eBPF with Nix: laptop to testbed

Yifei Sun - PhD Student at

Inria, ENS de Lyon, Université Grenoble Alpes

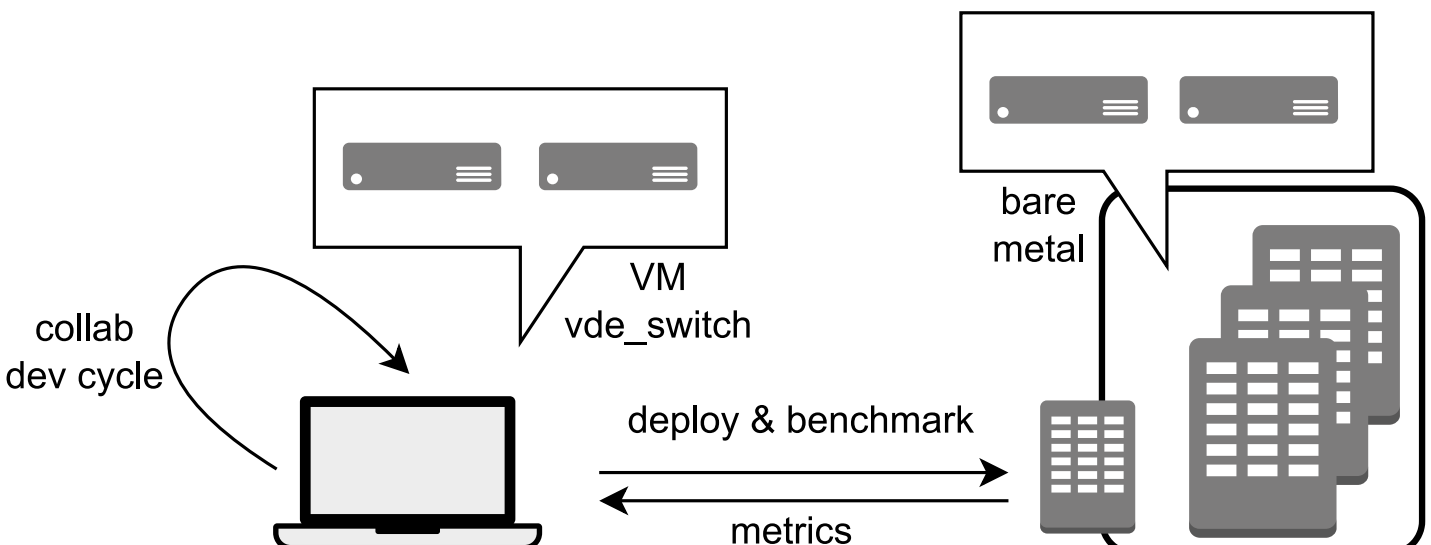
- Website: https://0.0.9.5.f.2.0.6.2.ip6.arpa
- Email: fosdem@0.0.9.5.f.2.0.6.2.ip6.arpa

Goals

- DevOps
-

Goals

- DevOps
- Demonstrate how I use Nix to do BPF related work



Background

I started a project

- Multicast caching system for networked FS over XDP
-
-

Background

I started a project

- Multicast caching system for networked FS over XDP
- Its running late
-

Background

I started a project

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

Background

Nix

- **Declarative** & functional
-
-
-

---

[1] https://zero-to-nix.com/concepts/derivations/
[2] https://zero-to-nix.com/concepts/closures/

Background

Nix

- **Declarative** & functional
- Source code → Derivations[1] → Closure[2]
-
-

[1] https://zero-to-nix.com/concepts/derivations/
[2] https://zero-to-nix.com/concepts/closures/

Background

Nix

- **Declarative** & functional
- Source code → Derivations[1] → Closure[2]
- NixOS: operating system as closure
-

[1] https://zero-to-nix.com/concepts/derivations/
[2] https://zero-to-nix.com/concepts/closures/

Background

Nix

- **Declarative** & functional
- Source code → Derivations[1] → Closure[2]
- NixOS: operating system as closure
- Nix and NixOS devroom @ UA2.118 (Henriot)

---

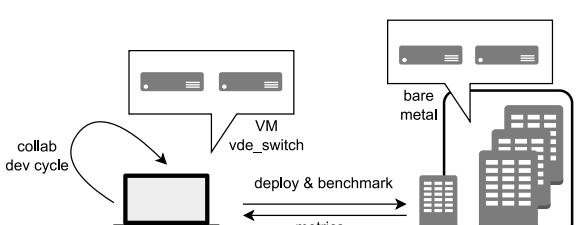
[1] https://zero-to-nix.com/concepts/derivations/
[2] https://zero-to-nix.com/concepts/closures/

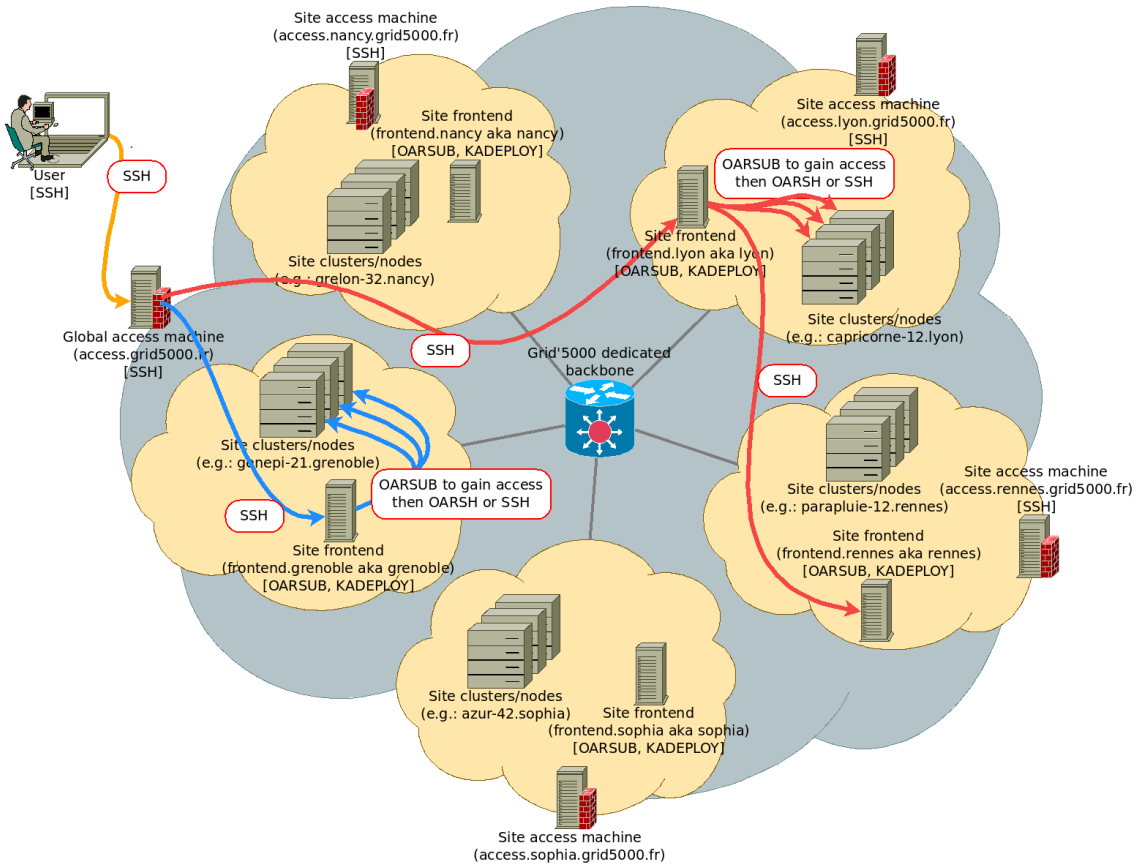
Background

Testbed (Grid'5000 @ SLICES-FR)

- Academic HPC cluster, reservation required
- **Ephemeral** bare metal machines

Background


VM
vde_switch
collab
dev cycle
bare
metal
deploy & benchmark
metrics


Site access machine
(access.nancy.grid5000.fr)
[SSH]
Site frontend
(frontend.nancy aka nancy)
[OARSUB, KADEPLOY]
Site clusters/nodes
(e.g.: grelon-32.nancy)
User
[SSH]
SSH
Global access machine
(access.grid5000.fr)
[SSH]
SSH
Site access machine
(access.lyon.grid5000.fr)
[SSH]
OARSUB to gain access
then OARSH or SSH
Site frontend
(frontend.lyon aka lyon)
[OARSUB, KADEPLOY]
Site clusters/nodes
(e.g.: capricorne-12.lyon)
Grid'5000 dedicated
backbone
SSH
Site clusters/nodes
(e.g.: genepi-21.grenoble)
OARSUB to gain access
then OARSH or SSH
SSH
Site frontend
(frontend.grenoble aka grenoble)
[OARSUB, KADEPLOY]
Site clusters/nodes
(e.g.: parapluie-12.rennes)
Site access machine
(access.rennes.grid5000.fr)
[SSH]
Site frontend
(frontend.rennes aka rennes)
[OARSUB, KADEPLOY]
Site clusters/nodes
(e.g.: azur-42.sophia)
Site frontend
(frontend.sophia aka sophia)
[OARSUB, KADEPLOY]
Site access machine
(access.sophia.grid5000.fr)
[SSH]

Problem

- Environment setup and collaboration
  - Headers, compiler, editor…
  - KConfig, QEMU, … (what if multiple machines are needed?)
-
-

Problem

- Environment setup and collaboration
  - Headers, compiler, editor…
  - KConfig, QEMU, … (what if multiple machines are needed?)
- Development, deployment and benchmark
  - Cluster boot, data collection, …
-

Problem

- Environment setup and collaboration
  - Headers, compiler, editor…
  - KConfig, QEMU, … (what if multiple machines are needed?)
- Development, deployment and benchmark
  - Cluster boot, data collection, …
- Peer review
  - Reproducing benchmark results

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

- Multi-flavor deployment tool for **ephemeral** experiments
  - systemd-nspawn
  - Docker
  - Bare metal
  - …
- Substitute with your own stuff

Userspace tooling

Pull packages from pinned `nixpkgs`

```
devShells.x86_64-linux.default = pkgs.mkShell {
  inputsFrom = [ <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;
```

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...";
  };
```

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 = [ ... ];
```

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;
  };
```

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";
```

- Boilerplate

One machine

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

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

- Boilerplate
- Declarative NixOS closure generation

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("")
  '';
}
```

- Boilerplate
- Declarative NixOS closure generation
- Imperative Python stmts to invoke tests

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  ...
```

What's in there?

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

Node closure:

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

What's in there?

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

Node closure:

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

Driver:

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

`<test>.driverInteractive` (Python repl)

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

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

Node closure:

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

Driver:

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

`<test>.driverInteractive` (Python repl)

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 timestamp
  // 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);
}
```

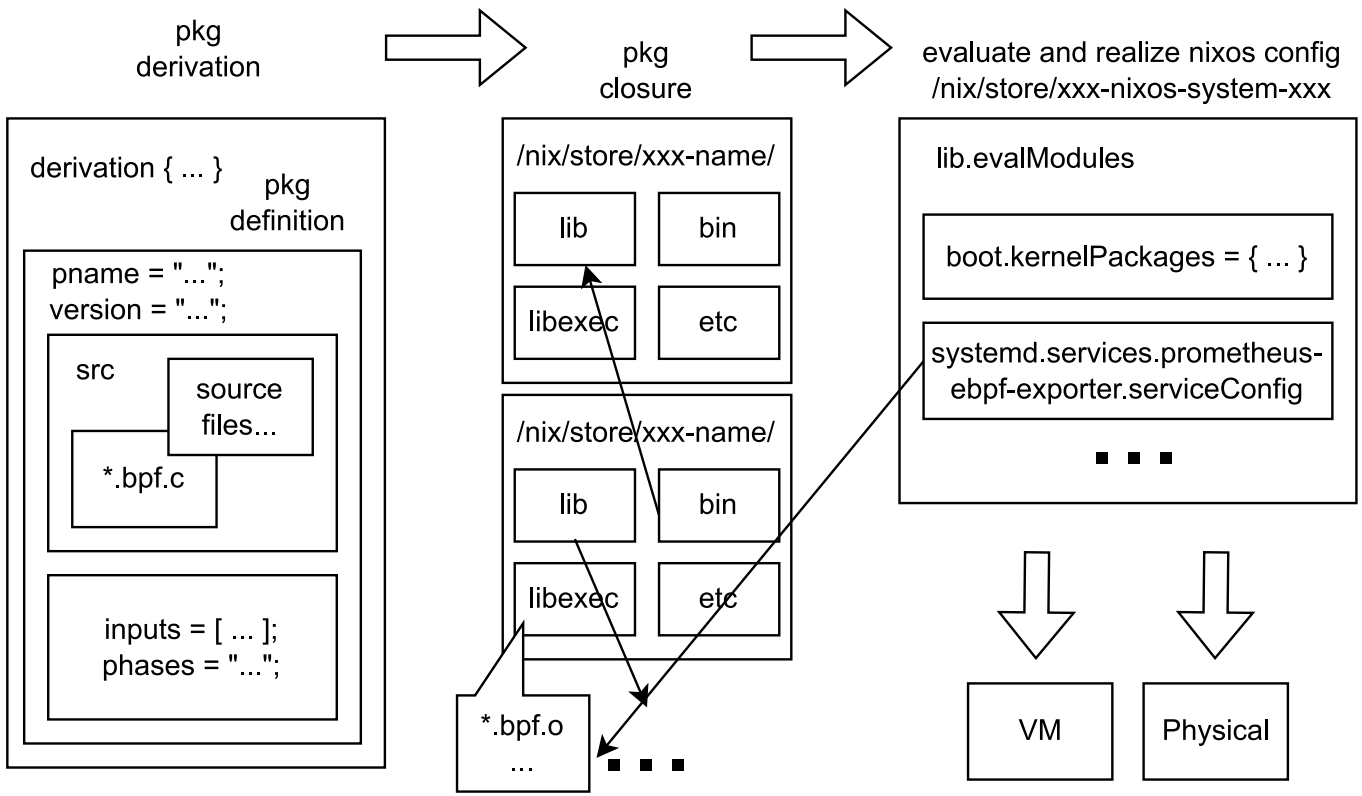
And count how many times we can footgun ourselves

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)

How?


pkg
derivation
pkg
closure
evaluate and realize nixos config
/nix/store/xxx-nixos-system-xxx
derivation { ... }
pkg
definition
pname = "...";
version = "...";
src
source
files...
*.bpf.c
inputs = [...];
phases = "...";
/nix/store/xxx-name/
lib
bin
libexec
etc
/nix/store/xxx-name/
lib
bin
libexec
etc
*.bpf.o
...
lib.evalModules
boot.kernelPackages = { ... }
systemd.services.prometheus-
ebpf-exporter.serviceConfig
VM
Physical

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

Conclusion

Less than 250 LoC (Nix)

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

```
services.prometheus.exporters.ebpf = {
  enable = true;
  # bpf object file names
  names = [
    "oomkill"
    "softirq-latency"
    ...
  ];
};
```

Questions?

- Code: git.sr.ht/~stepbrobd/fosdem
- https://team.inria.fr/datamove
- https://numpex.org
- Website: https://0.0.9.5.f.2.0.6.2.ip6.arpa
- Email: fosdem@0.0.9.5.f.2.0.6.2.ip6.arpa

Our team is hiring!