Helping Study BDP1

Martina Castellucci

July 2025

1. From Virtualization to Cloud Computing

Virtualization is the process of abstracting hardware resources to run multiple operating systems concurrently on a single physical machine. In classical virtualization, virtual machines (VMs) emulate entire hardware environments, each running its own full operating system. This approach offers strong isolation but introduces overhead in terms of memory, CPU, and startup time.

Cloud computing builds upon virtualization, but adds further abstraction. It provides on-demand access to computing resources over the network, supports dynamic elasticity, resource pooling, and pay-as-you-go models. However, pure virtualization lacks certain cloud properties such as self-service provisioning, network-based ubiquitous access, and elastic scalability.

2. Containers vs Virtual Machines

Docker introduces a lightweight form of virtualization, known as OS-level virtualization.

Unlike VMs, containers do not virtualize hardware. Instead, they share the host operating system's kernel and isolate applications at the process level.

Containers are ephemeral, portable, isolated, and efficient. They start in milliseconds, consume fewer resources, and offer consistency across environments. Docker containers run directly atop the host system, using a layered image system and union file systems.

Virtual machines are better suited for complete OS isolation and legacy workloads. Docker containers are ideal for microservices, DevOps workflows, and cloud-native applications.

3. Docker Architecture and Workflow

A Docker workflow begins with the definition of a Docker image using a `Dockerfile`. This file contains instructions to install packages, set environment variables, copy files, and define the container's entry point.

```
FROM ubuntu
RUN apt-get update && apt-get install -y fortunes cowsay lolcat
ENV PATH=/usr/games:$PATH
ENTRYPOINT fortune | cowsay | lolcat
```

To build the image:

```
docker build -t ubuntu_with_fortune .
```

This image can be executed using:

```
docker run -it ubuntu_with_fortune
```

To share images, Docker uses registries such as Docker Hub:

```
docker tag ubuntu_with_fortune myuser/ubuntu_with_fortune:1.0
docker push myuser/ubuntu_with_fortune:1.0
docker pull myuser/ubuntu_with_fortune:1.0
```

Essential Docker commands include `run`, `ps`, `images`, `search`, `commit`, `pull`, `push`, and `exec`. The `commit` command captures the state of a running container as a new image, while `exec` allows you to interact with a container process.

4. Docker Compose for Multi-Container Applications

`docker-compose` enables orchestration of multiple containers via a YAML configuration file. For instance, to deploy a

Wordpress site with MySQL backend:

```
version: '3'
services:
  database:
    image: mysql:5.7
    environment:
      - MYSQL_DATABASE=wordpress
      - MYSQL_USER=wordpress
      - MYSQL_PASSWORD=testwp
      - MYSQL_RANDOM_ROOT_PASSWORD=yes
    networks:
      - backend
  wordpress:
    image: wordpress:latest
    ports:
      - "8080:80"
    depends_on:
      - database
    environment:
      - WORDPRESS_DB_HOST=database:3306
      - WORDPRESS_DB_USER=wordpress
      - WORDPRESS_DB_PASSWORD=testwp
      - WORDPRESS_DB_NAME=wordpress
    networks:
      - backend
      - frontend
networks:
  backend:
  frontend:
```

Commands such as `docker-compose up --build`, `start`, `stop`, and `down` control the lifecycle of the application.

5. Docker Networking: The Bridge Concept

Docker containers are

network-isolated by default. The most common default driver is `bridge`, which creates an internal virtual network on the host. Containers on the same bridge network can communicate via IP or container names.

To create a custom bridge network:

```
docker network create mynet
docker run --network=mynet ...
```

Advanced drivers such as `overlay` or `macvlan` support multi-host networking or simulate physical interfaces.

6. Docker Volumes and Data Persistence

By default, data inside containers is ephemeral. To persist or share data,

Docker supports volumes.

- **Bind mounts**: mount host directory into container.
- **Named volumes**: managed by Docker and portable.

Example of bind mount:

```
docker run -v $HOME/data:/container_data ubuntu
```

Named volume:

```
docker volume create myvol
docker run -v myvol:/data ubuntu
```

7. udocker: Rootless Docker Execution

In environments where Docker cannot be used due to lack of root privileges (e.g., HPC clusters), **udocker** offers an alternative. It is a

user-space tool to execute Docker containers without requiring elevated permissions or installing the Docker daemon.

Installation and usage:

```
wget https://github.com/indigo-dc/udocker/releases/download/1.3.17/udocker-1.3.17.tar.gz
tar zxvf udocker-1.3.17.tar.gz
export PATH=$(pwd)/udocker-1.3.17/udocker:$PATH
udocker install

udocker pull dcesini/hpqc_2025:ubuntu_with_fortune_5.0
udocker run dcesini/hpqc_2025:ubuntu_with_fortune_5.0
```

Note: udocker does not allow building or committing new images. It is strictly for running prebuilt containers.

8. Final Thoughts

Docker revolutionizes the way software is built, shipped, and executed. It abstracts infrastructure, enforces isolation, and boosts developer productivity. With Compose and networking features, it becomes a powerful tool for orchestrating complex, scalable, and reproducible environments. In restricted systems, udocker fills the gap, enabling container-based execution without compromising security.

1. HPC vs HTC: Models and Architectures

High Performance Computing (HPC) refers to systems designed for tightly coupled parallel tasks. These systems typically use a parallel computing model, requiring communication between processors during task execution. HPC is deployed on clusters, composed of nodes connected via low-latency, high-bandwidth interconnects such as Infiniband, supporting shared-memory (OpenMP) or distributed-memory (MPI) paradigms.

High Throughput Computing (HTC), by contrast, focuses on executing large volumes of independent jobs over time. Tasks are loosely coupled, processed on distributed systems such as grids. HTC maximizes aggregate capacity rather than instantaneous performance.

HPC Characteristics

- Parallelism: tightly coupled tasks running concurrently
- Metrics: GFLOPS (Giga Floating Point Operations Per Second)
- Architecture: shared or distributed memory, high interconnect
- Applications: large-scale simulations, weather models, molecular dynamics

HTC Characteristics

- Distribution: independent jobs, often asynchronous
- Metrics: job throughput (e.g., jobs/day, tasks/hour)
- Architecture: distributed file systems, batch schedulers
- Applications: bioinformatics pipelines, Monte Carlo runs, rendering

2. HPC Clusters and HTC Grids

HPC Clusters are collections of interconnected compute nodes, coordinated via a head node and often operating under a job scheduler. They are optimized for parallel computing, where tasks share memory or exchange messages.

HTC Grids, in contrast, connect heterogeneous and geographically dispersed systems. Grids are optimal for task-level parallelism, where each node executes a different job with minimal or no inter-process communication.

Correlation: HPC clusters are suitable for coupled computations requiring synchronization, while HTC grids are ideal for scalable, independent workloads. Both models can coexist in hybrid environments.

3. Performance Metrics and Mathematical Models

HPC Performance: GFLOPS

The raw performance of an HPC system is measured in GFLOPS:

This metric reflects floating-point capacity, useful for numerical simulations.

HTC Performance

HTC systems are evaluated by throughput, i.e., how many jobs are completed over time:

Speedup and Efficiency

HTC Speedup (Crunching Factor):

HPC Speedup using Amdahl's Law:

Where:

- f is the serial fraction (non-parallelizable code)
- $1 - f$ is the parallel fraction
- n is the number of processors

As $n \to \infty$, the speedup tends to $1/f$. Serial fraction limits maximum achievable speedup.

Efficiency is defined as:

It expresses how well parallel resources are utilized.

4. Memory Architectures: UMA vs NUMA

- **UMA (Uniform Memory Access)**: equal memory access time for all CPUs. Common in smaller symmetric multiprocessing (SMP) systems.
- **NUMA (Non-Uniform Memory Access)**: memory access time varies by CPU location. Local memory is faster than remote.

NUMA improves scalability at the cost of programming complexity.

5. Programming Models: OpenMP and MPI

- **OpenMP**: compiler directives for shared memory parallelism. Easy to implement but limited to SMP systems.
- **MPI**: explicit message passing between processes for distributed memory. Ideal for large clusters.

6. Grid Computing Paradigm

Grid computing enables institutions to share resources across administrative boundaries. Key concepts:

1. 95 percent Law: Most CPU cycles go to embarrassingly parallel workloads.
2. Foster-Kesselman definition: Grids coordinate resources not under centralized control using open, standard protocols.

Grids emphasize interoperability, resource federation, and decentralized governance.

7. Cloud Computing: Models and Deployment

Cloud computing abstracts infrastructure to offer on-demand, elastic, scalable services accessible via the internet. Fulfills all NIST criteria:

- Self-service on demand: users provision resources without human interaction
- Network-based access: ubiquitous access from any device
- Resource pooling: multi-tenant architecture
- Elasticity: automatic scaling up/down
- Pay-per-use: measured billing based on consumption

Service Models

- IaaS (Infrastructure as a Service): raw compute/storage (e.g., AWS EC2)
- PaaS (Platform as a Service): runtime environments (e.g., Google App Engine)
- SaaS (Software as a Service): ready-to-use applications (e.g., Gmail)

Deployment Models

- Public cloud: services offered over the internet
- Private cloud: infrastructure restricted to one organization
- Hybrid cloud: mix of public and private resources

Design Principles

- Statelessness: applications avoid retaining user state
- Cloud-aware: designed to leverage elasticity and fault tolerance
- Multi-tenancy: shared resources across users with isolation
- Monolithic vs microservices: trade-off between simplicity and scalability

Final Considerations

HTC and HPC support distinct paradigms: HTC favors throughput and independence, while HPC favors speed-parallelism. Grid computing supports distributed HTC, while cloud computing provides an abstraction layer with elasticity and self-service. Mathematical models like Amdahl's Law and GFLOPS quantify scalability and performance. Architectural and programming choices (UMA, NUMA, OpenMP, MPI) critically affect system efficiency and application suitability.

Batch Systems, Computing Farms, and Clustering Infrastructure

In both High Throughput and High Performance environments, batch systems and computing farms form the operational backbone for large-scale job execution. A computing farm is a cluster of nodes where multiple jobs are processed in a distributed and coordinated fashion. These systems follow a client-server model, in which submit nodes prepare and dispatch jobs, worker nodes execute them, and a central manager or scheduler orchestrates task distribution.

Job Types and Workflows

Jobs handled in batch systems may include:

- Independent jobs: no communication or dependency between them
- Parametric jobs: same executable with different inputs (parameter sweeps)
- Parallel jobs: require coordination and possibly inter-node communication (e.g., MPI)
- DAG workflows (Directed Acyclic Graphs): a pipeline of tasks with explicit dependencies and execution order

Example pseudocode for a batch workflow:

```
# prologue

if not check\_if\_exec\_exist():
install\_exec()
if not check\_if\_db\_exist("db\_path"): exit(1)
check\_md5sum("db")
get\_query("input\_path")

# main

launch\_bwa("query","db")
launch\_blast("query","db")

# epilogue
```

```
md5sum("outfile")
gzip\_out("outfile")
save\_out("outfile")
clean\_all()
```

Scheduling Policies and Objectives

Schedulers manage how jobs are executed across the cluster, balancing priorities, fairness, and utilization:

- Fair Share: allocates resources based on historical usage, balancing user access
- Backfill: fills unused slots with small jobs to maximize cluster usage
- Reservation: guarantees resources for time-sensitive jobs

The primary goal of batch systems is to maximize efficiency, reduce wait times, and optimize overall resource utilization. The architecture is dynamic, allowing worker nodes to be added or removed depending on workload and system state.

HTCondor Architecture and Operation

HTCondor is a robust batch system designed specifically for HTC environments. Its architecture is composed of:

- Submit nodes: where users create and submit jobs (e.g., scripts, job files)
- Execute nodes: where jobs are run (also known as worker nodes)
- Central Manager: handles matchmaking, monitoring, and policy enforcement
- Queue: where submitted jobs wait for resource assignment

Example HTCondor job description file:

```
Executable = myexec.sh
Arguments = world.txt
Output = file1out
Error = condor.error
Log = condor.log
Queue
```

Submission and monitoring commands:

```
condor\_submit first\_batch.job
condor\_q
condor\_q -better-analyze <jobID>
condor\_history <jobID>
```

The system supports dynamic slot configurations and automatic fault recovery.

Filesystem Integration and Shared Data

For HTCondor to function across multiple nodes, a shared data space is needed. NFS (Network File System) is commonly used to mount a shared directory like `/data2` across all nodes. This setup allows consistent file access and output aggregation.
Example NFS export on server:

```
/data2 192.168.1.0/24(rw,sync,no\_wdelay)
```

Corresponding client mount entry:

```
192.168.1.1:/data2 /data2 nfs defaults 0 0
```

Example tools include:

- `install_an_HTCondor_machine.sh`: script to configure central manager, submit, and execute nodes
- `Howto-Install-an-NFS-Client-Server.sh`: script to configure NFS for shared directories
- `Howto-submit-a-batch-job-with-HTCondor.sh`: real submission example
- `Pseudocode-for-a-batch-job.txt`: workflow logic for multi-step execution

These components enable a scalable, shared-resource computing framework essential for executing thousands of jobs over time in HTC or parallel coordinated runs in HPC.

Networking and System Architecture Fundamentals

Modern computing infrastructures, whether in HPC, HTC, or datacenters, rely heavily on well-structured hardware and network designs. Understanding the underlying components and data movement is critical.

Hardware Components and Functional Units

A typical computer or node consists of:

- Central Processing Unit (CPU): executes instructions and controls operations
- Graphics Processing Unit (GPU): optimized for parallel floating-point calculations (used in ML, simulations)
- RAM (Random Access Memory): temporary working memory for fast-access data
- Cache: small, fast memory close to the CPU (L1, L2, L3 levels)
- Registers: fastest memory used directly by the CPU
- Mass Storage: long-term storage (e.g., HDD, SSD)

Example usage: programs like `bwa` and `blastn` perform substring matching and sequence alignment — these are CPU-intensive or GPU-accelerated tasks depending on the implementation.

Storage Architectures and Hierarchies

Modern datacenters use tiered storage:

- SSD (Solid State Drives): fast, expensive, for frequently accessed data (hot tier)
- HDD (Hard Disk Drives): bulk storage (warm tier)
- Tape Archives: lowest-cost storage for archival and backup (cold tier)

RAID (Redundant Array of Independent Disks) ensures performance and redundancy:

- RAID 0: striped, no redundancy, fast
- RAID 1: mirrored, high fault tolerance
- RAID 5: parity-based, balanced redundancy and speed

Storage systems:

- DAS (Direct Attached Storage): local, fast, isolated
- NAS (Network Attached Storage): shared via file system protocols
- SAN (Storage Area Network): block-level access over fiber or iSCSI

Networking Topologies and Infrastructure

Networking nodes connect systems for data sharing and orchestration. Key components include:

- Switch: connects devices on a LAN, operates at Layer 2 (Data Link)
- Router: connects networks, uses IP addresses, operates at Layer 3 (Network)
- Hub: legacy broadcast device (largely obsolete)
- LAN: Local Area Network (short range)
- WAN: Wide Area Network (geographically dispersed)
- Fiber Optic: high-bandwidth, low-latency cables for backbone infrastructure

IP addressing and network identity:

- MAC Address: physical hardware identifier (unique per NIC)
- IP Address: logical identifier (v4 or v6)
- DNS (Domain Name System): resolves hostnames to IP addresses

Performance metrics:

- Bandwidth: maximum data rate (e.g., Gbps)
- Latency: delay between request and response

Data Center Topologies and Examples

Top-of-Rack (ToR) architecture: each rack has a switch connecting to aggregation/core layers, minimizing cable length and latency.

INFN CNAF (Tier-1 datacenter): high-performance datacenter supporting national and international research. Tier-1 implies redundant power, cooling, and networking. These infrastructures are often part of federated or distributed computing networks (e.g., WLCG).

Cooling and Power Efficiency

Cooling systems ensure hardware stability and performance:

- Air cooling (fans, HVAC)
- Liquid cooling (direct-to-chip or immersion)

PUE (Power Usage Effectiveness):

$$PUE = \frac{\text{Total Facility Power}}{\text{IT Equipment Power}}$$

A lower PUE indicates better efficiency. Ideal values approach 1.0 (100

Policy and Data-Driven Workload Management

In large-scale distributed computing (e.g., HTCondor, SLURM, Kubernetes), policy-driven workload management allows scheduling to follow predefined rules and objectives, such as:

- Priority-based policies: give precedence to high-importance or deadline-constrained jobs
- Fair-share policies: enforce equitable resource distribution among users or projects
- Backfilling policies: fill idle nodes with short jobs to improve utilization
- Job aging and preemption: dynamically adjust scheduling based on wait time and resource class

Data-driven scheduling uses workload history and real-time metrics to make adaptive decisions. Examples include:

- Predicting resource availability and job run time
- Load-aware placement of tasks on optimal nodes
- Feedback loops to auto-scale or throttle resource allocation

These intelligent systems are fundamental to modern datacenters and scientific computing grids, enabling efficient, policy-compliant, and scalable resource orchestration.

This concludes the foundational description of infrastructure, networking, storage, system-level architecture, and dynamic workload orchestration in HPC, HTC, and modern cloud/datacenter computing.

Edge, Fog, Cloud Computing and IoT Technologies

Modern computing has evolved from centralized architectures toward distributed, latency-sensitive systems powered by edge and fog computing paradigms.

Edge Computing

Edge computing refers to data processing occurring close to the data source, such as IoT sensors, mobile devices, or embedded controllers. Its goal is to reduce latency and bandwidth use by avoiding round trips to centralized cloud infrastructures.

- Deployed on gateways, smart cameras, or industrial PLCs
- Real-time analytics for robotics, vehicles, monitoring
- Enhances privacy and autonomy by limiting data transmission

Fog Computing

Fog computing extends the cloud to the local area network, operating between the edge and the cloud. Fog nodes process, filter, and store data closer to the network's edge.

- Reduces burden on cloud and core networks
- Manages latency-sensitive and distributed applications
- Supports collaboration among edge devices

Cloud Computing in Context

While edge and fog computing process data locally, cloud computing provides centralized scalability, backup, orchestration, and storage.

- Ideal for batch processing, big data aggregation
- Complements edge/fog in hybrid architectures
- Provides APIs, elasticity, and high availability

Internet of Things (IoT)

IoT connects physical devices to the digital world. These devices produce and exchange data, often integrated with edge/fog/cloud layers.

- Examples: sensors, wearables, smart appliances
- Requires scalable networking and secure management
- Key to automation, smart cities, healthcare

Digital Twins

Digital twins are virtual replicas of physical assets or systems. They integrate real-time sensor data, simulation, and predictive analytics.

- Enables monitoring, diagnostics, and optimization
- Used in manufacturing, logistics, energy, healthcare
- Relies on cloud/edge for synchronization and compute

Together, these technologies create a continuum from sensors to cloud, enabling real-time, data-driven, intelligent systems in smart industries.

Visual Overview of the Full Architecture (Textual with verbatim)

The following detailed schematic represents the layered structure of the architecture described in the previous chapters. It reflects the flow from low-level hardware up to advanced cloud, edge, and digital twin technologies.

```
LEVEL 1: PHYSICAL LAYER
+------------------------------------------------------------------------------------+
\|                          PHYSICAL HARDWARE                                         |
\|  CPU (x86, ARM), GPU (CUDA), RAM, SSD/HDD, RAID, Cache, Registers                 |
+------------------------------------------------------------------------------------+
Note: Fundamental execution and memory units. Data is physically stored and processed here.
|
v
LEVEL 2: ABSTRACTION
+-------------------------------------------------+      +--------------------------+
\|             VIRTUALIZATION                      | ---> |   CONTAINERIZATION       |
\|  Hypervisors (KVM, Xen), VMs, OpenStack         |      |  Docker, Singularity     |
+-------------------------------------------------+      |  HTCondor, Podman        |
Note: Virtual machines emulate full OS environments; containers isolate apps using shared OS.
|
v
```

```
LEVEL 3: WORKLOAD MANAGEMENT
+-------------------------------------------------------------+
\|                BATCH / GRID / CLOUD SYSTEMS                |
\|  SLURM, HTCondor, Kubernetes, DAGMan, Queues               |
+-------------------------------------------------------------+
Note: Abstract scheduling, distribute and orchestrate jobs across many compute nodes.
|
v
LEVEL 4: STORAGE + SCHEDULING + NETWORKING
+----------------------------+   +-------------------+   +---------------------------+
\| PARALLEL / DIST. FILESYS |   |    SCHEDULING     |   |        NETWORKING         |
\| Lustre, Ceph, NFS, S3     |   | Fair Share, DAG   |   | OSI, TCP/IP, Switches     |
\| RAID 0/1/5, DAS/NAS/SAN   |   | Backfill, Preempt|   | Routers, LAN/WAN, Fiber   |
+----------------------------+   +-------------------+   +---------------------------+
Note: These three blocks operate together: storage provides data, scheduling organizes work,
and networking enables communication between all parts.
\                     |                        /
\_******************v*******************\_\_\_\_\_\_/
|
v
LEVEL 5: DATA CENTER + EDGE + CLOUD
+------------------------+   +---------------------------+   +---------------------------+
\|   TIER-1 DATA CENTER  |   |     EDGE / FOG COMPUTING |   |    CLOUD & DIGITAL TWINS  |
\| ToR Arch., PUE, HVAC  |   | IoT, Gateways, Low Lat.  |   | SaaS, PaaS, IaaS, Twins   |
+------------------------+   +---------------------------+   +---------------------------+
Note: Final layer combines centralized infrastructure (datacenters), distributed computation
(edge/fog), and abstract services + simulation (cloud and digital twins).
```

This refined textual layout clarifies the progression and relationships between components across all architectural levels—starting from hardware up to intelligent, distributed, cloud-aware platforms integrating IoT and real-time simulation.

Example Multiple-Choice and Conceptual Review Questions

Q1. Which of the following statements are true regarding Storage Area Networks (SAN)?

1. **SANs are typically used for file sharing and collaboration among users.**
2. **SANs provide block-level storage that can be accessed by servers over a network.**
3. **SANs offer high availability and redundancy features to ensure data integrity and uptime.**
4. ANs are primarily designed for personal and small office use.

Q2. Which of the following statements are true regarding Amdahl's Law?

1. Amdahl's Law suggests that increasing the number of processors will linearly increase the system's efficiency.
2. Amdahl's Law is used to calculate the execution speed of a program on a single processor.
3. According to Amdahl's Law, the maximum speedup achievable is inversely proportional to the fraction of the program that can be parallelized.
4. **Amdahl's Law states that the improvement in performance of a system is limited by the portion of the system that cannot be parallelized.**

Q3. What does the Docker command `docker ps -a` display?

1. Only running containers.
2. **All containers, including stopped ones.**
3. Active images.
4. Running logs.

Q4. What is the purpose of the CMD instruction in a Dockerfile?

1. Installs a package.
2. Copies files into the container.
3. **Specifies the default command to run when the container starts.**
4. Sets user permissions.

Q5. Which command builds a Docker image from a Dockerfile?

1. docker run
2. **docker build -t imagename .**
3. docker compose start
4. docker install

Q6. Which of the following are characteristics of NUMA (Non-Uniform Memory Access)?

1. **Memory access time depends on which processor accesses it.**
2. All processors access all memory equally.
3. It is used only in virtualized environments.
4. NUMA requires fiber networking.

Q7. What does `docker exec -it containerID bash` do?

1. Restarts the container.
2. **Opens an interactive bash shell in the container.**
3. Deletes the container logs.
4. Creates a new image.

Q8. What is typically managed by a container orchestration tool like Kubernetes or Docker Compose?

1. Physical hardware drivers.
2. VM BIOS settings.
3. **Container lifecycle, scaling, and networking.**
4. Direct network switch configurations.

Q9. What is a major feature of cloud-native applications?

1. **Statelessness and horizontal scalability.**
2. Dependence on specific hardware.
3. Manual service discovery.

4. Exclusive use of RAID.

Q10. What distinguishes SAN from NAS?

1. SAN uses file-level access, NAS uses block-level.
2. **SAN provides block-level access; NAS provides file-level access.**
3. SAN is only local; NAS is over fiber.
4. NAS does not support multi-user access.

Q11. What is the function of the `ENTRYPOINT` command in Docker?

1. It overrides CMD.
2. **It defines the executable that always runs in the container.**
3. It loads environment variables.
4. It maps ports.

Q12. What does the command `docker pull ubuntu` do?

1. Uploads a container.
2. **Downloads the Ubuntu image from Docker Hub.**
3. Deletes local images.
4. Mounts a volume.

Q13. What is the default port for HTTP used in web containers?

1. 443
2. **80**
3. 22
4. 3306

Q14. What does a DAG represent in computing?

1. A complete loop of tasks.
2. **A directed graph with no cycles showing task dependencies.**
3. A cache registry.
4. A data mount point.

Q15. Which metric best expresses power efficiency in a datacenter?

1. Uptime rate
2. CPU load
3. **PUE (Power Usage Effectiveness)**
4. IOPS

Q16. What type of memory is fastest but smallest in a processor?

1. RAM

2. HDD
3. **CPU Register**
4. SSD

Q17. What command shows current running Docker containers?

1. docker images
2. docker rm
3. **docker ps**
4. docker exec

Q18. What is the main function of RAID 5?

1. No redundancy
2. **Data striping with parity for fault tolerance**
3. Pure mirroring
4. Encryption

Q19. What describes edge computing?

1. **Computing near the data source to reduce latency**
2. Centralized processing only
3. Cloud container replication
4. DSL line provisioning

Q20. What does the IP 127.0.0.1 indicate?

1. DNS server
2. Gateway IP
3. **Loopback to localhost**
4. Public routing address

Q21. What is the advantage of horizontal scaling?

1. Decreased memory per process
2. **Add more nodes to increase capacity**
3. Increased fan speed
4. RAID conversion

Q22. What system is commonly used for job management in HTC?

1. Apache Spark
2. Hadoop HDFS
3. **HTCondor**
4. Proxmox

Q23. What is DAGMan in HTCondor?

1. Docker image tool
2. **Directed Acyclic Graph Manager for dependent jobs**
3. Disk space checker
4. Process pool allocator

Q24. What is the purpose of a checksum (e.g., MD5)?

1. File compression
2. **Verify file integrity after transfer or processing**
3. Optimize file loading speed
4. Log kernel messages

Q25. What is the difference between NAS and SAN?

1. NAS uses block-level access; SAN uses file-level.
2. **SAN = block-level; NAS = file-level storage shared over network**
3. SAN is wireless; NAS is wired only.
4. They are identical.

Q26. Which of the following statements about HPC is true?

1. HPC is optimized for small data tasks.
2. **HPC involves tightly coupled computation with high interconnect speeds.**
3. HPC only supports Windows.
4. HPC does not need a scheduler.

Q27. What is typically mounted in a Docker volume?

1. Web traffic
2. **Persistent storage or host directories into containers**
3. Log files into /dev/null
4. DNS records

Q28. Which of the following are characteristics of Docker containers?

1. **Isolated execution environments that share the host OS kernel**
2. Require a hypervisor to run
3. Are only supported on cloud platforms
4. Include a full GUI system

Q29. Which command is used to create a named Docker volume?

1. docker mount volume
2. **docker volume create myvol**

3. docker create-vol myvol
4. docker volume attach

Q30. What type of scalability does Kubernetes support natively?

1. Vertical scaling only
2. Manual scaling only
3. **Horizontal scaling with replica sets and autoscaling policies**
4. Kernel-level scaling

Q31. In NUMA systems, what causes latency variation?

1. High RAM clock speed
2. **Accessing memory that is attached to a different CPU socket**
3. SSD access patterns
4. Remote network delay

Q32. What distinguishes HTC from HPC in terms of workload characteristics?

1. **HTC jobs are loosely coupled and independent; HPC jobs are tightly coupled and synchronous**
2. HTC runs on GPUs only
3. HPC has no scheduling overhead
4. HTC is limited to file-based output

Q33. Which layer of the OSI model does IP operate on?

1. Data Link
2. **Network Layer (Layer 3)**
3. Application
4. Transport

Q34. Which of the following applies to PaaS services?

1. Gives control of hypervisors
2. Requires user to manage OS patches
3. **Provides runtime environment and abstracts away the OS layer**
4. Eliminates need for databases

Q35. What is the function of the command `docker logs <container>`?

1. Kills the container
2. Starts a new container with logs
3. **Displays standard output and error logs from the container**
4. Pulls the image again

Q36. Which of the following best defines digital twin?

1. **A real-time digital model of a physical system using sensor feedback and simulation**
2. A synchronized Git repository
3. A 3D model in CAD
4. A microservice clone

Q37. What is the function of `docker-compose up`?

1. Builds images but doesn't start them
2. **Builds and starts all services defined in a docker-compose.yml file**
3. Uninstalls Docker Compose
4. Pulls images only

Q38. What is the core difference between DAS and SAN?

1. SAN is always slower
2. DAS uses RAID 5 only
3. **DAS is directly connected to a single machine; SAN is network-attached block storage**
4. DAS has no filesystem