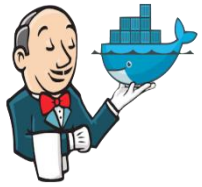

Debugging Docker Networking in Jenkins Pipelines

While working on a **Java-based microservices project** recently, I came across an interesting issue during deployment with **Docker**.

Issue:

All the containers were up and running smoothly but strangely, none of them were responding on their respective ports. Instead, they were bound only to **localhost (127.0.0.1)**. This meant that the microservices couldn't communicate with each other even though they were meant to be on the same Docker network.

Root Cause:

After some investigation and research, I found the real reason behind it: The issue turned out to be a **missing Docker network configuration** inside my Jenkins pipeline configuration.

- Because of that, each container was running in isolation instead of being attached to the shared network.

The Solution:

Once I updated the Jenkinsfile and included the correct Docker network configuration, everything started working flawlessly.!

This experience reminded me that sometimes, it's not about complex bugs it's about small configuration details that can make a huge impact.

Key Docker Networking:

- Always define a custom Docker network
- Ensure all containers share the same network for internal communication.
- If containers listen only on 127.0.0.1, verify they're properly exposed.
- Small misconfigurations in CI/CD pipelines can cause major deployment issues always double-check your YAML or Jenkinsfile.