Nix: Testing a Single NixOS Module in CI

Published by **Philipp Schuster** on 2023-02-02

In this blog post, I explain how to test a single NixOS module in CI. By single NixOS module, I mean a module that provides some options and configurations, but itself is not a valid NixOS configuration. The use case is to have a common module (in a git repository) that you want to integrate in a full NixOS configuration, when you don't have or want the full NixOS configuration at the same location.

Hint: *The article focuses on NixOS 22.11 and the regular, non-Flake way.*

Update: Thanks to Jacek Galowicz for providing me with a tip in the comments for an even simpler solution.

Let's take a look at the following custom module `latest-linux-kernel.nix`. Let's imagine that we have it in our "common NixOS configurations" GitHub repository and want to test its options and configurations for validity in CI:

```
{ pkgs, lib, config, options, ... }:

let
  cfg = config.foobar.common.latest-linux;
in
{
  options = {
    foobar.common.latest-linux.enable = lib.mkEnableOption "Use the latest stable Linux kernel";
  };

  config = lib.mkIf cfg.enable {
    # Use latest stable kernel.
    boot.kernelPackages = pkgs.linuxPackages_latest;
  };
}
```

In a real-world scenario, you would have multiple modules with one common entry module, but I omit this here for simplicity. How can you test the module(s) in CI, i.e., verify you didn't break any existing configuration options?

To test if a NixOS configuration builds, we can use `$ nixos-rebuild dry-build`, but we need a fully working `configuration.nix` for it. So what do we do? We can use a minimal

```nix
# Minimal configuration that is accepted by "nixos-rebuild" + import of
# modules to test.

{ config, pkgs, ... }:

{
  imports = [
    path-to-nix-module/latest-linux-kernel.nix
  ];

  # ---------------------------------------------------------------------------
  # test the properties from my NixOS Module
  foobar.common = {
    latest-linux.enable = true;
  };

  # ---------------------------------------------------------------------------

  nixpkgs.config.allowUnfree = true;
  system.stateVersion = "22.11";

  boot.loader.systemd-boot.enable = true;

  users.users.testuser.isNormalUser = true;

  # Some root file system so that nixos-rebuild doesn't fail.
  fileSystems."/" =
    {
      device = "/dev/disk/by-uuid/510da090-fb98-458e-86e1-bfd728741d02";
      fsType = "ext4";
    };
}
```

Background Information: How did I get to this configuration? I used an existing configuration and shrinked it down to a minimal version until `nixos-rebuild` failed.

To build the `configuration.nix` with `nixos-rebuild`, we need a modified `NIX_PATH` environmental variable, where `<nixos-config>` in `NIX_PATH` points to the `configuration.nix`. An even simpler approach (thanks Jacek!) than using `nixos-rebuild` is to use a `default.nix` with the following content. Note that I use a pinned version of `nixpkgs` here, which is important for build stability and reproducibility.

```nix
let
  nixpkgs =
    # Picked a recent commit from the "nixos-22.11-small"-branch.
    # If you change this, also alter the sha256 hash!
    let rev = "91111087ba0e0af9dcd76d754e5ef5ac59dd2b05";
    in
    builtins.fetchTarball {
      url = "https://github.com/NixOS/nixpkgs/archive/${rev}.tar.gz";
      sha256 = "sha256:07yllqlfq1wrnk2jsy9jgfd816hgif5k9bvf7fwslxyya62sm60d";
    };
  pkgs = import nixpkgs { };
```

`configuration.nix`, which will fail, if the module is not valid or a configuration option doesn't exist. If you like, you can create a script with a meaningful name to wrap this:

```
#!/usr/bin/env bash

DIR=$(dirname "$(realpath "$0")")
cd "$DIR" || exit

nix-build
```

An example configuration for GitHub's CI may look like this:

```
name: Build

on: [ push, pull_request ]

jobs:
  # Tests that a configuration.nix can include common NixOS modules.
  test_nixos_cfg:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: cachix/install-nix-action@v19
        with:
          nix_path: nixpkgs=channel:nixos-22.11
      - run: path/into/git/repo/test-build.sh
```

That's it. A working example can be found in my dotfile repository. Got any questions? Let me know!

Categories: PROGRAMMING & DEVOPS

Tags: CI Github Nix NixOS

Philipp Schuster

Hi, I'm Philipp and interested in Computer Science. I especially like low level development, making ugly things nice, and de-mystify "low level magic".

2 Comments

Jacek · 2023-02-04 at 11:06

```
 let
pkgs = import ... {};
config = pkgs.nixos [ ./configuration.nix ];
in
config.config.system.build.toplevel
```

This way you don't need to tamper with environment variables and it becomes very easy to have multiple configs checked against multiple versions of nixpkgs. This way you can check if your module works with stable and unstable nixpkgs for example.
The CI command then reduces to `NIX_PATH= nix-build oneNixFileWithAllConfigs.nix`

REPLY

Philipp Schuster · 2023-02-06 at 19:23

Thanks a lot for the suggestion! I'll look into it.

REPLY

Leave a Reply

Name *

Email *

Website

What's on your mind?

☐ Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Because this blog is a few years old and used to be a more or less personal german blog, old posts are in german. Newer posts are in english.

Related Posts

PROGRAMMING & DEVOPS

Remotely deploying NixOS configuration using SSH Jump Host

nixos-rebuild --target-host <user@host> is a powerful tool to remotely deploy a NixOS configuration. However, there are scenarios where you can't reach a machine directly but need an SSH jump host. Suppose you want to deploy Read more…

PROGRAMMING & DEVOPS

Enabling TP-Link Archer TX20U Nano on NixOS and Linux 6.12

Recently, I've bought a TP-Link Archer TX20U Nano WiFi USB dongle (vendor website) for my desktop PC. The driver for Windows is provided by the stick itself, as it exposes a storage volume with the Read more…

PROGRAMMING & DEVOPS

Live-Migration of QEMU/KVM VMs with libvirt: Command Cheat Sheet and Tips

Since recently, I am working with libvirt to perform live-migration VMs with QEMU/KVM VMs (QEMU as VMM and KVM as hypervisor). A few things were not very clear nor well documented. Therefore, I'd like to Read more…

PHIPS BLOG