

Docker Advanced: Networks, Volumes, and Microservices

Objectives :

- Run multiple containers together using Docker Compose
- Understand microservices communication using RabbitMQ
- Observe container-to-container networking
- Understand data persistence using Docker volumes
- Edit and understand YAML configuration files

Concept Overview :

Microservices

Microservices architecture is a modern software design approach in which an application is developed as a collection of small, independent, and loosely coupled services. Each microservice is responsible for a specific business capability, runs as an independent process, and communicates with other services using lightweight protocols such as HTTP/REST, gRPC, or message queues. So, instead of building one large application, modern systems are split into small services that do one job well and communicate with each other.

In this lab:

- One service produces messages
- One service consumes and stores messages

- RabbitMQ acts as the message broker

RabbitMQ

RabbitMQ is an open-source message broker that enables asynchronous communication between different components of an application. It acts as an intermediary that routes messages from producers to consumers by implementing the Advanced Message Queuing Protocol (AMQP).

Advantages of using RabbitMQ as a message queue system:

- 1) **Asynchronous Communication**
Send messages without waiting for an immediate response. This helps services work independently and improves overall system performance.
- 2) **Loose Coupling Between Services**
The sender and receiver do not need to know about each other. RabbitMQ sits in between, making the system easier to modify and scale.
- 3) **Reliable Message Delivery**
Messages can be stored safely in queues until they are processed. This prevents data loss even if a consumer service temporarily fails.
- 4) **Scalable Message Handling**
Multiple consumers can read from the same queue, allowing RabbitMQ to handle increased workload efficiently in cloud environments.
- 5) **Flexible Message Routing**
RabbitMQ supports different routing methods (direct, topic, fanout), making it easy to send messages to the right service based on rules.

Docker Networks

Docker automatically creates a private network for containers defined in Docker Compose. This allows:

- Containers can talk to each other using container names instead of IP addresses, simplifying configuration in cloud deployments.
- Secure internal communication (without being exposed to the public)
- Containers on different networks cannot see each other, improving security and preventing unwanted access

Docker Volumes

By default, data inside a container is lost when the container stops. Volumes store data outside containers, ensuring data is not lost when containers stop or restart. Additionally, multiple containers can access the same volume, making it useful for shared data in cloud applications.

In this lab:

- Chat history is stored in a file
- You will compare behavior with and without volumes

Prerequisites before starting the lab :

1. Ubuntu (native or WSL)
2. Docker installed
3. Docker Compose installed

Do NOT attempt this lab directly on Windows CMD or PowerShell. (This lab uses Linux-based containers, volume mounts, and networking features that behave inconsistently on Windows CMD/PowerShell). To avoid environment-related issues, use Ubuntu (native or WSL).

Step 1: Verify Docker Installation

1. Open the terminal
2. Set your SRN in the command line (you would have completed this step during the docker basics lab)

```
export SRN=PES1UG23CSXXX
```

3. Verify using -

```
echo $SRN
```

4. Modify the terminal prompt -

```
export PS1="[u@$SRN \W]\$ "
```

5. Run the Docker test container:

```
docker run hello-world
```

Paste the screenshot as SS1.

SS1

```
siddhant@fedora:~$ export SRN=PES1UG23CSXXX
siddhant@fedora:~$ echo $SRN
PES1UG23CSXXX
siddhant@fedora:~$ export PS1="[\u@$SRN \W]\$ "
[siddhant@PES1UG23CSXXX ~]$ sudo docker run hello-world
[sudo] password for siddhant:

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

[siddhant@PES1UG23CSXXX ~]$ █
```

(If you have followed exact commands for the docker installation as lab 1 , then docker compose comes built in and you can skip the installation step. But if u have older version then you have to install docker compose separately)

(Can skip if latest docker + docker desktop installed i.e, lab 1 and directly upload the verification screenshot)

Step 2: Verify Docker Compose Installation

1. Check if exists using -

```
docker-compose --version
```

Expected output - Docker Compose version <.....>

2. If not installed, proceed with the following steps -

```
curl -L "https://github.com/docker/compose/releases/download/$(curl -s https://api.github.com/repos/docker/compose/releases/latest | grep 'tag_name' | cut -d'"'"' -f4)/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

```
chmod +x /usr/local/bin/docker-compose
```

3. Verify again -

```
docker-compose --version
```

Verify Docker Compose Installation

SS2

```
[siddhant@PES1UG23CSXXX ~]$ sudo docker compose version
Docker Compose version v5.0.0-desktop.1
[siddhant@PES1UG23CSXXX ~]$
```

Step 3: Project Setup

1. Create a new directory for this lab and navigate into it -

a) `mkdir docker-advanced-lab`

b) `cd docker-advanced-lab`

2. Copy the provided project files into this directory

The directory structure should look like this -

```
[siddhant@PES1UG23CSXXX ~]$ cd docker-advanced-lab/
[siddhant@PES1UG23CSXXX docker-advanced-lab]$ ls
chat.py  docker-compose.yml  Dockerfile
[siddhant@PES1UG23CSXXX docker-advanced-lab]$
```

3. Verify with -

`ls`

Step 4: Observe Container Isolation and communication

1. To build the image **(Do no run this unless you haven't followed the prereq doc shared before)**-

`docker build -t chat-app .`

2. Running rabbitmq -

`docker run -d --name rabbitmq rabbitmq:3-management`

What is happening here?

- A RabbitMQ container is started in detached mode (-d)
- The container is given a logical name: rabbitmq

SS3

```
[siddhant@PES1UG23CSXXX docker-advanced-lab]$ sudo docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
3578195c2098   rabbitmq:3-management              "docker-entrypoint.s..." 4 minutes ago  Up 4 minutes  4369/tcp,
5671-5672/tcp, 15671-15672/tcp, 15691-15692/tcp, 25672/tcp
NAMES
rabbitmq
[siddhant@PES1UG23CSXXX docker-advanced-lab]$
```

Common issues :

In case of image pull failure : `docker pull rabbitmq:3-management`

If container exits immediately : a) `docker ps -a`

b) `docker logs rabbitmq`

Name conflict : a) `docker ps -a | grep rabbitmq`

b) `docker rm rabbitmq`

3. Run Chat container -

`docker run -it --name user-a chat-app`

This tells Docker to:

- create a new container from an image (if it doesn't exist)
- start that container immediately
- Name the container as user-a

SS4

```
[siddhant@PES1UG23CSXXX docker-advanced-lab]$ sudo docker run -it --name user-a chat-app

--- No Previous Chat History ---

--- Chat Started (unknown-user connecting to localhost) ---
RabbitMQ not ready, retrying... (1/5)RabbitMQ not ready, retrying... (1/5)

RabbitMQ not ready, retrying... (2/5)RabbitMQ not ready, retrying... (2/5)

RabbitMQ not ready, retrying... (3/5)
RabbitMQ not ready, retrying... (3/5)
RabbitMQ not ready, retrying... (4/5)
RabbitMQ not ready, retrying... (4/5)
RabbitMQ not ready, retrying... (5/5)
RabbitMQ not ready, retrying... (5/5)

ERROR: Could not connect to RabbitMQ at localhost

Falling back to standalone mode...
=====
STANDALONE MODE (No RabbitMQ)
=====

--- No Previous Chat History ---

Messages will not be sent to any other container.

Type message (standalone):
ERROR: Could not connect to RabbitMQ at localhost
i'm lonely
Sent by unknown-user (standalone): i'm lonely
Type message (standalone):
```

(Click ctrl+c to exit)

If you get an error here, that means you have done everything right so far. The application fails to connect to RabbitMQ as the containers are isolated and there is no shared network and hence, hostname `rabbitmq` cannot be resolved.

Step 5: Creating the network

Now, we will manually create a network and start the containers with the network flag.

Let's create the Network manually first:

`docker network create chat-net`

```
[siddhant@PES1UG23CSXXX docker-advanced-lab]$ sudo docker network create chat-net
08ea692d9308072e490a09ab53cd530ee7fd358c8ec060348638b4f2b8258180
[siddhant@PES1UG23CSXXX docker-advanced-lab]$
```

BEFORE THE NEXT STEP - Let's perform a quick cleanup:

`docker stop user-a rabbitmq`

`docker rm user-a rabbitmq`

Let's re-run the rabbitmq service:

`docker run -d --name rabbitmq --network chat-net rabbitmq:3-management`

```
[siddhant@PES1UG23CSXXX docker-advanced-lab]$ sudo docker run -d --name rabbitmq --network chat-net rabbitmq:3-management
[sudo] password for siddhant:
3955d9f3951596ee9808f8790b5bef9415a6c06f0d7f2f9597b26f3319dba506
[siddhant@PES1UG23CSXXX docker-advanced-lab]$
```

What is happening here?

- A RabbitMQ container is started in detached mode (-d)
- The container is explicitly attached to a user-defined Docker network called chat-net
- The container is given a logical name: rabbitmq

Because RabbitMQ is named rabbitmq and attached to chat-net, any container on the same network can reach it using just the name rabbitmq (no IP address required).

(If you get a name conflict either here or in step 4, remove the existing container using the command below -

```
docker rm -f rabbitmq
```

And rerun the command.)

Now we will run the two chat client containers **(in different terminals)** to simulate the messaging app between two different users:

Terminal 1 -

```
docker run -it --name user-a \  
  
--network chat-net \  
  
-e RABBIT_HOST=rabbitmq \  
  
-e QUEUE_NAME=queue_a \  
  
-e TARGET_QUEUE=queue_b \  
  
chat-app
```

In Terminal 2 -

```
docker run -it --name user-b \  
  
--network chat-net \  
  
-e RABBIT_HOST=rabbitmq \  
  
-e QUEUE_NAME=queue_b \  
  
-e TARGET_QUEUE=queue_a \  
  
chat-app
```

Explanation -

docker run → start a new container

-it → interactive chat client

--name user-a → logical container name

--network chat-net → attach to same Docker network

RABBIT_HOST=rabbitmq → connect to RabbitMQ container

QUEUE_NAME=queue_a → queue this user listens to

TARGET_QUEUE=queue_b → queue this user sends messages to

chat-app → image built earlier

Note : If you see an error for user-a / user-b already being in use, remove the container using -

`docker rm -f user-a user-b`

And try rerunning:)

SS5 -

```
[siddhant@PES1UG23CSXXX docker-advanced-lab]$ sudo docker run -it --name user-a \
--network chat-net \
-e RABBIT_HOST=rabbitmq \
-e QUEUE_NAME=queue_a \
-e TARGET_QUEUE=queue_b \
chat-app

--- No Previous Chat History ---

--- Chat Started (unknown-user connecting to rabbitmq) ---
Type message: hello
Sent by unknown-user: hello
Type message:
Received from unknown-user: hello friend
Type message: hope you're enjoying the lab :)
Sent by unknown-user: hope you're enjoying the lab :)
Type message:
Received from unknown-user: no it's pretty boring :(
Type message:
```

SS6 -

```
[siddhant@PES1UG23CSXXX docker-advanced-lab]$ sudo docker run -it --name user-b \
--network chat-net \
-e RABBIT_HOST=rabbitmq \
-e QUEUE_NAME=queue_b \
-e TARGET_QUEUE=queue_a \
chat-app
[sudo] password for siddhant:

--- No Previous Chat History ---

--- Chat Started (unknown-user connecting to rabbitmq) ---
Type message:
Received from unknown-user: hello
Type message: hello friend
Sent by unknown-user: hello friend
Type message:
Received from unknown-user: hope you're enjoying the lab :)
Type message: no it's pretty boring :(
Sent by unknown-user: no it's pretty boring :(
Type message:
```

(Make sure the screenshots display both the terminals running)

Note: If you get a message saying connection refused, inspect using -

`docker ps`

`docker inspect rabbitmq | grep chat-net`

Key takeaways-

- Containers do not talk to each other directly.
- They communicate through a **message broker**

Advantages -

- **Loose coupling** - Components are minimally dependent on each other so that if one service changes, crashes, or scales, it does so **without breaking** the others.
- **Asynchronous communication** - The sender does not wait for the receiver to respond.

- **Fault-tolerant design**- In case of failures, messages persist in the queue.

At this point, the system works — but it is fragile!!!

There are too many commands, manual network management, manual container startup and hard to reproduce.

Solution - Docker Compose

Step 6: Simplifying using yaml

Let's simplify this, we will use something called Docker Compose. Docker Compose is a tool for running multiple container applications and configuring them according to our wishes with a single yaml file. The above painful task will be extremely simple once you see the magic of docker compose.

BEFORE THE NEXT STEP - Let's perform a quick cleanup:

```
docker stop user-a user-b rabbitmq
```

```
docker rm user-a user-b rabbitmq
```

```
docker network rm chat-net
```

SS7

```
[siddhant@PES1UG23CSXXX docker-advanced-lab]$ sudo docker stop user-a user-b rabbitmq
user-a
user-b
rabbitmq
[siddhant@PES1UG23CSXXX docker-advanced-lab]$ sudo docker rm user-a user-b rabbitmq
user-a
user-b
rabbitmq
[siddhant@PES1UG23CSXXX docker-advanced-lab]$ sudo docker network rm chat-net
chat-net
[siddhant@PES1UG23CSXXX docker-advanced-lab]$ █
```

We already have the yaml file in the folder.

YAML as a system blueprint

The YAML file describes:

- Containers
- Networks
- Environment variables
- Relationships

Instead of *remembering commands*, we **declare the system**.

All we have to do is hop in the directory and use the docker compose command to start all the three services. Run the below commands in three different terminals.

In terminal 1:

```
docker compose up rabbitmq
```

SS8

```
ment_metrics_collection = true"
rabbitmq-1 | 2026-02-02 16:13:33.082091+00:00 [warning] <0.725.0> To test RabbitMQ as if the feature was
removed, set this in your configuration:
rabbitmq-1 | 2026-02-02 16:13:33.082091+00:00 [warning] <0.725.0>      "deprecated_features.permit.manage
ment_metrics_collection = false"
rabbitmq-1 | 2026-02-02 16:13:34.108233+00:00 [info] <0.762.0> Management plugin: HTTP (non-TLS) listen
er started on port 15672
rabbitmq-1 | 2026-02-02 16:13:34.108368+00:00 [info] <0.792.0> Statistics database started.
rabbitmq-1 | 2026-02-02 16:13:34.108427+00:00 [info] <0.791.0> Starting worker pool 'management_worker_p
ool' with 3 processes in it
rabbitmq-1 | 2026-02-02 16:13:34.114720+00:00 [info] <0.810.0> Prometheus metrics: HTTP (non-TLS) listen
er started on port 15692
rabbitmq-1 | 2026-02-02 16:13:34.114844+00:00 [info] <0.696.0> Ready to start client connection listener
s
rabbitmq-1 | 2026-02-02 16:13:34.115636+00:00 [info] <0.854.0> started TCP listener on [::]:5672
rabbitmq-1 | 2026-02-02 16:13:34.144507+00:00 [info] <0.696.0> Server startup complete; 5 plugins starte
d.
rabbitmq-1 | 2026-02-02 16:13:34.144507+00:00 [info] <0.696.0> * rabbitmq_prometheus
rabbitmq-1 | 2026-02-02 16:13:34.144507+00:00 [info] <0.696.0> * rabbitmq_federation
rabbitmq-1 | 2026-02-02 16:13:34.144507+00:00 [info] <0.696.0> * rabbitmq_management
rabbitmq-1 | 2026-02-02 16:13:34.144507+00:00 [info] <0.696.0> * rabbitmq_management_agent
rabbitmq-1 | 2026-02-02 16:13:34.144507+00:00 [info] <0.696.0> * rabbitmq_web_dispatch
rabbitmq-1 | completed with 5 plugins.
rabbitmq-1 | 2026-02-02 16:13:34.217997+00:00 [info] <0.9.0> Time to start RabbitMQ: 3254 ms
█
w Enable Watch  d Detach
```

In terminal 2:

cd docker-advanced-lab

docker compose run --rm user-a

SS9

```
[siddhant@PES1UG23CSXXX docker-advanced-lab]$ sudo docker compose run --rm user-a
[sudo] password for siddhant:
WARN[0000] /home/siddhant/docker-advanced-lab/docker-compose.yml: the attribute `version` is obsolete, it
will be ignored, please remove it to avoid potential confusion
[+] 2/2t 2/2
✓ Volume docker-advanced-lab_chat-history Created 0.0s
✓ Container docker-advanced-lab-rabbitmq-1 Running 0.0s
Image docker-advanced-lab-user-a Building
[+] Building 3.1s (12/12) FINISHED
=> [internal] load local bake definitions 0.0s
=> => reading from stdin 544B 0.0s
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 382B 0.0s
=> [internal] load metadata for docker.io/library/python:3.9-slim 2.6s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [1/5] FROM docker.io/library/python:3.9-slim@sha256:2d97f6910b16bd338d3060f261f53f144965f75559 0.0s
=> => resolve docker.io/library/python:3.9-slim@sha256:2d97f6910b16bd338d3060f261f53f144965f75559 0.0s
=> [internal] load build context 0.0s
=> => transferring context: 88B 0.0s
=> CACHED [2/5] RUN pip install pika 0.0s
=> CACHED [3/5] WORKDIR /app 0.0s
=> CACHED [4/5] COPY chat.py . 0.0s
=> CACHED [5/5] RUN mkdir -p /app/data 0.0s
=> exporting to image 0.2s
=> => exporting layers 0.0s
=> => exporting manifest sha256:aa92a324db0eead1779a8266eddaad7c81494bc24bd383e464617d3750739bd7 0.0s
=> => exporting config sha256:cef32b922d35f9ccd4b3237ee964e0035b0ef92fc5234b16450358712d57bd82 0.0s
=> => exporting attestation manifest sha256:5aeba71dfc991151c2326be5d7124946db46b3bb4231a4fef1a10 0.0s
=> => exporting manifest list sha256:5213b877c81920ede2c17bb9c56463c6085f600836d151ca0a24c428a518 0.0s
=> => naming to docker.io/library/docker-advanced-lab-user-a:latest 0.0s
=> => unpacking to docker.io/library/docker-advanced-lab-user-a:latest 0.0s
=> resolving provenance for metadata file 0.0s
Image docker-advanced-lab-user-a Built
Container docker-advanced-lab-user-a-run-8b9f0048e755 Creating
Container docker-advanced-lab-user-a-run-8b9f0048e755 Created

--- No Previous Chat History ---

--- Chat Started (user-a connecting to rabbitmq) ---
Type message:
Received from user-b: hello
Type message: wazzzaaaaaa
Sent by user-a: wazzzaaaaaa
Type message: █
```

In terminal 3:

```
cd docker-advanced-lab
```

```
docker compose run --rm user-b
```

SS10

```
[siddhant@PES1UG23CSXXX docker-advanced-lab]$ sudo docker compose run --rm user-b
[sudo] password for siddhant:
WARN[0000] /home/siddhant/docker-advanced-lab/docker-compose.yml: the attribute `version` is obsolete, it
will be ignored, please remove it to avoid potential confusion
[+] 1/1t 1/1
✓ Container docker-advanced-lab-rabbitmq-1 Running 0.0s
Image docker-advanced-lab-user-b Building
[+] Building 1.0s (12/12) FINISHED
=> [internal] load local bake definitions 0.0s
=> => reading from stdin 544B 0.0s
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 382B 0.0s
=> [internal] load metadata for docker.io/library/python:3.9-slim 0.5s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [1/5] FROM docker.io/library/python:3.9-slim@sha256:2d97f6910b16bd338d3060f261f53f144965f75559 0.0s
=> => resolve docker.io/library/python:3.9-slim@sha256:2d97f6910b16bd338d3060f261f53f144965f75559 0.0s
=> [internal] load build context 0.0s
=> => transferring context: 88B 0.0s
=> CACHED [2/5] RUN pip install pika 0.0s
=> CACHED [3/5] WORKDIR /app 0.0s
=> CACHED [4/5] COPY chat.py . 0.0s
=> CACHED [5/5] RUN mkdir -p /app/data 0.0s
=> exporting to image 0.2s
=> => exporting layers 0.0s
=> => exporting manifest sha256:267be98e2ca3443e5a9ac4aeffe3ddefff0ed4a257c5da81c972b4c2d1fb42b4 0.0s
=> => exporting config sha256:f2056ba6479fa762f54c9af8f9905b231bebf2a24c7dfd49fd94b25e81e9615b 0.0s
=> => exporting attestation manifest sha256:077ad80b42dd3496ddc7d2913f920c3f5d0de7a3c098b21d71e7d 0.0s
=> => exporting manifest list sha256:980d3c7ea7cfe585366f11d9d508279b808f9ea5ef3783f0d753a79dc4e9 0.0s
=> => naming to docker.io/library/docker-advanced-lab-user-b:latest 0.0s
=> => unpacking to docker.io/library/docker-advanced-lab-user-b:latest 0.0s
=> resolving provenance for metadata file 0.0s
Image docker-advanced-lab-user-b Built
Container docker-advanced-lab-user-b-run-d7765df8fc7f Creating
Container docker-advanced-lab-user-b-run-d7765df8fc7f Created

--- No Previous Chat History ---

--- Chat Started (user-b connecting to rabbitmq) ---
Type message: hello
Sent by user-b: hello
Type message:
Received from user-a: wazzzaaaaaa
Type message:
```

Woah! Was that not very very simple?

Step 7: Verify chat history

Have you noticed the chat history section which has been blank all this while. We will fill that up now. Docker offers something called docker volumes, which allows containers to read and write to mounted filesystems.

In the given docker-compose.yml file, **uncomment the three sections where we configure volumes(read and understand)** and run the entire system again in three terminals like before. Send a few messages and kill the system. Run the same set of commands again and you will see the chat history updated!

Terminal 1:

```
docker-compose up rabbitmq
```

Terminal 2:

```
cd docker-advanced-lab
```

```
docker compose run --rm user-a
```

Terminal 3:

```
cd docker-advanced-lab
```

```
docker compose run --rm user-b
```

Kill all the processes, and run them again.

SS11

```
[siddhant@PES1UG23CSXXX docker-advanced-lab]$ sudo docker compose run --rm user-a
WARN[0000] /home/siddhant/docker-advanced-lab/docker-compose.yml: the attribute `version` is obsolete, it
will be ignored, please remove it to avoid potential confusion
[+] 1/1t 1/1
  ✓ Container docker-advanced-lab-rabbitmq-1 Running 0.0s
Container docker-advanced-lab-user-a-run-05caed9e33e2 Creating
Container docker-advanced-lab-user-a-run-05caed9e33e2 Created

--- Chat History ---
Sent by user-b: hello
Received from user-b: hello
Sent by user-a: wazzzaaaaaa
Received from user-a: wazzzaaaaaa
--- End of History ---

--- Chat Started (user-a connecting to rabbitmq) ---
Type message:
Received from user-b: wow
Type message: we have the histor saved, docker volumes are so fun
Sent by user-a: we have the histor saved, docker volumes are so fun
Type message:
```

SS12

```
[siddhant@PES1UG23CSXXX docker-advanced-lab]$ sudo docker compose run --rm user-b
WARN[0000] /home/siddhant/docker-advanced-lab/docker-compose.yml: the attribute `version` is obsolete, it
will be ignored, please remove it to avoid potential confusion
[+] 1/1t 1/1
✓ Container docker-advanced-lab-rabbitmq-1 Running 0.0s
Container docker-advanced-lab-user-b-run-4685a74eb8e2 Creating
Container docker-advanced-lab-user-b-run-4685a74eb8e2 Created

--- Chat History ---
Sent by user-b: hello
Received from user-b: hello
Sent by user-a: wazzzaaaaaa
Received from user-a: wazzzaaaaaa
--- End of History ---

--- Chat Started (user-b connecting to rabbitmq) ---
Type message: wow
Sent by user-b: wow
Type message:
Received from user-a: we have the histor saved, docker volumes are so fun
Type message:
```

Key takeaway -

Containers are disposable (they can be stopped / restarted) and the data inside containers is not persistent by default. By using volumes - data remains intact regardless of what happens to the containers.

Cleanup steps -

docker compose down

docker compose down -v

docker rm -f user-a user-b rabbitmq

docker network rm chat-net

docker rm chat-app

Conclusion

Docker networks solve communication.

Docker Compose solves orchestration.

Docker volumes solve persistence.