8.4 Network interface assignments

Docker appears to assign network connections to containers in alphabetical order. E.g., connecting networks LAN1 and LAN2 to a container would result in LAN1 being connected to device eth0 – regardless of the order in which LANs are defined within the start.config file. Understanding this ordering may be helpful for networking labs, e.g., when defining routes.

9 Building, Maintaining and Publishing Labs

This section describes how labs are built, maintained and published. Additional information on tools and strategies intended for use by outside developers are described in section 10

Typically, when a Labtainer is started, the container's associated Docker images are pulled from the Docker Hub if they are not already local on the Linux host. When building and editing labs, the designer desires to run images reflecting recent changes that have been made. The framework includes logic to identify dependencies within containers whose image content has changed, and it will rebuild those images, (using the Docker build command). The framework will only rebuild those images that have changed. The designer can force the rebuild of all images within a lab by appending the "-f" switch to the end of the "rebuild.py" command. That switch is not intended for routine use because it wastes time and masks errors in our dependency logic.

If you build a new Labtainer exercise, the container images will not be on the Docker Hub unless you put them there. If you create your own public repository on the Docker Hub (https://hub.docker.com/), you can populate that with your lab(s) by setting the "REGISTRY_ID" value in the start.config file for the lab(s). You would then use the distrib/publish.py script to build, tag and push your lab container images to your registry. Please refer to the section 10.

9.1 NPS Development Operations

When building lab images at NPS, please set the LABTAINER_NPS environment variable to "YES", e.g.,

```
export LABTAINER_NPS=YES
```

This will force packages to be retrieved from the local NPS mirrors (centosmirror.uc.nps.edu or ubuntumirror.uc.nps.edu). Refer to section 9.4 for additional information. If builds fail with this environment set, it is likely due to trying to install packages not present in the mirror. In those cases, edit the Dockerfile to remove this line:

```
ENV APT_SOURCE $apt_source
```

That will force use of the original apt-sources for that container.

Labs must be checked into the local Git repository in order to be distributed. After creating and testing a new lab, use the scripts/designer/bin/cleanlab4svn.py script to remove temporary files that do not belong in git. Use the publish.py script (described above) to publish the lab containers. The distrib/mkdist.sh script is used by NPS to create the distribution tar file. This script relies on your local Git repository as the source to the Labtainer scripts and labs. Use the mk-devel-distrib.sh script to publish the developer configuration of the tar file.

The mkdist.sh and mk-devel-distrib.sh scripts include "myshare" variables that define a path to a directory shared with the development VM's host. The scripts will place the resulting tar files in this directory. You must then manually transfer the updated tar files (including the `labtainer_pdf.zip` file) to the Liferay server at