Running Labtainers from within GNS3

March 4, 2020

Contents

1 Introduction

This document describes the integration of GNS3 with Labtainers. The goal of this integration is to allow lab exercises developed for Labtainers to run within the GNS3 environment. From the student's perspective, they see the GNS3 GUI and select their lab as a standard GNS3 project. Labtainer components will be visible in the GNS3 workspace along with any other components defined in the lab, e.g., GNS3 routers. Pressing the "play" button starts that lab, and any Labtainers terminal windows that were defined for the lab will open as the components come online.

All components within the running lab are known to GNS3 as standard GNS3 nodes. Those that happen to be based on Labtainers containers will call-back into Labtainers modules, e.g., to open new terminals, or perform configuration steps during startup. All component networking is via GNS3 network management.

The lab development workflow is summarised as follows:

- Develop a lab using standard Labtainers tools and techniques, ignoring true routers, switches and such.
- Run a script that creates GNS3-Labtainers instances of the lab's Docker images
- Use GNS3 to lay out the topology of the newly created images, along with other devices such as routers.
- Run a script to automatically configure the GNS3 network configurations to match those defined for Labtainers
- Restart GNS3 and test the lab
- The new "lab" then consists of the GNS3 project directory and the Labtainers lab directory.

2 Requirements

It is assumed your development system is a recent Linux distribution. You will require these packages:

- pthon3-setuptools
- python3-dev
- python3-pyqt5
- g++

Install GNS3 from their website in order to get the ubridge software property configured.

`https://docs.gns3.com/1QXVIihk7dsOL7Xr7Bmz4zRzTsJ02wklfImGuHwTlaA4/index.html`

Add yourself to the ubridge group and logout and in to get permissions set per GNS3 guidance.

3 Cloning and installing GNS3 and Labtainers

3.1 Labtainers repo (optional)

This subsection is optional, and is intended to give access to GNS3 updates to Labtainers directly from github without waiting for formal Labtainers releases.

Get the gns3 branch from Labtainers:

```
git clone https://github.com/mfthomps/labtainers.git
cd labtainers
git checkout gns3
```

Define the `LABTAINERS_DIR` environment variable to point to the Labtainers directory. Also define $PATH to include '/bin' and $LABTAINER_DIR/scripts/designer/bin (Log out and in, or create a new bash shell to take effect.) Create the capinout executable (Ignore the warning messages):

```
cd $LABTAINER_DIR/tool-src/capinout
./mkit.sh
```

Run the docker install script for your operating system located in $LABTAINERS_DIR/setup_scripts to install docker and other packages to run Labtainers. Ex. For Ubuntu developers run install-docker-ubuntu.sh.

Run any Labtainers lab from the labtainer-student directory to test that it works and to capture a Labtainers email identifier.

3.2 GNS3 Repos

Get the GNS3 server from the forked github repo. First change your directory to where you want the repos to exist. Then:

```
git clone --single-branch --branch labtainers https://github.com/mfthomps/gns3-server.git
cd gns3-server
sudo python3 setup.py install
```

Change directory to where you want the repo, and then get the GNS3 GUI from the gns3 github repo:

```
git clone --single-branch --branch labtainers  https://github.com/mfthomps/gns3-gui.git
cd gns3-gui
sudo python3 setup.py install
```

3.2.1 Updating local git repos

Note below that your Labtainers repo uses the "gns3" branch – and your GNS3 repo uses the "labtainers" branch. From the local GNS3 repo:

```
git pull origin labtainers
```

From the local Labtainers repo:

```
git pull origin gns3
```

After pulling, always run `sudo python3 setup.py install` to get the latest build of GNS3. Do this as well after making local edits to the source code to see changes in the build.

3.2.2 Updating remote repo

This step is only for use by Labtainers framework developers. The GNS3 repo:

```
git push --set-upstream origin labtainers
```

The Labtainer repo:

```
git push --set-upstream origin gns3
```

4 Starting GNS3

Run the server from a terminal with LABTAINER_DIR defined.

```
gns3server --local --log /tmp/gns3log
```

The server does not display to stdout. Tail the /tmp/gns3log to see status.
Run the gui from a different terminal or tab:

```
gns3
```

Use the `-s` option to run as a student (to hide unused GUI objects, unused popup menu items and nodes and links that are to be hidden from the student.) Optionally provide the path to the gns3 project file to avoid the project dialog.

5 Porting a Labtainers lab

This example illustrates porting the telnetlab from Labtainers to run in the GNS3 environment. It is assumed the gns3-server has been started. This workflow can be further automated to automatically generate network connections. Or to fully define the project file based on predefined templates.

These steps assume the Labtainer Docker images exist on the machine, e.g., you've peformed a rebuild.py. If not, run the Labtainers lab to cause them to be pulled.

If there is a logo that you wish to appear on the GNS3 display, put that file in

```
<lab>/config/logo.png
```

Clicking on the resulting logo will display the text found in `<lab>/config/about.txt`. Go to the `$LABTAINER_DIR/scripts/gns3` directory:

```
cd $LABTAINER_DIR/scripts/gns3
```

Create modified Docker images for the lab:

```
./noNet.py telnetlab
```

NOTE: Make note of the container image names displayed. You will use these when adding appliance templates below.

View the Labtainer network topology.

```
./showNet.py telnetlab
```

Use the GNS3 gui to define GNS3 Docker container templates for each of the images you created, and to then use those templates to create a GNS3 instance of the lab.

- Start the gui, e.g., run `gns3`.

  Note: Starting the gns3 program with the student flag, e.g.,

  ```
  gns3 -s
  ```

  will cause the GUI to hide toolbars and widgets that students should not interact with. The container state after quitting gns3 will also persist. Without the student flag a lab's containers state wil reset upon pressing start once.

- Open a new project, assigning the same name as the Labtainers lab, e.g., "telnetlab"

- Use `Browse all devices / Add appliance template` to add the new container images created via the `noNet.py` command (DO NOT select the original container image names, e.g., having "student" as a suffix). Accept all defaults, except set the number of "Adapaters" to the quantity displayed using `showNet.py` for each component.

- Drag each container image from the list on the left of the GUI onto the workspace. (Do not try to fix the component names, that will be done later).

- Use the gui's `Add a Link` function to connect each component per the output of `showNet.py`. If a network has more than 2 componenets, add an ethernet hub and connect the components to the ethernet hub.

- Close GNS3 and gns3-server. Run the `./genNet.py telnetlab telnetlab` command to generate GNS3 network connection files, to set the logo, and to fix the component names. You will not see the revised names until you restart the lab.

- Start the GUI and test the lab.

Changes made to Labtainers container images will be picked up by GNS3 the next time the GUI is started, assuming you re-run the noNet.py command. There is no need to redefine appliance templates.

6 Remote access to containers

See the *Lab Designer User Guide* section on "Remote access and control of Labtainers" for information on remote management and remote access to containers within a lab.

7 Simulated USB drives

This section outlines Labtainers/GNS3 support for simulating insertion of USB drives. The goal is for the udevd on selected containers to recognize and respond to these events, i.e., based on rules in `/etc/udev/rules.d`.

This scheme works by sharing the `/dev` directory between **selected** containers and the host. The containers are selected in the `start.config` file. (NOTE: even though the devices do not appear on the other containers, the other container udev rules will trigger, and such they should be deleted or replaced with different scripts.)

A summary of the steps are:

- Build and install the linux dummy_hcd kernel module.
- Create a simple script that uses modprobe to create the simulated device on the VM; and a script to remove it. Put these scripts in a subdirectory of the lab directory along with a disk image for the USB (i.e., the "backing file"). This can be created with dd and mkext3...
- Identify those scripts within the start.config
- Gizmo to mount the device on the container after it is *authorized* is TBD, but tractable, e.g., in the rules.d file?
- That is it.

7.1 Linux dummy_hcd module build and installation

These instructions were tested on an Ubuntu 18 VM.

- Get kernel source: `sudo aptitude install linux-source` (You can remove this package prior to minting the VM image)
- Use git clone to retrive two files some kind soul had posted:

```
git clone https://github.com/serianox/DKMS-dummy_hcd.git
```

- Copy those two files to a new directory created at `/usr/src/dummy_hcd-01/` on the VM.
- Use dkms to build and install the module, as sudo:

```
dkms add -m dummy_hcd -v 0.1
dkms build -m dummy_hcd -v 0.1
dkms install -m dummy_hcd -v 0.1
```

- Load the module with `modprobe dummy_hcd`
- Add `dummy_hcd` to /etc/modules so it loads on the next boot.
- Test by:
  - creating a backing file as file system

```
dd if=/dev/zero of=/tmp/usb.img bs=1k count=1k
mkfs.ext2 -F usb.img
```

  - Create the simulated device:

```
sudo modprobe g_mass_storage file=/tmp/usb.img \
   idVendor=0x1d6b idProduct=0x0104 iManufacturer=Myself \
   iProduct=VirtualBlockDevice iSerialNumber=123
```

  - Check that it exists with `lsusb`
  - Delete device with `sudo modprobe -r g_mass_storage`

7.2 Configure the lab

- Create a subdirectory of your lab, e.g., `host_data` into which you will place a few scripts and the disk image.
- Use the two modprobe commands above as templates for creating two scripts, one for creating the device, and one for removing it (for use when stopping the lab so the device does not persist.
- Set sudoers to not require a password to run the modprobe script. Add something like the following using `visudo -f /etc/sudoers`

  ```
  mike ALL=(root) NOPASSWD: /home/mike/git/Labtainers/labs/usbtest/host_data/usb_crea
  ```

  and do the same for the script that deletes the device on stop.
- Disable auto mount in the VM (TBD, do these gsettings work?):

  ```
  gsettings set org.gnome.desktop.media-handling automount false
  gsettings set org.gnome.desktop.media-handling automount-open false
  ```

7.3 Implementation notes

Running the modprobe command on the target container has little advantage because the results are not seen on the container unless the /dev directory is shared between the container and the host. In which case, all containers having shared /dev will see it. However you can limited the sharing of /dev to only those containers that have dynamic USB requirements. Those with static "devices" can be mapped at the time the container is created. (TBD: add option to map selected devices between host and a container.) Also, running the `modprobe` on the host lets the backing store file live on the host, where its contents can be dynamically modified, e.g., to introduce malware into the usb drive.

- Perhaps put the mounting in a second udev rule, to run if the device persists?
- Student can insert a USB into a stopped container. What then is the iadrive supposed to do when the system starts?

8 GNS3 interface changes for Labtainers

Once the lab is defined for use in GNS3, it can be run by opening the associated project file. The following features were added to the GNS interface:

- Labtainers containers will be parameterized and initialized when the container is started. It requires a student email address in `/.local/share/labtainers/`
- Terminal windows defined for the Labtainers lab will be opened.
- Double clicking on a Labtainers component will open another Labtainers window similar to the `moreterm.py` function.
- The "check work" button will perform automated assessment similar to the Labtainers `checkwork` function.
- Lab manuals are displayed by clicking the "Lab Manual" button (question mark)

- The state of Labtainers containers is maintained after GNS3 is terminated. A new fresh instance of the lab can be generated by pressing the "Restart Lab" button.
- All Labtainers containers support X11 applications using the host system X11 server.
- Starting the gns3 program with the student flag will cause the GUI to hide toolbars and widgets that students should not interact with.
- Right clicking on a Labtainer node will display a pop-up menu including an option to insert a thumb drive. This will cause a `sudo mount` command to be run on the container, with arguments provided in start.config THUMB_VOLUME configuration value for that container.
- Use of the `-s` option when starting gns3 will cause any cloud endpoint nodes to be hidden from the student, along with any links to them. It will also hide any Labtainer containers having `HIDE YES` in the start.config.

9 History

This section describes steps taken to create the Labtainers version of GNS3. The steps outlined here are not intended to be repeated by developers.

The gns3 server was forked into https://github.com/mfthomps/gns3-server.git
The branch "2.1" was then cloned the tag set to tag v2.1.21:

```
git clone --single-branch --branch 2.1 https://github.com/mfthomps/gns3-server.git
git checkout tags/v2.1.21
```

A new "labtainers" branch was defined for the gns3 fork:

```
git tag v1.2.1-labtainers
git checkout -b labtainers
git push --set-upstream origin labtainers
```

The node.py script was altered to increase the timeout to account for parameterization of containers. This should be revisited to be perhaps less crude.

To add icons:

- Add icon to `gns3-gui/resources/images`
- Add ¡file¿images/yourIconFileName¡/file¿ to the list in resources.qrc located in `gns3-gui/resources`.
- Run gns3-gui/scripts/build_pyqt.py –ressources
- Install qtcreator.
- Open the `gns3-gui/gns3/gui/main_window.ui` file with qtcreator.
- Edit an object from the Action Editor bar, and click on the button with 3 dots on the 'Icon' row.
- Select your icon from the 'images' section, press OK, and press OK again.
- After saving your changes, run gns3-gui/scripts/build_pyqt.py –ressources.
- Run 'sudo python3 setup.py install' in the gns3-gui directory.

Added lab manual button and check work button.

10 Notes

The startup file is provided to start_window.py, and that is provided to loadPath, and that results in a call to Topology.loadProject. The nodes and links are created in graphics_view.py