Reproducible Network Experiments using NixOS

Michael Hackl, Kilian Holzinger*, Henning Stubbe*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: michael.hackl@tum.de, holzinger@net.in.tum.de, stubbe@net.in.tum.de

***Abstract*—This is a technical report about making NixOS available on the Chair of Network Architectures and Services' testbeds. Our goal is to make it easy for conductors of network experiments to make their experiments reproducible. NixOS was chosen for its reproducible and declarative system and package management. In the end, we will have integrated NixOS into the chair's testbed infrastructure and conductors can choose to use a ready-made image of NixOS for their experiments.**

***Index Terms*—reproducibility, experiments, testbed, pos, NixOS**

1. Introduction

An important part of science is the verifiability of results. To promote verifiability, we want to help conductors make their experiments on the chair's testbeds more reproducible, i. e., the experiments can be replicated and will still have the same outcome.

While the reproducibility of an experiment has many facets, in this paper, we focus on the operating system: different researchers should be able to set up their machines to the same operating system state that the conductor had when he or she performed the experiment. There are two parts to this operating system state: the installed programs and the configuration. Both parts are dealt with in this paper.

For this reason, we will continue Zhou Lu's previous Bachelor's thesis "Reproducible Research Infrastructure with NixOS" [1], [2] by

- first giving an overview of the environment where the experiments take place, i. e., the testbeds of chair I8, and outlining NixOS from a reproducibility perspective,
- then showing how to make NixOS available on testbeds using pos,
- and finally thoroughly describing the implementation details of this process.

2. Background Information

All of the following information about the Chair of Network Architectures and Services' (I8) testbeds is from its wiki page [3].

2.1. Testbed Machines

Each testbed of the I8 testbeds consists of a management node and test nodes. Every authorized user can connect to the management node via SSH and, as the name suggests, manage the test nodes from there. The test nodes are bare-metal servers on which the experiments can be executed. Some are connected among each other with specified network links, over which experiments can be run. They do not have an operating system installed on them.

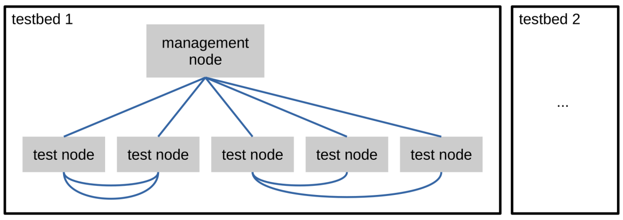

Figure 1: A simplified representation of the structure of the testbeds.

2.2. pos

The management and test nodes work together with the help of pos [4]. pos, which stands for "Plain Orchestrating Service", is the main tool for conducting experiments on the testbeds. It is used for

- managing the access to the test nodes between the users through allocations and reservations with a calendar,
- managing test nodes in terms of powering the test nodes on or off and providing them an operating system to boot,
- configuring test nodes with the parameters of an experiment,
- conducting an experiment on test nodes by executing a script or individual commands on them,
- and gathering the results and artifacts of an experiment.

pos consists of three main components:

- The *pos daemon (posd)* is running on the management node and manages the experiments. It provides a REST (representational state transfer) API for the other components.
- posd can be controlled with *poslib and pos-cli* from the management node. poslib is a python library and pos-cli is a command line interface. pos-cli uses poslib internally.
- *postools* is available on the test nodes and does the communication with posd, e. g. synchronizing

the test nodes with checkpoints and taking care of other common useful tasks for experiments. postools can be used as a python library or as a command line interface.

2.3. Operating System Images

The management node can control the test nodes per IPMI (Intelligent Platform Management Interface). This interface allows posd to do low-level (independent of an operating system) tasks such as powering the machines on. The test nodes then boot by means of network booting (Preboot eXecution Environment, PXE), i. e., they use files from the network instead of from a built-in storage. These files are provided by pos. pos allows us to choose the operating system which it supplies the test nodes with, e. g. Debian Bullseye. The operating systems, however, are so-called live boot operating systems, i. e., they do not need to be installed before they can be used.

New operating systems, such as NixOS in our case, can be added to pos in the form of images. An image is a directory that contains the following files:

- `vmlinuz`: the Linux kernel,
- `initrd.img`: the initial root file system image,
- and (optionally) `bootparameters.yml`: the kernel boot parameters that pos should add[1].

Such an image needs to be stored in `/srv/testbed/images/staging` on the management node, which is the place where every user is allowed to add new images.

The standard way of creating images for the I8 testbeds is using mandelstamm [6], a collection of scripts, on the so-called builder image. It supports different operating systems via a module system. The main script `build.sh` executes the desired module (which is just another script) and then packs the result into an image.

2.4. Bootstrapping

After a test node boots, it needs to be set up. pos does this by executing the python program `host.py` [7] on the test node through SSH. It sets the configured environment variables, installs postools, and updates the SSH keys.

This program is not compatible with NixOS as is and needs to be adapted to work correctly.

2.5. Motivation for the Choice of NixOS

There are already several versions of Debian and Ubuntu available for use as an operating system on the test nodes. Also, by using network booting, the test nodes do not store state between the experiments and therefore already incentivize making experiments reproducible. In this section we explore why it is worth it nonetheless to make NixOS available on the testbeds and what it can improve on the current situation.

NixOS [8] is a Linux distribution based on the Nix package manager. Both the system configuration (e. g. the `/etc` directory) and the package management, are handled in a purely functional language called Nix expressions. These Nix expressions describe derivations, which are tasks that define everything that is needed in order to build a package. The Nix Packages collection (nixpkgs) [9] contains Nix expressions for many commonly used packages. By specifying the git revision of the Nix Packages collection repository, the versions of all packages are clearly defined.

To install a package, the corresponding derivation has to be realized. The output is then stored in a central place: the Nix store. When some packages are not needed anymore, e. g. old versions after an update, the Nix store can be cleaned of old and unused packages by calling the garbage collection.

NixOS uses source-based package management with a binary cache. That means that in contrast to, for example Debian, whose packages are distributed as binaries, its packages are distributed in source form. However, to save build time, NixOS can download pre-built binaries from a binary cache when the inputs of the derivation match the inputs of the cached version. This model is great for reproducibility because in case a package is not available anymore for download in the future, Nix can automatically build it again as long as its source code can be found. Since the Nix expressions are (supposed to be) deterministic, the resulting binary is identical.

The system configuration of NixOS is declaratively defined in the file `/etc/nixos/configuration.nix`. A changed configuration can be applied with the program `nix-rebuild`. This means that one has to share just this one file (and the files that are referenced from it) and others can reproduce the whole system configuration (kernel, system services, applications, configuration files, etc.) except for mutable state (e. g. the `/var` directory). This is more efficient and less error-prone than using shell scripts and manual commands to configure the system.

3. Using NixOS in pos Testbeds

The general command to specify which operating system image to use is

```
pos nodes image <node> debian-bullseye
```

when using one of the provided images, e. g. Debian Bullseye. To apply this choice, we subsequently need to restart the test node:

```
pos nodes reset <node>
```

If we want to use a self-made image, we need to add the staging argument to our above command:

```
pos nodes image --staging <node> <image>
```

But we may not need to build the NixOS images ourselves: mandelstamm-ci [10] is a program that builds images for the testbeds using mandelstamm at predefined intervals. In case we want to build the image ourselves anyway, e. g. because we want to change the configuration for the image beforehand or mandelstamm-ci does not build the images for our testbed, we can do it as follows in the "builder-bullseye" image:

```
MANDELSTAMM_TARGET=copy mandelstamm/build.sh
↪  mandelstamm/modules/nixos-22.11.sh <image
↪  name>
```

The NixOS image is already preconfigured like the other images on the testbeds, e. g. useful programs are installed, and the timezone is set correctly.

1. This is currently not disclosed in the wiki [3], but can be read about in an issue of the pos daemon project [5].

4. Implementation Details and Contributions

Our goal is to make NixOS available on the test nodes for experiments.

We base our work on Lu's Bachelor's thesis [1], which explains how to make a NixOS image and provide it to pos and describes some changes to pos and the NixOS image in order to make pos and NixOS compatible with each other so that pos can carry out its tasks, which we cover in Section 4.1 and Section 4.2 respectively.

The following are our contributions: We extend the changes to pos and the NixOS image in Section 4.2. Then we automate the build process of the NixOS image with mandelstamm in Section 4.3 and add it to mandelstamm-ci in Section 4.4. Eventually, we add some guidance on the usage of NixOS in the testbeds' wiki in Section 4.5.

4.1. Building a Basic NixOS Image

Nixpkgs already provides a way to build PXE images of NixOS [8], [9]. In the name of easier reproducibility, we use this official way of creating our NixOS image instead of building it with mandelstamm (though later we will add a feature to mandelstamm that allows external programs—in this case `nix-build`—to build images). This means that our first step is to install the Nix package manager:

```
apt install nix-setup-systemd
```

We use the existing package manager to install Nix instead of piping the contents of the official installer URL to bash in order to save the manual creation of a non-root user account, have better compatibility with Debian, etc. After that, we clone the nixpkgs repository [9], which contains the build instructions:

```
git clone --depth 1
↪  https://github.com/NixOS/nixpkgs.git
↪  --branch nixos-22.11
```

To finally build a NixOS image, we then run:

```
nix-build -A netboot.x86_64-linux
↪  nixpkgs/nixos/release.nix
```

This yields us three files, two of which—`bzImage` and `initrd`—we just need to rename to `vmlinuz` and `initrd.img` respectively to fit as a pos image. The third file—`netboot.ipxe`—needs to be adapted to work with pos. It contains the kernel boot parameters (particularly "init") that are required for booting NixOS. To extract them from this file and write them to the `bootparameters.yml` file, we use the command from Figure A.4 in Lu's thesis [1]:

```
grep --regexp='.*init=.* initrd=initrd.*'
↪  <'netboot.ipxe' | sed 's/.*init=\(.*\)
↪  initrd=initrd.*/init: \1/'
↪  >'bootparameters.yml'
```

4.2. Changes to pos and the NixOS Image

Now the image in itself is done. But pos still needs to be expanded to be able to deal with NixOS and NixOS configured to work together with pos. Some of these changes have already been made (see Figures A.2 and A.3 in thesis [1] and merge request [11]), others we make ourselves (see merge requests [12], [13]).

4.2.1. NixOS Configuration.

4.2.1. NixOS Configuration. The NixOS image is assimilated to the existing OS images by including the same configuration that the module `common.sh` applies to all other mandelstamm images in the NixOS configuration. This involves installing likely useful programs, setting the hostname to be received via DHCP (Dynamic Host Configuration Protocol), setting the timezone to "Europe/Berlin", and adding SSH keys.

Furthermore, symbolic links are created in the image to the programs that posd and `host.py` (introduced in Sections 2.2 and 2.4 respectively) expected at certain locations. Python, for example, is linked to `/usr/bin/python`.

We modify the pos daemon to not expect python at a fixed location with `/usr/bin/python` but instead use the environment with `/usr/bin/env python`. The equivalent has already been done for postools [14]. This means that we can remove the creation of the symbolic links from the NixOS configuration.

4.2.2. NixOS Configuration File Location. The corresponding options for the aforementioned configuration are directly included in the base file for the image `netboot-minimal.nix` in the local nixpkgs repository.

This method of changing the configuration of the image has a problem: While the changes in the local repository do carry over to the image (which means that a rebuild retains this configuration), an update (`nix-channel --update`) overwrites them. This means that a subsequent `nixos-rebuild` resets everything.

To solve this, we move the whole configuration to a separate file called `testbed.nix`. We import this file for the build of the image and set that the file is included in the image and referenced from the NixOS configuration file `/etc/nixos/configuration.nix` in the image using the "configuration" argument of `release.nix`:

```
nix-build nixpkgs/nixos/release.nix -A
↪  netboot.x86_64-linux --arg configuration
↪  '{ pkgs, ... }: { imports = [
↪  ../files/testbed.nix ];
↪  installer.cloneConfigIncludes = [
↪  (pkgs.writeText "testbed.nix"
↪  (builtins.readFile ../files/testbed.nix))
↪  ]; }'
```

4.2.3. postools Installation. To accommodate the special way to install software of NixOS, the file `default.nix` [15] containing a Nix expression for building a Nix package of postools is added at `/srv/testbed/files/luz/default.nix` on the management nodes of two testbeds[2]. When bootstrapping a test node, the file is downloaded to the test node and used by `host.py` to install postools to the default profile (available in all user environments).

4.2.4. Hostname Correction. `host.py` needs the short hostname for the communication with pos, but in NixOS the hostname is set to the fully qualified domain name obtained over DHCP, e. g. "klaipeda.baltikum.net.in.tum.de". Therefore `host.py` only uses the part before the first dot of the hostname when bootstrapping NixOS.

2. This file was added to the management nodes "coinbase" and "kaunas" [11]. In the final section of this paper we suggest making this file no longer necessary as future work.

4.2.5. NixOS Identification.

The identification of NixOS in host.py for the decision on how to deal with the hostname and how to install postools fails, i. e., it does not execute the NixOS-specific statements, which causes the bootstrapping process to abort. We fix this by making the comparison that tests for NixOS case insensitive.

4.2.6. Direct Build of bootparameters.yml.

We can produce the bootparameters.yml file directly with Nix instead of adapting netboot.ipxe with sed. To do so, we overwrite the Nix derivation system.build.netbootIpxeScript (in the file netboot.nix [9]) using “mkForce” (explained in [8, Section 67.3.2. Setting Priorities]) to generate our bootparameters.yml file instead of the standard netboot.ipxe file:

```
system.build.netbootIpxeScript = lib.mkForce
↪  (pkgs.writeTextDir "bootparameters.yml" ''
  init: "${config.system.build.toplevel}/init"
'');
```

4.3. Automated Building With mandelstamm

While mandelstamm is not compatible with the way we want to build a NixOS image (with nix-build, see Section 4.1) as is, we will extend it accordingly.

mandelstamm has a feature that allows us to specify with the environment variable MANDELSTAMM_TARGET the format in which it should pack the image. The two options are “traditional” and “squashfs”. They were originally introduced to compress images that are too large. We add a third packing format “copy” to these options that just copies the files that a module created to the output directory, i. e., it does not pack them first as the other two options do. This allows for building an (already packed) NixOS image in a mandelstamm module.

As the building of images happens on the builder image, we add “nix-setup-systemd” to the packages to be installed there.

Next, we create the file testbed.nix (with the previously mentioned content, see Section 4.2.1) and a new module for NixOS for versions 22.05 and 22.11 respectively in the mandelstamm repository. These new modules contain the commands for building NixOS that we already discussed. The noteworthy points here are that we do not add the result of the nix-build command as a root of the garbage collector with the option --no-out-link and that we call nix-store --gc at the end to collect the garbage. We do this because, unlike when building the other images, where everything happens in a tmpfs (Temporary File System), there are files left over after building NixOS, namely in the Nix store.

Furthermore, we automatically add the SSH key for pos, which is found in a git submodule of the mandelstamm repository, to the NixOS configuration so that pos is allowed to log in.

Eventually, we bundle all this in merge request [12]. Now NixOS images can be built using mandelstamm in the “builder-bullseye” image using the command shown in Section 3.

4.4. Adding NixOS to mandelstamm-ci

Because we implemented the creation of a NixOS image in mandelstamm and mandelstamm-ci works together well with mandelstamm, we can easily automate the building process with mandelstamm-ci: With merge request [16] we add an entry for our build module to the configuration file of mandelstamm-ci mandelbauer-config.yaml to make it build a NixOS image regularly and we specify the packing format for this build to be “copy”.

4.5. Adding Instructions to the Testbeds’ Wiki

At the very end, we add instructions for using NixOS in the testbeds to the testbeds’ wiki [17].

5. Conclusion

NixOS is now available on the chair’s testbeds as an automatically built image and can also be built on other testbeds using pos by following the same procedure. That means that conductors can now easily choose NixOS for their experiments and make their experiments more reproducible this way.

To allow others to reproduce the NixOS environment for their experiments, conductors only have to provide the nixpkgs git revision their system was built with and their configuration.

Last, we suggest some judicious changes that we do not implement but leave for future work:

- Provide a universal way of installing postools so that bootstrapping is possible on all testbeds without needing to manually deploy a file once on each testbed beforehand. This could be done by including the Nix expression for postools directly in the NixOS configuration at the image generation instead of installing postools during the bootstrap process.
- The command nixos-rebuild test applies changes to the system even when using the unchanged configuration file from the image. Find out why this happens and address it.
- The command nixos-version, which shows among other things the git revision the system was built from, reports the dummy values from the release.nix file in [9]. Only after running nixos-rebuild test --upgrade it shows the correct version. The correct values should be provided from the beginning by giving them as arguments to release.nix in the build command.

References

[1] Z. Lu, “Reproducible Research Infrastructure with NixOS,” Bachelor’s thesis, Technical University of Munich, 2021.

[2] ——, “Accompanying GitLab Project to ‘Reproducible Research Infrastructure with NixOS’: NixOS for pos,” https://gitlab.lrz.de/netintum/teaching/tumi8-theses/ba-lu/nixpkgs, [Online; accessed 4-March-2023].

[3] “Orchestration of Testbeds at I8,” https://gitlab.lrz.de/I8-testbeds/wiki/-/wikis/home, [Online; accessed 4-March-2023].

[4] S. Gallenmüller, D. Scholz, H. Stubbe, and G. Carle, “The pos framework: A methodology and toolchain for reproducible network experiments,” in *Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT ’21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 259–266. [Online]. Available: https://doi.org/10.1145/3485983.3494841

[5] “Issue in the pos Daemon Project: Per image (additional) default boot parameter,” https://gitlab.lrz.de/I8-testbeds/pos/daemon/-/issues/209, [Online; accessed 4-March-2023].

[6] “mandelstamm Project,” https://gitlab.lrz.de/I8-testbeds/mandelstamm, [Online; accessed 4-March-2023].

[7] “host.py in the pos Daemon Repository,” https://gitlab.lrz.de/I8-testbeds/pos/daemon/-/blob/master/posd/nodes/boot/host.py, [Online; accessed 4-March-2023].

[8] “NixOS Manual,” https://nixos.org/manual/nixos/stable/index.html, [Online; accessed 4-March-2023].

[9] “Nixpkgs Repository on GitHub,” https://github.com/NixOS/nixpkgs, [Online; accessed 4-March-2023].

[10] “mandelstamm-ci Project,” https://gitlab.lrz.de/I8-testbeds/mandelstamm-ci, [Online; accessed 4-March-2023].

[11] Z. Lu, “Merge Request to the pos Daemon Repository: Nixos support,” https://gitlab.lrz.de/I8-testbeds/pos/daemon/-/merge_requests/323, [Online; accessed 4-March-2023].

[12] “Merge Request to the mandelstamm Repository: Add NixOS,” https://gitlab.lrz.de/I8-testbeds/mandelstamm/-/merge_requests/37, [Online; accessed 4-March-2023].

[13] “Merge Request to the pos Daemon Repository: Increase NixOS compatibility,” https://gitlab.lrz.de/I8-testbeds/pos/daemon/-/merge_requests/399, [Online; accessed 4-March-2023].

[14] “Issue in the postools Project: please use /usr/bin/env in shebangs,” https://gitlab.lrz.de/I8-testbeds/pos/tools/-/issues/24, [Online; accessed 4-March-2023].

[15] Z. Lu, “default.nix in the Accompanying GitLab Project to ‘Reproducible Research Infrastructure with NixOS’: NixOS for pos,” https://gitlab.lrz.de/netintum/teaching/tumi8-theses/ba-lu/nixpkgs/-/blob/master/pos_python_patch/default.nix, [Online; accessed 4-March-2023].

[16] “Merge Request to the mandelstamm-ci Repository: Add NixOS 22.11 to the build schedule,” https://gitlab.lrz.de/I8-testbeds/mandelstamm-ci/-/merge_requests/8, [Online; accessed 4-March-2023].

[17] M. Hackl, “NixOS,” https://gitlab.lrz.de/I8-testbeds/wiki/-/wikis/for-users/testbed-images/NixOS, [Online; accessed 5-March-2023].