INR Lab Assignment
Linux containers and IPv6 *

Mick Pouw Arno Bakker
(Arno.Bakker@os3.nl)

Feedback deadline:
November 17, 2017 10:00 CET

Abstract

This lab session is split into two parts. In the first part you will deploy Linux containers using LXD and will get familiar with its concepts. In the second part you will use LXD and a tool called `pogo` to build a small IPv6 network for which you will enable IPv6 auto-configuration.

In the following lab exercises you will be using a lot of throw-away Linux containers. This means that data is lost as soon as you destroy the instances. Keep careful note of what you do and where, so that you document your progress.

Creating the LXD Xen image

Task 1. Create a new Xen image as per Lab 1, running Ubuntu **17.04 Zesty**. It must run Zesty, as pogo is currently not available on a lower version. Make sure your virtual machine can be reached using `ssh` by installing the relevant packages. Be generous with the amount of RAM (4GB+). We will configure this image to run `pogo`, and we will use this same image for future INR lab exercises. *Hint: Extract the zesty configuration from the xen-tools_4.7-1 package. The pogo gitlab page describes this.*[1]

Task 2. Once the machine is up and running, install the `screen` package and use **ssh-copy-id** to enable ssh key login to this machine from your workstation. Start a screen session. Try to get familiarized with the screen keyboard commands. What does `screen -ls` do ? *Hint: `http://www.linuxjournal.com/article/6340` is a good reference for screen beginners.*

Creating a container

It is very easy to spin up a container using the LXC toolset. To build the base container you will use a helper script, which is part of **pogo**, a tool for virtual networks.

Read about LXD on `https://lxd.readthedocs.io/en/stable-2.0/` and `https://help.ubuntu.com/lts/serverguide/lxd.html`. It might be useful to find a cheat sheet for the lxc tool.

Task 3. On the virtual machine also install the `lxd` package, this will install a current lxd environment. Also install the `net-tools`, `tcpdump` and `openvswitch-switch` packages.

*Based on earlier work by C. Dumitru and Jeroen van der Ham. Version November 7, 2017.

[1] https://gitlab.os3.nl/Networking/pogo.git

Task 4. Install `git` and `python` and clone the git repository from `https://gitlab.os3.nl/Networking/pogo.git`. In the directory you will find the `create-base-container.sh` script that upon execution will create a container named `ogobase`. Open the `create-base-container.sh` with a text editor and try to understand what it does. Execute the script as root. Once the script finishes check that the container has been created using the lxc command. *Do not dive into the rest of the scripts for now!*

Task 5. a. Clone the basecontainer using the `lxc` toolset and name this **lx1**. Start this the container and check that everything works.

b. Stop and delete the **lx1** container using the `lxc` command from the host machine.

c. Investigate what other functionalities `lxc` provides, and briefly describe each function in your log.

Setting up a simple IPv4 network

At this point you can clone Linux containers. It is time to to connect them using a virtual network. The `openvswitch-switch` package can provide network connectivity between the containers.

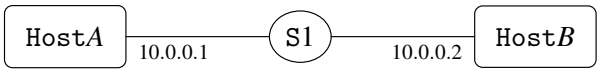

Figure 1: A simple network

Task 6. What nictypes are available for Linux containers and how do they work?

Task 7. Investigate `openvswitch-switch` and then set up the simple network depicted in Figure 1 (S1 is a openvswitch switch), so that both instances can ping each other via IPv4 successfully. The hosts can be created by cloning the base container.

Hint: Use `ovs-vsctl` to create a bridge. You can connect your virtual machines to this switch by executing e.g.
`lxc config device add` {HostA} `eth66 nic nictype=bridged parent=`{S1}
Remove the switch and cloned containers afterwards.

IPv6

Figure 2: A simple IPv6 network

For the IPv6 section you will use `pogo`. pogo creates and manages virtual topologies. It hides all the scripting required to create and start the linux containers.

Task 8. Using the `configurations/other/simple.cfg` and `configurations/other/bridge.cfg` as guidelines, create a config file that replicates the network in Figure 2. Because you will not use any dynamic routing protocol and no bridges are involved, set all the roles of the devices to *host*. For the IP addressing use the following IP blocks:

- IPv4 - `a.b.0.0/17` - where $a = length(first\_name)$ and $b = length(last\_name)$. Configure the first /23 subnet on the A-R segment and the 42nd /23 subnet on the R-B segment.
- IPv6 - `2001:0db8:0x00:0y00::1/58` - where

$$x = (\sum_{i=1}^{a} int(first\_name[i])) \bmod 16 \tag{1}$$

and

$$y = (\sum_{i=1}^{b} int(last\_name[i])) \bmod 16 \tag{2}$$

`int('a')` is the ASCII value of a, 97. In simple English: convert the letters of your name to ASCII values,sum them up and get remainder of modulo 16 division.

For the network segment A-R you should use the first /64 block of the /58 while for R-B you should use the 42nd /64 block. At first configure IPs manually. You will switch to auto configuration later.

Make sure you write down in your logs the values of a, b, x and y! Also create a simple network diagram depicting the subnets and the IPs configured on each host interface.

Task 9. Create the network environment by using `python pogo.py create config.cfg`.

Task 10. Bring up the network using `python pogo.py start config.cfg` and do the following:

- Inspect the IP configuration (addresses, routing table) for all `A,B,R` (IPv4 and IPv6)
- Check connectivity between A-R, B-R, A-B over IPv4 and IPv6. For IPv6 use **both the link local and the global addresses**.
- Add IPv4 and IPv6 static routes on A and B such that there is connectivity between the two. Show A can reach B via IPv4 and IPv6 *Hint: do not forget to enable routing on R!*

Auto configuration

Task 11. Stop and start your network using **pogo**.

Task 12. On `R` configure and enable the `radvd` daemon, using the IP blocks mentioned in the previous task. Explain all the configuration parameters used. As with the previous task, inspect the IP configuration, do a connectivity check and explain the differences.

Task 13. Explain how the IPv6 address received by host A was derived.

Task 14. Why wasn't it necessary to manually add routes ?

Task 15. Stop the `radvd` service and see if the network breaks. Explain why.

Analysis

Task 16. Check the tcpdump path on the host system. It should contain pcap files of the last pogo run. These dumps should contain all the relevant packets regarding auto-configuration.

Task 17. Explain the auto-negotiation process that takes place over the A-R segment using the packet trace as supporting material. Decode and explain all the interesting packets.

Task 18. Bonus Remove all the IP addressing from the network config file. Install and configure `avahi` for IPv6 on `R` and hosts, such that the SSH service on A is accessible from B by just by typing `ssh hostA.local`. Do not use any `/etc/hosts`! You will need to build a new base container for this (and cannot use IPv4).

Task 19. Bonus Fork the network script on Gitlab and add any extra functionality that you find useful (eg. sanity check for config files, easier config syntax, status check for running networks, etc.)