Using Linux Containers in VIRL

version 20160428

VIRL Developers

April 18, 2016

Contents

Using Linux Containers in VIRL.

Introduction

This document should explain the integration of Linux Containers (LXCs) into VIRL simulations in sufficient detail to allow users to create, manage and share their own LXC subtypes, images and templates.

Linux Containers can be used to enhance topologies, using a relatively lightweight yet flexible 'host' node. One must bear in mind that whereas virtual machine-based host nodes have the full functionality of the operating system running within the virtual machine, LXCs usually run just a single or a handful of applications.

Information on using the applications within the LXC is not the purpose of this guide.

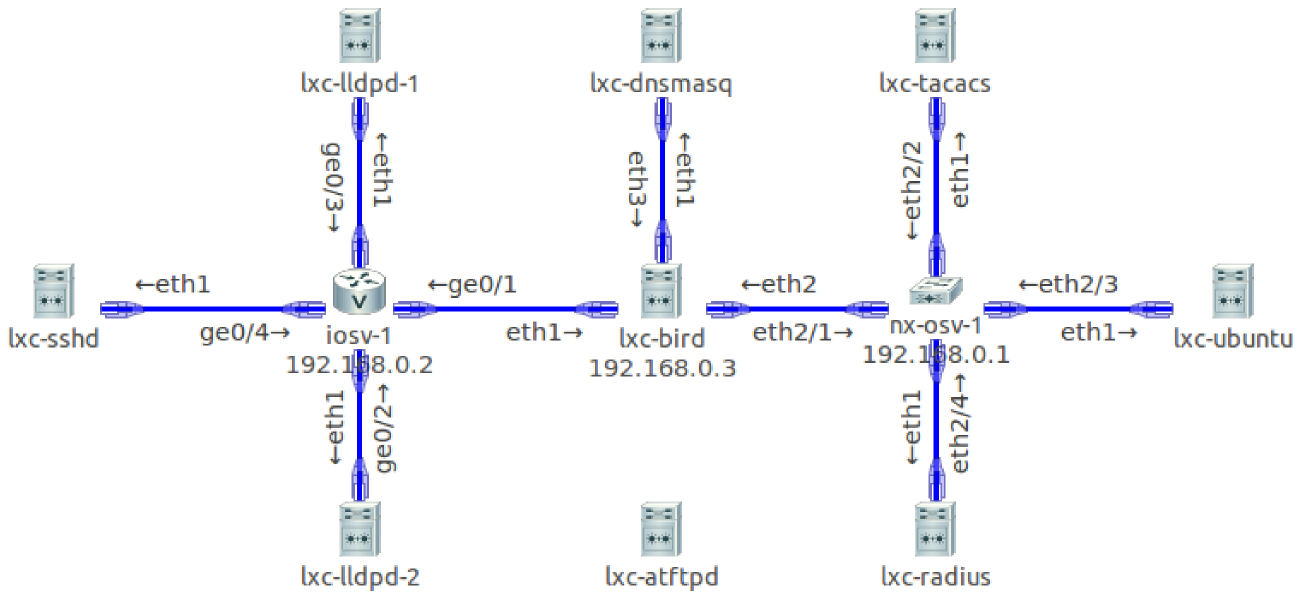
This document is intended to provide examples of how to create containerized applications that can be 'packaged' within an LXC. The configurations for the example programs covered in this guide are samples at best. Users should use the documentation from the target applications in order to learn about their use.

This guide is meant to show the features supported by the VIRL releases as of April 2016, with VIRL-CORE version 0.10.24.x and above. Most of the functionality has been present in earlier releases as well, such as 0.10.21.x .

In this guide, we'll create a topology with 8 different LXC types:

- The lxc with an Ubuntu 'cloud' distribution installed which is usable as a virtual machine 'server' replacement
- The lxc-sshd - possibly the smallest node you can log into
- The lxc-dnsmasq - shows how additional commands can be executed
- The lxc-lldpd - shows how simple applications can be packaged
- The lxc-atftpd - shows how template arguments can be used
- The lxc-tacacs - provides an example of a more complex application
- The lxc-radius - provides an example of working with an application that is not part of the Ubuntu apt-get package repository system, as well as support for image sharing - The lxc-bird - an example of a routing stack application operating within an LXC

The topology built in this document is shown below:



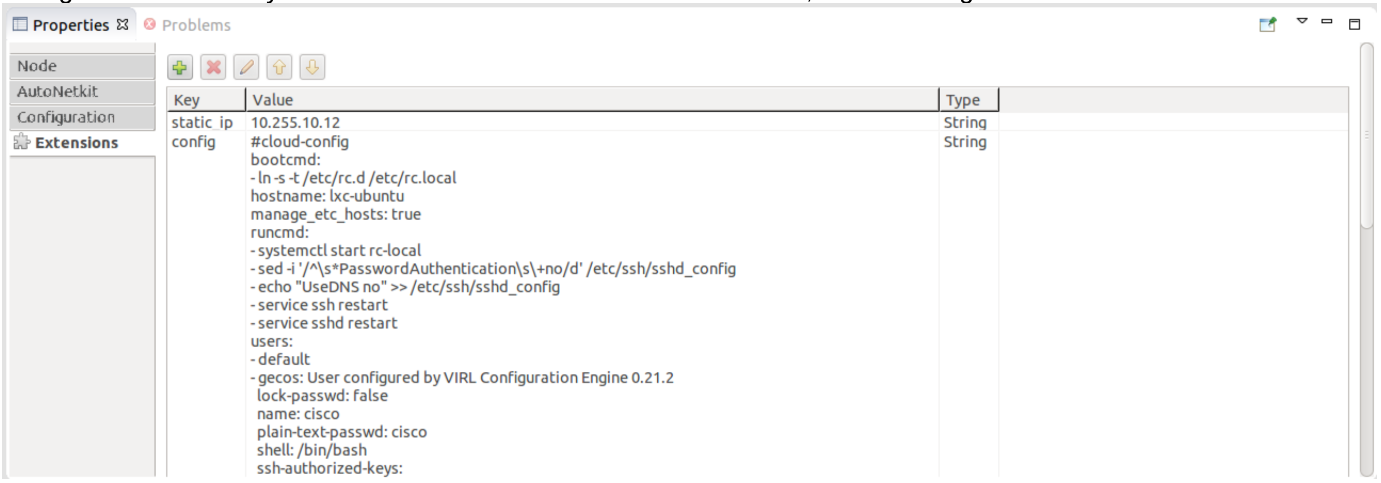
The topology can be designed by using the lxc-sshd subtype which is available from the VM Maestro (VMM) palette, for every LXC node except for 'lxc-ubuntu', which is the 'standard' lxc subtype. The subtypes can be altered at any time.

The node names used are purely for clarity in this example topology.

After the topology is drawn, it can be run through the configuration engine, AutoNetkit (ANK).

> ***Note***
>
> ANK has a set of subtypes it recognizes and can create configuration entries for. It will treat each unknown subtype as if it was a regular host (i.e. the server VM subtype), but the initial configuration will not be generated for that node. A future version will likely support and recognize all subtypes starting with the prefix 'lxc-', and generate the same configuration as for any LXC subtype known to it.

The initial configuration created by ANK for each of the recognized node subtypes will be used when a given node is started. The configuration can be edited in VMM in the node properties tab when the node is selected. The configuration is actually stored in the .VIRL file as a node 'extension', called 'config'.



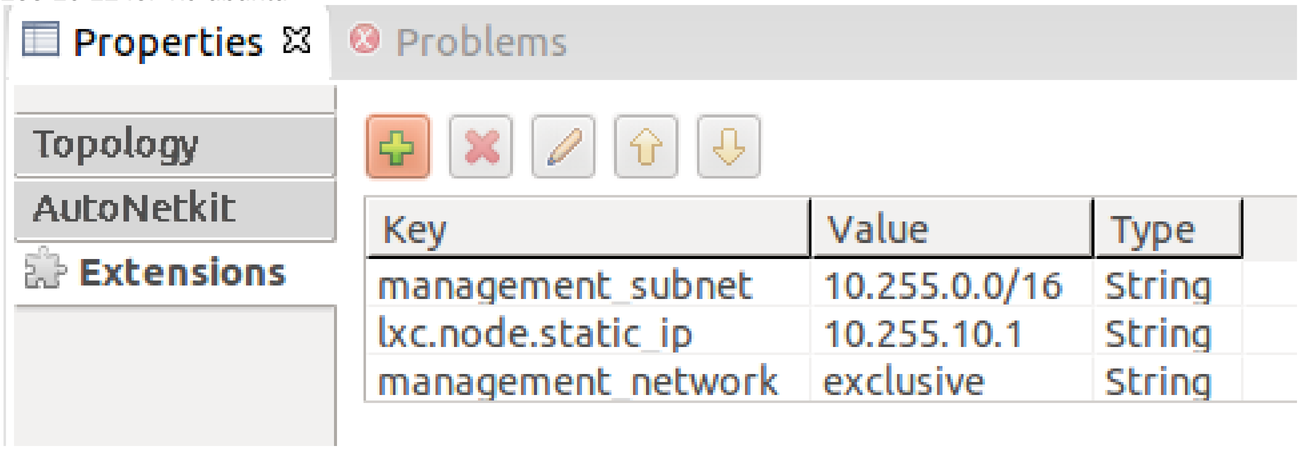
Key	Value	Type
static_ip	10.255.10.12	String
config	(see below)	String

```
#cloud-config
bootcmd:
- ln -s -t /etc/rc.d /etc/rc.local
hostname: lxc-ubuntu
manage_etc_hosts: true
runcmd:
- systemctl start rc-local
- sed -i '/^\s*PasswordAuthentication\s\+no/d' /etc/ssh/sshd_config
- echo "UseDNS no" >> /etc/ssh/sshd_config
- service ssh restart
- service sshd restart
users:
- default
- gecos: User configured by VIRL Configuration Engine 0.21.2
  lock-passwd: false
  name: cisco
  plain-text-passwd: cisco
  shell: /bin/bash
  ssh-authorized-keys:
```

As seen in the image above, other extensions can be added to the nodes. An example used in the demo topology is 'static_ip', which is used to assign an IP address on the Out-of-band management network to each node.

The static IP address must be unique and must be valid for the subnet for the given network management network (e.g. 10.255.0.0/16 used in the demo). If static IPs are used, care must be taken if starting multiple simulations (IP address clash) or when using the 'Shared flat network' for management.

The example topology uses a 'private simulation network' for the Out-of-band management network, where a custom subnet can be set, as shown below. All nodes in the example topology are given a static IP address, starting with 10.255.10.1 for the management LXC for the simulation, then continuing left-to-right and top-to-bottom, ending with 10.255.10.12 for lxc-ubuntu.



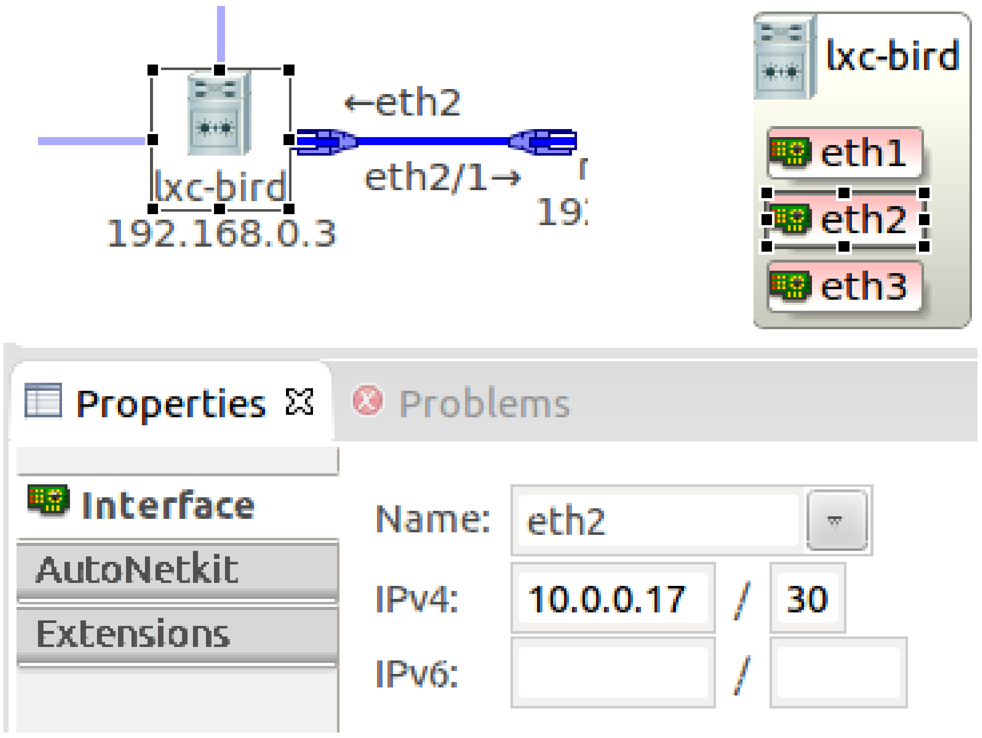
Key	Value	Type
management_subnet	10.255.0.0/16	String
lxc.node.static_ip	10.255.10.1	String
management_network	exclusive	String

Another piece of information added to the topology by ANK, besides the initial 'config', are the individual IP addresses assigned to each network interface.

They are significant in LXCs, as they are used as the default IP addresses even if the ANK-generated initial node configuration is not present.

The addresses can be viewed and edited in VM Maestro Interface properties by selecting the interface in the Node editor window of the Design view.

The user has complete control over the subnet and address allocation of 'internal' links (those between nodes in the topology), whereas for external and management networks, different rules apply. The IPs need to match the subnet for the network, and are reserved across topologies. The 'static_ip' extensions can be used to force a given IP; a semi-random IP is selected each time the node starts otherwise.



Name: eth2
IPv4: 10.0.0.17 / 30
IPv6: /

> **Note**
>
> ANK will replace the configuration of each node each time the 'Build initial configurations' button is pressed unless you disable the 'Auto-generate' field in the AutoNetkit tab. All modifications to the configuration will be lost.

> **Note**
>
> When IP address assignments are changed in the initial configuration of any node, the changes won't be read back into the individual interfaces. It's better to also set them to match, or to clear both address and prefix fields.

The subtype lxc as an alternative server

The first LXC subtype supported by VIRL for design in topologies is the 'lxc' node, with the same intent and feature support as the VM-based alternative, 'server'.

Comparison to other node subtypes

The 'lxc' subtype runs a full distribution of a Linux operating system and is intended to provide a drop-in replacement for the 'server' virtual machine node type. The LXC image contains a full file system, augmented by the LXC template. The other LXCs on the other hand, attach themselves to the host's file system, meaning that only files not present on the host need to be added to the LXC, resulting in smaller LXC packages and a more light-weight resource requirements overall.

The lxc subtype, being a 'full' system, is largely independent of its host system, with the exception of the running Linux kernel, which it shares with the host system, in a separated-out manner called 'namespaces'. Hence, to the LXC node, it appears as if it is in its own world, whereas the host system sees all processes and resources the LXC consumes.

This independence means that Linux distributions other than that of the underlying VIRL host system can be deployed as LXCs. The mechanisms which are used to achieve this are beyond the scope of this document.

Another consequence of the independence of the full distribution 'lxc' subtype is that things like package management and administrative access through sudo are supported on the 'lxc', where they are included in and contained by the separate file system. The other LXCs should not (need to) have such mechanisms available, as they are geared towards running a single application or two, prepared in advance.

The lxc subtype also has a full system initialization mechanism, such as Upstart or SystemD, which means many services can be started within the container in the same way they do on a regular Linux system.

Having such flexibility also has drawbacks - the full LXC consumes more storage and memory and is much slower to start up than smaller LXCs.

The 'server' virtual machine nodes managed by VIRL (using OpenStack) can share disk image contents, meaning that only the differences to the original image have to be stored for each node. As of the April release, VIRL supports use of overlayfs to have a similar image sharing arrangement.

As each LXC is started, its image is unpacked to a separate location, which can result in bigger overall storage consumption than the servers (and is much bigger than the small LXCs). If overlayfs is supported by the subtype and template, the image only needs to be unpacked once, and will be shared between nodes. The 'lxc' subtype was converted to support this feature.

LXC creation in VIRL is not parallelized, nor run in the background, nor is it performed by OpenStack. This means that LXC node creation runs in a serial manner.Without overlayfs, it can make deployment of larger numbers of larger LXCs a potential bottleneck for simulation startup (e.g. the Ubuntu 14.04 LXC is 580 MB unpacked, which does affect disk I/O eventually).

There's room for improvement and that's why the remainder of this guide focuses on the smaller LXCs, after explaining some other commonalities with them.

Initial configuration through cloud-init

The initial configuration for the 'lxc' subtypes, like the 'server' subtype, uses the cloud-init system.

The cloud-config file represents a YAML dictionary structure. Essentially, it's a text file with keys mapping to values or lists or blocks of text, with comments starting with a hash (#).

Cloud-init is a set of programs that run as the node boots. Cloud-init processes the YAML structure and applies the directives. The documentation of what is available is extensive.

It is important to note that by default, no user can log into a cloud-init image, hence users must always be configured in cloud-init (unless the image has had user accounts manually added before being loaded into the VIRL system).

Note

Cloud-init is very sensitive with respect to the YAML syntax, including the number of spaces at the start of each line. Even syntactically correct YAML (see validator) can still be processed and rejected, which, in combination with the above note on users, will result in a booted system where no one can log into. VIRL can't help you there!

What AutoNetKit generates

ANK will generate the same cloud-init configuration structure for all nodes it deems as 'hosts' (e.g. 'server', 'lxc', or 'lxc-sshd' subtypes).

The main differences between node configurations created by ANK are in the assigned IP addresses and the route entries.

An example configuration, with comments below each directive added, follows:

```
#cloud-config
## identifies the type of the file, must be preserved
```

```
bootcmd:
- ln -s -t /etc/rc.d /etc/rc.local
## list of commands to execute as soon as possible on first boot
## the content is for Fedora compatibility, can be safely removed

hostname: lxc-ubuntu
## Override hostname; default uses the node name

manage_etc_hosts: true
## Create /etc/hosts, so that e.g. own hostname can be reached

runcmd:
- systemctl start rc-local
- sed -i '/-\s*PasswordAuthentication\s\+no/d' /etc/ssh/sshd_config
- echo "UseDNS no" >> /etc/ssh/sshd_config
- service ssh restart
- service sshd restart
## List of commands run later in the first boot, when other things like
## users are already configured

users:
- default
- gecos: User configured by VIRL Configuration Engine 0.21.2
  lock-passwd: false
  name: cisco
  plain-text-passwd: cisco
  shell: /bin/bash
  ssh-authorized-keys:
  - VIRL-USER-SSH-PUBLIC-KEY
  sudo: ALL=(ALL) ALL
## create a set of users on the system.
## by default a cisco/cisco user is created, which can become root using
## sudo -s and its own password.
## An SSH key replaces the ALL-CAPS text whenever the node starts if one
## was configured in the UWM preferences for the user launching the sim.
## each user starts with a - entry at this indentation level

write_files:
- path: /etc/systemd/system/dhclient@.service
  content: |
    [Unit]
    Description=Run dhclient on %i interface
    After=network.target
    [Service]
    Type=oneshot
    ExecStart=/sbin/dhclient %i -pf /var/run/dhclient.%i.pid -lf /var/lib/dhclient/dhclient.
    RemainAfterExit=yes
  owner: root:root
  permissions: '0644'
## Create files in the node at specified paths, with specified ownership and content.
## Containing directories are created automatically.
## Each file is a new entry starting with - at this level. Path and content are mandatory.
## The above entry is for SystemD/Fedora compatibility and can be removed.

- path: /etc/rc.local
  owner: root:root
  permissions: '0755'
  content: |
    #!/bin/sh
    ifconfig eth1 up 10.0.0.13 netmask 255.255.255.252
    route add -host 192.168.0.1 gw 10.0.0.14 dev eth1
    route add -host 192.168.0.2 gw 10.0.0.14 dev eth1
    route add -net 10.0.0.0/8 gw 10.0.0.14 dev eth1
    exit 0
## Another entry in the write_files directive. Instructs cloud-init to create /etc/rc.local,
## a file commonly used to add late boot-time commands to execute each time the node boots.
## note the permissions - the file must be executable.
## Also, mind the spaces, as always; the pipe character may be suffixed with a dash.
## ANK uses this file to configure IP addresses and routes (or to run DHCP) for each interfa
```

LXC images

LXC images contain all the necessary files for the LXC that cannot be obtained from the host's file system when the LXC is being created, or files the LXC wants to use instead of those provided. They must be packaged in a.tar.gz archive, with the content of the archive closely matching the structure of the file system for the LXC.

LXC templates

The containers aren't merely collections of files that form the file system for an LXC. A script, usually written in the bash scripting language, is the program that configures and actually prepares the LXC file system. It is invoked by the underlying lxc-create program invoked by VIRL when the LXC node is instantiated.

The template script takes several input parameters, most important of which is the location where the files for the LXC are to be placed (rootfs) and an input LXC config file that controls most aspects of the LXCs, including networking and file sharing with its host system.

The input config file is distinct from the cloud-init data; it is generated by VIRL when the node starts. The options are explained in the manual page for lxc.container.conf.

The template processes and modifies the input config file.

Note

VIRL starts the template script under the root user; the script executes as a normal program on the host system, therefore caution should be taken in what templates are allowed to be added to the system since it is possible to include malicious operations.

In current releases, users without 'admin' rights are allowed to add their own templates and then start nodes using them. A future release will allow restricting template uploads to 'admin' users only.

The templates used and shipped with VIRL are modified versions of templates from the lxc-templates Ubuntu package, usually found at /usr/share/lxc/templates, when installed. The scripts are licensed under the GNU LGPL license, and may be modified and distributed only under the terms of that license.

The modified templates shipping with VIRL are their own source code, located at /usr/local/lib/python2.7/dist-packages/virl_pkg_data/low_level/lxc. Users may modify and upload new templates based off of them. Note that templates uploaded through the UWM > Node Resources > Containers page take precedence over the bundled templates.

The 'lxc' subtype uses a modified lxc-ubuntu template, which merely skips over its original features for caching and downloading images from the Internet. The inner workings of that template go beyond the scope of this guide.

After the template runs, lxc-create finalizes the setup of the lxc; another program, lxc-start, is then invoked by VIRL to actually start the node.

Things we don't do with LXC

While the LXCs started by VIRL are designed to interact seamlessly with virtual machine-based nodes managed through OpenStack, and serve as a lighter alternative to server VMs, there are some features present on the VM-based nodes that are not supported on LXCs.

Notably, no VNC or serial port access is defined for LXCs; while a VNC server can be launched in an LXC, and there are provisions for serial port definition in containers, the mechanisms for exposing these capabilities do not readily co-exist with the methods used for virtual machines managed by OpenStack.

OpenStack is capable of clustering several compute hosts, distributing the VMs among them; VIRL will only start LXCs on the host where the main applications are running. This is normally the controller node running all the various OpenStack services.

A notable drawback of current releases of VIRL is that Ethernet MAC addresses of ports created through OpenStack are not propagated to the LXCs, meaning random MACs will be assigned to their interfaces. As a result, DHCP services provided by OpenStack external (FLAT/SNAT) and management networks will not function properly.

Another issue is that link down events are not reflected to the LXC's interfaces; no traffic will flow if an interface is disabled in UWM or VM Maestro but the interface link status will be reported as 'up' as far as the LXC is concerned .

The SSHD LXC as the shortest way to get a node

If one needs to place a node somewhere in the topology, and doesn't need anything beyond what's already present on the system itself, there's no easier way to do it than with the lxc-sshd node type.

The lxc-sshd, and the related lxc-tiny, use a mechanism called 'bind-mounting' to create (read-only) views of its host's file system. Each mount represents a single file or directory, grafted into a path in the LXC.

The LXC itself has a skeleton tree of directories, where several directories serve as mount points - empty directories where the mounted directory trees are mapped. Files can be mounted on an individual basis onto (nonexistent) file paths in the LXC.

> ***Note***
>
> Files inside a nonempty mount point will be hidden by the view.

As the name suggests, the main process in the lxc-sshd is a SSH server, allowing one to log into the node through the same means as into VM servers.

This makes the subtype the recommended base for all other LXCs, such as the lxc-routem, lxc-ostinato-drone, lxc-iperf, and all the example types that will follow in this guide.

Using cloud-init configuration without cloud-init

A subset of the cloud-config directive handlers are provided by the VIRL server. Just after the LXC is created using its template, but before it is started, the cloud-config is processed resulting in a set of files placed into the rootfs directory for the LXC node.

The supported directives, with their limitations, are:

- **users**

  as with the regular LXC, no user is created by default, except for users added by the template (e.g. root, and sshd); sudo is not available in an LXC, and no sudoers files will be added

- **groups**

  additional Unix groups to create. For each user, the group with the same name is always its primary; only users defined in the users section should be listed as members

- **write_files**

  the owner and group are currently ignored; only 'root' and users/groups defined in the above sections should be used

- **bootcmd, runcmd**

  creates executable files in /etc/lxc/ with the same name; it is up to the initialization process defined by the template to execute them

- **hostname**

  writes the /etc/hostname file

- **manage_etc_hosts**

  an /etc/hosts file is created. If the node is connected to a FLAT network, a 'virl-server' entry is added with the IP of the VIRL host
- **lxc-create**

  an extension; it may be a list or a simple string without quotes that is appended to the argument list for the template; it's up to the template code to accept and process these arguments

The sshd template

The template for the sshd node has been adapted for improved workflows with SSH in a VIRL topology, and to execute the bootcmd, runcmd and rc.local files.

All the other templates are based on the sshd template, so its sections are explained below.

Note

The lxc-tiny subtype is reserved for possible future small LXCs that aren't based on the sshd template. It itself does neither name nor ship with any template - each node of this subtype requires a template name to be set for it; otherwise, the simulation launch will fail with an error.

If modified templates are distributed, the fact and type of the modification should be noted in the comments. The next section checks whether the LXC was started in an unsupported way, which won't be the case when started by VIRL.

Then it sets up the PATH, a variable with colon-separated directory paths where executable programs are to be found.

```
#!/bin/bash

#
# lxc: Linux Container library

# Authors:
# Daniel Lezcano <daniel.lezcano@free.fr>
# Modified by Cisco Systems, Inc. for use by VIRL lxcs on 2015-10-22.
# The following license text pertains exclusively to this file.

# This library is free software; you can redistribute it and/or
# modify it under the terms of the GNU Lesser General Public
# License as published by the Free Software Foundation; either
# version 2.1 of the License, or (at your option) any later version.

# This library is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# Lesser General Public License for more details.

# You should have received a copy of the GNU Lesser General Public
# License along with this library; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

# Detect use under userns (unsupported)
for arg in "$@"; do
    [ "$arg" = "--" ] && break
    if [ "$arg" = "--mapped-uid" -o "$arg" = "--mapped-gid" ]; then
        echo "This template can't be used for unprivileged containers." 1>&2
        echo "You may want to try the \"download\" template instead." 1>&2
        exit 1
    fi
```

```
done

# Make sure the usual locations are in PATH
export PATH=$PATH:/usr/sbin:/usr/bin:/sbin:/bin
```

Next up is the 'install_sshd' function. This function creates all the directories necessary for sshd (and some added, like /etc/lxc and /etc/virl) in the rootfs.

```
install_sshd()
{
    rootfs=$1

    tree="\
$rootfs/var/run/sshd \
$rootfs/var/empty/sshd \
$rootfs/var/lib/empty/sshd \
$rootfs/var/log \
$rootfs/etc/alternatives \
$rootfs/etc/init.d \
$rootfs/etc/rc.d \
$rootfs/etc/ssh \
$rootfs/etc/sysconfig/network-scripts \
$rootfs/etc/lxc \
$rootfs/etc/virl \
$rootfs/dev/shm \
$rootfs/run/shm \
$rootfs/proc \
$rootfs/sys \
$rootfs/bin \
$rootfs/sbin \
$rootfs/usr \
$rootfs/tmp \
$rootfs/home/ \
$rootfs/root \
$rootfs/lib \
$rootfs/lib64"

    mkdir -p $tree
    if [ $? -ne 0 ]; then
        return 1
    fi

    return 0
}
```

After that follows the 'install_tarball' function; it extracts the LXC image, if one was supplied to the template, to the /lxc directory inside the LXC's rootfs.

```
install_tarball()
{
    rootfs=$1
    target=$rootfs/lxc
    if [ -z "$tarball" ] ; then
        return 0
    fi
    mkdir -p "$target"
    tar xf "$tarball" -C "$target"
}
```

The 'configure_sshd' function creates two users for the LXC, root and sshd, just by creating the two files. VIRL will append to these files (and also /etc/shadow and /etc/gshadow) any users and groups configured through cloud-config.

The sshd configuration file, and a file for environment variables of logged-in users are added alongside. Finally, an SSH host key is generated for the machine.

```
configure_sshd()
{
    rootfs=$1
    chmod 1777 $rootfs/tmp

    cp /etc/services $rootfs/etc
    cat <<EOF > $rootfs/etc/passwd
root:x:0:0:root:/root:/bin/bash
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
EOF

    cat <<EOF > $rootfs/etc/group
root:x:0:root
sshd:x:74:
EOF

    # by default setup root password with no password
    cat <<EOF > $rootfs/etc/ssh/sshd_config
Port 22
Protocol 2
HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_dsa_key
UsePrivilegeSeparation yes
KeyRegenerationInterval 3600
ServerKeyBits 768
SyslogFacility AUTH
LogLevel INFO
LoginGraceTime 120
PermitRootLogin no
StrictModes yes
RSAAuthentication yes
PubkeyAuthentication yes
IgnoreRhosts yes
RhostsRSAAuthentication no
HostbasedAuthentication no
PermitEmptyPasswords yes
ChallengeResponseAuthentication no
UseDNS no
MaxSessions 60
PasswordAuthentication yes
EOF

    cat <<EOF > $rootfs/etc/ssh/ssh_config
UserKnownHostsFile=/dev/null
StrictHostKeyChecking=no
EOF

    key_path="$rootfs/etc/ssh/ssh_host_rsa_key"
    if [ -n "$host_key" -a -f "$host_key" ]; then
        cp "$host_key" "$key_path"
        chmod 600 "$key_path"
        ssh-keygen -y -f "$key_path" > "$key_path.pub"
    else
        ssh-keygen -t rsa -N "" -f "$key_path"
        #ssh-keygen -t dsa -N "" -f $rootfs/etc/ssh/ssh_host_dsa_key
    fi

    cat <<'EOF' >> $rootfs/etc/profile
PS1="\u@\h$ "
export PATH="$PATH:/lxc/bin:/lxc/usr/bin"
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/lxc/lib:/lxc/usr/lib"
EOF

    return 0
}
```

The 'copy_configuration' section processes the lxc container config file. The main section here is the added definitions of lxc.mount.entry lines, each creating a bound mount off of the host file system (sysfs and proc are special here).

All the mount points are 'readonly'; different directories cannot be combined into the same directory. It's best practice not to mount different shares over each other. That's why the 'install_tarball' section is designed to extract the image to the /lxc location - it's clearer to tell what originates from the image and what from the host.

What it also does is to mount its own path as the /sbin/init file inside the LXC file system. When the LXC is started, /sbin/init is executed, which means this template will be executed again.

```
copy_configuration()
{
    path=$1
    rootfs=$2
    name=$3
    template_path=$4

    grep -q "^lxc.rootfs" $path/config 2>/dev/null || echo "lxc.rootfs = $rootfs" >> $path/c
cat <<EOF >> $path/config
lxc.pts = 1024
lxc.kmsg = 0
lxc.cap.drop = sys_module mac_admin mac_override sys_time

# When using LXC with apparmor, uncomment the next line to run unconfined:
#lxc.aa_profile = unconfined

lxc.mount.entry = /dev dev none ro,bind 0 0
lxc.mount.entry = /lib lib none ro,bind 0 0
lxc.mount.entry = /bin bin none ro,bind 0 0
lxc.mount.entry = /usr usr none ro,bind 0 0
lxc.mount.entry = /sbin sbin none ro,bind 0 0
lxc.mount.entry = tmpfs var/run/sshd tmpfs mode=0644 0 0
lxc.mount.entry = $template_path sbin/init none ro,bind 0 0
lxc.mount.entry = proc proc proc nodev,noexec,nosuid 0 0
lxc.mount.entry = sysfs sys sysfs ro 0 0
lxc.mount.entry = /etc/init.d etc/init.d none ro,bind 0 0
lxc.mount.entry = /etc/alternatives etc/alternatives none ro,bind 0 0
lxc.mount.entry = /home/virl/ lxc/virl ro,bind 0 0
EOF

    # Create the mount point
    mkdir -p $rootfs/lxc/virl

    # Oracle Linux and Fedora need the following two bind mounted
    if [ -d /etc/sysconfig/network-scripts ]; then
        cat <<EOF >> $path/config
lxc.mount.entry = /etc/sysconfig/network-scripts etc/sysconfig/network-scripts none ro,bind
EOF
    fi

    if [ -d /etc/rc.d ]; then
        cat <<EOF >> $path/config
lxc.mount.entry = /etc/rc.d etc/rc.d none ro,bind 0 0
EOF
    fi

    if [ "$(uname -m)" = "x86_64" ]; then
        cat <<EOF >> $path/config
lxc.mount.entry = /lib64 lib64 none ro,bind 0 0
EOF
    fi

    # setup sysctl config
    cat <<EOF >> /etc/sysctl.conf
net.ipv6.conf.default.autoconf = 0
net.ipv6.conf.all.autoconf = 0
EOF

}
```

Next are helper functions - one dummy (usage()), where help texts could be added on how to run the template. The other (check_for_cmd()) finds the path to an executable or exits the template with failure.

```
usage()
{
    cat <<EOF
$1 -h|--help -p|--path=<path> [--rootfs=<path>]
EOF
    return 0
}

check_for_cmd()
{
    cmd_path=`type $1`
    if [ $? -ne 0 ]; then
        echo "The command '$1' $cmd_path is not accessible on the system"
        exit 1
    fi
    # we use cut instead of awk because awk is alternatives symlink on ubuntu
    # and /etc/alternatives isn't bind mounted
    cmd_path=`echo $cmd_path | cut -d ' ' -f 3`
}
```

The next section is for processing the parameters passed to the template, both short and long options (e.g. -n does the same as --name). Options taking parameters are suffixed with a colon, the long options are comma-separated.

```
long_opts="help,rootfs:,path:,name:,auth-key:,host-key:,userid:,groupid:,tarball:"
options=$(getopt -o hp:n:S:R:I:G:T: -l $long_opts  -- "$@")
if [ $? -ne 0 ]; then
        usage $(basename $0)
    exit 1
fi
eval set -- "$options"

while true
do
    case "$1" in
        -h|--help)      usage $0 && exit 0;;
        -p|--path)      path=$2; shift 2;;
        --rootfs)       rootfs=$2; shift 2;;
        -n|--name)      name=$2; shift 2;;
        -S|--auth-key)  auth_key=$2; shift 2;;
        -R|--host-key)  host_key=$2; shift 2;;
        -I|--userid)    userid=$2; shift 2;;
        -G|--groupid)   groupid=$2; shift 2;;
        -T|--tarball)   tarball=$2; shift 2;;
        --)             shift 1; break ;;
        \*)              break ;;
    esac
done
```

The next section (below) enforces that the template is run as the root user. As mentioned in the 'copy_configuration' section, the template sets itself up as the /sbin/init for the LXC. This means that the template will be run under that name. The next section deals with what to do during that time. It will still run as root, but within the container - all file paths are taken from the rootfs for that node.

The DHCP section is disabled, as there typically isn't a DHCP server that would answer the node's requests, since the OpenStack dhcp server expects different MAC addresses to be used when the requests are made.

Next, the bootcmd, runcmd and rc.local files are executed, in order. There is nothing for the template to setup or start in between, thus their meaning is not distinguished in any way for the LXCs. Also, the template executes bootcmd and runcmd every time - one can easily remove or disable those scripts after they ran.

Finally, the mandatory init.lxc helper program is executed, instructed to start sshd. The LXC will be considered running as long as init.lxc is running too. The init.lxc program will exit when the program it's told to start (sshd) exits, or when the container is stopped, along with all other processes inside that LXC.

After that, the template's job is done as far as running the LXC is concerned.

```
if [ "$(id -u)" != "0" ]; then
    echo "This script should be run as 'root'"
    exit 1
fi

if [ $0 = "/sbin/init" ]; then

    PATH="$PATH:/bin:/sbin:/usr/sbin:/usr/bin:/lxc/usr/bin"
    check_for_cmd /usr/sbin/init.lxc
    check_for_cmd sshd
    sshd_path=$cmd_path

    # run dhcp?
    #if [ -f /run-dhcp ]; then
    # no
    if false; then
        check_for_cmd dhclient
        check_for_cmd ifconfig
        touch /etc/fstab
        rm -f /dhclient.conf
        cat > /dhclient.conf << EOF
send host-name "<hostname>";
EOF
        ifconfig eth0 up
        dhclient eth0 -cf /dhclient.conf
        echo "Container IP address:"
        ifconfig eth0 | grep inet
    fi

    if [ -x /etc/lxc/bootcmd ] ; then
        /etc/lxc/bootcmd
    fi

    if type -P sysctl ; then
        sysctl -p
    fi
    if type -P ethtool ; then
        ls -1 /sys/class/net | grep eth | while read ifname ; do
```

```
            ethtool -K $ifname tx off
        done
    fi
    if [ -x /etc/lxc/runcmd ] ; then
        /etc/lxc/runcmd
    fi

    if [ -x /etc/rc.local ] ; then
        /etc/rc.local
    fi

    exec /usr/sbin/init.lxc -- $sshd_path
    exit 1
fi
```

Below the /sbin/init mode, the main template code is executed. Parameters are validated and error messages generated if they aren't OK. The functions defined above are called in order to set up the node.

```
if [ -z "$path" ]; then
    echo "'path' parameter is required"
    exit 1
fi

# detect rootfs
config="$path/config"
if [ -z "$rootfs" ]; then
    if grep -q '^lxc.rootfs' $config 2>/dev/null ; then
        rootfs=$(awk -F= '/^lxc.rootfs =/{ print $2 }' $config)
    else
        rootfs=$path/rootfs
    fi
fi

if [ -z "$userid" ]; then
    echo "Userid is missing"
fi

if [ -z "$groupid" ]; then
    echo "Groupid is missing"
fi

install_sshd $rootfs
if [ $? -ne 0 ]; then
    echo "failed to install sshd's rootfs"
    exit 1
fi

configure_sshd $rootfs
if [ $? -ne 0 ]; then
    echo "failed to configure sshd template"
    exit 1
fi

install_tarball $rootfs
if [ $? -ne 0 ]; then
    echo "failed to install local tarball"
    exit 1
fi

copy_configuration $path $rootfs $name $0
if [ $? -ne 0 ]; then
```

```
    echo "failed to write configuration file"
    exit 1
fi
```

The optional image

The lxc-sshd can live without any extra files added to it. As seen above, the tarball parameter is optional. It is not an error to not provide an image for any given node.

VIRL's default mode of operation assigns an image to each node, using a default defined by the subtype, or using the subtype name if undefined. To allow an LXC subtype to make the image optional, the default must be set to 'empty'.

If and only if a subtype says the image is optional, the nodes will not get an image, unless they specifically request one on a per-node basis, by setting the image name in the VM Maestro node properties.

One can create a suitable image file using the tar command.

```
virl@virl:~/cxl/image-1$ tree
.
■■■ dir1
■   ■■■ subdir1
■       ■■■ file1
■■■ dir2
■   ■■■ subdir2
■       ■■■ file2
■■■ file3

4 directories, 3 files
virl@virl:~/cxl/image-1$ tar czvf lxc-image1.tar.gz dir1 dir2/subdir2 file3
dir1/
dir1/subdir1/
dir1/subdir1/file1
dir2/subdir2/
dir2/subdir2/file2
file3
```

Directories are added recursively. The paths for extraction will be the same as the ones used when the files were added. The lxc-sshd template will extract the image into the /lxc location, e.g. creating /lxc/dir2/subdir2/file2.

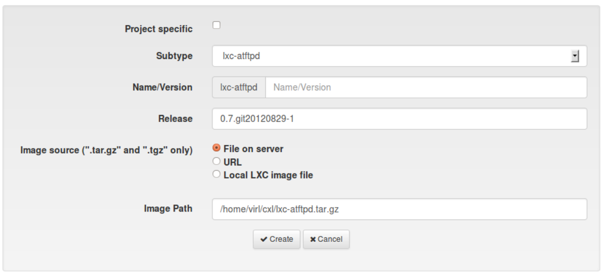
The image can be added to the VIRL system through the UWM Containers page. While not strictly enforced, one should assign each image to the subtype it's designed for.

The name of the image need not be provided in the dialog. The default selected by the subtype will be used, even if it's wildly different from the subtype name. Otherwise, 'subtype name with a suffix' is enforced by UWM for user-provided names.

For subtypes with optional images, the default name of the added image is the name of the subtype, but as mentioned previously, each node wishing to use it must say so explicitly.

Unlike with OpenStack VM images, only one image of a given name may be present in the VIRL system for a given project at a time; this applies to the images shared among all projects as well.

The revision field can be used to track the version of the image, leaving image names constant for portable topologies regardless of which version is used.



Project specific	☐
Subtype	lxc-atftpd
Name/Version	lxc-atftpd Name/Version
Release	0.7.git20120829-1
Image source (".tar.gz" and ".tgz" only)	[✓ File on server] URL Local LXC image file
Image Path	/home/virl/cxl/lxc-atftpd.tar.gz

Customizing templates

Hopefully the explanation on the base template for sshd provided enough information on where changes can be placed and what they do.

For example, the template could be modified to add a view to the /home/virl directory on the host system to all lxc-sshd nodes. A new lxc.mount.entry can be added to the lxc container configuration, and the target mount point - an empty directory inside the lxc prepared. It is important to not end the paths with the directory separator, and not to begin the LXC path with one either. The LXC will fail to start with a message in the given LXC's log about a missing path that isn't supposed to exist anyway, e.g. "lxc-start 1457629785.666 ERROR lxc_conf - conf.c:mount_entry:1711 - No such file or directory - failed to mount 'tmpfs' on '/usr/lib/x86_64-linux-gnu/lxc/var/run/sshd'". The mount point directory needs to be created in the rootfs tree prior to starting the LXC.

The differences to the template would look like this (lines added with plus).

```
--- a/sshd.lxc
+++ b/sshd.lxc
@@ -179,8 +179,12 @@
 lxc.mount.entry = sysfs sys sysfs ro 0 0
 lxc.mount.entry = /etc/init.d etc/init.d none ro,bind 0 0
 lxc.mount.entry = /etc/alternatives etc/alternatives none ro,bind 0 0
+lxc.mount.entry = /home/virl lxc/virl none ro,bind 0 0
 EOF

+    # Create the mount point
+    mkdir -p $rootfs/lxc/virl
+
     # Oracle Linux and Fedora need the following two bind mounted
     if [ -d /etc/sysconfig/network-scripts ]; then
         cat <<EOF >> $path/config
```

One can add the template in the same UWM > Node Resources > Containers page. Essentially the same rules apply, except there is no 'empty' template rule.

Another point to make is that each individual LXC node may pick the template to run as that node's 'flavor' property - to that end, the default_flavor of the subtype provides the name of the template to run if a node doesn't pick one. Also, VM Maestro provides them in the list of flavors, mixed with the OpenStack ones.

When the lxc-sshd node is started with the uploaded template, one has instant access to all the files in the target directory (barring permission settings).

Running other programs (dnsmasq)

The intent of LXCs is having a small number of various applications running in a node, ideally started automatically without user intervention.

The easiest programs to add are ones that are already present on the host, bind-mounted from the VIRL host's /usr directory. An example of this is dnsmasq, which actually serves as the DHCP server for OpenStack networks.

The example below will demonstrate the use of the 'dnsmasq' server - as a DNS server for all nodes in the topology.

Adding a subtype

The simplest method of adding new LXC node subtype, is to base off of the properties of an existing subtype in the UWM > Node Resources > Subtypes page. Given an existing subtype, admins can 'fork' off a new one, changing a few of the available attributes.

The current subtype form is primarily designed for driving OpenStack VMs, hence many of the attributes do not make much sense for LXCs. What a new subtype requires is a distinct name; all other changes are optional. In fact, only a few attributes should ever be changed at all, them being:

Name of new subtype

As a convention, lxc-<main-app-name> is recommended.

Description of plugin

Node subtypes are the only plugins supported.

Protocol for network CLI

If the node isn't supposed to run sshd, can be set to none. An LXC may even be configured to run a telnet server. The server can be reached over the management network of the simulation, thus the telnet/ssh over LXC option of the node in the VM Maestro Active canvas, or the management port in the UWM page for the simulation can be used.

Name of icon for GUI

The GUI is VM Maestro. The names come from a fixed set; one can pick them in the VM Maestro preferences dialog for subtypes.

Show subtype on GUI palette

If enabled, an icon will be shown on the palette; otherwise, the subtype can be switched for all selected nodes in the node properties panel.

RAM (MB) allocated per node

Notionally, LXCs can be memory limited through control groups

Extra comma-separated image properties

Here, the subtype designer can place command line arguments for the template when it's run by lxc-create. They are actually space-separated for LXCs. They will be placed on the command line after the optional lxc-create value from the cloud-config of that node.

Name of default image

The name of default image, or 'empty' if the subtype shouldn't use one.

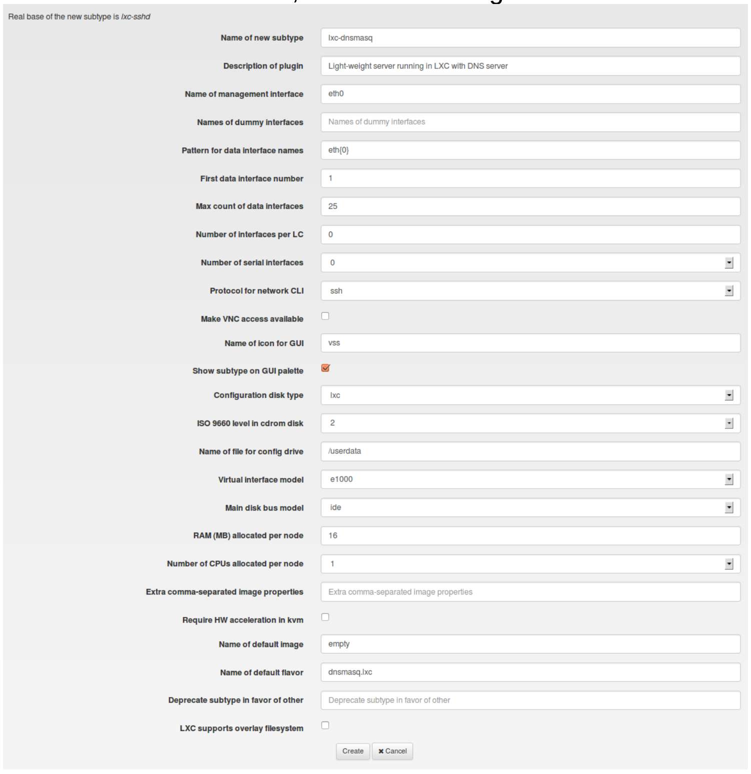
Name of default flavor

As mentioned, this actually controls the default template name for LXCs.

LXC supports overlay filesystem

Instructs the backend to prepare an overlay file system for the template. Further explained in the radius section.

Changing remaining attributes has no effect at best, and breaks things at worst.



Real base of the new subtype is *lxc-sshd*

Name of new subtype	lxc-dnsmasq
Description of plugin	Light-weight server running in LXC with DNS server
Name of management interface	eth0
Names of dummy interfaces	Names of dummy interfaces
Pattern for data interface names	eth{0}
First data interface number	1
Max count of data interfaces	25
Number of interfaces per LC	0
Number of serial interfaces	0
Protocol for network CLI	ssh
Make VNC access available	☐
Name of icon for GUI	vss
Show subtype on GUI palette	☑
Configuration disk type	lxc
ISO 9660 level in cdrom disk	2
Name of file for config drive	/userdata
Virtual interface model	e1000
Main disk bus model	ide
RAM (MB) allocated per node	16
Number of CPUs allocated per node	1
Extra comma-separated image properties	Extra comma-separated image properties
Require HW acceleration in kvm	☐
Name of default image	empty
Name of default flavor	dnsmasq.lxc
Deprecate subtype in favor of other	Deprecate subtype in favor of other
LXC supports overlay filesystem	☐

Propagating new subtypes

Any new subtypes defined this way can be used in VM Maestro To make them available, perform the 'Fetch from server' button in VM Maestro's subtypes dialog.

Switching between LXC subtypes is very straightforward; since no interface name changes are supported, the node will remain essentially unchanged as far as the topology is concerned.

To distribute subtype definitions, one can use the export and import function in the UWM Subtypes page. The examples in this guide use the following subtypes:

```
{
  "dynamic-subtypes": [
    {
      "plugin_base": "lxc-sshd",
      "plugin_desc": "Light-weight server running in LXC with TFTPD",
      "hw_ram": 32,
      "plugin_name": "lxc-atftpd",
      "hw_vm_extra": "",
      "baseline_image": "lxc-atftpd",
      "baseline_flavor": "atftpd.lxc",
      "hw_disk_bus": "ide"
    },
    {
      "plugin_base": "lxc-sshd",
      "plugin_desc": "Light-weight server running in LXC with BIRD",
      "hw_ram": 32,
      "plugin_name": "lxc-bird",
      "hw_vm_extra": "",
      "baseline_image": "lxc-bird",
      "baseline_flavor": "bird.lxc",
      "hw_disk_bus": "ide"
    },
    {
      "plugin_base": "lxc-sshd",
      "plugin_desc": "Light-weight server running in LXC with DNS server",
      "hw_ram": 32,
      "plugin_name": "lxc-dnsmasq",
      "hw_vm_extra": "",
      "baseline_flavor": "dnsmasq.lxc",
      "hw_disk_bus": "ide"
    },
    {
      "plugin_base": "lxc-sshd",
      "plugin_desc": "Light-weight server running in LXC with LLDPD",
      "hw_ram": 32,
      "plugin_name": "lxc-lldpd",
      "hw_vm_extra": "",
      "baseline_image": "lxc-lldpd",
      "baseline_flavor": "lldpd.lxc",
      "hw_disk_bus": "ide"
    },
    {
      "plugin_base": "lxc-sshd",
      "plugin_desc": "Light-weight server running in LXC with FreeRADIUS",
      "hw_ram": 32,
      "plugin_name": "lxc-radius",
      "hw_vm_extra": "",
      "baseline_image": "lxc-radius",
      "baseline_flavor": "radius.lxc",
      "hw_disk_bus": "ide",
      "overlay_support": true
    },
    {
      "plugin_base": "lxc-sshd",
      "plugin_desc": "Light-weight server running in LXC with TACACS+",
      "hw_ram": 32,
      "plugin_name": "lxc-tacacs",
      "hw_vm_extra": "",
      "baseline_image": "lxc-tacacs",
      "baseline_flavor": "tacacs.lxc",
      "hw_disk_bus": "ide"
    }
  ]
}
```

Adding a template

The template for dnsmasq isn't much different from the base sshd.lxc:

```
--- sshd.lxc
+++ dnsmasq.lxc
@@ -5,7 +5,7 @@

 # Authors:
 # Daniel Lezcano <daniel.lezcano@free.fr>
-# Modified by Cisco Systems, Inc. for use by VIRL lxcs on 2015-10-22.
+# Modified by Cisco Systems, Inc. for use by VIRL lxcs on 2016-01-22.
 # The following license text pertains exclusively to this file.

 # This library is free software; you can redistribute it and/or
@@ -94,11 +94,13 @@
     cat <<EOF > $rootfs/etc/passwd
 root:x:0:0:root:/root:/bin/bash
 sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
+nobody:x:99:99:Nobody:/nonexistent:/bin/sh
 EOF

     cat <<EOF > $rootfs/etc/group
 root:x:0:root
 sshd:x:74:
+nogroup:x:99:
 EOF

     # by default setup root password with no password
@@ -148,6 +150,9 @@
 export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/lxc/lib:/lxc/usr/lib"
 EOF

+    cat <<EOF >> $rootfs/etc/profile
+alias rosh='socat UNIX-CONNECT:$rosh_path -,echo=0,icanon=0'
+EOF
     return 0
 }

@@ -228,6 +233,9 @@
     cmd_path=`echo $cmd_path | cut -d ' ' -f 3`
 }

+# location where root shell socket is to be placed
+rosh_path="/var/run/rosh.sock"
+
 long_opts="help,rootfs:,path:,name:,auth-key:,host-key:,userid:,groupid:,tarball:"
 options=$(getopt -o hp:n:S:R:I:G:T: -l $long_opts  -- "$@")
 if [ $? -ne 0 ]; then
@@ -264,6 +272,8 @@
```

```
     check_for_cmd /usr/sbin/init.lxc
     check_for_cmd sshd
     sshd_path=$cmd_path
+    check_for_cmd dnsmasq
+    dnsmasq_path=$cmd_path

     # run dhcp?
     #if [ -f /run-dhcp ]; then
@@ -302,6 +312,10 @@
         /etc/rc.local
     fi

+    # run dnsmasq in background; configured by /etc/dnsmasq.conf
+    $dnsmasq_path &
+    # run also a process listening on a UNIX socket, executing bash on connect
+    socat "UNIX-LISTEN:$rosh_path,fork,mode=0666" "EXEC:/bin/bash -i,stderr,setsid,pty,ctty
     exec /usr/sbin/init.lxc -- $sshd_path
     exit 1
 fi
```

Again, besides marking the date of the change for minimal compliance reasons, lines were only added. The nobody:nogroup user:group pair was added, since that's what dnsmasq switches to in order to avoid being misused if compromised.

The 'check_for_cmd' is used to enforce that dnsmasq is actually found on the host. It will exit the template and fail the start of the LXC if not.

Just after the node was configured through rc.local, commands can be run to make the node behave as intended. Note that the path to the dnsmasq executable was stored in a variable; this is not necessary, but it's good practice.

What is usually necessary is the ampersand (&) at the end of that command; the init process must finish within a time limit set by VIRL in order to declare the node started; until the init.lxc executes, the lxc isn't started properly. The node will be declared 'failed' otherwise and automatically destroyed.

In bash, the ampersand instructs the command preceding it to run in background. The programs that are executed this way will survive the completion of the template script if they are designed to remain in operation (and they didn't fail) for as long as they like, or until the LXC is destroyed when the node or the whole simulation is stopped.

The dnsmasq server's execution is placed in the 'init' section of the template for two reasons. One, the command should be executed automatically and make the DNS services available without user intervention. Two, dnsmasq must be run as the root user.

When any user logs in, they cannot execute commands as another user, hence cannot kill or restart the dnsmasq process.

Another thing that was added to the template to partially solve this problem is 'rosh', short for 'root shell'. It makes use of 'socat'. One socat process is run on init by root that creates and opens a UNIX socket, writable to all other local users, waiting for connections. For each connected user, it creates a new bash shell. The location of the socket is used in several places of the template, hence it's stored in a variable.

For convenience, the means to connect to the socket by a logged in user are put into the /etc/profile file under the 'rosh' alias. This file is read by bash on its startup, making 'rosh' available just like any other command.

Note

The shell may be opened multiple times at once, but it is limited in the interactive reactions - e.g., backspace may not work, Ctrl-W does.

Custom configurations for the topology

Neither the configuration for the dnsmasq service, nor the configuration entries required on all of the other nodes in the topology in order for them to use the DNS name server running on the LXC, will be generated automatically by

ANK. Thus, the configurations for each node need to be manually modified to turn on this additional feature.

The modifications for LXC or Linux instances typically require file-level changes, either as modifications to the rc.local file, or new files placed in specific locations.

By default, dnsmasq reads the file /etc/dnsmasq.conf for its configuration. In this file, additional hosts file can be configured and, for example, names can be associated with the nodes in the topology via mapping. The /etc/hostname file should not be used for this purpose though, hence it can be named /etc/lxc/hosts.

```
- path: /etc/dnsmasq.conf
  owner: root:root
  permissions: '0600'
  content: |
    #localise-queries
    no-resolv
    no-hosts
    addn-hosts=/etc/lxc/hosts
    log-facility=/tmp/dnsmasq.log
- path: /etc/lxc/hosts
  owner: root:root
  permissions: '0666'
  content: |
    10.255.10.1  mgmt-lxc.m
    10.255.10.2  lxc-sshd.m
    10.255.10.3  lxc-lldpd-1.m
    10.255.10.4  iosv-1.m
    10.255.10.5  lxc-lldpd-2.m
    10.255.10.6  lxc-dnsmasq.m
    10.255.10.7  lxc-bird.m
    10.255.10.8  lxc-atftpd.m
    10.255.10.9  lxc-tacacs.m
    10.255.10.10 nx-osv-1.m
    10.255.10.11 lxc-radius.m
    10.255.10.12 lxc-ubuntu.m

    10.0.0.22 lxc-sshd.s
    10.0.0.38 lxc-lldpd-1.s
    10.0.0.34 lxc-lldpd-2.s
    10.0.0.6  lxc-dnsmasq.s
    10.0.0.5  lxc-bird.s
    10.0.0.25 lxc-tacacs.s
    10.0.0.29 lxc-radius.s
    10.0.0.13 lxc-ubuntu.s
    192.168.0.2 iosv-1.s
    192.168.0.1 nx-osv-1.s
```

Not even the lxc-dnsmasq node itself will start using the name server on its own. Hence, an /etc/resolv.conf file can be placed in each of the LXC nodes. For the IOS routers, the startup config needs to modified.

For cloud-config:

```
- path: /etc/resolv.conf
  owner: root:root
  permissions: '0644'
  content: |
    nameserver 10.255.10.6
    nameserver 10.0.0.6
    search m
```

For iosv-1, one needs to set the management vrf:

```
ip name-server vrf Mgmt-intf 10.255.10.6
ip name-server 10.0.0.6
```

For nx-osv-1 it's:

```
ip name-server 10.255.10.6 use-vrf management
ip name-server 10.0.0.6
```

Runtime

```
cisco@lxc-sshd$ dig lxc-ubuntu.m

; <<>> DiG 9.9.5-3ubuntu0.5-Ubuntu <<>> lxc-ubuntu.m
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15567
;; flags: qr aa rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;lxc-ubuntu.m.                  IN      A

;; ANSWER SECTION:
lxc-ubuntu.m.           0       IN      A       10.255.10.12

;; Query time: 0 msec
;; SERVER: 10.255.10.6#53(10.255.10.6)
;; WHEN: Fri Feb 05 21:42:28 UTC 2016
;; MSG SIZE  rcvd: 46

cisco@lxc-sshd$ dig lxc-ubuntu.s

; <<>> DiG 9.9.5-3ubuntu0.5-Ubuntu <<>> lxc-ubuntu.s
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18018
;; flags: qr aa rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;lxc-ubuntu.s.                  IN      A

;; ANSWER SECTION:
lxc-ubuntu.s.           0       IN      A       10.0.0.13

;; Query time: 0 msec
;; SERVER: 10.255.10.6#53(10.255.10.6)
;; WHEN: Fri Feb 05 21:42:32 UTC 2016
;; MSG SIZE  rcvd: 46

cisco@lxc-sshd$ dig lxc-ubuntu.s @10.0.0.6

; <<>> DiG 9.9.5-3ubuntu0.5-Ubuntu <<>> lxc-ubuntu.s @10.0.0.6
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12828
;; flags: qr aa rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;lxc-ubuntu.s.                  IN      A

;; ANSWER SECTION:
lxc-ubuntu.s.           0       IN      A       10.0.0.13

;; Query time: 4 msec
;; SERVER: 10.0.0.6#53(10.0.0.6)
;; WHEN: Fri Feb 05 21:43:13 UTC 2016
;; MSG SIZE  rcvd: 46
```

```
nx-osv-1# show ip route lxc-bird.m vrf management
IP Route Table for VRF "management"
'*' denotes best ucast next-hop
'**' denotes best mcast next-hop
'[x/y]' denotes [preference/metric]
'%<string>' in via output denotes VRF <string>

Resolved "lxc-bird.m" to lxc-bird.m (10.255.10.7)
10.255.0.0/16, ubest/mbest: 1/0, attached
    via 10.255.10.10, mgmt0, [0/0], 03:59:49, direct
nx-osv-1# ping lxc-bird.m vrf management
% Invalid host/interface lxc-bird.m
```

Since /etc/lxc/hosts was made world-writable, any logged-in user can modify it, within limits. The running dnsmasq will reload the hosts files whenever it receives the HUP signal, which can only be sent to it by root - the 'rosh' can be used for this purpose.

```
cisco@lxc-dnsmasq$ sed -i -e '/lxc-bird.s/s/^[0-9.]*/192.168.0.3/' /etc/lxc/hosts
sed: couldn't open temporary file /etc/lxc/sedkEtjWN: Permission denied
cisco@lxc-dnsmasq$ cp /etc/lxc/hosts /tmp
cisco@lxc-dnsmasq$ sed -e '/lxc-bird.s/s/^[0-9.]*/192.168.0.3/' /tmp/hosts >/etc/lxc/hosts
cisco@lxc-dnsmasq$ rosh
bash-4.3# id -a    uid=0(root) gid=0(root) groups=0(root)
bash-4.3# source /etc/profile
root@lxc-dnsmasq$ pkill -HUP dnsmasq
root@lxc-dnsmasq$ exit
cisco@lxc-dnsmasq$ dig lxc-bird.s
; <<>> DiG 9.9.5-3ubuntu0.5-Ubuntu <<>> lxc-bird.s
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 49060
;; flags: qr aa rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;lxc-bird.s.                    IN      A

;; ANSWER SECTION:
lxc-bird.s.             0       IN      A       192.168.0.3

;; Query time: 0 msec
;; SERVER: 10.255.10.6#53(10.255.10.6)
;; WHEN: Sat Feb 06 14:20:38 UTC 2016
;; MSG SIZE  rcvd: 44

cisco@lxc-dnsmasq$ ping lxc-bird.c
ping: unknown host lxc-bird.c
cisco@lxc-dnsmasq$ ping lxc-bird.s
PING lxc-bird.s (192.168.0.3) 56(84) bytes of data.
64 bytes from lxc-bird.s (192.168.0.3): icmp_seq=1 ttl=64 time=0.103 ms
64 bytes from lxc-bird.s (192.168.0.3): icmp_seq=2 ttl=64 time=0.194 ms
64 bytes from lxc-bird.s (192.168.0.3): icmp_seq=3 ttl=64 time=0.179 ms
^C
--- lxc-bird.s ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.103/0.158/0.194/0.042 ms
```

Creating images with new packages (lldpd)

While VIRL users can safely rely on the availability of dnsmasq, not all programs are always a given, in that they may not be present on the host's file system. An individual user can always install new software on the host, but it may clutter it up, especially if the packages involved set up new services that may start automatically.

It also needs to be kept in mind whenever topologies need to move between VIRL systems.

Thus, additional files needed for an LXC can be provided via LXC images in order to provide portability.

Getting things out of the repository

Luckily, most programs widely available already have an Ubuntu package ready for installation. The .deb files contain the compiled executables at the correct paths ready for inclusion into the file system.

One can search for a given package through the apt-cache search function or use the handy http://packages.ubuntu.com site for convenient browsing.

For example, searching for 'lldp' in the repositories for the 14.04 LTS version, (also known as 'trusty'), results in http://packages.ubuntu.com/trusty/lldpd, or the following search output:

```
virl@virl:~$ apt-cache search lldp
ladvd - minimal LLDP/CDP sender
liblldpctl-dev - implementation of IEEE 802.1ab (LLDP) - development files
lldpad - Link Layer Discovery Protocol Implementation (Runtime)
lldpad-dev - Link Layer Discovery Protocol Implementation (Development headers)
lldpd - implementation of IEEE 802.1ab (LLDP)
```

Once the package is known, one can download and look at what's inside.

```
virl@virl:~$ mkdir cxl
virl@virl:~$ cd cxl
virl@virl:~/cxl$ apt-get download lldpd
Get:1 http://us.archive.ubuntu.com/ubuntu/ trusty/universe lldpd amd64 0.7.7-1 [122 kB]
Fetched 122 kB in 0s (221 kB/s)
virl@virl:~/cxl$ mkdir lldpd
virl@virl:~/cxl$ dpkg -X lldpd_0.7.7-1_amd64.deb lldpd
./
./lib/
./lib/systemd/
./lib/systemd/system/
./lib/systemd/system/lldpd.service
./etc/
./etc/init/
./etc/init/lldpd.conf
./etc/init.d/
./etc/init.d/lldpd
./etc/default/
./etc/default/lldpd
./usr/
./usr/lib/
./usr/lib/liblldpctl.so.4.2.0
./usr/sbin/
./usr/sbin/lldpd
./usr/sbin/lldpcli
./usr/share/
./usr/share/doc/
./usr/share/doc/lldpd/
./usr/share/doc/lldpd/NEWS.gz
./usr/share/doc/lldpd/README.Debian
./usr/share/doc/lldpd/changelog.Debian.gz
./usr/share/doc/lldpd/copyright
./usr/share/lintian/
```

```
./usr/share/lintian/overrides/
./usr/share/lintian/overrides/lldpd
./usr/share/man/
./usr/share/man/man8/
./usr/share/man/man8/lldpcli.8.gz
./usr/share/man/man8/lldpctl.8.gz
./usr/share/man/man8/lldpd.8.gz
./usr/lib/liblldpctl.so.4
./usr/sbin/lldpctl
```

Packaging

As seen, the lldpd package isn't very complicated; the most important thing there are the binaries in /usr/sbin. Since the init system inside a small LXC uses neither sysvinit or upstart, the init scripts are fairly useless.

We can also see that there are documentation and example config files. The LXCs can do without these files. While it is a matter of preference, not including unnecessary files will make the LXCs just that little bit smaller and faster to get extracted and started.

Note

The packages in Ubuntu repositories are most likely open source, which means they are freely distributable, under some conditions. The conditions usually require retention of copyright and licensing information with the distribution. If LXC image are to be distributed, such files must stay in place.

```
root@virl:~/cxl# rm -r lldpd/{lib,etc,usr/share}
root@virl:~/cxl# tree lldpd
lldpd
■■■ usr
    ■■■ lib
    ■   ■■■ liblldpctl.so.4 -> liblldpctl.so.4.2.0
    ■   ■■■ liblldpctl.so.4.2.0
    ■■■ sbin
        ■■■ lldpcli
        ■■■ lldpctl -> lldpcli
        ■■■ lldpd
```

When satisfied with the contents of the packaging directory, it can be packaged into an LXC image and subsequently added into the VIRL system through UWM.

The example below using the UWM CLI client to create the LXC image, setting its 'version' field, which becomes part of the name, as well as subtype and revision. It uses the absolute local path on the server so that the image won't have to be uploaded as part of the API request ($PWD contains the path of working directory, in the example /home/virl/cxl). Use virl_uwm_client lxc-image-create --help for further information.

```
virl@virl:~/cxl$ virl_uwm_client --quiet --json lxc-image-create   -v 'one' \
> -t lxc-lldpd -r 0.7.19-1 -S $PWD/lxc-lldpd.tar.gz
{
  "lxc-image": {
    "properties": {
      "release": "0.7.19-1",
      "subtype": "lxc-lldpd",
      "version": "one"
    },
    "name": "lxc-lldpd-one",
    "filepath": "/var/local/virl/lxc/images/c64f06bb-c1a4-491e-9436-44a467ca74a0__lxc-lldpd-
    "created": "2016-02-01 08:00:44",
    "updated": "2016-02-01 08:00:44",
    "filename": "c64f06bb-c1a4-491e-9436-44a467ca74a0__lxc-lldpd-one.tar",
    "public": "True",
    "id": "c64f06bb-c1a4-491e-9436-44a467ca74a0",
    "projects": [
      "uwmadmin"
    ],
    "size": 440320
  },
  "disk-usage": {
    "percentage-usage": "42.37",
    "used-disk-space-GB": "29.67",
    "total-disk-space-GB": "70.04"
  }
}
```

Note

As the example suggests, the client currently requires a version id, hence is unable to create the image with the default name derived from its subtype. This will be fixed in a subsequent release .

Modifying the template

One thing that was done differently for the above package is that the resulting tarball has a root directory embedded in every path. During extraction, that directory can be skipped with the '--strip-components' parameter to tar. The package can be created so that it does not to include that directory, as a matter of preference.

Moreover, the programs in the package will be extracted to /lxc/usr/sbin instead of the more common /lxc/usr/bin - many programs that are supposed to be services will follow this convention. The paths must be adjusted for this, so that files can be found and executed.

By trial and error, and eventually by reading the published source code of the project, one can find out that lldpd uses the '_lldpd' user and group just like dnsmasq uses nobody:nogroup.

The lldpd service comes with a CLI client. The service and the CLI client communicate through a UNIX socket, a special file created by lldpd as it starts. This file needs (root) privileges for programs including lldpcli to access it. The user logging into the system at runtime won't have the necessary permissions. One can wait for the socket to get created and then modify its permissions to make it world writable. While this is not recommended in general practice, one can do it in simulations.

The lldpd has some command line options that may be valuable depending on the role of the LXC in question; one can check whether the LXC's cloud-config created a file with the options to pass in the template init section; it's useful not to require one though.

The resulting template differences from the original sshd template might be:

```
--- sshd.lxc
+++ lldpd.lxc
@@ -5,7 +5,7 @@

 # Authors:
 # Daniel Lezcano <daniel.lezcano@free.fr>
-# Modified by Cisco Systems, Inc. for use by VIRL lxcs on 2015-10-22.
+# Modified by Cisco Systems, Inc. for use by VIRL lxcs on 2016-01-22.
 # The following license text pertains exclusively to this file.

 # This library is free software; you can redistribute it and/or
@@ -81,7 +81,7 @@
         return 0
     fi
     mkdir -p "$target"
-    tar xf "$tarball" -C "$target"
+    tar xf "$tarball" -C "$target" --strip-components=1
```

```
 }


@@ -94,11 +94,13 @@
     cat <<EOF > $rootfs/etc/passwd
 root:x:0:0:root:/root:/bin/bash
 sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
+_lldpd:x:99:99:Nobody:/nonexistent:/bin/sh
 EOF

     cat <<EOF > $rootfs/etc/group
 root:x:0:root
 sshd:x:74:
+_lldpd:x:99:
 EOF

     # by default setup root password with no password
@@ -144,7 +146,7 @@

     cat <<'EOF' >> $rootfs/etc/profile
 PS1="\u@\h$ "
-export PATH="$PATH:/lxc/bin:/lxc/usr/bin"
+export PATH="$PATH:/lxc/bin:/lxc/usr/bin:/lxc/usr/sbin"
 export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/lxc/lib:/lxc/usr/lib"
 EOF

@@ -259,11 +261,14 @@
 fi

 if [ $0 = "/sbin/init" ]; then
-
-    PATH="$PATH:/bin:/sbin:/usr/sbin:/usr/bin:/lxc/usr/bin"
+    exec &> /tmp/lxc-start.log
+    set -x
+    PATH="$PATH:/bin:/sbin:/usr/sbin:/usr/bin:/lxc/usr/sbin"
     check_for_cmd /usr/sbin/init.lxc
     check_for_cmd sshd
     sshd_path=$cmd_path
+    check_for_cmd lldpd
+    lldpd_path=$cmd_path

     # run dhcp?
     #if [ -f /run-dhcp ]; then
@@ -302,6 +307,23 @@
         /etc/rc.local
     fi

+    lldpd_conf="/etc/lxc/lldpd.conf"
+    if [ -f "$lldpd_conf" ]; then
+        lldpd_args=$(< $lldpd_conf)
+    fi
+    $lldpd_path $lldpd_args &>/tmp/lldpd.log &
+    (
+        # make the communication socket writable to all once it appears
+        sock_path=/var/run/lldpd.socket
+        # wait for at most 10 seconds
+        for ii in `seq 10` ; do
+            if [ -e "$sock_path" ] ; then
+                chmod a+rw "$sock_path"
+                break
+             fi
+             sleep 1
+         done
+     ) &
      exec /usr/sbin/init.lxc -- $sshd_path
      exit 1
 fi
```

Configuration

There is a configuration file that is read in by the lldpd process on startup that can be used to pass in additional arguments. Note that the template code does not handle whitespace in the arguments.

```
- path: /etc/lxc/lldpd.conf
  owner: root:root
  permissions: '0644'
  content: -c -l -S Some_Nexus_7000_node_description
```

In respect to the example topology, the routers need to have lldp enabled since this is not present by default. On nx-osv-1, it is enabled by the command 'feature lldp' which can be added alongside the existing 'feature' lines created by ANK.

On the iosv-1 node, the global-level command 'lldp run' can be added, followed by a new line with the exclamation mark.

At runtime, one can access, control and configure the lldp daemon through lldpcli:

```
cisco@lxc-lldpd-1$ lldpcli
[lldpcli] # sh conf
-------------------------------------------------------------------------------
Global configuration:
-------------------------------------------------------------------------------
Configuration:
  Transmit delay: 30
  Transmit hold: 4
  Receive mode: no
  Pattern for management addresses: (none)
  Interface pattern: (none)
  Interface pattern for chassis ID: (none)
  Override description with: Some_Nexus_7000_node_description
  Override platform with: Linux
  Override system name with: (none)
  Advertise version: yes
  Update interface descriptions: no
  Promiscuous mode on managed interfaces: no
  Disable LLDP-MED inventory: yes
  LLDP-MED fast start mechanism: yes
  LLDP-MED fast start interval: 1
  Source MAC for LLDP frames on bond slaves: local
  Portid TLV Subtype for lldp frames: unknown
-------------------------------------------------------------------------------
[lldpcli] # sh nei
-------------------------------------------------------------------------------
LLDP neighbors:
-------------------------------------------------------------------------------
Interface:    eth0, via: CDPv2, RID: 2, Time: 0 day, 00:00:24
  Chassis:
    ChassisID:    local nx-osv-1(TB3EBFC79BB)
    SysName:      nx-osv-1(TB3EBFC79BB)
    SysDescr:     N7K-C7018 running on
                  Cisco Nexus Operating System (NX-OS) Software, Version 7.2(0)D1(1)
    MgmtIP:       10.255.10.10
    Capability:   Bridge, on
    Capability:   Router, on
  Port:
    PortID:       ifname mgmt0
    PortDescr:    mgmt0
-------------------------------------------------------------------------------
Interface:    eth1, via: LLDP, RID: 3, Time: 0 day, 00:00:23
  Chassis:
    ChassisID:    mac fa:16:3e:4b:aa:ed
    SysName:      iosv-1.virl.info
    SysDescr:     Cisco IOS Software, IOSv Software (VIOS-ADVENTERPRISEK9-M), Version 15.6(1
                  Technical Support: http://www.cisco.com/techsupport
                  Copyright (c) 1986-2015 by Cisco Systems, Inc.
                  Compiled Fri 20-Nov-15 13:39 by prod_rel_team
    MgmtIP:       10.255.10.4
    Capability:   Bridge, off
    Capability:   Router, on
  Port:
    PortID:       ifname Gi0/3
    PortDescr:    to lxc-lldpd-1
-------------------------------------------------------------------------------
[lldpcli] #
```

```
iosv-1#show lldp neighbor GigabitEthernet 0/3 detail
------------------------------------------------
Local Intf: Gi0/3
Chassis id: 1227.b99b.c34a
Port id: ea59.11e4.3372
Port Description: eth1
System Name: lxc-lldpd-1.m

System Description:
Some_Nexus_7000_node_description

Time remaining: 119 seconds
System Capabilities: B,W,R,S
Enabled Capabilities: R,S
Management Addresses:
    IP: 10.255.10.3
    IPV6: FE80::1027:B9FF:FE9B:C34A
Auto Negotiation - not supported
Physical media capabilities - not advertised
Media Attachment Unit type: 33
Vlan ID: - not advertised


Total entries displayed: 1
```

Template parameters (atftpd)

In the section of code around the 'getopt' command, command line arguments may be passed in the template. Both the subtype's 'Additional image properties' and the individual node's cloud-config may be used to pass in additional arguments to the template.

Let's use a tftp server, atftpd, as an example. The package contents are rather trivial by comparison to the previous lldpd example - it's just the single executable file.

```
root@virl:~/cxl# rm -r atftpd/{etc/,usr/sbin/in.tftpd,usr/share}
root@virl:~/cxl# tree atftpd
atftpd
```

```
■■■ usr
    ■■■ sbin
            ■■■ atftpd
```

Adding additional parameters to the template

For the LXC template, the pattern used in the sshd template can be reused to provide a directory on the host system. The parameters will accept the location of the files on the VIRL host. Another will switch from default 'read-only' to 'read-write' mode.

Any bind-mounts can be made writable by the template but it is strongly recommended to not even try to mount the host's system directories. There is a potential to cause serious harm in this way.

In addition to the atftpd system, the sshd's internal sftp server can be set to use the same directory; note that e.g. scp does not use the sftp subsystem.

Atftpd also uses the nobody:nogroup user and group.

The additional options (--mount and --writable) also have short versions, -M and -W. The mount, when handled, takes an additional parameter, indicated by the colon suffix. Both options are provided as parameters for the 'getopt' command. Both need to be handled by setting the appropriate variable $mount_point and $mount_mode. There is also the $share_dir variable, containing the location for the share inside the LXC, which is reused at different locations in the file.

The template may 'require' the additional options to be provided, but this is discouraged; it's better if the template won't fail if the parameters aren't provided. Its better to write an error message, e.g. in the /tmp/ directory of the LXC and let the node start normally.

The differences to the original sshd template look like this:

```
--- sshd.lxc
+++ atftpd.lxc
@@ -5,7 +5,7 @@

 # Authors:
 # Daniel Lezcano <daniel.lezcano@free.fr>
-# Modified by Cisco Systems, Inc. for use by VIRL lxcs on 2015-10-22.
+# Modified by Cisco Systems, Inc. for use by VIRL lxcs on 2016-01-22.
 # The following license text pertains exclusively to this file.

 # This library is free software; you can redistribute it and/or
@@ -81,7 +81,7 @@
         return 0
     fi
     mkdir -p "$target"
-    tar xf "$tarball" -C "$target"
+    tar xf "$tarball" -C "$target" --strip-components=1
 }


@@ -94,11 +94,13 @@
     cat <<EOF > $rootfs/etc/passwd
 root:x:0:0:root:/root:/bin/bash
 sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
+nobody:x:99:99:Nobody:/nonexistent:/bin/sh
 EOF

     cat <<EOF > $rootfs/etc/group
 root:x:0:root
 sshd:x:74:
+nogroup:x:99:
 EOF
```

```
     # by default setup root password with no password
@@ -125,6 +127,7 @@
 UseDNS no
 MaxSessions 60
 PasswordAuthentication yes
+Subsystem sftp internal-sftp -d /$share_dir
 EOF

     cat <<EOF > $rootfs/etc/ssh/ssh_config
@@ -159,6 +162,11 @@
     name=$3
     template_path=$4

+    if [ -n "$mount_path" ] ; then
+        mount_entry="lxc.mount.entry = $mount_path $share_dir none $mount_mode,bind 0 0"
+        mkdir -p "$rootfs/$share_dir"
+    fi
+
     grep -q "^lxc.rootfs" $path/config 2>/dev/null || echo "lxc.rootfs = $rootfs" >> $path/
 cat <<EOF >> $path/config
 lxc.pts = 1024
@@ -179,6 +187,7 @@
 lxc.mount.entry = sysfs sys sysfs ro 0 0
 lxc.mount.entry = /etc/init.d etc/init.d none ro,bind 0 0
 lxc.mount.entry = /etc/alternatives etc/alternatives none ro,bind 0 0
+$mount_entry
 EOF

     # Oracle Linux and Fedora need the following two bind mounted
@@ -229,13 +238,17 @@
 }

 long_opts="help,rootfs:,path:,name:,auth-key:,host-key:,userid:,groupid:,tarball:"
-options=$(getopt -o hp:n:S:R:I:G:T: -l $long_opts  -- "$@")
+long_opts="$long_opts,mount:,writable"
+options=$(getopt -o hp:n:S:R:I:G:T:M:W -l $long_opts  -- "$@")
 if [ $? -ne 0 ]; then
         usage $(basename $0)
     exit 1
 fi
 eval set -- "$options"
-
+# constant for the mount point which will be shared
+share_dir='lxc/share'
+# default to read-only
+mount_mode="ro"
 while true
 do
     case "$1" in
@@ -248,6 +261,8 @@
         -I|--userid)    userid=$2; shift 2;;
         -G|--groupid)   groupid=$2; shift 2;;
         -T|--tarball)   tarball=$2; shift 2;;
+        -M|--mount)     mount_path=$2; shift 2;;
+        -W|--writable)  mount_mode="rw"; shift 1;;
         --)             shift 1; break ;;
         *)              break ;; # always end with*
     esac
@@ -259,11 +274,14 @@
 fi
 if [ $0 = "/sbin/init" ]; then
-
-    PATH="$PATH:/bin:/sbin:/usr/sbin:/usr/bin:/lxc/usr/bin"
+    exec &> /tmp/lxc-start.log
+    set -x
+    PATH="$PATH:/bin:/sbin:/usr/sbin:/usr/bin:/lxc/usr/sbin"
     check_for_cmd /usr/sbin/init.lxc
     check_for_cmd sshd
     sshd_path=$cmd_path
+    check_for_cmd atftpd
+    atftpd_path=$cmd_path

     # run dhcp?
     #if [ -f /run-dhcp ]; then
@@ -302,6 +320,7 @@
         /etc/rc.local
     fi

+    $atftpd_path --daemon "/$share_dir" &
     exec /usr/sbin/init.lxc -- $sshd_path
     exit 1
 fi
```

Configuration

The lxc-atftpd node can set the CLI arguments in the cloud-config:

```
lxc-create: -M /home/virl/cxl/shared -W
```

The mounted directory must exist:

```
virl@virl:~/cxl$ mkdir shared
virl@virl:~/cxl$ chmod g=rwxst,o=rwX shared
virl@virl:~/cxl$ echo 'hello' > shared/hello
```

Now, the node configuration does not have access to identify the user who is launching the LXC. Thus, something fancy such as per-user shared directories aren't achievable. However each node can pick a different location (directory) if required.

The location is made writable; if something writes to the location, it will have set the ownership of the file to a different user than 'virl'. It uses 'nobody' with user id '99'.

```
nx-osv-1# copy running-config tftp://10.255.10.8/nx-osv-1.config.txt vrf management
Trying to connect to tftp server......
Connection to Server Established.
TFTP put operation was successful
Copy complete.
nx-osv-1# copy tftp://10.255.10.8/hello hello vrf management
Trying to connect to tftp server......
Connection to Server Established.
TFTP get operation was successful
Copy complete, now saving to disk (please wait)...
nx-osv-1# tail hello
hello
```

```
virl@virl:~/cxl$ tree -ugh shared
shared
■■■ [virl     virl        6]  hello
■■■ [99       virl      15K]  nx-osv-1.config.txt

0 directories, 2 files
```

Note that there is some confusion between sftp and scp - paths will be relative to the home directory when using 'scp'. For instance, on an LXC:

```
cisco@lxc-sshd$ sftp lxc-atftpd.m
Warning: Permanently added 'lxc-atftpd.m,10.255.10.8' (RSA) to the list of known hosts.
cisco@lxc-atftpd.m's password:
Connected to lxc-atftpd.m.
sftp> dir
hello                     nx-osv-1.config.txt
sftp> get hello
Fetching /lxc/share/hello to hello
/lxc/share/hello                              100%    6     0.0KB/s   00:00
sftp> pwd
Remote working directory: /lxc/share
sftp> ^D
cisco@lxc-sshd$ scp lxc-atftpd.m:hello .
Warning: Permanently added 'lxc-atftpd.m,10.255.10.8' (RSA) to the list of known hosts.
cisco@lxc-atftpd.m's password:
scp: hello: No such file or directory
cisco@lxc-sshd$ scp lxc-atftpd.m:/lxc/share/hello .
Warning: Permanently added 'lxc-atftpd.m,10.255.10.8' (RSA) to the list of known hosts.
cisco@lxc-atftpd.m's password:
hello
```

Complex packages and dependencies (tac_plus)

Programs don't always come in one package and even more rarely as single standalone programs! Even the previous examples have their dependencies which are fortunately all found in the regular places searched when an application is loading .

This section will discuss handling multiple packages using the example of a TACACS+ server.

Package dependencies

Looking at the package page, http://packages.ubuntu.com/trusty/tacacs+, it is obvious that the package has build dependencies. One can check whether these are all present on the host system:

```
virl@virl:~/cxl$ dpkg -l libc6 libpam0g libtacacs+1 libwrap0 python
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                                     Version                  Architecture
+++-========================================-========================-=======================
ii  libc6:amd64                              2.19-0ubuntu6.6          amd64
ii  libpam0g:amd64                           1.1.8-1ubuntu2           amd64
ii  libwrap0:amd64                           7.6.q-25                 amd64
ii  python                                   2.7.5-5ubuntu3           amd64
dpkg-query: no packages found matching libtacacs+1
```

This means that libtacacs+1 is also a package that needs to be included in the image for the LXC. That package has dependencies of its own, potentially going recursively, until all dependencies can be resolved.

```
root@virl:~/cxl# apt-get download tacacs+ libtacacs+1
Get:1 http://us.archive.ubuntu.com/ubuntu/ trusty/universe libtacacs+1 amd64 4.0.4.26-3 [35.
Get:2 http://us.archive.ubuntu.com/ubuntu/ trusty/universe tacacs+ amd64 4.0.4.26-3 [97.8 kB
Fetched 133 kB in 0s (233 kB/s)
root@virl:~/cxl# mkdir tacacs+
root@virl:~/cxl# dpkg -X tacacs+_4.0.4.26-3_amd64.deb  tacacs+
./
./etc/
./etc/tacacs+/
./etc/tacacs+/tac_plus.conf
```

```
./etc/init.d/
./etc/init.d/tacacs_plus
./etc/default/
./etc/default/tacacs+
./etc/logrotate.d/
./etc/logrotate.d/tacacs+
./usr/
./usr/sbin/
./usr/sbin/tac_plus
./usr/sbin/do_auth
./usr/sbin/tac_pwd
./usr/share/
./usr/share/doc/
./usr/share/doc/tacacs+/
./usr/share/doc/tacacs+/FAQ.gz
./usr/share/doc/tacacs+/copyright
./usr/share/lintian/
./usr/share/lintian/overrides/
./usr/share/lintian/overrides/tacacs+
./usr/share/man/
./usr/share/man/man8/
./usr/share/man/man8/tac_plus.8.gz
./usr/share/man/man8/tac_pwd.8.gz
./usr/share/man/man8/do_auth.8.gz
./usr/share/man/man5/
./usr/share/man/man5/tac_plus.conf.5.gz
./usr/share/doc/tacacs+/changelog.Debian.gz
root@virl:~/cxl# dpkg -X libtacacs+1_4.0.4.26-3_amd64.deb  tacacs+
./
./usr/
./usr/lib/
./usr/lib/x86_64-linux-gnu/
./usr/lib/x86_64-linux-gnu/tacacs/
./usr/lib/x86_64-linux-gnu/tacacs/libtacacs.so.1.0.0
./usr/share/
./usr/share/doc/
./usr/share/doc/libtacacs+1/
./usr/share/doc/libtacacs+1/changelog.Debian.gz
./usr/share/doc/libtacacs+1/copyright
./usr/share/lintian/
./usr/share/lintian/overrides/
./usr/share/lintian/overrides/libtacacs+1
./usr/lib/x86_64-linux-gnu/tacacs/libtacacs.so.1
root@virl:~/cxl# rm -r tacacs+/{etc,usr/share}
root@virl:~/cxl# tree tacacs+
tacacs+
■■■ usr
    ■■■ lib
    ■   ■■■ x86_64-linux-gnu
    ■       ■■■ tacacs
    ■           ■■■ libtacacs.so.1 -> libtacacs.so.1.0.0
    ■           ■■■ libtacacs.so.1.0.0
    ■■■ sbin
        ■■■ do_auth
        ■■■ tac_plus
        ■■■ tac_pwd
```

Making programs find their libraries

The program tac_plus requires the library to execute. By default, it would not be found - /lxc/usr/lib/x86_64-linux-gnu/tacacs isn't a directory that gets searched for libraries. Similar to the 'PATH' variable, the environment variable 'LD_LIBRARY_PATH' can contain additional directories to search.

To know what is missing from a binary, the ldd program may be used.

```
cisco@lxc-tacacs$ ldd /lxc/usr/sbin/tac_plus
    linux-vdso.so.1 =>  (0x00007ffd00d79000)
    libwrap.so.0 => /lib/x86_64-linux-gnu/libwrap.so.0 (0x00007f33be146000)
    libtacacs.so.1 => not found
    libpam.so.0 => /lib/x86_64-linux-gnu/libpam.so.0 (0x00007f33bdf38000)
    libcrypt.so.1 => /lib/x86_64-linux-gnu/libcrypt.so.1 (0x00007f33bdcff000)
    libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f33bdae1000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f33bd71c000)
    libnsl.so.1 => /lib/x86_64-linux-gnu/libnsl.so.1 (0x00007f33bd502000)
    libaudit.so.1 => /lib/x86_64-linux-gnu/libaudit.so.1 (0x00007f33bd2de000)
    libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f33bd0da000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f33be56e000)
cisco@lxc-tacacs$ export LD_LIBRARY_PATH="/lxc/usr/lib/x86_64-linux-gnu/tacacs"
cisco@lxc-tacacs$ ldd /lxc/usr/sbin/tac_plus
    linux-vdso.so.1 =>  (0x00007fff899cb000)
    libwrap.so.0 => /lib/x86_64-linux-gnu/libwrap.so.0 (0x00007f65d309e000)
    libtacacs.so.1 => /lxc/usr/lib/x86_64-linux-gnu/tacacs/libtacacs.so.1 (0x00007f65d2e4500
    libpam.so.0 => /lib/x86_64-linux-gnu/libpam.so.0 (0x00007f65d2c37000)
    libcrypt.so.1 => /lib/x86_64-linux-gnu/libcrypt.so.1 (0x00007f65d29fe000)
    libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f65d27e0000)
    libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f65d241b000)
    libnsl.so.1 => /lib/x86_64-linux-gnu/libnsl.so.1 (0x00007f65d2201000)
    libaudit.so.1 => /lib/x86_64-linux-gnu/libaudit.so.1 (0x00007f65d1fdd000)
    libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f65d1dd9000)
    /lib64/ld-linux-x86-64.so.2 (0x00007f65d34c6000)
```

Similar PATH-like variables exist for scripting languages too, e.g. PYTHONPATH, PERL5LIB, or CLASSPATH for Java.

The template in this example doesn't have many new things added to it; setting the paths in /etc/profile ensures that all logged in users can find the programs, and the programs can find their libraries.

Note

It is common to append additional paths to the path variables, but it may be useful to prepend them sometimes. If a library or binary file is present on the host system, it will be found and then loaded or run in preference of the files provided by the LXC image. If that file is incompatible for any reason, the application being run may fail to start .

```
--- sshd.lxc
+++ tacacs.lxc
@@ -5,7 +5,7 @@

 # Authors:
 # Daniel Lezcano <daniel.lezcano@free.fr>
-# Modified by Cisco Systems, Inc. for use by VIRL lxcs on 2015-10-22.
+# Modified by Cisco Systems, Inc. for use by VIRL lxcs on 2016-01-22.
 # The following license text pertains exclusively to this file.

 # This library is free software; you can redistribute it and/or
@@ -81,7 +81,7 @@
         return 0
     fi
     mkdir -p "$target"
```

```
-    tar xf "$tarball" -C "$target"
+    tar xf "$tarball" -C "$target" --strip-components=1
 }


@@ -145,7 +145,7 @@
     cat <<'EOF' >> $rootfs/etc/profile
 PS1="\u@\h$ "
 export PATH="$PATH:/lxc/bin:/lxc/usr/bin"
-export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/lxc/lib:/lxc/usr/lib"
+export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/lxc/lib:/lxc/usr/lib:/lxc/usr/lib/x86_64-linux-gn
 EOF

     return 0
@@ -259,11 +259,15 @@
 fi

 if [ $0 = "/sbin/init" ]; then
-
-    PATH="$PATH:/bin:/sbin:/usr/sbin:/usr/bin:/lxc/usr/bin"
+    exec >&/tmp/lxc-start.log
+    set -x
+    PATH="$PATH:/bin:/sbin:/usr/sbin:/usr/bin:/lxc/usr/sbin"
+    source /etc/profile
     check_for_cmd /usr/sbin/init.lxc
     check_for_cmd sshd
     sshd_path=$cmd_path
+    check_for_cmd tac_plus
+    tac_plus_path=$cmd_path

     # run dhcp?
     #if [ -f /run-dhcp ]; then
@@ -302,6 +306,7 @@
         /etc/rc.local
     fi

+    $tac_plus_path -d 511 -C /etc/tac_plus/tac_plus.conf &> /tmp/tac_plus.log &
     exec /usr/sbin/init.lxc -- $sshd_path
     exit 1
 fi
```

Configuration

Configuration for the server gets placed into the tac_plus.conf file used by the template.

```
- path: /etc/tac_plus/tac_plus.conf
  owner: root:root
  permissions: '0666'
  content: |
    key = 123-my_tacacs_key

    group = netadmin {
        default service = permit
        service = exec {
                priv-lvl = 15
        }
    }

    user = lexicon {
        login = cleartext "lexicon"
        member = netadmin
    }

    user = cisco {
        login = cleartext "cisco"
        member = netadmin
    }
```

Again, additional configuration is required on each of the nodes in the topology in order for them to use the Tacacs+ service provided by the container. In the case of the iosv-1 node, the following configuration lines need to be added (ANK will not provide these):

```
aaa new-model
aaa authentication login default group tacacs+ local
aaa authentication enable default group tacacs+
aaa authorization exec default group tacacs+ local
!
tacacs server lxc-tacacs
    address ipv4 10.0.0.25
    key 123-my_tacacs_key
!
line vty 0 4
 transport input ssh telnet
 exec-timeout 720 0
 password cisco
 login local
 login authentication default
line con 0
 password cisco
 login authentication default
!
```

Getting outside the repository (lldpd redux)

Not all programs are in the Ubuntu 14.04 LTS repository. Some may have newer versions available in newer repositories, yet others might not be present in any repository. For the latter case, one can still get all the necessary files and connect them all through PATH variables.

For the former, all is not lost - the newer packages might still run on the system, if the changes aren't too large.

For example, the lldpd package in the upcoming Ubuntu 16.04 'Xenial Xerus' release is much more advanced than what's shipping in Ubuntu 14.04 'Trusty'. See for comparison the search results site.

The new lldpcli has many more features and a configuration file can be loaded by lldpd, as long as it finds the lldpcli command too.

The new package has a new dependency (libjansson4); that's already covered the same way tacacs+ was.

One thing that differs is that the .deb packages need to be downloaded manually. It can be done even from the package page itself, for example, at http://packages.ubuntu.com/xenial/amd64/libjansson4/download.

The package structure, at a minimum, isn't too different from the original:

```
virl@virl:~/cxl$ tree lldpd
lldpd
■■■ usr
    ■■■ lib
    ■   ■■■ libjansson.so.4 -> libjansson.so.4.7.0
    ■   ■■■ libjansson.so.4.7.0
    ■   ■■■ liblldpctl.so.4 -> liblldpctl.so.4.7.0
    ■   ■■■ liblldpctl.so.4.7.0
    ■■■ sbin
        ■■■ lldpcli
        ■■■ lldpctl -> lldpcli
        ■■■ lldpd
```

The differences in the template are minor:

```
--- a/lldpd.lxc
+++ b/lldpd.lxc
@@ -264,11 +264,14 @@
     exec &> /tmp/lxc-start.log
     set -x
     PATH="$PATH:/bin:/sbin:/usr/sbin:/usr/bin:/lxc/usr/sbin"
+    source /etc/profile
     check_for_cmd /usr/sbin/init.lxc
     check_for_cmd sshd
     sshd_path=$cmd_path
     check_for_cmd lldpd
     lldpd_path=$cmd_path
+    check_for_cmd lldpcli
+    lldpcli_path=$cmd_path

     # run dhcp?
     #if [ -f /run-dhcp ]; then
@@ -311,7 +314,7 @@
     if [ -f "$lldpd_conf" ]; then
         lldpd_args=$(< $lldpd_conf)
     fi
-    $lldpd_path $lldpd_args &>/tmp/lldpd.log &
+    $lldpd_path -c $lldpcli_path $lldpd_args &>/tmp/lldpd.log &
     (
         # make the communication socket writable to all once it appears
         sock_path=/var/run/lldpd.socket
```

This allows the use of a different configuration file for lldpd:

```
- path: /etc/lldpd.conf
  owner: root:root
  permissions: '0644'
  content: |
    configure system description 'Description for this node'
    pause
```

Brand new packages (radius 3)

Some programs do not have an Ubuntu package at all, or may use a version that is EOL already. Or one simply wants to run the latest version. Such a package needs to be built from scratch, which requires some more involved steps. Lets take FreeRADIUS, 3.0.X as an example.

One other feature explained here is the new support of overlayfs for use by VIRL topology nodes. The overlay file system is a method of merging a 'lower' directory, with an 'upper' directory into a common view mounted at a specified location. Any changes made to the view only affect the 'upper' directory.

If a subtype is marked as supporting overlayfs, the simulation engine unpacks the LXC image into a common location, serving as the unmodifiable 'lower' directory for all nodes using that image. For each node, an empty 'upper' directory gets created, and an overlay is mounted where the LXC node's rootfs will lie.

The template, cloud-init support in the simengine, and the running LXC see the overlay as a regular directory, without the need to setup anything themselves.

This saves both setup times and the storage space, no longer having a separate copy set up for each node.

Preparing the package

As always, the package is merely a collection of files at correct locations. At the time of writing, Ubuntu Xenial has FreeRADIUS at 2.2.8, which is deemed to be 'end-of-life' (see http://freeradius.org/download.html).

The 3.0.X package can be built from source, since the project is open source. The source can be received from GitHub, using a valid HTTPS_PROXY, if necessary.

```
virl@virl:~/cxl$ mkdir radius radiusrc
virl@virl:~/cxl$ cd radiusrc
virl@virl:~/cxl/radiusrc$ HTTPS_PROXY="" git clone https://github.com/FreeRADIUS/freeradius-
virl@virl:~/cxl/radiusrc$ sudo apt-get install libtalloc-dev
virl@virl:~/cxl/radiusrc$ cd freeradius-server
```

Instructions on how to build any particular software may vary greatly and are not subject of this guide. The necessary steps should be included in the package's documentation.

Since the package is being built from source as part of this process, the build process can be directed to place the resulting files into the required locations. Note the use of the 'R' parameter when running the make command instead of the more common 'DESTDIR' to override the location for installation files.

```
virl@virl:~/cxl/radiusrc/freeradius-server$ git checkout 3.0.11 # optional
virl@virl:~/cxl/radiusrc/freeradius-server$ ./configure --prefix=/lxc/usr
virl@virl:~/cxl/radiusrc/freeradius-server$ make -j8
virl@virl:~/cxl/radiusrc/freeradius-server$ make R=/home/virl/cxl/radius install
virl@virl:~/cxl/radiusrc/freeradius-server$ cd ../../radius
```

Note

Since the project is licensed under the GPL, licenses shall not be removed from the binary package if it is to be distributed to third parties.

A difference from the previous image examples is due to the overlay support - all paths in the image must be absolute, since there's no support for either stripping components, nor for mounting the image at a subdirectory.

```
virl@virl:~/cxl/radius$ rm -r lxc/usr/{share/{man,doc},lib/{*.a,*.la},include}
virl@virl:~/cxl/radius$ tar czf ../lxc-radius.tar.gz lxc
```

The template

Instead of providing the path to the image as an argument for the template via the -T option, VIRL presents the overlaid rootfs path with the -O option. Note that the argument is only provided for convenience; the rootfs option will have the same value.

The template needs to recognize the -O option and act accordingly - enforce the rootfs in the container configuration file and assume the image is already unpacked. The template can and should be written to accept both -O or -T as options, preferring overlay support.

The image will be already present in the rootfs at the time the template is run. All paths in the example image start with /lxc, because they must be separate from the bind-mounted host directories.

One cannot e.g. bind-mount the host's /usr directory and keep a /usr directory from the image - the new mount hides anything already existing under the mount point.

What one may do is add, remove and modify any files coming from the shared image without affecting the original image directory, nor any other LXC node using that same image.

The default file locations for all radius files are part of the configuration file radiusd.conf, hence they are kept at the place where they were unpacked. A symbolic link is created to point /etc/raddb to the /lxc path.

The server enforces the permissions of its configuration, hence it won't be available to the logged-in user for editing. Also, the server does support the HUP signal mechanism discussed with dnsmasq. The 'rosh' approach to making modifications to the config and reloading may be useful here as well.

The server will complain about the OpenSSL version used in 'trusty', which, although patched, still bears the version number of a known vulnerable release.

The differences to the original sshd template look like this:

```
--- sshd.lxc
+++ radius.lxc
@@ -5,7 +5,7 @@

 # Authors:
 # Daniel Lezcano <daniel.lezcano@free.fr>
-# Modified by Cisco Systems, Inc. for use by VIRL lxcs on 2015-10-22.
+# Modified by Cisco Systems, Inc. for use by VIRL lxcs on 2016-03-22.
 # The following license text pertains exclusively to this file.

 # This library is free software; you can redistribute it and/or
@@ -76,12 +76,13 @@
 install_tarball()
 {
     rootfs=$1
-    target=$rootfs/lxc
-    if [ -z "$tarball" ] ; then
-        return 0
+    if [ -n "$overlay" ] ; then
+        true
+    elif [ -n "$tarball" ] ; then
+        mkdir -p "$rootfs"
+        tar xf "$tarball" -C "$rootfs"
     fi
-    mkdir -p "$target"
-    tar xf "$tarball" -C "$target"
+    (cd $rootfs/etc && ln -s ../lxc/usr/etc/raddb .)
 }


@@ -144,10 +145,13 @@

     cat <<'EOF' >> $rootfs/etc/profile
 PS1="\u@\h$ "
-export PATH="$PATH:/lxc/bin:/lxc/usr/bin"
+export PATH="$PATH:/lxc/bin:/lxc/usr/bin:/lxc/usr/sbin"
 export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/lxc/lib:/lxc/usr/lib"
 EOF

+    cat <<EOF >> $rootfs/etc/profile
+alias rosh='socat UNIX-CONNECT:$rosh_path -,echo=0,icanon=0'
+EOF
     return 0
 }

@@ -159,7 +163,13 @@
     name=$3
     template_path=$4

-    grep -q "^lxc.rootfs" $path/config 2>/dev/null || echo "lxc.rootfs = $rootfs" >> $path/
+    if [ -n "$overlay" ] ; then
+        sed -i -e '/^lxc.rootfs/d' $path/config
+        echo "lxc.rootfs = $overlay" >> $path/config
+    else
+        grep -q "^lxc.rootfs" $path/config 2>/dev/null || echo "lxc.rootfs = $rootfs" >> $p
+    fi
+
 cat <<EOF >> $path/config
 lxc.pts = 1024
 lxc.kmsg = 0
```

```
@@ -228,8 +238,11 @@
     cmd_path=`echo $cmd_path |cut -d ' ' -f 3`
 }

-long_opts="help,rootfs:,path:,name:,auth-key:,host-key:,userid:,groupid:,tarball:"
-options=$(getopt -o hp:n:S:R:I:G:T: -l $long_opts  -- "$@")
+# location where root shell socket is to be placed
+rosh_path="/var/run/rosh.sock"
+
+long_opts="help,rootfs:,path:,name:,auth-key:,host-key:,userid:,groupid:,tarball:,overlay:"
+options=$(getopt -o hp:n:S:R:I:G:T:O: -l $long_opts  -- "$@")
 if [ $? -ne 0 ]; then
         usage $(basename $0)
     exit 1
@@ -248,6 +261,7 @@
         -I|--userid)    userid=$2; shift 2;;
         -G|--groupid)   groupid=$2; shift 2;;
         -T|--tarball)   tarball=$2; shift 2;;
+        -O|--overlay)   overlay=$2; shift 2;;
         --)             shift 1; break ;;
         *)              break ;;
     esac
@@ -259,11 +273,14 @@
 fi

 if [ $0 = "/sbin/init" ]; then
-
-    PATH="$PATH:/bin:/sbin:/usr/sbin:/usr/bin:/lxc/usr/bin"
+    exec &> /tmp/lxc-start.log
+    set -x
+    PATH="$PATH:/bin:/sbin:/usr/sbin:/usr/bin:/lxc/usr/sbin"
     check_for_cmd /usr/sbin/init.lxc
     check_for_cmd sshd
     sshd_path=$cmd_path
+    check_for_cmd radiusd
+    radiusd_path=$cmd_path

     # run dhcp?
     #if [ -f /run-dhcp ]; then
@@ -302,6 +319,10 @@
         /etc/rc.local
     fi

+    sed -i -e "/allow_vulnerable_openssl/s/no/CVE-2014-0160/" /etc/raddb/radiusd.conf
+    $radiusd_path -X -d /lxc/usr/etc/raddb &>/tmp/radius.log &
+    # run also a process listening on a UNIX socket, executing bash on connect
+    socat "UNIX-LISTEN:$rosh_path,fork,mode=0666" "EXEC:/bin/bash -i,stderr,setsid,pty,ctty
     exec /usr/sbin/init.lxc -- $sshd_path
     exit 1
 fi
```

The configuration

Thanks to the way overlayfs is setup by the backend, the /etc/raddb symlink created previously by the template can be used instead of the actual path where the config is placed by the image without issues. This is because it is relative and thus valid even during the processing of the cloud-init data happening after creation of the container and before its run, executed from outside the LXC. Note that the user configuration requires the use of tabs instead of spaces. Hence, each line of the contents starts with four spaces, with a tab after. There is a tab after username too.

```
- path: /etc/raddb/clients.conf
  owner: root:root
  permissions: '0644'
  content: |
    client nx-osv-1 {
        ipaddr = 10.0.0.30
        secret =  123radiuskey
        nas_type = cisco
    }
- path: /etc/raddb/users
  owner: root:root
  permissions: '0644'
  content: |
    "lexicon"   Cleartext-Password := "lexicon"
            Service-Type = NAS-Prompt-User,
            cisco-avpair = "shell:priv-lvl=15",
            cisco-avpair = "shell:roles*\"network-admin vdc-admin\""
```

The configuration for nx-osv-1 might look like this:

```
radius-server host 10.0.0.29 key 123radiuskey authentication accounting
aaa group server radius lxc-radius
    server 10.0.0.29

aaa authentication login default group lxc-radius
aaa accounting default group lxc-radius
aaa authentication login error-enable
```

This minimal version works seemingly better than the lxc-tacacs:

```
nx-osv-1# show users
NAME     LINE         TIME         IDLE          PID COMMENT
lexicon  pts/0        Feb  8 05:54   .         16942 (mgmt-lxc.m) *
nx-osv-1# test aaa server radius 10.0.0.29 lexicon lexicon
user has been authenticated
```

Routing (bird)

As noted previously, the lxc-sshd subtype is treated by ANK the same way as the 'server' nodes, configuring it for a role of a host, with a router connected (iosv-1) to it, selected as its gateway to all other IPs generated by ANK (i.e. no default route).

If a host node isn't connected to a router, no host is designated as a gateway and therefore no routes are generated for the host node.

For example, the lxc-dnsmasq node is connected to lxc-bird, which is considered to be another host as far as ANK is concerned. As a result, the node only gets its IP on eth1 from ANK. Fortunately, one can override the default or any other route manually.

```
- path: /etc/rc.local
  owner: root:root
  permissions: '0755'
  content: |
    #!/bin/sh
    ifconfig eth1 up 10.0.0.6 netmask 255.255.255.252
    route del default
    route add default gw 10.0.0.5
    exit 0
```

If the node is connected to an external network (such as flat, flat1 or snat), a default gateway entry will be created pointing to the external network interface. Otherwise, the management network interface has the default route.

Note

The project management network has a provision for dynamic NAT to the 'ext-net' network, same network as the SNAT nodes. This is typically connected to the eth3 interface of the VIRL host.

Image and template

BIRD is a routing daemon for Linux that can support BGP, RIP and OSPF protocols.

The package contains a daemon and a cli client:

```
virl@virl:~/cxl$ tree bird
bird
■■■ usr
    ■■■ sbin
        ■■■ bird
        ■■■ bird6
        ■■■ birdc
        ■■■ birdc6
```

The template is very similar to example shown previously; there is a nobody:nogroup user, and a socket by which the client communicates with the server, except that it's in a slightly different place.

```
--- sshd.lxc
+++ bird.lxc
@@ -5,7 +5,7 @@

 # Authors:
 # Daniel Lezcano <daniel.lezcano@free.fr>
-# Modified by Cisco Systems, Inc. for use by VIRL lxcs on 2015-10-22.
+# Modified by Cisco Systems, Inc. for use by VIRL lxcs on 2016-01-22.
 # The following license text pertains exclusively to this file.

 # This library is free software; you can redistribute it and/or
@@ -81,7 +81,7 @@
         return 0
     fi
     mkdir -p "$target"
-    tar xf "$tarball" -C "$target"
+    tar xf "$tarball" -C "$target" --strip-components=1
 }


@@ -144,7 +144,7 @@

     cat <<'EOF' >> $rootfs/etc/profile
 PS1="\u@\h$ "
-export PATH="$PATH:/lxc/bin:/lxc/usr/bin"
+export PATH="$PATH:/lxc/bin:/lxc/usr/sbin"
 export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/lxc/lib:/lxc/usr/lib"
 EOF

@@ -259,11 +259,14 @@
 fi

 if [ $0 = "/sbin/init" ]; then
-
-    PATH="$PATH:/bin:/sbin:/usr/sbin:/usr/bin:/lxc/usr/bin"
+    exec &> /tmp/lxc-start.log
+    set -x
+    PATH="$PATH:/bin:/sbin:/usr/sbin:/usr/bin:/lxc/usr/sbin"
     check_for_cmd /usr/sbin/init.lxc
     check_for_cmd sshd
     sshd_path=$cmd_path
+    check_for_cmd bird
+    bird_path=$cmd_path

     # run dhcp?
     #if [ -f /run-dhcp ]; then
@@ -302,6 +305,23 @@
         /etc/rc.local
     fi

+    ln -s /var/run /run
+    mkdir -p /run/bird
+    chmod a+rwX /run/bird
+    chmod -R a+rwX /etc/bird
+    sock_path=/run/bird/bird.ctl
+    $bird_path -s "$sock_path" &
+    (
+        # make the communication socket writable to all once it appears
+        # wait for at most 10 seconds
+        for ii in `seq 10` ; do
+            if [ -e "$sock_path" ] ; then
+                chmod a+rw "$sock_path"
+                break
+            fi
+            sleep 1
+        done
+    ) &
     exec /usr/sbin/init.lxc -- $sshd_path
     exit 1
 fi
```

Configuration

Routers get a router id assigned to them by ANK. It is stored in the 'ipv4' attribute of the node itself and gets configured on the router loopback0 interface. Something similar can be performed manually by configuring an IP address in the appropriate field in VM Maestro and by creating an empty bridge interface ('router-id' in the example below) on node startup using the rc.local method.

The bridge is probably the easiest virtual interface to create. A loopback isn't suitable for this purpose on a Linux host, as it's not supposed to route potentially outgoing traffic (i.e., one wouldn't be able to ping it from the outside).

```
- path: /etc/rc.local
  owner: root:root
  permissions: '0755'
  content: |
    #!/bin/sh
    ifconfig eth1 up 10.0.0.10 netmask 255.255.255.252
    ifconfig eth2 up 10.0.0.17 netmask 255.255.255.252
    ifconfig eth3 up 10.0.0.5 netmask 255.255.255.252
    ip link add router-id type bridge
    ip addr add 192.168.0.3/32 dev router-id
    ip link set dev router-id up
    exit 0
```

The OSPF configuration applied in BIRD ensures that all three ethernet interfaces get enabled in the protocol, with an additional stub network for the router-id bridge.

One important note is that the OSPF 'hello' and 'dead timers' must be the same as on the neighboring Cisco routers; they are best set to their default of 10 and 40, respectively, to avoid the need to reconfigure all the routers too.

For OSPF, the Cisco routers are already configured sufficiently to be able to establish an adjacency but configuration needs to be applied to BIRD as per below.

```
- path: /etc/bird/bird.conf
  owner: root:root
  permissions: '0666'
  content: |
    router id 192.168.0.3;

    protocol device {
            scan time 10;
    }

    protocol kernel {
            export all;
            scan time 15;
    }

    protocol ospf MyOSPF {
        tick 2;
        rfc1583compat yes;
        area 0.0.0.0 {
            stub no;
            stubnet 192.168.0.3/32;
            interface "eth1", "eth2", "eth3" {
                hello 10;
                dead 40;
                retransmit 6;
                cost 10;
                transmit delay 5;
                dead count 5;
                wait 50;
                type broadcast;
                #authentication simple;
                #password "pass";
            };
        };
    }
```

Runtime

The initial routing table for the lxc as configured by ANK will appears similar to the example below:

```
cisco@lxc-bird$ ip route
10.0.0.4/30 dev eth3  proto kernel  scope link  src 10.0.0.5
10.0.0.8/30 dev eth1  proto kernel  scope link  src 10.0.0.10
10.0.0.16/30 dev eth2  proto kernel  scope link  src 10.0.0.17
10.255.0.0/16 dev eth0  proto kernel  scope link  src 10.255.10.7
```

The BIRD runtime client, birdc, can be used to get configuration and status data from the daemon. The configure command reloads the configuration from the file, conveniently made world writable by cloud-config in order to allow writing to by the logged-in users too.

```
cisco@lxc-bird$ birdc
BIRD 1.4.0 ready.
bird> sh ospf
MyOSPF:
RFC1583 compatibility: enable
Stub router: No
```

```
RT scheduler tick: 2
Number of areas: 1
Number of LSAs in DB:   5
        Area: 0.0.0.0 (0) [BACKBONE]
                Stub:   No
                NSSA:   No
                Transit:        No
                Number of interfaces:   3
                Number of neighbors:    2
                Number of adjacent neighbors:   2
bird> sh ospf state

area 0.0.0.0

        router 192.168.0.1
                distance 10
                network 10.0.0.16/30 metric 40
                stubnet 192.168.0.1/32 metric 1
                stubnet 10.0.0.24/30 metric 40
                stubnet 10.0.0.12/30 metric 40
                stubnet 10.0.0.28/30 metric 40

        router 192.168.0.2
                distance 10
                network 10.0.0.8/30 metric 1
                stubnet 192.168.0.2/32 metric 1
                stubnet 10.0.0.20/30 metric 1
                stubnet 10.0.0.36/30 metric 1
                stubnet 10.0.0.32/30 metric 1

        router 192.168.0.3
                distance 0
                network 10.0.0.8/30 metric 10
                network 10.0.0.16/30 metric 10
                stubnet 10.0.0.4/30 metric 10
                stubnet 192.168.0.3/32 metric 10

        network 10.0.0.8/30
                dr 192.168.0.3
                distance 10
                router 192.168.0.3
                router 192.168.0.2

        network 10.0.0.16/30
                dr 192.168.0.3
                distance 10
                router 192.168.0.3
                router 192.168.0.1
```

Once the routes converge, the routing table will be filled as well, depending on BIRD filtering rules.

```
cisco@lxc-bird$ ip route
10.0.0.4/30 dev eth3  proto kernel  scope link  src 10.0.0.5
10.0.0.8/30 dev eth1  proto kernel  scope link  src 10.0.0.10
10.0.0.12/30 via 10.0.0.18 dev eth2  proto bird
10.0.0.16/30 dev eth2  proto kernel  scope link  src 10.0.0.17
10.0.0.20/30 via 10.0.0.9 dev eth1  proto bird
10.0.0.24/30 via 10.0.0.18 dev eth2  proto bird
10.0.0.28/30 via 10.0.0.18 dev eth2  proto bird
10.0.0.32/30 via 10.0.0.9 dev eth1  proto bird
10.0.0.36/30 via 10.0.0.9 dev eth1  proto bird
```

```
10.255.0.0/16 dev eth0  proto kernel  scope link  src 10.255.10.7
192.168.0.1 via 10.0.0.18 dev eth2  proto bird
192.168.0.2 via 10.0.0.9 dev eth1  proto bird
```

Once active, connectivity can be confirmed by some simple checks:

```
cisco@lxc-sshd$ ping 192.168.0.3
PING 192.168.0.3 (192.168.0.3) 56(84) bytes of data.
64 bytes from 192.168.0.3: icmp_seq=2 ttl=63 time=3.36 ms
64 bytes from 192.168.0.3: icmp_seq=3 ttl=63 time=1.74 ms
^C
--- 192.168.0.3 ping statistics ---
3 packets transmitted, 2 received, 33% packet loss, time 2009ms
rtt min/avg/max/mdev = 1.740/2.553/3.367/0.815 ms
cisco@lxc-sshd$ tracepath lxc-ubuntu.s
 1?: [LOCALHOST]                                         pmtu 1500
 1:  10.0.0.21                                             2.954ms
 1:  10.0.0.21                                             3.007ms
 2:  10.0.0.10                                             2.841ms
 3:  10.0.0.18                                             5.047ms
 4:  lxc-ubuntu.s                                          4.941ms reached
     Resume: pmtu 1500 hops 4 back 4
```

```
iosv-1#sh ip ospf neighbor

Neighbor ID     Pri   State           Dead Time   Address         Interface
192.168.0.3       1   FULL/DR         00:00:38    10.0.0.10       GigabitEthernet0/1
iosv-1#sh ip ospf rib

            OSPF Router with ID (192.168.0.2) (Process ID 1)


                Base Topology (MTID 0)

    OSPF local RIB
Codes: * - Best, > - Installed in global RIB

*>  10.0.0.4/30, Intra, cost 11, area 0
      via 10.0.0.10, GigabitEthernet0/1
*   10.0.0.8/30, Intra, cost 1, area 0, Connected
      via 10.0.0.9, GigabitEthernet0/1
*>  10.0.0.12/30, Intra, cost 51, area 0
      via 10.0.0.10, GigabitEthernet0/1
*>  10.0.0.16/30, Intra, cost 11, area 0
      via 10.0.0.10, GigabitEthernet0/1
*   10.0.0.20/30, Intra, cost 1, area 0, Connected
      via 10.0.0.21, GigabitEthernet0/4
*>  10.0.0.24/30, Intra, cost 51, area 0
      via 10.0.0.10, GigabitEthernet0/1
*>  10.0.0.28/30, Intra, cost 51, area 0
      via 10.0.0.10, GigabitEthernet0/1
*   10.0.0.32/30, Intra, cost 1, area 0, Connected
      via 10.0.0.33, GigabitEthernet0/2
*   10.0.0.36/30, Intra, cost 1, area 0, Connected
      via 10.0.0.37, GigabitEthernet0/3
*>  192.168.0.1/32, Intra, cost 12, area 0
      via 10.0.0.10, GigabitEthernet0/1
*   192.168.0.2/32, Intra, cost 1, area 0, Connected
      via 192.168.0.2, Loopback0
*>  192.168.0.3/32, Intra, cost 11, area 0
```

```
      via 10.0.0.10, GigabitEthernet0/1

_*
```

Debugging and troubleshooting

Where things are

When an LXC is successfully created and exists (even if it were shut down), it will be displayed in the lxc-ls -f output. Note that only 'root' can see LXCs.

```
virl@virl:~/cxl$ sudo -s
root@virl:~/cxl# lxc-ls -f
NAME                                       STATE    IPV4
-----------------------------------------------------------------------------------------------
virl-15d5acbe-f620-4342-8425-996efabaa317  RUNNING  10.255.10.8
virl-2910bc87-302b-470b-b229-ff7f153f7d36  RUNNING  10.0.0.25, 10.255.10.9
virl-5574a772-0198-4b24-8c68-da300dda60db  RUNNING  10.0.0.13, 10.255.10.12
virl-97d7afc8-a2e7-470f-86e1-3f4d69e755e4  RUNNING  10.255.10.1, 172.16.1.191
virl-b7cb467c-835a-46fc-bc35-93db7f582f41  RUNNING  10.0.0.22, 10.255.10.2
virl-bc3e0933-486e-488c-9400-b5111e0300f2  RUNNING  10.0.0.34, 10.255.10.5
virl-ccbcd091-614a-45b7-b445-54a180ca725e  RUNNING  10.0.0.6, 10.255.10.6
virl-df427bc9-7b09-4e93-9cfc-df8fe1d03729  RUNNING  10.0.0.10, 10.0.0.17, 10.0.0.5, 10.255.1
virl-e2a71e25-1193-4843-8cd1-b8d2abfc09ed  RUNNING  10.0.0.29, 10.255.10.11
virl-f74d92ed-9e13-4597-a5e4-409c97423a95  RUNNING  10.0.0.38, 10.255.10.3
```

The root file systems, as well as their configs, are located under /var/lib/lxc. You can also see the relative sizes of the LXCs (the example below was made without overlayfs support; can you spot the Ubuntu and RADIUS instances?). Files may be modified in the file system on the fly; the LXC will see the change immediately.

Commands (by default, a shell) can be executed inside the LXC by root using the syntax lxc-attach --name virl-id [--command arguments].

```
root@virl:~/cxl# du -hs /var/lib/lxc/*
176K    /var/lib/lxc/virl-15d5acbe-f620-4342-8425-996efabaa317
624K    /var/lib/lxc/virl-2910bc87-302b-470b-b229-ff7f153f7d36
665M    /var/lib/lxc/virl-5574a772-0198-4b24-8c68-da300dda60db
116K    /var/lib/lxc/virl-97d7afc8-a2e7-470f-86e1-3f4d69e755e4
112K    /var/lib/lxc/virl-b7cb467c-835a-46fc-bc35-93db7f582f41
544K    /var/lib/lxc/virl-bc3e0933-486e-488c-9400-b5111e0300f2
120K    /var/lib/lxc/virl-ccbcd091-614a-45b7-b445-54a180ca725e
1.1M    /var/lib/lxc/virl-df427bc9-7b09-4e93-9cfc-df8fe1d03729
58M     /var/lib/lxc/virl-e2a71e25-1193-4843-8cd1-b8d2abfc09ed
544K    /var/lib/lxc/virl-f74d92ed-9e13-4597-a5e4-409c97423a95
root@virl:~/cxl# lxc-attach \
> --name virl-e2a71e25-1193-4843-8cd1-b8d2abfc09ed \
> -- /lxc/usr/sbin/radmin -h
Usage: radmin [ args ]
  -d raddb_dir    Configuration files are in "raddbdir/".
  -D <dictdir>    Set main dictionary directory (defaults to /lxc/usr/share/freeradius).
  -e command      Execute 'command' and then exit.
  -E              Echo commands as they are being executed.
  -f socket_file  Open socket_file directly, without reading radius.conf
  -h              Print usage help information.
  -i input_file   Read commands from 'input_file'.
  -n name         Read raddb/name.conf instead of raddb/radiusd.conf
  -q              Quiet mode.
```

The node IDs, while unique, aren't very descriptive of what's inside. The config file contains the 'utsname' attribute to sort them out by the name given in the topology but it cannot disambiguate between simulations. The IP addresses above may help in that case.

```
root@virl:~/cxl# grep utsname /var/lib/lxc/*/config*
/var/lib/lxc/virl-15d5acbe-f620-4342-8425-996efabaa317/config:lxc.utsname = lxc-atftpd
/var/lib/lxc/virl-2910bc87-302b-470b-b229-ff7f153f7d36/config:lxc.utsname = lxc-tacacs
/var/lib/lxc/virl-5574a772-0198-4b24-8c68-da300dda60db/config:lxc.utsname = lxc-ubuntu
/var/lib/lxc/virl-5574a772-0198-4b24-8c68-da300dda60db/config:lxc.utsname = virl-5574a772-01
/var/lib/lxc/virl-97d7afc8-a2e7-470f-86e1-3f4d69e755e4/config:lxc.utsname = mgmt-OluWiu
/var/lib/lxc/virl-b7cb467c-835a-46fc-bc35-93db7f582f41/config:lxc.utsname = lxc-sshd
/var/lib/lxc/virl-bc3e0933-486e-488c-9400-b5111e0300f2/config:lxc.utsname = lxc-lldpd-2
/var/lib/lxc/virl-ccbcd091-614a-45b7-b445-54a180ca725e/config:lxc.utsname = lxc-dnsmasq
/var/lib/lxc/virl-df427bc9-7b09-4e93-9cfc-df8fe1d03729/config:lxc.utsname = lxc-bird
/var/lib/lxc/virl-e2a71e25-1193-4843-8cd1-b8d2abfc09ed/config:lxc.utsname = lxc-radius
/var/lib/lxc/virl-f74d92ed-9e13-4597-a5e4-409c97423a95/config:lxc.utsname = lxc-lldpd-1
```

Logging

The operations of bringing up a node are first logged to the STD log at /var/local/virl/logs/std_server.log. The output of the lxc-create is appended to the container.log file in the same directory.

Once a node is created, its cloud-config will be performed if it is a small LXC. After that, it will be started. The logs for the startup process are located at /var/log/lxc/<node-id>.log. Information such as misconfigurations of mounts and networking will have messages logged in there.

The LXC template can assist with logging as well; by using the 'set -x' command, all subsequent commands and parameters will be logged to stderr. When in 'init' mode, the LXC template may redirect its outputs to a file inside the LXC for subsequent review. The output of long-term commands should be redirected to logs of their own. Many of the example templates above do just that.

The applications being run may have logs of their own, and configurations or command line switches to control logging behavior. Refer to the manuals of the programs themselves for options on doing so.

Debugging

If an LXC has invalid or wrong configuration so that the LXC cannot be started properly, it will likely not work at all. One common symptom of that is that the IP addresses won't be configured properly. VIRL looks for that and will declare a failure to start the container, if the addresses seen in the LXC aren't a superset of those it configured from the interface and port parameters; more IPs than configured are allowed, but the original IPs must be present.

When a failed node is detected this way, it will be automatically deleted. This also means that the contents of /var/lib/lxc for the LXC will be deleted, which may hinder debugging. For this reason, one can configure VIRL not to delete failed LXCs on start. The nodes will get into ERROR state instead. The configuration may be changed as indicated below, and then the command 'service virl-std restart' issued.

```
# add into file /etc/virl/common.cfg
# then execute sudo service virl-std restart
[orchestration]
destroy_failed_lxc = False
```

If a node fails after an unsuccessful start, its state in lxc-ls will be STOPPED, and in VIRL, SHUTOFF. The node needs to be stopped by VIRL, or lxc-destroy called on it, in order to be able to start a new one in its stead.

Note that if the LXC node is using an overlayfs, it should be unmounted first; lxc-destroy will not be able to remove paths coming from the image, hence declare the destruction as failed otherwise. On the other hand, an unmounted upper directory will be cleaned by lxc-destroy properly.

The logs might help with determining what went wrong; or trying to execute the commands from the template after logging or attaching to the LXC. If messages from the program itself aren't enough even when its own debugging is enabled, the command's system calls may be logged by prepending the command with 'strace'.

Note that the template file may be edited in place by root - a node may be stopped and then restarted without restarting the whole simulation, with any edits to the template taking effect at the next start. The images and templates can also be replaced any time through UWM - again, they will be used the next time a node is started.

In all cases not covered here, Google is your friend, and the community forums might contain helpers as well .