Que - 01.

Serverless architecture offers a highly scalable solution for backend services, eliminating the need for traditional server management. Here's how to leverage it:

1. Choose a Serverless Platform:

- **AWS Lambda:** Known for its extensive ecosystem and integration with other AWS services.
- **Azure Functions:** Offers a similar experience to Lambda with deep integration into the Azure cloud.
- **Firebase Cloud Functions:** Ideal for real-time applications and mobile development.

2. Trigger Mechanisms:

- **HTTP Triggers:** Invoke functions in response to API requests.
- **Event Triggers:** React to events from other services (e.g., S3, DynamoDB, CloudWatch).
- **Schedule Triggers:** Run functions on a predefined schedule.

4. Automatic Scaling:

- Serverless platforms handle scaling automatically based on demand. Functions are provisioned and deprovisioned as needed.

5. Cold Starts:

- Be aware of cold starts, which occur when a function is invoked for the first time after a period of inactivity. Optimize code and use caching to minimize their impact.

6. Leverage Managed Services:

- Utilize managed services like:
    - **API Gateway:** Handle API requests and routing.
    - **DynamoDB:** Provide a scalable NoSQL database.
    - **S3:** Store and retrieve files.
    - **CloudWatch:** Monitor and log function performance.

7. Consider Hybrid Approaches:

- For complex workloads, combine serverless functions with traditional infrastructure or managed services for specific tasks.

Example:

- **API Endpoint:** Create a Lambda function triggered by an API Gateway.
- **Database Interaction:** Use DynamoDB to store and retrieve data.
- **Scaling:** The serverless platform automatically scales the Lambda function and DynamoDB provisioned capacity based on traffic.

Benefits of Serverless:

- **Scalability:** Automatic scaling based on demand.
- **Cost-Efficiency:** Pay only for the resources used.
- **Focus on Code:** Spend more time on application logic and less on infrastructure.

Que 3.

Redis: A popular in-memory data structure store with support for various data types.

Cache-Aside: The application queries the cache first. If the data is not found, it fetches it from the database, stores it in the cache, and returns it to the client.

Que - 5.

Define Roles and Permissions:

- **Roles:** Create a list of roles (e.g., admin, user, guest).
- **Permissions:** Define specific actions that users can perform (e.g., read, write, delete).
- **Role-Permission Mapping:** Associate roles with specific permissions.

User Authentication:

- Implement a user authentication mechanism (e.g., JWT, session-based) to verify user identity before applying RBAC.

Que 7.

Code Optimization:

- **Avoid Unnecessary Allocations:** Minimize object creation and allocations.
- **Efficient Data Structures:** Choose data structures that are suitable for your use case (e.g., arrays, maps, sets).
- **Optimized Algorithms:** Select algorithms with lower memory complexity.

Caching Strategies:

- **Cache Frequently Accessed Data:** Store frequently used data in memory to reduce database queries and object creations.
- **Appropriate Caching Strategies:** Choose caching strategies like in-memory caching or distributed caching based on your needs.