Resilience: The Ability to Recover from Failures Without Downtime

Definition: If a system can recover from failures without downtime, it is considered **resilient**.

Types of Failures in Systems:

1) **Automatically Recoverable:**
   - Some systems can recover from failures automatically without human intervention.
2) **Graceful Recovery:**
   - Some systems recover gracefully, maintaining certain functionalities.
3) **Complete Failure:**
   - Some systems cannot recover from failures and will crash, causing full unavailability.

Resilient System

A **resilient system** ensures that if some part of the system fails, **other parts continue to function** instead of causing a complete crash or making the system entirely unavailable.

A **resilient system** should have mechanisms to:

1) **Detect** failures
2) **Handle** failures
3) **Recover** from failures

How Can a System Detect Failures?

1) **Health Checks and Monitoring**
   - **Monitoring Tools:**
     - Grafana (visualization)
     - Prometheus (metrics collection)
     - AWS CloudWatch, etc.
   - **Health Checks:**
     - Spring Boot Actuator (for exposing system health endpoints)
2) **Circuit Breaker Pattern**
   - Temporarily **stops sending requests** to a failing service.
   - Prevents cascading failures and reduces system load.
3) **Retries**
   - Automatically **retry failed requests**.
   - To avoid excessive load, retries should use **exponential backoff** (increase wait time between retries instead of retrying immediately).

How Can a System Handle Failures?

1) **Circuit Breaker Pattern**
   - Detects failures.
   - Stops routing requests to the failing service.
   - Prevents overwhelming the failing service with retries.
2) **Retry Mechanism**
   - Ensures **continuity of functionality** by reattempting failed operations.
   - Reduces **data loss** in case of temporary failures.
3) **Periodic Backups / Snapshots**
   - Ensures **quick recovery** in case of data loss.
   - Reduces the risk of **losing important data**.
4) **More Instances (Redundancy & Scaling)**
   - Prevents **Single Point of Failure (SPOF)** by running multiple instances of a service.
5) **Event Sourcing**
   - Stores events **instead of final state**, allowing the system to rebuild state later if needed.

6) **Idempotency**
   - Ensures that **retrying a request does not cause duplicate actions.**

How Can a System Recover from Failures?

1) **Backups / Snapshots**
   - Restore lost data and ensure quick system recovery.
2) **Event Sourcing**
   - Replay stored events to reconstruct system state after a failure.
3) **Retry Failed Requests (while avoiding sending requests to failed systems)**
   - Automatically retry failed operations.