eBPF with Nix: laptop to testbed

Yifei Sun

Inria, ENS de Lyon, Université Grenoble Alpes

Background

I started a project

- Multicast caching system for networked FS over XDP
- Its running late
- So here I am…

Background

Nix

- Declarative & functional
- Source code -> Derivations -> Closure
- NixOS: operating system as closure

Testbed (Grid'5000 @ SLICES-FR)

- Academic, reservation required
- Ephemeral bare metal machines

Problem

- Environment setup and collboration
  - Headers, compiler, editor…
  - KConfig, QEMU, … (what if multiple machine is needed?)
- Development, deployment and benchmark
  - Cluster boot, data collection, …
- Peer review
  - Ease of result reproduction

What worked for me

NixOS VM tests

- Basically Nix + Python + QEMU
- Multi-machines, different kernels, networking
- Binary cache, and benefits from using Nix

NixOS Compose

- Deployment tool
- Substitute with your own stuff

Userspace tooling

Pull packages from pinned `nixpkgs`

```
devShells.x86_64-linux.default = pkgs.mkShell {
  inputsFrom = [ <locally defined derivations> ];
  packages = with pkgs.llvmPackages; [ clang-unwrapped libllvm ];
};
```

- Compilers
- Libraries
- …

Get a kernel

```
kernel = {
  version = "6.19.0-rc5+multikernel";
  modDirVersion = "6.19.0-rc5";
  stdenv = pkgs.gcc13Stdenv;

  # or ./. or fileset ...
  src = fetchFromGitHub {
    owner = "multikernel";
    repo = "linux";
    rev = "a3b4530cc04fe16ddef6b251baac488df3cae79";
    hash = "sha256-mum7rTLU5xUS2qex7br+EotjPyp0...";
  };

  kernelPatches = [ ... ];

  structuredExtraConfig = {
    MULTIKERNEL = lib.kernel.yes;
  };
};
```

```
boot.kernelPackages = pkgs.linuxPackagesFor (
  pkgs.callPackage (
    { buildLinux, fetchFromGitHub, ... } @ args:
    buildLinux (
      args
      //
      kernel # <--
      //
      (args.argsOverride or { })
    )
  )
  { }
);
```

One machine

```
pkgs.testers.runNixOSTest {
  name = "one-machine-test";

  nodes.machine1 = {
    imports = [ nixosModules.kernel ];
    services.scx.enable = true;
  };

  testScript = ''
    machine1.wait_for_unit("default.target")
    machine1.succeed("")
    machine1.fail("")
  '';
}
```

More machines?

```
pkgs.testers.runNixOSTest {
  name = "lots-of-machine-test";

  nodes.machine1.imports = [ nixosModules.grafana ];
  nodes.machine2.imports = with nixosModules; [
    kernel exporter ebpf benchmark
  ];

  testScript = ''
    start_all()
    machine1...
    machine2...
  '';
}
```

What's in there?

```
nix-repl> test =
pkgs.testers.runNixOSTest { ... }
nix-repl> :p test.
test.config                test.name
test.driver                test.nodes
test.driverInteractive     ...
```

Python test driver (was Perl ~2009)

- Nodes
  - `qemu`
- VLANs
  - `vde_switch`

Node closure:

`<test>.nodes.<name>.system.build.toplevel`

Driver:

`<test>.driver` (run `testScript`)

`<test>.driverInteractive` (Python shell)

Mock syscall

Say we want to troll ourselves:

```
SEC("ksyscall/statx")
int BPF_KSYSCALL(fsd_statx_entry, ... statx(2) args) {
  // generate a map entry to collect start ts
  // check path, if not match return
  // else override with a static statx content
  struct statx stx = { ... };
  bpf_probe_write_user(statxbuf, &stx, sizeof(stx));
  return bpf_override_return(ctx, 0);
}
```

And count how many times we can footgun ourselves

With a counter and a histogram

Declarative userspace program

- Auto-load the the program (feat. `ebpf_exporter`)
- Collect the metrics and plot them (feat. Prometheus & Grafana)
- Local testing (feat. NixOS VM test)
- Deployment (feat. NixOS-Compose)

Local testing

For simplicity

- We will be using a readily available userspace tool
  - Loading the program
  - Read the map and re-expose the content over Prometheus

Complication is fast

- Build once and its immutable
- Push cache to server (or have a CI server build it)
- SBOM

Debugging is easy

- SSH backdoor enable with a knob

Demo

- Build interactive driver closure

```
nom build .#checks.x86_64-linux.default.driverInteractive
```

- Start the driver

```
$ ./result/bin/nixos-test-driver
start vlan
running vlan (pid 3859017; ctl /run/user/1000/vde1.ctl)
SSH backdoor enabled, the machines can be accessed like this:
    collector:  ssh -o User=root vsock/3
    exporter:   ssh -o User=root vsock/4
```

Straight to prod

Bit-perfect reproducibility (*: for some store paths)

Everything is in closure

- Deployment harness is easy to write

Demo

- Build deployment closure (instrumented with NixOS test)

  ```
  nxc build
  ```

- Schedule couple machines and deploy the closure to cluster

Effort

Less than 250 LoC

- Portable modules
- Composable with other services
- Adding new programs to deployment only adds a couple characters

```
services.prometheus.exporters.ebpf =
{
  enable = true;
  names = [ "a" "b" "c" ... ];
};
```