Run NixOS Integration Tests on macOS

March 8, 2024 by Jacek Galowicz

Big news: You can now run NixOS integration tests on macOS without any changes to the test! This article helps you set up your Mac and get going.

If this does not appear significant to you, chances are that you haven't seen what the NixOS integration test driver is capable of. In that case, you might want to peek into the article that highlights some of the most impressive integration tests of NixOS's `nixpkgs` repository and the article about how the NixOS integration test driver works, first.

Last week, Gabriella Gonzalez upstreamed changes on nixpkgs that made the NixOS integration test driver finally run on macOS, too. This means that NixOS integration tests still run normal NixOS VMs, but the qemu processes and the Python test driver run natively on macOS.

These changes require the `nixos-test` and `apple-virt` capability flags set in Nix, which is going to be auto-detected by Nix 2.19 and newer.

Continue to read this article to see how to get it to run on *your* Mac today.

Setting up macOS for NixOS Integration Tests

Of course, we need to install Nix first. If you haven't, please have a look at the article about how to install Nix and nix-darwin on macOS.

Then, to run NixOS integration tests on macOS, we need *two* prerequisites:

1. A Linux Builder (Can be a remote Linux machine, or the local `linux-builder`)
2. The `system-features` Nix capability flags need `nixos-test` and `apple-virt` (Automatically detected on Nix 2.19 and newer)

The Linux builder could be any remote Linux machine that is configured as a remote builder or the macOS `linux-builder` that already comes with nixpkgs. As we have seen in the article about the `linux-builder`, it is tedious to install it manually but very straightforward to do it with `nix-darwin`. Hence, we strongly suggest configuring nix-darwin first as explained in this article, and then coming back to follow the next steps.

> *The alternative way to configure this without nix-darwin would be manually editing `/etc/nix/nix.conf` and setting up the Linux builder manually using the steps described in the Linux builder article.*

The **`nix-darwin`** config snippet that performs both changes looks like this:

```
# configuration.nix
{ config, pkgs, lib, ... }:

{
  # Run the linux-builder as a background service
  nix.linux-builder.enable = true;

  # Add needed system-features to the nix daemon
  # Starting with Nix 2.19, this will be automatic
  nix.settings.system-features = [
    "nixos-test"
    "apple-virt"
  ];
}
```

Running **`darwin-rebuild switch --flake <path/to/my/config>`** with these added configuration attributes makes our Mac ready to go.

After everything goes well, we can check if everything is in order:

```
$ nix show-config system-features
apple-virt apple-virtualization nixos-test

$ sudo launchctl list org.nixos.linux-builder
{
  "LimitLoadToSessionType" = "System";
  "Label" = "org.nixos.linux-builder";
  "OnDemand" = false;
  "LastExitStatus" = 0;
  "PID" = 604;
  "Program" = "/bin/sh";
  "ProgramArguments" = (
    "/bin/sh";
    "-c";
    "/bin/wait4path /nix/store && exec
/nix/store/ab8iafqq7x0r13dmsy82q99kddvwrarp-linux-builder-start";
  );
};
```

We are ready to go. Please note that the default Linux builder settings do not yet provide high performance. To increase the performance of the Linux builder, have a look at the Linux builder article, where we crank them up.

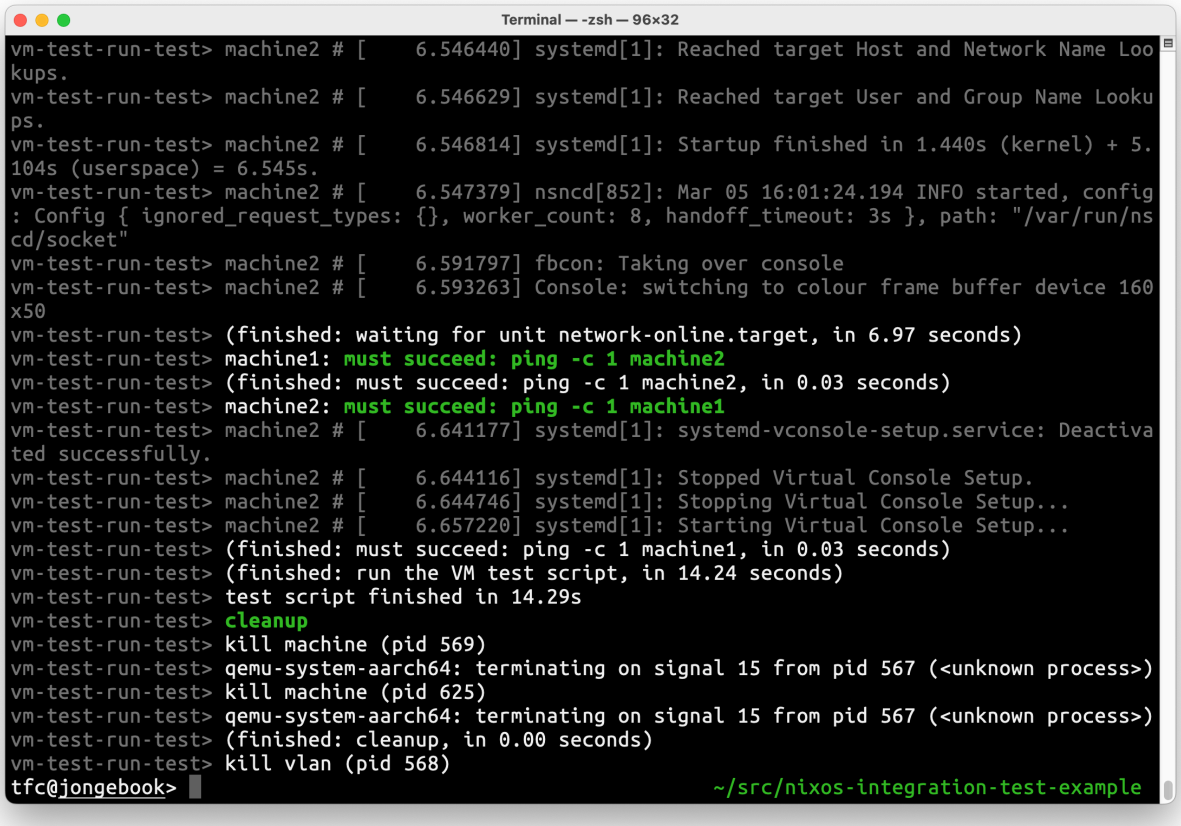
Run an Integration Test

To check if this works, we can now simply run:

```
$ nix -L build github:tfc/nixos-integration-test-example
```

> *We can also run one of the other ~700 official NixOS integration tests by running* **`nix -L build nixpkgs#legacyPackages.aarch64-darwin.nixosTests.login`**. *Please note that some/many of them might not work as macOS support for the test driver is new and they have not been tested on macOS, yet.*

After some download and build time, You will see a lot of Linux boot and service log messages scroll by and end up with the output of a successful test:


Terminal — -zsh — 96×32

```
vm-test-run-test> machine2 # [    6.546440] systemd[1]: Reached target Host and Network Name Loo
kups.
vm-test-run-test> machine2 # [    6.546629] systemd[1]: Reached target User and Group Name Looku
ps.
vm-test-run-test> machine2 # [    6.546814] systemd[1]: Startup finished in 1.440s (kernel) + 5.
104s (userspace) = 6.545s.
vm-test-run-test> machine2 # [    6.547379] nsncd[852]: Mar 05 16:01:24.194 INFO started, config
: Config { ignored_request_types: {}, worker_count: 8, handoff_timeout: 3s }, path: "/var/run/ns
cd/socket"
vm-test-run-test> machine2 # [    6.591797] fbcon: Taking over console
vm-test-run-test> machine2 # [    6.593263] Console: switching to colour frame buffer device 160
x50
vm-test-run-test> (finished: waiting for unit network-online.target, in 6.97 seconds)
vm-test-run-test> machine1: must succeed: ping -c 1 machine2
vm-test-run-test> (finished: must succeed: ping -c 1 machine2, in 0.03 seconds)
vm-test-run-test> machine2: must succeed: ping -c 1 machine1
vm-test-run-test> machine2 # [    6.641177] systemd[1]: systemd-vconsole-setup.service: Deactiva
ted successfully.
vm-test-run-test> machine2 # [    6.644116] systemd[1]: Stopped Virtual Console Setup.
vm-test-run-test> machine2 # [    6.644746] systemd[1]: Stopping Virtual Console Setup...
vm-test-run-test> machine2 # [    6.657220] systemd[1]: Starting Virtual Console Setup...
vm-test-run-test> (finished: must succeed: ping -c 1 machine1, in 0.03 seconds)
vm-test-run-test> (finished: run the VM test script, in 14.24 seconds)
vm-test-run-test> test script finished in 14.29s
vm-test-run-test> cleanup
vm-test-run-test> kill machine (pid 569)
vm-test-run-test> qemu-system-aarch64: terminating on signal 15 from pid 567 (<unknown process>)
vm-test-run-test> kill machine (pid 625)
vm-test-run-test> qemu-system-aarch64: terminating on signal 15 from pid 567 (<unknown process>)
vm-test-run-test> (finished: cleanup, in 0.00 seconds)
vm-test-run-test> kill vlan (pid 568)
tfc@jongebook>                                               ~/src/nixos-integration-test-example
```

NixOS integration tests are now easy to run on macOS and nearly as fast as on Linux

For everyone who didn’t have time to study our other NixOS integration test-related articles - this is a little overview of how to assemble your own test to play with it yourself (this is essentially the same test that just ran):

We start with a **`flake.nix`** file in our repository that collects all the inputs and calls a **`test.nix`** file. It doesn’t have to be in a separate file, but the **`runNixOSTest`** pattern is most pleasant this way.

```
# flake.nix
{
  description = "Example NixOS Integration Tests";

  inputs = {
    nixpkgs.url = "github:NixOS/nixpkgs/nixos-unstable";
    flake-parts.url = "github:hercules-ci/flake-parts";
    flake-parts.inputs.nixpkgs-lib.follows = "nixpkgs";
  };

  outputs = inputs: inputs.flake-parts.lib.mkFlake { inherit inputs; } {
    systems = [
      "x86_64-linux" "aarch64-linux" "aarch64-darwin" "x86_64-darwin"
    ];
    perSystem = { config, pkgs, ... }: {
      packages.default = pkgs.testers.runNixOSTest ./test.nix;

      # put the packages into `checks` so `nix flake check` runs them,
too
      checks = config.packages;
    };
  };
}
```

The **`test.nix`** file describes our test name, the VM network, and finally also the actual test:

```
# test.nix
{
  name = "An awesome test.";

  nodes = {
    machine1 = { pkgs, ... }: {
      # Empty config sets some defaults
    };
    machine2 = { pkgs, ... }: { };
  };

  testScript = ''
    machine1.wait_for_unit("network-online.target")
    machine2.wait_for_unit("network-online.target")

    machine1.succeed("ping -c 1 machine2")
    machine2.succeed("ping -c 1 machine1")
  '';
}
```

With these two files in a local folder, we can simply run **`nix -L build`** to run the test. I suggest the **`-L`** parameter in the beginning because the log output is interesting to watch while discovering what the NixOS integration test driver is capable of.

What do do from here? Read part 1 and part 2 of our article series on the NixOS integration test driver to get a taste of what it is capable of. Then, have a look at the NixOS integration test driver documentation. Have fun!

Debugging Mode

What to do when a test fails? Change something, watch complex minutes-running tests run from beginning to end all over again and again, and hope that the latest change finally fixes it? Surely not.

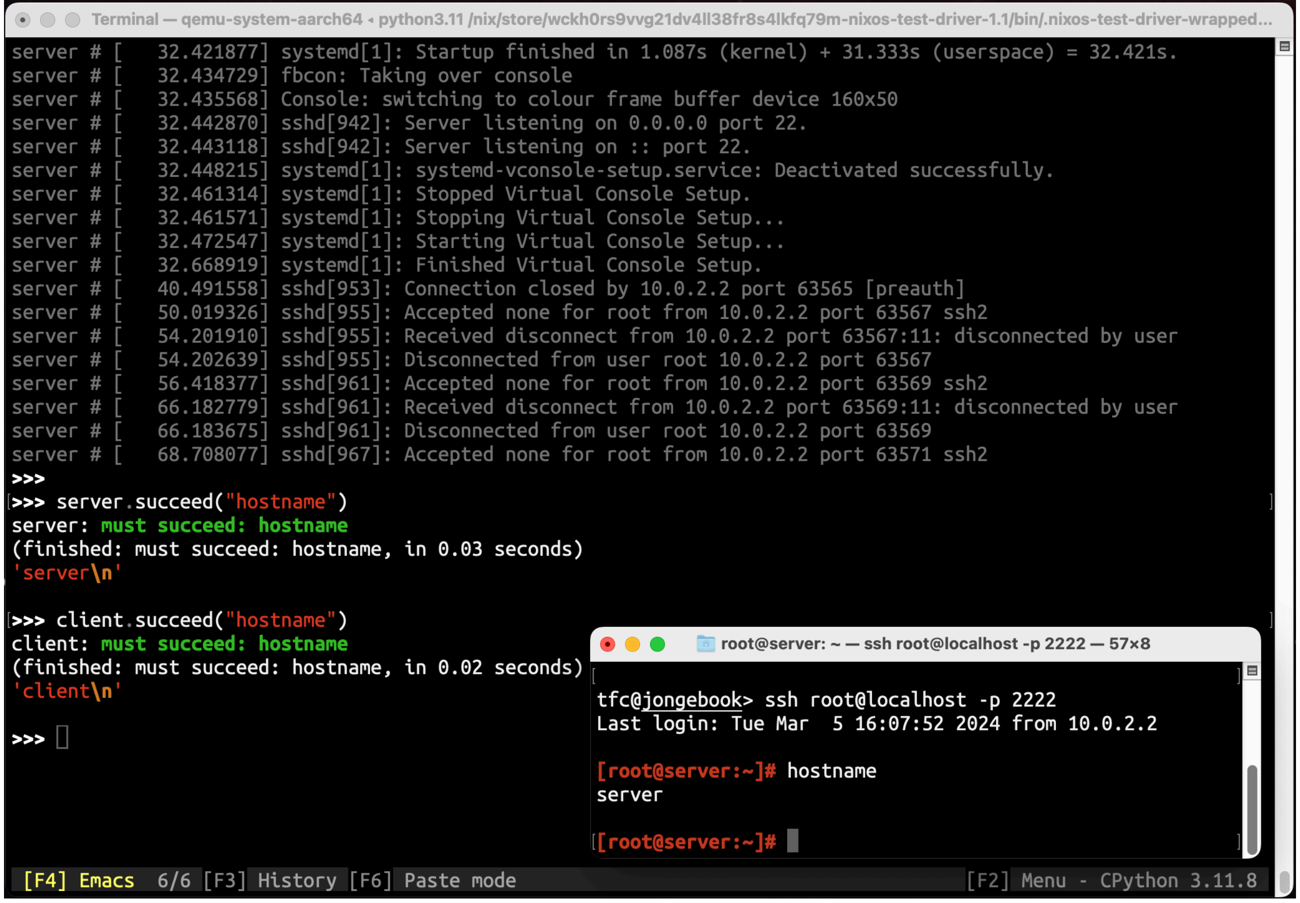
Debugging our Integration tests, as demonstrated in the last article, works right out of the box.

We can run the NixOS integration test driver outside of Nix in the interactive mode:

```
$ nix run .#echo.driverInteractive -- --interactive
warning: Git tree '/Users/tfc/src/nixos-integration-test-example' is
dirty
Machine state will be reset. To keep it, pass --keep-vm-state
start all VLans
start vlan
running vlan (pid 3680; ctl /tmp/vde1.ctl)
(finished: start all VLans, in 0.00 seconds)
additionally exposed symbols:
    machine1, machine2,
    vlan1,
    start_all, test_script, machines, vlans, driver, log, os,
create_machine,
    subtest, run_tests, join_all, retry, serial_stdout_off,
serial_stdout_on,
    polling_condition, Machine
>>>
```

In the Python shell, we can use any of the functionality of the integration test driver (see also the documentation).

The test definitions in the example repository from the last article contain additional configuration that is only active in the interactive mode. With that, the `server` VM of the *echo* test (same repo) for example, is also reachable via SSH on the host port 2222, as shown in this screenshot:


Terminal — qemu-system-aarch64 ◂ python3.11 /nix/store/wckh0rs9vvg21dv4ll38fr8s4lkfq79m-nixos-test-driver-1.1/bin/.nixos-test-driver-wrapped...
root@server: ~ — ssh root@localhost -p 2222 — 57×8

```
server # [   32.421877] systemd[1]: Startup finished in 1.087s (kernel) + 31.333s (userspace) = 32.421s.
server # [   32.434729] fbcon: Taking over console
server # [   32.435568] Console: switching to colour frame buffer device 160x50
server # [   32.442870] sshd[942]: Server listening on 0.0.0.0 port 22.
server # [   32.443118] sshd[942]: Server listening on :: port 22.
server # [   32.448215] systemd[1]: systemd-vconsole-setup.service: Deactivated successfully.
server # [   32.461314] systemd[1]: Stopped Virtual Console Setup.
server # [   32.461571] systemd[1]: Stopping Virtual Console Setup...
server # [   32.472547] systemd[1]: Starting Virtual Console Setup...
server # [   32.668919] systemd[1]: Finished Virtual Console Setup.
server # [   40.491558] sshd[953]: Connection closed by 10.0.2.2 port 63565 [preauth]
server # [   50.019326] sshd[955]: Accepted none for root from 10.0.2.2 port 63567 ssh2
server # [   54.201910] sshd[955]: Received disconnect from 10.0.2.2 port 63567:11: disconnected by user
server # [   54.202639] sshd[955]: Disconnected from user root 10.0.2.2 port 63567
server # [   56.418377] sshd[961]: Accepted none for root from 10.0.2.2 port 63569 ssh2
server # [   66.182779] sshd[961]: Received disconnect from 10.0.2.2 port 63569:11: disconnected by user
server # [   66.183675] sshd[961]: Disconnected from user root 10.0.2.2 port 63569
server # [   68.708077] sshd[967]: Accepted none for root from 10.0.2.2 port 63571 ssh2
>>>
>>> server.succeed("hostname")
server: must succeed: hostname
(finished: must succeed: hostname, in 0.03 seconds)
'server\n'

>>> client.succeed("hostname")
client: must succeed: hostname
(finished: must succeed: hostname, in 0.02 seconds)
'client\n'

>>> 
 [F4] Emacs  6/6 [F3] History [F6] Paste mode                    [F2] Menu - CPython 3.11.8
```

```
tfc@jongebook> ssh root@localhost -p 2222
Last login: Tue Mar  5 16:07:52 2024 from 10.0.2.2

[root@server:~]# hostname
server

[root@server:~]# 
```

This screenshot shows the interactive test driver for the **echo** test from the last article. It has two VMs.

This way we can avoid having the full test run again and fail. Instead, we discover what works on the running interactive VMs.

Summary

Being able to build Linux systems and also test them in the NixOS integration test driver on macOS is a big enabler. This is especially true in the corporate context where the big IT departments supply employees with either Windows laptops or Macs.

The VMs are a bit slower on macOS, but with more users and some debugging, this performance difference will most probably vanish over time.

Are you interested in unlocking this power at scale in your corporate environment? We helped many companies get up to speed with Nix by both training teams and setting up Nix build and cache infrastructure in ways that fit into the existing IT landscape. We have helped companies win with Nix and know what the winning setups look like. Schedule a free call with us today!

NIXCADEMY

If you are strategically investing in Nix and NixOS, looking for NixOS training and mentoring, and/or consulting, then the Nixcademy is your place to go!

SERVICES

Nix & NixOS 101 Training

Advanced Nix Trainings

Individual Mentoring

Consulting

Blog

USEFUL LINKS

Nix & NixOS Cheatsheet

Impressum / Imprint

Datenschutz / Data Protection

CONTACT

hello@nixcademy.com

Schedule Meeting

+ 49 1523 7191800