@fntlnz

Sysdig

eBPF-powered distributed Kubernetes performance analysis

Lorenzo Fontana. Open Source Software Engineer, Sysdig.

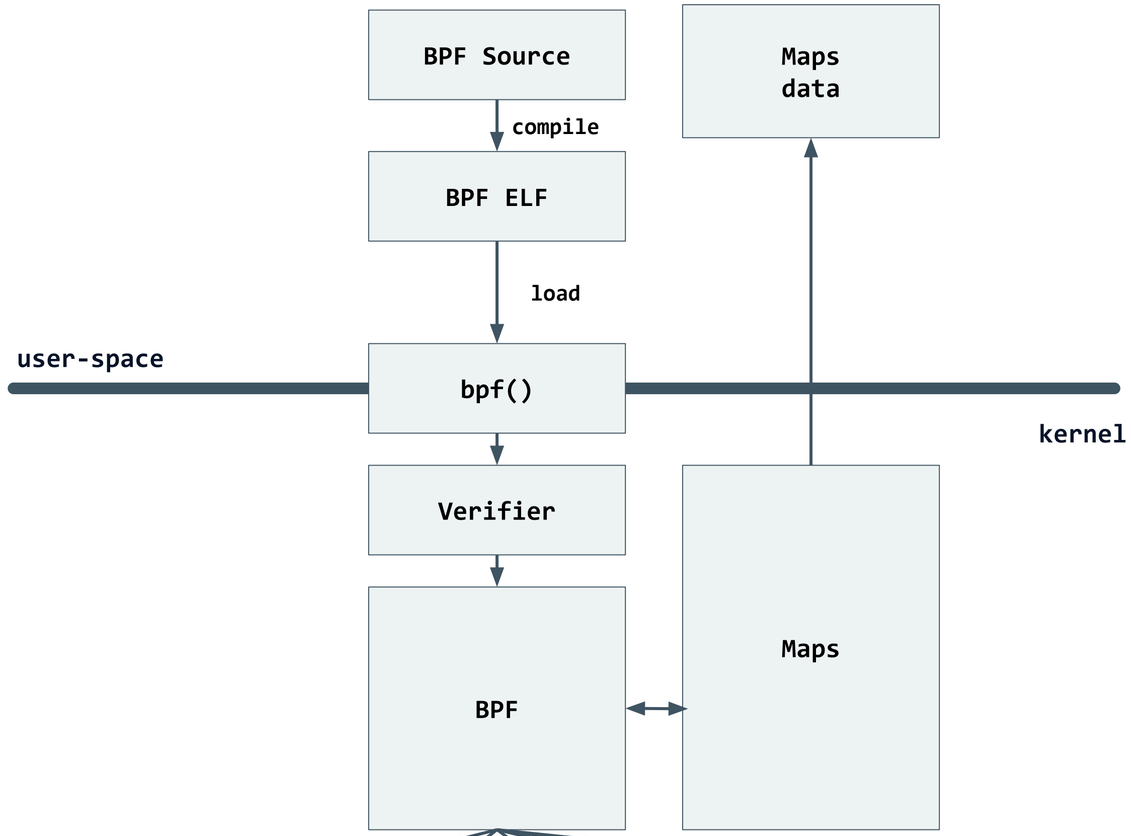
eBPF

Berkley Packet Filter

extended BPF

extended because it’s **not just** packets anymore

How does eBFP work?



BPF_PROG_TYPE_SOCKET_FILTER
BPF_PROG_TYPE_KPROBE
BPF_PROG_TYPE_TRACEPOINT
BPF_PROG_TYPE_RAW_TRACEPOINT
BPF_PROG_TYPE_XDP
BPF_PROG_TYPE_PERF_EVENT
BPF_PROG_TYPE_CGROUP_SKB
BPF_PROG_TYPE_CGROUP_SOCK
BPF_PROG_TYPE_SOCK_OPS
BPF_PROG_TYPE_SK_SKB
BPF_PROG_TYPE_SK_MSG
BPF_PROG_TYPE_SCHED_CLS
BPF_PROG_TYPE_SCHED_ACT

http://bit.ly/bpf_prog_types

BPF_MAP_CREATE
BPF_MAP_LOOKUP_ELEM
BPF_MAP_UPDATE_ELEM
BPF_MAP_DELETE_ELEM
BPF_MAP_GET_NEXT_KEY

http://bit.ly/bpf_map_types

Aggregate events at kernel level

and deal with just a few

instead of thousands of them

What is performance analysis about?

Performance analysis is a **quantitative** and **systematic** approach to **identify** performance issues in a software by doing:

- Measurement of time
- Measurement of space
- Measurement of complexity
- Profiling
- Code Instrumentation

What about Kubernetes ?

Just use a container

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: happy-ebpf
spec:
  shareProcessNamespace: true
  containers:
  - name: execsnoop
    image: calavera/execsnoop # <-- the actual image containing the eBPF program
    securityContext:
    - privileged: true
    volumeMounts:
    - name: sys # mount the debug filesystem
      mountPath: /sys
      readOnly: true
    - name: headers # mount the kernel headers required by bcc
      mountPath: /usr/src
      readOnly: true
    - name: modules # mount the kernel modules required by bcc
      mountPath: /lib/modules
      readOnly: true
  - name: container doing random work
    image: yourcompany/yourapp # <-- your actual application
  ...
```

- 👋 A sidecar container sharing the process namespace
- 👋 You just provide an image with an eBPF loader and program in it
- 👋 Not extremely generic but does the job!
- 👋 A very flexible approach!

```yaml
---
apiVersion: v1
kind: Namespace
metadata:
  name: pkts-ns
---
apiVersion: bpf.sh/v1alpha1
kind: BPF
metadata:
  name: pkts-bpf
  namespace: pkts-ns
spec:
  program:
    valueFrom:
      configMapKeyRef:
        name: pkts-config
        key: pkts.o
---
apiVersion: v1
binaryData:
  pkts.o:
f0VMRgIBAQAAAAAAAAAAAAAEA9wABAAAAAAAAAAAAAAAAAAAAAAAAAAAFADAAAAAAAAAAAAAAEAAAAAAAAEAACgABAL8WAAAAAAAAMAAAABcAAABjCvz/A
AAAALcBAAABAAAAYxr4/wAAAAC/ogAAAAAAAAcCAAD8////GAEAAAAAAAAAAAAAAAAAAIUAAAABAAAAv6MAAAAAAAAAHAwAA+P
///xUABAAAAAAAYQEAAAAAAAAAHAQAAAQAAAGMQAAAAAAAAAvwMAAAAAAAAC/ogAAAAAAAAcCAAD8
////GAEAAAAAAAAAAAAAAAAAALcEAAAAAAAAAhQAAAAIAAAC3AAAAAAAAAAJUAAAAAAAAAAAQAAAAQAAAAEAAAAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAEdQTAD+
////AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAaQAAAAAAAAwCIAAAAAAAAAAAAAAAAAAAAAAAUAAAABAABgAAAAAAAAAAAAAAAAAAAAAAAAAAHQAAABAABwAAA
AAAAAAAAAAAAAAAAAAAAFAAAABAABQAAAAAAAAAAAAAAAAAAAAAAAAAJgAAABAAAwAAAAAAAAAAAAAAAAAAAAAAAAAAOAAAAAAAAAABAAAAABAAAAJgAAAAAAA
AAAQAAAAQAAAAFBAIDAC50ZXh0AG1hcHMvcGFja2V0cwBjb3VudG1hcABfdmVyc2lvbgBzb2NrZXRfcHJvZwAucmVsc29ja2V0L3Byb2cALmxsdm1
fYWRkcnNpZwBfbGljZW5zZQAuc3RydGFiAC5zeW10YWIATEJCMF8yAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAWQAAAAMAAAAAAAAAAAAAAAAAAAAAAAAAAAA3AIAAAAAAAABwAAAAAAAAAAAAAAAAAAAAAAAAQAAAAAAAAAAAAAAA
AAAAAEAAAABAAAABgAAAAAAAAAAAAAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAA2AAAAAQAAAAYAAAAAAAAAAA
AAAAAAAABAAAAAAAAAAMgAAAAAAAAAAAAAAAAAAAAAAIAAAAAAAAAAAAAAAAAAAAAAMgAAAAkAAAAAAAAAAAAAAAAAAAAAAAAAAAAuAIAAAAAAAAAgAAAAAAA
AAAkAAAADAAAACAAAAAAAAAAAQAAAAAAAAAAAcAAAABAAAAAwAAAAAAAAAAAAAAAAAAAAAAgBAAAAAAAAAGAEAAAAAAAAAAAAAAAAAAAAAQAAAAAAAAAAAAAA
AAAAAABRAAAAAQAAAAMAAAAAAAAAAAAAAAAAAAAAAAgAgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAHgAAAAEAAAADAAAAAAAAAA
AAAAAAAAAAAJAIAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAEIAAAADTP9vAAAAgAAAAAAAAAAAAAAAAANgCAAAAAAAAABAAAAA
AAAAAJAAAAAAAAAAAEAAAAAAAAAAAAAAAAAAAABhAAAAAgAAAAAAAAAAAAAAAAAAAAAAAAAAoAgAAAAAAAJAAAAAAAAAAAAQAAAAIAAAAIAAAAAAAAAABg
AAAAAAAAA
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: pkts-config
  namespace: pkts-ns
```


@fntlnz

Want something more generic?

- 👏 Here's an experiment I've been working with @leodido
- 👏 It loads eBPF ELF objects using a CRD
- 👏 Same as the container example but you don't have to write the loader
- 👏 Exposes a Prometheus endpoint

It's called kube-bpf
https://github.com/bpftools/kube-bpf

https://github.com/bpftools/kube-bpf/blob/master/examples/pkts.c

```c
struct bpf_map_def SEC("maps/packets") countmap = {
    .type = BPF_MAP_TYPE_HASH,
    .key_size = sizeof(int),
    .value_size = sizeof(int),
    .max_entries = 256,
};

SEC("socket/prog")
int socket_prog(struct __sk_buff *skb) {
  int proto = load_byte(skb, ETH_HLEN + offsetof(struct iphdr, protocol));
  int one = 1;
  int *el = bpf_map_lookup_elem(&countmap, &proto);
  if (el) {
    (*el)++;
  } else {
    el = &one;
  }
  bpf_map_update_elem(&countmap, &proto, el, BPF_ANY);
  return 0;
}

char _license[] SEC("license") = "GPL";

unsigned int _version SEC("version") = 0xFFFFFFFE; // this tells to the ELF loader to set the current running
kernel version
```

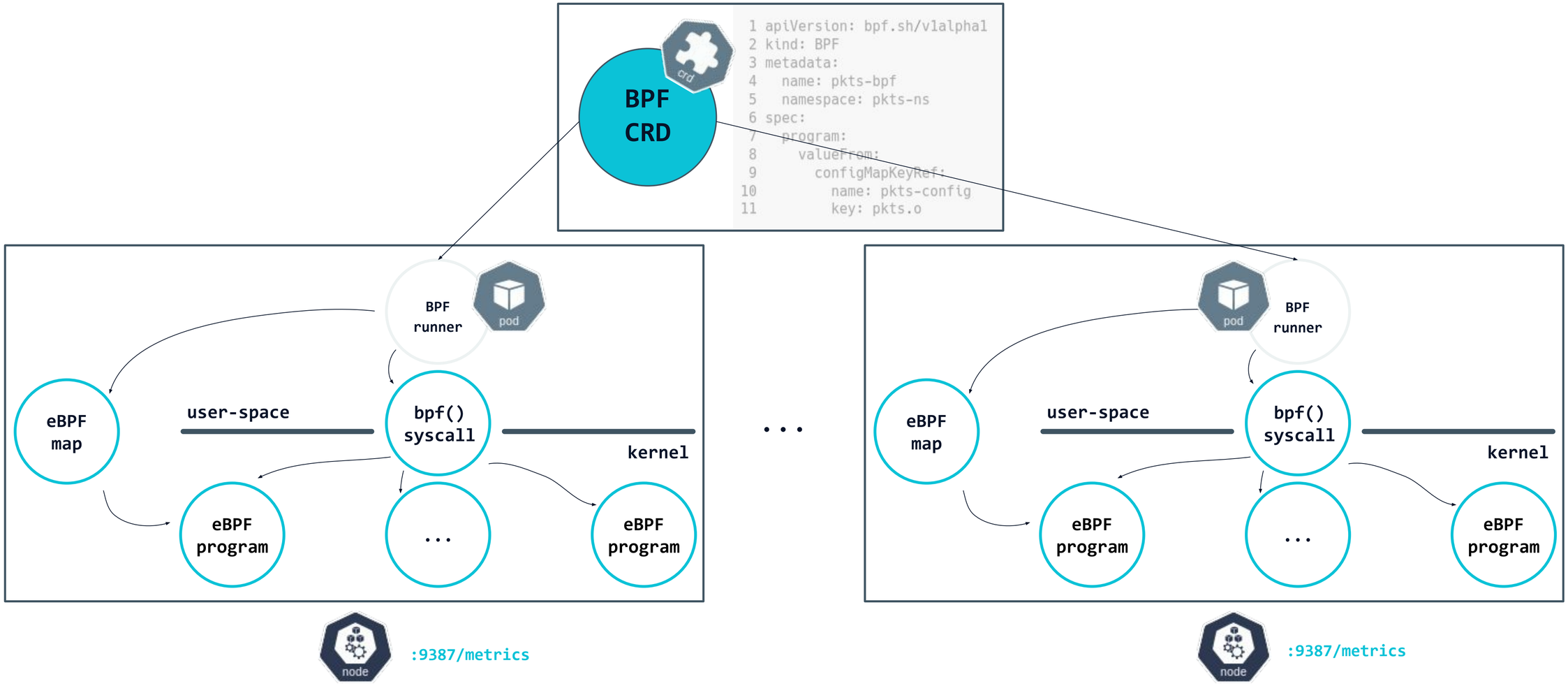
pkts.c

- 👋 Counts all the packets
- 👋 Uses a map to keep a counter
- 👋 It’s an HASH map so that it can assign the counter to a packet type

ip-10-12-0-136.ec2.internal:9387/metrics

```
# HELP test_packets No. of packets per protocol (key), node
# TYPE test_packets counter
test_packets{key="00001",node="127.0.0.1"} 8                    # <- ICMP
test_packets{key="00002",node="127.0.0.1"} 1                    # <- IGMP
test_packets{key="00006",node="127.0.0.1"} 551                  # <- TCP
test_packets{key="00008",node="127.0.0.1"} 1                    # <- EGP
test_packets{key="00017",node="127.0.0.1"} 15930                # <- UDP
test_packets{key="00089",node="127.0.0.1"} 9                    # <- OSPF
test_packets{key="00233",node="127.0.0.1"} 1                    # <- ?
# EOF
```

Here's the evil plan



Get the code!

github.com/bpftools/kube-bpf

demo

https://github.com/fntlnz/oscon-19-demo/tree/master/kubectl-trace

eBPF tracing in the kubectl!

https://github.com/iovisor/kubectl-trace

kubectl-trace

Run bpftrace program (from file)

```
kubectl trace run 127.0.0.1 -f read.bt -a
trace 9df7388a-f0b4-11e8-ae05-8c164500a77e created
^C

@start[12509]: 49914871556264
@start[12856]: 49914833559762
@start[12865]: 49914847759523
@start[12866]: 49914848563942
@start[12867]: 49914872764939


@times:
[512, 1K)              85 |@@@@                                                |
[1K, 2K)              767 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[2K, 4K)              700 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[4K, 8K)              920 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[8K, 16K)             751 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[16K, 32K)            393 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[32K, 64K)             90 |@@@@@                                               |
[64K, 128K)            14 |                                                    |
[128K, 256K)            3 |                                                    |
[256K, 512K)            4 |                                                    |
[512K, 1M)              2 |                                                    |
[1M, 2M)                2 |                                                    |
[2M, 4M)                2 |                                                    |
```

Ctrl-C tells the program to plot the results using hist()

Maps

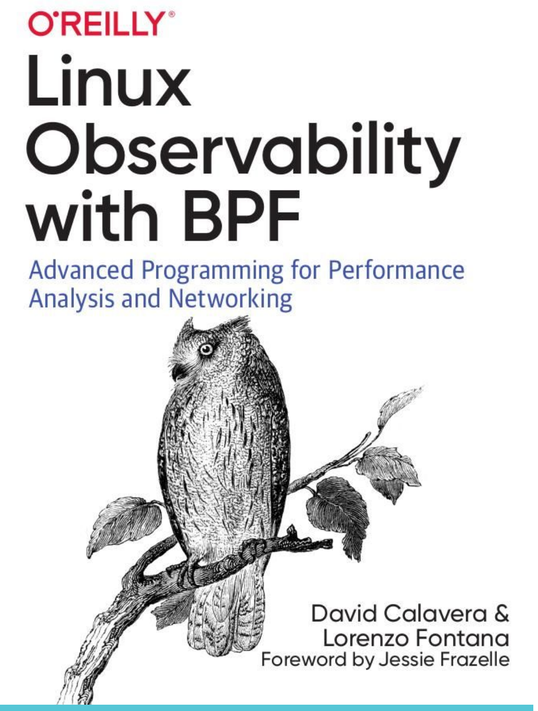
The output histogram

demo

https://github.com/fntlnz/oscon-19-demo/tree/master/kube-bpf

Wait wait wait wait!

There’s a book!



- 👋 From me and **David Calavera**
- 👋 Almost published
- 👋 Preorder on Amazon.com, **DO IT!**
- 👋 Early Release on **O’Reilly Safari**
- 👋 Foreword by **Jessie Frazelle**

All the acronyms

Computer people loves acronyms

BPF: Berkley Packet Filter

eBPF: Extended Berkley Packet Filter

CRD: Custom Resource Definition (Kubernetes)

Thanks. 

Reach me out @fntlnz on twitter & github!