**1. What is Microservices Architecture?**

Microservices architecture allows avoiding monolith application for the large system. It provides loose coupling between collaborating processes which running independently in different environments with tight cohesion.

* ***Loose Coupling***– Effect of changes isolated
* ***Tight Cohesion***– Code perform a single well-defined task
* ***Loose Coupling***– Application build from collaboration services or processes, so any process change without affecting another process.
* ***Tight Cohesion***-An individual service or process that deals with a single view of data.

## **Microservices Benefits**

* The smaller code base is easy to maintain.
* Easy to scale as an individual component.
* Technology diversity i.e. we can mix libraries, databases, frameworks etc.
* Fault isolation i.e. a process failure should not bring the whole system down.
* Better support for smaller and parallel team.
* Independent deployment
* Deployment time reduce

## **Principles of Microservices**

There are the following principles of Microservices:

* Single Responsibility principle
* Modelled around business domain
* Isolate Failure
* Infrastructure automation
* Deploy independently

## **Disadvantages of Microservices**

* Microservices has all the associated complexities of the distributed system.
* There is a higher chance of failure during communication between different services.
* Difficult to manage a large number of services.
* The developer needs to solve the problem, such as network latency and load balancing.
* Complex testing over a distributed environment.

[How to handle microservice Interaction when one of the microservice is down](https://stackoverflow.com/questions/50562495/how-to-handle-microservice-interaction-when-one-of-the-microservice-is-down)

if we have 3 microservices M1,M2,M3 . M1 is interacting with M2 and M2 is interacting with M3. In case M2 microservice cluster is down how should we handle this situation?

When any one of the microservice is down, Interaction between services becomes very critical as **isolation of failure, resilience and fault tolerance** are some of key characteristics for any microservice based architecture.

* **Importance of Circuit breaker and Fallback Mechanism:**

Hystrix implements the [circuit breaker pattern](https://martinfowler.com/bliki/CircuitBreaker.html) which is useful when a service failure can cause cascading failure all the way up to the user. When calls to a particular service exceed **circuitBreaker.requestVolumeThreshold** (default: 20 requests) and the failure percentage is greater than **circuitBreaker.errorThresholdPercentage** (default: >50%) in a rolling window defined by **metrics.rollingStats.timeInMilliseconds** (default: 10 seconds), the circuit opens and further calls are not made.

In cases of error and an open circuit, a fallback can be provided by the developer. Fallbacks may be chained so that the first fallback makes some other business call. check out [Fallback Implementation of Hystrix](https://github.com/Netflix/Hystrix/wiki/How-To-Use#Fallback)

As mentioned in the comment, there are many ways you can go about it,

Case 1: all are independent services, trivial case, no need to do anything, call all the services in blocking or non-blocking way, calling service 2 will in both case result in timeout

Case 2: services are dependent M2 depends on M1 and M3 depends on M2

option a) M1 can wait for service M2 to come back up, doing periodic pings or fetching details from registry or naming server if M2 is up or not

option b) use hystrix as a circuit breaker implementation and handle fallback gracefully in M3 or your orchestrator(guy who is calling these services i.e M1,M2,M3 in order)