**1. How do you decide between monolithic and microservices architecture for a new application?**

✅ **Answer:**

* **Use Monolith if:**
  + The application is small or has tightly coupled features.
  + Quick development and deployment are required.
  + There are no scalability concerns in the short term.
* **Use Microservices if:**
  + The system needs independent scalability.
  + Different teams work on separate functionalities.
  + There’s a need for continuous delivery and rapid iterations.

**2. What are the key considerations while migrating a monolithic application to microservices?**

✅ **Answer:**

* **Identify business domains** and split them into microservices.
* **Decouple dependencies** (databases, shared services).
* Implement **API Gateway** for routing and security.
* Ensure **data consistency** (Saga pattern, event-driven architecture).
* Gradually **refactor and deploy services in phases** to avoid disruption.

**3. How do you ensure data consistency in a distributed microservices environment?**

✅ **Answer:**

* Use **event sourcing** to track changes over time.
* Implement the **Saga pattern** for distributed transactions.
* Apply **idempotency** to avoid duplicate transactions.
* Use **optimistic locking** instead of pessimistic locks for better performance.

**4. How do you handle logging and debugging in microservices?**

✅ **Answer:**

* Use **Centralized Logging** (ELK Stack – Elasticsearch, Logstash, Kibana).
* Implement **distributed tracing** (Zipkin, OpenTelemetry).
* Correlate logs using **trace IDs** across services.
* Capture structured logs in **JSON format** for better analysis.

**5. What are the key strategies for handling failures in a microservices environment?**

✅ **Answer:**

* **Retry mechanism** with exponential backoff.
* **Circuit breaker pattern** (Resilience4j, Hystrix) to prevent cascading failures.
* **Fail-fast and fallback mechanisms** to degrade gracefully.
* Implement **timeouts** to prevent long waits.

**6. How would you implement authentication and authorization in a microservices system?**

✅ **Answer:**

* Use **OAuth2 & OpenID Connect** for authentication.
* Implement **JWT (JSON Web Token)** for secure, stateless authentication.
* Centralize authentication with **Keycloak, Okta, or AWS Cognito**.
* Use **RBAC (Role-Based Access Control)** for service-level security.

**7. How do you protect APIs from excessive requests and DDoS attacks?**

✅ **Answer:**

* Implement **rate limiting** (Token Bucket, Leaky Bucket algorithms).
* Use **API Gateway** (Kong, Apigee) for traffic control.
* Apply **WAF (Web Application Firewall)** to filter malicious requests.
* Enable **CAPTCHA and authentication throttling** for login endpoints.

**8. What strategies can be used to reduce cold start issues in cloud-based microservices?**

✅ **Answer:**

* Use **warm-up requests** to keep instances active.
* Optimize **JVM startup time** (GraalVM, Ahead-of-Time compilation).
* Use **container image optimizations** (lazy loading, minimal base images).
* Keep **hot standby instances** in AWS Lambda or Fargate.

**9. How do you ensure that microservices are properly communicating during high traffic?**

✅ **Answer:**

* Use **service discovery** (Eureka, Consul, Kubernetes).
* Implement **load balancing** (Ribbon, Nginx, HAProxy).
* Enable **backpressure handling** (Reactive programming with WebFlux).
* Use **message queues (Kafka, RabbitMQ)** for async communication.

**10. What are the best practices for designing RESTful APIs?**

✅ **Answer:**

* Use **proper HTTP methods** (GET, POST, PUT, DELETE).
* Follow **resource-based URIs** (/users/{id}, not /getUser).
* Implement **HATEOAS** for discoverability.
* Ensure **versioning** (/api/v1/resource).

**11. What are the main bottlenecks in a Spring Boot application, and how do you address them?**

✅ **Answer:**

* **Slow database queries:** Optimize indexes, caching, and connection pooling.
* **Memory leaks:** Profile memory usage with VisualVM or YourKit.
* **Thread pool exhaustion:** Configure thread pools properly in ExecutorService.
* **Blocking I/O operations:** Use **Reactive Programming (WebFlux)** for better throughput.

**12. How do you handle large file uploads efficiently in a REST API?**

✅ **Answer:**

* Use **multipart uploads** to split large files.
* Store in **cloud storage (S3, Azure Blob, GCS)** instead of a database.
* Stream file processing **asynchronously** instead of loading into memory.
* Use **content delivery networks (CDNs)** for fast access.

**13. How do you prevent over-fetching and under-fetching in APIs?**

✅ **Answer:**

* Implement **GraphQL** for flexible queries.
* Use **projection queries** to fetch only required fields.
* Introduce **pagination and filtering** for large data sets.
* Use **DTOs (Data Transfer Objects)** to return optimized responses.

**14. How do you handle database schema changes in a live production system?**

✅ **Answer:**

* Use **Liquibase or Flyway** for schema versioning.
* Apply **zero-downtime migrations** (rolling updates).
* Introduce **backward-compatible changes** (add new columns instead of dropping).
* Use **feature flags** for gradual rollout.

**15. How do you handle cascading failures in a distributed system?**

✅ **Answer:**

* Implement **bulkheads** to isolate failures.
* Use **circuit breakers** to prevent repeated failures.
* Apply **timeouts and retries** with exponential backoff.
* Monitor system health using **distributed tracing (Jaeger, Zipkin).**

**16. How do you measure API performance and latency?**

✅ **Answer:**

* Use **APM (Application Performance Monitoring) tools** like New Relic, Datadog.
* Measure **p99 latency** instead of average response time.
* Enable **OpenTelemetry metrics** in Spring Boot.
* Use **JMeter or Gatling** for load testing.

**17. What are the best practices for handling secrets and configuration in a cloud-native application?**

✅ **Answer:**

* Store secrets in **AWS Secrets Manager, HashiCorp Vault, Kubernetes Secrets**.
* Use **Environment Variables** instead of hardcoding credentials.
* Enable **role-based access control (RBAC)** for secret access.
* Rotate keys and tokens periodically.

**18. How do you optimize database connection pooling in Spring Boot?**

✅ **Answer:**

* Use **HikariCP** for optimal performance.
* Tune **maximum pool size** based on available DB connections.
* Enable **statement caching** to reuse prepared statements.
* Monitor DB connections with **Actuator metrics**.

**19. How do you ensure message ordering in Kafka?**

✅ **Answer:**

* Use **key-based partitioning** (messages with the same key go to the same partition).
* Set **max.in.flight.requests.per.connection=1** to process one message at a time.
* Enable **idempotent producers** to prevent duplicate processing.

**20. How do you avoid common pitfalls in API versioning?**

✅ **Answer:**

* Use **URI versioning** (/api/v1/resource).
* Support **content negotiation (Accept headers)**.
* Implement **deprecated version warnings** before removal.
* Use **feature flags** for gradual adoption of new versions.