Big Data Processing Infrastructures - Theory and Practice Guide

Full index

1. Introduction to Big Data

2. The 5 Vs + Types of Analytics

3. Computing Architecture: CPU, RAM, Storage

4. File Systems, RAID, and Checksums

5. SSH, IPs, Disk Formatting, ext4 FS

6. NFS, Batch Systems and HTCondor

7. DAS, NAS, SAN and Distributed File Systems

8. Cloud Computing: IaaS, PaaS, SaaS

9. Virtualization and Docker

10. Docker Compose and Multi-Container Apps

11. Udocker + Batch Jobs on HTCondor

12. Grid Computing, Middleware and X.509 Certificates

13. HPC vs HTC: Key Differences and Applications

14. Parallel Programming: OpenMP and MPI

15. Container Persistence: Volumes, Mounts, Commits

16. Fog and Edge Computing + IoT

17. Exascale and European Projects (MontBlanc, ExaNest)

Section 1 – Introduction to Big Data

Big Data refers to datasets that are too large, too fast, or too complex for traditional data processing systems. It is characterized by multiple dimensions known as the **5 Vs**:

1. **Volume** – Huge amounts of data (terabytes to zettabytes).
2. **Velocity** – Data generated and processed at high speed.
3. **Variety** – Different types of data: structured, semi-structured, and unstructured.
4. **Veracity** – Quality and trustworthiness of the data.
5. **Value** – Insights extracted to support decisions and automation.

Real-World Applications

Human Mobility and Disease Modeling

Call Detail Records (CDRs) from mobile phones can be used to track human movement, which helps model the spread of diseases like dengue.

Traffic Prediction

Google Maps uses GPS data from millions of users to estimate traffic conditions in real time. Data is anonymized and aggregated.

Scientific Applications

- High-energy physics (e.g., CERN)
- Astronomy (e.g., SKA telescope)
- Genomics and personalized medicine

These domains generate massive datasets that require **real-time analytics**, high-throughput infrastructure, and intelligent resource management.

The 5 Vs of Big Data

V	Description
Volume	Massive amount of data (TB to ZB)
Velocity	Fast data generation and real-time processing
Variety	Data from different sources and in different formats
Veracity	Trust, quality, and integrity of the data
Value	Insight and utility gained from processing the data

Types of Analytics

- **Descriptive** – What happened? *(e.g., sales in Q1)*
- **Predictive** – What is likely to happen? *(e.g., customer churn prediction)*
- **Prescriptive** – What should be done? *(e.g., recommend actions or automation)*

All these analytics require **high-quality, high-speed** data, making big data infrastructures essential.

> **Tip**: Big Data is a relative concept. If your data exceeds your current infrastructure's capacity to store, process, or analyze efficiently, then *you* have a Big Data challenge!

```
import pandas as pd
import time
```

```
# Simulating a Big Data stream (velocity) using a growing dataframe
data = []

print("Simulating data stream...")
for i in range(5):
    new_data = {"timestamp": time.time(), "sensor": f"S{i}", "value":
i * 2}
    data.append(new_data)
    df = pd.DataFrame(data)
    print(df.tail(1))  # show last row
    time.sleep(1)  # simulate data velocity
```

Section 2 – The 5 Vs in Depth + Analytics

The 5 Vs of Big Data – In Depth

1. **Volume**

- Refers to the sheer amount of data generated every second.
- Examples: genomics datasets, sensor logs, satellite imagery.
- Typical units: Terabytes (TB), Petabytes (PB), Zettabytes (ZB).
- Requires scalable storage and distributed systems.

2. **Velocity**

- Refers to the speed at which data is generated, collected, and processed.
- Real-time or near real-time examples: financial markets, autonomous vehicles, IoT sensors.
- Technologies: streaming platforms like Apache Kafka, Spark Streaming.

3. **Variety**

- Refers to the different forms of data: text, images, audio, video, logs.
- Structured: SQL tables; Semi-structured: JSON/XML; Unstructured: medical images, tweets.
- Requires flexible data ingestion and processing pipelines.

4. **Veracity**

- Measures the trustworthiness, accuracy, and consistency of the data.
- Involves data cleaning, validation, outlier detection.
- Critical in fields like healthcare, finance, research.

5. **Value**

- The most important “V”: turning raw data into actionable insight.
- Applications: predictive maintenance, fraud detection, personalization.

- The **goal** of big data is to extract **value** from complexity.

Analytics with Big Data

Descriptive Analytics

- What happened?
- Example: "How many new users joined last month?"

Predictive Analytics

- What will likely happen?
- Uses machine learning, historical data, statistics.
- Example: "Will this customer churn next month?"

Prescriptive Analytics

- What should we do?
- Suggests actions based on predictions and outcomes.
- Example: "Offer a discount to prevent churn."

These types of analytics are central to business intelligence, personalized medicine, and operational efficiency.

Use Case: Personalized Healthcare

A hospital collects:

- Patient clinical data (structured)
- Wearable sensor output (semi-structured)
- MRI images (unstructured)

They want to:

- Describe historical patient visits (Descriptive)
- Predict readmission risk (Predictive)
- Recommend intervention plans (Prescriptive)

This is a perfect real-world example that involves **all 5 Vs** and all types of analytics.

```
import pandas as pd
import numpy as np
import time
import matplotlib.pyplot as plt

# Simulating "Volume" with 100,000 records
volume_data = pd.DataFrame({
    'user_id': np.random.randint(1000, 5000, 100000),
    'timestamp': pd.date_range("2025-01-01", periods=100000,
freq='s'),
    'value': np.random.normal(loc=50, scale=10, size=100000)
```

```
})

print(f"Simulated dataset shape (Volume): {volume_data.shape}")
display(volume_data.head())

# Simulating "Velocity": streaming effect
print("Simulating Velocity - streaming 5 rows/sec...")
for i in range(5):
    time.sleep(1)
    print(volume_data.iloc[i])
```

Section 3 – Computing Architecture: CPU, RAM, Storage

CPU – Central Processing Unit

- Executes instructions of programs.
- Main components:
  - **ALU** (Arithmetic Logic Unit): performs operations
  - **Control Unit**: fetches and decodes instructions
  - **Registers**: fast memory inside the CPU

Modern CPUs are **multi-core**:

- Each core can process instructions independently.
- Some CPUs support **hyper-threading** (1 physical core = 2 logical threads).

RAM – Random Access Memory

- Temporary memory used for fast access during computation.
- Volatile: data is lost when the machine is turned off.
- Much faster than disk storage (nanoseconds vs milliseconds).
- Used for:
  - Storing data being used by CPU
  - Loading operating system and programs

Virtual Memory:

- If RAM is full, part of the disk is used (called *swap space*).
- Much slower than physical RAM.

Commands:

```bash free -m # shows RAM and swap memory

---
```

 Cella 3 (Markdown)

```markdown

# Storage – Disk and Filesystem

- **Persistent** data storage (non-volatile).
- Can be:
  - Hard Disk Drives (HDDs)
  - Solid State Drives (SSDs)
  - NVMe (very fast SSDs)
  - Tapes (used in data centers for archiving)

## Filesystems:

- Organize data in files/directories
- Examples: ext4, NTFS, XFS

## Commands:

```bash df -h # show disk usage lsblk # show all attached block devices

```
mkfs.ext4 /dev/sdb1   # format disk with ext4 filesystem
```

 Cella 4 (Python)

```python

# Simulate memory inspection using psutil

import psutil

# Get memory info

mem = psutil.virtual_memory() swap = psutil.swap_memory()

print(f"Total RAM: {mem.total / 1e9:.2f} GB") print(f"Available RAM: {mem.available / 1e9:.2f} GB") print(f"Swap Total: {swap.total / 1e9:.2f} GB") print(f"Swap Used: {swap.used / 1e9:.2f} GB")
```

□ Section 4 – File Systems, RAID, and Data Integrity (Checksum)

What is a File System?

A **file system** controls how data is stored and retrieved on a disk. It:

- Organizes files and directories
- Keeps track of free/used disk space
- Manages permissions and access

Common File Systems:

- **ext4**: widely used in Linux
- **NTFS**: default on Windows
- **XFS**: scalable, used in HPC systems

Format a disk:

```bash mkfs.ext4 /dev/sdb1

```
mkdir /mnt/data
mount /dev/sdb1 /mnt/data
```

---

□ Cella 2 (Markdown)

```markdown

## RAID – Redundant Array of Independent Disks

**RAID** combines multiple disks to:

- Increase performance
- Improve reliability and redundancy

### RAID Types:

- **RAID 0**: striping, faster but no redundancy
- **RAID 1**: mirroring, each file is duplicated
- **RAID 5**: block-level striping with parity, redundancy + efficiency

RAID is managed by hardware controllers or software tools like `mdadm`.

### RAID example command:

```bash mdadm --create /dev/md0 --level=5 --raid-devices=3 /dev/sd[b-d]
```

 Cella 3 (Markdown)

```markdown

## ✔ Checksum – Data Integrity Verification

A **checksum** is a small-size data fingerprint calculated from a file.

- Detects accidental corruption during transmission or storage.
- Common tools: `md5sum`, `sha256sum`

### Use case:

After downloading a file, verify integrity: ```bash md5sum file.txt

---

 Cella 4 (Python)

```python

# Example: compute MD5 checksum of a file in Python

import hashlib

def calculate_md5(filename): with open(filename, "rb") as f: file_hash = hashlib.md5() while chunk := f.read(8192): file_hash.update(chunk) return file_hash.hexdigest()

# Replace 'example.txt' with the path to your file

# print(calculate_md5("example.txt"))

# Section 5 – SSH, IP, Disk Formatting & ext4 Filesystem

## SSH – Secure Shell

SSH is used to securely connect to remote machines over a network.

- Requires **SSH client** on your machine and **SSH server** on the remote machine.
- Uses a pair of keys: **private key** (kept secret) and **public key** (stored on server).

## Example:

```bash ssh username@server_ip

```
ssh-keygen -t rsa

ssh-copy-id username@server_ip
```

---

 Cella 2 (Markdown)

```markdown

# IP Address – Identifying a Machine

Each machine on a network has an **IP address**.

- **Public IP**: reachable from the internet (can change every time).
- **Private IP**: used inside local networks (static).

## Commands:

```bash ifconfig # or ip a

---

 Cella 3 (Markdown)

```markdown

# Partitioning and Formatting a Disk

To use a new disk, you must:

1. **Partition it**: define sections on the disk
2. **Format it**: create a filesystem (e.g., ext4)
3. **Mount it**: make it accessible via the filesystem tree

## Step-by-step:

```bash sudo fdisk /dev/sdb # interactive tool
```

Create new partition with: n → p → enter

Save with: w

```
mkfs.ext4 /dev/sdb1       # format with ext4 filesystem

mkdir /data2              # create mount point
mount /dev/sdb1 /data2    # mount the new filesystem
```

 Cella 4 (Markdown)

```markdown

## Persistent Mounting with /etc/fstab

To make mounting permanent across reboots, add an entry to `/etc/fstab`:

```bash /dev/sdb1 /data2 ext4 defaults 0 2
```

```
mount -a    # apply fstab entries
```

Section 6 – HTCondor and Batch Systems

What is a Batch System?

A **batch system** allows jobs (programs/scripts) to be queued and run later on shared computing resources. It's commonly used in data centers and HPC environments.

Advantages:

- Manages thousands of jobs efficiently
- Prioritizes based on user, job type, or time
- Optimizes resource usage (CPU, RAM)

Job types:

- Single job (1 core)
- Collection (many jobs)
- DAG (jobs with dependencies)
- Parallel job (multi-core)

⚙ HTCondor Overview

HTCondor is a popular open-source batch system for job scheduling and execution.
```

Architecture:

- **Submit node**: where users submit jobs
- **Master node**: manages the cluster
- **Execute node (worker)**: runs the job

HTCondor daemons:

- `condor_master`: controls other daemons
- `condor_schedd`: manages the job queue
- `condor_startd`: advertises available resources
- `condor_exec`: runs the actual job

Submitting a Job with HTCondor

Step 1 – Create a script to run:

```bash echo "echo Hello HTCondor!" > script.sh chmod +x script.sh

# job.sub

Executable = script.sh Log = job.log Output = job.out Error = job.err Queue

```
condor_submit job.sub

condor_q
condor_status
```

# DAG Workflows in HTCondor

A **DAG (Directed Acyclic Graph)** describes a job dependency structure:

- A job is executed only when its parent jobs have completed.
- Used for complex pipelines (e.g. preprocess → compute → postprocess).

## Step 1 – Create job scripts

```bash echo "echo Preprocessing step" > pre.sh echo "echo Analysis step" > main.sh echo "echo Postprocessing step" > post.sh

chmod +x pre.sh main.sh post.sh

```
Executable = pre.sh
Output     = pre.out
Error      = pre.err
Log        = pre.log
Queue
```

```
Executable = main.sh
Output     = main.out
Error      = main.err
Log        = main.log
Queue

Executable = post.sh
Output     = post.out
Error      = post.err
Log        = post.log
Queue
```

 Cella 5 (Markdown)

```markdown

### Step 3 – Create the DAG file

```text

# workflow.dag

JOB A pre.sub JOB B main.sub JOB C post.sub

PARENT A CHILD B PARENT B CHILD C

```
condor_submit_dag workflow.dag

tail -f workflow.dag.dagman.out
```

Section 7 – Storage Architectures: DAS, NAS, SAN and Distributed File Systems

Types of Storage Architectures

1. DAS – Direct Attached Storage

- A disk directly connected to a computer via cable (e.g., USB, SATA).
- Simple, fast, not shareable across the network.

2. NAS – Network Attached Storage

- A storage server connected to the network.
- Shared via network protocols like NFS or SMB.

- Accessible by multiple users over a LAN.

3. SAN – Storage Area Network

- A high-speed dedicated network that provides access to block-level storage.
- Uses protocols like Fibre Channel or iSCSI.
- Scalable and high performance, used in data centers.

Comparison Table

Feature	DAS	NAS	SAN
Connection	Direct (e.g., USB, SATA)	Over LAN (Ethernet)	Dedicated network (FC/iSCSI)
Protocol	None	NFS, SMB	Block-level (Fibre, iSCSI)
Sharable	No	Yes	Yes
Performance	High	Medium	High
Cost	Low	Medium	High
Complexity	Low	Medium	High

Distributed vs Parallel File Systems

Distributed File System (DFS)

- Data is distributed across multiple servers.
- Files accessed via network protocols (e.g., NFS, SMB).
- Examples: HDFS, GlusterFS.

Parallel File System (PFS)

- Allows simultaneous access from multiple clients.
- Optimized for high-performance computing.
- Uses **metadata servers** and **data servers**.
- Examples: Lustre, IBM Spectrum Scale (GPFS).

In HPC environments, parallel file systems are preferred for speed and scalability. Distributed file systems are used where availability and redundancy are more important.

Both types often provide:

- Global namespace
- Redundancy (RAID, replication)
- Scalability for large datasets

Section 8 – Cloud Computing: IaaS, PaaS, SaaS

Cloud computing is a model for delivering computing resources (servers, storage, applications) over the internet, **on-demand**, and with **pay-per-use** pricing.

☁ Key Characteristics:

- **Self-service**: resources can be requested automatically
- **On-demand**: available instantly when needed
- **Elasticity**: can scale up/down dynamically
- **Measured service**: pay only for what you use
- **Network access**: available over the internet

Cloud Service Models

Model	Description	Example
IaaS – Infrastructure as a Service	Provides virtual machines, storage, networks	Amazon EC2, Google Compute Engine
PaaS – Platform as a Service	Provides OS, databases, web servers	Google App Engine, Heroku
SaaS – Software as a Service	Delivers apps over the web	Gmail, Dropbox, Netflix

Summary:

- **IaaS** → You manage software, OS
- **PaaS** → You deploy code, platform is ready
- **SaaS** → You just use the application

Cloud Deployment Models

- **Public Cloud**: Services offered over the internet (e.g., AWS, Azure)
- **Private Cloud**: Used by one organization (e.g., CERN private cloud)
- **Hybrid Cloud**: Mix of public + private clouds
- **Community Cloud**: Shared by multiple orgs with common goals

  Example: A hospital can use a private cloud for medical records and a public cloud for public health reports.

Virtual Machines and Containers

- **Virtual Machines (VMs)**: Emulate full hardware, run any OS
- **Containers** (e.g., Docker): Lightweight, share the host OS, faster

Feature	Virtual Machine	Container
OS	Full OS	Shared Host OS
Startup Time	Minutes	Seconds
Resource Use	High	Low
Portability	Medium	High

Feature	Virtual Machine	Container

Containers are ideal for cloud-native applications and microservices.

Practical Example – Running a Web Server with Docker

Docker is a powerful tool for packaging and running applications in containers. It is a **PaaS-like tool** because it provides the entire runtime environment for your app.

Let's try running a lightweight web server:

```bash
docker run -d -p 8080:80 nginx
```

http://localhost:8080

Cella 6 (Markdown)

```markdown
# Clean Up

To stop and remove the container:

```bash
docker ps # find the container ID
docker stop <container_id> # stop the running container
docker rm <container_id> # remove the container

docker rmi nginx

# Section 8.1 – AWS: Cloud Computing in Practice

Amazon Web Services (AWS) is the most widely used public cloud platform. It provides scalable solutions for compute, storage, batch jobs, and data analytics – making it a real-world implementation of IaaS, PaaS, and SaaS.

## Key AWS Services

| Service | Type | Description |
|---|---|---|
| **EC2** | IaaS | Virtual machines for custom workloads |
| **S3** | Object Store | Cloud storage for files and datasets |
| **EBS** | Block Store | Persistent disks for EC2 instances |
| **Batch** | HTC | Schedules and runs batch jobs, like HTCondor |
| **Lambda** | Serverless | Executes code without servers, great for Edge tasks |

| Service | Type | Description |
|---|---|---|
| **ECS/EKS** | Container | Orchestration for Docker (native or Kubernetes) |
| **EMR** | Big Data | Manages Hadoop/Spark clusters for analytics |

## Example: Running Docker Jobs on AWS

- Store inputs/outputs in **S3**
- Launch containers on **EC2**
- Schedule jobs using **AWS Batch**
- Archive data with **S3 Glacier**

This pipeline mimics a full HTC/HPC stack in the cloud.

## Related Sections

- EC2 ↔ Section 8 (IaaS), Section 9 (Docker)
- Batch ↔ Section 6 (HTCondor), Section 13 (HTC)
- S3 ↔ Section 7 (Storage), Section 15 (Volumes)
- Lambda ↔ Section 16 (Edge Computing)

AWS connects theory with a powerful real-world infrastructure.

1. Which AWS service is most similar to HTCondor in scheduling jobs?

A) EC2

B) Lambda

C) AWS Batch 

D) S3

 AWS Batch is the cloud-native batch job scheduler that allows running containerized workloads automatically.

1. What is S3 mainly used for?

A) Block storage

B) Object-based storage 

C) Compute services

D) Running container images

 S3 stores unstructured data (files, logs, backups) in the cloud and is accessible via HTTP APIs.

1. Which AWS service lets you run containers using native orchestration?

A) EBS

B) ECS 

C) S3

D) EMR

 ECS (Elastic Container Service) runs Docker containers; EKS is its Kubernetes variant.

1. What does AWS Lambda allow you to do?

A) Schedule DAGs

B) Launch full VMs

C) Run code on demand without provisioning servers 

D) Store Docker images

 Lambda supports serverless execution: you only provide code, AWS handles the infrastructure.

1. Which AWS service is best suited to store cold, archived data over time?

A) EC2

B) S3

C) S3 Glacier 

D) EBS

 S3 Glacier is designed for long-term archival storage at low cost, with slower access time.

# Section 9 – Docker: Theory, Images, Volumes, and Dockerfiles

Docker is a **containerization platform** that lets you run applications in isolated environments called **containers**.

Each container includes:

- The application code
- Dependencies (libraries, runtimes)
- Configuration files

 Think of a container like a lightweight virtual machine, but faster and more efficient.

## Images and Containers

- A **Docker image** is a snapshot of a container (like a template).
- A **Docker container** is a running instance of that image.

## Key Commands:

```bash docker pull ubuntu # download an image docker run -it ubuntu bash # run it interactively docker ps # list running containers docker stop  # stop a container docker rm  # remove container docker rmi  # remove image

---

 Cella 3 (Markdown)

```markdown

# Docker Volumes – Persistent Data

By default, Docker containers are **ephemeral**: data is lost when the container is removed.

To make data persistent:

```bash docker run -v $(pwd)/data:/app/data ubuntu

---

 Cella 4 (Markdown)

```markdown

# Dockerfile – Build Custom Images

You can define your own image with a **Dockerfile** (list of instructions).

## Example:

```dockerfile

# Dockerfile

FROM ubuntu:20.04 RUN apt-get update && apt-get install -y python3 COPY script.py /app/script.py CMD ["python3", "/app/script.py"]

docker build -t myimage . docker run myimage

---

 Cella 5 (Markdown)

```markdown

## Recap: Docker Architecture

| Layer | Role |
|---|---|
| Docker Engine | Runs containers on your machine |
| Image | Blueprint of container (OS + software) |
| Container | Running instance of an image |
| Volume | Mounted storage for persistence |

> Docker helps you simulate SaaS, PaaS and even distribute software across infrastructures.

# Section 10 – Docker Compose and Multi-Container Applications

**Docker Compose** is a tool to define and run multi-container Docker applications using a single YAML configuration file (`docker-compose.yml`).

Use cases:

- Web app + database (e.g., WordPress + MySQL)
- Pipelines with services
- Local testing of production environments

> ⚙ Compose allows you to manage the entire stack with a single command.

## Example: WordPress + MySQL Stack

Here's a basic `docker-compose.yml` file:

```yaml version: '3.1'

services: wordpress: image: wordpress ports: - "8080:80" environment: WORDPRESS_DB_HOST: db WORDPRESS_DB_USER: root WORDPRESS_DB_PASSWORD: example WORDPRESS_DB_NAME: wordpress

db: image: mysql:5.7 environment: MYSQL_ROOT_PASSWORD: example MYSQL_DATABASE: wordpress
```

Cella 3 (Markdown)

```markdown

## How to Launch the Stack

1. Save the above file as `docker-compose.yml`
2. In the same folder, run:
```

```bash docker-compose up

docker-compose down
```

 Cella 4 (Markdown)

```markdown

# Inspect and Monitor

Check running containers:

```bash docker ps

docker-compose logs -f

docker-compose up --build
```

Section 11 – Udocker: Running Containers Without Root

Udocker is a tool developed by CERN that allows running Docker containers **without root privileges**.
Useful in:

- HPC environments
- University clusters
- Shared infrastructures without Docker support

 It runs containers entirely in user space – no need for `sudo` or Docker daemon.

⚙ How Udocker Works

- Emulates container execution via Python
- Uses technologies like `PTRACE`, `Fakechroot`, `LD_PRELOAD`
- Compatible with Docker images (pulled from Docker Hub)

Installation: ```bash curl https://raw.githubusercontent.com/indigo-dc/udocker/master/udocker.py > udocker chmod u+x udocker ./udocker install

./udocker pull ubuntu

./udocker create --name=mycontainer ubuntu

./udocker run mycontainer
```

 Cella 3 (Markdown)

```markdown

# Using Udocker with HTCondor (Batch Integration)

You can run Udocker containers inside a Condor job:

**job.sub**:

 With `Transfer_Input_Files`, Udocker and the script are sent to the remote node. No `sudo`, no `docker`, just your user account!

# Pros and Cons of Udocker

| Advantage | Limitation |
|---|---|
| No root needed | Slightly slower than Docker |
| Runs in user-space | No real isolation (not secure for multi-tenant clusters) |
| Compatible with Docker | Less optimized for performance |

 Best for testing and personal workloads in restricted systems.

# Section 12 – Grid Computing and Middleware

**Grid computing** connects multiple distributed computing resources across institutions to form a virtual supercomputer.

- Used in large-scale research (e.g. CERN, LHC)
- Resources are **heterogeneous** and distributed
- Tasks are coordinated using a **middleware layer**

 Grid computing is the foundation of many scientific computing initiatives in Europe and beyond.

# What is Middleware?

Middleware is the **software layer** that connects users with computing resources.

It manages:

- Authentication and authorization
- Job submission and scheduling
- Data transfer and replication

## Popular Grid Middleware:

- **gLite**: developed by the EGEE project
- **ARC**: Advanced Resource Connector
- **UNICORE**: used in European infrastructures
```

- **Globus Toolkit**: early US-based solution

X.509 Certificates – Secure Identity in the Grid

To use grid resources, users must authenticate with **X.509 personal certificates** (digital identities).

- Issued by a Certification Authority (CA)
- Stored in `.pem` files and used with proxies

Example commands:

```bash
voms-proxy-init --voms cms
grid-proxy-info -timeleft
```

Cella 4 (Markdown)

```markdown
```

Submitting a Job with Middleware (e.g., gLite)

```bash
glite-wms-job-submit -a job.jdl
glite-wms-job-status 
glite-wms-job-output --dir .
```

Cella 5 (Markdown)

```markdown
```

Example Use Cases

- LHC experiments at CERN
- Distributed sequencing in genomics
- Climate simulations across supercomputing centers
- Earth observation data analysis (e.g., Copernicus)

Grid computing laid the foundation for modern federated infrastructures like the European Open Science Cloud (EOSC).

Section 13 – HPC vs HTC

⚡ High Performance Computing (HPC)

- Focus: **Speed** and **parallel execution**.
- Used for tightly-coupled simulations that run in parallel (e.g., weather modeling, fluid dynamics).
- Requires low-latency, high-bandwidth interconnects (e.g., Infiniband).
- Uses **supercomputers** or **clusters** with fast CPUs/GPUs and parallel file systems.

 In HPC, all cores must collaborate at the same time – ideal for "one big job".

High Throughput Computing (HTC)

- Focus: **Capacity** over time.
- Used to execute **many independent jobs** (e.g., parameter sweeps, genome scans, Monte Carlo simulations).
- Tolerant of network latency and resource heterogeneity.
- Uses distributed grids, cloud systems, or opportunistic clusters (e.g., HTCondor, GridEngine).

 In HTC, jobs run when and where resources become available – ideal for "many small jobs".

Comparison Table

Feature	HPC	HTC
Goal	Fast execution (low latency)	High throughput (many jobs/day)
Job Type	Tightly coupled	Independent, loosely coupled
Scheduling	MPI/OpenMP	HTCondor/Grid Engine
Resources	Homogeneous, fast interconnects	Heterogeneous, commodity clusters
Use Cases	Simulations, modeling	Bioinformatics, rendering, analytics
Examples	Weather, CFD, protein folding	LHC, BLAST, parameter sweeps

When to Use Which?

Use **HPC** when:

- You need speed and synchronization
- Your problem is parallelizable with MPI/OpenMP

Use **HTC** when:

- You have many short, independent tasks
- You want to maximize resource utilization over time

 In practice, many institutions support both: HPC for simulation, HTC for analysis.

Section 14 – Parallel Programming: MPI and OpenMP

Parallel programming allows a program to run **simultaneously** on multiple CPUs or cores.

There are two main paradigms:

- **MPI (Message Passing Interface)** → for distributed memory (many nodes)

- **OpenMP (Open Multi-Processing)** → for shared memory (multi-core CPU)

 MPI = many computers; OpenMP = one computer with many cores

MPI – Message Passing Interface

- Each process has its own memory space.
- Communication between processes is done via messages.
- Scales across nodes in a cluster or supercomputer.
- Standard in HPC (Fortran, C, Python with `mpi4py`).

Key MPI Concepts:

- `mpirun` or `mpiexec` → launch N processes
- `MPI_Send` / `MPI_Recv` → exchange data
- `MPI_Bcast`, `MPI_Scatter`, `MPI_Gather` → group communication

Example:

```bash mpirun -np 4 ./program

---

 Cella 3 (Markdown)

```markdown

# OpenMP – Shared Memory Parallelism

- Runs threads within the same process → shared memory model.
- Easier to implement than MPI.
- Used for parallelizing loops or sections in C, C++, Fortran.

## Key Directives (in C/C++):

```c

#pragma omp parallel

#pragma omp for

#pragma omp critical

#pragma omp barrier

```c
#pragma omp parallel for
for (int i = 0; i < N; i++) {
    a[i] = b[i] + c[i];
}

gcc -fopenmp file.c -o program
```
```

 Cella 4 (Markdown)

```markdown
## ⚔ MPI vs OpenMP – Which One to Use?

| Feature | MPI | OpenMP |
|---|---|---|
| Memory Model | Distributed (multi-node) | Shared (single-node) |
| Scalability | Very high | Limited to one node |
| Complexity | Higher (manual message passing) | Lower (directives-based) |
| Performance | High | Moderate to high |
| Example Use | Climate model on a cluster | Parallelizing loop on a CPU |

 Hybrid approaches (MPI + OpenMP) are common in modern HPC systems.

# 🐳 Section 15 – Docker: Volumes, Mount, Commit

Docker containers are ephemeral by default: changes and files inside them are **lost** once stopped or removed.

To make data persistent or reusable, we use:

- **Volumes**: managed by Docker
- **Bind mounts**: direct mapping of host folders
- **Commit**: snapshot of a container turned into a new image

## 📦 Docker Volumes

Volumes are the **preferred** way to persist data.

```bash docker volume create mydata docker run -v mydata:/data ubuntu

```
docker volume ls
docker volume inspect mydata
```

 Cella 3 (Markdown)

```markdown
## 🔗 Bind Mounts

Bind mounts map a specific folder from the host into the container.

```bash docker run -v $(pwd)/input:/app/input ubuntu
```
```

 Cella 4 (Markdown)

```markdown

#  docker commit – Save Changes to an Image

You can turn a modified container into a reusable image:

```bash docker run -it ubuntu

# install packages, edit files...

docker ps -a # get container ID docker commit  my_custom_image

#  Section 16 – Fog & Edge Computing + IoT

## What is Fog Computing?

**Fog computing** extends cloud services closer to the edge of the network. It brings storage, processing, and networking **closer to where data is generated**.

- Acts as a **bridge** between IoT devices and cloud data centers.
- Reduces latency and bandwidth usage.
- Allows local decision-making before pushing data to the cloud.

 Fog computing was proposed by Cisco to support real-time analytics for IoT.

##  What is Edge Computing?

**Edge computing** is similar to fog, but it pushes computation even closer – **directly on the device** or gateway.

- Ideal for latency-critical applications
- Uses microcontrollers, smart cameras, sensors, Raspberry Pi, etc.
- Often runs lightweight models or scripts (TinyML, MQTT)

 All edge devices can be fog nodes, but not all fog nodes are edges.

## Internet of Things (IoT)

IoT refers to billions of connected devices that sense, communicate, and act.

- Examples: smart thermostats, industrial sensors, medical devices
- Generates **massive, continuous data streams**
- Requires:

  - Low-latency analysis (Edge/Fog)
  - Aggregated storage and learning (Cloud)

### Challenges:

- Limited power and storage on devices
- Connectivity constraints
- Security and data privacy

 Edge/fog computing help filter and process only the essential data before transmission.

## Example Use Case: Smart City Traffic Monitoring

- **Edge**: Cameras detect vehicle flow in real time
- **Fog**: Local servers aggregate data and apply congestion models
- **Cloud**: Stores historical data and retrains AI models

Result: real-time alerts, adaptive traffic lights, better long-term planning

# Section 17 – European Exascale Projects: MontBlanc, ExaNest, EuroHPC

Europe is investing heavily in building **exascale-class** computing infrastructures.

 An **exascale system** can perform over **$10^{18}$ operations per second (exaFLOP)**.

These initiatives support:

- Research and innovation
- European digital sovereignty
- Scientific computing (physics, biology, climate)

## MontBlanc Project

- Aim: Design energy-efficient Exascale systems based on **ARM processors**
- Led by Barcelona Supercomputing Center
- Focus:
  - Low-power CPUs
  - Programming models for HPC
  - Scalability and performance

### Outcomes:

- Prototype platforms tested in HPC centers
- Contributed to EuroHPC strategic planning

## ⚡ ExaNest

- Focus: Interconnects, storage, fault tolerance
- Designed for **data-intensive HPC**
- Emphasis on:
  - Non-volatile memory (NVM)
  - Low-latency communication
  - Efficient I/O subsystems

 ExaNest complements compute-focused projects with scalable I/O and networking.

## EuroHPC JU (Joint Undertaking)

- Goal: Position Europe among the top 3 in supercomputing
- Public-private partnership co-funded by:
  - European Commission
  - Participating states
  - Industry and academia

### Objectives:

- Build and operate Exascale supercomputers in Europe
- Support R&D for HPC software and AI
- Enable access to academic and industrial users

### Key Systems:

- **LEONARDO** (Italy)
- **LUMI** (Finland)
- **MARENOSTRUM5** (Spain)

## Final Summary – Big Data Processing Infrastructures (Sections 1–17, with Cross-links)

## 1–2: Big Data Foundations

Defined by the 5 Vs: Volume, Velocity, Variety, Veracity, Value.

Enable Descriptive, Predictive, and Prescriptive Analytics.

Cross-links:

Drives Cloud Computing (Section 8) and Edge/Fog (Section 16).

Requires scalable Storage (Section 7) and orchestration via HTCondor (Section 6).

Motivates use of Docker (Section 9) for reproducibility and Grid (Section 12) for large-scale distribution.

## 3–5: Computing Architecture, Filesystems, SSH

CPU, RAM, and Disks form the hardware base.

File systems (ext4) and RAID ensure structure and reliability.

SSH and IP addressing are critical for remote job management.

Cross-links:

Needed before configuring Docker (9) or Udocker (11).

Required in all infrastructures: Grid, Cloud, and HPC setups.

Essential for launching jobs via HTCondor (6).

## 6: HTCondor and Batch Scheduling

Efficiently manages thousands of jobs with support for DAG workflows.

Cross-links:

Can execute Docker/Udocker containers (Sections 9, 11).

Used in HTC workloads (13).

Common in Grid infrastructures (12).

## 7: DAS, NAS, SAN & Filesystems

Compares direct, network, and high-speed storage systems.

Introduces Distributed FS (HDFS) and Parallel FS (Lustre).

Cross-links:

Affects performance of Big Data pipelines (1).

Used in Docker volumes (15) and Grid jobs (12).

Fundamental in HPC environments (13) for parallel I/O.

## ☁ 8: Cloud Computing (IaaS, PaaS, SaaS)

Explains how computing, platforms, and software are delivered on-demand.

Cross-links:

Hosts Docker containers and Udocker userspaces.

Integrates with Big Data, HTC, and Edge computing.

Essential for building hybrid architectures (Edge + Cloud, Section 16).

## 9–10: Docker & Compose

Containerization for portability and reproducibility.

Compose orchestrates multi-service applications.

Cross-links:

Base for running apps in Cloud (8) or HTCondor (6).

Used in Edge deployments (16) and non-root environments with Udocker (11).

Enables persistent analysis pipelines (see Section 15).

## 11: Udocker (Rootless Containers)

Run Docker images without root, ideal for HPC or Grid nodes.

Cross-links:

Compatible with HTCondor (6), Docker images (9), and Grid jobs (12).

Facilitates container use in multi-user systems.

## 12: Grid Computing & Middleware

Distributed computing infrastructure (e.g., LHC).

Uses X.509 certificates and job submission tools.

Cross-links:

Shares goals with HTC (13), depends on SSH/IP (5).

Interacts with storage systems (7), Udocker, and HTCondor.

## ⚡ 13: HPC vs HTC

HPC: tightly coupled, synchronous jobs.

HTC: many independent jobs over time.

Cross-links:

Determines use of MPI/OpenMP (14) vs HTCondor (6).

Informs hardware & software decisions: Cloud, Grid, Edge, Docker.

## 14: MPI & OpenMP

Tools for parallel programming: MPI (multi-node), OpenMP (multi-core).

Cross-links:

Used in HPC workloads (13).

Dependent on CPU/RAM/disks (3), and Parallel FS (7).

May be wrapped inside containers (Docker or Udocker).

## 15: Docker Volumes, Mount, Commit

Ensures data persistence and reusability inside containers.

Cross-links:

Crucial for Docker pipelines (9–10).

Supports Edge/Fog setups (16), Cloud-based workflows (8).

Alternative to filesystem setup in Section 4.

## 16: Fog & Edge Computing + IoT

Brings processing closer to the data source.

Cross-links:

Complements Cloud (8) and supports real-time Big Data (1).

Uses Docker lightweight containers (9), and sometimes Udocker (11).

Generates data that must be stored using DAS/NAS (7).

## 17: European Exascale Projects

MontBlanc (ARM-based), ExaNest (I/O & networking), EuroHPC (policy & funding).

Cross-links:

Real-world implementation of HPC (13), MPI (14), parallel FS (7).

Strategic use of Cloud, Grid, and containers.

| Topic | Connected To | Why it matters |
|---|---|---|
| **Big Data (1–2)** | Cloud (8), Edge (16), Grid (12), Analytics (2), Storage (4–7) | Big Data drives the need for fast, scalable, real-time infrastructures |
| **CPU/ RAM/Disk (3)** | Docker (9), HTCondor (6), HPC (13), MPI/OpenMP (14) | Physical resources limit/enable parallel computing and containers |
| **Filesystems & RAID (4)** | Storage (7), Docker Volumes (15), Grid (12) | File access speed and fault tolerance are essential for stable job execution |
| **SSH, IP, ext4 (5)** | Docker (9), Udocker (11), Grid Middleware (12), Cloud VMs (8) | Remote access, filesystem setup, and networking are prerequisites for |

| Topic | Connected To | Why it matters |
|---|---|---|
| | | orchestration |
| **HTCondor (6)** | DAG jobs, Docker (9), Udocker (11), HTC (13), Grid (12) | Submission engine for batch, scalable, and rootless workloads |
| **DAS/ NAS/SAN (7)** | Distributed FS (7), Docker volumes (15), Cloud (8) | Storage choice impacts speed, access, and reliability |
| **Cloud Computing (8)** | Docker (9), Compose (10), Fog (16), Big Data (1), Virtualization (11) | The backbone of modern elastic computing |
| **Docker & Compose (9–10)** | Cloud (8), Data Persistence (15), Batch Jobs (6), Edge/Fog (16), Udocker (11) | Portable environments for apps and pipelines |
| **Udocker (11)** | HTCondor (6), Grid (12), HPC clusters (13) | Enables Docker in restricted, rootless systems |
| **Grid + Middleware (12)** | Certificates (X.509), HTC (13), Distributed FS (7), LHC-style workflows | Federated computing across institutions |
| **HPC vs HTC (13)** | MPI (14), HTCondor (6), Grid (12), Cloud (8) | Differentiates job type and resource usage |
| **MPI & OpenMP (14)** | HPC (13), Supercomputers (17), Parallel File Systems (7) | Core to parallel scientific applications |
| **Volumes, Commit (15)** | Docker (9), Compose (10), Storage (7), Reproducibility | Makes containers persistent and shareable |
| **Edge & Fog (16)** | IoT devices, Cloud (8), Docker lightweight containers (9), Real-time analytics | Brings processing closer to where data is generated |
| **EuroHPC Projects (17)** | HPC (13), MPI (14), Infrastructure funding, ARM architectures (MontBlanc) | Future of European computing at exascale scale |

# QUESTIONS:

1. What is the main purpose of the 5 Vs in Big Data?

A) To classify file systems

B) To measure memory usage

C) To describe data characteristics and challenges 

D) To configure network latency

 Motivation: The 5 Vs (Volume, Velocity, Variety, Veracity, Value) describe key dimensions of big data complexity.

1. Which component of a computer stores data temporarily and is volatile?

A) HDD

B) SSD

C) RAM 

D) RAID

 Motivation: RAM is fast, temporary memory used during active processes, and is erased when powered off.

1. Which command shows disk usage in human-readable format on Linux?

A) lsblk

B) free -h

C) df -h 

D) ps aux

 Motivation: df -h lists disk space usage by mounted filesystems using human-readable units (MB, GB).

1. HTCondor is used for:

A) Encrypting data

B) Managing file systems

C) Scheduling and running batch jobs 

D) Creating containers

 Motivation: HTCondor is a batch job scheduler widely used for high-throughput computing.

1. Which storage architecture is best for real-time, high-speed block-level access?

A) NAS

B) DAS

C) SAN 

D) HDFS

 Motivation: SAN provides low-latency, high-speed access at the block level, ideal for databases and HPC.

1. In Docker, how do you persist data outside the container lifecycle?

A) Commit

B) Run in interactive mode

C) Use volumes 

D) Restart the container

 Motivation: Volumes allow data persistence even if the container is removed.

1. What does docker-compose up do?

A) Builds a Dockerfile

B) Compiles binaries

C) Starts multiple containers as defined in a YAML file 

D) Installs Docker

 Motivation: docker-compose up launches all services described in docker-compose.yml.

1. What is the role of the condor_submit command?

A) Launches containers

B) Transfers files

C) Submits a job to the HTCondor queue 

D) Starts the HTCondor daemon

 Motivation: condor_submit is the command-line tool to enqueue jobs in HTCondor.

1. What makes Fog Computing different from traditional cloud?

A) It is more expensive

B) It moves computing closer to data sources 

C) It only supports file transfers

D) It replaces Docker

 Motivation: Fog computing adds an intermediate layer that processes data near IoT devices to reduce latency.

1. Which parallel model shares memory between threads?

A) MPI

B) OpenMP 

C) GridFTP

D) Condor DAG

 Motivation: OpenMP runs threads in the same memory space, ideal for shared-memory systems.

1. What does docker commit do?

A) Pushes image to Docker Hub

B) Cleans up containers

C) Saves container changes as a new image 

D) Removes logs

 Motivation: It snapshots the current state of a running container and creates a new image.

1. What type of IP address is only valid within private LANs?

A) Public

B) Static

C) Private 

D) Dynamic DNS

 Motivation: Private IPs (e.g., 192.168.x.x) are used for internal communication and are not routable on the internet.

1. Which is a real Grid middleware used in Europe?

A) Kubernetes

B) gLite 

C) Ansible

D) Hadoop

 Motivation: gLite is a middleware developed in the EGEE project for Grid resource management.

1. What is the function of X.509 certificates in Grid computing?

A) Encrypt backups

B) Authenticate users and create proxy credentials 

C) Format disks

D) Activate Docker

 Motivation: X.509 certificates are digital identities used to securely access Grid infrastructures.

1. Which file system is best suited for massive distributed data like in Hadoop?

A) ext4

B) NFS

C) HDFS 

D) XFS

 Motivation: HDFS (Hadoop Distributed File System) is designed for large-scale, distributed storage.

1. What command allows checking your public key connection to a server?

A) sudo

B) ping

C) ssh user@host 

D) scp

 Motivation: SSH provides secure login to a remote machine using public/private key pairs.

1. What is MPI mostly used for?

A) Unstructured text mining

B) Real-time streaming

C) Inter-process communication in parallel programs 

D) Web server deployment

 Motivation: MPI is the standard for message passing in distributed-memory systems.

1. Which section is responsible for fault-tolerant I/O in ExaNest?

A) CPU design

B) Network cards

C) Storage architecture 

D) Grid jobs

 Motivation: ExaNest focuses on storage, interconnects, and I/O performance for exascale computing.

1. What happens if you stop a container without using a volume?

A) Data remains

B) You get a backup

C) Data is lost 

D) It converts to a VM

 Motivation: Without volumes or bind mounts, container data is ephemeral.

1. Which technology is designed for lightweight container execution in restricted systems?

A) VirtualBox

B) Singularity

C) Udocker 

D) Kubernetes

 Motivation: Udocker allows unprivileged users to run Docker images without root access.

# ADVANCED QUESTIONS

1. Which of the following best explains why docker commit should be used cautiously in production workflows?

A) It deletes all previous container layers

B) It creates a new image that is not reproducible or version-controlled 

C) It automatically pushes the image to Docker Hub

D) It increases the container runtime speed

 Motivation: docker commit is non-reproducible because it snapshots a state manually; better to use a Dockerfile in production for traceability.

1. What happens if you use docker run -v /tmp:/tmp and write a file in /tmp inside the container?

A) The file is lost when the container stops

B) The file appears in the host's /tmp directory 

C) The file is copied to the image

D) Docker blocks writing in /tmp

 Motivation: A bind mount like /tmp:/tmp directly maps the host directory; changes are reflected on the host filesystem.

1. Which of the following RAID levels can tolerate the failure of any one disk without data loss AND offers capacity gain?

A) RAID 0

B) RAID 1

C) RAID 5 

D) RAID 10

 Motivation: RAID 5 uses parity and tolerates one disk failure while offering better storage utilization than RAID 1.

1. In HTCondor, which file contains job dependency definitions in a DAG workflow?

A) job.sub

B) job.jdl

C) workflow.dag 

D) condor.cfg

 Motivation: workflow.dag defines dependencies (PARENT, CHILD); submit files like job.sub define individual jobs.

1. What will docker run --rm ubuntu echo hello do?

A) Create a container that runs and is removed automatically 

B) Start an interactive terminal

C) Remove the image after execution

D) Fail because of missing volume

 Motivation: The --rm flag removes the container after execution, making it ideal for short-lived commands.

1. Which file system provides POSIX compliance and parallel access in HPC clusters?

A) NFS

B) GlusterFS

C) Lustre 

D) HDFS

 Motivation: Lustre is optimized for parallel I/O and POSIX compliance, used in HPC centers like CINECA.

1. If your X.509 certificate has expired, what will happen when you try to submit a Grid job?

A) The job will run with warnings

B) The job will be rejected due to failed authentication 

C) The middleware will auto-renew it

D) Only file transfers will fail

 Motivation: An expired proxy or cert makes authentication fail, and the middleware won't accept the job.

1. You want to run a containerized tool in a cluster where Docker is not allowed. What is your best option?

A) Docker Compose

B) Kubernetes

C) Udocker 

D) GridFTP

 Motivation: Udocker allows users to run containers in user space without requiring Docker privileges.

1. What is the key difference between mounting a disk and defining it in /etc/fstab?

A) mount requires reboot

B) fstab can only mount volumes

C) mount is temporary unless /etc/fstab includes it 

D) fstab runs only in Docker containers

 Motivation: mount applies immediately but is lost after reboot unless it's declared in /etc/fstab.

1. Which of the following situations would most benefit from using HTC rather than HPC?

A) Molecular dynamics simulation

B) Live video rendering

C) High-resolution climate modeling

D) Repeating a BLAST search 10,000 times with different inputs 

 Motivation: HTC is ideal for many independent jobs (BLAST with multiple genomes), while HPC is for synchronized parallel computing.

| Term | Quick Definition |
|---|---|
| **5 Vs** | Volume, Velocity, Variety, Veracity, Value – describe Big Data complexity. |
| **HTCondor** | Batch job scheduler used in HTC environments; supports DAG workflows. |
| **RAID** | Redundant disk setup for speed (RAID 0), mirroring (RAID 1), or redundancy (RAID 5). |
| **ext4** | Common Linux file system; journaling and reliable. |
| **mount** | Temporarily attaches a file system to a directory tree. |
| **fstab** | Linux config file to automatically mount file systems on boot. |
| **Docker** | Platform for running isolated, portable containers. |
| **Docker Compose** | Defines and runs multi-container apps using a YAML file. |
| **Volume (Docker)** | External persistent storage for Docker containers. |
| **docker commit** | Creates a new image from a container's current state – not ideal for reproducibility. |
| **Udocker** | Tool to run Docker containers in user space without root privileges. |
| **Grid Computing** | Federation of distributed resources managed via middleware and certificates. |
| **X.509 Certificate** | Digital identity used to authenticate Grid users and submit jobs securely. |
| **MPI** | Message Passing Interface – used for multi-node, distributed-memory |

| Term | Quick Definition |
|---|---|
| | parallelism. |
| **OpenMP** | API for multi-threading in shared-memory systems (single node). |
| **HPC** | High Performance Computing – few, large, tightly coupled parallel jobs. |
| **HTC** | High Throughput Computing – many small, independent jobs executed over time. |
| **DAS / NAS / SAN** | Storage architectures: DAS = local, NAS = network file access, SAN = high-speed block. |
| **HDFS / Lustre** | HDFS = distributed file system for Big Data; Lustre = parallel file system for HPC. |
| **Edge Computing** | Processing data directly at the source (sensors, devices). |
| **Fog Computing** | Intermediate layer between edge and cloud for real-time data filtering. |
| **MontBlanc / ExaNest** | EU projects building energy-efficient Exascale systems (ARM, I/O, networking). |
| **Cloud (IaaS/PaaS/SaaS)** | Cloud service models: Infrastructure (EC2), Platform (App Engine), Software (Gmail). |