@fntlnz

eBPF powered, distributed Kubernetes performance analysis

Yes, the title is very long...

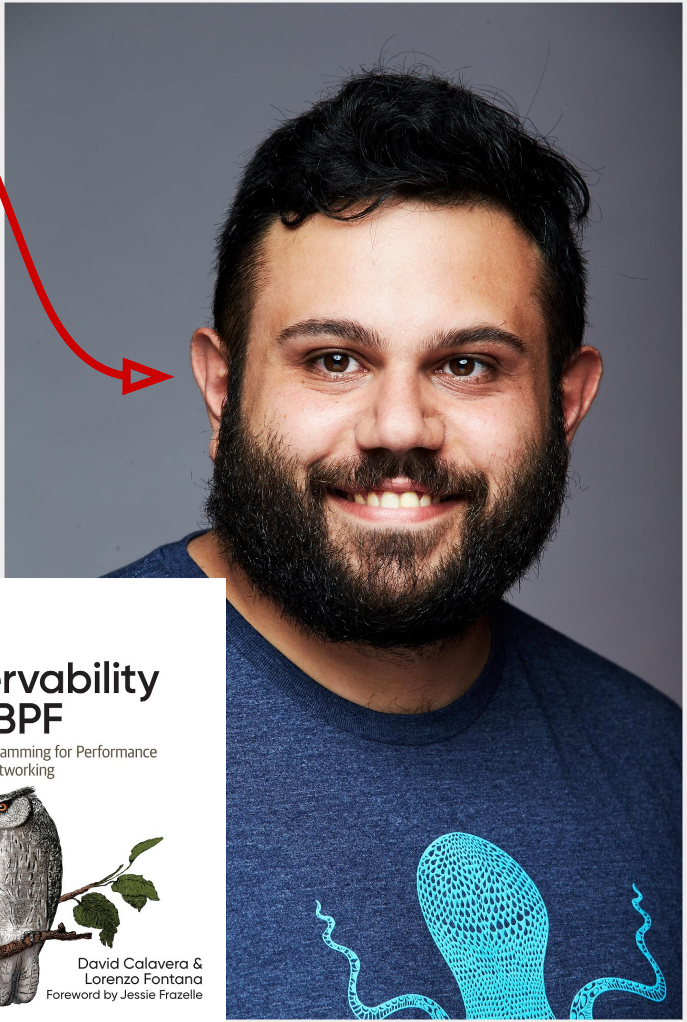
O’Reilly Velocity - Berlin, 2019

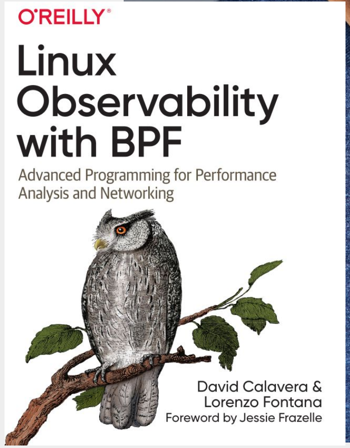
Friendly person

Lorenzo Fontana

Open Source Software Engineer
Sysdig

Tweets at **@fntlnz**

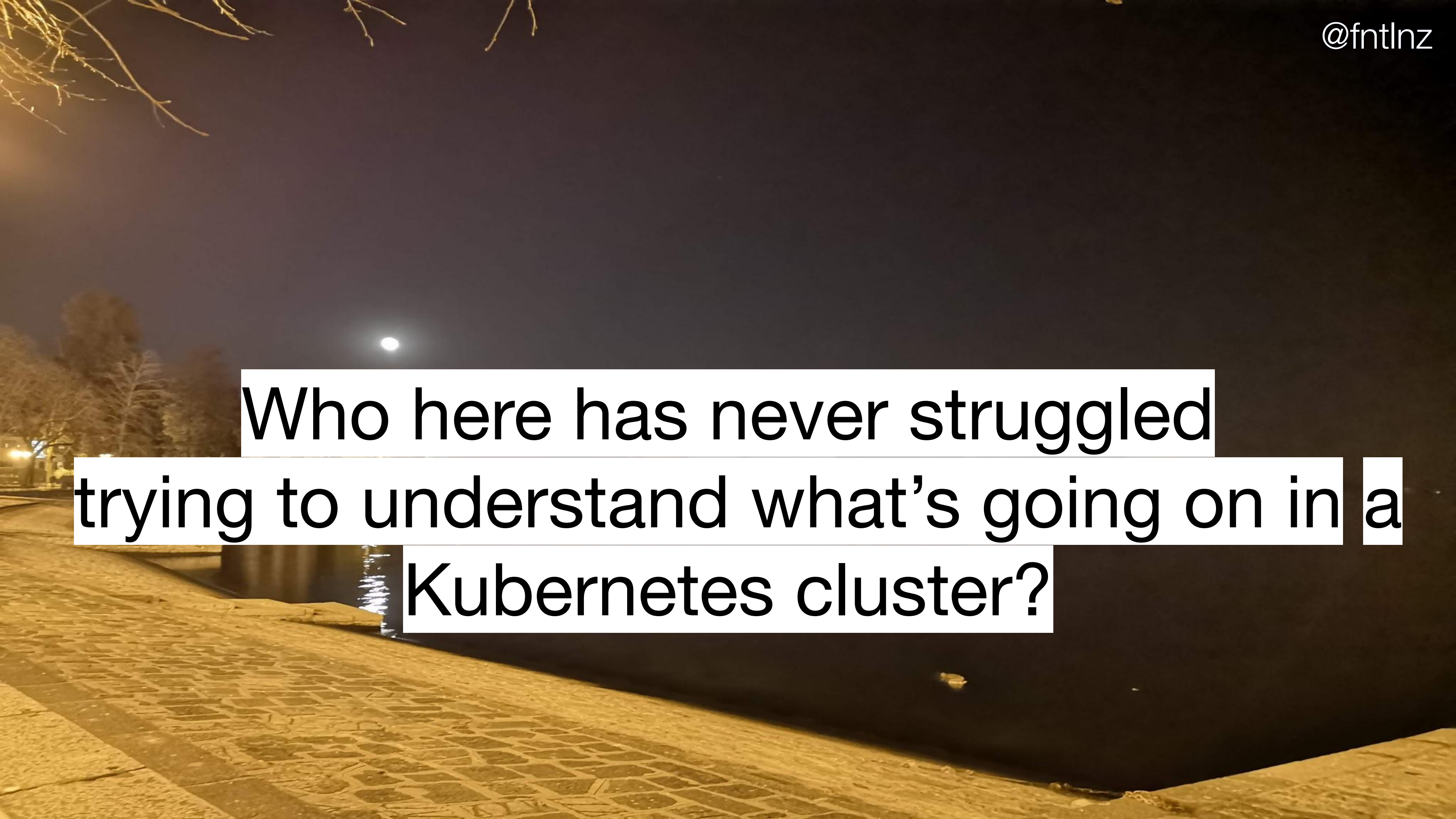



O'REILLY

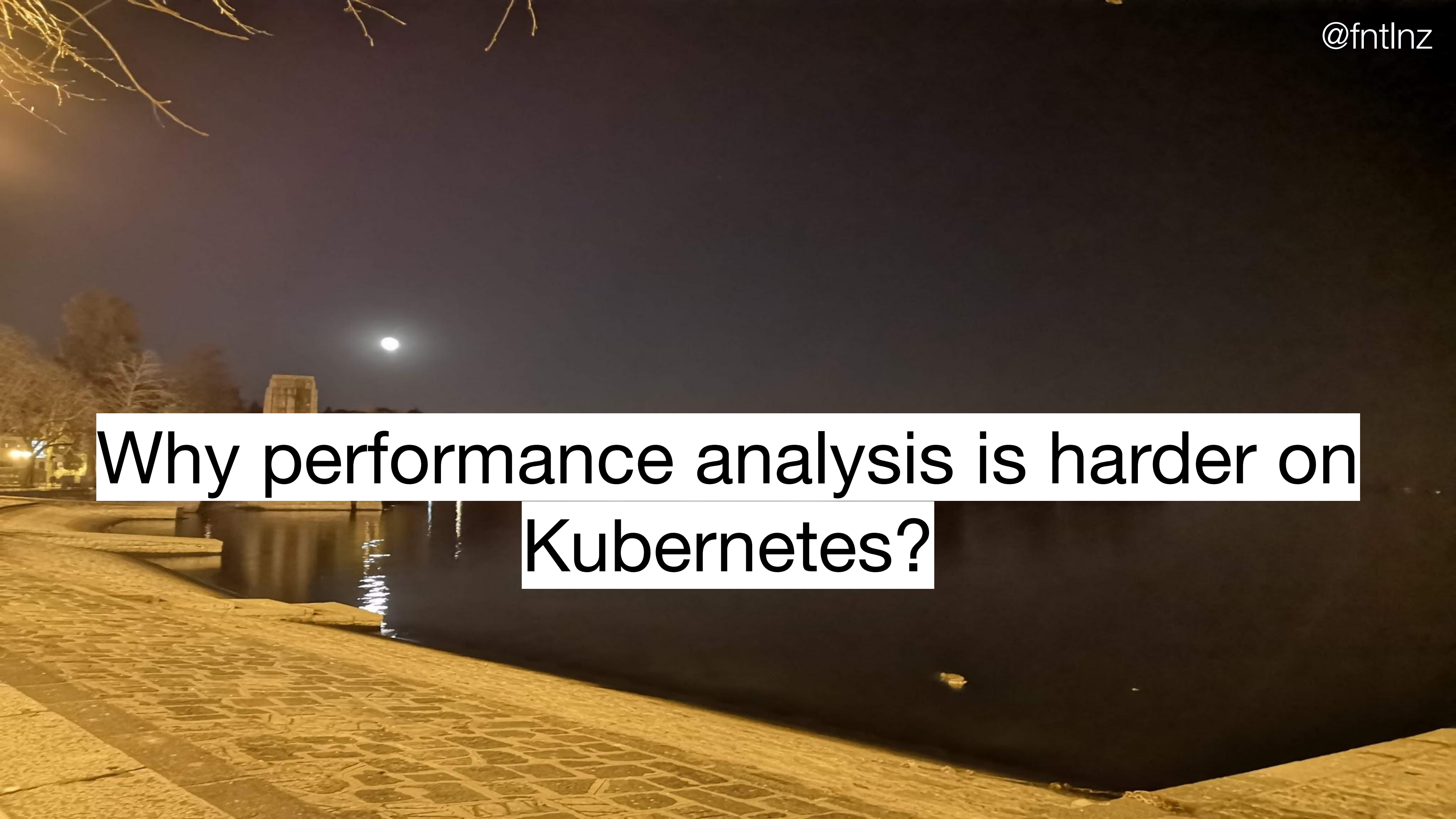
Linux Observability with BPF

Advanced Programming for Performance Analysis and Networking

David Calavera & Lorenzo Fontana
Foreword by Jessie Frazelle

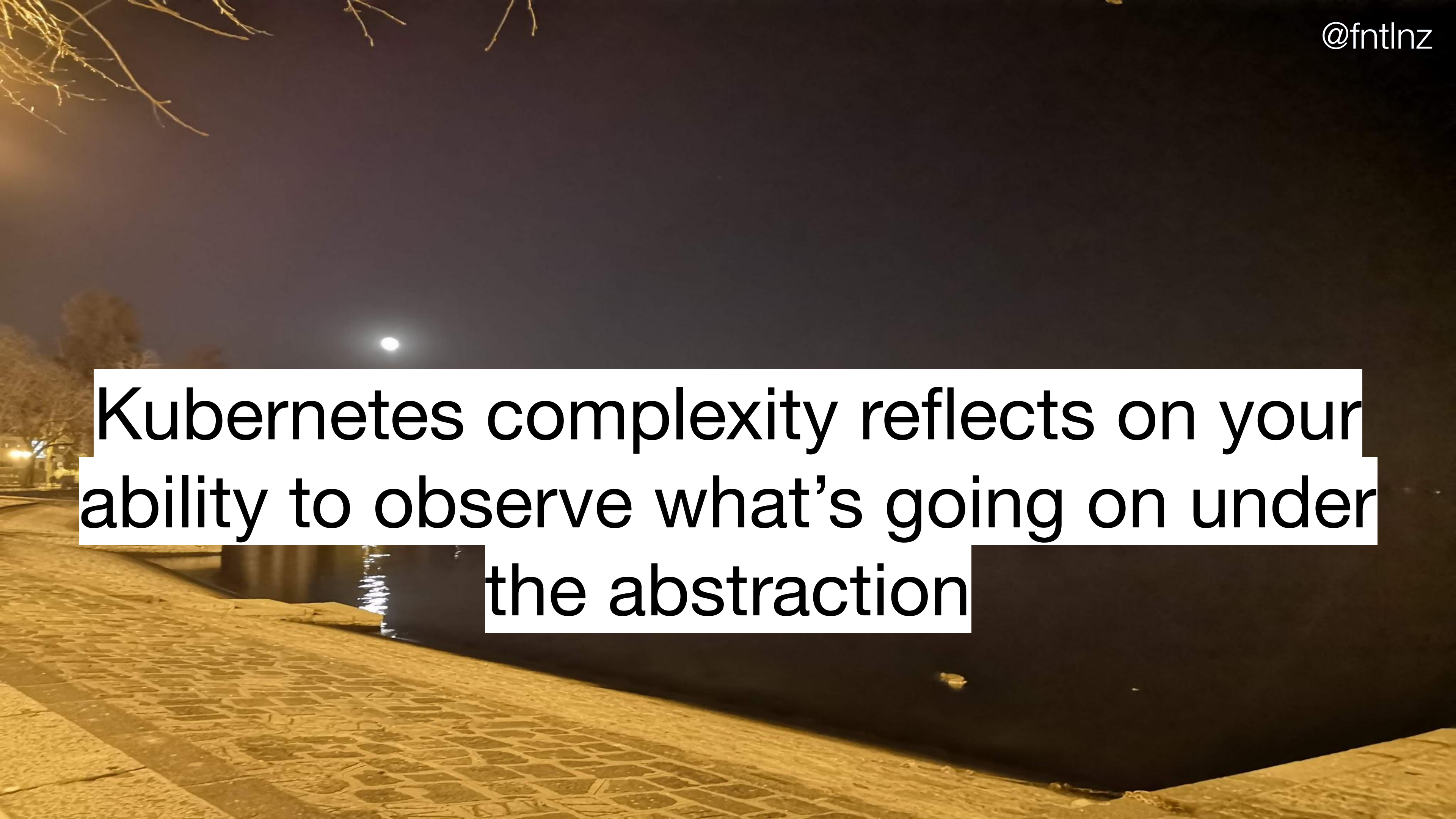


Who here has never struggled trying to understand what’s going on in a Kubernetes cluster?



Why performance analysis is harder on Kubernetes?

Kubernetes is an **abstraction** layer

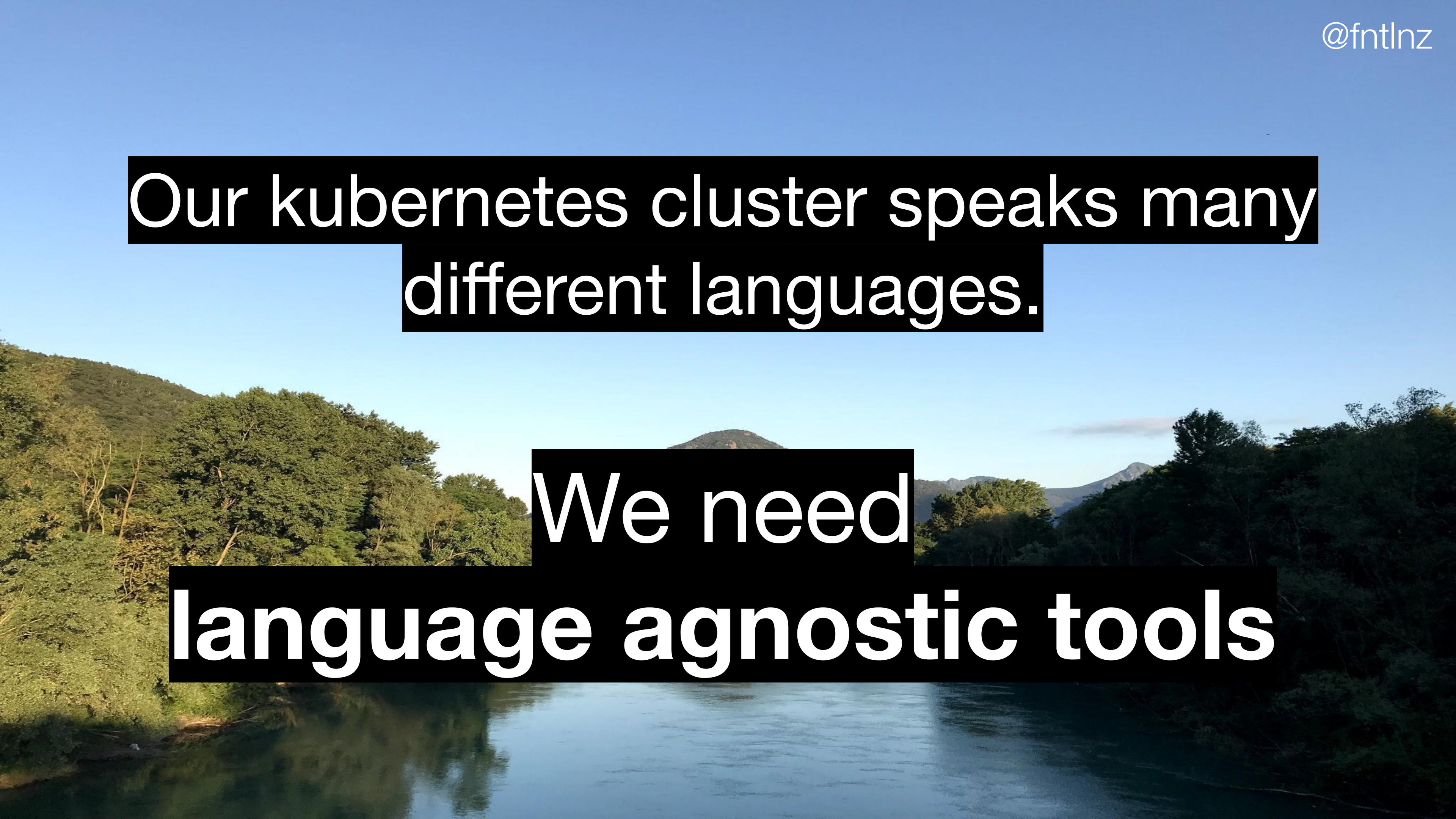


Kubernetes complexity reflects on your ability to observe what’s going on under the abstraction

@fntlnz

Performance analysis on Kubernetes **makes me cry**

Performance analysis tooling
is very tied to the languages



Our kubernetes cluster speaks many different languages.

We need **language agnostic tools**

What are my options then?

strace

Kernel modules

Valgrind

kubernetes unaware

Many options

Top, htop, iotop, etc..

eBPF

In-code

(as having the performance analysis code in the application itself)

Read /proc and /sys filesystems

perf

Slows down applications,
makes them unstable

strace

HARD to write, maintain,
crazy stuff, DEATH

Kernel modules

Slows down applications

Valgrind

kubernetes unaware

Many options

Very limited

Top, htop, iotop,
etc..

Can see everything
Very programmable
Fast
Lots of tools available

eBPF

Good luck with the
performance impact

In-code

(as having the performance
analysis code in the
application itself)

Very limited

Read /proc and
/sys filesystems

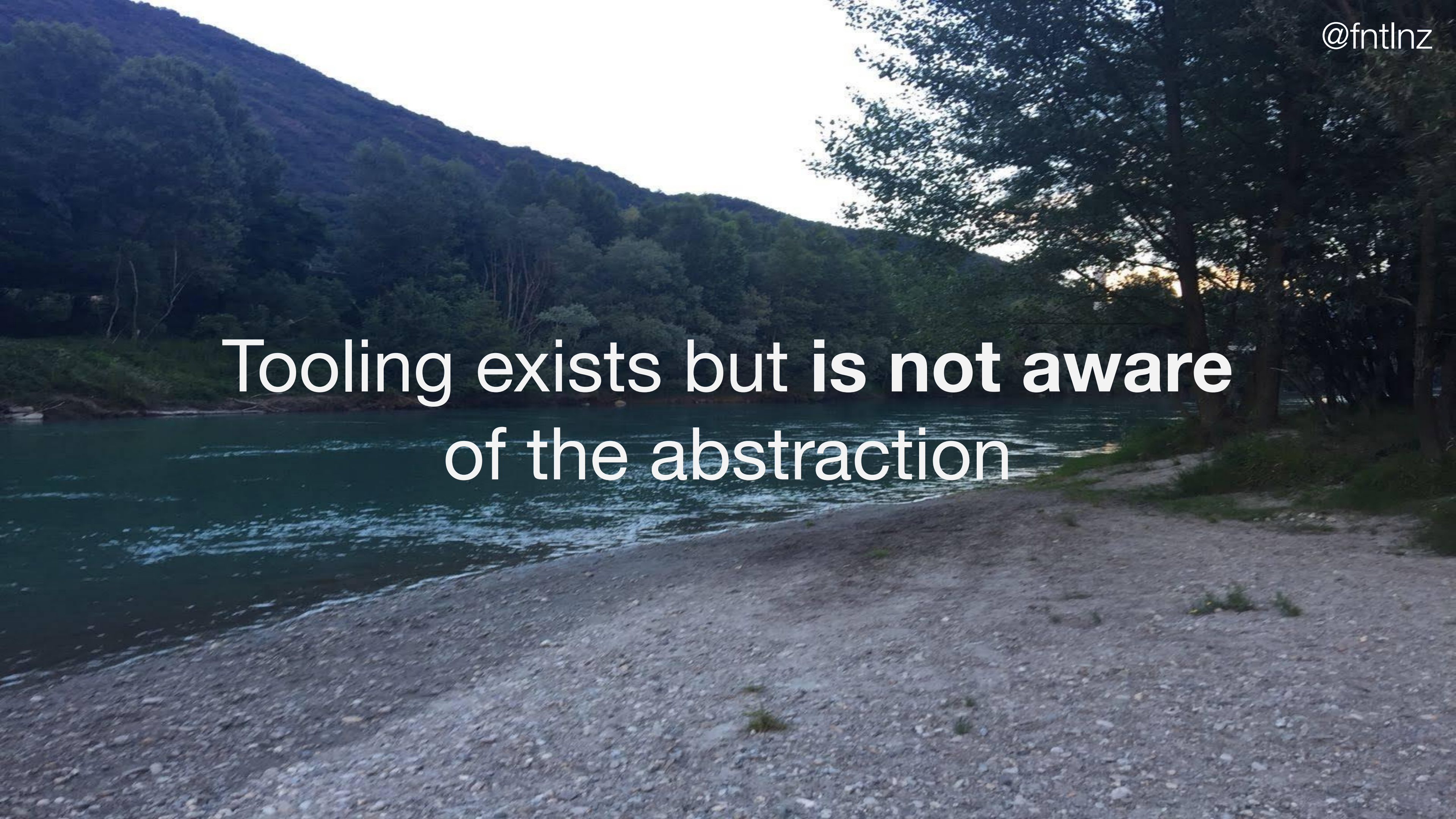
Can see everything
Can also use eBPF
Very limited in integrating
with other tools

perf

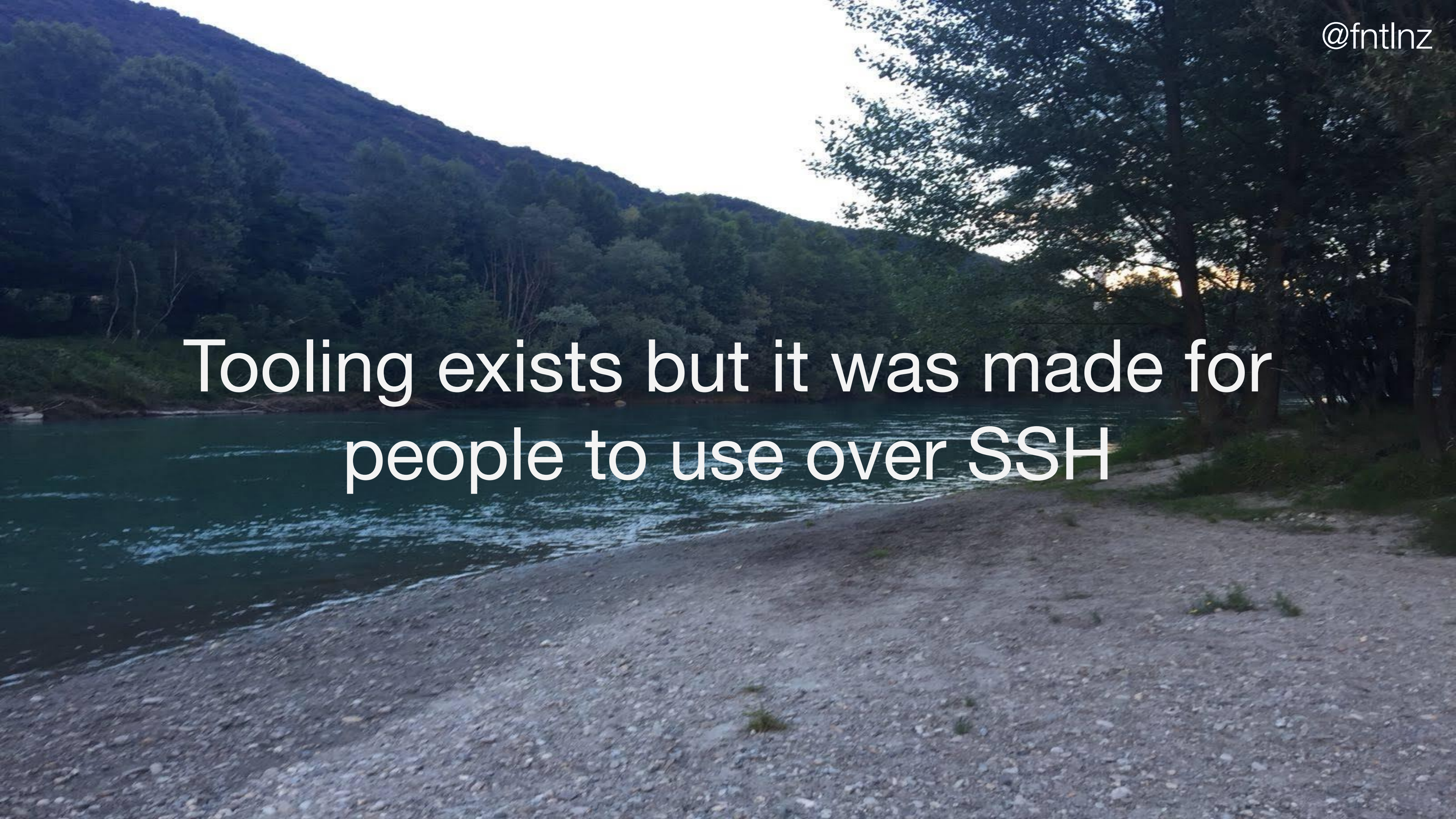
Ok, but.....

@fntlnz

Kubernetes is distributed



Tooling exists but **is not aware** of the abstraction



Tooling exists but it was made for people to use over SSH

Kubernetes SSH is the kubectl

Kubernetes SSH is the ~~kubectl~~ kube-cattle

Abstraction

Application

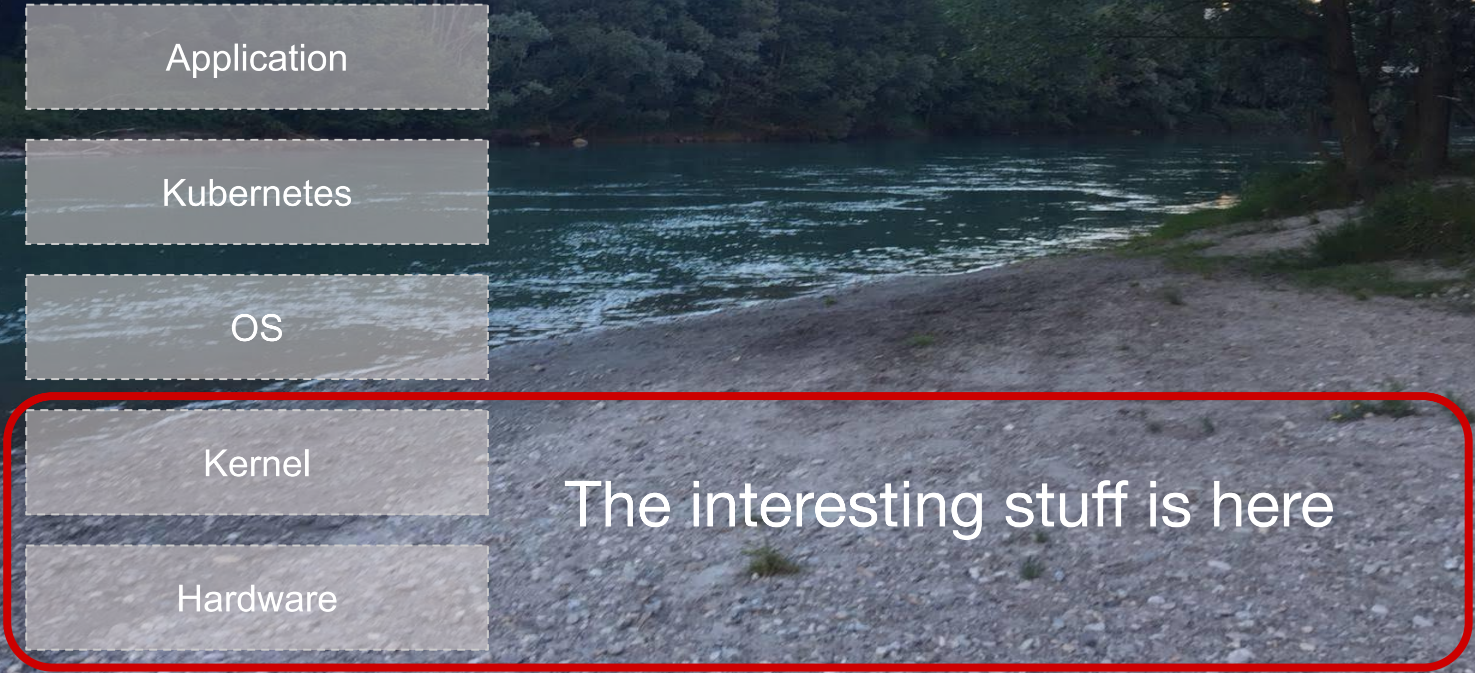
Kubernetes

OS

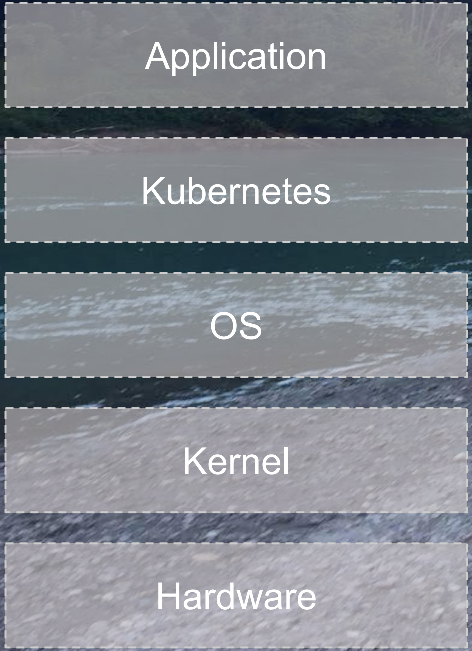
Kernel

Hardware

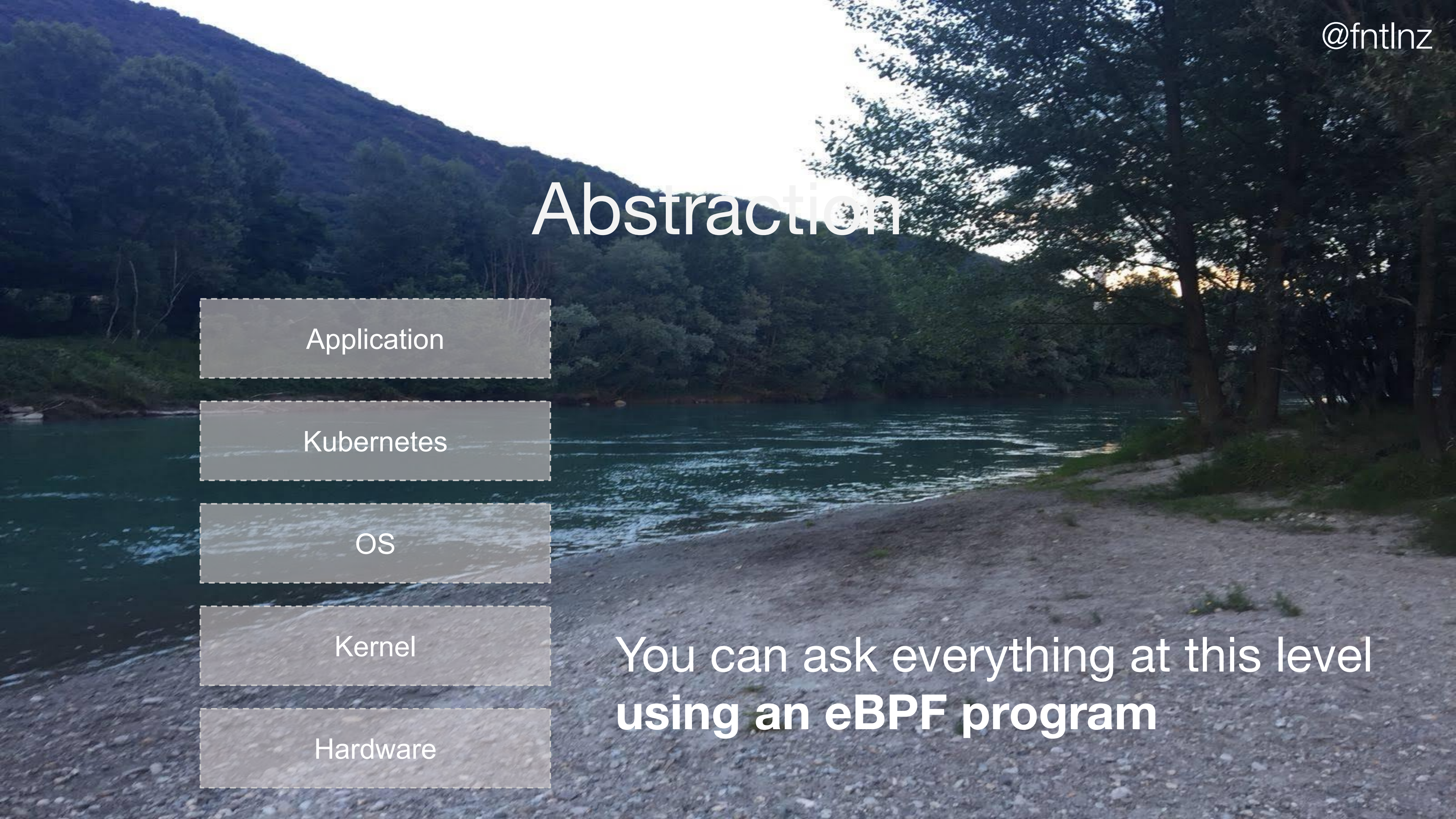
Abstraction



Abstraction



And it knows about the whole thing...



Abstraction

- Application
- Kubernetes
- OS
- Kernel
- Hardware

You can ask everything at this level **using an eBPF program**

How to Kubernetes + eBPF?

They want to be together, we need to help them.

eBPF in a POD

@fntlnz

eBPF using a CRD

eBPF in the kubectl

eBPF in a POD

Easy peasy lemon squeezy

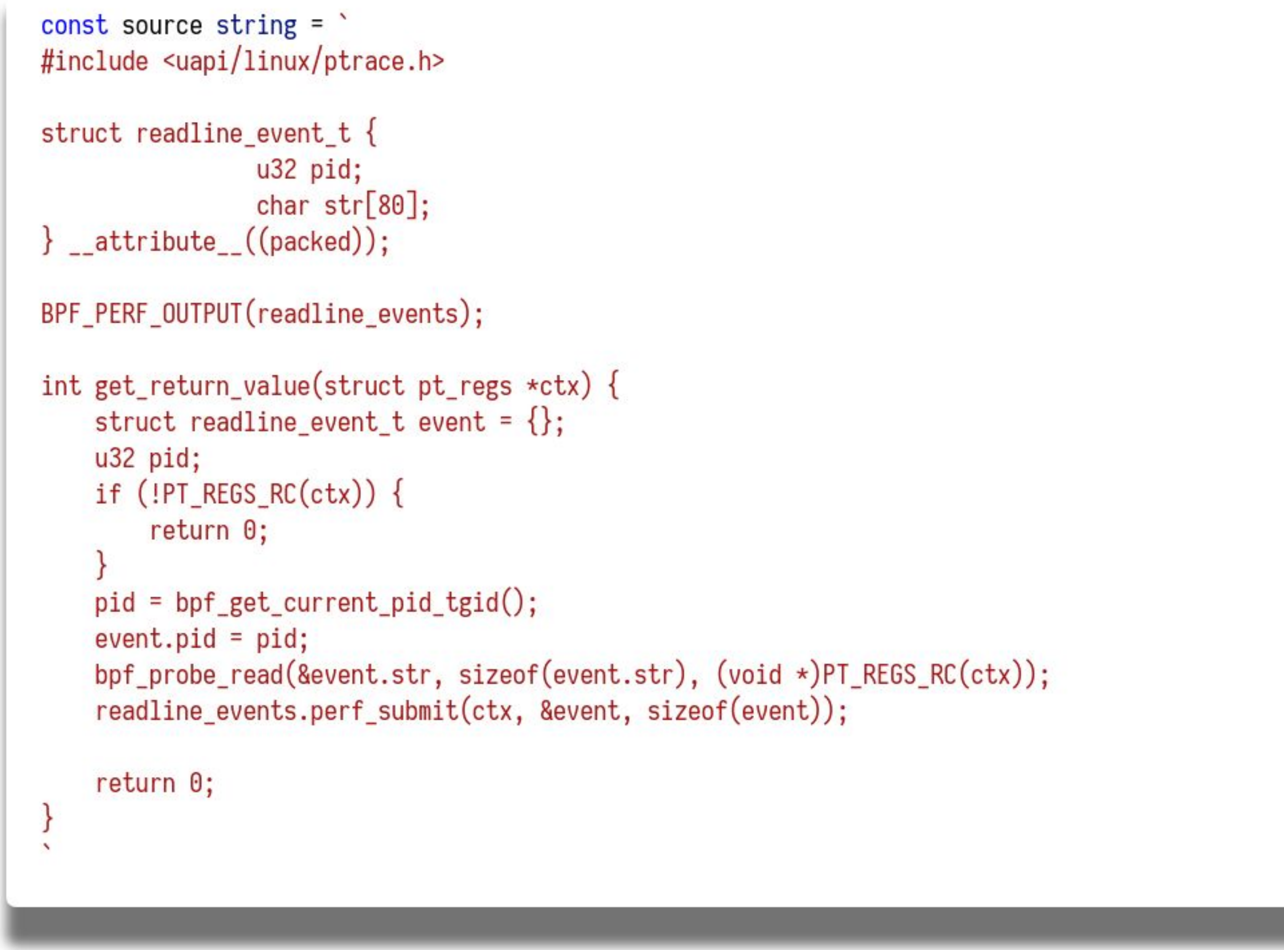
Pros:

- Very customizable
- Easy deployment
- No need to install anything

Cons:

- Need to write boilerplate

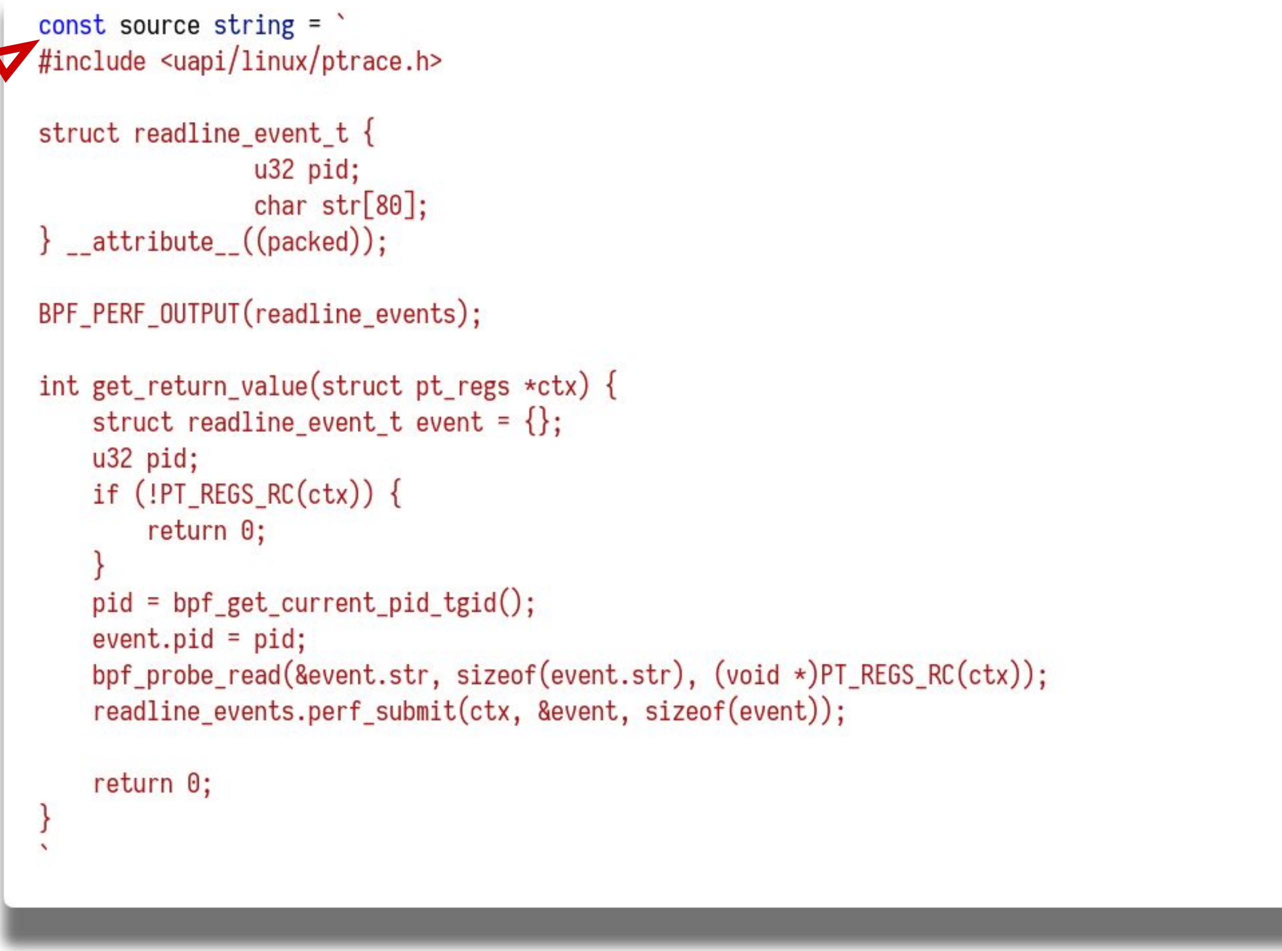
eBPF in a Pod



```
const source string = `
#include <uapi/linux/ptrace.h>

struct readline_event_t {
                u32 pid;
                char str[80];
} __attribute__((packed));

BPF_PERF_OUTPUT(readline_events);

int get_return_value(struct pt_regs *ctx) {
    struct readline_event_t event = {};
    u32 pid;
    if (!PT_REGS_RC(ctx)) {
        return 0;
    }
    pid = bpf_get_current_pid_tgid();
    event.pid = pid;
    bpf_probe_read(&event.str, sizeof(event.str), (void *)PT_REGS_RC(ctx));
    readline_events.perf_submit(ctx, &event, sizeof(event));

    return 0;
}
`
```

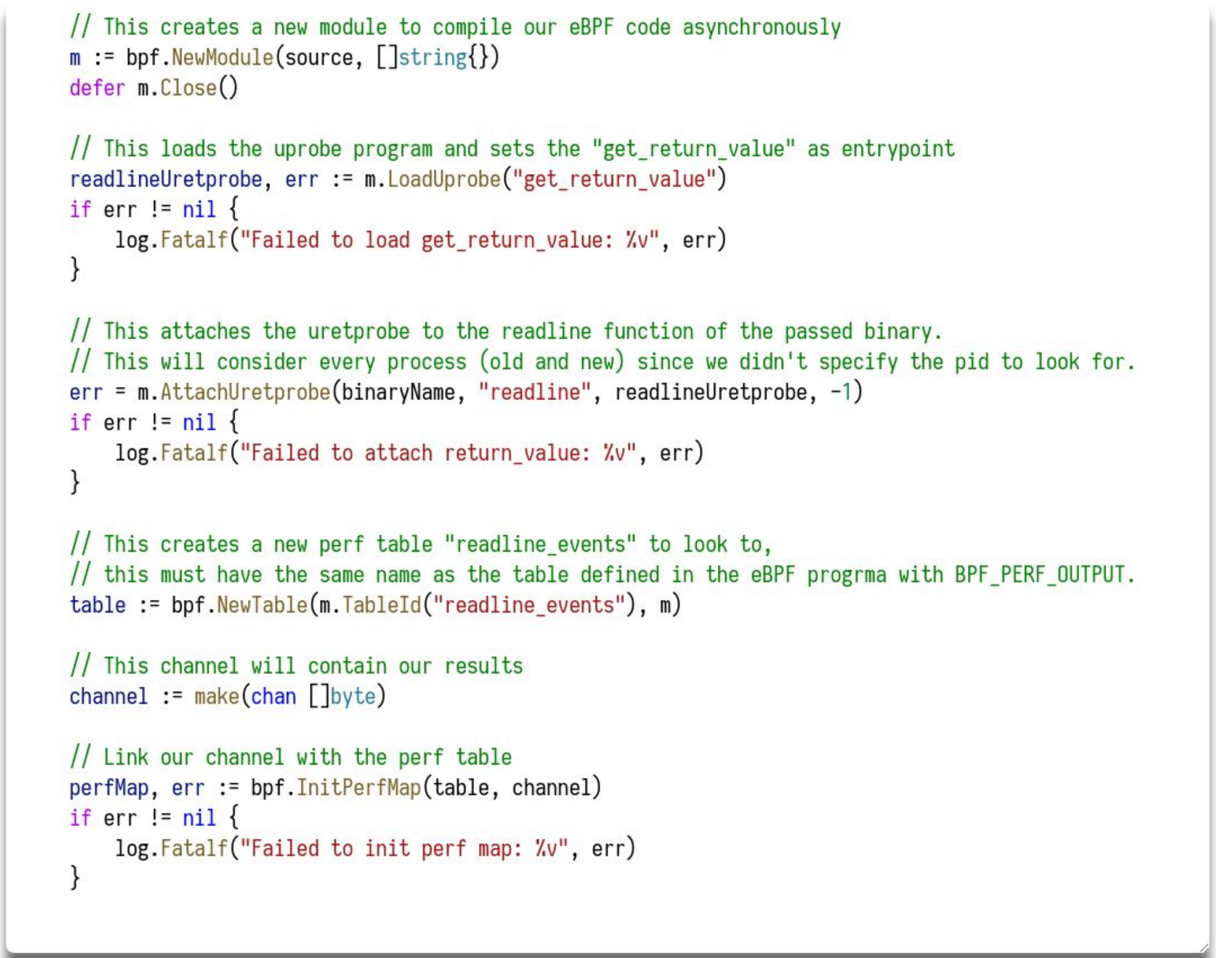
eBPF in a Pod

Yes, this is a Go constant containing C code



```
const source string = `
#include <uapi/linux/ptrace.h>

struct readline_event_t {
                u32 pid;
                char str[80];
} __attribute__((packed));

BPF_PERF_OUTPUT(readline_events);

int get_return_value(struct pt_regs *ctx) {
    struct readline_event_t event = {};
    u32 pid;
    if (!PT_REGS_RC(ctx)) {
        return 0;
    }
    pid = bpf_get_current_pid_tgid();
    event.pid = pid;
    bpf_probe_read(&event.str, sizeof(event.str), (void *)PT_REGS_RC(ctx));
    readline_events.perf_submit(ctx, &event, sizeof(event));

    return 0;
}
`
```

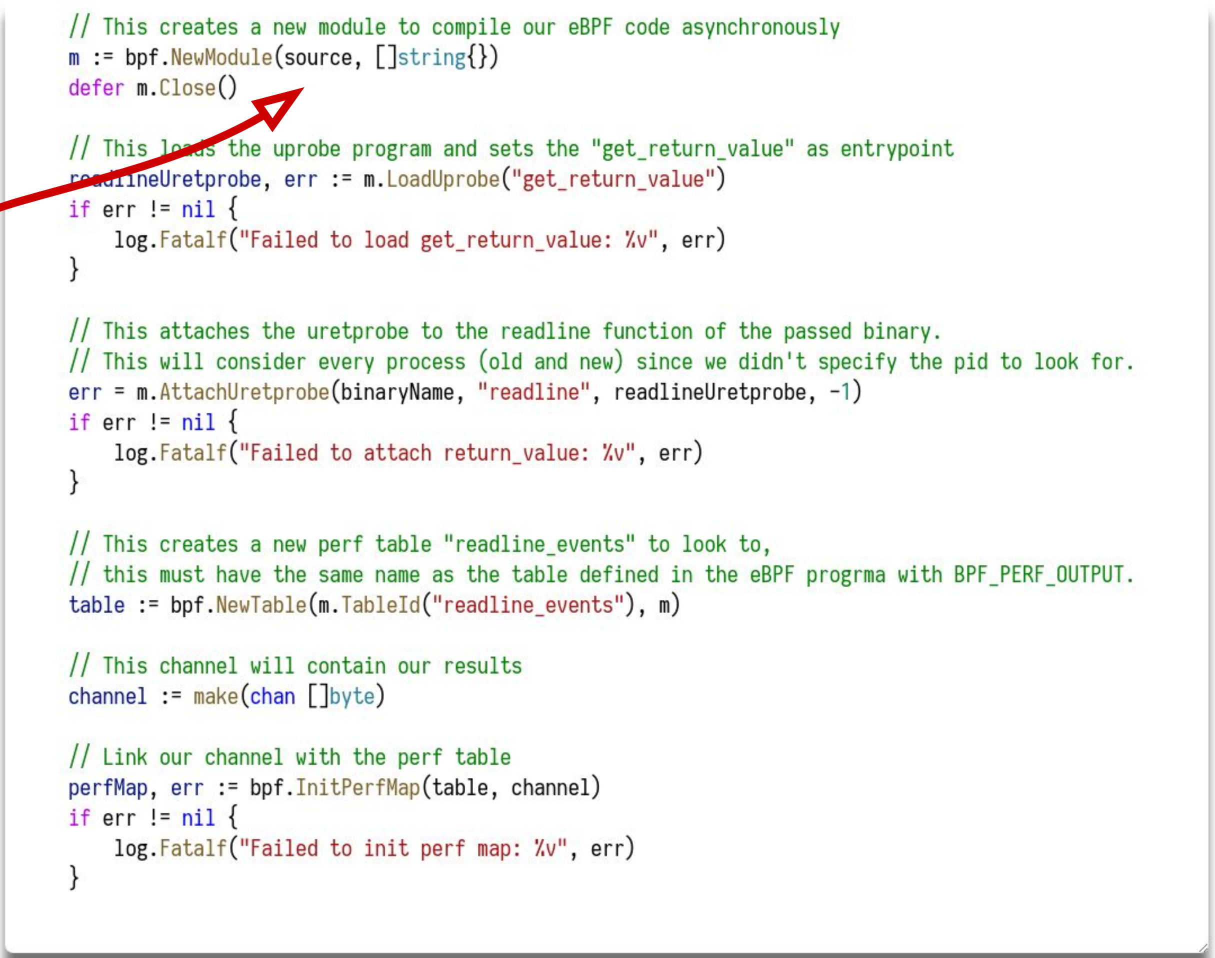
eBPF in a Pod



```go
// This creates a new module to compile our eBPF code asynchronously
m := bpf.NewModule(source, []string{})
defer m.Close()

// This loads the uprobe program and sets the "get_return_value" as entrypoint
readlineUretprobe, err := m.LoadUprobe("get_return_value")
if err != nil {
    log.Fatalf("Failed to load get_return_value: %v", err)
}

// This attaches the uretprobe to the readline function of the passed binary.
// This will consider every process (old and new) since we didn't specify the pid to look for.
err = m.AttachUretprobe(binaryName, "readline", readlineUretprobe, -1)
if err != nil {
    log.Fatalf("Failed to attach return_value: %v", err)
}

// This creates a new perf table "readline_events" to look to,
// this must have the same name as the table defined in the eBPF progrma with BPF_PERF_OUTPUT.
table := bpf.NewTable(m.TableId("readline_events"), m)

// This channel will contain our results
channel := make(chan []byte)

// Link our channel with the perf table
perfMap, err := bpf.InitPerfMap(table, channel)
if err != nil {
    log.Fatalf("Failed to init perf map: %v", err)
}
```

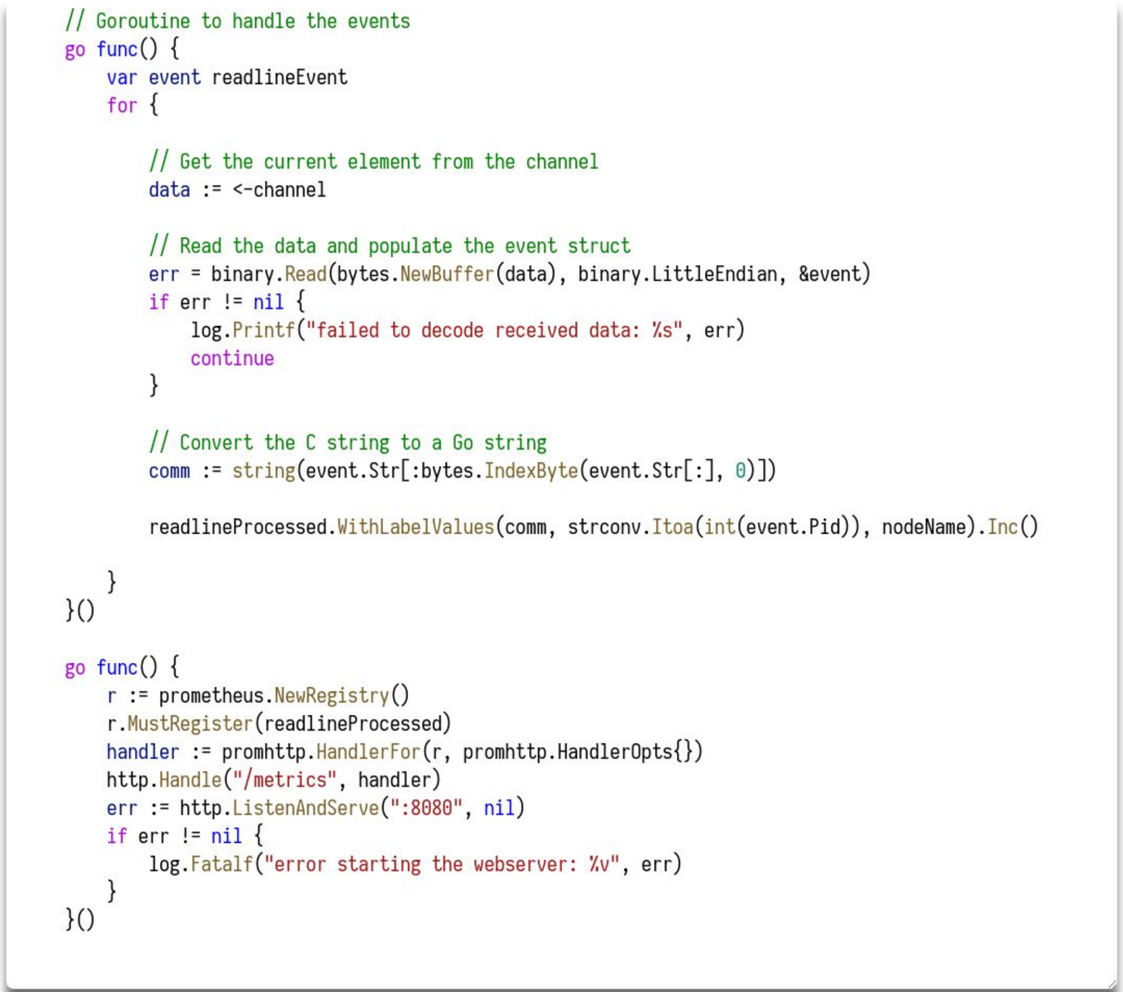
eBPF in a Pod

The C code



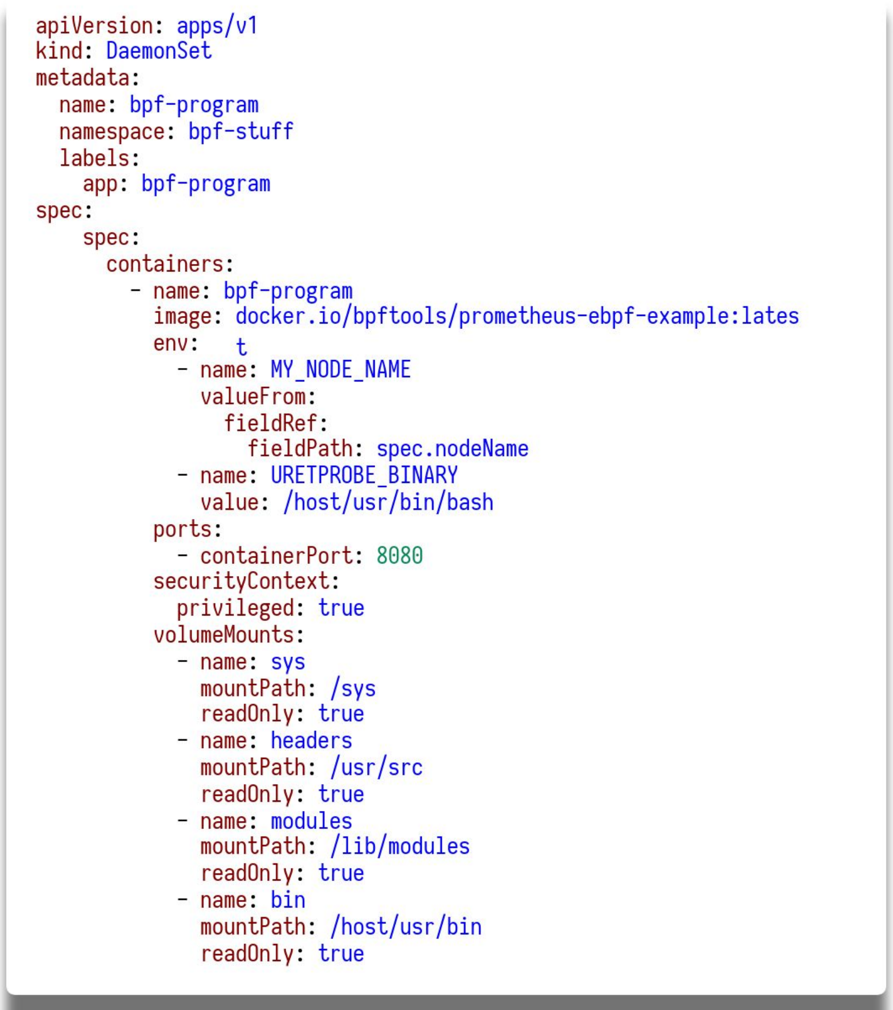
```go
// This creates a new module to compile our eBPF code asynchronously
m := bpf.NewModule(source, []string{})
defer m.Close()

// This loads the uprobe program and sets the "get_return_value" as entrypoint
readlineUretprobe, err := m.LoadUprobe("get_return_value")
if err != nil {
    log.Fatalf("Failed to load get_return_value: %v", err)
}

// This attaches the uretprobe to the readline function of the passed binary.
// This will consider every process (old and new) since we didn't specify the pid to look for.
err = m.AttachUretprobe(binaryName, "readline", readlineUretprobe, -1)
if err != nil {
    log.Fatalf("Failed to attach return_value: %v", err)
}

// This creates a new perf table "readline_events" to look to,
// this must have the same name as the table defined in the eBPF progrma with BPF_PERF_OUTPUT.
table := bpf.NewTable(m.TableId("readline_events"), m)

// This channel will contain our results
channel := make(chan []byte)

// Link our channel with the perf table
perfMap, err := bpf.InitPerfMap(table, channel)
if err != nil {
    log.Fatalf("Failed to init perf map: %v", err)
}
```

eBPF in a Pod



```go
// Goroutine to handle the events
go func() {
    var event readlineEvent
    for {

        // Get the current element from the channel
        data := <-channel

        // Read the data and populate the event struct
        err = binary.Read(bytes.NewBuffer(data), binary.LittleEndian, &event)
        if err != nil {
            log.Printf("failed to decode received data: %s", err)
            continue
        }

        // Convert the C string to a Go string
        comm := string(event.Str[:bytes.IndexByte(event.Str[:], 0)])

        readlineProcessed.WithLabelValues(comm, strconv.Itoa(int(event.Pid)), nodeName).Inc()

    }
}()

go func() {
    r := prometheus.NewRegistry()
    r.MustRegister(readlineProcessed)
    handler := promhttp.HandlerFor(r, promhttp.HandlerOpts{})
    http.Handle("/metrics", handler)
    err := http.ListenAndServe(":8080", nil)
    if err != nil {
        log.Fatalf("error starting the webserver: %v", err)
    }
}()
```

eBPF in a Pod

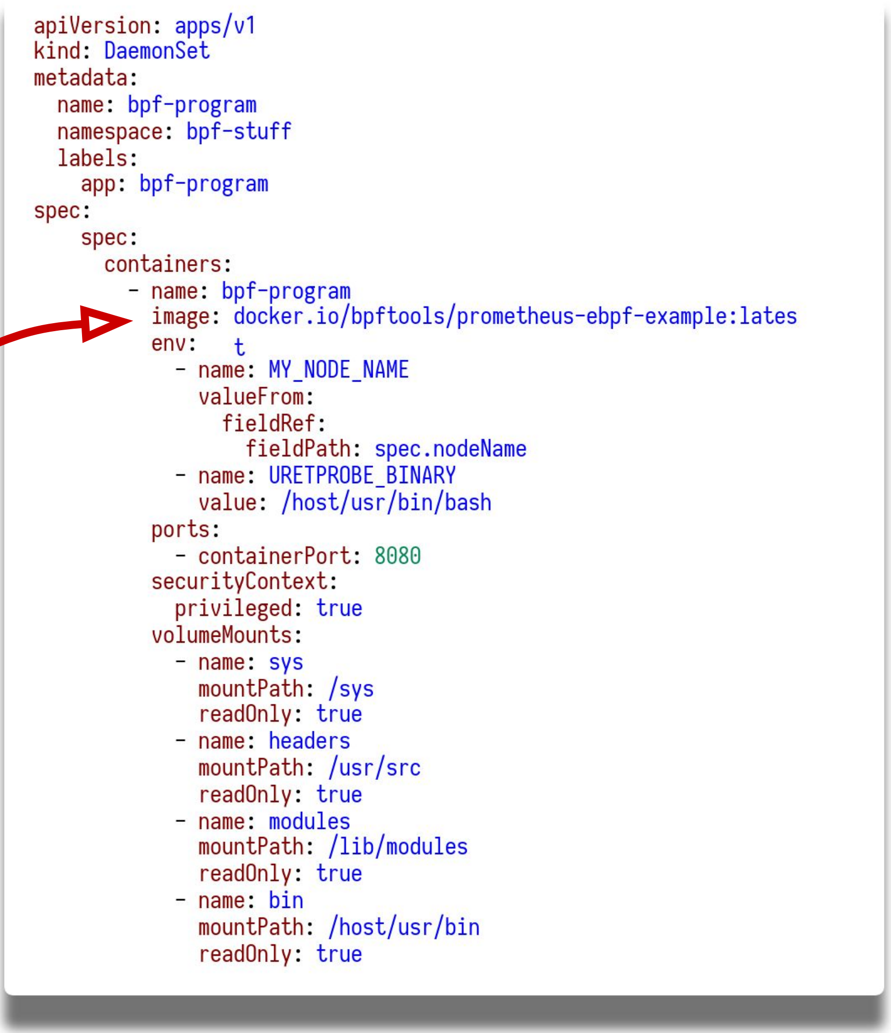


```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: bpf-program
  namespace: bpf-stuff
  labels:
    app: bpf-program
spec:
    spec:
      containers:
        - name: bpf-program
          image: docker.io/bpftools/prometheus-ebpf-example:lates
          env:    t
            - name: MY_NODE_NAME
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            - name: URETPROBE_BINARY
              value: /host/usr/bin/bash
          ports:
            - containerPort: 8080
          securityContext:
            privileged: true
          volumeMounts:
            - name: sys
              mountPath: /sys
              readOnly: true
            - name: headers
              mountPath: /usr/src
              readOnly: true
            - name: modules
              mountPath: /lib/modules
              readOnly: true
            - name: bin
              mountPath: /host/usr/bin
              readOnly: true
```

@fntlnz

eBPF in a Pod

This image uses a compiled version of our BPF loader as entrypoint



```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: bpf-program
  namespace: bpf-stuff
  labels:
    app: bpf-program
spec:
    spec:
      containers:
        - name: bpf-program
          image: docker.io/bpftools/prometheus-ebpf-example:lates
t
          env:
            - name: MY_NODE_NAME
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            - name: URETPROBE_BINARY
              value: /host/usr/bin/bash
          ports:
            - containerPort: 8080
          securityContext:
            privileged: true
          volumeMounts:
            - name: sys
              mountPath: /sys
              readOnly: true
            - name: headers
              mountPath: /usr/src
              readOnly: true
            - name: modules
              mountPath: /lib/modules
              readOnly: true
            - name: bin
              mountPath: /host/usr/bin
              readOnly: true
```

@fntlnz

eBPF in a Pod

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: bpf-program
  namespace: bpf-stuff
  labels:
    app: bpf-program
spec:
    spec:
      containers:
        - name: bpf-program
          image: docker.io/bpftools/prometheus-ebpf-example:lates
          env:   t
            - name: MY_NODE_NAME
              valueFrom:
```

```
kubectl apply -f https://raw.githubusercontent.com/bpftools/prometheus-ebpf-example/master/daemonset.yaml
```

```yaml
              mountPath: /sys
              readOnly: true
            - name: headers
              mountPath: /usr/src
              readOnly: true
            - name: modules
              mountPath: /lib/modules
              readOnly: true
            - name: bin
              mountPath: /host/usr/bin
              readOnly: true
```

@fntlnz

eBPF in a Pod

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: bpf-program
  namespace: bpf-stuff
  labels:
    app: bpf-program
spec:
    spec:
      containers:
        - name: bpf-program
          image: docker.io/bpftools/prometheus-ebpf-example:lates
          env:    t
            - name: MY_NODE_NAME
```

```
kubectl get pods -n bpf-stuff
NAME                 READY   STATUS    RESTARTS   AGE
bpf-program-rtr2x    1/1     Running   0          5h20m
```

```
              mountPath: /sys
              readOnly: true
            - name: headers
              mountPath: /usr/src
              readOnly: true
            - name: modules
              mountPath: /lib/modules
              readOnly: true
            - name: bin
              mountPath: /host/usr/bin
              readOnly: true
```

eBPF in a Pod

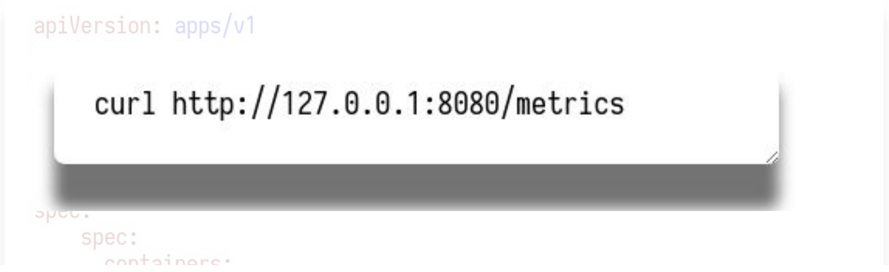
```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: bpf-program
  namespace: bpf-stuff
  labels:
    app: bpf-program
spec:
    spec:
      containers:
        - name: bpf-program
          image: docker.io/bpftools/prometheus-ebpf-example:lates
          env:    t
            - name: MY_NODE_NAME
```

```
kubectl port-forward daemonset/bpf-program -n bpf-stuff 8080:8080
```

```
            - name: sys
              mountPath: /sys
              readOnly: true
            - name: headers
              mountPath: /usr/src
              readOnly: true
            - name: modules
              mountPath: /lib/modules
              readOnly: true
            - name: bin
              mountPath: /host/usr/bin
              readOnly: true
```

@fntlnz

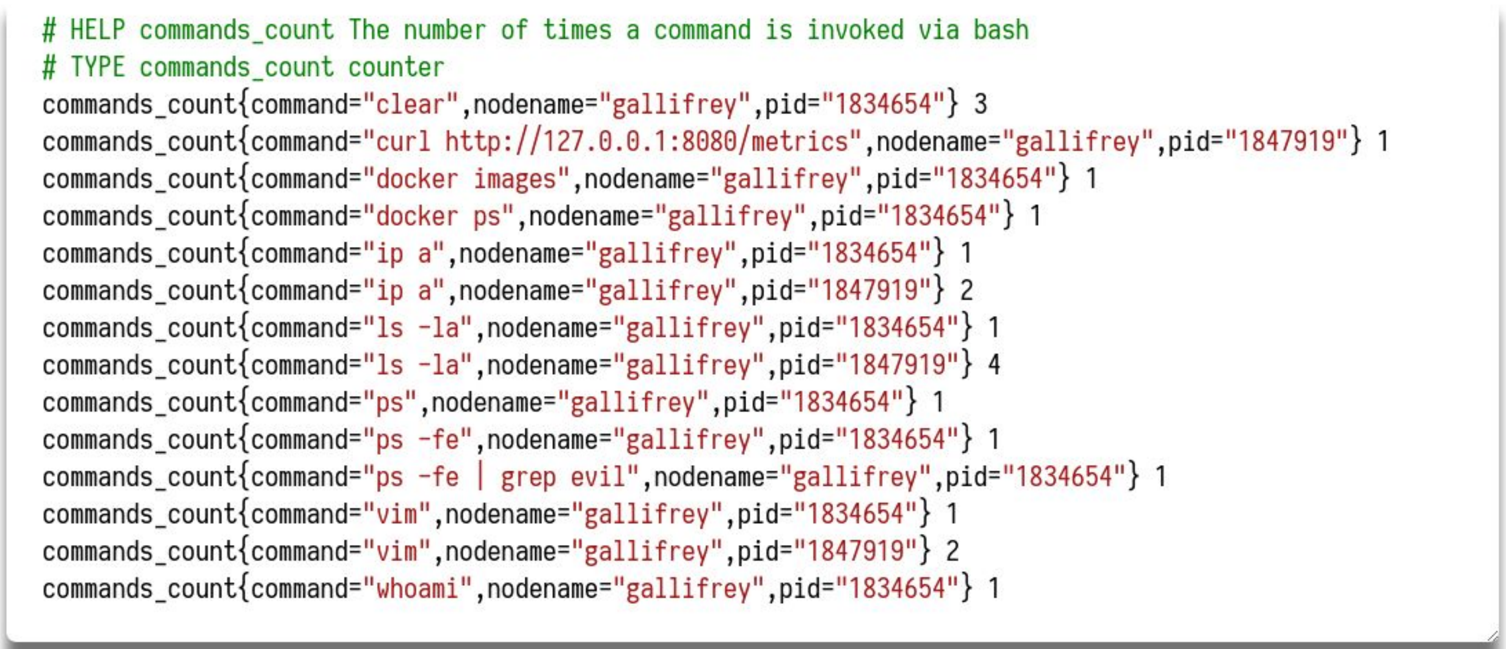
eBPF in a Pod



```
apiVersion: apps/v1
```

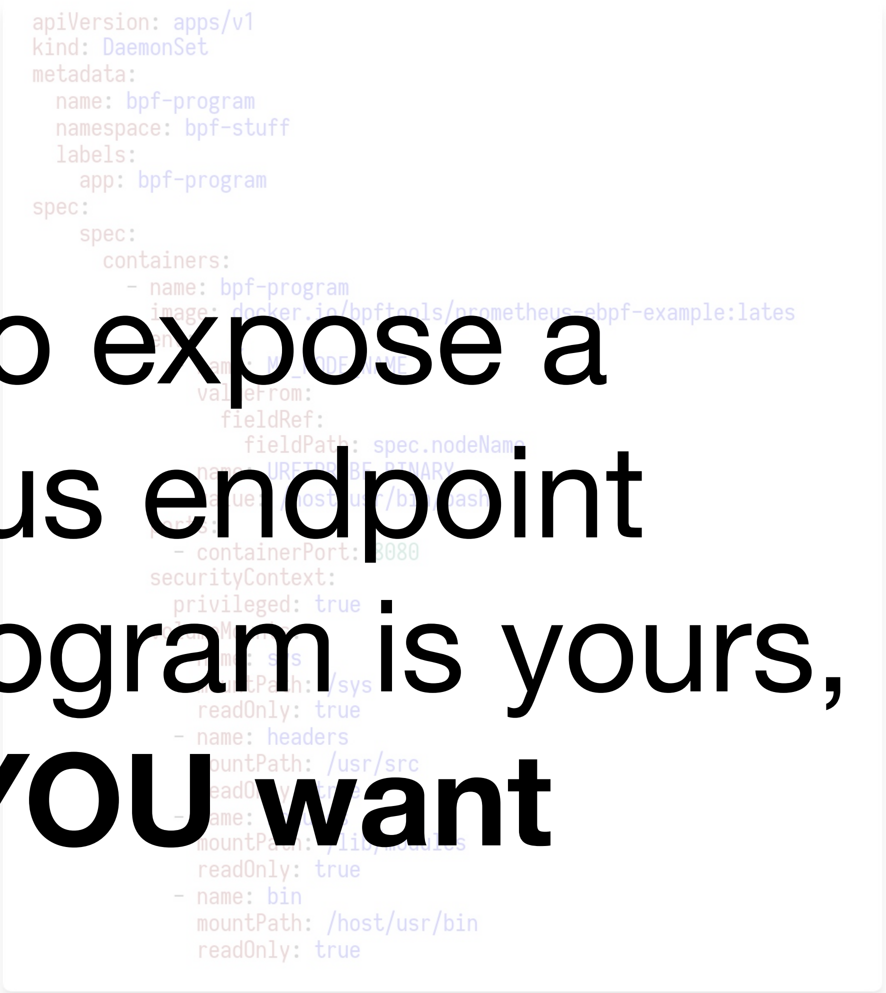
```
curl http://127.0.0.1:8080/metrics
```

```
spec:
    spec:
      containers:
```



```
# HELP commands_count The number of times a command is invoked via bash
# TYPE commands_count counter
commands_count{command="clear",nodename="gallifrey",pid="1834654"} 3
commands_count{command="curl http://127.0.0.1:8080/metrics",nodename="gallifrey",pid="1847919"} 1
commands_count{command="docker images",nodename="gallifrey",pid="1834654"} 1
commands_count{command="docker ps",nodename="gallifrey",pid="1834654"} 1
commands_count{command="ip a",nodename="gallifrey",pid="1834654"} 1
commands_count{command="ip a",nodename="gallifrey",pid="1847919"} 2
commands_count{command="ls -la",nodename="gallifrey",pid="1834654"} 1
commands_count{command="ls -la",nodename="gallifrey",pid="1847919"} 4
commands_count{command="ps",nodename="gallifrey",pid="1834654"} 1
commands_count{command="ps -fe",nodename="gallifrey",pid="1834654"} 1
commands_count{command="ps -fe | grep evil",nodename="gallifrey",pid="1834654"} 1
commands_count{command="vim",nodename="gallifrey",pid="1834654"} 1
commands_count{command="vim",nodename="gallifrey",pid="1847919"} 2
commands_count{command="whoami",nodename="gallifrey",pid="1834654"} 1
```

eBPF in a Pod



```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: bpf-program
  namespace: bpf-stuff
  labels:
    app: bpf-program
spec:
    spec:
      containers:
        - name: bpf-program
          image: docker.io/bpftools/prometheus-ebpf-example:lates
          env:
            - name: MY_NODE_NAME
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            - name: URETPROBE_BINARY
              value: /host/usr/bin/bash
          ports:
            - containerPort: 8080
          securityContext:
            privileged: true
          volumeMounts:
            - name: sys
              mountPath: /sys
              readOnly: true
            - name: headers
              mountPath: /usr/src
              readOnly: true
            - name: modules
              mountPath: /lib/modules
              readOnly: true
            - name: bin
              mountPath: /host/usr/bin
              readOnly: true
```

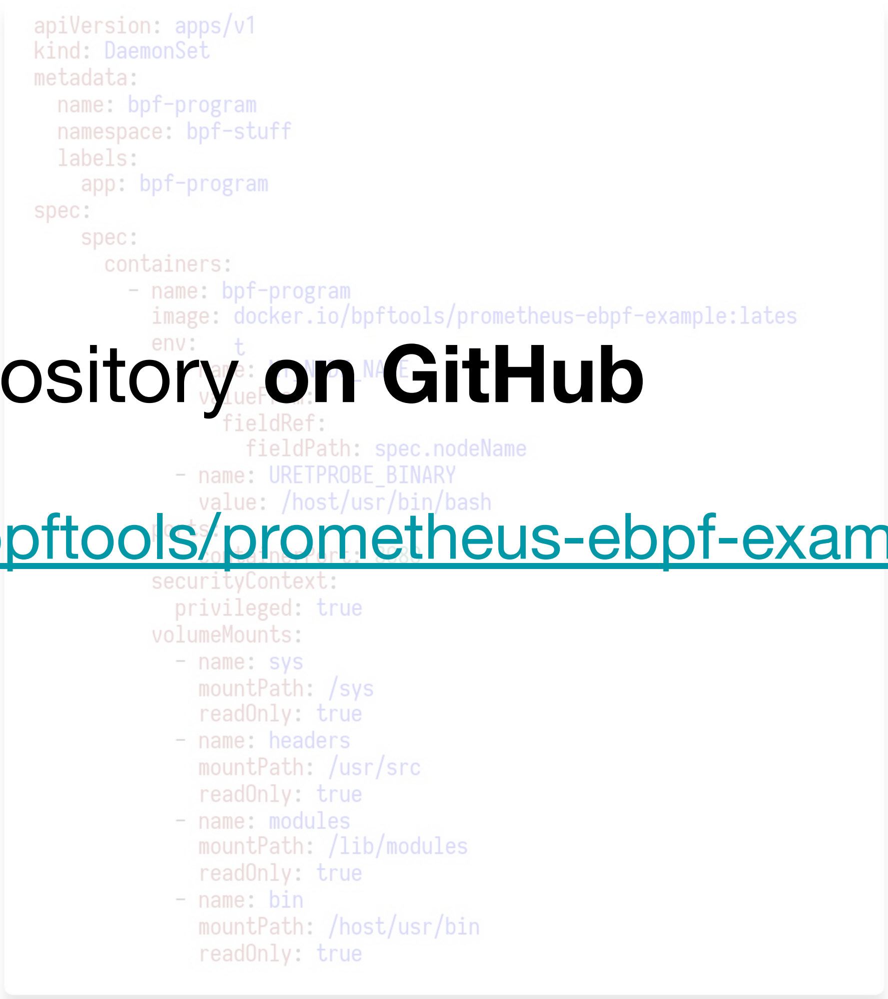
I wanted to expose a Prometheus endpoint but the program is yours, **do what YOU want**

@fntlnz

eBPF in a Pod

Full example repository **on GitHub**

https://github.com/bpftools/prometheus-ebpf-example



```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: bpf-program
  namespace: bpf-stuff
  labels:
    app: bpf-program
spec:
    spec:
      containers:
        - name: bpf-program
          image: docker.io/bpftools/prometheus-ebpf-example:lates
          env:    t
            - name: MY_NODE_NAME
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            - name: URETPROBE_BINARY
              value: /host/usr/bin/bash
          ports:
            - containerPort: 8080
          securityContext:
            privileged: true
          volumeMounts:
            - name: sys
              mountPath: /sys
              readOnly: true
            - name: headers
              mountPath: /usr/src
              readOnly: true
            - name: modules
              mountPath: /lib/modules
              readOnly: true
            - name: bin
              mountPath: /host/usr/bin
              readOnly: true
```

eBPF using a CRD

I’m that Kind: of person

Pros:

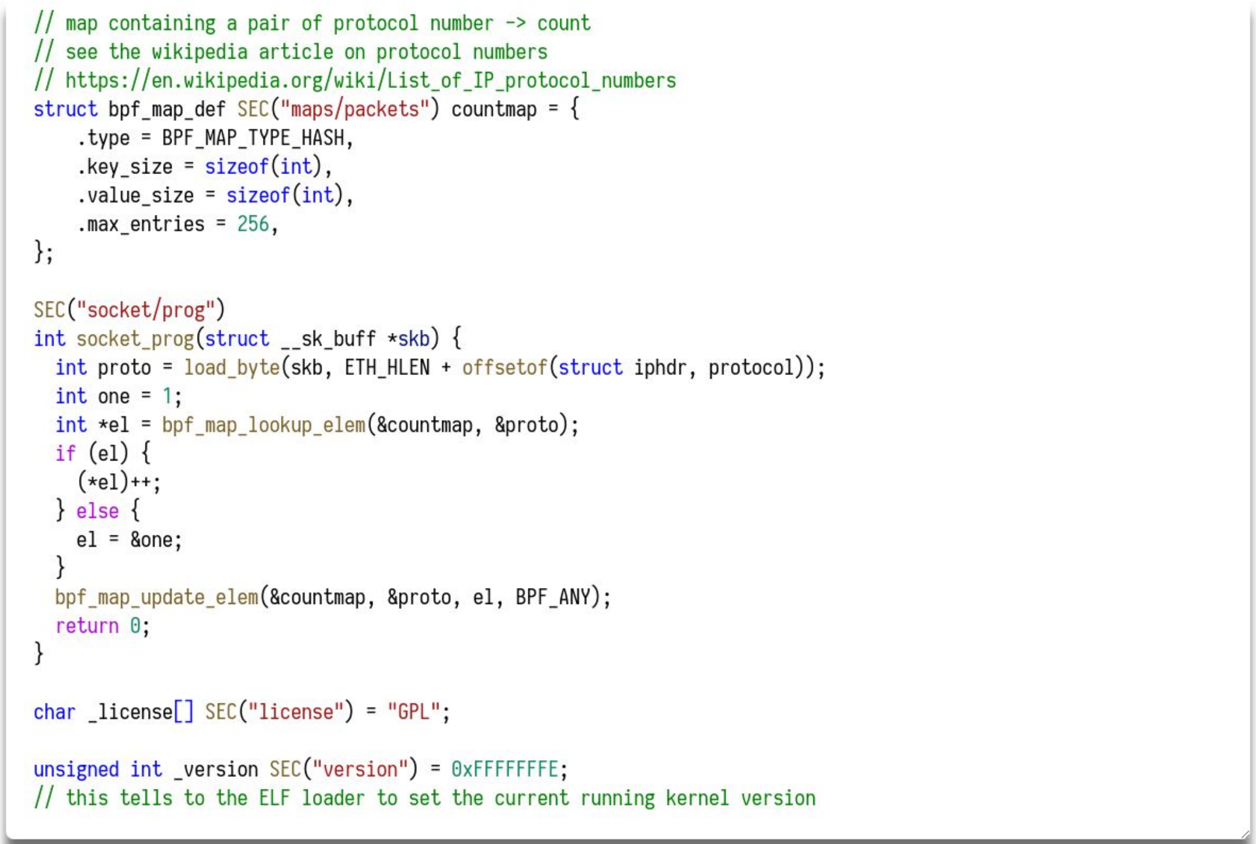
- No boilerplate
- Easy to use
- Automatically expose a Prometheus endpoint for every map you create
- A pod on every node

Cons:

- Need to deploy the Controller
- Not very extensible

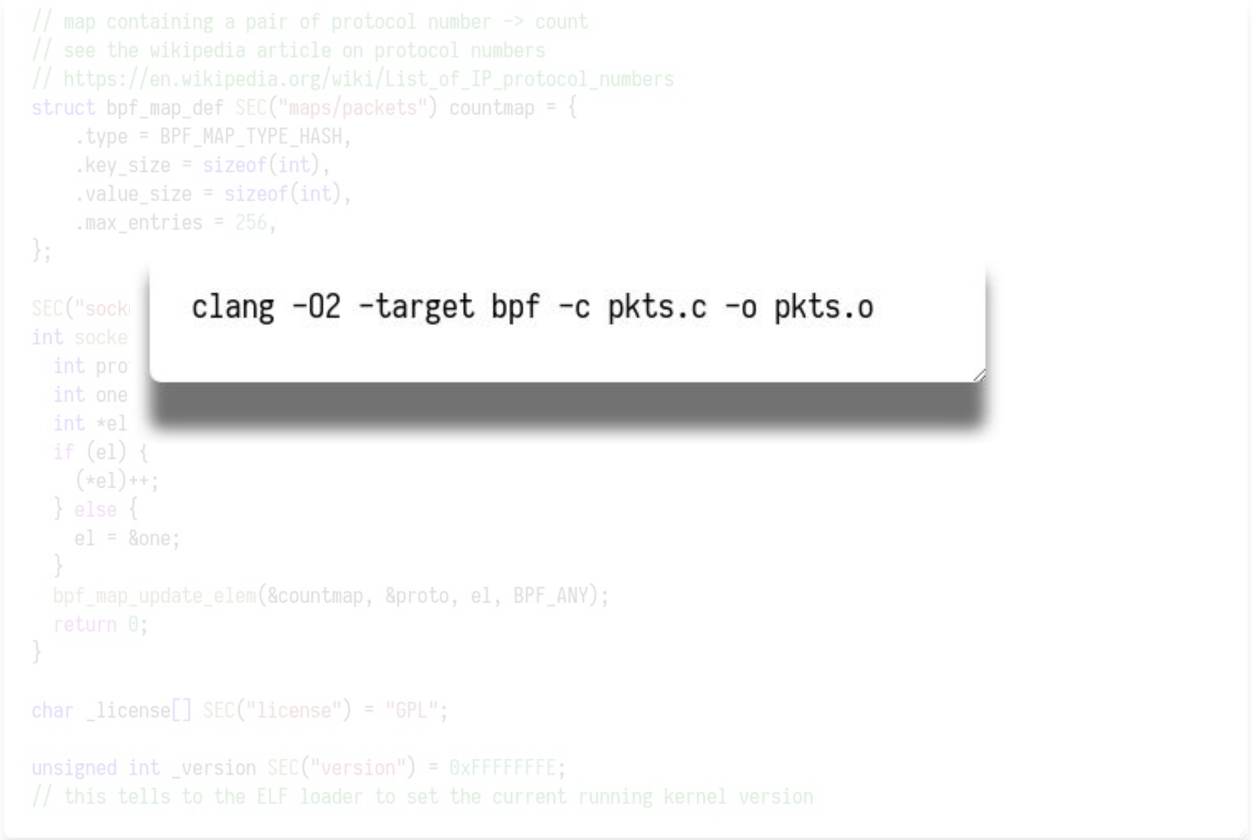
@fntlnz

eBPF using a CRD



```c
// map containing a pair of protocol number -> count
// see the wikipedia article on protocol numbers
// https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers
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
```

eBPF using a CRD



```
// map containing a pair of protocol number -> count
// see the wikipedia article on protocol numbers
// https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers
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("sock
int socke
  int pro
  int one
  int *el
  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
```

```
clang -O2 -target bpf -c pkts.c -o pkts.o
```

@fntlnz

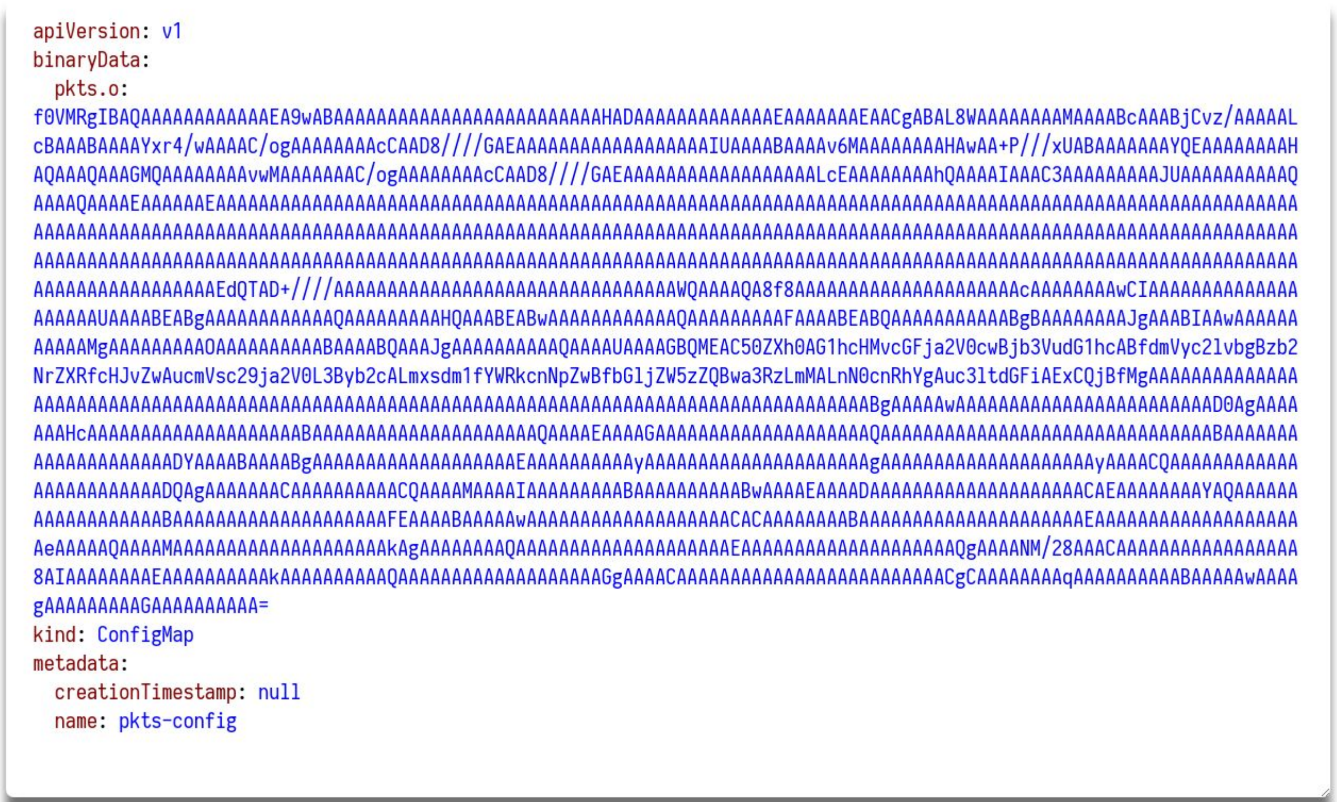
eBPF using a CRD

```
// map containing a pair of protocol number -> count
// see the wikipedia article on protocol numbers
// https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers
struct bpf_map_def SEC("maps/packets") countmap = {
    .type = BPF_MAP_TYPE_HASH,
    .key_size = sizeof(int),
    .value_size = sizeof(int),
```

```
kubectl create configmap --from-file pkts.o pkts -o yaml --dry-run >> "pkts.yaml"
```

```
  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
```

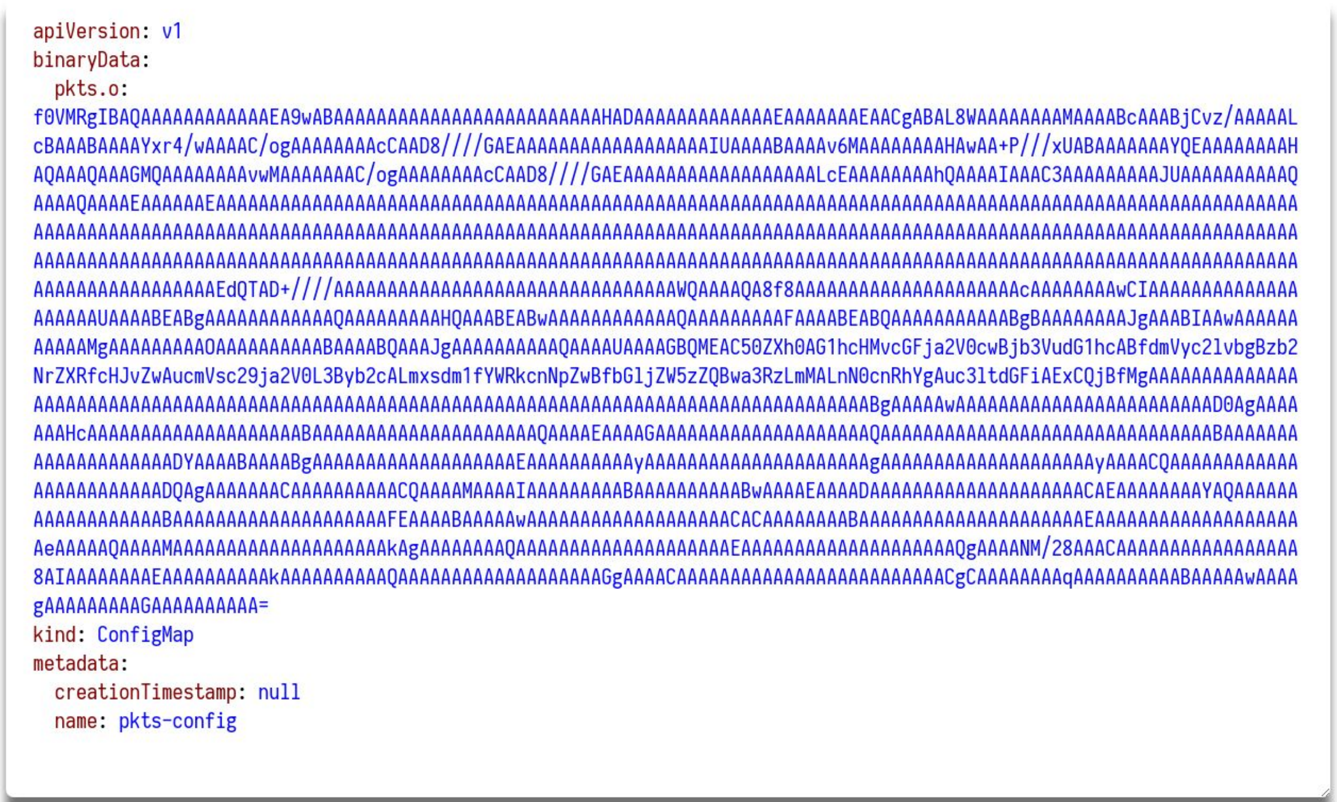
eBPF using a CRD



```yaml
apiVersion: v1
binaryData:
  pkts.o:
f0VMRgIBAQAAAAAAAAAAAAAEA9wABAAAAAAAAAAAAAAAAAAAAAAAAAAAHADAAAAAAAAAAAAAAEAAAAAAAAEAACgABAL8WAAAAAAAAAMAAAAABcAAABjCvz/AAAAAL
cBAAABAAAAYxr4/wAAAAC/ogAAAAAAAAcCAAD8////GAEAAAAAAAAAAAAAAAAIUAAAABAAAAv6MAAAAAAAAAHAwAA+P///xUABAAAAAAAAYQEAAAAAAAAAH
AQAAAQAAAGMQAAAAAAAAvwMAAAAAAAAC/ogAAAAAAAAcCAAD8////GAEAAAAAAAAAAAAAAAAALcEAAAAAAAAAhQAAAAIAAAC3AAAAAAAAAAJUAAAAAAAAAAAQ
AAAAQAAAAEAAAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAEdQTAD+////AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAWQAAAAQA8f8AAAAAAAAAAAAAAAAAAAAAcAAAAAAAAwCIAAAAAAAAAAAAAAA
AAAAAAUAAAABEABgAAAAAAAAAAAAAQAAAAAAAAAAHQAAABEABwAAAAAAAAAAAAQAAAAAAAAAAFAAAABEABQAAAAAAAAAAAABgBAAAAAAAAJgAAABIAAwAAAAAA
AAAAMgAAAAAAAAAOAAAAAAAAAAAABAAAABQAAAJgAAAAAAAAAAAQAAAAUAAAAGBQMEAC50ZXh0AG1hcHMvcGFja2V0cwBjb3VudG1hcABfdmVyc2lvbgBzb2
NrZXRfcHJvZwAucmVsc29ja2V0L3Byb2cALmxsdm1fYWRkcnNpZwBfbGljZW5zZQBwa3RzLmMALnN0cnRhYgAuc3ltdGFiAExCQjBfMgAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABgAAAAAwAAAAAAAAAAAAAAAAAAAAAAAAAD0AgAAAA
AAAHcAAAAAAAAAAAAAAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAAQAAAAEAAAAGAAAAAAAAAAAAAAAAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABAAAAAAA
AAAAAAAAAAAAADYAAAABAAAABgAAAAAAAAAAAAAAAAAAAAAAEAAAAAAAAAAyAAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAAAAAAAAAAAAAAyAAAACQAAAAAAAAAAAAAA
AAAAAAAAAAAADQAgAAAAAAACAAAAAAAAAAACQAAAAMAAAAIAAAAAAAAAABAAAAAAAAAAABwAAAAEAAAADAAAAAAAAAAAAAAAAAAAAACAEAAAAAAAAYAQAAAAAAA
AAAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAFEAAAABAAAAAwAAAAAAAAAAAAAAAAAAAAACACAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAAEAAAAAAAAAAAAAAAAAAAA
AeAAAAAQAAAAMAAAAAAAAAAAAAAAAAAAAAAkAgAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAAAQgAAAANM/28AAACAAAAAAAAAAAAAAAAAAAA
8AIAAAAAAAAAEAAAAAAAAAAAkAAAAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAGgAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAACgCAAAAAAAAAqAAAAAAAAAAAABAAAAAAwAAAA
gAAAAAAAAAGAAAAAAAAAAAA=
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: pkts-config
```

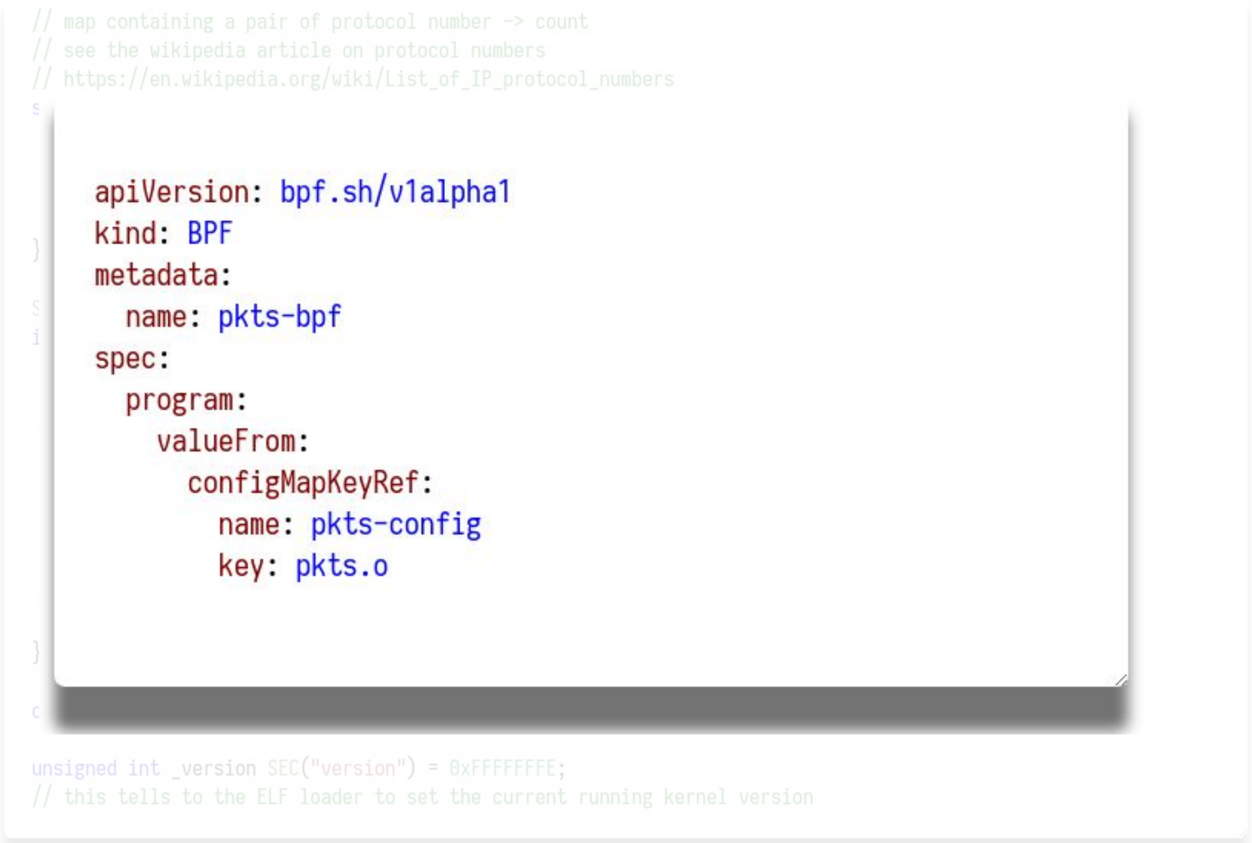
eBPF using a CRD

Base64 ELF



```yaml
apiVersion: v1
binaryData:
  pkts.o:
f0VMRgIBAQAAAAAAAAAAAAAEA9wABAAAAAAAAAAAAAAAAAAAAAAAAAAHADAAAAAAAAAAAAAAEAAAAAAAAEAACgABAL8WAAAAAAAAAMAAAABcAAABjCvz/AAAAAL
cBAAABAAAAYxr4/wAAAAC/ogAAAAAAAAcCAAD8////GAEAAAAAAAAAAAAAAAAIUAAAABAAAAv6MAAAAAAAAAHAwAA+P///xUABAAAAAAAYQEAAAAAAAAH
AQAAAQAAAGMQAAAAAAAAAvwMAAAAAAAC/ogAAAAAAAAcCAAD8////GAEAAAAAAAAAAAAAAAAALcEAAAAAAAAAhQAAAAIAAAC3AAAAAAAAAAJUAAAAAAAAAAQ
AAAAQAAAAEAAAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAEdQTAD+////AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAWQAAAAQA8f8AAAAAAAAAAAAAAAAAAAAAAAcAAAAAAAAAwCIAAAAAAAAAAAAAAAA
AAAAAAUAAAABEABgAAAAAAAAAAAAAAQAAAAAAAAAAAHQAAABEABwAAAAAAAAAAAAAAAQAAAAAAAAAAAFAAAABEABQAAAAAAAAAAAAABgBAAAAAAAAAJgAAABIAAwAAAAAAA
AAAAAMgAAAAAAAAAAOAAAAAAAAAAAABAAAABQAAAJgAAAAAAAAAAAAQAAAAUAAAAGBQMEAC50ZXh0AG1hcHMvcGFja2V0cwBjb3VudG1hcABfdmVyc2lvbgBzb2
NrZXRfcHJvZwAucmVsc29ja2V0L3Byb2cALmxsdm1fYWRkcnNpZwBfbGljZW5zZQBwa3RzLmMALnN0cnRhYgAuc3ltdGFiAExCQjBfMgAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABgAAAAAwAAAAAAAAAAAAAAAAAAAAAAAAAAAD0AgAAAA
AAAHcAAAAAAAAAAAAAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAQAAAAEAAAAGAAAAAAAAAAAAAAAAAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABAAAAAAA
AAAAAAAAAAAAADYAAAABAAAABgAAAAAAAAAAAAAAAAAAAAAEAAAAAAAAAAAyAAAAAAAAAAAAAAAAAAAAAAAgAAAAAAAAAAAAAAAAAAAAAAyAAAACQAAAAAAAAAAAAAAA
AAAAAAAAAAAAADQAgAAAAAAACAAAAAAAAAAACQAAAAMAAAAIAAAAAAAAAABAAAAAAAAAAABwAAAAEAAAADAAAAAAAAAAAAAAAAAAAAAACAEAAAAAAAAYAQAAAAAAA
AAAAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAFEAAAABAAAAAwAAAAAAAAAAAAAAAAAAAAACACAAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAEAAAAAAAAAAAAAAAAAAA
AeAAAAAQAAAAMAAAAAAAAAAAAAAAAAAAAAAAAAkAgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAAAQgAAAANM/28AAACAAAAAAAAAAAAAAAAAA
8AIAAAAAAAAAEAAAAAAAAAAAAkAAAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAGgAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAACgCAAAAAAAAAqAAAAAAAAAAABAAAAAwAAAA
gAAAAAAAAAGAAAAAAAAAAA=
kind: ConfigMap
metadata:
  creationTimestamp: null
  name: pkts-config
```

eBPF using a CRD

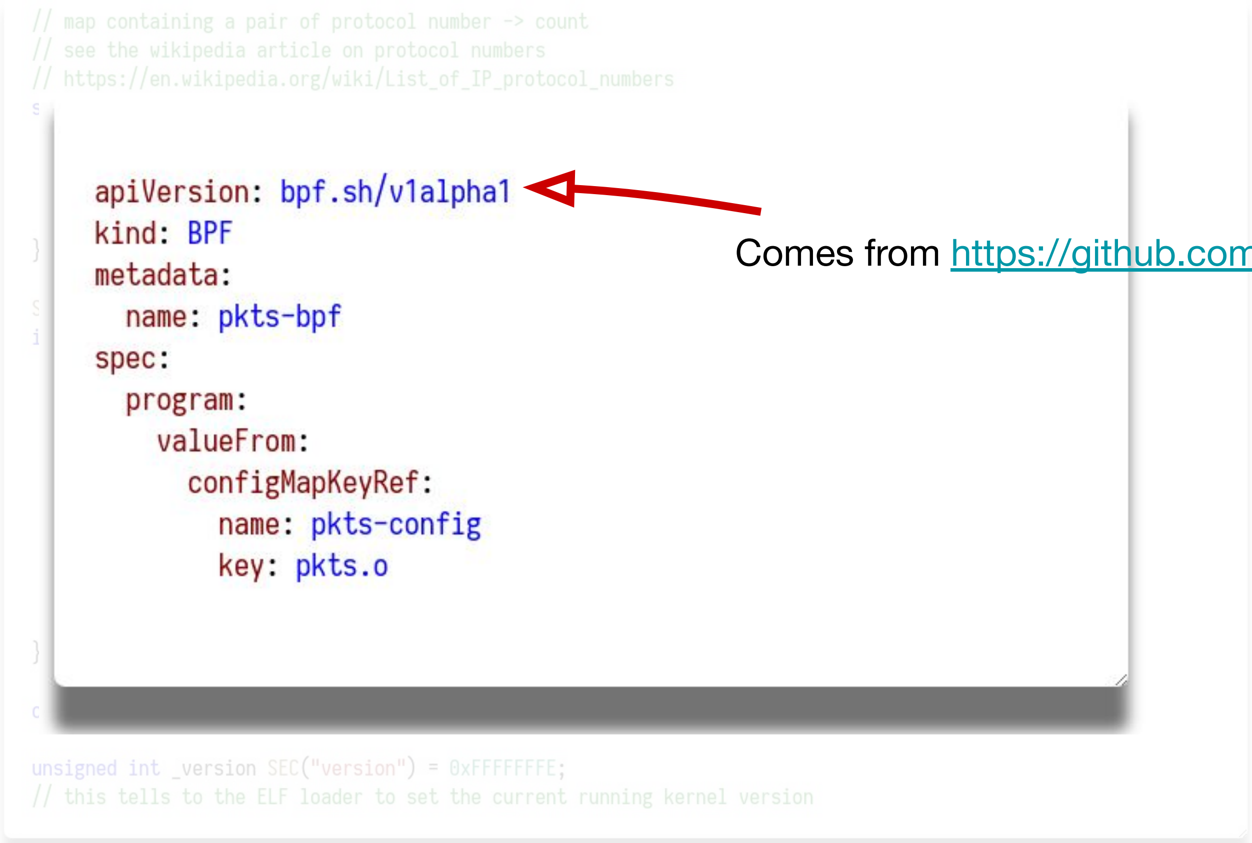


```
// map containing a pair of protocol number -> count
// see the wikipedia article on protocol numbers
// https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers
```

```yaml
apiVersion: bpf.sh/v1alpha1
kind: BPF
metadata:
  name: pkts-bpf
spec:
  program:
    valueFrom:
      configMapKeyRef:
        name: pkts-config
        key: pkts.o
```

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

eBPF using a CRD



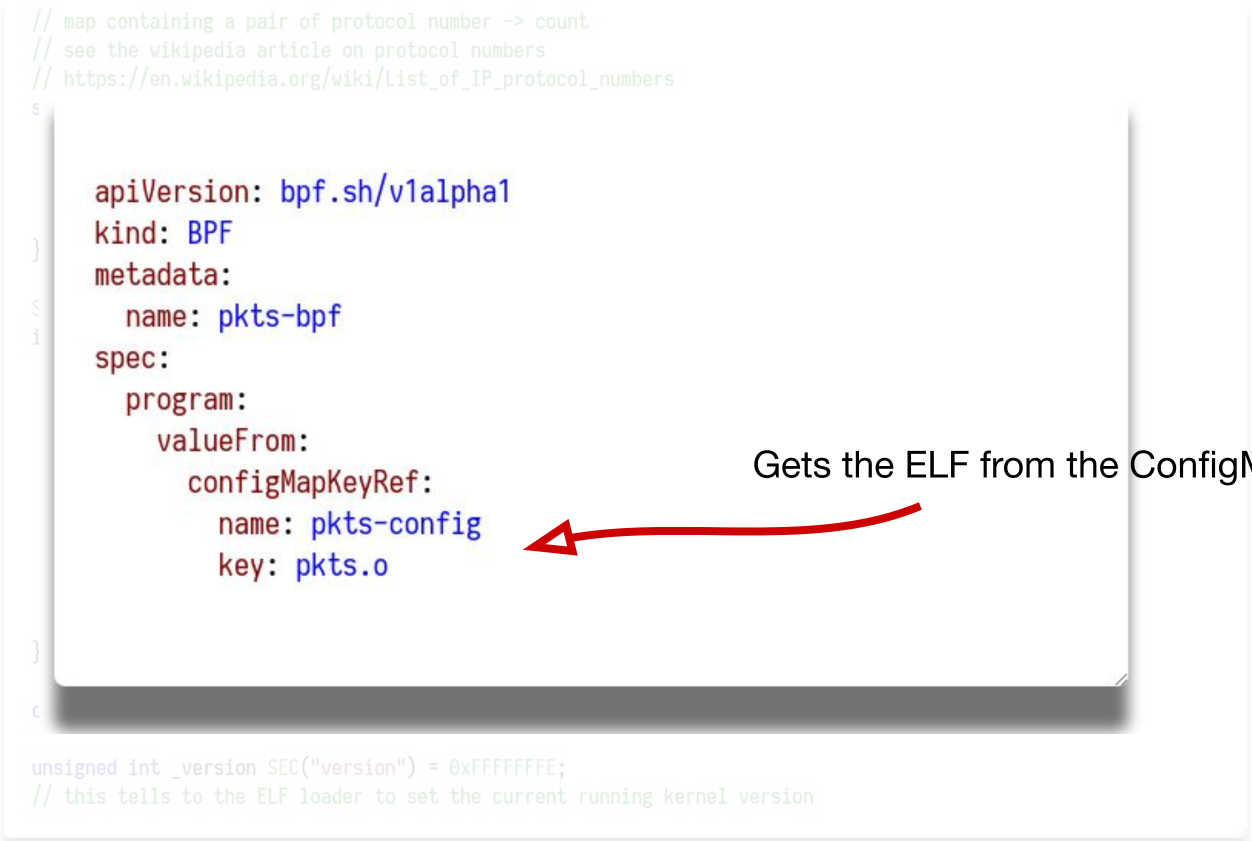
```
// map containing a pair of protocol number -> count
// see the wikipedia article on protocol numbers
// https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers
```

```yaml
apiVersion: bpf.sh/v1alpha1
kind: BPF
metadata:
  name: pkts-bpf
spec:
  program:
    valueFrom:
      configMapKeyRef:
        name: pkts-config
        key: pkts.o
```

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

Comes from https://github.com/bpftools/kube-bpf

eBPF using a CRD



```
// map containing a pair of protocol number -> count
// see the wikipedia article on protocol numbers
// https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers
```

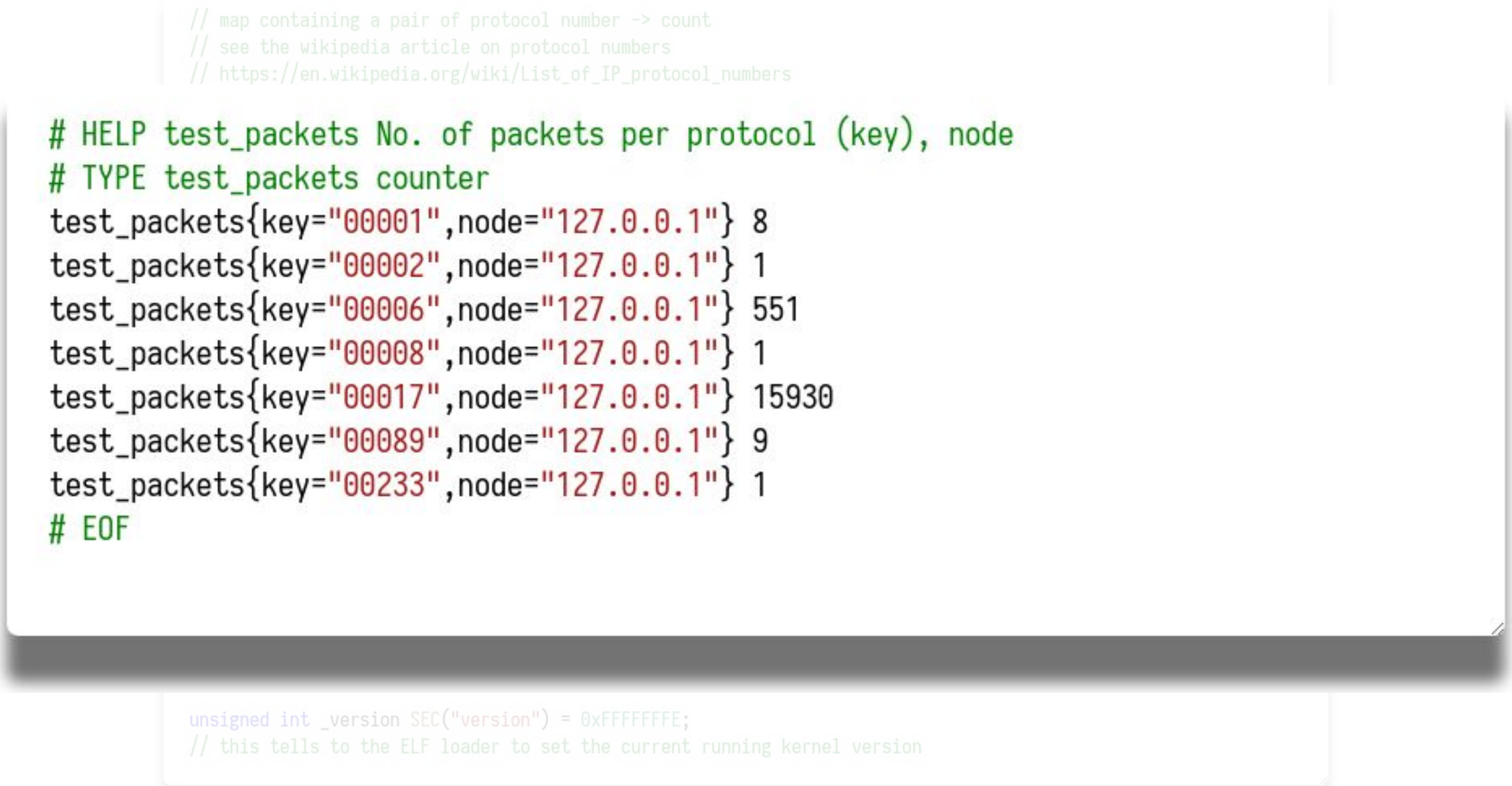
```yaml
apiVersion: bpf.sh/v1alpha1
kind: BPF
metadata:
  name: pkts-bpf
spec:
  program:
    valueFrom:
      configMapKeyRef:
        name: pkts-config
        key: pkts.o
```

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

Gets the ELF from the ConfigMap

@fntlnz

eBPF using a CRD



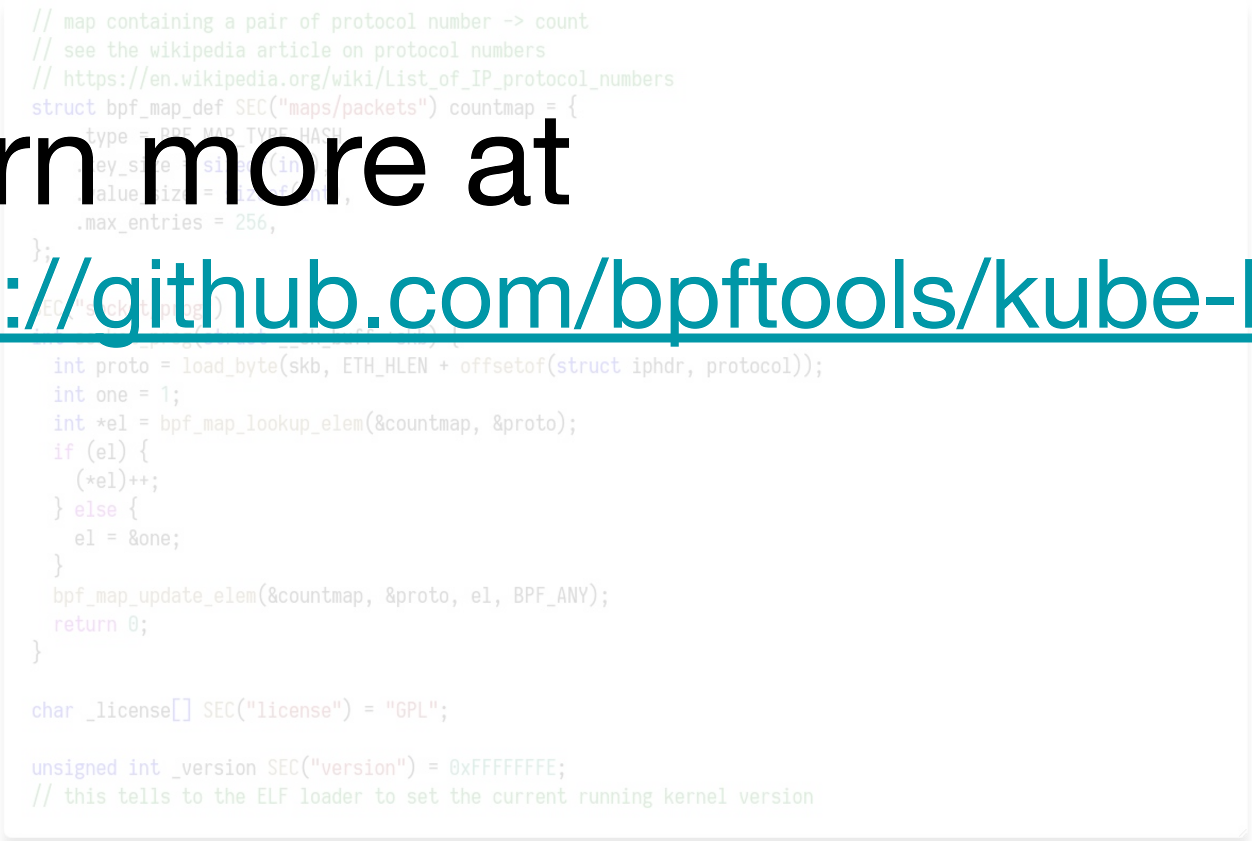
```
// map containing a pair of protocol number -> count
// see the wikipedia article on protocol numbers
// https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers
```

```
# HELP test_packets No. of packets per protocol (key), node
# TYPE test_packets counter
test_packets{key="00001",node="127.0.0.1"} 8
test_packets{key="00002",node="127.0.0.1"} 1
test_packets{key="00006",node="127.0.0.1"} 551
test_packets{key="00008",node="127.0.0.1"} 1
test_packets{key="00017",node="127.0.0.1"} 15930
test_packets{key="00089",node="127.0.0.1"} 9
test_packets{key="00233",node="127.0.0.1"} 1
# EOF
```

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

@fntlnz

eBPF using a CRD



```
// map containing a pair of protocol number -> count
// see the wikipedia article on protocol numbers
// https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers
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
```

Learn more at

https://github.com/bpftools/kube-bpf

eBPF in the kubectl

Like DTrace but for kubernetes

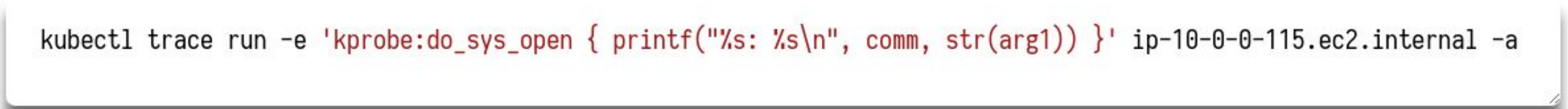
Pros:

- Uses the bpftrace DSL
- Very powerful
- Unix philosophy

Cons:

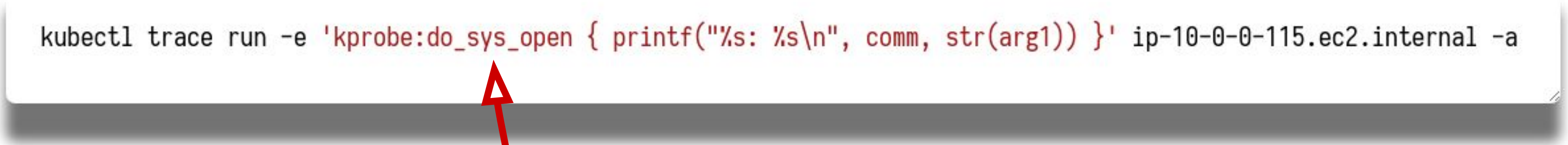
- Can only do what bpftrace can do
- No custom logic, just use the DSL

eBPF in the kubectl



```
kubectl trace run -e 'kprobe:do_sys_open { printf("%s: %s\n", comm, str(arg1)) }' ip-10-0-0-115.ec2.internal -a
```

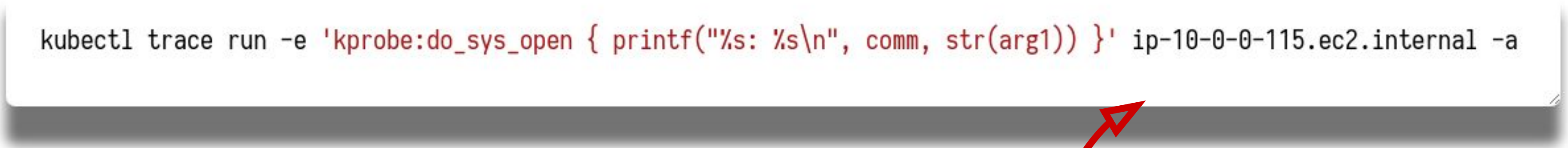
eBPF in the kubectl



```
kubectl trace run -e 'kprobe:do_sys_open { printf("%s: %s\n", comm, str(arg1)) }' ip-10-0-0-115.ec2.internal -a
```

Every time the open syscall is executed print the opened file name

eBPF in the kubectl

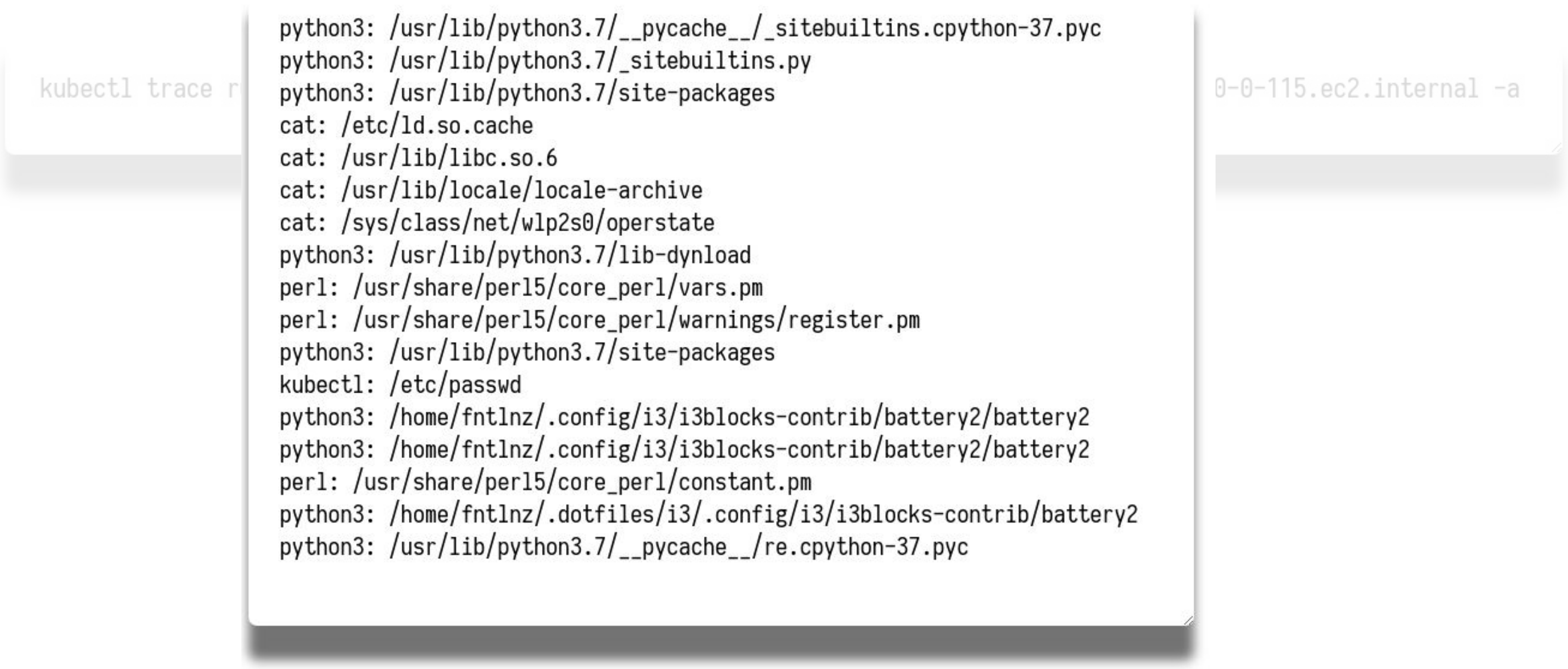


```
kubectl trace run -e 'kprobe:do_sys_open { printf("%s: %s\n", comm, str(arg1)) }' ip-10-0-0-115.ec2.internal -a
```

Only on this specific node

@fntlnz

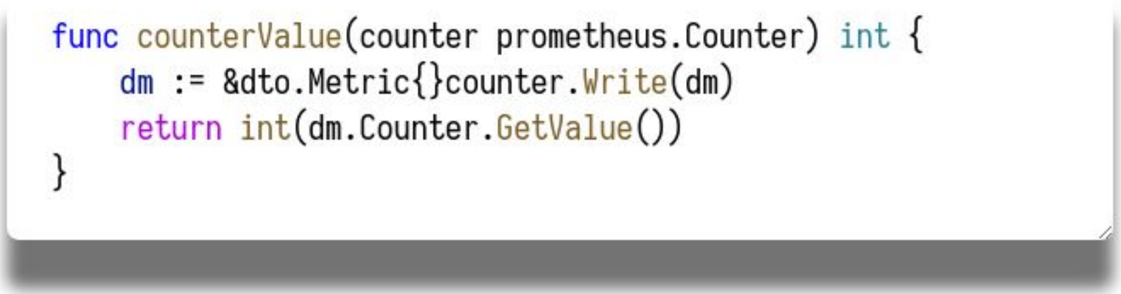
eBPF in the kubectl



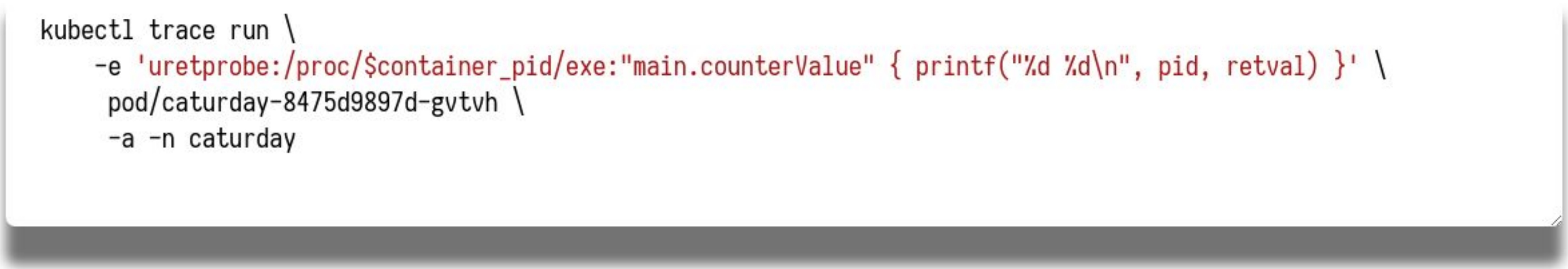
```
kubectl trace r
0-0-115.ec2.internal -a
```

```
python3: /usr/lib/python3.7/__pycache__/_sitebuiltins.cpython-37.pyc
python3: /usr/lib/python3.7/_sitebuiltins.py
python3: /usr/lib/python3.7/site-packages
cat: /etc/ld.so.cache
cat: /usr/lib/libc.so.6
cat: /usr/lib/locale/locale-archive
cat: /sys/class/net/wlp2s0/operstate
python3: /usr/lib/python3.7/lib-dynload
perl: /usr/share/perl5/core_perl/vars.pm
perl: /usr/share/perl5/core_perl/warnings/register.pm
python3: /usr/lib/python3.7/site-packages
kubectl: /etc/passwd
python3: /home/fntlnz/.config/i3/i3blocks-contrib/battery2/battery2
python3: /home/fntlnz/.config/i3/i3blocks-contrib/battery2/battery2
perl: /usr/share/perl5/core_perl/constant.pm
python3: /home/fntlnz/.dotfiles/i3/.config/i3/i3blocks-contrib/battery2
python3: /usr/lib/python3.7/__pycache__/re.cpython-37.pyc
```

eBPF in the kubectl



```go
func counterValue(counter prometheus.Counter) int {
    dm := &dto.Metric{}counter.Write(dm)
    return int(dm.Counter.GetValue())
}
```

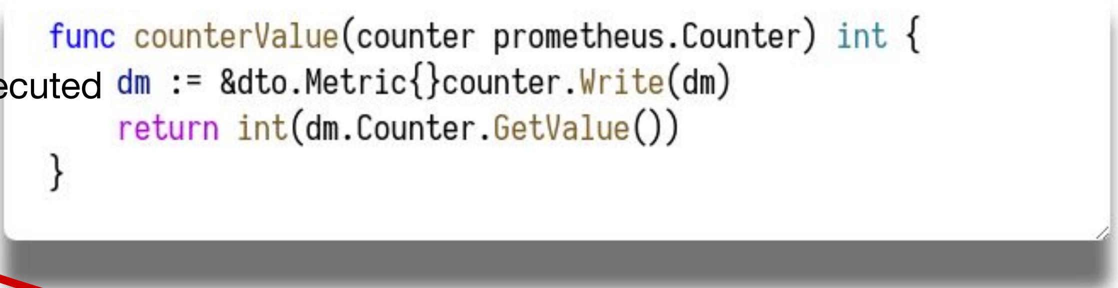


```
kubectl trace run \
    -e 'uretprobe:/proc/$container_pid/exe:"main.counterValue" { printf("%d %d\n", pid, retval) }' \
     pod/caturday-8475d9897d-gvtvh \
     -a -n caturday
```

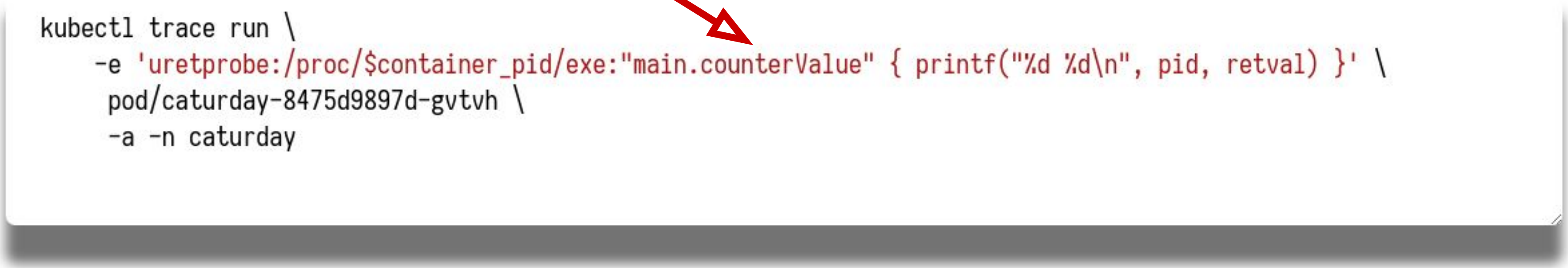
@fntlnz

eBPF in the kubectl

Every time the function is executed print the return value

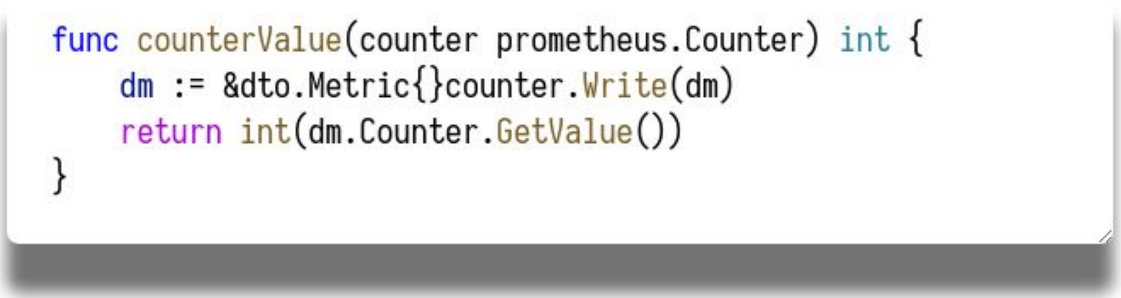


```
func counterValue(counter prometheus.Counter) int {
	dm := &dto.Metric{}counter.Write(dm)
	return int(dm.Counter.GetValue())
}
```

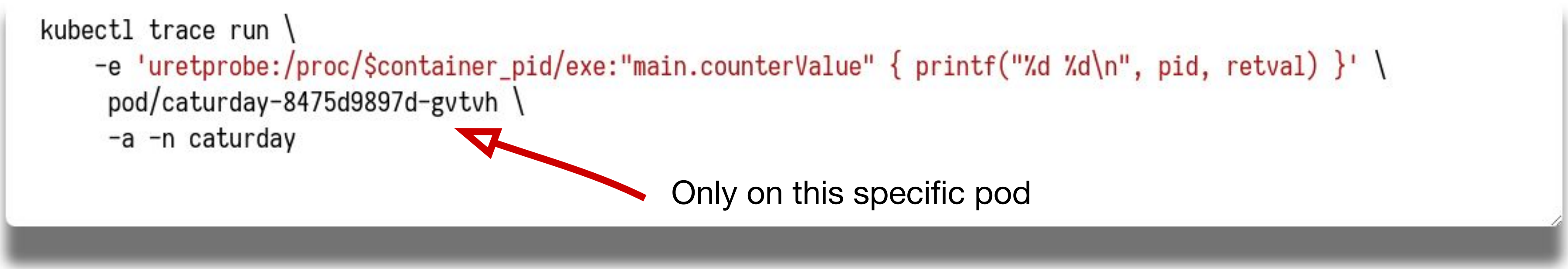


```
kubectl trace run \
    -e 'uretprobe:/proc/$container_pid/exe:"main.counterValue" { printf("%d %d\n", pid, retval) }' \
     pod/caturday-8475d9897d-gvtvh \
     -a -n caturday
```

eBPF in the kubectl



```go
func counterValue(counter prometheus.Counter) int {
    dm := &dto.Metric{}counter.Write(dm)
    return int(dm.Counter.GetValue())
}
```



```
kubectl trace run \
    -e 'uretprobe:/proc/$container_pid/exe:"main.counterValue" { printf("%d %d\n", pid, retval) }' \
     pod/caturday-8475d9897d-gvtvh \
     -a -n caturday
```

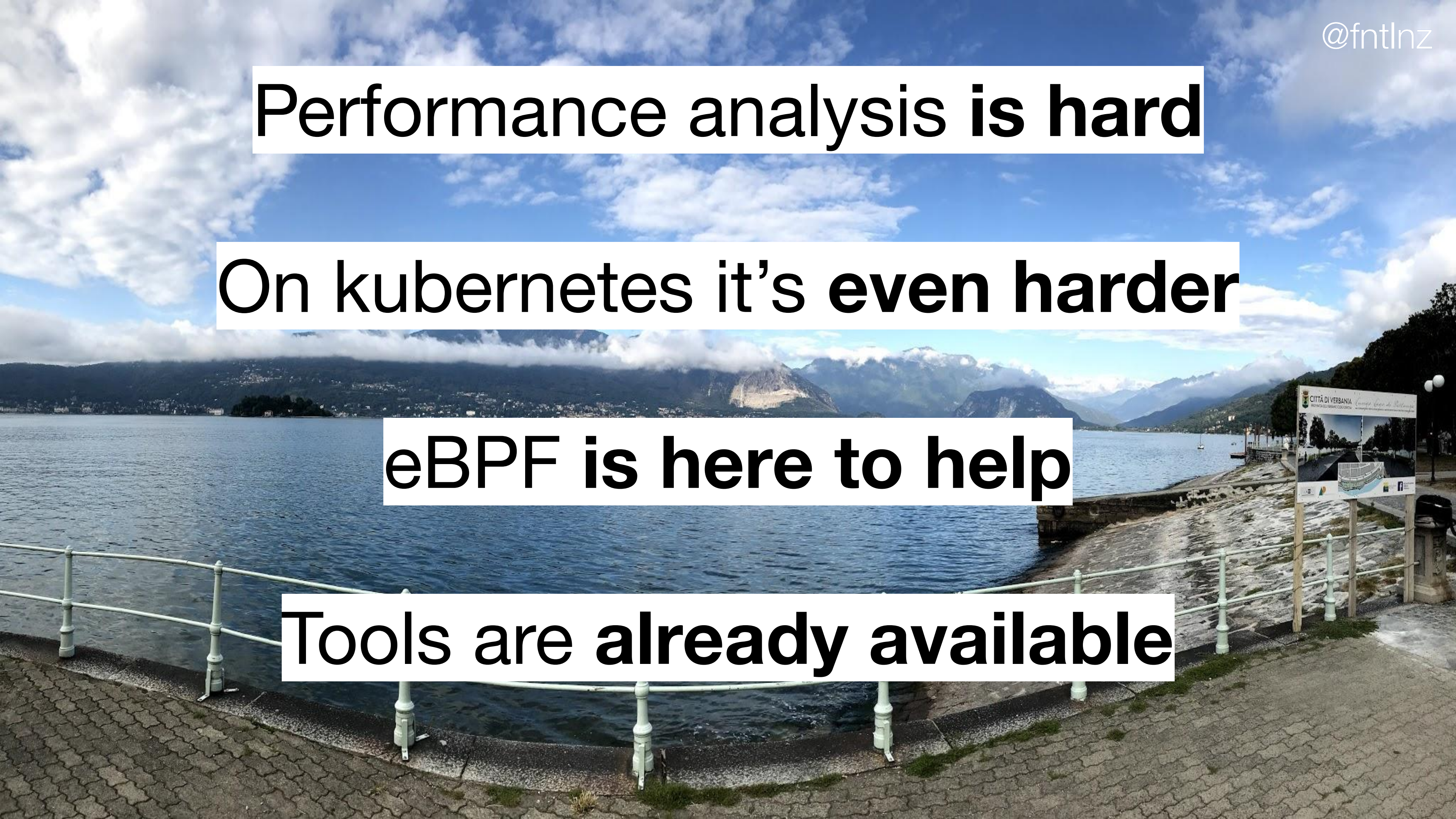
Only on this specific pod

eBPF in the kubectl

```
// map containing a pair of protocol number -> count
// see the wikipedia article on protocol numbers
// https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers
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
```

Learn more at

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



Performance analysis **is hard**

On kubernetes it's **even harder**

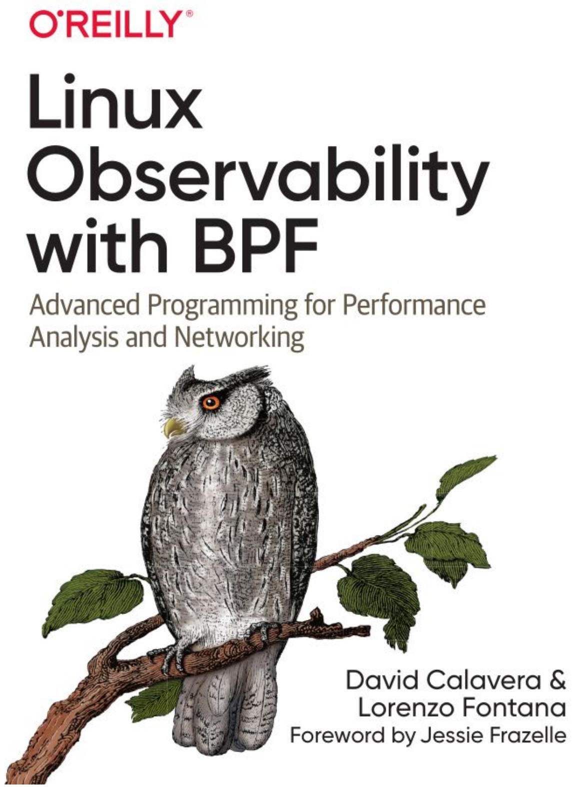
eBPF **is here to help**

Tools are **already available**

Kubernetes eBPF links for y'all

- https://github.com/bpftools/kube-bpf
- https://github.com/iovisor/kubectl-trace
- https://github.com/falcosecurity/falco
- https://github.com/draios/sysdig
- https://github.com/bpftools/linux-observability-with-bpf

Linux Observability with BPF



O'REILLY

Linux Observability with BPF

Advanced Programming for Performance Analysis and Networking

David Calavera & Lorenzo Fontana

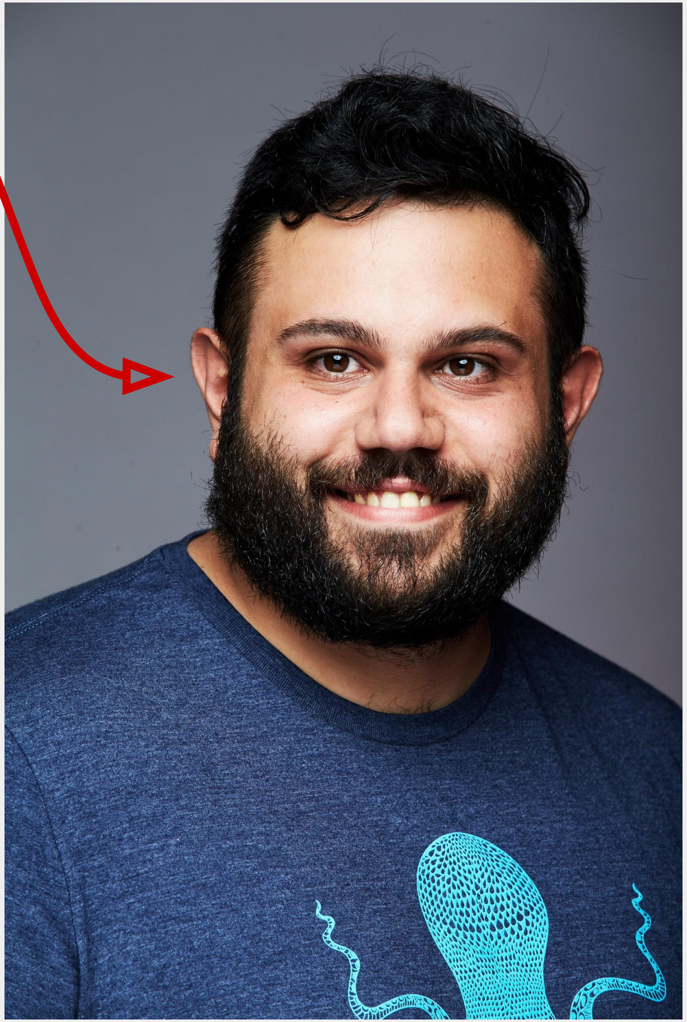
Foreword by Jessie Frazelle

- Get the PDF for free (link below)
- From me and **David Calavera** (@calavera)
- Foreword by **Jessie Frazelle** (@jessfraz)
- There's stuff I learned in it
- It's **complimentary** to this talk
- Still looking at this slide, **go get your copy**

FREE COPY COURTESY OF SYSDIG

setns.run/bpf-book

Friendly person



Thanks

Tweets at **@fntlnz**

My DMs are open!