**Name – Nabanita Sarkar**

**Emp ID – 815944**

**Microservices Architecture Notes:**

Microservice Architecture is a special design pattern of Service-oriented Architecture. It is an open source methodology. In this type of service architecture, all the processes will communicate with each other with the smallest granularity to implement a big system or service. Microservice is a service-based application development methodology. In this methodology, big applications will be divided into smallest independent service units. Microservice is the process of implementing Service-oriented Architecture (SOA) by dividing the entire application as a collection of interconnected services, where each service will serve only one business need.

**Advantages of Microservices:**

* Microservices work well with agile development processes and satisfy the increasing need for a more fluid flow of information.
* Microservices are independently deployable and allow for more team autonomy
* Each microservice can be deployed independently, as needed, enabling continuous improvement and faster app updates.
* Specific microservices can be assigned to specific development teams, which allows them to focus solely on one service or feature. This means teams can work autonomously without worrying what’s going on with the rest of the app.
* Microservices are independently scalable.
* As demand for an app increases, it’s easier to scale using microservices. You can increase resources to the most needed microservices rather than scaling an entire app. This also means scaling is faster and often more cost-efficient as well.
* Microservices reduce downtime through fault isolation.
* If a specific microservice fails, you can isolate that failure to that single service and prevent cascading failures that would cause the app to crash. This fault isolation means that your critical application can stay up and running even when one of its modules fails.

**Testing Microservices:**

**Unit Testing**

An often overlooked practice when testing microservices is unit testing. [What are unit tests](https://www.parasoft.com/solutions/unit-testing/)? These tests verify that the methods and classes developers write work as expected. While unit testing is a highly technical task for developers, a robust suite of unit tests provides a critical safety net for catching unintended consequences when developers change the code. And it pays dividends in alerting developers to exactly where in the code they’ve broken existing functionality.

This is a valuable practice for writing high-quality software. However, unit testing by itself isn’t enough. As an analogy, just because all the parts of an engine are machined to perfect specification doesn’t mean the engine will run and perform as expected.

**Component Testing**

This microservices testing does not concentrate on how the developer wrote the microservices code but instead focuses on running the microservice as a black box and testing the traffic moving over the interface. From the perspective of a single microservice, you’re now testing the engine to ensure it’s delivering on its requirements.

In most cases, you’re testing a REST service. So you want automated tests that act as clients of the service, sending various positive and negative requests to the service and verifying the responses that the service returns.

**Integration Testing**

When using service virtualization to simplify and stabilize testing the microservice as an individual component, you also want to test that the microservice works with the other REAL microservices involved. Developers often do this at a “QA” or “integration” stage, where many of the required systems in the overall ecosystem are deployed and integrated together. With this testing practice, you’re beginning to assemble the car to make sure every part fits and works together but you’re not testing it on the road yet.

**End-To-End Testing**

Also called system testing. At some point, a big web of microservices has entry points where the application’s end users interact. For example, a Netflix app on your Apple TV talks to microservices within Netflix’s data center. But they represent only a small portion of their core functionality, which are small, individual components responsible for specific things like a recommendations service, a video streaming service, an account details service, and so on. So this, too, is an opportunity for testing microservices.

**Monolith Architecture:**

A monolithic architecture is the traditional unified model for the design of a software program. Monolithic, in this context, means composed all in one piece. According to the Cambridge dictionary, the [adjective monolithic](https://dictionary.cambridge.org/us/dictionary/english/monolithic) also means both too large and unable to be changed.

Monolithic software is designed to be self-contained; components of the program are interconnected and interdependent rather than [loosely coupled](https://www.techtarget.com/searchnetworking/definition/loose-coupling) as is the case with modular software programs. In a tightly-coupled architecture, each component and its associated components must be present in order for code to be executed or compiled.