Petabyte scale storage-> Hadoop (MASSIVE)

In Memory Analysis-> Spark (FASTEST)

If the data is not here, bring it here!

Batch: event, process-> collects the data (GROUP or collection)-> sends this out!
SQL rows, NoSQL documents, files, any data collected over a period of time!

-> Stream that is large in size
-> SQL: Apache HIVE
-> NoSQL: Apache HBase
-> Gmail, Google marketing analytics, outlook!

Stream: CONTINUOUS movement (LITTLE OR NO COLLECTION!)
-> batch very small in size!
-> Data is incoming
-> processing streams-> Spark, Beam, Storm
-> Data is outgoing
-> building streams-> Kafka

BEAM-> Batch + StrEAM
-> SINGLE Program to handle both batch and streaming data!!!

Architecture Patterns:

1. **Lambda-> 2 separate batch and stream architectures**
2. **Kappa-> UNIFIED**

Cloud->
3 kind of services:

1. **New to cloud: IaaS: VIRTUAL MACHINES**
    1. **Migrating to the cloud for the first time- risk appetite is MINIMUM!**
    2. **Lift and shift**
    3. **VM Clusters: Azure- VM Scale Sets, EC2 Elastic Pools, Autoscaling groups**
    4. **IaaS advantage-> familiar environment, physical risk belongs to the cloud**
    5. **Disadvantage-> no CLOUD leverage!**
        1. **Wasting RAM, CPU and disk-> you will paying for it, even if not used!**
        2. **Everything else except the OS is still our problem!**

2. **Platform as a service**
    1. **All installations are DONE for us!**
    2. **A platform is what we are paying for**
        1. **Pricing-> payment for usage**
    3. **TWO types:**
        1. **Serverless- we do not have ACCESS to underlying hardware**
        2. **Managed Services-> there are VMs, but the cloud manages them for us-> we do not "need" access to underlying hardware**

Big data = data too big for 1 machine

Size of 1 machine = 2 numbers

Store-> 1 ,2 , 3

SQL/NoSQL/File/Win/Linux:
(Linked List)
M1-> 1,2 ——replication—— M3-> 1,2
M2-> 3
M4-> 3

EXPENSIVE and BUSINESS LOSS!

In 2000s=> GOOGLE-> index the entire internet!
Colossus -> FS built by GCP to ensure mass storage without massive fee!

SHARDING
M1-> 1,2 m2-> 2,3 m3-> 3,1

HDFS

HTML-> <input type='button'

Program small data
big data
how-to-do
what-to-do
a=1 a=? 1
TRUE
b=2 b=? 2
TRUE
c= a+b. c=? 3
TRUE

print(c). ??? “3” execute->”3”

imperative

declarative

immediate

delayed

Stack & heap managed by us DAG

Delayed executions-> EAGER and LAZY
Eager evaluation-> solve immediately (when possible)
Lazy evaluation-> Don’t solve till you really need it!

DAG -> directed acyclic graph
-> edges (functions or process or data flow or automation) and nodes (data or objects)
ALL big data technologies->
Spark, Hadoop, Databricks, DW, Tensorflow, dialogflow, airflow, GCP composer....
-> all these techs are HIGHLY interoperable!

Online Transactional Process-> OLTP
-> MySQL, SQL, PostgreSQL, MongoDB, GraphDB, gremlin, neo4j
Online Analytical Process-> OLAP
-> SQL DW, HBase, hive, BigQuery, BigTable, Synapse, EMR,

Small Data

big data

Terminology OLTP OLAP
Style Imperative Declarative
Use Editing Searching

Big Data- OLTP v/s OLAP

OLTP-> databases
-> normalisation
-> insert and edit/Transactions!
-> backend for web/mobile apps!

OLAP-> Data lake & data warehouse
-> denormalisation
-> insert and search/Analytics!
-> backend for AI/DE/DA/Emails/Marketing

Code-free ORCHESTRATOR
-> Data Factory

Pipelines-> ETL, EL or ELT
Extract-Transform-Load
-> mission specific single pipeline!
-> 1 objective-> either ML/DE/DA
Or
Extract-Load
-> no cleaning is required!
-> Data Visualization in Power BI!
Or
Extract-Load-Transform
-> SAME data has multiple use-cases!
-> dataset1-> MLEng, DE, DA, DV
-> extract from multiple sources, and DUMP into a Data Lake first!
-> From Data Lake-> we create mission specific ETL pipelines!

Components:

1. **Cloud NEUTRAL**
    1. **Sources: Any Transactional DB**
    2. **ETL Tool**
    3. **Data Lake: Hadoop or HDFS compliant!**
2. **Azure**
    1. **Source: Azure SQL Database (dummy data)**
    2. **ETL Tool: Data Factory (codeless), Databricks (Py/Scala/SQL/R) or Spark (Py/Java/Scala/SQL) or Apache beam (Py/Java) or HDInsight (native Spark/Hadoop/Kafka/Hive/HBase) or Synapse Analytics (Py/Scala/C#.NET/SQL/R)**
    3. **Data Lake: HDFS Compliant Azure Data Lake Storage (ADLS Gen2)**
3. **GCP**
    1. **Source: Cloud SQL (SQL Server or MySQL or PostgreSQL)**
    2. **ETL Tool:**

Blob-> AWS/Azure/GCP->
FLAT Storage!
No FOLDERS!!!
FOLDERS ARE ONLY IN THE UI and FILENAME! Not in reality!!!

Real folders-> NATIVE HADOOP IS THE ONLY OPTION!
Azure-> HDInsight (Hadoop), AWS-> EMR, GCP-> DataProc

Azure's advantage:
-> BLOB Storage is capable of creating HIERARCHIES-> which provide SAME functionality as folders!

Hierarchal Namespace-> checkbox
-> Folder-like structure
-> folder specific access control
-> better management of data
-> DA, DE, ML

HN-> enables faster processing than BLOB storages!

HN with Blob Storage-> HDFS compliant

Copy paste the data from HDFS to Blobs->

In programs-> wherever you've "hdfs://" replace it with "adls" or "s3" or "gcs://"

ACL-> linux style RWX permissions for User, Group and Others!

DataFactory:

1. **Linked Service: Connect to source and destination**
2. **Datasets: Representation/Schema for my source and destination**
3. **Pipelines: Build a pipeline to transfer the data from source schema to the destination schema**