Microservices Architecture সম্পর্কিত গুরুত্বপূর্ণ Technologies গুলোকে বাংলায় ব্যাখ্যা করছি, যাতে আপনি সহজে বুঝতে পারেন কোন কোন টেকনোলজি ব্যবহার হয় এবং কেন হয়।

**🔹 ১. API Gateway**

**👉 বাংলা ব্যাখ্যা:**

Microservices গুলোর সামনে একটি API Gateway রাখা হয়, যেটা সব রিকোয়েস্ট প্রথমে গ্রহণ করে এবং উপযুক্ত microservice এর কাছে পাঠিয়ে দেয়।

**✅ সুবিধা:**

* ক্লায়েন্টের সাথে সরাসরি সব সার্ভিসের সংযোগের দরকার হয় না।
* Security, Rate Limiting, Load Balancing এর সুবিধা দেয়।

**📌 উদাহরণ:**

* **Ocelot** (.NET এ)
* **Kong**
* **Nginx**
* **AWS API Gateway**

**🔹 ২. Service Discovery**

**👉 বাংলা ব্যাখ্যা:**

যেহেতু Microservice গুলো ডাইনামিক (auto scale / restart হতে পারে), তাই কোন সার্ভিস কোথায় আছে সেটার Location খুঁজে বের করার জন্য Service Discovery ব্যবহার হয়।

**✅ সুবিধা:**

* সার্ভিস গুলো একে অপরকে খুঁজে পেতে পারে।
* Load Balancing এ সহায়তা করে।

**📌 উদাহরণ:**

* **Consul**
* **Eureka** (Spring Cloud)
* **Zookeeper**

**🔹 ৩. Containerization**

**👉 বাংলা ব্যাখ্যা:**

Microservice গুলোকে ছোট ছোট আলাদা আলাদা কনটেইনারে রাখা হয়, যাতে সেগুলো স্বতন্ত্রভাবে রান করতে পারে।

**✅ সুবিধা:**

* Isolation, Scalability, এবং Deployment সহজ হয়।

**📌 উদাহরণ:**

* **Docker**
* **Podman**

**🔹 ৪. Container Orchestration**

**👉 বাংলা ব্যাখ্যা:**

অনেকগুলো container একসাথে চললে সেগুলোকে পরিচালনা করার জন্য Orchestrator ব্যবহার হয়।

**✅ সুবিধা:**

* Auto-scaling, Load Balancing, Rollback, Monitoring ইত্যাদি সহজ হয়।

**📌 উদাহরণ:**

* **Kubernetes (K8s)**
* **Docker Swarm**
* **AWS ECS**

**🔹 ৫. CI/CD (Continuous Integration / Continuous Deployment)**

**👉 বাংলা ব্যাখ্যা:**

Microservices architecture এ, প্রতি সার্ভিস আলাদা আলাদা ডেভেলপ/ডিপ্লয় হয়, তাই CI/CD অটোমেশন খুব গুরুত্বপূর্ণ।

**✅ সুবিধা:**

* প্রতিবার কোড চেঞ্জ হলে সেটা অটো বিল্ড ও ডিপ্লয় হয়।

**📌 উদাহরণ:**

* **GitHub Actions**
* **Jenkins**
* **GitLab CI**
* **Azure DevOps**

**🔹 ৬. Communication Protocols**

**👉 বাংলা ব্যাখ্যা:**

Microservice গুলোর মধ্যে যোগাযোগের জন্য কিছু প্রোটোকল বা পদ্ধতি ব্যবহৃত হয়।

**✅ উদাহরণ:**

* **HTTP REST API**
* **gRPC** (binary protocol, fast)
* **Message Queue** (asynchronous communication)

**🔹 ৭. Message Broker / Event Bus**

**👉 বাংলা ব্যাখ্যা:**

সার্ভিসগুলো asynchronous ভাবে একে অপরকে মেসেজ পাঠাতে চাইলে Message Broker ব্যবহৃত হয়।

**✅ সুবিধা:**

* সার্ভিস ডিকাপলড থাকে।
* ইভেন্ট-ভিত্তিক আর্কিটেকচার সম্ভব হয়।

**📌 উদাহরণ:**

* **RabbitMQ**
* **Apache Kafka**
* **Azure Service Bus**
* **AWS SQS**

**🔹 ৮. Distributed Logging & Tracing**

**👉 বাংলা ব্যাখ্যা:**

Microservice গুলো বিভিন্ন সার্ভারে রান করতে পারে, তাই পুরো সিস্টেমের জন্য central লগিং দরকার হয়।

**✅ সুবিধা:**

* Debugging সহজ হয়।
* Performance বোঝা যায়।

**📌 উদাহরণ:**

* **ELK Stack** (Elasticsearch, Logstash, Kibana)
* **Grafana + Loki**
* **Jaeger** (Tracing)
* **Zipkin** (Tracing)

**🔹 ৯. Database per service**

**👉 বাংলা ব্যাখ্যা:**

Microservices এ প্রতিটি সার্ভিস নিজের Database রাখে, যাতে সার্ভিসগুলো একে অপরের উপর নির্ভরশীল না হয়।

**✅ সুবিধা:**

* Loose Coupling নিশ্চিত হয়।
* Scalability & Security বজায় থাকে।

**🔹 ১০. Security (Authentication & Authorization)**

**👉 বাংলা ব্যাখ্যা:**

Multiple microservices access করার সময় Central Auth System দরকার হয়।

**📌 উদাহরণ:**

* **OAuth2 / OpenID Connect**
* **IdentityServer**
* **Keycloak**
* **JWT (JSON Web Token)**

**🔹 ১১. Configuration Management**

**👉 বাংলা ব্যাখ্যা:**

সার্ভিসের কনফিগারেশনগুলো আলাদা করে central ভাবে রাখা হয় যাতে প্রোডাকশনে পরিবর্তন করা সহজ হয়।

**📌 উদাহরণ:**

* **Spring Cloud Config**
* **Azure App Configuration**
* **HashiCorp Vault (with secrets)**

**চাইলে আমি প্রতিটি টেকনোলজির ওপর আলাদা করে বিস্তারিত বাংলা টিউটোরিয়াল বা ডেমো প্রজেক্ট তৈরি করে দিতেও পারি।**

আপনি কি চান শুরুটা কোথা থেকে হোক? যেমন:

* API Gateway?
* Docker & Kubernetes?
* Message Queue?
* Identity Server?

চলুন এবার **Microservices Architecture**-এর প্রতিটি গুরুত্বপূর্ণ Technology-র উপর আরও বিস্তারিতভাবে বাংলায় আলোচনা করি।

**🔹 ১. API Gateway – এক্সিট ও এন্ট্রি গেট**

**🧠 কি করে?**

API Gateway এমন একটা গেটওয়ে, যেটা ক্লায়েন্ট (মোবাইল/ফ্রন্টএন্ড অ্যাপ) থেকে আসা রিকোয়েস্টগুলো গ্রহণ করে এবং সেই অনুযায়ী সেগুলো পেছনের উপযুক্ত Microservice-এ পাঠিয়ে দেয়।

**🧩 ফিচার:**

* Request Routing (যে রিকোয়েস্ট যে সার্ভিসের জন্য, সেখানে পাঠায়)
* Authentication & Authorization (একটি সেন্ট্রাল জায়গা থেকে ইউজার যাচাই)
* Load Balancing (রিকোয়েস্টগুলো সমভাবে ভাগ করে)
* Caching
* Rate Limiting
* Logging

**📦 বাস্তব উদাহরণ:**

1. ইউজার লগইন করতে চাইলে api.domain.com/login—এ API Gateway রিকোয়েস্ট নিয়ে **Auth Service**-এ পাঠায়।
2. অর্ডার তৈরি করতে চাইলে api.domain.com/order—এ রিকোয়েস্ট গেলে **Order Service**-এ পাঠায়।

**🔹 ২. Service Discovery – কোথায় কোন সার্ভিস আছে?**

**🧠 কেন দরকার?**

Microservices গুলো সাধারণত dynamic port বা IP তে রান করে (Docker / Kubernetes), তাই ঠিক কোন IP তে কোন সার্ভিস চলছে সেটা জানার জন্য সার্ভিস ডিসকভারি দরকার।

**🧩 কিভাবে কাজ করে?**

* প্রতিটি সার্ভিস নিজে রেজিস্টার করে ডিসকভারি সার্ভারে (যেমন Eureka, Consul)
* অন্য সার্ভিস তখন Discovery Service থেকে খুঁজে বের করে কার IP কী, কোথায় কল করতে হবে

**🔹 ৩. Docker – মাইক্রোসার্ভিস containment**

**🧠 কাজ:**

Docker একটি virtualization tool, যেটা OS-level isolation দেয়। প্রতিটি Microservice আলাদা Docker container-এ রান হয়।

**✅ সুবিধা:**

* একই সার্ভিস সব environment-এ একইভাবে চলে
* Deployment ও Scaling সহজ
* Dependency Conflict এড়ানো যায়

**🎯 উদাহরণ:**

AuthService → authservice:latest নামে Docker Image  
OrderService → orderservice:latest

**🔹 ৪. Kubernetes – অনেক সার্ভিসের পরিচালনাকারী**

**🧠 কি করে?**

Docker container গুলোকে orchestrate করে অর্থাৎ manage করে:

**🧩 ফিচার:**

* Auto-scaling
* Load balancing
* Health check & Restart
* Deployment strategy (Rolling Update, Canary)

**🔹 ৫. CI/CD – Build এবং Deploy অটোমেটিক**

**🧠 কেন দরকার?**

Microservices architecture এ প্রতিটি সার্ভিস আলাদা আলাদা থাকে, তাই প্রত্যেকটার জন্য স্বতন্ত্র Build & Deployment pipeline দরকার।

**🛠️ টুলস:**

* GitHub Actions → Push করলেই Build & Deploy হবে
* Jenkins → Code Merge হলেই নতুন Docker image build করবে

**🔹 ৬. Communication: REST API vs Message Queue**

**🧠 REST:**

* Sync Communication (অপেক্ষা করে রেসপন্সের জন্য)
* সাধারন HTTP Method: GET, POST, PUT, DELETE

**🧠 Message Queue:**

* Async Communication (মেসেজ পাঠিয়ে দেয়, রেসপন্সের জন্য অপেক্ষা করে না)
* Decoupled architecture

**🔔 উদাহরণ:**

* OrderService মেসেজ পাঠায় InventoryService কে RabbitMQ দিয়ে: "এই পণ্যের স্টক কমাও"

**🔹 ৭. RabbitMQ / Kafka – বার্তাবাহক**

**🧠 কিভাবে কাজ করে?**

1. **Producer**: যেটা মেসেজ পাঠায়
2. **Queue/Topic**: যেটা মেসেজ ধরে রাখে
3. **Consumer**: যেটা মেসেজ গ্রহণ করে এবং প্রসেস করে

**🧩 বাস্তব ব্যবহার:**

* ইভেন্ট-ভিত্তিক Communication (Event-Driven Architecture)
* অর্ডার প্লেস করলে, Payment, Inventory, Email Notification আলাদা আলাদা queue তে যায়

**🔹 ৮. Centralized Logging – সমস্ত সার্ভিসের লগ একত্রে দেখা**

**🧠 সমস্যা:**

যেহেতু সার্ভিসগুলো বিভিন্ন জায়গায় ছড়িয়ে থাকে, তাই একজায়গায় সব লগ দেখতে চাই।

**🧩 সমাধান:**

* **Log Aggregator**: যেমন Logstash, Fluentd
* **Storage & Search**: Elasticsearch
* **Visualization**: Kibana, Grafana

**🔹 ৯. Distributed Tracing – কে কত সময় নিল বুঝি কিভাবে?**

**🧠 সমস্যা:**

একটা API কল একাধিক সার্ভিসের মধ্য দিয়ে যায়। কোন সার্ভিসে দেরি হচ্ছে সেটা বোঝা কঠিন।

**🧩 সমাধান:**

* **Jaeger / Zipkin** → Request path trace করে
* দেখায় কোন সার্ভিসে কতটা সময় লাগলো

**🔹 ১০. Authentication & Authorization – ইউজার যাচাই করা**

**🧠 কিভাবে?**

1. Central Auth Service থাকে (OAuth2 / OpenID Connect)
2. User login করলে JWT Token পাওয়া যায়
3. অন্য সার্ভিসগুলো সেই টোকেন verify করে

**📦 টুলস:**

* IdentityServer
* Keycloak
* Auth0
* JWT

**🔹 ১১. Configuration Server**

**🧠 সমস্যাঃ**

প্রতিটি সার্ভিসে .env বা appsettings.json আলাদা রাখতে হয়।

**✅ সমাধান:**

Centralized Config Server ব্যবহার করে সব সার্ভিস একসাথে কনফিগ পড়ে।

**📦 উদাহরণ:**

* Spring Cloud Config
* Azure App Configuration
* HashiCorp Vault (secret management সহ)

**🔹 ১২. Database per Microservice – আলাদা ডেটাবেস**

**🧠 কারণ:**

Microservice গুলো একে অপরের উপর নির্ভরশীল না থাকুক।

**🧩 Design:**

* AuthService → Users DB
* OrderService → Orders DB
* InventoryService → Inventory DB

➡️ প্রয়োজনে Data Sync করতে পারে Event বা API মাধ্যমে।

**🔚 উপসংহার:**

Microservices Architecture মানে শুধু ছোট ছোট সার্ভিস বানানো নয়, বরং তার চারপাশে থাকা বিশাল Ecosystem টাও ভালোভাবে বুঝে নেওয়া।

চলুন এবার **Microservices Architecture-এর Advanced Level Concepts এবং Technologies** বাংলায় আরও গভীরভাবে ব্যাখ্যা করি। এই অংশে আমরা এমন বিষয়গুলো কভার করব যেগুলো **Lead Software Engineer**, **Architect**, বা **DevOps Engineer** হিসেবে জানা অত্যন্ত গুরুত্বপূর্ণ।

**🔥 অ্যাডভান্সড Microservices Concepts (বাংলায় বিস্তারিত)**

**🔹 ১. Circuit Breaker Pattern – সার্ভিস ফেল করলে কী হবে?**

**🔍 সমস্যা:**

Microservice A → B → C কল করছে। যদি C down হয়ে যায়, তাহলে পুরো সিস্টেমই ধ্বসে পড়ে।

**✅ সমাধান:**

Circuit Breaker প্যাটার্ন ব্যবহার করে যদি কোনো সার্ভিস বারবার ফেল করে, তখন সাময়িকভাবে সেটাকে ডাকা বন্ধ করে দেয় এবং fallback রেসপন্স দিয়ে দেয়।

**📦 টুলস:**

* **Polly** (C#)
* **Resilience4j** (Java)
* **Istio (Service Mesh)**

**🧠 বাংলা উদাহরণ:**

AuthService → UserService → EmailService

EmailService যদি down হয়, তবে fallback দেয়: “ইমেইল সার্ভিস এখন বন্ধ আছে, পরে আবার চেষ্টা করুন।”

**🔹 ২. Service Mesh (Istio, Linkerd) – মাইক্রোসার্ভিসের ট্র্যাফিক কন্ট্রোল**

**📌 কেন দরকার?**

Microservices গুলোর মধ্যে communication, security, observability, retries ইত্যাদি automate করতে।

**✅ সুবিধা:**

* Traffic Routing
* mTLS Security (সার্ভিস টু সার্ভিস encryption)
* Circuit Breaking
* Canary Deployments

**📦 জনপ্রিয় Tools:**

* **Istio (Kubernetes জন্য)**
* **Linkerd**

**🧠 বাস্তব চিত্র:**

আপনার কোডে কোনো পরিবর্তন ছাড়াই Istio sidecar proxy ব্যবহার করে সার্ভিসের ট্র্যাফিক নিয়ন্ত্রণ করা যায়।

**🔹 ৩. Event-Driven Architecture (EDA)**

**📌 বর্ণনা:**

Microservice গুলো ইভেন্ট তৈরি করে এবং অন্য সার্ভিসগুলো সেই ইভেন্টে রেসপন্ড করে।

**✅ সুবিধা:**

* Loose Coupling
* Scalability
* Better async integration

**🎯 উদাহরণ:**

OrderService → “OrderPlaced” ইভেন্ট প্রকাশ করে  
PaymentService → “OrderPlaced” শুনে পেমেন্ট প্রসেস শুরু করে  
InventoryService → “OrderPlaced” শুনে স্টক কমায়

**📦 টুলস:**

* Kafka (High throughput, distributed log)
* RabbitMQ
* NATS
* Azure EventGrid

**🔹 ৪. Saga Pattern – Distributed Transactions**

**📌 সমস্যা:**

Microservices এ একাধিক সার্ভিসে একসাথে transaction করা যায় না (ACID fails)।

**✅ সমাধান:**

Saga pattern - এ প্রতিটি সার্ভিস নিজের কাজ শেষ করে, তারপর পরের সার্ভিস কাজ করে। ভুল হলে compensation action নেয়।

**🎯 উদাহরণ:**

1. OrderService → Order Created
2. PaymentService → Payment Done
3. InventoryService → Stock Deducted
4. যদি কোনো ধাপে ভুল হয়, আগের সার্ভিসগুলো rollback (e.g. stock ফেরত দেওয়া, পেমেন্ট রিফান্ড)

**🔹 ৫. Canary Deployment / Blue-Green Deployment**

**📌 উদ্দেশ্য:**

প্রোডাকশনে নতুন version ধাপে ধাপে ছাড়ার জন্য, যাতে সব ইউজার না দেখে, আগে একটু দেখা যায়।

**✅ সুবিধা:**

* Error কম হয়
* Rollback সহজ

**🎯 উদাহরণ:**

* Blue-Green: পুরনো version (blue) ও নতুন version (green) একই সাথে চলে; ইউজারকে ধীরে ধীরে switch করা হয়
* Canary: মাত্র 5% ইউজার নতুন ভার্সন পায়; সফল হলে ধীরে ধীরে বাড়ানো হয়

**🔹 ৬. Domain-Driven Design (DDD) for Microservices**

**📌 বর্ণনা:**

Microservice design করার সময় business domain অনুযায়ী service গুলো ভাগ করা হয়।

**🧠 Key Concepts:**

* **Bounded Context**: প্রতিটি সার্ভিস নিজস্ব কনটেক্সটে operate করে
* **Ubiquitous Language**: কোড ও বিজনেস ভাষা এক রাখতে হবে
* **Aggregate Root, Value Object, Entity** ইত্যাদি

**🎯 উদাহরণ:**

* Auth → User Aggregate
* Order → Order Aggregate
* Product → Product Aggregate

**🔹 ৭. Multi-Tenancy Architecture**

**📌 উদ্দেশ্য:**

একই সফটওয়্যারে একাধিক কোম্পানি বা কাস্টমারকে সার্ভিস দেওয়া।

**🧠 Design Options:**

* Database per tenant
* Shared DB, separate schema
* Shared DB & shared schema (with tenant ID)

**📦 Challenge:**

* নিরাপত্তা (এক টেন্যান্ট যেন অন্য টেন্যান্টের ডেটা না দেখে)
* পারফর্মেন্স
* Billing, usage tracking

**🔹 ৮. Security – End-to-End**

**📌 কৌশল:**

* **Zero Trust Network**: সবকিছু যাচাই করো, কিছুই ধরেই নিও না
* **mTLS**: সার্ভিস টু সার্ভিস SSL communication
* **OAuth2 / OpenID Connect**: ইউজার অথেন্টিকেশন
* **JWT**: সার্ভিস টু সার্ভিস অথোরাইজেশন

**📦 টুলস:**

* IdentityServer4
* Keycloak
* AWS Cognito
* Azure B2C

**🔹 ৯. Monitoring & Alerting**

**📌 উদ্দেশ্য:**

সার্ভিসের Health, Usage, Error rate, Latency বুঝতে এবং Alert পেতে।

**📦 Tools:**

* Prometheus (Metrics Collection)
* Grafana (Dashboard)
* Jaeger / Zipkin (Tracing)
* AlertManager / PagerDuty (Alert System)

**🔹 ১০. Infrastructure as Code (IaC)**

**📌 উদ্দেশ্য:**

Cloud Infra কে কোড দিয়ে তৈরি করা ও মেইনটেইন করা।

**📦 টুলস:**

* Terraform (Cloud-Agnostic)
* Bicep / ARM (Azure)
* AWS CloudFormation
* Pulumi

**✅ উপসংহার (Conclusion):**

Microservices Architecture বাস্তবায়ন মানেই শুধু অনেকগুলো সার্ভিস বানানো নয় – বরং নিচের প্রতিটি বিষয় *বুঝে, মাপজোক করে* প্রয়োগ করা:

* Design Patterns (Circuit Breaker, Saga, Event-Driven)
* Secure Communication
* Autonomous Deployment
* Observability and Tracing
* Deployment Strategy

নিশ্চিত! এখানে **Microservices Architecture-এর আরও গভীর, অ্যাডভান্সড স্তরের বাংলা ব্যাখ্যা** দিচ্ছি, যেখানে আপনি পাবেন প্র্যাকটিক্যাল ধারণা, ডিজাইন প্যাটার্ন, এবং বাস্তব জীবনের সমস্যার সমাধান।

**🔥 Microservices Architecture - অ্যাডভান্সড লেভেল বাংলা গাইড**

**১. Circuit Breaker প্যাটার্ন**

* সমস্যা: যখন একটা সার্ভিস বারবার ফেইল করে তখন পুরো সিস্টেম থমকে যেতে পারে।
* সমাধান: Circuit Breaker প্যাটার্নে একটি “সুইচ” থাকে। সার্ভিস যদি ক্রমাগত ফেইল করে, তাহলে ওই সার্ভিসকে আর কল করা বন্ধ করে দেয় এবং ফেইল-সেফ (Fallback) মেসেজ দেয়।
* টুল: Polly (.NET), Resilience4j (Java)

**২. Service Mesh (Istio, Linkerd)**

* কাজ: সার্ভিসগুলোর মধ্যে ট্র্যাফিক ম্যানেজমেন্ট, নিরাপত্তা, মনিটরিং, রিট্রাই ইত্যাদি অটোমেট করে।
* সুবিধা: ডেভেলপারদের কোডে হাত না দেয়া ছাড়াই সার্ভিসের যোগাযোগ ও নিরাপত্তা নিয়ন্ত্রণ করা যায়।
* ব্যবহার: Kubernetes ক্লাস্টারে Istio ইনস্টল করে সার্ভিসের চারপাশে “Sidecar Proxy” হিসেবে কাজ করে।

**৩. Saga Pattern (Distributed Transaction Management)**

* সমস্যা: একাধিক মাইক্রোসার্ভিসে পারস্পরিক সম্পর্কযুক্ত ডেটা পরিবর্তন করার সময় ACID ট্রানজেকশন বজায় রাখা কঠিন।
* সমাধান: Saga pattern এ প্রতিটি সার্ভিসের কাজ স্বতন্ত্রভাবে সম্পন্ন হয় এবং যদি কোনো পর্যায়ে ব্যর্থতা হয়, তবে পূর্ববর্তী কাজগুলো প্রতিপূরণ (Compensation) করা হয়।
* উদাহরণ: অর্ডার প্লেস → পেমেন্ট → ইনভেন্টরি ডিডাকশন, ব্যর্থ হলে পেমেন্ট রিফান্ড এবং অর্ডার বাতিল।

**৪. Event-Driven Architecture**

* সার্ভিসগুলো ইভেন্ট তৈরি ও প্রকাশ করে এবং অন্য সার্ভিসগুলো সেই ইভেন্ট শুনে কাজ করে।
* সুবিধা: Loose coupling, asynchronous processing, scalability।
* টুলস: Apache Kafka, RabbitMQ, NATS

**৫. Canary ও Blue-Green Deployment**

* নতুন ভার্সন ধাপে ধাপে চালু করে সমস্যা হলে সহজে রোলব্যাক দেয়া যায়।
* Blue-Green এ একসাথে দুই ভার্সন চলে, Canary তে নতুন ভার্সন প্রথমে কম সংখ্যক ইউজারের কাছে দেওয়া হয়।

**৬. Domain-Driven Design (DDD)**

* Microservices গুলোকে বিজনেস ডোমেইনের ভিত্তিতে ভাগ করা।
* Bounded Context (প্রতিটি সার্ভিসের নিজস্ব সীমা)
* Ubiquitous Language (ডেভেলপার ও বিজনেস টিম একই ভাষায় কথা বলে)

**৭. Distributed Tracing (Jaeger, Zipkin)**

* একাধিক সার্ভিসে ছড়িয়ে থাকা রিকোয়েস্টের পারফরমেন্স এবং ব্যর্থতা ট্র্যাক করে।
* Latency ও bottleneck সনাক্ত করতে সাহায্য করে।

**৮. Security**

* সার্ভিস টু সার্ভিস যোগাযোগে mTLS দিয়ে এনক্রিপশন।
* OAuth2, JWT ভিত্তিক Authentication ও Authorization।
* Zero Trust Security Model।

**৯. Infrastructure as Code (IaC)**

* টেরাফর্ম, AWS CloudFormation ইত্যাদি ব্যবহার করে ক্লাউড ইনফ্রাস্ট্রাকচার কোডের মাধ্যমে ম্যানেজ করা।

**১০. Observability**

* Metrics, Logs, Traces সমন্বয়ে পুরো সিস্টেমের অবস্থা বুঝতে সাহায্য করে।
* টুলস: Prometheus, Grafana, ELK Stack