### **Step 1: Setting Up RabbitMQ Using Docker**

1. **Install Docker**:
   * Ensure Docker is installed on your system. If not, download and install Docker from [Docker's official website](https://www.docker.com/).
2. **Pull the RabbitMQ Docker Image**:

Pull the official RabbitMQ Docker image with management plugins enabled:  
bash  
Copy code  
docker pull rabbitmq:3-management

### **Step 2: Basic Configuration and Plugins**

1. **Start RabbitMQ Container with Management Plugin**:

Run a RabbitMQ container with the management plugin enabled:  
bash  
Copy code  
docker run -d --hostname rabbit-master --name rabbitmq-master -p 15672:15672 -p 5672:5672 rabbitmq:3-management

1. **Access RabbitMQ Management Console**:
   * Open a browser and navigate to http://localhost:15672.
   * Default credentials:
     + Username: guest
     + Password: guest
2. **Enable Additional Plugins (Shovel and Federation)**:

Enter the running container:  
bash  
Copy code  
docker exec -it rabbitmq-master bash

Enable Shovel and Federation plugins:  
bash  
Copy code  
rabbitmq-plugins enable rabbitmq\_shovel rabbitmq\_federation

### **Step 3: Understanding RabbitMQ Components**

1. **Exchanges**:
   * Define an exchange as the entry point where messages are received from producers.
   * Types: direct, fanout, topic, and headers.
2. **Queues**:
   * Queues store messages until they are processed by consumers.
3. **Bindings**:
   * Bindings link exchanges to queues based on routing keys.
4. **Routing Keys**:
   * Used to route messages to appropriate queues.

### **Step 4: RabbitMQ with Docker**

1. **Deploy RabbitMQ Master Node**:
   * Already deployed in Step 2. This will act as the master node.
2. **Deploy RabbitMQ Slave Nodes**:

Start two additional RabbitMQ containers for slave nodes:  
bash  
Copy code  
docker run -d --hostname rabbit-slave1 --name rabbitmq-slave1 -p 15673:15672 -p 5673:5672 rabbitmq:3-management

docker run -d --hostname rabbit-slave2 --name rabbitmq-slave2 -p 15674:15672 -p 5674:5672 rabbitmq:3-management

1. **Set Up RabbitMQ Cluster**:
   * Join the slave nodes to the master node cluster:

Connect to the slave node's container:  
bash  
Copy code  
docker exec -it rabbitmq-slave1 bash

Join the slave to the master:  
bash  
Copy code  
rabbitmqctl stop\_app

rabbitmqctl join\_cluster rabbit@rabbit-master

rabbitmqctl start\_app

* + - Repeat the same for rabbitmq-slave2.

### **Step 5: Docker Compose for Multi-Container Applications**

1. **Create a docker-compose.yml File**:

Define services for master and slave nodes:  
yaml  
Copy code  
version: '3'

services:

rabbitmq-master:

image: rabbitmq:3-management

container\_name: rabbitmq-master

hostname: rabbit-master

ports:

- "15672:15672"

- "5672:5672"

networks:

- rabbitmq-network

rabbitmq-slave1:

image: rabbitmq:3-management

container\_name: rabbitmq-slave1

hostname: rabbit-slave1

ports:

- "15673:15672"

- "5673:5672"

networks:

- rabbitmq-network

environment:

- RABBITMQ\_NODENAME=rabbit@rabbit-slave1

rabbitmq-slave2:

image: rabbitmq:3-management

container\_name: rabbitmq-slave2

hostname: rabbit-slave2

ports:

- "15674:15672"

- "5674:5672"

networks:

- rabbitmq-network

environment:

- RABBITMQ\_NODENAME=rabbit@rabbit-slave2

networks:

rabbitmq-network:

driver: bridge

1. **Deploy the Cluster**:

Run docker-compose to bring up all containers:  
bash  
Copy code  
docker-compose up -d

### **Step 6: Managing and Scaling RabbitMQ in Docker**

1. **Scaling Containers**:
   * To scale RabbitMQ nodes, modify the docker-compose.yml and add more slave services or adjust the number of replicas.
2. **Health Checks**:
   * Implement health checks in docker-compose.yml to monitor the status of the containers.

### **Step 7: Exchanges, Queues, Bindings, and Routing Keys**

1. **Create Exchanges and Queues**:
   * In the RabbitMQ management console, create exchanges and queues needed for your application.
2. **Define Bindings and Routing Keys**:
   * Set up bindings to connect exchanges to queues using appropriate routing keys.

### **Step 8: Message Flow in RabbitMQ**

1. **Message Producer**:
   * Create a producer that sends messages to the appropriate exchange with a routing key.
2. **Message Consumer**:
   * Create consumers that listen to queues and process messages.

### **Step 9: Message Acknowledgment and Persistence**

1. **Enable Message Persistence**:

When declaring queues, ensure they are durable:  
java  
Copy code  
channel.queueDeclare("myQueue", true, false, false, null);

Ensure that messages are marked as persistent:  
java  
Copy code  
AMQP.BasicProperties properties = new AMQP.BasicProperties.Builder()

.deliveryMode(2) // 2 means persistent

.build();

channel.basicPublish("myExchange", "routingKey", properties, message.getBytes());

1. **Acknowledge Messages**:
   * Consumers should acknowledge messages after processing:  
     java  
     Copy code  
     channel.basicAck(deliveryTag, false);

### **Step 10: Add Fault Tolerance**

#### **Set Up Mirrored Queues**

1. **Create and Apply a Policy for Mirrored Queues**:
   * RabbitMQ’s mirrored queues feature allows you to replicate queues across multiple nodes, providing fault tolerance.
   * To set up mirrored queues, apply a policy to all queues or specific queues that require mirroring.

Access the RabbitMQ management console or use the RabbitMQ CLI:  
bash  
Copy code  
rabbitmqctl set\_policy ha-all ".\*" '{"ha-mode":"all"}'

* + - "ha-all": The name of the policy.
    - ".\*": A regular expression to match the names of queues to be mirrored. The regex ".\*" matches all queues.
    - {"ha-mode":"all"}: The policy definition, where "ha-mode":"all" mirrors the queue to all nodes in the cluster.

1. **Verifying the Policy**:
   * After setting the policy, you can verify that the queues are mirrored across all nodes.
   * In the RabbitMQ management console, navigate to the "Queues" tab and inspect the mirrored queues. They should show multiple nodes under the "Nodes" column.
2. **Customizing Mirroring Policies**:
   * If you need more control over which queues are mirrored or want to limit mirroring to a subset of nodes, you can use more advanced configurations:

**Mirror to specific nodes**:  
bash  
Copy code  
rabbitmqctl set\_policy ha-two ".\*" '{"ha-mode":"exactly","ha-params":2,"ha-sync-mode":"automatic"}'

* + - * "ha-mode":"exactly": Specifies the exact number of nodes to mirror to.
      * "ha-params":2: Number of nodes to mirror.
      * "ha-sync-mode":"automatic": Ensures that messages are synchronized automatically across nodes.

1. **Testing Mirrored Queues**:
   * To ensure that the mirrored queues are functioning correctly, publish some messages to a queue and observe the behavior across nodes.

#### **Test Fault Tolerance**

1. **Gracefully Stop a Node**:

Stop one of the slave nodes to test the system’s resilience:  
bash  
Copy code  
docker stop rabbitmq-slave1

Alternatively, you can force-stop the node to simulate a crash:  
bash  
Copy code  
docker kill rabbitmq-slave1

1. **Monitor the Cluster Behavior**:
   * Check the RabbitMQ management console for the status of the remaining nodes.
   * The queues should still be accessible, and consumers should be able to process messages without interruption.
2. **Failover and Recovery**:
   * When a node fails, RabbitMQ automatically reroutes operations to the remaining nodes in the cluster.
   * After stopping a node, send a few test messages to verify that the remaining nodes are handling the load.
3. **Restart the Node**:

Bring the stopped node back online:  
bash  
Copy code  
docker start rabbitmq-slave1

* + RabbitMQ will automatically reintegrate the node into the cluster, and the queues will start syncing again if "ha-sync-mode":"automatic" is set.

1. **Check for Message Loss**:
   * During the downtime, no messages should be lost if everything is configured correctly.
   * Verify that all messages were processed and delivered as expected.
2. **Simulate Master Node Failure**:

Test the failover by stopping the master node to see how the system handles the failure:  
bash  
Copy code  
docker stop rabbitmq-master

* + The slaves should continue processing the messages. After verifying, restart the master node to bring it back into the cluster.