# SIT314 – Week 8 Class Discussion

## Microservices

### What is a microservice?

A microservice is a small, independent, and self-contained unit of software that performs a specific business function within a larger application. Each microservice runs in its own process, has its own database or data storage, communicates with other microservices via lightweight protocols (e.g., HTTP REST, gRPC, messaging queues), and can be deployed, scaled, and updated independently.  
  
Example: In an e-commerce app, the 'payment', 'cart', and 'product catalog' can each be separate microservices.

### What is a microservices architecture and how does it relate to the SoC principle?

A microservices architecture is a design pattern where an application is broken down into multiple, loosely coupled services. This approach enables independent deployment, fine-grained scalability, and flexibility in choosing technologies per service.  
  
It reflects the Separation of Concerns (SoC) principle by dividing an application into focused modules and assigning each microservice responsibility for a single concern, making the system easier to maintain, test, and evolve.

### What are the similarities in the goals of microservices and IoT? And what are the challenges?

Similarities:

* - Decentralization: Both aim to distribute functionality across many small, autonomous components.  
  - Scalability: Both support horizontal scaling and dynamic environments.  
  - Resilience: Faults in one component should not crash the whole system.  
  - Interoperability: Both rely heavily on APIs and standard protocols for communication.

Challenges:

* - Complexity: Coordinating and managing many distributed services or devices.  
  - Latency & Network Reliability: Communication may be impacted by slow or unreliable networks.  
  - Security: Increased attack surfaces due to multiple endpoints.  
  - Data Consistency: Ensuring consistency across services or sensors in near real-time.

## Domain-Driven Design (DDD)

### What is Domain-Driven Design (DDD) for microservices?

Domain-Driven Design (DDD) is a software design approach focused on modeling software based on real-world business domains. For microservices, DDD helps define what each microservice should do, what data it owns, and how it interacts with other services. DDD ensures that microservices are aligned with real business processes rather than arbitrary technical separations.

### How does each DDD process work?

1. Analyse Domain: Understand the business logic, rules, and goals. Collaborate with domain experts to identify key problems and workflows.  
  
2. Define Bounded Contexts: Separate the system into clearly defined zones where specific terms and rules apply.  
  
3. Define Entities, Aggregates, and Services:  
 - Entities: Objects with a unique identity (e.g., Order, Customer).  
 - Aggregates: Groups of related entities treated as a single unit for consistency.  
 - Services: Operations that don’t fit naturally into entities.  
  
4. Identify Microservices: Define microservices based on bounded contexts and aggregates. Each service manages its own data, has clear API boundaries, and aligns with a business domain.

### How does the "Drone Delivery" use case illustrate the principles of DDD?

In a Drone Delivery System:  
  
1. Domain Analysis:  
 - Domains: Flight Scheduling, Drone Maintenance, Package Delivery, Weather Monitoring.  
  
2. Bounded Contexts:  
 - DroneFleetManagement, PackageRouting, FlightAuthorization, RealTimeTracking.  
  
3. Entities and Aggregates:  
 - Entities: Drone, Package, FlightPath.  
 - Aggregates: DeliveryMission (includes drone, flight path, and package).  
 - Services: PathOptimizerService, WeatherAlertService.  
  
4. Microservices:  
 - DroneService for hardware tracking,  
 - MissionPlannerService for route calculation,  
 - MaintenanceService for battery and service logs.  
  
DDD ensures that each microservice in the drone ecosystem maps to a specific, meaningful function aligned with the real-world logistics of drone delivery.