Integration testing with NixOS virtual machines

Contents

What will you learn?

This tutorial introduces Nixpkgs functionality for testing NixOS configurations. It also shows how to set up distributed test scenarios that involve multiple machines.

What do you need?

- A working Nix installation on Linux, or NixOS
- Basic knowledge of the Nix language
- Basic knowledge of NixOS configuration

Skip to main content

Introduction

Nixpkgs provides a test environment to automate integration testing for distributed systems. It allows defining tests based on a set of declarative NixOS configurations and using a Python shell to interact with them through QEMU as the backend. Those tests are widely used to ensure that NixOS works as intended, so in general they are called NixOS Tests. They can be written and launched outside of NixOS, on any Linux machine[1].

Integration tests are reproducible due to the design properties of Nix, making them a valuable part of a continuous integration (CI) pipeline.

The `testers.runNixOSTest` function

NixOS VM tests are defined using the `testers.runNixOSTest` function. The pattern for NixOS VM tests looks like this:

```nix
let
  nixpkgs = fetchTarball "https://github.com/NixOS/nixpkgs/tarball/nixos-23.11";
  pkgs = import nixpkgs { config = {}; overlays = []; };
in

pkgs.testers.runNixOSTest {
  name = "test-name";
  nodes = {
    machine1 = { config, pkgs, ... }: {
      # ...
    };
    machine2 = { config, pkgs, ... }: {
      # ...
    };
  };
  testScript = { nodes, ... }: ''
    # ...
  '';
}
```

The function `testers.runNixOSTest` takes a module to specify the test options. Because this module only sets configuration values, one can use the abbreviated module notation.

The following configuration values must be set:

- `name` defines the name of the test.
- `nodes` contains a set of named configurations, because a test script can involve more

Skip to main content

- `testScript` defines the Python test script, either as literal string or as a function that takes a `nodes` attribute. This Python test script can access the virtual machines via the names used for the `nodes`. It has super user rights in the virtual machines. In the Python script each virtual machine is accessible via the `machine` object. NixOS provides various methods to run tests on these configurations.

The test framework automatically starts the virtual machines and runs the Python script.

Minimal example

As a minimal test on the default configuration, we will check if the user `root` and `alice` can run Firefox. We will build the example up from scratch.

1. Use a pinned version of Nixpkgs, and explicitly set configuration options and overlays to avoid them being inadvertently overridden by global configuration:

```
let
  nixpkgs = fetchTarball "https://github.com/NixOS/nixpkgs/tarball/nixos-23.1
  pkgs = import nixpkgs { config = {}; overlays = []; };
in

pkgs.testers.runNixOSTest {
  # ...
}
```

2. Label the test with a descriptive name:

```
name = "minimal-test";
```

3. Because this example only uses one virtual machine, the node we specify is simply called `machine`. This name is arbitrary and can be chosen freely. As configuration you use the relevant parts of the default configuration, that we used in a previous tutorial:

```
nodes.machine = { config, pkgs, ... }: {
  users.users.alice = {
    isNormalUser = true;
    extraGroups = [ "wheel" ];
    packages = with pkgs; [
      firefox
      tree
    ];
  };

```

Skip to main content

```
  system.stateVersion = "23.11";
};
```

4. This is the test script:

```
machine.wait_for_unit("default.target")
machine.succeed("su -- alice -c 'which firefox'")
machine.fail("su -- root -c 'which firefox'")
```

This Python script refers to `machine` which is the name chosen for the virtual machine configuration used in the `nodes` attribute set.

The script waits until systemd reaches `default.target`. It uses the `su` command to switch between users and the `which` command to check if the user has access to `firefox`. It expects that the command `which firefox` to succeed for user `alice` and to fail for `root`.

This script will be the value of the `testScript` attribute.

The complete `minimal-test.nix` file content looks like the following:

```
let
  nixpkgs = fetchTarball "https://github.com/NixOS/nixpkgs/tarball/nixos-23.11"
  pkgs = import nixpkgs { config = {}; overlays = []; };
in

pkgs.testers.runNixOSTest {
  name = "minimal-test";

  nodes.machine = { config, pkgs, ... }: {

    users.users.alice = {
      isNormalUser = true;
      extraGroups = [ "wheel" ];
      packages = with pkgs; [
        firefox
        tree
      ];
    };

    system.stateVersion = "23.11";
  };

  testScript = ''
    machine.wait_for_unit("default.target")
    machine.succeed("su -- alice -c 'which firefox'")
    machine.fail("su -- root -c 'which firefox'")
  '';
}
```

Skip to main content

Running tests

To set up all machines and run the test script:

```
$ nix-build minimal-test.nix
```

```
...
test script finished in 10.96s
cleaning up
killing machine (pid 10)
(0.00 seconds)
/nix/store/bx7z3imvxxpwkkza10vb23czhw7873w2-vm-test-run-minimal-test
```

Interactive Python shell in the virtual machine

When developing tests or when something breaks, it's useful to interactively tinker with the test or access a terminal for a machine.

To start an interactive Python session with the testing framework:

```
$ $(nix-build -A driverInteractive minimal-test.nix)/bin/nixos-test-driver
```

Here you can run any of the testing operations. Execute the `testScript` attribute from `minimal-test.nix` with the `test_script()` function.

If a virtual machine is not yet started, the test environment takes care of it on the first call of a method on a `machine` object.

But you can also manually trigger the start of the virtual machine with:

```
>>> machine.start()
```

for a specific node,

or

```
>>> start_all()
```

Skip to main content

for all nodes.

You can enter a interactive shell on the virtual machine using:

```
>>> machine.shell_interact()
```

and run shell commands like:

```
uname -a
```

```
Linux server 5.10.37 #1-NixOS SMP Fri May 14 07:50:46 UTC 2021 x86_64 GNU/Linux
```

Re-running successful tests

Tests with multiple virtual machines

Tests can involve multiple virtual machines, for example to test client-server-communication.

The following example setup includes:

- A virtual machine named `server` running nginx with default configuration.
- A virtual machine named `client` that has `curl` available to make an HTTP request.
- A `testScript` orchestrating testing logic between `client` and `server`.

The complete `client-server-test.nix` file content looks like the following:

```
let
  nixpkgs = fetchTarball "https://github.com/NixOS/nixpkgs/tarball/nixos-23.11";
  pkgs = import nixpkgs { config = {}; overlays = []; };
in

pkgs.testers.runNixOSTest {
  name = "client-server-test";

  nodes.server = { pkgs, ... }: {
    networking = {
      firewall = {
        allowedTCPPorts = [ 80 ];
      };
    };
    services.nginx = {
```

Skip to main content

```
  };

  nodes.client = { pkgs, ... }: {
    environment.systemPackages = with pkgs; [
      curl
    ];
  };

  testScript = ''
    server.wait_for_unit("default.target")
    client.wait_for_unit("default.target")
    client.succeed("curl http://server/ | grep -o \"Welcome to nginx!\"")
  '';
}
```

The test script performs the following steps:

1. Start the server and wait for it to be ready.
2. Start the client and wait for it to be ready.
3. Run `curl` on the client and use `grep` to check the expected return string. The test passes or fails based on the return value.

Run the test:

```
$ nix-build client-server-test.nix
```

Additional information regarding NixOS tests

- Running integration tests on CI requires hardware acceleration, which many CIs do not support.

  To run integration tests in GitHub Actions see how to disable hardware acceleration.

- NixOS comes with a large set of tests that can serve as educational examples.

  A good inspiration is Matrix bridging with an IRC.

Next steps

- Module system deep dive
- Building a bootable ISO image

Skip to main content

[1] Support for running NixOS VM tests on macOS is also implemented but currently undocumented.