Submitted By,
Adesh Oak
(aoak@iu.edu)

LECTURE SUMMARY
WEEK 11

In this week, we discussed 4 out of the 7 said software architecture styles viz.

- Service-based Architecture Style
- Event Driven Architecture Style
- Space-based Architecture Style
- Microservices Architecture Style

To begin with, we must understand what **Architecture quantum** means. The formal definition of Architecture quantum defines it as,

An **independently deployable** artifact with **high functional cohesion** and **synchronous connascence**.

Now, this definition has 3 key concepts:
1. **Synchronous connascence:** It basically refers to synchronous calls within an application context. It is a phrase used to describe a situation in which two or more system components are closely coupled regarding timing and execution order. In order to ensure proper system behavior, the components must run in a specified order or at a specific time.
2. **High functional cohesion:** A situation where the components of a module or component are closely related in terms of the functionality they offer is referred to as having high functional cohesion in software architecture design. To put it another way, a module or component has high functional cohesion when all its parts work together to complete a single, clearly defined task or set of activities.
3. **Independently deployable:** A condition in which modules or components of a system can be deployed or changed independently of one another is referred to as being independently deployable in software architecture design.

Architecture Styles –

1. **Service based Architecture Style:**

   A method of building complicated systems using a group of loosely connected, independently deployable services is called as service-based architecture. Typically, each service offers a particular functionality or business capacity and interacts with other services using defined protocols like REST or SOAP.

   A service-based architecture can include a variety of **topologies**, including monolithic, layered, microservices, and serverless. In contrast to layered architecture, which separates the application into layers based on functionality, monolithic architecture packages all application logic into a single executable file. The application is divided into a number of separate, interconnected services in a microservices architecture. In a serverless architecture, serverless functions that are triggered by particular events are used to build the application.

   In a service-based architecture, the **granularity of the services** can vary from coarse to fine. Whereas fine-grained services are more specialized and narrowly focused, coarse-grained services offer a wider range of functionality.

   Database partitioning is a crucial factor to consider in service-based architecture since it affects the system's scalability and performance. A huge database is divided into smaller, more manageable portions that can be stored on many servers through a process known as database partitioning. There are various partitioning techniques, including hybrid, vertical, and horizontal partitioning.

   Compared to conventional monolithic systems, service-based architecture has more flexibility, scalability, and modularity. For complicated applications that need flexibility, scalability, and adaptability, service-based architecture makes sense. Large-scale applications, remote teams, sophisticated business logic, frequently changing applications, and high availability needs all benefit greatly from it. Before choosing a service-based architecture, it's crucial to take the application's unique requirements into account.

2. **Event Driven Architecture Style:**

   A software architectural approach known as **event-driven architecture** places a focus on the creation, detection, consumption, and response to events that take place within a system or across several systems. Instead of waiting for client requests to initiate actions, the system architecture in Event Driven Architecture is created to respond to events quickly and effectively.

   Event-based architectures are more reactive and adaptable than request-based systems, which prioritize client requests as the source of action. Compared to typical request-based systems, event-driven architecture can offer enhanced responsiveness, scalability, and

fault tolerance. To provide successful event detection, management, and handling, it involves thorough design and implementation.

Publish/subscribe, point-to-point, broker, and mediator are some of the topology variations available for event-driven architecture. In a **mediator topology**, a mediator component passes messages between producers and consumers, as opposed to a message broker acting as an intermediate in a **broker topology**.

Business processes and workflows in event-driven architecture are defined using **BPEL (Business Process Execution Language)**. It enables the definition of intricate, protracted processes involving numerous systems and services.

Scalability, fault tolerance, and modularity are three **rating** criteria that can be used to evaluate event-driven architecture. The ability of the system to handle more workloads and events as demand increases is referred to as scalability. The ability of the system to continue operating even if one or more components fail is referred to as fault tolerance. The degree to which a system may be broken down into smaller, easier-to-manage components that can be created and deployed independently is referred to as its modularity.

3. Space-based Architecture Style:

A shared memory space is prioritized in the **space-based architecture style**, a distributed computer architecture style, to store and manage data among a network of nodes. Data in Space based architecture, is managed by a messaging grid, a data grid, and data pumps and disseminated among a number of nodes.

The **messaging grid** oversees controlling inter-node communication, enabling effective message passing and event-driven processing. The **data grid** oversees maintaining and storing the data while also offering quick and scalable access to the data. **Data pumps** oversee transferring data between nodes, making sure that it is constantly accessible when and where it is required.

In Space-based architecture, **data collisions** can happen when numerous nodes attempt to access or alter the same data at the same time. Many approaches, including optimistic concurrency control and transaction management, can be used to overcome this.

Distributed or replicated caching solutions can be applied to space-based architecture. Data is partitioned and spread across several nodes during distributed caching, but during replicated caching, data is replicated across multiple nodes for improved availability and fault tolerance.

Scalability, fault tolerance, and performance are just a few of the **ratings** criteria that can be used to rank Space based architecture style. Space based architecture style can offer greater scalability, fault tolerance, and performance in comparison to conventional client-

server architectures. To ensure efficient data handling and management, it does, however, require proper design and implementation.

4. Microservices Architecture Style:

A software development strategy called **microservice architecture** divides systems into several tiny, independently deployable services. Each service focuses on a single business capability, and they all interact with one another via clearly defined APIs.

The limits of monolithic architectures, which became more challenging to maintain and expand as applications grew in complexity, gave rise to microservice architecture in the early 2010s. **Decoupling services**, which allows for autonomous service development, deployment, and scaling without affecting other services, is a key component of microservice design. Keeping services compact and concentrated on a single business feature, eliminating shared state between services, and utilizing asynchronous communication to lessen coupling are all examples of **guidelines for granularity** in microservice architecture. Another crucial component of microservice architecture is **operational reuse**. It places a focus on the development of reusable building pieces that may be utilized across various applications, including authentication or logging services.

The advantages of microservice architecture include increased scalability, maintainability, and agility. Applications can be divided into smaller, easier-to-manage services, making it simpler to introduce new features, address issues, and scale the system to handle fluctuations in demand. Microservice architecture faces several difficulties, including a rise in complexity, the necessity for efficient service discovery and management, and the cost of overseeing numerous services.

In recent years, cloud-native apps and DevOps approaches have given the microservice architecture a boost in popularity. It offers an adaptable and scalable method for developing software that enables businesses to create and implement sophisticated applications more quickly and effectively.

***In the second lecture, we focused on in-class coding and made a database for our application and connected the application to the database. ***