* 1. **Enterprise Applications and Business Processes**

**What is an Enterprise Solution?**

* Refers to **one or more enterprise applications** that produce, process and exchange data in order to fulfil business requirements

**Characteristics of Enterprise Applications and Solutions**

* **Support business activities & processes**
* Store a lot of **persistent data**
  + In files, relational databases, NoSQL databases, etc.
* Data is often accessed and processed **concurrently**
  + Potentially for many internal and external users
* Provide a lot of **user interface** screens
  + The same data can be processed or viewed in different ways in different processes by different people.
* Not all processing happens real-time **online**; some **offline** (batch) processing is used.
  + Example: generating monthly financial statements
* Can be **large or small, simple or complex**
  + Examples: large complex CRM; small simple calendar app;
* **Process data** according to **business logics/requirements**
  + Can be in the form of executable rules. E.g., For every $10 purchased, give the customer 1 loyalty point
  + It is a challenge to **make applications easily adaptable to changing requirements**
* A large enterprise often has applications that were built over many years **using different technologies**
  + Often need toexchange data among applications
  + It is a challenge to **enable data exchange among various kinds of applications**

**4 Learning outcomes:**

1. Appreciate **the existence of different enterprise applications** used in an enterprise
2. Understand the **role of an enterprise application in the context of a business process**
3. Understand the **need for data exchange** between different enterprise applications
4. Understand the differences among the **three categories of enterprise applications**

Appreciate **the existence of different enterprise applications** used in an enterprise

* Enterprise applications may be referred to as **IT Systems**, **IT applications**, **software systems**, **software programs**, etc.
* An **activity** in a business process often relies on **enterprise applications**.
* An activity performed by a person in an enterprise can affect other parts of the enterprise or other enterprises according to the business processes

Enterprise applications supporting the activities **need to interact and collaborate**.

* The same application can be **(re)used** to support different activities in different business processes

Understand the **role of an enterprise application in the context of a business process**

* In the context of a business process, an **enterprise application** can
  + - Provide interface for a user to perform a business activity
    - Apply pre-defined business rules (logic) to process input data
    - Analyze data
    - Produce new data
    - Store data
    - Retrieve data
    - Send data to another application

Understand the **need for data exchange** between different enterprise applications

* An enterprise is bound to **have** **many applications** that support different business activities and handle different data relevant to different parts of the enterprise
* Some business processes **require applications from two or more enterprises** to collaborate
* **Not all activities or data can be supported or handled in one application**
* Automating the exchange of data among applications helps to **automate business processes** and improve productivity

Understand the differences among the **three categories of enterprise applications**

* 1. Commercial-Off-The-Shelf
  2. Custom
  3. Legacy

A close-up of a document

AI-generated content may be incorrect.

* 1. **Monolithic Applications versus Microservices**

1. **Learning outcomes:**
2. Define a **monolithic** application
3. Define a **microservice** and **microservices architecture**

Define a **monolithic** application

A diagram of function

AI-generated content may be incorrect.

* In the past, an enterprise application is usually **built as a single unit** with a single executable
* It usually comprises three main parts
  + Client-side User Interface (e.g., HTML pages and javascript running on the browser)
  + Server-side application that handles:
    - * The HTTP requests,
      * Executes business logic,
      * Retrieve and update data from the database,
      * Select and populate HTML view to be sent to the browser,
      * …
  + A database (consisting many tables covering multiple entities)
* The **server-side application** is usually deployed as **a single package**, referred to as a **monolithic application** or **monolith**

**Some Characteristics of a Monolith**

* Has a **single code base** implementing **all functions** needed for various business processes
* Typically, the entire application is **developed in one programming language** 
  + - Easier for different parts of the application to be compatible (PROS)
    - Less flexibility for the choices of technologies (CONS)
    - Less suitable for heterogeneous environments (CONS)
* Over the years, the **single code base** may become **large and complex**, difficult to understand and maintain
* **Deployment** of a monolith is often either **all or none**
  + Not easy to deploy only a part of the application due to dependencies
  + A change to one function often requires **redeploying the entire application package**
  + **Scaling only a specific function is not possible**, the entire application has to be scaled

Define a **microservice** and **microservices architecture**

A diagram of a microservice

AI-generated content may be incorrect.

A microservice may be defined as a single unit that implements only one or a few (instead of many) functionalities needed to support business requirements and can be used by other applications (or microservices) over the network in a standard interface that is independent of programming languages and platforms

**Some Characteristics of a Microservice**

* Microservices are **"loosely coupled"** with each other
  + It can be implemented in a **programming language of its own**; E.g., one microservice is implemented in Java; the other in Python;
  + It can be deployed and run on a **platform of its own**; E.g., one microservice runs on Windows; the other on Linux;
  + It usually **has its own data store** when a data store is needed; E.g., an Order microservice has its own database with an Order table; a Customer microservice has its own database with a Customer table
  + It can be **scaled independently** E.g., instances of the same microservice that is more frequently used than others can be deployed to many machines to support concurrent processing
  + Its implementation can be **changed independently**, as long as its interface for invocation remains the same
* Microservices architecture is a **style** or **pattern**, not rigorous rules
* Each microservice is **relatively small and simple** in comparison with usual enterprise applications
* A microservice can be developed and managed by a small agile team independently
* Each development team can manage one or a few microservices
* Microservices **exchange data** with each other through a standard interface by using **commonly used data formats and communication technologies**
  + JSON over HTTP transport; messaging
* Microservices architecture promotes a **methodology** that develops an enterprise application or solution as an **assembly of loosely-coupled microservices**

**Comparisons between monolithic application and microservices architecture:**

A table with text on it

AI-generated content may be incorrect.

**1.4 REST APIs  
JSON Data Format**

**2 Learning outcomes:**

1. Demonstrate an understanding of some basic concepts of REST APIs
2. Understand the role of JSON in helping applications exchange data

Demonstrate an understanding of some basic concepts of REST APIs

* Representational State Transfer (REST)
* A retrospective definition of an architectural style that includes a set of principles, properties, and constraints for HTTP-based applications
* It uses the following **standards:**
* **HTTP** (Hypertext Transfer Protocol)
* **URL** (Uniform Resource Locators, e.g., https://www.google.com/maps) or **URI** (Uniform Resource Identifiers, e.g., a 10-digit ISBN 0-599-87118-0; a DOI 10.1145/3106237.3121282)
* **JSON**, **HTML**, XML, JPEG, etc.
* APIs (Application Programming Interfaces)

A screenshot of a computer

AI-generated content may be incorrect.

**Some Characteristics of REST**

* **Client-server** model
* **Simple uniform** (standardized) interface
* Each API is **language and platform agnostic**
  + Its implementation can be in any language of choice.
  + Its deployment can be on any platform of choice.
  + It can be invoked on any OS and with any language.
* Typically supports only the **HTTP / HTTPS transport**
* One type of **WEB SERVICE**
* Supports a **variety of data formats**
  + TEXT, JSON, HTML, XML, OData, JPEG, MP4, etc.
* Uses **"self-describing"** data
* **Layered** system
  + Allow proxies, gateways, firewalls, caches, load balancers, etc. in-between clients and servers.
* Each API is often **stateless**
  + The server processes each API invocation in isolation; it doesn't maintain states about the client or previous API invocations.
  + The client needs to send all necessary context information to the server at the same time when invoking the API.
  + Lack standards for security, transactions, etc. that need to keep states.

**Examples of RESTful APIs:**

* + **Amazon**
  + **eBay**
  + **Facebook**
  + **Flickr**
  + **Google**
  + **PayPal**
  + **SendGrid**
  + **Twitter**
  + **Yahoo**

Understand the role of JSON in helping applications exchange data

* JSON (JavaScript Object Notation) is a lightweight data-interchange format
* **Easy** for humans **to read and write**
* **Easy** for machines to **parse and generate** 
  + Many languages have built-in supports
* Uses a **text format** that is **independent of programming languages and platforms**
* JSON can be **"self-describing"**
* often both **human readable** and **machine-readable**
* **Allow hierarchy**, that is, values within values
* Can use **JSON schema** to define and validate the correctness of JSON data
  + Conceptually similar to defining and validating database tables according to a database schema

**Example of JSON document:**

A screenshot of a computer code

AI-generated content may be incorrect.

**JSON syntax rules:**

* Data is a collection of **values**
* Values have **types** (a.k.a. data types)
  + number, string, Boolean (true or false), array, object, null (an empty value)
* **Curly braces hold objects**
  + An object is an **unordered** set of **name-value pairs**
  + A name is a string, also referred to as an attribute or a key
  + A value can be of any allowed type
  + **Pairs are separated by commas**
  + **The name and the value in a pair is usually separated by a colon ':'**
* Square brackets hold arrays
  + An array is an ordered list of zero or more value of any allowed type
  + **Array values are usually separated by commas**

**A screenshot of a computer code

AI-generated content may be incorrect.**

**1.5 Basic Networking  
HTTP**

**2 Learning outcomes:**

1. Demonstrate an understanding of some fundamental concepts of **networking**
2. Demonstrate an understanding of **HTTP Methods**

Demonstrate an understanding of some fundamental concepts of **networking**

* -IP address
* DNS
* -CMD Window
* -DHCP
* -Ping
* -Loop Back Address
* -Port Numbers
* -Load Balance
* -Network Address Translation (NAT)

**IP Address**

**A diagram of a computer network

AI-generated content may be incorrect.**

Each machine may belong to multiple (sub)networks at the same time. It can have multiple IP addresses at the same time too. And, different machines in different (sub)networks can have the same IP address at the same time.

Broadly speaking, two machines can communicate with each other as long as they have a way to identify each other in a network. But in reality, there are many other factors in play that may affect the communication, e.g., firewalls installed on the machines and in the network may block the communication; the machines may belong to different (sub)networks, and they don't recognize each other's IP; etc.

* Unique Identification of
  + A machine (also called a host) on the network
  + IPv4: 4-byte numbers separated by '.'  
    E.g., 10.10.16.18, 172.16.100.145, 192.168.0.101, 202.161.55.11
  + IPv6 addresses have 16 bytes

**DNS (Domain name System)**  
The process of DNS resolution involves converting a hostname (such as [www.example.com](https://www.example.com/)) into a computer-friendly IP address (such as [192.168.1.1](https://192.168.1.1/)). An IP address is given to each device on the Internet, and that address is necessary to find the appropriate Internet device - like a street address is used to find a particular home. When a user wants to load a webpage, a translation must occur between what a user types into their web browser ([example.com](https://example.com/)) and the machine-friendly address necessary to locate the [example.com](https://example.com/) webpage.

**CMD window on Windows**

* + ipconfig
    - To check addresses
* hostname
* To check hostname
* Static vs Dynamic address

**DHCP (Dynamic Host Configuration Protocol)**

DHCP (Dynamic Host Configuration Protocol) automatically assigns IP addresses and other network settings to devices on a network (while DNS (Domain Name System) translates human-readable domain names into IP addresses.)

**Ping**

* Ping is a tool used to **test whether a particular host/machine is reachable across a network**
* The program sends packets to the target host and listens for reply from the target

Example

* + - ping google.com
    - ping 10.10.15.17

**Loop Back Address**

* **127.0.0.1** is the standard IP address referring to the **current host**
* If a program tries to connect to 127.0.0.1, it is immediately ***looped back***to the machine that currently runs the program
* Usually used for testing purposes, where you do not have another machine for conducting the test
* This is also called **localhost**

**Port Numbers**

* Ports are used to identify a particular **process/application** running on a machine so that the data sent to the machine can be relayed to the right application

**Load Balancer**

* Route network traffics to different receivers and dispatch processing workloads among them
* The client/user may not need to know the actual servers’ IPs; they can communicate with the servers indirectly via intermediaries in the network, such as gateways, routers, load balancers, etc.
* The workload of receiving and processing data is thus (often) evenly distributed among all the receivers, which is known as load balancing (round robin or random)
* A queue is an example of a load balancer.

A diagram of a network and load balance

AI-generated content may be incorrect.

**Network Address Translation (NAT)**

**How NAT Works Step by Step**

1. **Device Sends Request**
   1. A device in the private network (e.g., 192.168.1.2) sends a request to a website (example.com).
   2. The request includes the source IP (192.168.1.2) and a randomly assigned source port (5001).
2. **Router Modifies the Packet**
   1. The router running NAT changes the source IP to the router’s public IP (203.0.113.1) and assigns a new source port (20001).
   2. The mapping (192.168.1.2:5001 → 203.0.113.1:20001) is stored in a translation table.
3. **Request Reaches the Destination**
   1. The website (example.com) receives the request from 203.0.113.1:20001 and responds to this address.
4. **Router Forwards the Response**
   1. When the router receives the response, it looks up the translation table and replaces 203.0.113.1:20001 with 192.168.1.2:5001, forwarding the packet to the correct internal device.

**Why is NAT Important?**

1. **IPv4 Address Conservation**
   1. Since IPv4 addresses are limited, NAT allows multiple devices to share a single public IP address.
2. **Security**
   1. NAT hides internal IP addresses, making it harder for external attackers to directly access devices.
3. **Flexibility**
   1. Allows internal networks to use private IP ranges (e.g., 192.168.x.x, 10.x.x.x, 172.16.x.x) without requiring a public IP for every device.

**HTTP Request-Response**

**A yellow rectangular sign with black text

AI-generated content may be incorrect.**

**A list of data being used

AI-generated content may be incorrect.**

**2.1 Basic Networking  
HTTP**

**4 Learning outcomes:**

1. Understand the benefits of exposing and using functionalities through **standard interfaces**
2. Know the concept of a **service** and a **microservice**
3. Know the concept of **Service-Oriented Architecture (SOA)**
4. Know the concept of building an enterprise solution as an **assembly of (micro)services**

**A diagram of a book store

AI-generated content may be incorrect.**

**A diagram of a microservices

AI-generated content may be incorrect.**

Understand the benefits of exposing and using functionalities through **standard interfaces**

* Enable **more convenient data exchange** among different enterprise applications by using commonly used data formats and communication technologies across platforms and networks
  + E.g., JSON over HTTP transport
* Language and platform agnostic
* Applications become **less coupled** with each other
  + Different applications can be developed and deployed with different technologies
  + All kinds of applications, including legacy ones, can be made more easily compatible with others, through **standardized** integration modules, adaptors, connectors, plug-ins, wrappers, etc.

Know the concept of a **service** and a **microservice**

* A service may be defined as:
  + A unit that provides **one or many functionalities** needed to support business requirements and **can be used** by other applications or services over the network via a **standard interface** that is independent of programming languages and platforms
  + Recap:

A close-up of a text

AI-generated content may be incorrect.

* The **Service Provider Interface** and the **Service Consumer Interface** refer to some **code** that is at the provider and consumer end respectively and enables data exchange between the provider and the consumer.A diagram of a product management system

  AI-generated content may be incorrect.

Interactions happen through the **service interfaces** (both provider and consumer interfaces), instead of direct function calls of the systems.

Know the concept of **Service-Oriented Architecture (SOA)**

* Consider turning all IT systems into services (using wrapper service)
* Each service can be small, simple, or large, complex
* Develop and maintain enterprise solutions as **assemblies** of loosely-coupled services
* The type of wrapper service is dependent on the type of IT system
* The functionality from lower layers can be **reused** by higher layers, saving the efforts needed to develop the functionalities need by the higher layers.
* **Each unit in each layer** may be developed and **maintained by different teams**.
* Manage assemblies at various **layers** of different functionality and/or complexity. Sample SOA layers with **services**:

A diagram of a computer network

AI-generated content may be incorrect.

Sample SOA layers with **microservices**:

A diagram of a network

AI-generated content may be incorrect.

An “**atomic**“ (micro)service often means a **self-contained** single unit that may not be further broken down into smaller units to be beneficial for an enterprise.

* It usually implements **simple** functionalities involving one kind of data entity.

A “**composite**“ (micro)service often means some functionality that is **composed** from other (micro)services. The composition of (micro)services is often implemented via IPC, *not* via normal function calls.

* It often involves functionalities that correspond to business processes, which are usually more **complex** than atomic (micro)services.

A white background with black text

AI-generated content may be incorrect.

**4.1 Docker  
Service Deployment**

**2 Learning outcomes:**

1. Understand the concepts of **containers** and **containerization** in the context of microservices
2. **Deploy** microservices using **docker** technology

Understand the concepts of **containers** and **containerization** in the context of microservices

* **Containerization** of a microservice
  + Package necessary items needed to run a microservice into an **image** that can be transferred to and run as a **container** on another machine
* A **Docker image** is a **lightweight, standalone, and executable package** that includes everything needed to run a piece of software:
  + - Code
    - Runtime
    - Libraries
    - Dependencies
    - Environment variables
    - Docker images act as **blueprints** for creating **containers**.
* What is a **Docker container?**
* It provides an **isolated environment** for applications to run consistently across different systems.
  + is a **runtime instance** of a Docker image
  + Example: Running a container from a "Flask App Image" launches an isolated Python web server.

**Deploy** microservices using **docker** technology

* What is a **Docker?**
* It is an open **platform** for developers and system administrators to **package**, **transfer**, and **run** microservices (or any\* application) on any\* machine either on premise or on the cloud
* What is a **Dockerfile?**
* **Dockerfile** contains the instructions for Docker to build an image (FROM, COPY, RUN, etc.)

A diagram of a docker

AI-generated content may be incorrect.