

INTERNATIONAL ADVANCED LEVEL
INFORMATION TECHNOLOGY

UNIT 2

UNIT 3: Topic 13: Enabling Technologies



Academic Year: 2024-2025

✍ Prepared by: Irikefe Hope Oniovosa

1. VIRTUALISATION

A. Concept & Benefits

1. What Is Virtualisation?

- The creation of **software-based environments** (virtual machines or containers) that mimic physical hardware or OS instances.
- Allows multiple “virtual” systems to run on a single physical host.

2. Key Benefits:

- **Resource Efficiency:** Share CPU, memory, and storage across multiple virtual instances to maximize hardware usage.
- **Isolation:** Each VM or container is sandboxed, so faults or security breaches in one don't affect others.
- **Scalability:** Spin up or tear down new environments on demand to match workload.
- **Testing & Development:** Create disposable test environments that mirror production without needing extra hardware.

B. Virtualisation Techniques

1. Containerisation (OS-Level Virtualisation)

- **Definition:** Packages an application and its dependencies into a lightweight container that shares the host OS kernel.
- **Examples:** Docker, Kubernetes.
- **Characteristics:**
 - Fast startup/shutdown.
 - Small overhead (no full guest OS).
 - Ideal for microservices and rapid deployment.

2. Virtual Machines (Hardware-Level Virtualisation)

- **Definition:** Emulates entire hardware stack, including its own guest operating system on top of a hypervisor.
- **Examples:** VMware Workstation, VirtualBox, Hyper-V.
- **Characteristics:**
 - Full OS isolation.
 - Greater overhead (each VM runs its own kernel).

- Suitable for running multiple different OS types or legacy environments.

C. Activity: Containers vs. VMs

1. Comparison Table

Feature	Containers	Virtual Machines
Isolation	OS-level (shared kernel)	Hardware-level (full kernel)
Overhead	Low	High
Startup Time	Seconds	Minutes
Use-Case Examples	Microservices, CI/CD	Legacy apps, multi-OS test
Resource Usage	Efficient, lean	Less efficient, heavier

2. Group Task (5 minutes):

- **In Small Groups**, list **three scenarios** where you would choose containers over VMs, and **three** where VMs are more appropriate.
- **Share** one scenario per group with the class.

2. DISTRIBUTED SYSTEMS

Concept & Need

- **Definition:** A distributed system is a collection of independent computers (nodes) that appear to users as a single, coherent system.
- **Use Cases:**
 - Global web services (e.g., DNS, CDNs)
 - Microservices architectures backing e-commerce sites
 - Big Data processing clusters (e.g., Hadoop, Spark)

Key Challenges

1. Failure & Fault Tolerance

- Nodes can crash or lose network connectivity.
- **Mitigation:** Replication, leader election (e.g., Raft, Paxos), health checks, automatic failover.

2. Concurrency

- Multiple processes access shared resources simultaneously.
- **Mitigation:** Distributed locks, consensus protocols, idempotent operations.

3. Replication & Consistency

- Keeping copies of data in sync across nodes.
- **Models:**
 - **Strong consistency:** All reads see the latest write (e.g., Google Spanner).
 - **Eventual consistency:** Updates propagate asynchronously (e.g., DynamoDB, Cassandra).

4. Performance

- **Latency:** Network delays between geographic regions.
- **Throughput:** Aggregate volume of operations the system can handle.
- **Mitigation:** Data partitioning (sharding), caching layers (e.g., Redis), CDN for static assets.

Case Study: Microservices Architecture

- **Example:** An online retailer splits its monolith into services—catalog, orders, payments, notifications.
- **Trade-Offs:**
 - **Pros:** Independent scaling, fault isolation, technology heterogeneity.
 - **Cons:** Operational complexity, network overhead, distributed tracing needed for debugging.

3. HUMAN-COMPUTER INTERACTION

Importance of UI/UX

- A well-designed interface drives user satisfaction, reduces errors, and boosts productivity.

Modalities

1. Visual

- Graphical layouts, color, typography, iconography.

2. Audio

- Voice commands (e.g., Siri), alerts, screen-reader audio.

3. Haptic

- Touch feedback—vibrations on mobile, force-feedback in gaming controllers.

Ergonomic Principles

- **Comfort:** Reduce strain with adjustable controls, clear fonts, adequate spacing.
- **Accessibility:** Support keyboard navigation, high contrast, scalable text sizes.
- **Minimize Cognitive Load:** Consistent layouts, familiar patterns, clear affordances.

Interface Design Patterns

- **Menus & Toolbars:** Group commands logically.
- **Icons & Buttons:** Use recognizable symbols; provide text labels.
- **Windows & Dialogs:** Focus on a single task per modal when needed.
- **Pointers & Cursors:** Change cursor shape to indicate interactivity (e.g., hand pointer).

Accessibility Considerations

- **Keyboard Focus:** Ensure all interactive elements can be reached via Tab/Shift-Tab.
- **Screen Readers:** Use ARIA landmarks and labels so content is read logically.
- **Contrast & Text Size:** Meet WCAG standards for readability.

Workshop: Sketching an Accessible App Interface

1. **Task:** On paper or whiteboard, sketch a simple mobile or desktop app screen (e.g., a login form or news feed).
2. **Focus:**
 - **Layout:** Clear visual hierarchy with headings, form fields, primary action button.
 - **Accessibility:** Show labels for inputs, high-contrast text, a visible focus indicator for keyboard users.
 - **Ergonomics:** Place primary buttons within easy thumb reach (mobile) or near cursor rest positions (desktop).
3. **Share & Critique:** Volunteers present their sketches; discuss strengths and potential improvements.

4. STORING AND SECURING DATA

A. Cloud Storage

1. Object Storage (e.g., Amazon S3)

- **Concept:** Stores data as discrete “objects” (file + metadata) in buckets.
- **Access:** Via REST APIs or SDKs (PUT/GET).
- **Use Cases:** Static assets, backups, logs, data lakes.
- **Key Features:** High durability (11 nines), virtually unlimited scale, pay-as-you-go.

2. Block Storage (e.g., AWS EBS, Azure Disk, Google Persistent Disk)

- **Concept:** Presents raw disk volumes that you attach to VMs.
- **Access:** Mounted as file systems (ext4, NTFS) by the guest OS.
- **Use Cases:** Databases, virtual machines, applications requiring low-latency I/O.
- **Key Features:** Consistent performance, snapshot capabilities, encryption at rest.

B. Security Mechanisms

1. File Encryption

- **Symmetric Encryption:**
 - Uses a single shared key (e.g., AES-256) to encrypt/decrypt.
 - **Example (OpenSSL):**

```
bash
CopyEdit
# Encrypt
openssl enc -aes-256-cbc -salt -in data.csv -out data.csv.enc \
-pass pass:MyStrongP@ssw0rd
# Decrypt
openssl enc -d -aes-256-cbc -in data.csv.enc -out data.csv \
-pass pass:MyStrongP@ssw0rd
```

- **Password Protection:**
 - Keys/passwords must be managed securely (e.g., AWS KMS, Azure Key Vault).

2. DBMS Controls

- **User Authentication:**
 - Enforce strong passwords, use integrated identity (LDAP, AD).
- **Role-Based Access Control (RBAC):**
 - Define roles (e.g., reader, writer, admin) and grant privileges to roles.
 - Assign users to roles rather than granting per-user privileges.
- **Views:**
 - Create restricted views that expose only certain columns or rows.
 - Users query the view instead of the base table.

C. Hands-On: Create a Restricted-Access Database User

Platform: PostgreSQL (commands work similarly in MySQL)

1. Connect as Admin:

```
psql -U postgres -d schooldb
```

2. Create a Read-Only Role & User:

```
-- Create a role with no login, owning no privileges
CREATE ROLE readonly NOINHERIT;
GRANT CONNECT ON DATABASE schooldb TO readonly;
```

3. Grant Schema Usage & Table SELECT:

```
-- Allow usage of public schema
GRANT USAGE ON SCHEMA public TO readonly;
-- Allow selecting from tables
GRANT SELECT ON ALL TABLES IN SCHEMA public TO readonly;
-- Future tables
ALTER DEFAULT PRIVILEGES IN SCHEMA public
    GRANT SELECT ON TABLES TO readonly;
```

4. Create a User and Assign Role:

```
CREATE USER reporting_user WITH PASSWORD 'SecureP@ss!';
GRANT readonly TO reporting_user;
```

5. (Optional) Create a Restricted View:

```
-- Only expose student names and majors
CREATE VIEW student_overview AS
    SELECT student_id, first_name, last_name, major
    FROM students;
GRANT SELECT ON student_overview TO readonly;
-- Revoke direct table access if needed
REVOKE SELECT ON students FROM readonly;
```

6. Test as the New User:

```
psql -U reporting_user -d schooldb
SELECT * FROM student_overview;    -- Allowed
SELECT * FROM students;
```

5. ENCRYPTION

A. Symmetric Encryption

- **Principles:**
 - Uses a single **shared secret key** to both encrypt and decrypt data.
 - Fast and efficient for large volumes of data.
- **Common Algorithm:**
 - **AES (Advanced Encryption Standard):**
 - Block cipher with key sizes of 128, 192, or 256 bits.
 - Widely used for disk encryption, HTTPS, VPN tunnels.
- **Key Management Challenge:**
 - Securely distributing and storing the shared key without interception.

B. Asymmetric Encryption

- **Principles:**
 - Uses a **public** key (to encrypt) and a **private** key (to decrypt).
 - Enables secure key exchange over untrusted networks.
- **Common Algorithm:**
 - **RSA:** Based on the mathematical difficulty of factoring large primes.
- **Use Cases:**
 - Secure email (PGP), TLS handshakes, digital signatures.

C. Certificate-Based Encryption

- **Digital Certificates:**
 - Bind a public key to an entity (person, server) via a **Certificate Authority (CA)**.
 - Contain identity information, public key, expiration date, and CA's signature.
- **Certificate Authorities:**
 - Trusted third parties (e.g., Let's Encrypt, DigiCert) that issue and revoke certificates.
- **Use Cases:**
 - HTTPS websites, code signing, enterprise VPN authentication.

D. Demonstration

1. **Symmetric Example (AES):**

- Use a tool or script to encrypt a plaintext message with AES-256 and then decrypt it using the same key.

2. **Asymmetric Example (RSA):**

- Generate an RSA key pair.
- Encrypt the same plaintext using the **public** key.
- Decrypt it using the **private** key.

3. **Certificate Example:**

- Show a simple HTTPS connection in a browser, inspect the server certificate, and highlight the CA signature and validity period.

ASSESSMENT

Formative

- **Quick Quiz:**
 - Identify differences between symmetric, asymmetric, and certificate-based encryption.
 - Match algorithms (AES, RSA) to their appropriate use cases.
- **Group Discussion:**
 - Debate pros and cons of symmetric vs. asymmetric encryption for a given scenario (e.g., bulk data transfer vs. secure key exchange).
- **Peer Review:**
 - Exchange HCI interface sketches and evaluate accessibility concerns identified earlier.

Summative

- **Project:**
 - **Containerized App Deployment:**
 - Deploy a sample microservice in a Docker container.
 - Secure its data at rest with AES encryption.
 - Configure a PostgreSQL instance with two roles (admin and read-only) and demonstrate role-based access controls.
- **Written Assignment:**
 - **Essay (500–700 words):**
 - Compare symmetric vs. asymmetric encryption: cover performance, key management, and scalability.
 - Describe real-world use cases where each method is preferred (e.g., disk encryption, TLS handshakes, email security).

ADDITIONAL RESOURCES

Videos

- **Basics of Virtualization and Containerization**
 [Basics of Virtualization and Containerization](#) YouTube
- **Explaining Distributed Systems Like I'm 5**
 [Explaining Distributed Systems Like I'm 5](#) YouTube
- **Cryptography Basics: Intro to Cybersecurity**
 [Cryptography Basics: Intro to Cybersecurity](#) YouTube

Websites

- **Docker Documentation**
 <https://docs.docker.com>
- **VirtualBox Documentation**
 <https://www.virtualbox.org/wiki/Documentation>
- **MDN Web Docs (Accessibility)**
 <https://developer.mozilla.org/docs/Web/Accessibility>
- **OWASP Cryptographic Storage Cheat Sheet**
 https://owasp.org/www-project-cheat-sheets/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html

Tools

- **Docker** (<https://www.docker.com>)
- **VirtualBox** (<https://www.virtualbox.org>)
- **OpenSSL** (<https://www.openssl.org>)
- **MySQL** (<https://www.mysql.com>)
- **PostgreSQL** (<https://www.postgresql.org>)