**MSA**

Microservice Architecture (MSA) is an architectural style that structures an application as a collection of loosely coupled services, which implement business capabilities. This approach enables the rapid, frequent, and reliable delivery of large, complex applications. It also allows an organization to evolve its technology stack.

**Components of Microservice Architecture**

* **Microservices:** Individual Services: Each microservice is a small, autonomous service that focuses on a single business capability. They can be developed, deployed, and scaled independently.
* **API Gateway:**
  + **Single Entry Point:** Acts as the single entry point for all client requests to the backend services. It routes requests to the appropriate microservice and aggregates the results to provide seamless client experiences.
* **Backend for Frontend (BFF):** Customized backend services tailored for each type of client (mobile, web, etc.).
* **Service Discovery:** Registry and Discovery: Services register themselves with a service discovery tool upon startup. This tool allows services to find and communicate with each other by looking up their addresses in a dynamic environment.
* **Configuration Management:** External Configuration Store: Externalizes configuration from the service code so that the application’s configuration can be changed without the need to redeploy the service.
* **Communication Protocol:** 
  + **Synchronous (REST, gRPC):** Services communicate with each other using synchronous protocols like HTTP/REST or gRPC for immediate requests and responses.
  + **Asynchronous (Event-Driven):** Services communicate by sending events, which other services listen for, allowing for loose coupling and scalability.
* **Database per Service:** Decentralized Data Management: Each microservice manages its own database, either different instances of the same database technology or entirely different database systems.
* **Load Balancer:**
  + **Distribution of Requests:** Distributes incoming requests evenly across instances of microservices to ensure no single instance becomes overwhelmed.
* **Circuit Breaker:**
  + **Failure Protection:** Temporarily halts operations to a particular service when failures reach a certain threshold, preventing the failure from cascading to other services.
* **Distributed Tracing and Logging:** Monitoring and Troubleshooting: Helps in tracking the flow of requests and messages across microservices, making it easier to monitor and troubleshoot issues.
* **Security:**
  + **Authentication and Authorization**: Ensures that only authenticated users can access the services and only have access to the resources they are authorized for.
* **Containerization and Orchestration:**
  + **Deployment and Scaling:** Using containerization (e.g., Docker) and orchestration tools (e.g., Kubernetes) helps in deploying, managing, and scaling microservices.
* **CI/CD Pipeline:**
  + **Continuous Integration/Delivery:** Automates the process of integrating code changes and delivering or deploying these changes quickly and reliably.

***Why change the way we build applications?***

* Customers want faster delivery
* Performance and scalability
* Performance and scalability
* Build Applications which are:
  + Flexible
  + Resilient
  + Scalable

***Resiliency Patterns***

* Client-side load balancing
* Circuit breakers pattern
* Fallback pattern
* Bulkhead pattern

***Security patterns***

* Authentication
* Authorization
* Credential management and propagation

***Logging and Tracing patterns***

* Log correlation
* Log aggregation
* Microservice tracing

***Microservice build/deployment patterns***

* **Blue-Green Deployment:** Switching between two versions of the application to reduce downtime and risk.
* **Canary Deployment:** Gradually rolling out a change to a subset of users before making it available to everyone.
* Phoenix servers

