CHAPTER 7

IT SYSTEMS ARCHITECTURE

IT systems architecture is not merely a technical blueprint — it is a **strategic framework** that **aligns technology with business goals**, so organisations can **scale, innovate and stay competitive**. Covers how **hardware, software, networks and storage** interact, design **best practices**, emerging trends (cloud-native, edge, AI) and **programming languages**.

CHAPTER ROADMAP

1. Key components — 4
2. Interactions & dependencies — 3
3. Scalability & flexibility — H vs V
4. Best practices for design — 4
5. Latest trends — 4
6. Programming languages — types

DEFINITION

IT systems architecture refers to the overall design and structure of an organisation's IT systems — the hardware, software, networks and protocols that enable it to function efficiently.

A well-designed architecture ensures systems **work together seamlessly**, supporting **operational requirements, scalability and security**; it determines the **efficiency, performance and adaptability** of the infrastructure.

1 KEY COMPONENTS OF IT SYSTEMS ARCHITECTURE

HARDWARE — physical foundation

SOFTWARE — programs that run on it

NETWORKS — the "circulatory system"

STORAGE — holds data & information

1.1 Hardware — physical devices & infrastructure that support IT systems

Servers

Powerful computers that **handle requests from clients** and provide resources — data storage, application hosting, network services. **Backbone** of IT infrastructure (databases, email, web services).

e.g. A **server farm / data centre** hosts the company's applications and databases.

Storage Devices

Hard drives, SSDs, network-attached storage (NAS) store data & applications; accessible when needed and backed up for **disaster recovery**.

e.g. Cloud storage (Google Cloud, Microsoft Azure) → scalability, redundancy, access anywhere.

Networking Equipment

Routers, switches, firewalls enable communication between devices; smooth data flow and **security by controlling access**.

e.g. A corporate **intranet** connects all computers for secure access to shared files.

Peripheral Devices

Input/output devices — keyboards, mice, printers, scanners — allow users to **interact with the system**.

STRATEGIC VIEW — A well-designed architecture **aligns IT infrastructure with business goals** → innovation, efficiency, growth, and the ability to respond to changing market demands.

1.2 Software — applications & programs that run on hardware

System Software
OS & utilities

Operating systems and **utility programs** that manage hardware resources and provide a platform for applications.

Role: OS is the **intermediary between hardware and application software** — manages memory, processing power, storage.

e.g. Windows, Linux, macOS, Unix

Application Software
task-specific

Programs that help users perform **specific tasks** — word processing, accounting, CRM.

Role: perform business functions, analyse data, interact with the system.

e.g. Microsoft Office, SAP ERP, Google Chrome

Middleware
the connector

Software that **connects different applications and systems** so they can communicate & share data. Services: **messaging, authentication, database access**.

e.g. Airline integrates flight booking with third-party **payment gateways** & loyalty programs.

1.3 Networks — the "circulatory system" of IT architecture

Data flows between hardware, software and users; connect **internal** systems (workstations) and **external** entities (customers, partners). Rely on protocols like **TCP/IP**.

LAN · Local Area Network
Devices within a **limited area** (office, building) — **fast & secure** communication.

e.g. Corporate intranet for shared files, databases, applications.

WAN · Wide Area Network
Devices over a **larger geographic area** — multiple office locations; remote sites.

e.g. Multinational connects offices in different countries.

Internet
Global network of millions of devices — cloud services, customers, global sharing.

e.g. E-commerce platform connects with customers, processes orders.



1.4 Storage — holds data accessed by users & applications

LOCAL STORAGE
On **individual devices** (hard drives, SSDs). Suitable for **small-scale** needs but **lacks scalability**.
e.g., a PC stores files on its internal hard drive.

vs

CENTRALIZED STORAGE
In a **data centre or cloud → scalability, data redundancy, easy accessibility**.
e.g., Google Drive or Dropbox to share files across the organisation.

2 IT SYSTEMS INTERACTIONS & DEPENDENCIES

Components must work together seamlessly; dependencies need **careful management** for **reliability and performance**.

Hardware ↔ Software
compatibility

Must be **compatible** for optimal performance. OS **manages hardware resources**; applications **use them** to execute tasks.

e.g. Virtual machines run multiple OSs & apps on a single physical server.

Software ↔ Network
connectivity

Apps rely on the network to exchange data & collaborate; **network performance directly impacts** app efficiency.

e.g. Video conferencing needs a fast, stable connection.

Storage ↔ Network
cloud dependency

With cloud computing, storage is **network-dependent**; network **speed & reliability** determine retrieval speed.

e.g. Files in Google Drive / Dropbox need reliable internet to upload/download.

3 SCALABILITY & FLEXIBILITY IN IT SYSTEMS

SCALABILITY

Capacity of a system to handle increasing workloads by adding resources.

FLEXIBILITY

Systems can integrate new technologies, adjust to organisational needs and stay agile.

HORIZONTAL · SCALE-OUT



add MORE machines / servers

VERTICAL · SCALE-UP



make ONE machine more powerful

Horizontal Scaling (Scale-out)

Add **more machines/servers**; workload **distributed across nodes**. Common in **cloud** environments & distributed computing.

e.g. Netflix adds servers during traffic spikes (new show, live concert) → smooth streaming.

- **No hardware limitations** — no theoretical limit on machines
- **Fault tolerance** — load shifts if one fails
- **Distributed computing** — parallel processing (DBs, web, CDNs)
- **Elasticity** — AWS, Google Cloud, Azure add/remove virtual servers

Vertical Scaling (Scale-up)

Add **more resources (CPU, RAM, storage)** to an **existing server**.

e.g. Bank's transaction **database server** upgraded with more CPUs, memory, storage for peak hours.

Advantages

- **Simplicity**
- **Fewer dependencies**
- Ideal for **monolithic** apps

Limitations

- **Hardware limits** (max CPUs/RAM)
- **Single point of failure**
- **Cost** of high-end hardware

4 BEST PRACTICES FOR DESIGNING IT SYSTEMS ARCHITECTURE

Adapt to evolving business needs, **minimise downtime** and **protect against security threats**.

① Modular Design

microservices

Break the system into **smaller, independent components**, each with a specific function (**microservices architecture**). Changes to one module **don't affect others**.

- Flexibility
- Ease of maintenance
- Improved collaboration

EXAMPLE & USE CASE

Web app: authentication, payment, inventory as separate services — upgrade payment without touching authentication. E-commerce: add recommendation features without affecting order processing.

② Redundancy & Failover

backups

Redundancy = backups for critical components. **Failover** = **automated** switch to a backup when the primary fails → minimal downtime.

- High availability
- Resilience
- Business continuity

EXAMPLE & USE CASE

Online bank with servers in multiple regions — power outage → auto-switch to another region. Cloud storage replicates data across data centres.

③ Security by Design

every layer

Integrate security **at every layer — not an afterthought**. Includes encryption, access control, IDS, regular security audits.

- Encryption (transit & rest)
- RBAC access controls
- Intrusion detection & prevention

EXAMPLE & USE CASE

Healthcare: end-to-end encryption, RBAC for doctors/nurses, IDS. Bank: **MFA**, RBAC, encrypted transactions, incident response plan.

④ Scalability & Flexibility

future growth

Design to **accommodate growth** and fluctuating demand; integrate new technologies seamlessly.

- Cloud services (AWS, Azure, GCP)
- Virtual machines
- Distributed systems

EXAMPLE & USE CASE

Startup scales cloud resources with users, cuts back in low demand. Retailer uses **auto-scaling** for Eid sales, then scales down for cost efficiency.

Practice	Main purpose	Keyword to spot in an MCQ
Modular design	Change one part without affecting others	microservices, independent modules
Redundancy & failover	Keep services running when something fails	backup, automatic switch, high availability
Security by design	Protection built into every layer	not an afterthought, encryption, RBAC, IDS
Scalability & flexibility	Grow / shrink with demand	auto-scaling, cloud, VMs, distributed

5 LATEST TRENDS IN IT SYSTEMS ARCHITECTURE

Cloud-Native

Kubernetes & containerisation → flexible, scalable deployments.

e.g. Startup launches a global app with minimal hardware.

Edge Computing

Processing shifts **closer to data sources** → real-time, less latency.

e.g. Smart city edge servers analyse traffic in real time.

AI Integration

Machine learning optimises resource allocation.

e.g. Google's data centres cut cooling costs by **40%**.

Zero Trust Security

Continuous verification replaces traditional perimeters.

e.g. Tech firm secures **5,000 remote employees**.

6 INTRODUCTION TO PROGRAMMING LANGUAGES

Types — by level of abstraction, functionality and purpose

◀ close to hardware (machine code, assembly) — easy to read, problem-focused ▶

Low-Level

Assembly & machine code; direct control of memory & processor. For **OS, device drivers, embedded systems**.

but difficult; needs hardware knowledge

High-Level

Easy to read; focus on problems, not hardware; built-in features & libraries.

e.g. Python, Java, C++ — websites, software, business programs

Scripting

Usually **interpreted** — runs **line by line**, no separate compilation.

e.g. JavaScript, Python, PHP — web dev, automation, quick apps

Domain-Specific (DSL)

Designed for **specific tasks** in specialised fields.

e.g. SQL (databases), **HTML/CSS** (web design), **R** (data analysis)

Popular Programming Languages

Language	Known for	Key point
Python	Simplicity & versatility — web, **data science**, automation, **AI**	Libraries: **NumPy** (numerical), **TensorFlow** (ML); rapid prototyping
JavaScript	**Backbone of web development** — interactivity	React, Node.js, Angular → **full-stack** in one language
Java	Robust, **object-oriented** — enterprise apps, **Android**, web services	**"Write once, run anywhere"** via the **JVM**
C++	Extension of C — **performance & control** of resources	OS, embedded, games, simulations; **manual memory management**
SQL	**Standard for relational databases**	**Declarative** — specify *what*, not *how*

EXAM FOCUS · MCQ TRAPS

commonly confused points

1 Scale-out vs scale-up
Horizontal = **scale-out** = more machines. Vertical = **scale-up** = bigger machine.

2 Single point of failure
A limitation of **vertical** scaling. Horizontal offers **fault tolerance**.

3 Monolithic apps
Benefit from **vertical** scaling — not designed for distributed computing.

4 Middleware
Connects **applications** to each other (messaging, authentication, DB access) — not an OS.

5 System vs application software
Windows/Linux = **system**. MS Office, SAP ERP, Chrome = **application**.

6 LAN vs WAN
LAN = one office/building. WAN = **multiple sites** / countries.

7 Failover
An **automated** switch to backup — part of redundancy; gives **high availability**.

8 Security by design
Built in at **every layer**, **not an afterthought**.

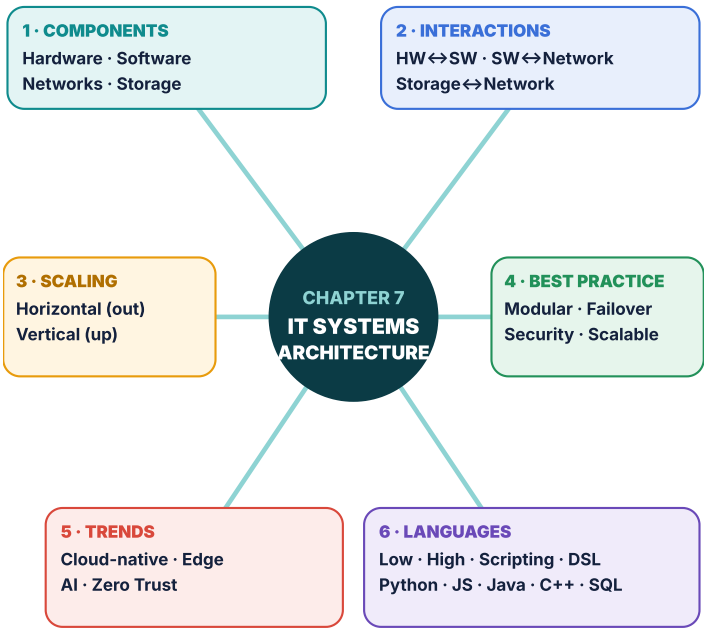
9 Scripting languages
Interpreted line by line — no separate compilation step.

10 SQL is a DSL
Also **declarative**. Java's portability = **JVM**. Kubernetes = **cloud-native**.

CLASSIFY IT

1	Netflix adds servers in a spike	Horizontal
2	Add RAM & CPUs to DB server	Vertical
3	Router, switch, firewall	Networking hardware
4	Booking ↔ payment gateway link	Middleware
5	Backup region takes over	Failover
6	Payment module upgraded alone	Modular design
7	Smart-city traffic analysed locally	Edge computing
8	Assembly for device drivers	Low-level

REVISION MAP



Build
Hardware + software + networks + storage.

Connect
Components depend on each other — esp. the network.

Grow
Scale out (more) or scale up (bigger).

Design well
Modular, redundant, secure, scalable.

Code
Pick the language level that fits the task.