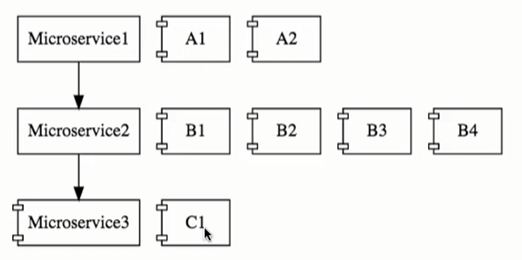
Microservices Notes

Q1. Microservice, Microservice Architecture definition

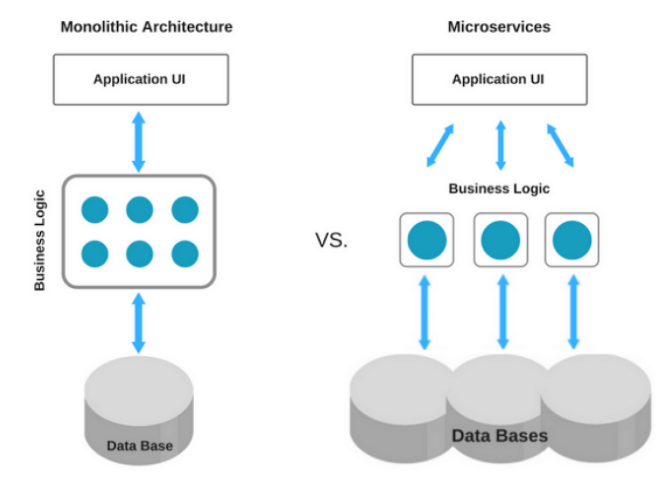
* Microservice Architecture is an approach to develop a single application as a suite of small autonomous services called microservices, each running its own process, loosely coupled & communicate with each other over standard protocols.
* These services are built around business capabilities & independently deployable by fully automated deployment machinery.
* Microservices are RESTful web services, well-chosen small deployable units & cloud enabled. Cloud enabled means if there is more load on one microservice, then we can easily bring up another instance of that microservice. Same if there is less load one microservice.



* The communication b/w microservices is a stateless communication where each pair of request & response is an independent transaction.
* In MSA, data is federated i.e., each microservice is responsible for its own data model & data.

Q2. Monolithic Architecture

* Monolithic Architecture is an approach to develop the entire application as a single piece that is designed to be self – contained i.e., all the components of the application are interconnected & tightly coupled i.e., each component along with its associated components must be present in order to execute the functionality.



|  |  |
| --- | --- |
| Advantages of Monolith  1. Very simple to develop & one directional.  2. Very simple to deploy  3. Very simple to scale | Challenges with Monolith Architecture  1. Large & complex application  2. Slow development & deployment  3. Blocks continuous development  4. Unscalable in terms of each component, Unreliable because of tightly coupled components, Inflexible. |

Q3. Monolithic Vs Microservice Architecture

|  |  |  |
| --- | --- | --- |
| **No.** | **Monolithic Architecture** | **Microservice Architecture** |
| 1. | Monolithic Architecture is like a big container where all the components of an application are assembled together. | Microservice Architecture is composed of small autonomous services where each service delivers a specific business goal. |
| 2. | Components are tightly coupled. | Components/Services are loosely coupled. |
| 3. | Fault tolerance is difficult i.e., If any specific feature is not working, the complete system goes down. | Fault tolerance is easy i.e., if even one service goes down, others can continue to work. |
| 4. | Service startup takes more time. | Service startup is relatively quick. |
| 5. | Application scaling is challenging & wasteful  i.e., since service is not isolated, individual resource allocation not possible. | Individual resource allocation is possible i.e., More H/w resources can be allocated to the service that is frequently used. |
| 6. | Low availability | High availability |
| 7. | Data is centralized. | Data is federated. This allows individual microservice to adopt a data model best suited for its needs. |
|  | Change in data model affects the entire DB. | Change in data model of one microservice doesn’t affect other microservice. |
| 8. | Large team & considerable team management effort is required. | Parallel & faster development small focused team. |
| 9. | NA | Interacts with other microservices by using well – defined interface. |
| 10. | Put emphasis on the entire project. | Microservice works on principle that focus on products not project. |

Q4. Principle being Microservices

1. Independent & Autonomous services
2. Resilient services (Fault isolation)
3. Real time load balancing
4. Availability
5. Seamless API integration & continuous monitoring
6. Auto provisioning
7. Scalability
8. Decentralization
9. Continuous delivery through Devops integration

Q5. Challenges with building microservices

1. Bounded context: how to identify boundary for each microservices.
2. Configuration management
3. Dynamic scale up & down
4. Visibility & monitoring
5. Fault tolerance

* To the typical problems which are present for distributed systems in the cloud, Spring Cloud provides a range of solutions.
* Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems. (e.g., Configuration management, service discovery, circuit breakers, intelligent routing, micro proxy, control bus, one time token, global locks, leadership election, distributed sessions, cluster state)

Q6. What is design Pattern & why do we need design patterns

* Software design pattern is defined as software template or a description to solve a problem that occurs in multiple instances while designing a software application or a software framework.
* To ensure that all the teams follow the same process or may be same pattern we can use the concept of design pattern.
* By using the design patterns, the team working on various projects use the same pattern to build the similar application.

Q7. Microservice Design Pattern

1. Aggregator Design Pattern (based on DRY principle)

2. API Gateway Design Pattern

3. Chained or Chain of Responsibility

4. Asynchronous messaging Design Pattern

5. Database or Shared DB design Pattern

6. Event Sourcing Design Pattern

7. Command Query Responsibility Segregator (CQRS)

8. Circuit breaker design pattern

9. Branch Design pattern

10. Decomposition design pattern