SECTION A — FOUNDATIONS OF BIG DATA

Q1. Explain the 5 Vs of Big Data. Why are they significant in analytics?

Answer:
The 5 Vs — *Volume, Velocity, Variety, Veracity,* and *Value* — describe the defining characteristics of Big Data.

- **Volume** refers to the enormous amount of data generated every second — from sensors, logs, clicks, transactions, etc. Traditional databases can't store or process petabytes efficiently, so distributed systems like Hadoop are needed.

- **Velocity** captures the *speed* of data generation and processing. In real-time analytics (e.g., stock trades, IoT monitoring), systems must handle data streams within milliseconds.

- **Variety** emphasizes heterogeneous data types — structured (tables), semi-structured (JSON, XML), and unstructured (images, video, logs). Big Data tools must support all formats.

- **Veracity** reflects data *quality and reliability*. Big Data is often incomplete or inconsistent, requiring cleaning, validation, and redundancy checks.

- **Value** is the end goal — extracting actionable insight from the massive raw data.
  Together, the 5 Vs guide storage design, processing architecture, and tool selection in modern analytics pipelines.

Q2. Differentiate between traditional RDBMS and Big Data systems.

Answer:
Traditional RDBMSs manage structured data with fixed schemas and rely on vertical scaling (adding more power to one machine). They use ACID transactions and handle gigabyte-scale workloads.
Big Data systems (Hadoop, Spark, NoSQL) manage petabytes of mixed data through horizontal scaling (adding machines). They use distributed storage, eventual consistency, and schema-on-read models.
RDBMS is ideal for small, consistent, real-time transactions; Big Data systems excel at massive, analytical, batch or stream processing across clusters.

Q3. Discuss how cloud computing supports Big Data Analytics architectures.

Answer:
Cloud computing provides the elasticity and scalability Big Data requires.

- **Infrastructure as a Service (IaaS):** offers on-demand compute/storage (e.g., AWS EC2, EBS).
- **Platform as a Service (PaaS):** pre-built Hadoop/Spark clusters (e.g., EMR, Dataproc).
- **Software as a Service (SaaS):** analytics dashboards (e.g., BigQuery, Snowflake).
  Clouds abstract hardware management, enable fault-tolerant distributed storage, and integrate monitoring and billing. This eliminates the need for owning data centers while allowing flexible cluster scaling, global data access, and cost optimization for variable workloads.

SECTION B — LINUX SYSTEMS & FILE STRUCTURE

Q4. Explain the hierarchy of the Linux file system and the purpose of default directories.

Answer:
Linux follows a single-root hierarchy beginning at `/` (root).

- **/bin** — essential binaries (e.g., ls, cat, cp).
- **/boot** — bootloader files (kernel images).
- **/dev** — device files (disks, terminals).
- **/etc** — configuration files for system and apps.
- **/home** — user directories.
- **/lib / lib64** — shared libraries used by programs.
- **/tmp** — temporary files cleared at reboot.
- **/usr** — user utilities, applications, docs.
- **/var** — variable data like logs, caches, mail spools.
  This standardized layout ensures portability and predictable behavior of shell scripts and

services.

Q5. Why is Linux preferred in Big Data environments?

Answer:
Linux is open-source, stable, and optimized for multi-user, networked operations. Its powerful command-line tools, secure permissions, and scripting support allow automation of repetitive Big Data tasks. Hadoop, Spark, Docker, and Kubernetes are all designed natively for Linux, providing deep integration with kernel-level features such as cgroups, namespaces, and schedulers that underpin cluster resource isolation and performance.

SECTION C — VIRTUALIZATION & CONTAINERIZATION

Q6. Define virtualization. Describe the main types with examples.

Answer:
Virtualization abstracts physical hardware into multiple simulated environments.

1. **Application Virtualization** — isolates an app from the host OS using a virtual layer (e.g., Microsoft App-V, Citrix).

2. **Server Virtualization** — divides one physical server into multiple VMs using a hypervisor (e.g., VMware ESXi, KVM).

3. **Desktop Virtualization** — user desktops hosted centrally and streamed remotely (VDI, Citrix XenDesktop).

4. **Storage Virtualization** — pools physical disks into a single logical storage (SAN/NAS abstraction).

5. **Network Virtualization** — converts switches, routers, firewalls into software-defined components (SDN, vLAN, VXLAN).

6. **Data Virtualization** — provides unified real-time view of distributed data sources without physically moving data.
   All rely on a **hypervisor** that manages resource sharing and isolation between VMs.

Q7. Distinguish between Type 1 and Type 2 Hypervisors with examples.

Answer:
A **Type 1 hypervisor** runs directly on the host hardware ("bare metal"), controlling resources efficiently — examples: Hyper-V, VMware ESXi, Xen.
A **Type 2 hypervisor** runs on top of a host OS as a regular application — examples: Oracle VirtualBox, VMware Workstation.
Type 1 offers higher performance and security (used in servers and data centers), while Type 2 is better for personal or development use.

Q8. Explain snapshots, migration, and failure recovery in virtualization.

Answer:
A **snapshot** captures the complete VM state (disk + memory) at a given moment, allowing rollback if an update fails.
Migration moves a VM from one host to another — either live (minimal downtime) or offline — useful for load balancing and maintenance.
Failover automatically restarts VMs on alternate hosts if hardware fails, ensuring high availability. Together, they make virtualized infrastructures resilient and maintainable.

Q9. What is containerization, and how does it differ from virtualization?

Answer:
Containerization virtualizes at the *OS-level* instead of hardware. Containers share the host OS kernel but run isolated user-spaces with their own libraries and dependencies.
Unlike full VMs (each with its own OS), containers are lightweight, start in seconds, and consume fewer resources. Tools like Docker and Kubernetes enable "build once, run anywhere."

Feature	Virtualization	Containerization
Isolation	Full OS per VM	Shared kernel + namespace isolation
Startup time	Minutes	Seconds
Performance overhead	High	Low
Use case	Multiple OS on same hardware	Microservices, DevOps, Big Data pipelines

---

SECTION D — STORAGE & HARDWARE IN BDA

Q10. Compare HDD, SSD, and Intel Optane technologies in context of Big Data.

Answer:

- **HDD:** Magnetic platters + mechanical heads; cheap, large capacity but slow (100–200 MB/s). Ideal for cold/archival data lakes.

- **SSD:** NAND flash, no moving parts, high speed (0.1 ms latency, 500 MB/s – 7 GB/s). Used for hot data, ETL, real-time analytics.

- **Intel Optane (NVRAM):** persistent memory bridging DRAM and SSD speeds (~1–3 GB/s, low latency ~300 ns). Suited for caching, checkpointing, and in-memory analytics at lower cost than DRAM.
  A hybrid strategy — HDD for storage, SSD/Optane for compute nodes — gives both capacity and speed.

Q11. What are disk and filesystem block sizes, and why do they matter?

Answer:
A **disk block** is the smallest physical read/write unit (usually 512 B – 4 KB).
A **filesystem block** (logical) groups multiple disk blocks (commonly 4 KB – 8 KB).
In HDFS, blocks are **128 MB (default)** to minimize seek overhead and maximize transfer throughput. Larger blocks reduce metadata load on the NameNode and enable efficient sequential reads.
Choosing proper block sizes ensures balance between parallelism (more blocks) and I/O efficiency (fewer seeks).

SECTION E — DATA LOCALITY & HADOOP ARCHITECTURE

Q12. Define data locality and explain its importance in Hadoop.

Answer:
Data locality means moving computation *to where the data resides*, instead of transferring data over the network.
In Hadoop, the scheduler places map tasks on nodes holding required HDFS blocks (node-local). If unavailable, it tries rack-local, then off-rack.

This minimizes network I/O, speeds execution, and maximizes cluster efficiency. It's crucial because network bandwidth is limited compared to local disk throughput.

Q13. What are the roles of NameNode and DataNode in HDFS?

Answer:

- **NameNode:** manages metadata — directory structure, block locations, permissions, replication factors. It does not store actual data.

- **DataNode:** stores the real HDFS blocks on local disks and reports heartbeats and block lists to the NameNode.
  If a DataNode fails, the NameNode detects it and replicates missing blocks elsewhere.
  Together they provide scalable and fault-tolerant storage across commodity hardware.

Q14. Explain how YARN enhances Hadoop's resource management.

Answer:
YARN (Yet Another Resource Negotiator) separates job scheduling from processing.

- **ResourceManager:** global scheduler allocating containers (CPU, RAM) to apps.

- **NodeManager:** runs on each node to manage containers and report health.

- **ApplicationMaster:** per-job manager that requests resources and monitors tasks.
  This modular design lets multiple frameworks (MapReduce, Spark, Hive) share the same cluster resources efficiently.

Q15. How does replication in HDFS differ from RAID?

Answer:
RAID duplicates data at the disk level within a single server for redundancy.
HDFS replication occurs across different machines (default 3 copies on distinct racks).
RAID improves local read speed and fault tolerance inside a server, while HDFS ensures cluster-wide data availability and automatic recovery from node failures.

SECTION F — MAPREDUCE MODEL

Q16. Describe the MapReduce programming model with an example.

Answer:
MapReduce divides data processing into two phases:

1. **Map Phase:** Takes input (key, value) pairs and emits intermediate (key, value) pairs. Example — reading weather records and emitting (year, temperature).

2. **Shuffle & Sort:** Groups all values by key so that each reducer gets all records for one key.

3. **Reduce Phase:** Processes each group to produce final output (e.g., max temperature per year).
   It provides fault tolerance and automatic parallelization across nodes.

Q17. Write pseudocode for MapReduce to compute the maximum temperature per year.

Answer:

```
map(String record):
    year = record[0:4]
    temp = parseInt(record[5:8])
    if temp != MISSING:
        emit(year, temp)

reduce(String year, Iterator temps):
    maxTemp = -∞
    for t in temps:
        if t > maxTemp:
            maxTemp = t
    emit(year, maxTemp)
```

Explanation:
The map function extracts year and temperature from each record. Hadoop groups all records for the same year, and the reduce function outputs the maximum temperature for each year.

Q18. What is the role of the Combiner in MapReduce?

Answer:
A Combiner is a mini-reducer that runs on the map output to reduce data transfer between map and reduce phases.
For example, if each mapper outputs (word, 1) for a word count job, the Combiner aggregates counts locally before sending to reducers. It minimizes network bandwidth usage and speeds up processing.

Q19. Explain the concept of input splits and how they affect load balancing.

Answer:
Input splits are logical divisions of input data for Map tasks. Normally one split ≈ one HDFS block (128 MB).
Having many small splits improves parallelism but adds overhead; too few large splits cause uneven workloads.
Hadoop's scheduler balances load by reassigning unfinished tasks to idle nodes and by speculative execution of straggling tasks.

SECTION G — LINUX & BASH SCRIPTING

Q20. What is a shell script, and why is Bash commonly used?

Answer:
A shell script is a file containing commands executed by a shell interpreter to automate tasks.
Bash (Bourne Again Shell) is Linux's default because it supports variables, loops, functions, command history, and is POSIX-compliant.
It automates system operations like backups, monitoring, and Docker control, which is essential for Big Data cluster administration.

Q21. Explain the purpose of shebang (#!) and chmod +x in scripts.

Answer:

- **Shebang (#!)** at the top of a script (e.g., `#!/bin/bash`) tells the OS which interpreter to use. Without it, the default shell may run the script incorrectly.

- **chmod +x filename** adds execute permission so the file can be run as a program. Together they make the script directly executable from the command line.

Q22. Explain positional parameters in Bash with an example.

Answer:
Positional parameters represent arguments passed to a script: `$0` (script name), `$1`, `$2`, etc.
Example:

```
#!/bin/bash
echo "Script: $0"
echo "First arg: $1"
echo "Second arg: $2"
```

If run as `./greet.sh Zuha Aqib`, output is:

```
Script: ./greet.sh
First arg: Zuha
Second arg: Aqib
```

This allows scripts to accept dynamic inputs for automation.

Q23. Write a Bash script to create a backup of all .txt files into a timestamped directory.

Answer:

```
#!/bin/bash
backup_dir="backup_$(date +%Y%m%d_%H%M%S)"
mkdir -p "$backup_dir"
cp *.txt "$backup_dir"/
echo "Backup completed to $backup_dir"
```

This script creates a new folder named with current date/time and copies all text files into it, simplifying data archiving.

Q24. Explain what the fi keyword means in Bash.

Answer:
`fi` is the closing keyword for an `if` statement in Bash.
Example:

```
if [ $x -gt 10 ]; then
  echo "Greater than 10"
fi
```

Bash reads sequentially, so it needs `fi` to mark the end of the conditional block.

Q25. What does the set -e command do, and why is it used?

Answer:
`set -e` forces the script to terminate if any command returns a non-zero (error) status.
It ensures that subsequent commands don't run on partial or failed states, improving reliability for automation scripts like data uploads or backups.

SECTION H — DOCKER & STORAGE MANAGEMENT

Q26. Explain the architecture of Docker.

Answer:
Docker follows a client-server architecture:

- **Docker Client:** CLI that sends API requests.
- **Docker Daemon (dockerd):** manages images, containers, and networks.
- **containerd:** handles low-level container lifecycle.
- **runc:** executes containers via Linux kernel features (namespaces, cgroups).
  The daemon and client communicate via REST API over Unix sockets. Together, they enable consistent, isolated app deployment.

Q27. What is a Docker volume, and why is it important?

Answer:
A Docker **volume** is external storage mounted into containers to persist data beyond container lifecycle.
Without volumes, data is lost when a container stops. Volumes allow sharing files between containers, backups, and state persistence (e.g., database data).
They are managed by Docker itself, often stored under `/var/lib/docker/volumes/`.

Q28. Distinguish between COPY, ADD, and RUN commands in Dockerfiles.

Answer:

- **COPY:** copies local files into the image.
- **ADD:** similar to COPY but also supports remote URLs and auto-extraction of archives.
- **RUN:** executes a command inside the image to build layers (e.g., installing packages).
  Example:

```
FROM ubuntu
RUN apt-get install -y curl
ADD https://example.com/file.tar.gz /tmp/
COPY app/ /app/
```

Q29. Explain how Docker ensures isolation between containers.

Answer:
Docker uses Linux **namespaces** (to isolate process IDs, networks, mounts, users) and **cgroups** (to control CPU and memory usage).
Each container believes it has its own system, even though it shares the kernel.
This process-level isolation ensures stability and security while maintaining near-native performance.

Q30. Why are Docker containers faster than virtual machines?

Answer:
VMs virtualize entire hardware stacks and boot full guest OS instances, consuming heavy

memory and CPU.
Containers share the host OS kernel and only package necessary dependencies.
Hence, containers start in seconds, occupy minimal resources, and achieve near-native performance — making them ideal for distributed Big Data environments.

✅ **End of Exam Paper**