Q9: Failover and Redundancy

To ensure the high availability of the services and handle potential system failures, I would implement a comprehensive failover and redundancy strategy covering:

1. **Service Level Redundancy**
    - **Stateless Microservices:**
        - Multiple instances of each service deployed across availability zones.
        - Services designed to be stateless, allowing any instance to handle requests.
        - Independent scaling and replacement of service instances.
    - **Load Balancing:**
        - Load balancers distribute traffic across healthy service instances.
        - Health checks automatically remove unhealthy instances from rotation.
        - Sticky sessions for WebSockets while maintaining failover capability.
    - **Service-Specific Recovery:**
        - Authentication Service: Multiple instances with shared JWT verification capabilities.
        - Document Service: State recovery from MongoDB and version vectors.
        - Real-time Collaboration Service: WebSocket reconnection protocols.
        - Kafka Consumer Service: Offset tracking for recovery after restarts.
2. **Data Storage Redundancy**
    - **MongoDB Cluster Redundancy:**
        - Sharded cluster with replica sets for each shard.
        - Automatic primary/secondary failover within replica sets.
        - Data distributed across multiple availability zones.
        - Distributed locks for coordinating critical operations.
    - **PostgreSQL High Availability:**
        - Primary/standby configuration with automatic failover.
        - Synchronous replication for critical data (user credentials, permissions).
        - Read replicas for distributing query loads.
        - Regular backups with point-in-time recovery.
    - **Redis Cache Redundancy:**
        - Redis cluster with sentinel for automatic failover.
        - Cross-AZ deployment for resilience.
        - Graceful degradation to MongoDB if Redis is unavailable.
        - Data in Redis is treated as ephemeral with recovery from source data.
3. **Messaging System Redundancy**
    - **Kafka Reliability:**
        - Multi-broker Kafka cluster with replication.
        - Topic replication across multiple brokers.
        - Automated leader election for topics.
        - Consumer group rebalancing for Kafka Consumer Service instances.
    - **Kafka Failure Recovery Process:**
        - Document Service detects Kafka unavailability.
        - Fallback to direct MongoDB operation retrieval.
        - Operations continue functioning with degraded performance.
        - Automatic recovery and catch-up when Kafka is restored.

4. **System Failure Handling**
   - **Server Failure Recovery:**
     - New servers retrieve the last processed version vector from MongoDB.
     - Pending CRDT operations are fetched based on timestamp.
     - Latest document snapshot is loaded from MongoDB.
     - Operations are applied in causal order using version vectors.
     - Server becomes consistent with the rest of the system.

   - **Partial System Failures:**
     - Circuit breakers prevent cascading failures.
     - Retry mechanisms with exponential backoff for transient issues.
     - Fallback to degraded functionality when dependencies are unavailable.
     - Prioritization of critical operations during recovery.

   - **Network Partition Handling:**
     - CRDT-based approach tolerates temporary network partitions.
     - Version vectors track causality across partitions.
     - Eventual consistency when partitions heal.

5. **Disaster Recovery**
   - **Multi-Region Strategy:**
     - Option to deploy to multiple geographic regions.
     - Cross-region data replication for critical data.
     - Regional failover for complete region outage.

   - **Backup and Restore:**
     - Regular automated backups of PostgreSQL and MongoDB data.
     - Document version history preserved in MongoDB.
     - Point-in-time recovery capability.
     - Regular disaster recovery testing.

   - **Data Integrity Protection:**
     - Immutable document version snapshots.
     - CRDT operations stored with idempotency guarantees.
     - Response data integrity protected with transaction guarantees.