

# **Cloud Computing Fundamentals**

## **IN401 – M1S1 – 5 Credits**

# Course Grading

---

- Partial Exam
- Final Exam
- Second Session

# Course Contents

---

- **Chapter 1: Introduction to Cloud Computing**
- **Chapter 2: Infrastructure as a Service (IaaS)**
- **Chapter 3: Platform as a Service (PaaS)**
- **Chapter 4: Software as a Service (SaaS)**
- **Chapter 5: Cloud Native Technologies & Architectures**
- **Chapter 6: Cloud Security**
- **Chapter 7: Practical Case Studies**

---

## **Chapter 4:**

# **Software as a Service (SaaS)**

# Content

---

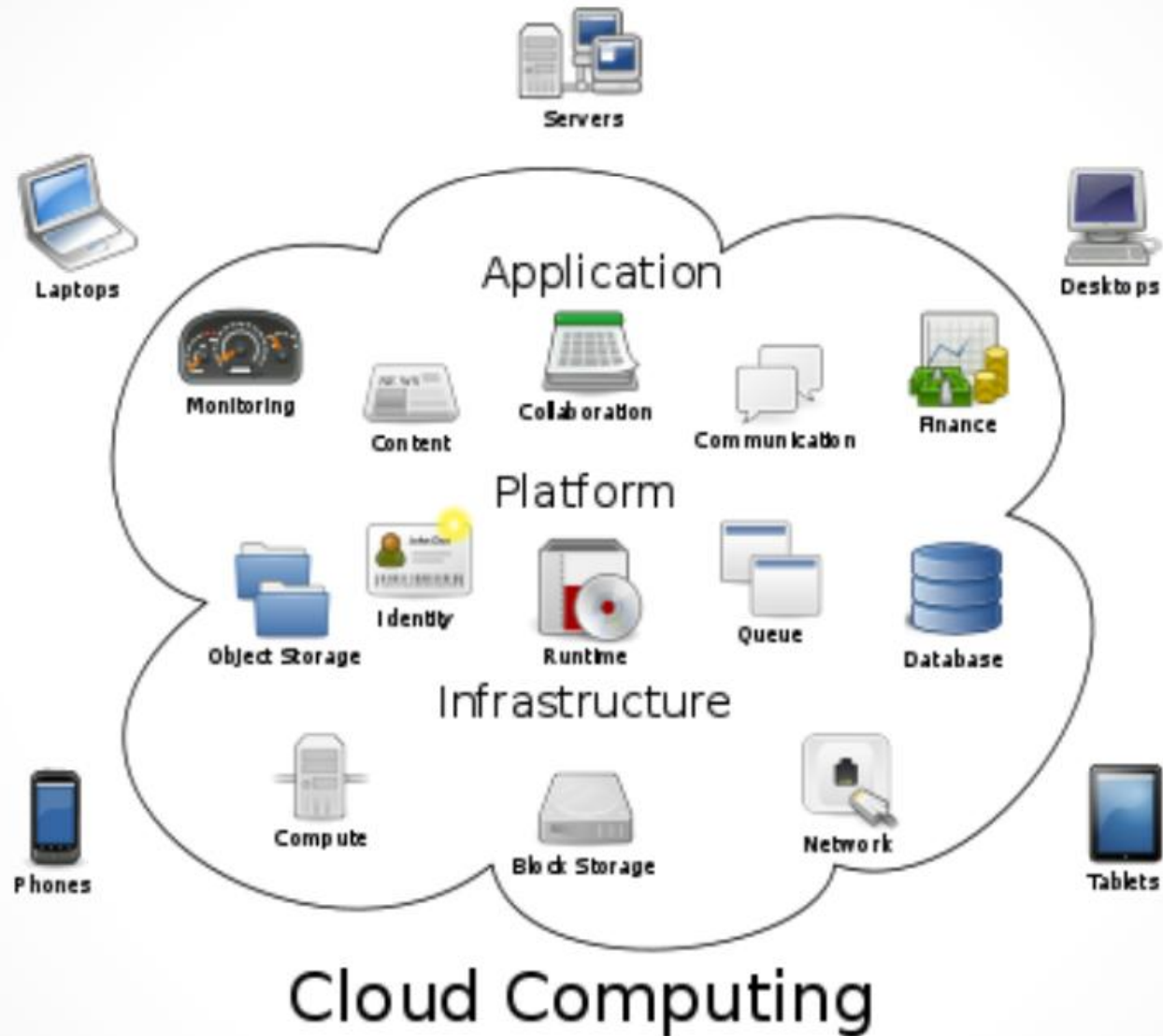
- Introduction
- SaaS architecture (SOA and multi-tenancy)
- SaaS integration concepts: APIs, authentication...

# Introduction

---

- [Software-as-a-Service \(SaaS\)](#) is a way of delivering services and applications over the Internet.
- Instead of installing and maintaining software, we simply access it via the Internet, freeing ourselves from the complex software and hardware management.
- It removes the need to install and run applications on our own computers or data centers eliminating the expenses of hardware as well as software maintenance.

# The Big Picture



Software as a Service is located in the application level of the stack

# What is SaaS ?

---

- **Definition:** Software as a Service (SaaS), a.k.a. on-demand software, is a software delivery model in which software and its associated data are hosted centrally and accessed using a thin-client, usually a web browser over the internet.
- Simply put, SaaS is a method for delivering software that provides remote access to software as a web-based service. The software service can be purchased with a monthly fee and pay as you go.

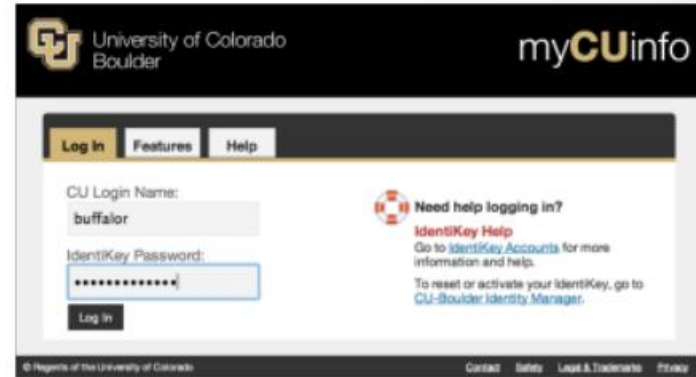
# An example

---

- Imagine you are the founder of a start-up company and you need to deal with large number of new customers.
- Buying a full version Customer Relationship Management (CRM) Software is expensive.
- With SaaS, you can buy a web-based CRM software that is pay as you go and scales to demand!
- Benefits: Save money on software license, cut cost on maintenance, and hardware purchase.  
Combined with lower start-up cost and a faster return on investment!

# SaaS is everywhere

---



# Amazon Web Services

---

- Beginning in 2006, Amazon web services provides a wide range of services and solutions for powering applications. They fall under the following categories:
  - o Storage
    - Amazon simple storage services (S3)
    - Amazon Elastic Book Store (EBS)
  - o Networking
    - Amazon Virtual Private Cloud (VPC)
    - Amazon Route53
  - o Database
    - Amazon Dynamo DB
    - Amazon Relational Database Service (RDS)
  - o Compute
    - Amazon Elastic Cloud Compute (EC2)
    - Amazon Elastic Map Reduce (EMR)

# Advantages

---

- o **Easy to use** - Most SaaS applications do not require more than a web browser to run.
- o **Cheap**- The pay as you go pricing model of SaaS makes it affordable to small businesses and individuals.
- o **Scalability**: SaaS application can be easily scaled up or down to meet consumer demand. Consumers do not need to worry about additional computing infrastructure to scale up.
- o Applications are **less prone to data** loss since data is being stored in the cloud.
- o **Velocity of change** in SaaS applications is much faster.

# Advantages

---

- o Compared to traditional applications; SaaS applications are **less clunky**. They do not require users to install/uninstall binary code on their machines.
- o Due to the delivery nature of SaaS through the internet, SaaS applications are able to run on a **wide variety of devices**.
- o Allows for **better collaboration** between teams since the data is stored in a central location.
- o SaaS favors an **Agile development** life cycle. Software changes and frequent and on-demand. Most SaaS services are updated about every 2 weeks and users are most time unaware of these changes.

# Drawbacks of SaaS

---

## **1. Robustness:**

SaaS software may not be as robust (functionality wise) as traditional software applications due to browser limitations. Consider Google Doc & Microsoft Office.

## **2. Privacy:**

Having all of a user's data sit in the cloud raises security & privacy concerns. SaaS providers are usually the target of hack exploits e.g. Google servers have been the target of exploits purportedly from China in the last several years.

# Drawbacks of SaaS

---

## **3. Security:**

Attack detection, malicious code detection,...

## **4. Reliability:**

In the rare event of a SaaS provider going down, a wide range of dependent clients could be affected.

For example, when Amazon EC2 service went down in April 2011, it took down FourSquare, Reddit, Quora and other well-known applications that run on it.

# Robustness

---

- SaaS applications may not be able to provide the same level of functionality as traditional applications. This is partly due to current limitations of the web browser. Consider Google doc and Microsoft Office.
- Most SaaS applications are intolerant to slow internet connections and this can lead to erratic behavior. For example, Google doc may not be synchronized well between teams in a low internet connection

# Privacy

---

Lots of issues arise with sensitive data stored in the cloud.

Common privacy questions include:

- Who has the access to the data? How to distribute the rights?
  - What type of data can be saved on the cloud, and locally?
- What about the confidential data?
- Don't we really have to worry about data sharing? Who is viewing our data, modifying the data, and re-distributing our data? With or without permission?
  - Data sharing between private and public clouds?

# Security

---

- SaaS applications are prone to attack because everything is sent over the internet.
- Data encryption and decryption.
- Communication protocols.
- Virtualization versus Multi-tenant architecture: which one is better in terms of the security?
- Transaction processing, networking issues.

# Reliability

---

- Although most SaaS applications are highly reliable, down time is still inevitable and can be very expensive-commercial SaaS software.
- The application, data, backups, everything are in the cloud, thus making it hard to recover from the server down time.
- You don't physically own the code, they are in the cloud

---

# **Service Oriented Architecture (SOA)**

# SOA (Service Oriented Architecture)

- SaaS is the methodology for providing services over the Internet.
- **SOA** is the software architecture that powers SaaS application and One of the most commonly seen practices for SaaS and cloud computing.

**Definition:** a set of principles and methodologies for designing and developing software in the form of interoperable services.

- It provides a way for consumers of services, such as web-based applications to be aware of available SOA-based services.

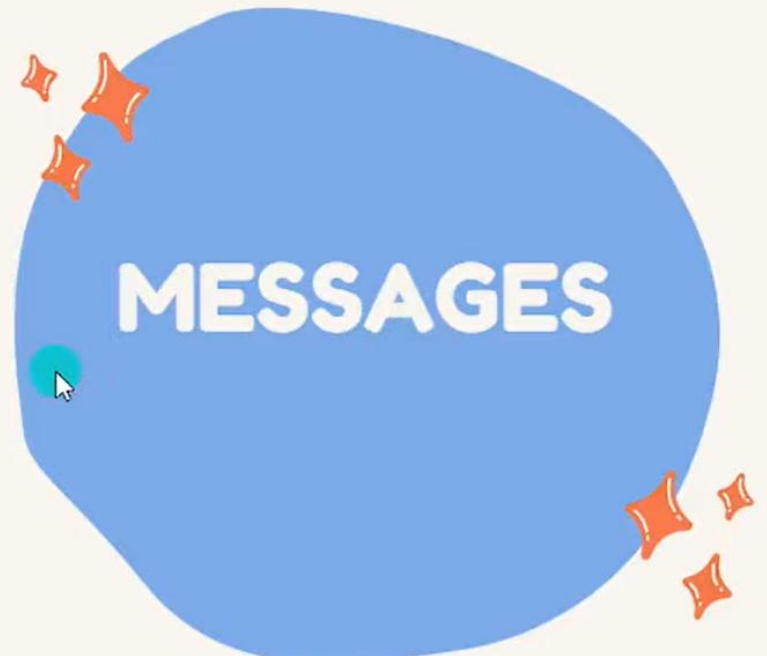
# Architecture: SOA

---

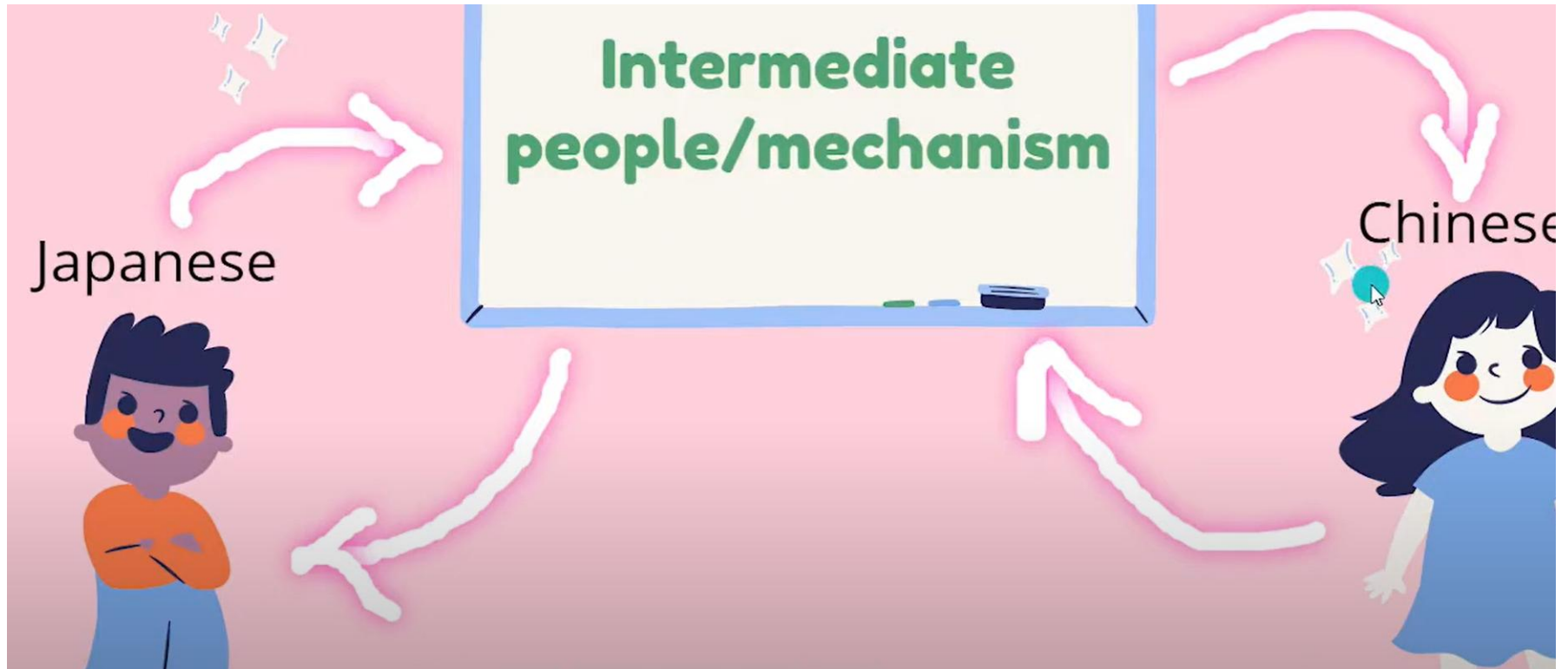
- SOA architecture structures applications by **breaking** them into independent, reusable software components called **services**, which communicate over a network using standardized interfaces.
- Then Applications are built as collection of independent services that communicate with each other over a network (through APIs)
- Instead of one big system (monolithic), we break it into smaller, reusable services- each doing one specific job- and those services talk together.
- Ex. Online Shopping app:
- Functions: Manage users, Products, Orders, notifications ...
- Services: User Service, Product service, Order Service, Notification Service...

- User Service: Handles login, signup, profile.
- Product Service: Stores product data
- Order Service: Manages orders and payments
- Notification Service: Sends emails or messages
- Each service: works independently, can be updated or replaced without affecting others and communicates via standard protocols (like http, rest, soap)
- Usually, services communicate via APIs in common formats (JSON or XML)

We can think SOA in 2 parts.

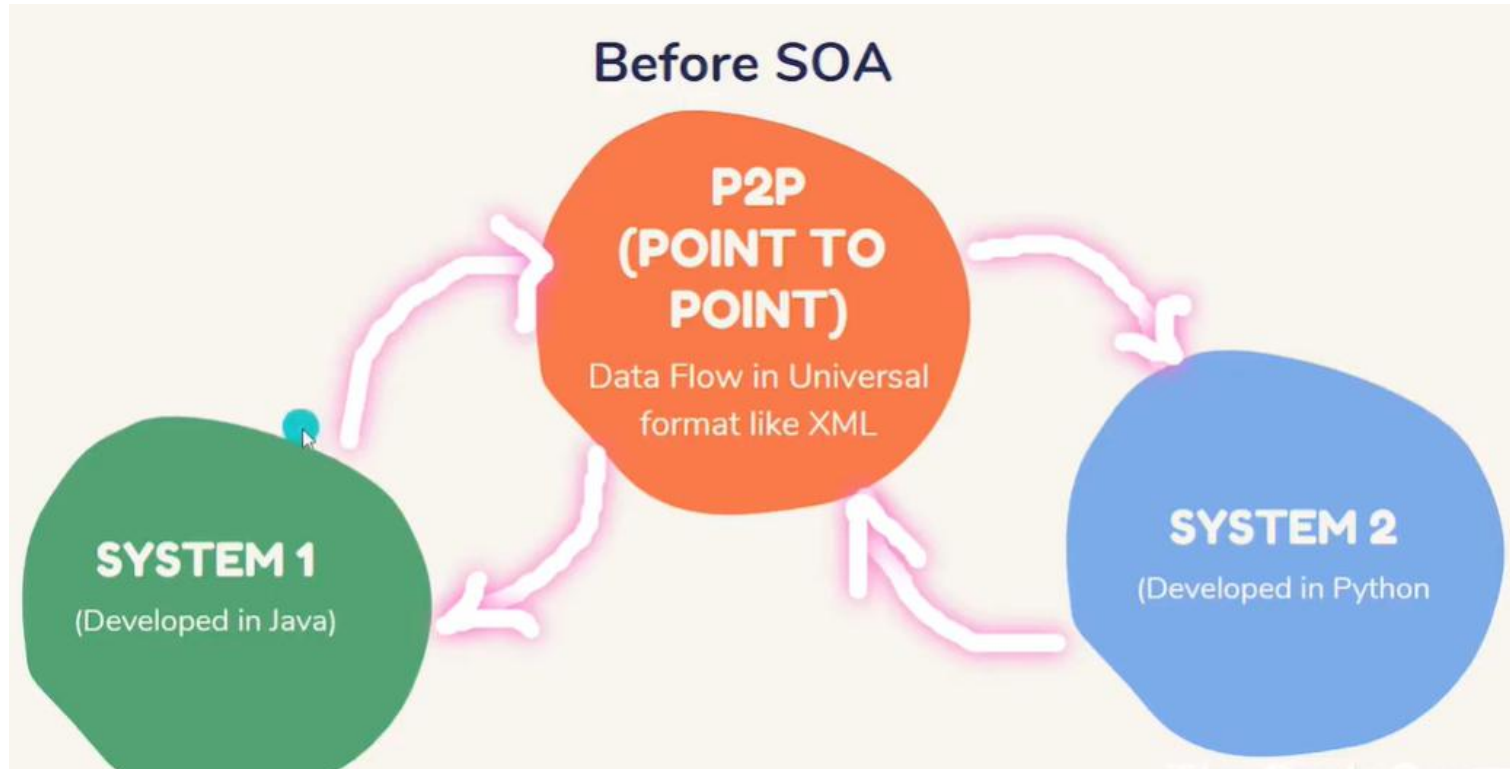


# SOA : Translator between systems



# Before SOA

---



# Before SOA

If System 2 Fails or Unvaliable



# Before SOA

If New System 3 Required to be added in Communication



TheCodeSpa

So System 3 needs to establish connectivity between 3-1 and 3-2

# With SOA: new layer is added

---



# With SOA

---

- System 1 required to integrate with system 2, then these systems do not need to know each other.
- System 1 needs to transfer data to SOA somehow. Then SOA is capable to transform the Data provided by System 1 to understandable format of System 2.
- If new System Comes then it only needs to plug in with SOA layer.
- When some System Fails then SOA can store & retry those requests to the target system when it's available.

# SOA key principles and characteristics

- **Services:**

Self-contained units of functionality, such as checking a customer's credit or processing a payment, that represent discrete business tasks.

- **Independence:**

Services are designed to operate with minimal dependencies on one another, making them easier to develop, deploy, and maintain.

- **Reusability:**

Services can be used in various applications and scenarios, reducing development time and costs.

- **Scalability :**

Up and down.

# SOA key principles and characteristics

- **Interoperability:**

SOA enables services to communicate and exchange data regardless of their underlying programming language, platform, or vendor.

- **Flexibility:**

in integrating different systems

- **Loose Coupling:**

Applications interact with services through standardized interfaces, abstracting the underlying implementation and reducing dependencies. This means when systems talk with each other the dependency which is required to talk is minimal. Failure of one system does not impact on other system.

# SOA vs Microservices

---

- **Microservices evolved from SOA**, and both aim to break large systems into smaller, manageable parts. They share common concepts.
- So, **microservices** are essentially a more granular, modern, independent **form** of SOA.

---

# Multi-tenancy

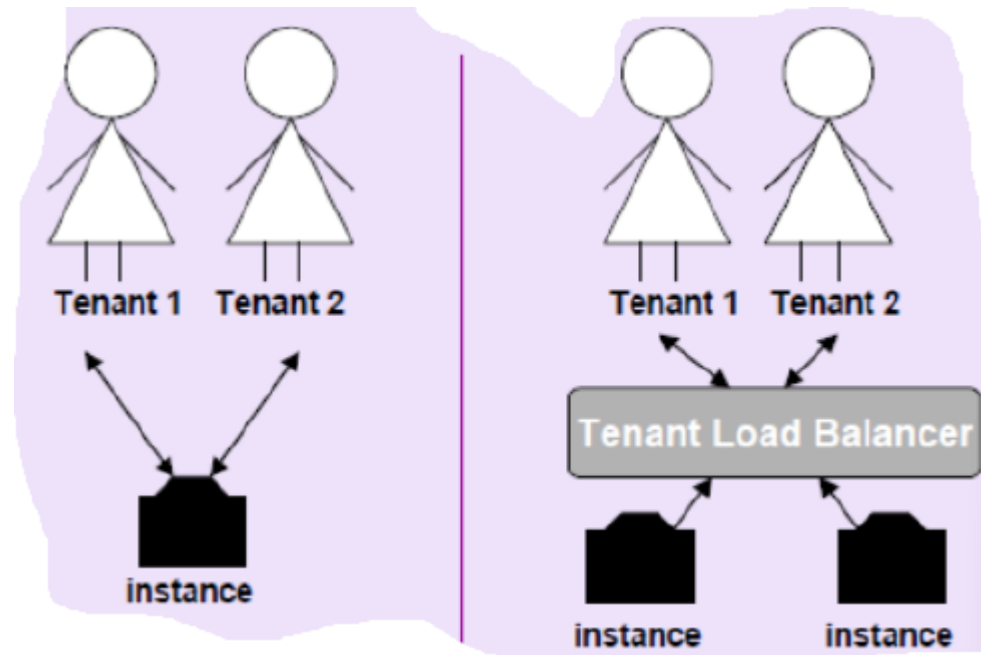
# Multi-tenancy architecture

---

- Multi-Tenancy refers to a principle in software architecture where a single instance of the software runs on a server, serving multiple client organizations.
- Multi-tenancy means one instance of a software application serves multiple customers, and each customer's: Data, Configurations, Workflows, and Access controls are kept isolated, even though they all use the same underlying system.

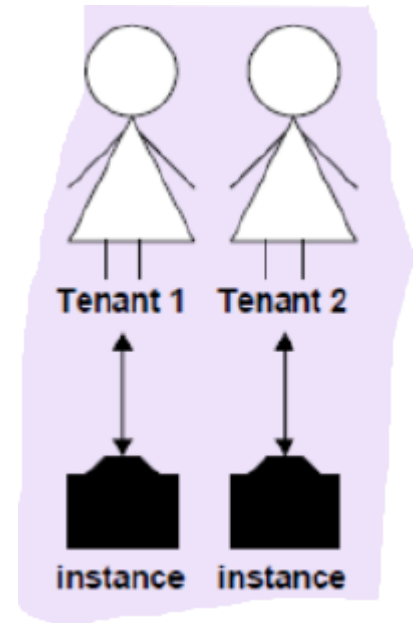
# Types of Multi-Tenancy Architectures

- 1. Shared Application + Shared Database
  - One application instance
  - One database
  - Tenants identified using tenant IDs
  - Cheapest and most scalable
  - Lower isolation



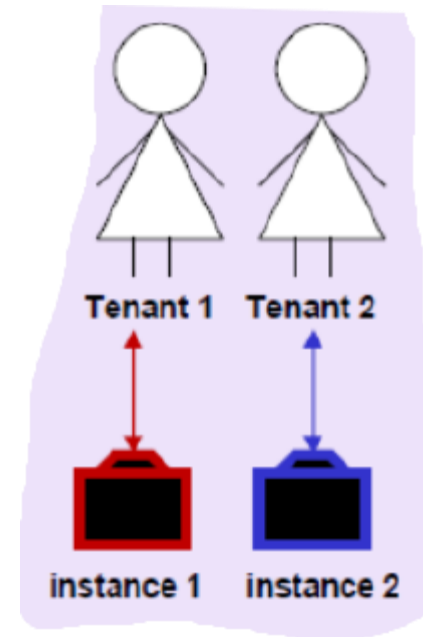
# Types of Multi-Tenancy Architectures

- 2. Shared Application + Separate Database
  - One application
  - Each tenant has its own database
  - Better security, logical isolation
  - More expensive



# Types of Multi-Tenancy Architectures

- 3. Separate Application + Separate Database
  - Each tenant has a dedicated app instance and DB
  - Highest isolation
  - Used for banking, government
  - Most expensive (rare in SaaS)



# Multi-tenancy advantages

---

- **Cost Efficiency:** Instead of giving each customer their own server, multiple customers share infrastructure.
- **Better Resource Utilization:** CPU, memory, and storage are used more efficiently.
- **Scalability:** Cloud providers can dynamically allocate resources.
- **Simplified Maintenance:** Updates and patches are applied once for the entire shared environment.
- **High Availability:** Cloud providers manage redundancy and failover for shared systems.

# Multi-tenancy disadvantages

---

## ■ 1. Performance Issues:

Tenants share CPU, memory, storage, and network. If one tenant uses too many resources (noisy neighbor problem), others may experience: Slow performance, Delayed response times, Reduced availability.

## ■ 2. Limited Customization:

Since all customers use the same application: Deep customization is difficult, Unique features for one tenant may affect others, Custom updates take longer, This is a challenge for businesses needing heavy personalization.

# Multi-tenancy disadvantages

---

## ■ **3. Complexity in Management:**

Ensuring tenant isolation requires: Complex access control, Advanced resource allocation, Tenant-aware logging & monitoring, Strong security policies, Architects must design carefully to avoid cross-tenant issues.

## ■ **4. Upgrade & Version Control Challenges:**

Updates apply to all tenants at once, which can cause: Downtime for all clients, Breaking changes for some tenants, Delays in rolling out features.

# Security in Multi-Tenancy

---

- **Advantages:**

- Tenant isolation (no tenant can see others' data)
- Access control (role-based)
- Encryption (data at rest and in transit)
- Network isolation (VPCs, firewalls)

- **Disadvantages:**

- Because multiple tenants share the same infrastructure, a misconfiguration or vulnerability may risk: Data leaks, Unauthorized access, Cross-tenant data exposure,...

---

# **SaaS Integration: Authentication**

# SaaS Integration

---

- **SaaS (Software as a Service)** refers to cloud-hosted applications that users access via the Internet (e.g., Google Workspace, Salesforce, Zoom, Office 365).
- **Integration** means connecting multiple SaaS applications together so they can **share data** and **work in sync**.
- Example: Linking **Salesforce** (CRM) with **Mailchimp** (email marketing) to automatically send emails to new leads.

# SaaS Authentication

---

## Core Concepts of SaaS Authentication

- Authentication is the security gatekeeper of a SaaS application, verifying a user's identity before granting access.
- In a multi-tenant SaaS environment, this needs to be both secure and flexible.

# Key Authentication Methods in SaaS

---

- **1. Username & Password:**

The most basic method, where users log in with credentials.

- **2. Multi-Factor Authentication (MFA):**

Adds additional verification such as: SMS code, Email OTP (One-Time Password), Authenticator app (Google Authenticator), Biometrics. It improves security, especially in multi-tenant SaaS.

- **3. Single Sign-On (SSO):**

Users log in once and access multiple systems. It uses common SSO protocols: SAML 2.0, OAuth 2.0, OpenID Connect. It is used in enterprise SaaS (e.g., Microsoft 365, Salesforce).

# Key Authentication Methods in SaaS

---

## ■ 4. Token-Based Authentication:

Instead of sending username/password each time, the system issues a token. Types of tokens: JWT (JSON Web Token), Access tokens, Refresh tokens. It is secure, stateless, and scalable for multi-tenant systems.

## ■ 5. OAuth 2.0 Authentication:

- Used for granting limited access to user accounts: "Sign in with Google", "Sign in with Facebook". Common in public SaaS applications.

## ■ 6. Tenant-Aware Authentication:

Subdomains: tenant1.app.com, Tenant IDs: Separate login pages, Tenant-specific identity providers (IdP)

# Authentication Challenges

---

## ■ Scalability

The authentication system must handle thousands or millions of logins.

## ■ Security

The authentication system must Prevent:

- Brute force
- Session hijacking
- Weak passwords
- Token theft

# Authentication and Authorization

---

- **Authentication** is the process of verifying who a user/application is, while **Authorization** determines what that authenticated user/application is allowed to do (Which data a user can access, Which features they can use, What actions they can perform).
- In SaaS integration, this is critical for security.
- The most common method for modern SaaS integration is using **Bearer Tokens** (usually derived from OAuth 2.0 or simple API keys).

# Authentication flow with Bearer tokens

Component	Description	Role in Security
<b>Client Sends Request</b>	The application initiates communication.	The request must include proof of identity and permission.
<b>Authorization Header</b>	A standard HTTP header used to carry the security token.	Format: <code>Authorization: Bearer &lt;TOKEN_STRING&gt;</code> . This token is the digital key.
<b>Bearer Token</b> (e.g., JWT, API Key)	A secret string issued by the SaaS provider, granting access.	It proves the client is authenticated and is authorized to access the requested resource. The token is checked on <b>every request</b> .
<b>SaaS Authentication Service</b>	The SaaS server component that verifies the token's validity and permissions.	If the token is invalid or expired, the server returns an <b>HTTP 401 Unauthorized</b> error.
<b>Access Granted/Denied</b>	The decision to fulfill the request.	If the token is valid, the request proceeds to the API logic. If the token is valid but lacks permissions (e.g., trying to delete a user without admin rights), the server returns <b>HTTP 403 Forbidden</b> .

# Example of OAuth 2.0 Flow

---

1. The user logs in with Google (OAuth provider).
  2. Google asks the user to grant permission to your app.
  3. If approved, Google sends back an **access token**.
  4. Your app uses that token to access Google APIs securely.
- 
- **No passwords are shared** — only temporary, scoped tokens.

---

# **SaaS Integration: API**

# API: the backbone of SaaS Integration

API (Application Programming Interface) is a set of rules that allows one software system to communicate with another.

- APIs define **how data is requested and exchanged** between services.

API Type	Description	Example
<b>REST API</b>	Uses HTTP methods (GET, POST, PUT, DELETE) to exchange JSON data	Google Maps API, GitHub API
<b>SOAP API</b>	Uses XML and strict contracts for enterprise systems	Salesforce SOAP API
<b>GraphQL</b>	Lets clients request exactly the data they need	Shopify, GitHub
<b>Webhooks</b>	Event-driven push notifications from one app to another	Stripe payment success callback

# SaaS API Integration

---

- APIs are the communication channels that allow different SaaS applications to share data and functionality. A well-designed API is crucial for successful integration.
  
- **A Typical SaaS Integration Flow:**
  1. A **trigger** occurs in a SaaS application (e.g., a new user signs up).
  2. The application uses its **API** to send this data to an integration platform or another system.
  3. The data is often processed by **middleware or an iPaaS** (Integration Platform as a Service), which transforms it into a format the destination system can understand.
  4. The transformed data is delivered to the target system (e.g., a CRM or ERP) via its API, completing the integration.

# SaaS API Design

---

To ensure your APIs are robust and scalable, consider these principles :

- RESTful Design:** Use nouns for resources (e.g., /users, /subscriptions) and proper HTTP methods (GET, POST, PUT, DELETE).
- API Versioning:** Version your APIs from the start (e.g., /api/v1/users) to avoid breaking existing integrations when you make updates.
- Rate Limiting:** Implement limits on how many requests can be made to your API to prevent abuse and ensure system stability.
- Secure Authentication:** Always use robust authentication like JWT tokens or API keys for your API endpoints

# API Communication Flow

Component	Description	Relevance to SaaS
<b>Client Application</b> (Your App)	The external software or system that needs data or service from the SaaS.	This is your React/Vite application (the <b>frontend</b> ) or your Node.js/Express app (the <b>backend</b> ).
<b>Request</b> ( <code>GET /users</code> )	A standardized HTTP message sent from the client to the server.	<b>CRUD</b> (Create, Read, Update, Delete) operations are usually mapped to HTTP methods: <code>GET</code> (Read), <code>POST</code> (Create), <code>PUT</code> / <code>PATCH</code> (Update), <code>DELETE</code> (Delete).
<b>SaaS API Endpoint</b>	The specific URL on the SaaS server that accepts the request.	This is the public entry point (e.g., <code>https://api.saasapp.com/v1/users</code> ).
<b>API Server/Logic</b>	The SaaS server-side code that executes the requested action.	Processes the request, interacts with the SaaS database, and generates the response.
<b>Response (Data)</b>	The data returned by the SaaS server, usually in <b>JSON</b> (JavaScript Object Notation) or XML format.	This data is then consumed and displayed by the Client Application.

# Other services

---

- Xaas: Everything as a service : from food to medical consultations....
- Aaas: Analytics as a service: from data to insights : ex. Outlier.
- Daas: Desktop as a service ex. Citrix
- Faas: Functions as a service
- Staas: Storage as a service
- ....