Assignment 3.1 - Container Visualization

Katerina Chinnappan, Rohit Shaw, Kjell Zijlemaker

Docker is quickly gaining huge popularity in the Cloud ecosystem as it offers a plethora of developer-friendly advantages over Virtual Machines – fast, portable, lightweight, memory efficient, ease of deployment and use. The ability to package a piece of software along with all its dependencies, deploy it in any computing environment, and achieve the same results, makes it even more attractive. The prime reason why everyone is embracing Docker over VM is because they use shared operating systems, allowing for efficient resource management. As a result, a well-configured container system could potentially allow for 4 or more times the number of application instances as that of a VM with the same hardware.

In this assignment, hands-on experience with Docker was gained. Two Docker containers were contextualized to package and deploy the previously-developed URL Shortener web service. The service was made available to other containers on the same/remote hosts by exposing ports. Bandwidth test, message broadcasting, and remote communication between the two containers were also performed.

Implementation

URL Shortener service

The first step to creating a Docker container is creating a `Dockerfile` containing all the necessary commands needed to set up the dependencies and execute. After creating the `Dockerfile`, the `host` and `port` were set in `REST.py` and `index.html` to make them visible from other containers. Then the Docker image was created and run, deploying the URL Shortener service accessible on `http://localhost:4000`.

Similarly, another container with the exact same contents, with the `port` set to `81` was packaged and deployed. A detailed step-by-step guide with all commands to set up and use the service can be found in `README.md`.

Testing bandwidth and local communication

To test the bandwidth and broadcasting of messages between the two containers, commands to install `iperf`, `socat`, and `netcat` were written in both the `Dockerfiles`.

The bandwidth was tested by running the command `iperf3 -s` on the server (first container) and `iperf3 -c` *`ip_addr`* on the client (second container).

To test local communication between the containers, the command `socat tcp-l:7777, reuseaddr, fork system:'echo hello; sleep 5; echo goodbye', nofork` was run on the server container and `socat tcp:`*`ip_addr`*`:7777 -` on the client container.

Similarly, to test the communication using `netcat`, `nc -l -p` *`port`* was run on the server container, after providing a *`port`*, and `nc` *`ip_addr port`* on the client, after providing the server *`ip_addr`* and *`port`*.

Remote container communication

To perform this, the two containers were deployed on different computing environments - one on the local machine and the other on a Linux VM on DAS-4. The two containers were then connected via a bridge network set up on *ZeroTier*, enabling them to communicate with each other using the IPs allocated to them by the *ZeroTier* network.

Conclusion

In conclusion, our experience with Docker went very smooth. All of us were new to containers and it was fun to learn a new and useful technology. Docker is very useful when wanting to completely isolate software from a big environment and give it access to only the parts of the environment it needs. Docker has an advantage over virtual machines and we can see this difference between this assignment and assignment 2.