding the “everyone come together” shuffle.
  - Instead, it **brings the small list everywhere**.
-

## In short:

- **Techy way:** MAPJOIN broadcasts a small table to all mappers, eliminating shuffle and reducing join cost.
  - **Everyday way:** It's like photocopying a short student list and distributing it to every classroom, instead of dragging all exam papers into one room for checking.
- 

## 75 Bucket Map Join

### 75.1 Settings:

- ```
1. SET hive.auto.convert.join = true;  
2. SET hive.optimize.bucketmapjoin = true;
```

These enable Hive to automatically convert eligible joins into **Bucket Map Joins** if conditions are met.

75.2 What is a Bucket Map Join?

- A **Bucket Map Join** is used when **both tables are bucketed on the same join key**.
- Hive skips the expensive shuffle step by directly joining **matching buckets**.

75.2.1 Conditions:

- Both tables must be **bucketed and sorted on the join key**.
- Bucket counts must match or be multiples:
 - Example: Table A has 8 buckets; Table B has 4 → Hive can still align them.
- Consistent schema and sorting must exist.

75.3 Internally in Hive:

1. Hive checks if both tables are bucketed on the **join key**.
2. Instead of shuffling rows across the cluster, Hive pairs **bucket N of Table A with bucket N of Table B**.
3. Since data is already organized, the join is **local and direct**.
4. This results in **super-fast joins** with very little network overhead.

75.4 Summary:

- **Best for large datasets** that are pre-bucketed.
 - Saves resources by avoiding shuffle.
 - Ideal for **recurring ETL pipelines** where data is consistently bucketed and sorted.
-

75.5 Layman's Explanation

Imagine two libraries:

- One library has books sorted into **drawers by subject (buckets)**.
- Another library also sorts its books into **drawers using the same subjects (same buckets)**.

Now, if you want to **combine both libraries' books by subject**:

- With a normal join → You'd have to **gather all books, reshuffle them by subject**, then merge. Very slow.

- With a bucket map join → You just **open drawer 1 in both libraries and merge them directly**, then move on to drawer 2, 3, etc.

👉 No reshuffling, just matching drawers one-to-one. Much faster!

In short:

- Techy way:** Bucket Map Join leverages pre-bucketed data on the same key to directly join matching buckets, avoiding shuffle and improving performance.
 - Everyday way:** It's like merging two filing cabinets that already use the same drawer system - you just pair drawer with drawer instead of re-sorting everything.
-

76 Complex Multi-Table JOIN (LEFT JOIN Example)

76.1 Query:

```
1. SELECT flightnum, year, month, dayofmonth, dayofweek,
2.       c.description, f.tailnum, p.aircraft_type,
3.       CONCAT(a.airport, ' ', a.city, ' ', a.state, ' ', a.country) origin,
4.       CONCAT(b.airport, ' ', b.city, ' ', b.state, ' ', b.country) dest
5. FROM flights f
6. LEFT JOIN carriers c ON f.uniquecarrier = c.code
7. LEFT JOIN airports a ON f.origin = a.iata
8. LEFT JOIN airports b ON f.dest = b.iata
9. LEFT JOIN planes p   ON p.tailnum = f.tailnum;
```

76.2 What it does:

- **Base table:** flights f (all flight records).
- **Joins:**
 - carriers c → Adds airline name (description) for each flight.
 - airports a → Adds **origin airport details**.
 - airports b → Adds **destination airport details**.
 - planes p → Adds aircraft type (aircraft_type).
- **LEFT JOIN** ensures:
 - All rows from flights are preserved.
 - If a match doesn't exist in the joined table, Hive inserts **NULL**.

76.3 Internally in Hive:

- Hive processes joins **from left to right**, always keeping all rows from flights.
 - Each table is joined **on its key** (e.g., f.origin = a.iata).
 - If a match is missing (say, origin airport not found in the airports table), Hive fills with **NULLs**.
 - Hive may optimize small reference tables (carriers, airports, planes) using **MapJoin** if they fit in memory.
-

76.4 Layman's Explanation

Think of this as **enriching a basic flight log with extra details**:

- You start with a **list of flights** (flight numbers, dates, carriers, tail numbers).
- Then:
 - You look up the **airline name** (from carriers).
 - You look up the **origin airport details** (from airports).
 - You look up the **destination airport details** (same airports table, joined again).
 - You look up the **aircraft type** (from planes).

Because it's a **LEFT JOIN**:

- Every flight stays in the list, even if some details (like missing plane type or airport info) aren't available → those fields will just be blank (NULL).

👉 So, this query builds a **rich, descriptive flight record** by pulling together information from four different tables.

In short:

- **Techy way:** A multi-table LEFT JOIN enriches a base table (flights) with details from multiple reference tables (carriers, airports, planes), preserving all flight records and filling in NULLs where matches are missing.

- **Everyday way:** It's like starting with a flight schedule and then looking up the airline, airport details, and plane type to make the record complete - but keeping the flight listed even if some details are missing.

77 VIEWS

77.1 What is a View in Hive?

- A **view** is a **virtual table** based on the result of an SQL query.
- Unlike a table, a view does **not store data**. Instead, it stores the **SQL query definition** in the **Hive Metastore**.

How it works:

1. When you create a view:

```
1. CREATE VIEW flight_summary AS  
2. SELECT year, month, COUNT(*) AS total_flights  
3. FROM flights  
4. GROUP BY year, month;
```

Hive saves this SQL in the **Metastore**.

2. When you query the view:

```
1. SELECT * FROM flight_summary;
```

Hive **expands the saved query** (the SELECT ... GROUP BY) and executes it in real time.

77.2 Key Characteristics:

- Views are **not materialized**:
 - No data is stored physically unless you explicitly persist it (e.g., with INSERT OVERWRITE).
- Views act like **shortcuts** to commonly used queries.

- **Performance** depends on the complexity of the underlying SQL.
-

77.3 Layman's Explanation

Think of a Hive **view** as a **saved recipe**:

- You don't actually cook and store the dish (data).
- You just write down the recipe (the query).
- Whenever you want the dish, you follow the recipe again (Hive runs the underlying SQL).

So:

- A **table** = the actual cooked food stored in the fridge.
- A **view** = the recipe you saved in your cookbook.

👉 Example:

If you often run a query to count flights per month, instead of retyping it every time, you save it as a **view** called `flight_summary`.

Next time, you just say:

1. `SELECT * FROM flight_summary;`

and Hive automatically runs the recipe for you.

In short:

- **Techy way:** A view is a virtual table that stores query definitions, not data. Hive expands the SQL at runtime.
- **Everyday way:** A view is like saving a recipe - you don't store the food, just the instructions. You cook (query) it fresh each time you need it.

78 CREATE VIEW flightsview AS SELECT...

Command:

```
1. CREATE VIEW flightsview AS  
2.   SELECT flightnum, year, month, ...,  
3.         CONCAT(a.airport, ' ', a.city, ' ', a.state, ' ', a.country) origin,  
4.         CONCAT(b.airport, ' ', b.city, ' ', b.state, ' ', b.country) dest  
5.   FROM flight_bkt f  
6.   LEFT JOIN carriers c ON f.uniquecarrier = c.code  
7.   LEFT JOIN airports a ON f.origin = a.iata  
8.   LEFT JOIN airports b ON f.dest = b.iata  
9.   LEFT JOIN planes p   ON p.tailnum = f.tailnum;
```

What this does:

1. **CREATE VIEW flightsview AS ...**
 - Creates a **virtual table** called flightsview.
 - The view doesn't store data itself - it just saves the SQL query.
2. **Selected columns:**
 - Basic flight info: flightnum, year, month, etc.

- Human-readable **origin and destination strings**:
 - CONCAT combines airport, city, state, and country.
- Carrier info: from carriers table.
- Plane info: from planes table.

3. LEFT JOINs:

- carriers: adds airline description.
- airports a: adds origin airport details.
- airports b: adds destination airport details.
- planes: adds aircraft type.

4. End result:

- The **entire SQL query** is stored as the definition of the view.
 - When you query flightsview, Hive expands and runs this SQL on the fly.
-

78.1.1 Layman's Explanation

Think of this like **saving a flight dashboard template**:

- Normally, to see a full flight record, you'd have to:
 - Look up the flight number.
 - Match it to the airline.
 - Match it to the origin and destination airports.

- Match it to the plane type.
- Format everything neatly with readable names.
- Instead of typing all of that SQL every time, you **save it once as a view** called `flightsview`.

👉 From then on, you can simply run:

```
1. SELECT * FROM flightsview;
```

and Hive will automatically fetch and join everything behind the scenes.

It's like creating a **shortcut button** for a very complex query.

78.1.2 In short:

- **Techy way:** `CREATE VIEW flightsview AS ...` stores a query definition that joins multiple tables and formats flight details, without storing data.
- **Everyday way:** A view is like saving a ready-made **flight dashboard template** - one click (querying the view) gives you all the details, instead of rebuilding the query every time.

79 SELECT * FROM flightsview LIMIT 10

Command:

```
1. SELECT * FROM flightsview LIMIT 10;
```

1. What it does:

1. SELECT * FROM flightsview

- Retrieves all columns from the **virtual table** flightsview.
- Remember: a view is just a stored query, so Hive expands the underlying SQL (with joins and formatting) and runs it.

2. LIMIT 10

- Restricts the output to just **10 rows**.
- Useful for testing, debugging, or quick inspection.

79.1 Purpose:

- This step verifies that:
 - The **view was created correctly**.
 - The **joins are working** (carriers, airports, planes).
 - The **formatting** (like CONCAT origin and destination) behaves as expected.
- Since the view is **not materialized**, Hive runs the underlying query each time, so performance depends on that SQL.

79.2 Layman's Explanation

Think of this like **testing your saved recipe**:

- You wrote down a detailed recipe (flightsview) that combines ingredients from different sources (carriers, airports, planes).
- Now, instead of cooking the full meal (all rows), you **cook just a small sample - 10 servings** - to make sure the recipe works.

👉 If the sample comes out right, you know the recipe is correct.

So:

- flightsview = saved recipe.
 - SELECT * ... LIMIT 10 = cooking a small test batch to check if everything's good before scaling up.
-

79.3 In short:

- **Techy way:** The query expands and executes the view definition, returning 10 rows for quick inspection.
- **Everyday way:** It's like testing a recipe by making just a small portion instead of the whole dish, to confirm it works.

80 SET hive.cli.print.header=true

Command:

```
1. SET hive.cli.print.header=true;
```

1. What it does:

- Configures the Hive CLI to **print column headers** in the output.
- Applies only to the **current session** (you'd need to re-run it next time you start Hive).
- Very useful when:
 - Exporting results to **CSV** or text files.
 - Sharing query outputs where **column names** are needed for clarity.

80.1 Example:

Without this setting:

```
1. N919ME 2007 1 NULL ...
```

With the setting enabled:

```
1. flightnum year month description ...
2. N919ME 2007 1 NULL ...
```

80.2 Layman's Explanation

Think of this as **adding labels to your spreadsheet**:

- Without headers → You just see rows of numbers/values, but you don't know what each column means.

- With headers → You see column names like “Flight Number,” “Year,” “Month,” making it much easier to understand.

👉 It's like printing a table with proper headings so others don't get lost.

80.3 In short:

- **Techy way:** SET hive.cli.print.header=true; tells Hive to include column names in query output for the current session, making exports and debugging easier.
- **Everyday way:** It's like labeling the columns in an Excel sheet before sharing it - so people know what each value means.

```
81 SELECT * FROM flightsview WHERE year=2008 AND  
month=1 LIMIT 10
```

Command:

1. `SELECT *`
2. `FROM flightsview`
3. `WHERE year = 2008 AND month = 1`
4. `LIMIT 10;`

1. What it does:

1. `SELECT * FROM flightsview`

- o Retrieves all columns from the virtual view `flightsview`.

2. `WHERE year = 2008 AND month = 1`

- o Applies a filter: only returns rows where the year is 2008 and the month is January.
- o This shows how you can treat a **view just like a regular table** - applying filters, joins, or aggregates.

3. `LIMIT 10`

- o Restricts the result to just 10 rows.
- o Prevents scanning/returning the full dataset.

81.1 Purpose:

- Helps **slice specific data segments** (e.g., January 2008 flights).
- Useful for **debugging, performance testing, or sampling large datasets**.
- Shows that views are fully **queryable objects**, not just static definitions.

81.2 Layman's Explanation

Think of this like using a **filter on your saved flight dashboard**:

- You already built flightsview (the recipe/shortcut for full flight details).
- Now, instead of seeing *all flights* ever, you say:
 - “Show me **only January 2008 flights**.”
- And since you don’t want the full list, you limit it to just **10 sample rows**.

👉 It’s like filtering an Excel sheet by date (January 2008) and then scrolling down to just the first 10 rows for a quick peek.

81.3 In short:

- **Techy way:** The query expands the flightsview definition, applies a filter (year=2008 AND month=1), and limits results to 10 rows.
- **Everyday way:** It’s like applying a date filter on a saved dashboard and looking at only the first few rows for a quick preview.

82 INSERT OVERWRITE DIRECTORY

Command:

```
1. INSERT OVERWRITE DIRECTORY '/user/Asiema/data/flights/View'  
2. ROW FORMAT DELIMITED  
3. FIELDS TERMINATED BY '|'  
4. SELECT *  
5. FROM flightsview  
6. WHERE year = 2008 AND month = 1  
7. LIMIT 10;
```

1. What it does:

- a) **INSERT OVERWRITE DIRECTORY ...**
 - Writes the query result directly into the specified **HDFS directory**.
 - Overwrites any existing data in that directory.
- b) **ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'**
 - Sets the **output format** to delimited text.
 - Here, the pipe symbol | is used as a column separator.
- c) **SELECT * FROM flightsview WHERE year=2008 AND month=1 LIMIT 10;**
 - Runs the query against the view.
 - Retrieves **10 rows of flights from January 2008**.
 - The result is written to HDFS in the chosen format.

Key Points:

- The output is a **flat file** (not a Hive table).
 - Useful for **exporting Hive results** into files that can be:
 - Processed by other tools (e.g., Spark, Hadoop MapReduce, Python, R).
 - Shared with non-Hive users.
-

2. Layman's Explanation

Think of this as **saving a filtered report from Hive into a file**:

- You ask Hive: "Give me January 2008 flights, just 10 rows."
- Instead of just displaying them on screen, you **save them into a file in HDFS**.
- You also decide that values in the file should be separated by | (like using commas in a CSV).

👉 It's like exporting a filtered Excel sheet into a **text file** that other programs can read.

82.1 In short:

- **Techy way:** This command runs a query on a Hive view, exports the results into HDFS as a delimited text file (pipe-separated), and overwrites existing content in that directory.
- **Everyday way:** It's like taking a filtered slice of your Hive data and saving it as a neat text file you can open or share, ready for use in other tools.

83 cd /disk/hive/apache-hive-3.1.2-bin/bin

Command:

```
1. cd /disk/hive/apache-hive-3.1.2-bin/bin  
2.  
3. hive -e "use airlines;  
4. set hive.cli.print.header=true;  
5. select * from flightsview where year=2008 and month=1 limit 10" \  
6. | sed 's/[\\t]/|/g' > /disk/data/Flight_View.csv
```

1. What it does:

- a) **hive -e "..."**
 - Runs HiveQL commands directly from the terminal (batch mode), without entering the Hive CLI manually.
- b) **use airlines;**
 - Switches to the airlines database.
- c) **set hive.cli.print.header=true;**
 - Ensures the output includes column headers.
- d) **select * from flightsview where year=2008 and month=1 limit 10;**
 - Runs the query on the view flightsview.
 - Retrieves 10 rows for January 2008.
- e) **| sed 's/[\\t]/|/g'**
 - The Hive output is tab-delimited by default.

- This command replaces tabs (\t) with pipes (|) using **sed** (a Unix text processing tool).
- Can be modified to replace tabs with commas (,) for CSV.

f) > /disk/data/Flight_View.csv

- Redirects the final output into a local CSV file (Flight_View.csv).
 - This saves the query result outside Hive for sharing, offline use, or further processing.
-

83.1 Layman's Explanation

Think of this as **exporting your Hive query into an Excel/CSV file you can share:**

- You run a Hive query that pulls 10 flights from January 2008.
- Normally, Hive just shows the results on screen.
- But here:
 - You **add column headers** (so the file is readable).
 - You **format the columns** using | (or commas).
 - You **save the result as a file** on your computer (Flight_View.csv).

👉 It's like downloading a filtered dataset from a web app and saving it as a CSV file for reporting or emailing.

83.2 In short:

- **Techy way:** This command executes a Hive query in batch mode, formats the output using Unix tools, and exports the result as a CSV file to local disk.
- **Everyday way:** It's like exporting part of your Hive dataset into a neat spreadsheet you can open, share, or analyze offline.

84 ALTER VIEW flightsview RENAME TO flights_view

Command:

```
1. ALTER VIEW flightsview RENAME TO flights_view;
```

1. What it does:

a) Renames the view

- Changes the name from flightsview → flights_view.

b) Underlying SQL logic stays the same

- The query definition that was originally used to create flightsview does not change.
- The view still points to the same query.

c) Metadata update only

- The rename operation only changes the entry in the **Hive Metastore**.
- No data is modified, deleted, or reprocessed.

84.1 Layman's Explanation

Think of this like **renaming a folder or file on your computer**:

- If you rename MyFlights.docx to Flights_Report.docx, the **contents of the document don't change.**
- Only the **label (name)** is updated so you can identify it better.

👉 Similarly, in Hive:

- flightsview → flights_view
- The actual SQL query (recipe) behind the view stays exactly the same.

This is especially useful for:

- Fixing typos in names.
- Following consistent **naming conventions** (e.g., using underscores).
- Making the schema easier to understand.

84.2 In short:

- **Techy way:** ALTER VIEW ... RENAME updates only the metadata in Hive Metastore, changing the view name but not the query definition.
- **Everyday way:** It's like renaming a file on your computer - the file's contents remain the same.

85 DROP VIEW flights_view

Command:

```
1. DROP VIEW flights_view;
```

1. What it does:

a) Deletes the view definition

- o Removes the view flights_view from the Hive **Metastore**.
- o Only the SQL definition stored in metadata is removed.

b) No impact on base tables or data

- o The underlying tables (flights, carriers, airports, etc.) remain intact.
- o The actual data is **not deleted**.

c) Permanent action

- o Once dropped, the view cannot be queried again unless recreated with a new CREATE VIEW.
-

85.1 Layman's Explanation

Think of this as **removing a shortcut, not the real file**:

- You had a saved shortcut (the view flights_view) that pointed to a recipe for fetching flight details.
- Dropping the view simply **deletes the shortcut**.
- The actual ingredients (the base tables with flight data) are still safe and untouched.

👉 You'd do this for **cleanup**, when a view is no longer useful or has been replaced with something better.

85.2 In short:

- **Techy way:** DROP VIEW removes the SQL definition of a view from Hive Metastore without affecting the underlying data.
- **Everyday way:** It's like deleting a desktop shortcut - the real files (tables) are still there, only the quick link is gone.

86 OPTIMIZATION

- The main goal of Hive optimization is to ensure **efficient resource usage**:
 - Reduce **query execution time**.
 - Minimize **computational cost**.
 - Lower **storage usage**.
- Hive queries can be slow or resource-heavy because Hive translates SQL into MapReduce (or Tez/Spark jobs). Optimization helps make these jobs more efficient.
- The document (optimization.txt) contains **commands and techniques** that can be applied across queries, such as:
 - **Bucketing & Partitioning** → reduce scanned data.
 - **MapJoin & Bucket Map Join** → reduce shuffle costs.
 - **Sampling** → avoid scanning entire datasets during exploration.
 - **Compression & File Formats** (ORC, Parquet) → save space and improve I/O performance.
 - **Indexing or CBO (Cost-Based Optimizer)** → let Hive decide the best query plan.
- It's important to **monitor performance** (execution time, CPU/memory usage, shuffle size) after applying optimizations to ensure they have the intended effect.

1. Layman's Explanation

Think of Hive optimization like **tuning a car for better mileage**:

- The car (Hive) will still take you from point A to point B (run your queries).
- But if it's not optimized, it may use **too much fuel (resources)**, take **longer routes (execution time)**, or wear out faster (high cost).

By applying optimizations:

- You **drive faster** (queries run quicker).
- You **use less fuel** (save memory and CPU).
- You **store more efficiently** (use better formats to save space).

👉 In simple terms: Optimization makes Hive queries **smarter, faster, and cheaper**.

86.1 In short:

- **Techy way:** Hive optimization involves using techniques like partitioning, bucketing, MapJoin, file formats, and cost-based optimization to improve execution efficiency and resource utilization.
- **Everyday way:** It's like tuning your car so it runs faster and uses less fuel - you reach your goal quicker without wasting resources.

87 COMPRESSION

1. Enable Compression for Output

```
1. SET hive.exec.compress.output=true;
```

- Compresses the **final query output** before writing to HDFS or external systems.
 - Saves storage and improves I/O performance.
-

87.1 Set Compression Codec

```
1. SET  
mapred.output.compression.codec=org.apache.hadoop.io.compress.SnappyCodec;
```

-- OR

```
1. SET  
mapred.output.compression.codec=org.apache.hadoop.io.compress.BZip2Codec;
```

- **Snappy** → Fast compression/decompression, lower ratio (good for analytics).
 - **BZip2** → Higher compression ratio, slower performance.
-

87.2 Enable Compression for Intermediate Data

```
1. SET hive.exec.compress.intermediate=true;
```

- Compresses **intermediate data** passed between mappers and reducers.
 - Reduces network I/O during MapReduce jobs.
-

87.3 Hive Execution Engine

Hive supports different backends to run queries:

- **mr** → Traditional MapReduce (slower, reliable, stable).
-

- **spark** → Uses Apache Spark (fast, in-memory).
 - **tez** → DAG-based execution, efficient for complex queries.
-

87.4 Check Current Engine

```
1. SET hive.execution.engine;
```

- Displays the execution engine currently in use.
-

87.5 Set Execution Engine

```
1. SET hive.execution.engine=tez;
```

-- or spark, or mr

- Switch between execution engines based on workload and cluster support.
-

87.5.1 Layman's Explanation

Think of Hive optimization like **choosing the best way to deliver packages (data)**:

Compression:

- **Without compression** → Packages are bulky, trucks (network) get overloaded, and delivery is slow.
- **With compression** → Packages are compact, trucks carry more, and delivery is faster.
 - Snappy = quick but not super tight packing.
 - BZip2 = tight packing but slower to wrap/unpack.

Execution Engine:

- Hive can choose different “delivery trucks” for executing queries:
 - **MapReduce (mr)** → Old reliable truck, slow but steady.
 - **Spark** → Sports car, loads data into memory, delivers very fast.
 - **Tez** → Smart delivery van that plans the best route (DAG), great for complex deliveries.

👉 By enabling compression and choosing the right engine, Hive queries run **faster, cheaper, and with less resource waste.**

87.5.2 In short:

- **Techy way:** Hive query optimization uses compression (Snappy, BZip2) and execution engines (mr, spark, tez) to minimize I/O and maximize efficiency.
- **Everyday way:** It's like packing your luggage efficiently (compression) and choosing the fastest vehicle (execution engine) to reach your destination quickly.

88 EXPLAIN

1. What is EXPLAIN?

- EXPLAIN shows the **execution plan** of a Hive query **without actually running it**.
 - It's used to analyze how Hive will process the query.
 - Helps identify performance bottlenecks before execution.
-

88.1 Key Use Cases:

1. **Compare performance** between queries (e.g., partitioned vs non-partitioned).
 2. **Understand query stages** (map, reduce, shuffle, etc.).
 3. **Debug issues** with joins, filters, or aggregations.
-

88.1.1 Example 1: Basic Usage

```
EXPLAIN SELECT year, COUNT(1) FROM flights GROUP BY year;
```

- Hive will show how it plans to scan the table, group records by year, and then count them.
 - Output explains which stages involve **Map** and **Reduce** operations.
-

88.1.2 Example 2: Partition Comparison

Without Partition

1. EXPLAIN SELECT year, AVG(depdelay)
2. FROM flights
3. WHERE year = 2008
4. GROUP BY year;

- Hive scans the **entire table**, then filters rows where year = 2008.

With Partition

1. EXPLAIN SELECT year, AVG(depdelay)
2. FROM flights_part
3. WHERE year = 2008
4. GROUP BY year;

- Hive scans **only the partition for year=2008**, skipping irrelevant data.
- This is much faster and more resource-efficient.

88.2 Layman's Explanation

Think of EXPLAIN like **asking Google Maps for directions before you start driving**:

- You don't actually drive yet (no query execution).
- Instead, you see the **route plan**:
 - Which roads you'll take.
 - How long each stage will take.

- Where traffic jams (bottlenecks) might occur.

👉 Similarly, in Hive:

- EXPLAIN tells you how the query will run - how much data will be scanned, how many stages (map/reduce) are involved, and whether optimizations (like partition pruning) are applied.

Example:

- **Without partitioning:** It's like searching the whole city to find one house.
 - **With partitioning:** It's like going straight to the correct neighborhood and only checking there.
-

88.3 In short:

- **Techy way:** EXPLAIN shows the query execution plan, useful for debugging and comparing optimizations like partitioning.
- **Everyday way:** It's like checking your route on Google Maps before starting the trip - you see the plan without driving yet.

89 Extended & Authorization Plan

Command:

```
EXPLAIN AUTHORIZATION
```

```
1. SELECT year, month, AVG(depdelay)  
2. FROM flight_ptd_clustd_raw  
3. WHERE year = 2004  
4. GROUP BY year, month  
5. ORDER BY month;
```

What It Does:

- **EXPLAIN AUTHORIZATION**
 - Similar to EXPLAIN EXTENDED, but also shows **authorization details**.
 - Helps verify **who has permission** to read/write tables involved in the query.
 - Useful in **multi-user Hadoop/Hive environments** where security matters.
- **Extended Plan** shows:
 1. **Logical Plan** – how Hive interprets the SQL.
 2. **Optimized Plan** – after applying query optimizations.
 3. **Physical Plan** – the actual MapReduce/Tez/Spark stages.

4. **Authorization Plan** – details about permissions required for tables/views.
-

Breakdown of the Query:

- **FROM flight_ptd_clustd_raw** → Reads from raw clustered flight data.
 - **WHERE year = 2004** → Filters rows for 2004 only.
 - **GROUP BY year, month** → Aggregates average departure delay per month.
 - **ORDER BY month** → Sorts results by month (1–12).
-

Layman's Explanation

Think of this like planning a **school project with permissions**:

1. First, you make a **plan of steps** (collect books, summarize chapters, create slides).
→ This is the **logical plan**.
2. Then, you figure out the **most efficient way** (maybe 3 students read books, 2 students make slides).
→ This is the **optimized plan**.
3. Finally, you **assign tasks to students** (who does what, in which order).
→ This is the **physical plan**.
4. But - you also need to check **who's allowed to access the library books**.
→ This is the **authorization plan**.

👉 In short:

- Normal EXPLAIN tells you **how Hive will run your query**.
 - **EXPLAIN AUTHORIZATION goes further** - it also checks **who's allowed to access the data** before running the query.
-

Why it matters:

- Ensures queries run efficiently **and securely**.
- Prevents errors where a user writes a query but doesn't have permission on one of the tables.

90 Data Processing and Transformation in Hive Using Azure VM

1. Sampling Techniques

- **Limit-based sampling** → LIMIT 20 (quick preview).
- **Random sampling** → ORDER BY RAND() (unbiased selection).
- **TABLESAMPLE (rows, buckets, data volume, percentage)** → flexible ways to grab subsets of data.
 - By number of rows.
 - By buckets (bucket 2 out of 4).
 - By fixed size (e.g., 1M rows).
 - By percentage (e.g., 1 percent).

2. Joins in Hive

- **Inner Join** → only matching records.
 - **Right Outer Join** → all from right table, matched with left.
 - **MapJoin (Broadcast Join)** → small table loaded into memory, avoids shuffle.
 - **Bucket Map Join** → works on bucketed tables, direct bucket-to-bucket match.
 - **Complex Multi-Table Join** → LEFT JOIN across multiple tables (carriers, airports, planes).
-

3. Views in Hive

- **Create View** → virtual table for reusable queries.
 - **Query View** → SELECT * FROM flightsview LIMIT 10.
 - **Export View Results** → INSERT OVERWRITE DIRECTORY
 - **Export with Hive CLI** → direct to .csv file.
 - **Rename View** → ALTER VIEW.
 - **Drop View** → DROP VIEW.
-

4. Optimization

- **Compression** → Snappy, BZip2 (saves storage, faster query execution).
-

- **Execution Engine** → mr, tez, spark.
 - **EXPLAIN** → shows query execution plan (before running).
 - **Extended & Authorization Plans** → deeper debugging.
 - **ANALYZE** → gathers table statistics for Cost-Based Optimizer (CBO).
 - **File Formats** → choose right one (SequenceFile, Parquet, Avro, ORC).
 - Parquet/ORC = columnar, best for analytics.
 - Avro = row-based, good for streaming/serialization.
-

With this structure, you've now got a **full end-to-end tutorial/manual** that explains **how to sample, join, optimize, and manage views in Hive - along with best practices for query efficiency inside Azure VM**

- **1. SequenceFile Format**

```
1. create external table flight_seq (
2.   year smallint, month tinyint, dayofmonth tinyint, ...
3. )
4. stored as sequencefile
5. location '/user/ubuntuhive/data/flights/flights_seq';
6.
```

Explanation:

- Binary key-value pair format, efficient for writing.
- Stored externally in HDFS.
- Good for backward compatibility.

Optimization Role:

- Quick writes.
 - Not great for analytics (no predicate pushdown, less compression).
-

- **2. Avro Format**

```
create external table flight_avro (...)
```

stored as avro

```
location '/user/ubuntuhive/data/flights/flights_avro';
```

Explanation:

- Row-based format.
- Supports **schema evolution** (adding/removing columns).

Optimization Role:

- Great for streaming pipelines & Kafka integration.
 - Slower for analytics than ORC/Parquet.
-

- **3. ORC (Optimized Row Columnar) Format**

```
create external table flight_orc (...)
```

stored as orc

```
location '/user/ubuntuhive/data/flights/flights_orc';
```

Explanation:

- Columnar format with **built-in indexing, compression, and predicate pushdown.**

Optimization Role:

- Excellent for analytics in Hive.
- High compression + fast query performance.
- Best for OLAP workloads.

- **4. Parquet Format**

```
create external table flights_pq (...)
```

stored as parquet

```
location '/user/ubuntuhive/data/flights/flights_pq';
```

Explanation:

- Columnar format widely used in **multi-system interoperability** (Spark, Hive, Impala, Presto).

Optimization Role:

- Ideal for **cross-platform use**.
- Supports predicate pushdown + efficient analytics.
- Similar to ORC but more versatile across ecosystems.

Comparison Summary

| Format | Storage | Best Use | Analytics | Compression | Schema |
|---------------------|---------|--------------------------------------|---|-------------|---|
| | | Case | Performance | | Evolution |
| SequenceFile | Binary | Quick writes,
legacy
Hadoop |  Poor | Low | Limited |
| Avro | Row | Streaming,
Kafka
pipelines |  Medium | Snappy |  Fully Supported |
| ORC | Column | Hive-native analytics
(OLAP) |  Excellent | High | Partial |
| Parquet | Column | Cross-platform
(Spark,
Presto) |  Excellent | High |  Supported |

👉 In short:

- Use **SequenceFile** only for backward compatibility.

- Use **Avro** for streaming & evolving schemas.
- Use **ORC** for **Hive-only heavy analytics**.
- Use **Parquet** if you want **flexibility across big data systems**.

91 Summary

Technical Explanation

- Learned **Hive fundamentals** and best practices for real-world data engineering.
 - Covered **VM provisioning** and memory allocation on **Azure VM**.
 - Step-by-step setup of **Big Data frameworks** (Hive + Hadoop on Azure).
 - Created tables in Hive after **loading datasets into HDFS**.
 - Understood **Hive architecture** and how **query optimization** improves performance.
 - Explored **Hive configurations**: partitions, bucketing, sampling, joins, and compression.
 - Discussed importance of **views** for reusing SQL queries efficiently.
 - Compared **file formats** (ORC, Parquet, Avro, SequenceFile) for performance, storage, and predicate pushdown.
 - Implemented **optimized read/write operations** using different file formats.
 - Followed best practices to **reduce execution time** and **cut storage costs**.
-

Layman's Explanation

Think of this training as learning how to manage a giant library:

- First, we set up the **library building** (Azure VM + Hive + Hadoop).

- We decided how to **store books** (partitions, buckets, file formats like ORC/Parquet).
- We learned shortcuts like **sampling** (reading just a few books instead of the whole library).
- We used **joins** to combine information from different sections of the library.
- We created **views** (like saved bookmarks) to quickly reuse searches.
- We compared **different types of shelves** (file formats) to see which ones save the most space and are easiest to search.
- Finally, we practiced **efficient searching techniques** so we don't waste time or storage.

In short: we turned Hive from a messy storage room into a **smart, efficient library** for big data.

92 Troubleshooting

92.1 Steps to Downgrade Hive

92.1.1 Remove/rename your current Hive

Move the Hive 4.1.0 directory so it doesn't interfere:

```
cd /disk/hive
```

```
mv apache-hive-4.1.0-bin apache-hive-4.1.0-bin.bak
```

This keeps a backup in case you need it.

92.1.2 Download Hive 3.1.2

Use the official Apache archives (3.1.2 is stable and works with Java 8):

```
wget https://archive.apache.org/dist/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz -P
```

```
/disk/hive
```

92.1.3 Extract Hive 3.1.2

```
cd /disk/hive
```

```
tar -xzf apache-hive-3.1.2-bin.tar.gz
```

You'll now have:

```
/disk/hive/apache-hive-3.1.2-bin
```

92.1.4 Update your environment variables

Open your `~/.bashrc` again:

```
nano ~/.bashrc
```

Find the Hive section and change:

```
export HIVE_HOME=/disk/hive/apache-hive-3.1.2-bin
```

```
export PATH=$PATH:$HIVE_HOME/bin
```

Save and reload:

```
source ~/.bashrc
```

92.1.5 Link Hive to Hadoop

Edit Hive config:

```
sudo nano $HIVE_HOME/bin/hive-config.sh
```

Add:

```
export HADOOP_HOME=/disk/hadoop/hadoop-3.4.1
```

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

92.1.6 Verify

Check versions:

```
hive --version
```

```
hadoop version
```

```
java -version
```

All should line up:

- Hive 3.1.2
- Hadoop 3.4.1
- Java 8