Docker Swarm - Container Communication

**Problem Statement:**

The problem is to demonstrate container communication using Docker Swarm. We want to create a Docker service with multiple replicas running an NGINX container. The goal is to verify that the containers in the Docker service can communicate with each other and can be accessed from the host machine.

**Procedure:**

1. Install Docker: Docker is installed on the Ubuntu machine to enable containerization.

2. Determine the desired IP address: In case the system has multiple network interfaces with different IP addresses, we need to identify the IP address of the desired network interface that we want to use for Docker Swarm.

3. Initialize Docker Swarm with the specified IP address: The Docker Swarm is initialized using the `sudo docker swarm init --advertise-addr` command, specifying the desired IP address. This step creates a Swarm manager and provides a token for joining worker nodes.

4. Create a Docker service: We create a Docker service using the `sudo docker service create` command. In this case, we create a service named "my-service" with 2 replicas running the NGINX container. The port 8080 of the host machine is mapped to port 80 of the NGINX containers.

5. Verify the service: We check the status of the Docker service using the `sudo docker service ls` command to ensure that the "my-service" is running with the expected number of replicas.

6. Test container communication: We access the NGINX welcome page in a web browser by visiting `http://localhost:8080` or using the `curl http://localhost:8080` command in the terminal. This confirms that the containers in the Docker service are accessible and can communicate with each other.

7. Scale the service: We scale the Docker service to have more replicas using the `sudo docker service scale` command. In this case, we increase the number of replicas to 4.

8. Verify the scaled service: We check the service status again using the `sudo docker service ls` command to ensure that the "my-service" now has four replicas.

9. Cleanup: Finally, we remove the Docker service using the `sudo docker service rm` command and stop Docker Swarm on the current node using the `sudo docker swarm leave --force` command.

**Coding:**

The main coding involves executing commands in the Ubuntu terminal. These commands include installing Docker, initializing Docker Swarm with the specified IP address, creating a Docker service, scaling the service, verifying the service status, and performing cleanup actions.

Certainly! Here's the modified code with the explicit IP address specified for Docker Swarm initialization:

**Step 1: Install Docker**

Open the terminal.

su

sudo apt update

sudo apt install docker.io

sudo systemctl start docker

sudo systemctl enable docker

**Step 2: Determine the desired IP address**

ifconfig

**Step 3: Initialize Docker Swarm with the specified IP address**

sudo docker swarm init --advertise-addr 10.0.2.15

**Step 4: Create a Docker service**

sudo docker service create --name my-service --replicas 2 -p 8080:80 nginx:latest

**Step 5: Verify the service**

sudo docker service ls

**Step 6: Test container communication**

Open a new terminal **–** Terminal 2

curl http://localhost:8080

This confirms that the containers in the Docker service are accessible via the specified port on the local machine.

**Terminal 1**

**Step 7: Scale the service**

sudo docker service scale my-service=4

**Step 8: Verify the scaled service**

sudo docker service ls

Verify that the "my-service" now has four replicas.

**Step 9: Cleanup**

sudo docker service rm my-service

sudo docker swarm leave --force

**Learning Outcome:**

By following the above code and procedure, you will learn the following:

1. Docker Swarm initialization: You will learn how to initialize Docker Swarm on a single machine and specify the IP address for advertising.

2. Creating Docker services: You will understand how to create Docker services using the `docker service create` command and specify the desired number of replicas and port mapping.

3. Scaling Docker services: You will learn how to scale Docker services to increase or decrease the number of replicas using the `docker service scale` command.

4. Verifying service status: You will learn how to check the status of Docker services using the `docker service ls` command to ensure they are running as expected.

5. Container communication: You will see how containers within a Docker service can communicate with each other and be accessed from the host machine.

6. Cleanup: You will learn how to remove Docker services and leave Docker Swarm on the current node.