**Basic Questions**

1. **What is RabbitMQ?**
   * RabbitMQ is an open-source message broker that facilitates communication between different applications or microservices by queuing, routing, and delivering messages.
2. **What are the main components of RabbitMQ?**
   * **Producer**: Application sending messages.
   * **Queue**: Buffer that stores messages.
   * **Exchange**: Routes messages to queues based on binding rules.
   * **Consumer**: Application or service that receives messages.
3. **What is the purpose of a message broker like RabbitMQ?**
   * Decouple applications for better scalability, reliability, and performance by handling message delivery asynchronously.
4. **How does RabbitMQ ensure reliability?**
   * **Acknowledgments**: Messages are confirmed by consumers upon processing.
   * **Persistence**: Messages can be saved to disk.
   * **Mirrored Queues**: High availability through replication.
   * **Dead Letter Exchange**: Unprocessable messages are redirected.

**Intermediate Questions**

1. **What are exchanges in RabbitMQ, and what are their types?**
   * **Exchanges** route messages to one or more queues. Types:
     + **Direct**: Routes based on an exact match with the routing key.
     + **Topic**: Matches routing keys with patterns.
     + **Fanout**: Broadcasts messages to all bound queues.
     + **Headers**: Routes based on message header values.
2. **What is a virtual host (vhost) in RabbitMQ?**
   * A virtual host is a logical separation within RabbitMQ, allowing multiple tenants to use isolated namespaces for queues, exchanges, and bindings.
3. **How does RabbitMQ handle message acknowledgment?**
   * RabbitMQ uses two types of acknowledgment:
     + **Automatic Ack**: Immediately confirms message delivery (may result in data loss if processing fails).
     + **Manual Ack**: Consumer explicitly acknowledges after successful processing.
4. **What are bindings in RabbitMQ?**
   * A binding is a link between a queue and an exchange. It defines how messages are routed from the exchange to the queue.
5. **How does RabbitMQ handle dead-letter messages?**
   * Messages that are unprocessed (due to rejection or TTL expiration) can be sent to a **Dead Letter Exchange**(DLX) for further handling or inspection.
6. **What is prefetch count in RabbitMQ?**
   * It limits the number of messages a consumer can process simultaneously, preventing a single consumer from being overloaded.

**Advanced Questions**

1. **Explain how RabbitMQ ensures high availability.**
   * **Clustering**: Multiple nodes work together.
   * **Mirrored Queues**: Replicate queues across nodes.
   * **Quorum Queues**: A modern alternative for high availability using the Raft consensus algorithm.
2. **What is the difference between persistent and transient messages?**
   * **Persistent Messages**: Survive broker restarts (stored on disk).
   * **Transient Messages**: Stored in memory and lost on broker restart.
3. **How can you monitor RabbitMQ?**
   * Using the RabbitMQ Management Plugin, Prometheus metrics, or third-party monitoring tools like Datadog.
4. **What are some common RabbitMQ issues, and how would you troubleshoot them?**
   * **High memory usage**: Check unacknowledged messages or persistent messages in queues.
   * **Message delays**: Investigate queue size, consumer performance, or network latency.
   * **Connection issues**: Review logs, ensure correct ports are open, and verify authentication.
5. **What are quorum queues, and why are they used?**
   * Quorum queues use the Raft protocol to achieve better consistency and reliability, especially in distributed systems.

**Scenario-Based Questions**

1. **How would you design a RabbitMQ system to handle message retries?**
   * Use a Dead Letter Exchange with a Time-to-Live (TTL) to requeue messages after a failure.
2. **How can you ensure message order in RabbitMQ?**
   * Use a single consumer per queue, as RabbitMQ does not guarantee order with multiple consumers.
3. **How would you scale RabbitMQ consumers?**
   * Add more consumers or use multiple queues with load balancing.
4. **How can you secure RabbitMQ?**
   * Enable TLS for encryption, configure user permissions, limit access to vhosts, and monitor activity.
5. **When would you choose RabbitMQ over Kafka?**
   * Use RabbitMQ for traditional message queuing, real-time tasks, or when you need support for complex routing.
   * Choose Kafka for high-throughput, distributed systems requiring durable logs.

**Types of Exchanges in RabbitMQ**

1. **Direct Exchange**
   * **Purpose**: Routes messages to queues whose binding key exactly matches the message’s routing key.
   * **Use Case**: Point-to-point communication where specific routing is needed.
   * **Example**:
     + Binding: queue1 with key orange.
     + Message sent with routing key orange will go to queue1.
2. **Topic Exchange**
   * **Purpose**: Routes messages based on pattern matching between the routing key and the queue’s binding key.
   * **Pattern Symbols**:
     + \*: Matches exactly one word.
     + #: Matches zero or more words.
   * **Use Case**: Publish/subscribe systems with complex routing logic.
   * **Example**:
     + Binding: queue1 with key animals.\*.
     + Message sent with routing key animals.dog will go to queue1.
3. **Fanout Exchange**
   * **Purpose**: Broadcasts messages to all queues bound to the exchange, ignoring routing keys.
   * **Use Case**: Use when all consumers need to receive the same message (e.g., event broadcasting).
   * **Example**:
     + Two queues (queue1 and queue2) bound to a fanout exchange receive all messages published to it.
4. **Headers Exchange**
   * **Purpose**: Routes messages based on message header attributes instead of routing keys.
   * **Use Case**: Advanced use cases where message metadata determines routing.
   * **Example**:
     + Binding: queue1 expects header { "format": "pdf", "type": "report" }.
     + Message with matching headers will be routed to queue1.

**Using RabbitMQ Exchanges**

**1. Direct Exchange**

* **Purpose**: Route messages to specific services based on exact routing keys.
* **Example**:
  + **Exchange**: direct\_orders
  + **Routing Keys**:
    - order.created: Sent to the order processing service.
    - order.cancelled: Sent to both inventory and notification services.
  + **Flow**:
    - When a new order is placed, the producer sends a message with order.created, which is routed to the order processing queue.

**2. Topic Exchange**

* **Purpose**: Enable routing based on patterns, such as message types and categories.
* **Example**:
  + **Exchange**: topic\_ecommerce
  + **Routing Keys**:
    - order.\*: Handles all order-related events.
    - inventory.low\_stock: Tracks low-stock warnings for inventory service.
    - notification.\*: Routes notifications to appropriate channels (e.g., email or SMS).
  + **Flow**:
    - A message with the routing key notification.email is routed to the email service queue.
    - A message with the routing key inventory.low\_stock notifies inventory management.

**3. Fanout Exchange**

* **Purpose**: Broadcast messages to multiple systems.
* **Example**:
  + **Exchange**: fanout\_order\_updates
  + **Use Case**: Notify multiple services about a new order:
    - Order Management
    - Inventory System
    - Analytics Service
  + **Flow**:
    - A message published to fanout\_order\_updates is sent to all queues bound to the exchange.

**4. Headers Exchange**

* **Purpose**: Route messages based on metadata (e.g., user preferences or order priority).
* **Example**:
  + **Exchange**: headers\_notifications
  + **Bindings**:
    - High-priority notifications: { "priority": "high", "channel": "sms" }
    - Low-priority notifications: { "priority": "low", "channel": "email" }
  + **Flow**:
    - A message with headers { "priority": "high", "channel": "sms" } is routed to the SMS notification queue.