2024 April

Question 1

a) Briefly explain the event-driven programming using an example in Microservices.

(02 marks)

Event-driven programming is a design paradigm in which the flow of the program is determined by events such as user actions, sensor outputs, or messages from other services. In microservices, event-driven architecture enables services to communicate asynchronously. For example, in an e-commerce system, when a customer places an order, an 'Order Placed' event is published to a message broker (like Kafka), and services like Payment, Inventory, and Shipping consume the event and act accordingly.

b) Discuss two (2) methods that can be used to define the scope of the microservices and to right-size them.

(04 marks)

1. Domain-Driven Design (DDD): Microservices should align with business domains using bounded contexts, ensuring services handle specific business functionalities.

2. Single Responsibility Principle (SRP): Each microservice should have a clear and focused responsibility, avoiding excessive coupling and ensuring scalability.

(05 Explain what an event streaming platform is.

(04 marks)

An event streaming platform is a system that allows real-time data streaming and processing. It enables microservices to communicate asynchronously through events. Apache Kafka is a popular event streaming platform that stores, processes, and distributes event-driven data reliably.

Question 2

Explain the following each of the concepts in the context of Microservices and Cloud Computing.

a) Software Architectural Styles

(05 marks)

1. Monolithic Architecture: A single unified application where all components are tightly coupled.

2. Microservices Architecture: Applications are decomposed into loosely coupled, independently deployable services.

3. Serverless Architecture: Developers deploy functions that execute on-demand in a cloud-managed environment.

4. Event-Driven Architecture: Systems react to events asynchronously, enabling scalability.

5. Layered Architecture: Organizes applications into logical layers (e.g., presentation, business logic, data access).

b) SAAS, PAAS, IAAS

(06 marks)

1. Software as a Service (SAAS): Fully managed applications accessed via the internet (e.g., Gmail, Dropbox).

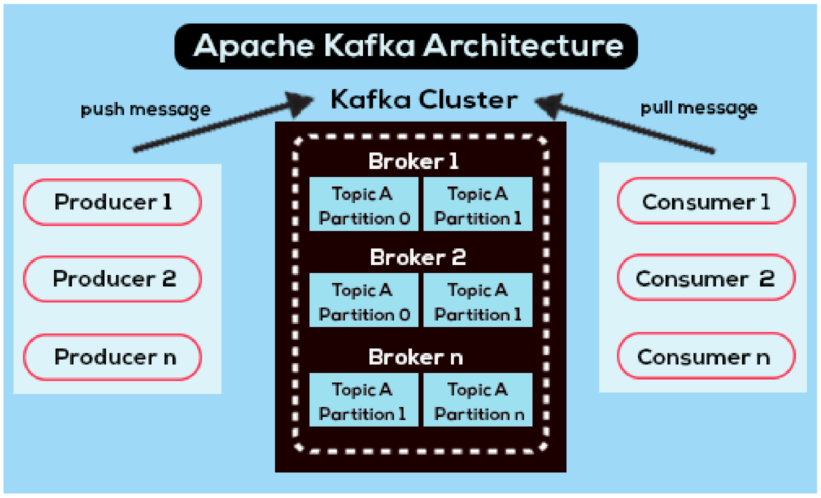
2. Platform as a Service (PAAS): Provides development environments (e.g., Google App Engine, Heroku).

3. Infrastructure as a Service (IAAS): Provides virtualized computing resources (e.g., AWS EC2, Microsoft Azure VMs).

c) Kafka

(05 marks)

Kafka is a distributed event-streaming platform used for real-time data streaming and message brokering. It ensures fault tolerance, high availability, and horizontal scalability, making it ideal for decoupling microservices and handling high-throughput data.



Question 3

a) Compare Cloud native and traditional software development approaches, based on five (5) comparative points.

(05 marks)

Feature	**Cloud-Native**	**Traditional Development**
Deployment	Containerized (Docker, Kubernetes)	Deployed on physical/virtual servers
Scalability	Auto-scalable	Manual scaling required
Architecture	Microservices, Serverless	Monolithic
Resilience	Built-in fault tolerance	Requires additional failover mechanisms
DevOps Integration	CI/CD pipelines	Manual deployments

b) Explain five (5) coding standards that should be followed when using cloud-native programming.

(05 arks)

1. Codebase: A single codebase tracked in version control (e.g., Git). Multiple deployments (dev, staging, production) should refer to the same codebase.

2. API First: Design applications with an API-first approach, ensuring all functionalities are exposed via APIs (e.g., RESTful APIs or gRPC).

3. Build, Release, Run: Strictly separate the build, release, and run stages. Automate the deployment pipeline using CI/CD tools.

4. Authentication and Authorization: Apply authentication (OAuth2, JWT) and encryption.

5. Config: Store configuration in environment variables (e.g., API keys, database URLs). Use tools like dotenv for local development and cloud configuration services for secure storage.

c) Assume that you are requested to containerize microservices developed with Spring Boot using spring build packs. List the steps of this process in sequence.

(05 arks)

1. Install pack CLI or use Spring Boot's spring-boot:build-image.

2. Navigate to the Spring Boot project directory.

3. Run ./mvnw spring-boot:build-image (Maven) or ./gradlew bootBuildImage (Gradle).

4. Push the containerized image to a container registry.

5. Deploy the container to a Kubernetes or cloud environment.

Answer Questions 4, and 5 based on the use case below.

Use case:

"Crystalline Apparel" is a leading organization in the apparel sector. Their operational line for manufacturing clothes which is known as the shop flow control system plays a critical role in meeting their customer demands. Currently, there are a lot of issues in tracking the manufacturing flow in the operational line as there is no system in place to monitor and track the manufacturing from raw material to finished products. Assume the Organization has consulted you to propose and implement a fully automated system that can track and monitor the end-to-end process. You are representing the consultancy firm.

Question 4

You are appointed as the new Solution Architect for this project and are supposed to come up with the solution architecture diagram which will be a base document for the developers to commence development.

a) Explain the best practices that you should follow while developing the architecture diagram.

(08 marks)

1. **Clarity and Simplicity**: Use standard notations (e.g., UML, BPMN) and avoid overcomplicating the diagram. Focus on high-level components.
2. **Layered Abstraction**: Organize the diagram into layers (presentation, application, data, infrastructure) to separate concerns.
3. **Consistency**: Maintain uniform symbols, colors, and labeling conventions. Include a legend.
4. **Modularity**: Highlight reusable components (e.g., microservices, APIs) to emphasize scalability.
5. **Stakeholder Relevance**: Tailor the diagram to the audience (e.g., technical details for developers, cost/security focus for executives).

6. **Security and Compliance**: Include firewalls, encryption, and IAM roles to address data protection.
7. **Documentation**: Supplement the diagram with text explaining design choices, trade-offs, and non-functional requirements.

b) Briefly explain why you should propose the Kubernetes container orchestration platform as a major component in your solution architecture diagram.

(07 marks)

- **Scalability**: Automatically scales microservices to handle fluctuating manufacturing workloads (e.g., peak production periods).
- **High Availability**: Ensures 24/7 uptime with self-healing pods and multi-node clusters.
- **CI/CD Integration**: Supports rolling updates for seamless deployment of new features without downtime.
- **Resource Optimization**: Efficiently manages hardware resources across distributed systems.
- **Microservices Orchestration**: Simplifies deployment and management of modular services (e.g., raw material tracking, quality checks).

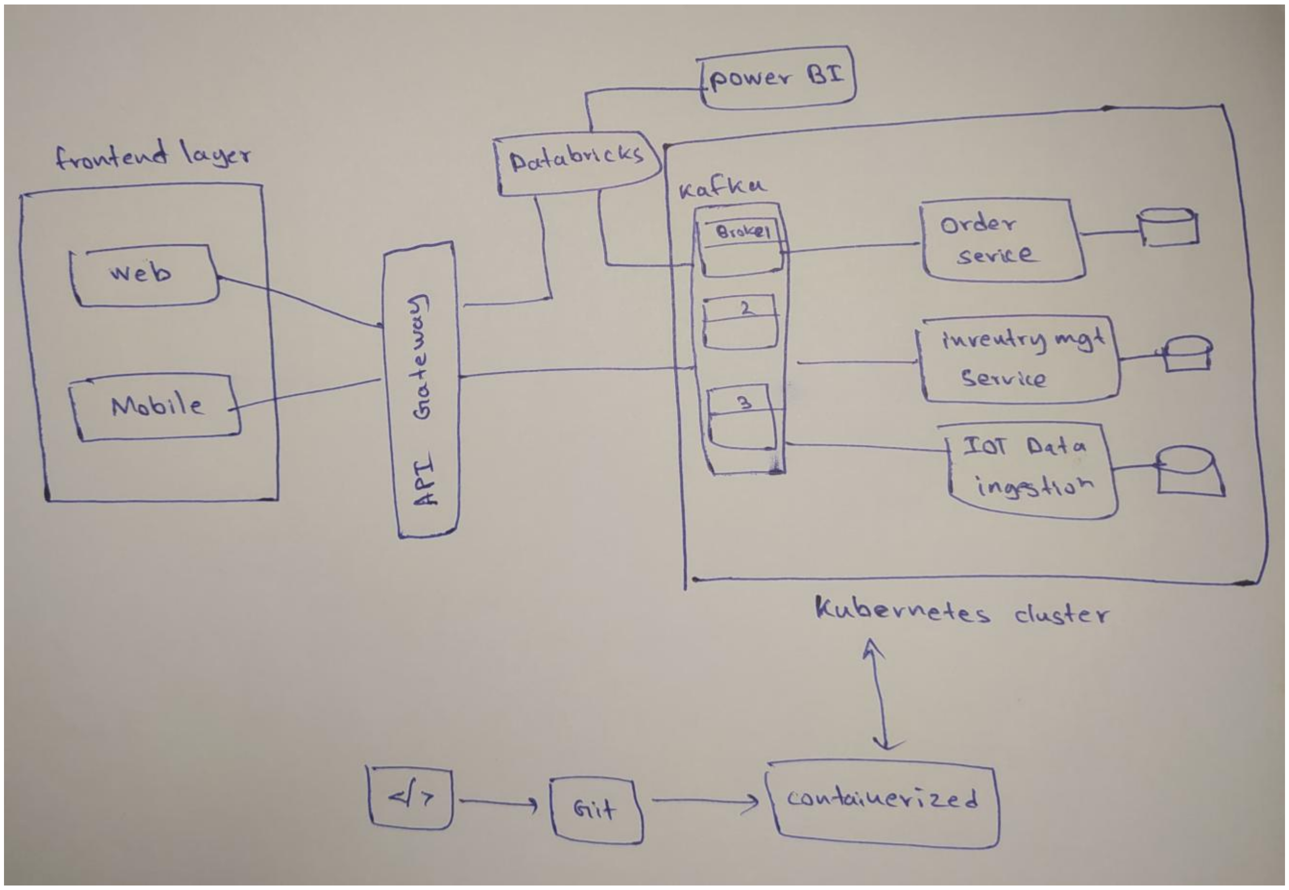
c) Explain briefly why you should use Databricks as the data platform for data analytics.

(05 marks)

- **Unified Analytics**: Combines batch/stream processing (via Apache Spark) for real-time monitoring of production lines.
- **Machine Learning**: Enables predictive analytics (e.g., defect detection, demand forecasting) using MLlib.
- **Collaboration**: Shared notebooks allow cross-functional teams (operations, data engineers) to collaborate.
- **Delta Lake Integration**: Provides ACID compliance for reliable data pipelines, critical for audit trails.
- **Cloud-Native**: Seamlessly integrates with cloud storage (e.g., AWS S3, Azure Blob) for scalable data lakes.

d) Draw the proposed solution architecture diagram.

(10 marks)



Question 5

As a Developer on this project, you are supposed to decide upon the software architecture style.

a) How do you justify microservices as the suitable architecture for the solution?

(10 marks)

- **Modularity**: The manufacturing process involves distinct components (e.g., raw material tracking, quality inspection, order fulfillment). Microservices allow each component to operate independently, enabling focused development and maintenance.
- **Scalability**: High-demand services (e.g., real-time IoT data ingestion) can scale horizontally without affecting others, crucial for handling peak production periods.
- **Fault Isolation**: A failure in one service (e.g., inventory management) won't disrupt the entire production line, ensuring system resilience.
- **Technology Flexibility**: Teams can use different tools (e.g., Python for analytics, Java for transactional services) suited to specific tasks.
- **CI/CD Compatibility**: Microservices align with Kubernetes and DevOps pipelines, enabling rapid deployment of updates (e.g., defect detection algorithms).

b) What are the design patterns you are suggesting for this solution?

(08 marks)

1. **API Gateway:**
    - **Centralizes authentication, routing, and load balancing for microservices (e.g., integrating ERP systems with IoT data streams).**
2. **Circuit Breaker:**
    - **Prevents cascading failures during outages (e.g., if the raw material tracking service crashes).**

3. **Event Sourcing:**
    - **Captures state changes as a sequence of events (e.g., tracking material movement via Apache Kafka), enabling audit trails and recovery.**
4. **CQRS (Command Query Responsibility Segregation):**
    - **Separates read/write operations to optimize performance (e.g., fast analytics queries vs. transactional updates).**
5. **Saga Pattern:**
    - **Manages distributed transactions (e.g., synchronizing inventory reduction with order fulfillment).**

c) If you are playing the Cloud Architect role in this project, briefly explain the migration strategy to move the on-premises developed microservices to the cloud

(12 marks)

1. **Containerization:**
    - **Package on-premises microservices into Docker containers for portability.**
2. **Phased Migration:**
    - **Phase 1: Lift-and-shift non-critical services (e.g., reporting modules) to cloud VMs for quick wins.**
    - **Phase 2: Refactor critical services (e.g., IoT data ingestion) to leverage cloud-native tools (e.g., AWS Kinesis/Azure Event Hubs).**
3. **Kubernetes Orchestration:**
    - **Deploy containers to managed Kubernetes platforms (e.g., AWS EKS, Azure AKS) for auto-scaling and self-healing.**
4. **Data Migration:**
    - **Use hybrid cloud storage (e.g., AWS Storage Gateway) for incremental transfer of production databases to cloud-native solutions (e.g., Amazon RDS, Azure Cosmos DB).**
5. **Networking and Security:**

    - **Establish secure VPN/ExpressRoute connections between on-premises and cloud.**
    - **Implement cloud IAM policies and encrypt data at rest/in transit.**
6. **Testing and Cutover:**
    - **Conduct A/B testing and monitor performance using cloud-native tools (e.g., CloudWatch).**
    - **Gradually shift traffic to cloud services after validation.**