Sharif University of Technology
Department of Electrical Engineering

Cloud-native Software-Defined Mobile Networks

Assignment 2

Containers and Docker

Before you start:

In this homework you are going to become familiar with containers, their networking and Docker. One of the most important skills in the DevOps world is working with git. So you should provide your solutions to this homework in a git repository (github.com, gitlab.com, hamgit.ir, etc.). Every problem has its own directory in the git repository that includes your files. Any documents must also be included in the repository in a text file (it is highly recommended to use markdown format). You can get up to 10 extra points for having clean and comprehensive commits that show the steps you took for the problems. You need to submit a file that only contains a link to your git repository. There should be no commits after the deadline in your repository.

Problem 1 (Container Networking) *45 points*

Use Linux network namespaces and create the topology in the following figure:

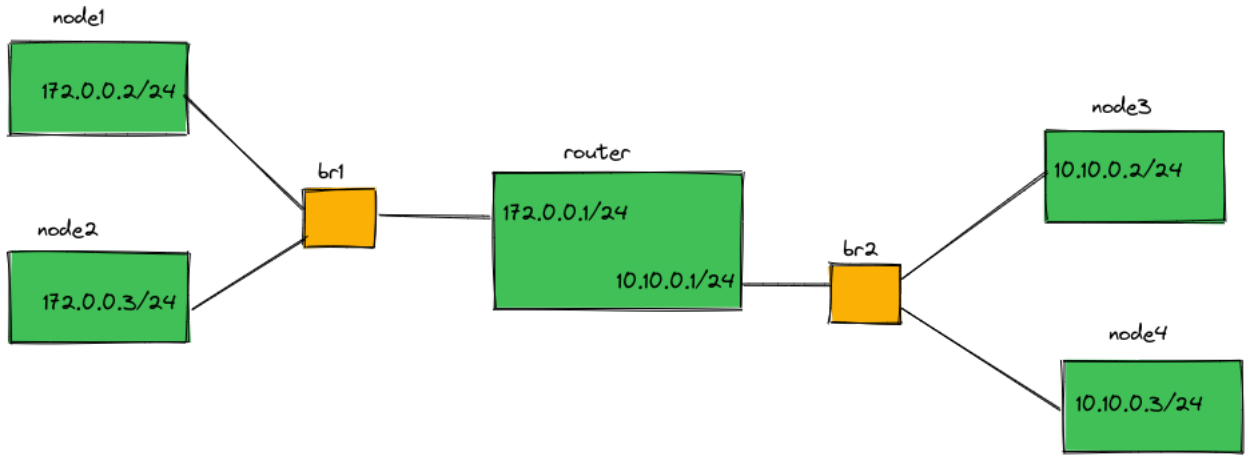

Figure 1: Network namespace topology with a router

Each of the nodes and the router are in a different network namespace (not the root namespace). The bridges are in the root network namespace. You should provide a bash script that creates the topology. When implementing your solution consider these:

- Every node must be able to ping every other node and the router.
- The 172.0.0.0/24 subnet can reach 10.10.1.0/24 only via the router and vice versa. So the router is the default gateway for all of the nodes.

You should provide another script that takes two node names as parameters and starts pinging the second node from the first one (the nodes are shown in green in figure 1). Suppose we run this command:

```
./your-script.sh node1 router
```

After running it node1 should start pinging the router and the output must be printed in the command line.

Now delete the router and its links to the bridges (Look at figure 2). How can we route packets from one subnet to another? Explain your solution (Including the rules in the root namespace. No implementation is required.).

What if the namespaces are on different servers (virtual machine or physical server) that can see each other in layer 2 (Look at figure 3)? Explain your solution (Including the rules on the servers. No implementation is required.).

Figure 2: Network namespace topology without a router

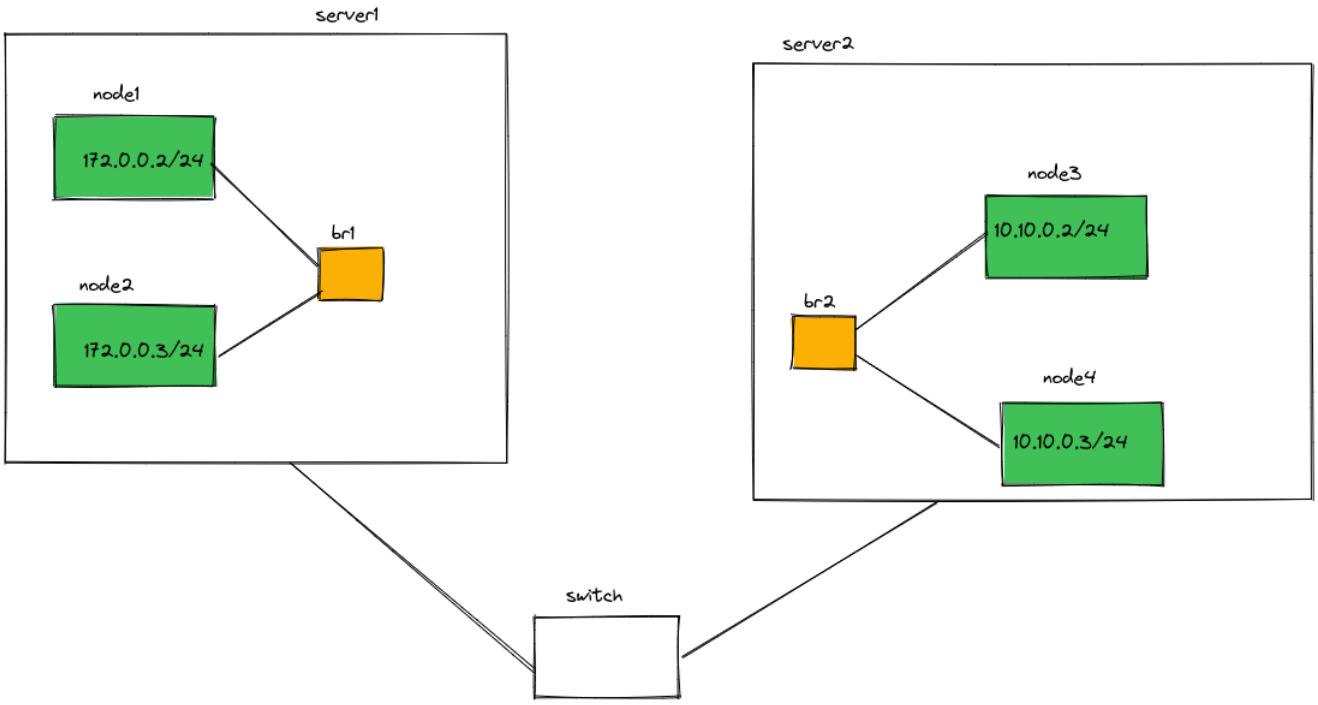

Figure 3: Network namespace topology

Deliverables:

- A bash script that creates the topology in figure 1 and another for pinging nodes.
- Two documents explaining your answers to questions regarding figures 2 and 3.

Problem 2 (Container Runtime) *35 + 10 points*

In this problem you are going to implement a container runtime (a very simple version of Docker). You can use any language to do this (Python, Bash, Golang, etc.). Your container runtime is a CLI that takes a parameter, hostname, and creates the container. After running your CLI like this:

```
your-cli myhostname
```

We should enter in the bash that exists in new namespaces and has myhostname as its hostname. Also by running

```
ps fax
```

in the container we should see the bash process as PID 1. Your container runtime must have these features:

- Creates these new namespaces for the container: `net`, `mnt`, `pid`, `uts`
- The container has a filesystem other than the host's root and its root is a directory that has contents of the Ubuntu 20.04 filesystem (like `ubuntu:20.04` image on dockerhub). Each container has its own separate root filesystem on the host.
- [BONUS] The CLI has a second optional argument in megabytes that limits memory usage of the container.

Deliverables:

- The files needed for your CLI to run.
- A document named `README.md` explaining how to run the CLI.

Problem 3 (Docker) *20 points*

This problem is about Docker. In this problem you write a simple HTTP server and dockerize it. You can use your language of choice for this problem.
Your HTTP server has only one endpoint: `/api/v1/status`. This endpoint must handle GET and POST HTTP methods. When the request method is GET, the server sends this JSON response:

```
{ "status": "OK" }
```

with status code 200. When the request method is POST with a body like this:

```
{ "status": "not OK" }
```

The server returns the body:

```
{ "status": "not OK" }
```

with status code 201. From now on every other GET request is responded with `"not OK"` status until another POST request changes it to something else. Your server listens on port 8000.

After implementing the web server, write a `Dockerfile` and build an image for your server. You should be able to create a container from that image and publish its port on your host.

Deliverables:

- The code of your HTTP server.
- A `Dockerfile` to build a docker image from your code.