Microservices

**Question: What do you understand by Microservices?**

**Answer:** Microservices or more appropriately Microservices Architecture is an SDLC approach based on which large applications are built as a collection of small functional modules. These functional modules are independently deployable, scalable, target specific business goals, and communicate with each other over standard protocols. Such modules can also be implemented using different programming languages, have their databases, and deployed on different software environments. Each module here is minimal and complete.

**Advantages**:

* Improved Scalability
* Fault Isolation
* Localized Complexity
* Increased Agility
* Simplified Debugging & Maintenance
* Better correspondence of developers with business users.
* Smaller development teams
* Better scope for technology upgradation.

**Disadvantages:**

* Complicated as a whole.
* Requires accurate pre-planning
* Modular dependencies are hard to calculate.
* Less control over third party applications
* Modular Interdependencies are challenging to track.
* More opportunities for malicious intrusions.
* Complete end-to-end testing is difficult.
* Deployment Challenges.

#### ****Question: What are the different strategies of Microservices Deployment?****

**Answer:**

* **Multiple Service Instance per Host:** Run single or multiple service instances of the application on single/multiple physical/virtual hosts.
* **Service Instance per Host:** Run a service instance per host.
* **Service Instance per Container:** Run each service instance in its respective container.
* **Serverless Deployment:** Package the service as a ZIP file and upload it to the Lambda function. The Lambda function is a stateless service that automatically runs enough micro-services to handle all requests.

**Monolithic vs. SOA vs. Microservices Architecture**

* **Monolithic Architecture:**
  + In **Monolithic** Architecture, all software components of the application are assembled and packed tightly.
  + In this type of architecture, different components of an application like UI, business logic, data access layer are combined into a single platform or program.
* **SOA (Service Oriented Architecture):**
  + **SOA** [Service Oriented Architecture] is a collection of services that communicate with each other through simple data passing or activity coordination.
  + In this architecture, individual components are loosely coupled and perform a discrete function. There are two main roles – service provider and service consumer. In SOA type, modules can be integrated and reused, making it flexible and reliable.
* **Microservices Architecture:**
  + **Microservices** Architecture is a collection of small functional modules. These functional modules are independently deployable, scalable, target specific business goals, and communicate with each other over standard protocols.
  + It is a type of SOA in which a series of autonomous components are built and combined to make an app. These components are integrated using APIs. This approach focuses on business priorities and capabilities and offers high agility, i.e. each component of the app can be built independently of the other.

**Cohesion and Coupling**

The strength of dependencies between services in a microservice architecture is said to be coupling. Cohesion refers to the related logic between two or more services. The entire concept of microservices is based on the ability to deploy and update service while keeping other services intact. Hence, loose coupling and high cohesion is the key to a microservice design.

#### ****Difference between Cohesion and Coupling?****

**Coupling:**It is the relationship between module A and another module B. Any module can be highly coupled (highly dependent), loosely coupled and uncoupled with other modules. The best coupling is loose coupling achieved through interfaces.

**Cohesion:** It is the relationship between 2 or more parts within a module. If a module has high cohesion, it means the module can perform a certain task with utmost efficiency on its own, without communication with other modules. High cohesion enhances the functional strength of a module.

