1️⃣ 5 Vs of Big Data

- **Volume:** massive scale of data (TB–PB).
- **Velocity:** data speed; real-time streams.
- **Variety:** structured / semi / unstructured.
- **Veracity:** reliability, noise, inconsistency.
- **Value:** extracting insights = ultimate goal.
  👉 Drives system choice, architecture, analytics type.

2️⃣ BDA Cloud Architectures

- **IaaS:** on-demand VMs, storage (AWS EC2, Azure VM).
- **PaaS:** managed Hadoop/Spark (EMR, Dataproc).
- **SaaS:** analytic apps (BigQuery, Snowflake).
- **Elastic scaling:** add/remove nodes anytime.
- **Pay-per-use:** cheaper than on-prem clusters.
- **Fault tolerance:** cloud HA + replication.
- **Hybrid clouds:** keep data local, process remotely.

3️⃣ Linux File System + Default Directories

Path	Purpose
`/`	root of all
`/bin`	essential binaries

`/boot`	bootloader, kernel
`/etc`	config files
`/dev`	device interfaces
`/home`	user dirs
`/lib`	shared libs
`/tmp`	temp files
`/usr`	user apps
`/var`	logs, cache, spool

Why Linux for BDA: open-source, secure, scriptable, stable, supports Docker, Hadoop, Kubernetes.

4️⃣ Virtualization – All Types (VERY IMP)

Concept: abstract physical hardware → multiple VMs via hypervisor.

Types:

1. **Application Virtualization:** isolated app container (MS App-V, ThinApp).
2. **Server Virtualization:** multiple OS per host (Hyper-V, VMware, KVM).
3. **Desktop Virtualization:** remote desktop hosted centrally (VDI).
4. **Storage Virtualization:** pool physical disks (SAN/NAS).
5. **Network Virtualization:** software-defined network (SDN, VXLAN).
6. **Data Virtualization:** access multiple sources as one logical view.

Snapshots: VM state backup → rollback.
Migration: move VM between hosts.

Failover: auto restart if host fails.
Need: resource utilization, cost efficiency, scalability, isolation.
Type-1 vs Type-2: bare-metal vs host-OS based.

5️⃣ Containerization (DOCKER)

- **OS-level virtualization** — containers share host kernel.
- **Faster, lighter, portable** than VMs.
- **Build once, run anywhere.**
- **Isolation:** namespaces (PID, NET, MNT, UTS, IPC, USER).
- **Control:** cgroups (CPU, RAM limits).
- **Images = layered templates.**
- **Dockerfile key cmds:**
    - `FROM` base image
    - `RUN` install software
    - `COPY` local → image
    - `ADD` remote + auto-extract
    - `WORKDIR` set working dir
    - `CMD` default run cmd
    - `ENTRYPOINT` fixed main program
    - `EXPOSE` port
    - `VOLUME` persistent storage
    - `USER` non-root exec

    - `LABEL`, `ARG`, `ENV` metadata/env vars

Command meanings (must know):
`docker run` start container,
`-d` detached,
`--name` name container,
`--net` join network,
`-p` port map,
`docker ps` list running,
`docker exec` run inside,
`docker stop/rm` stop/remove,
`docker volume ls` list volumes.

Docker Volume Commands (IMP):

- `docker volume create`, `inspect`, `rm`.
- Mount with `-v host_dir:container_dir`.
- Persist logs / DB data beyond container life.

Architecture layers:
Client → Docker Daemon → containerd → runc → Linux Kernel.

---

6 Disk & Filesystem Block Sizes (VERY IMP)

- **Disk block:** physical read/write unit (512 B–4 KB).
- **Filesystem block:** logical unit (4–8 KB).
- **HDFS block:** 128 MB (or 256 MB).
- **Large blocks = fewer seeks, faster streaming.**
- **Small blocks = more parallelism but overhead.**
- **Each block metadata ≈ 150 B → small files overload NameNode.**

7️⃣ HDD, SSD, Intel Optane (VERY IMP)

Feature	HDD	SSD	Optane
Tech	Magnetic platters	NAND Flash	3D XPoint (NVRAM)
Speed	100–200 MB/s	500 MB/s–7 GB/s	1–3 GB/s
Latency	5–10 ms	0.1 ms	300 ns
IOPS	~200	50k–1M	300k–800k
Cost	cheap	medium	high
Use	cold storage	hot data	memory-tier caching

Impact: HDD = data lake, SSD = ETL/real-time, Optane = hybrid tier (RAM + storage).

8️⃣ Hadoop Architecture (Data Locality – VERY IMP)

Goal: move computation to data.

HDFS Components:

- **NameNode:** metadata (namespace, permissions, replication).
- **DataNode:** actual data blocks, sends heartbeats + block reports.
- **Block size:** 128 MB; replicated (default = 3).
- **Replication policy:** 1 local + 1 same rack + 1 different rack.

YARN Components:

- **ResourceManager:** global scheduler.
- **NodeManager:** per-node container manager.

- **ApplicationMaster:** per-app manager → requests resources, monitors tasks.
- **Container:** isolated execution unit.

Fault Tolerance:
Heartbeats, re-replication, re-execution of failed tasks.

9️⃣ MapReduce Model + Pseudocode (IMP)

Concept: parallel process large data via (key, value) pairs.

- **Map:** transform input → (key, value).
- **Shuffle/Sort:** group values by key.
- **Reduce:** aggregate per key.
- **Combiner:** local mini-reduce → less network I/O.
- **Input Split:** logical piece of data (≈ HDFS block).
- **Speculative execution:** re-run slow tasks (stragglers).
- **Output:** written to HDFS.

Pseudocode – Max Temp

```
map(line):
  year=line[0:4]
  temp=int(line[5:8])
  if temp != MISSING:
      emit(year, temp)

reduce(year, temps):
  emit(year, max(temps))
```

🔟 Data Locality + Load Balancing

- **Node-local:** map where data lives.
- **Rack-local:** same rack if local slot busy.
- **Off-rack:** last resort.
- **Dynamic balancing:** idle nodes get new splits.
- **Goal:** minimize data transfer, maximize throughput.

1️⃣1️⃣ Bash Scripting (WITH COMMENTS REQUIRED)

Purpose: automate repetitive sys/admin tasks (Hadoop, Docker, backups).
Shebang: `#!/bin/bash` → interpreter.
Make executable: `chmod +x script.sh`.
Variables: `x=5`, `$x`.
Conditionals:

```
if [ $a -gt 5 ]; then
  echo "big"
fi
```

(`fi` ends `if` block)
Loops: `for`, `while`.
Functions:

```
greet() { echo "Hi $1"; }
```

Positional params: `$0`, `$1`, `$2`, `${10}` etc.
set -e: exit on error.
$?: last exit status.
IFS: field separator (default: space, tab, newline).
File check flags: `-e`, `-f`, `-d`, `-r`, `-w`, `-x`.
Redirect: `>` overwrite, `>>` append, `<` input.
Common tasks: backups, log analyzer, process monitor, parallel downloads.

Sample Q scripts to know:

- Backup system (compress, skip >50 MB).
- Log analyzer (/var/log/syslog → top 5 errors.csv).
- User creator (read users.txt → make users).
- Process monitor (kill >30% CPU / 500 MB RAM).
- Parallel download manager (3 URLs at a time).

---

12 Docker File Storage & Commands (VERY IMP)

Image Layers: cached; speeds rebuilds.
Containers: isolated runtime from image.
Volumes: persistent data, outside container lifecycle.
Bind Mount vs Volume: bind = host dir; volume = managed by Docker.
Network: bridge, host, none, user-defined.
Common tasks: build images, tag, push/pull, run Hadoop cluster via scripts.

---

13 HDFS vs RAID

Aspect	RAID	HDFS
Scope	single machine	cluster
Level	disk replication	block replication
Recovery	mirror switch	auto re-replication
Performance	faster read	parallel reads across nodes
Use	metadata servers	big data storage

---

14 Hardware & HPC vs Hadoop

- **HPC:** compute-intensive, MPI, shared filesystem (SAN).

- **Hadoop:** data-intensive, local disks (HDFS), fault-tolerant.
- **MPI issues:** manual data passing, fragile on failure.
- **Hadoop:** automatic fault recovery, simple programming model.

15️⃣ Intel Optane, SSD, HDD – BDA Relevance

- **ETL:** SSD → faster I/O.
- **Streaming:** Optane cache → low latency.
- **Batch archive:** HDD → capacity.
- **Hybrid tier:** DRAM (hot) + Optane (warm) + SSD/HDD (cold).

16️⃣ MR / HDFS Performance Math (know logic)

- Equal split ⇒ balanced tasks ⇒ min total time.
- Too small split ⇒ overhead.
- Too big ⇒ less parallelism.
- Job time dominated by slowest split (“straggler”).

17️⃣ Linux Permissions & Execution

- `chmod +x` → add execute.
- `ls -l` → check perms (`rwxr-xr-x`).
- `su`, `sudo` → root privileges.

- `cron` → schedule scripts (e.g., nightly Hive query).

1️⃣8️⃣ MR Pseudocode & Concepts Summary

- *Map:* filtering, preparing data.
- *Reduce:* aggregation.
- *Combiner:* optimization.
- *InputFormat:* TextInputFormat (default).
- *OutputFormat:* TextOutputFormat.
- *waitForCompletion(true)* → verbose output.
- *setOutputKeyClass/ValueClass* → define output types.

1️⃣9️⃣ Fault Tolerance Everywhere

- **HDFS:** replication + heartbeats.
- **MapReduce:** re-execute failed tasks.
- **YARN:** relaunch AM/container.
- **Docker:** restart policies.
- **Virtualization:** snapshot, migration, failover.

2️⃣0️⃣ Extra “Go-Through but Not Coming” Topics

- **rkt, Rackspace, Solaris Jails:** historical container tech — know only gist.
- **Docker networks:** not coming.

- **Dockerfile/Compose syntax writing:** not coming, just understand purpose.
- **Paxos / Raft / Unicast:** not coming.

2️⃣1️⃣ HIGH-YIELD QUICK FACTS

- **Default HDFS replication:** 3.
- **Default HDFS block:** 128 MB.
- **NameNode memory per block:** ~150 B.
- **Docker default bridge network:** `bridge`.
- **Linux shell invented:** 1970s (Bourne Shell).
- **Bash created:** 1989 (GNU).
- **Alpine Linux:** lightweight, uses `ash`, package mgr = `apk`.
- **chmod mnemonic:** r=4, w=2, x=1 → 755 = rwxr-xr-x.
- **YARN = Yet Another Resource Negotiator.**
- **Optane latency ≈ 300 ns vs SSD ~ 100 µs vs HDD ~ 10 ms.**

2️⃣2️⃣ Essential Comparisons to Memorize

Concept	Short Summary
Virtualization vs Containerization	VM = full OS; Container = shared kernel
HDD vs SSD vs Optane	Speed ↑ ← Cost ↑
Hadoop vs RDBMS	Schema-on-read vs Schema-on-write
RAID vs HDFS	same-machine vs cluster replication
HPC vs Hadoop	compute-intensive vs data-intensive

MR vs Spark	disk-based batch vs in-memory iterative
Type-1 vs Type-2 Hypervisor	bare-metal vs hosted
Data Locality vs Load Balance	less data transfer vs even task split

---

2️⃣3️⃣ MR/Docker/Linux Possible Code Questions

- MR: Write pseudocode for wordcount / max temp.
- Bash: backup, read file, arithmetic, if-else.
- Docker: explain effect of `docker run -d -p`.
- Hadoop: compute #blocks for given file size + block size.

2️⃣4️⃣ Memory Hooks

MapReduce in one line: parallelize, localize, tolerate, aggregate.
Virtualization goal: one machine, many worlds.
Container goal: one kernel, many apps.
HDFS principle: write-once, read-many.
Bash motto: automate everything.
Optane's role: memory that doesn't forget.

✅ **End of Final Cram Sheet**