KubeCon | CloudNativeCon

North America 2023

KubeCon | CloudNativeCon

North America 2023

Collecting Low-Level Metrics with eBPF

Mauricio Vásquez-Bernal
Microsoft

Agenda

- Metrics
- eBPF
- Metrics + ebpf
- Projects
  - ebpf_exporter
  - Tetragon
  - Inspektor Gadget

Metrics

Definition

- Measurement of a service captured at runtime
- Numerical data points that represent the health and performance of a system
  - Response time
  - Memory/cpu utilization
  - Error rate
  - Throughput
  - Anything else specific to your application

Why?

- Indicate availability and performance
- Used to alert and outage
- Trigger scheduling decisions

Metrics Types

- Counter
  - Single numerical value that can only be increased
  - Examples
    - Number of packets sent
    - Total of requests processed
- Gauge
  - Single numerical value that can go up and down
  - Examples
    - Number of open connections
    - Memory usage
- Histogram
  - Sample observations and counts them in configurable buckets
  - Examples
    - HTTP server response times
    - Disk I/O latency

Metrics Dimensions

- Attributes associated with the metrics
- Record additional information about the metric, not only the value
  - Network interface name and IP version for packets sent
    - packets_counter[eth0,ipv4] = 5000
    - packets_counter[eth0,ipv6] = 1520
    - packets_counter[eth1,ipv4] = 1
    - packets_counter[eth1,ipv6] = 200
- Used to aggregate and filter data
  - Get packets sent by interface name
- Cardinality refers to number of possible combinations of the attributes
  - Example above: 2 (iface name) x 2 (ip version) = 4
  - Cardinality highly impacts the resource usage of the observability systems

eBPF

What's eBPF?

- In-kernel bytecode virtual machine
- Allows to change kernel behavior without recompiling it
- Used for different purposes such as
    - Tracing
    - Networking
    - Security

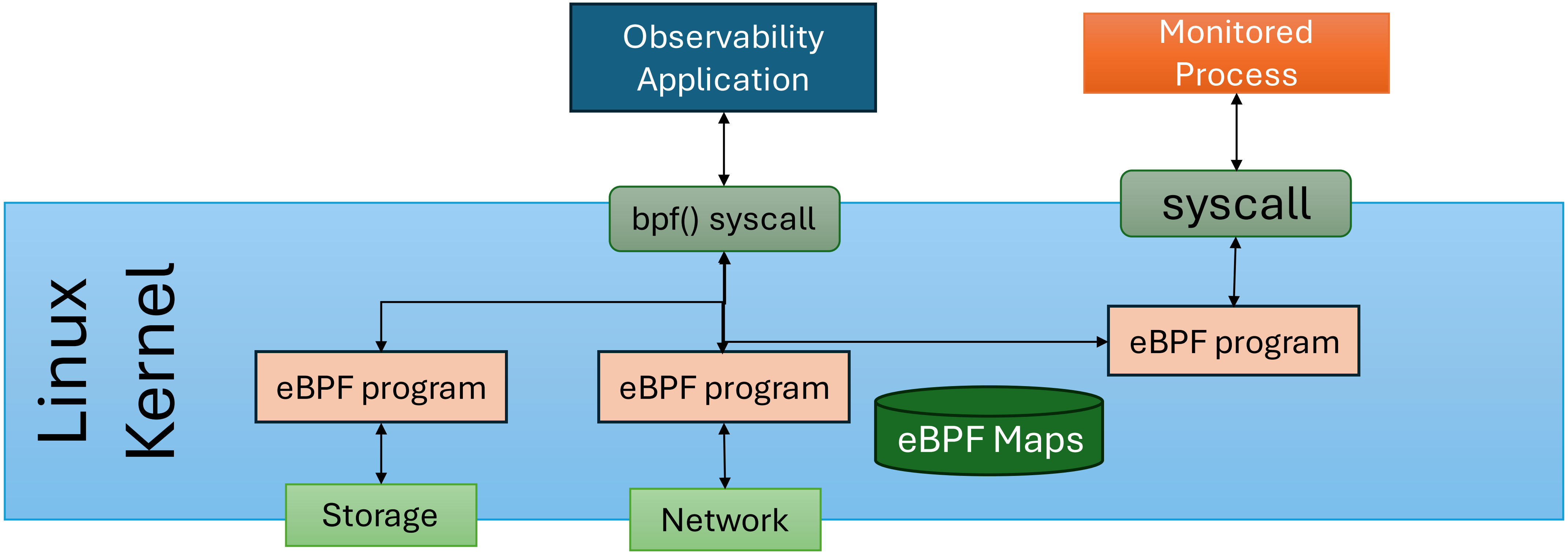
Why eBPF?

- Brings flexibility to the kernel
  - ○ No need to wait for a new kernel release to implement a new feature
- It's efficient
  - ○ Just-in-Time (JIT) compiler makes the performance overhead low
- It's safe
  - ○ User provided code runs in a "sandbox" environment in the kernel

eBPF Hooks

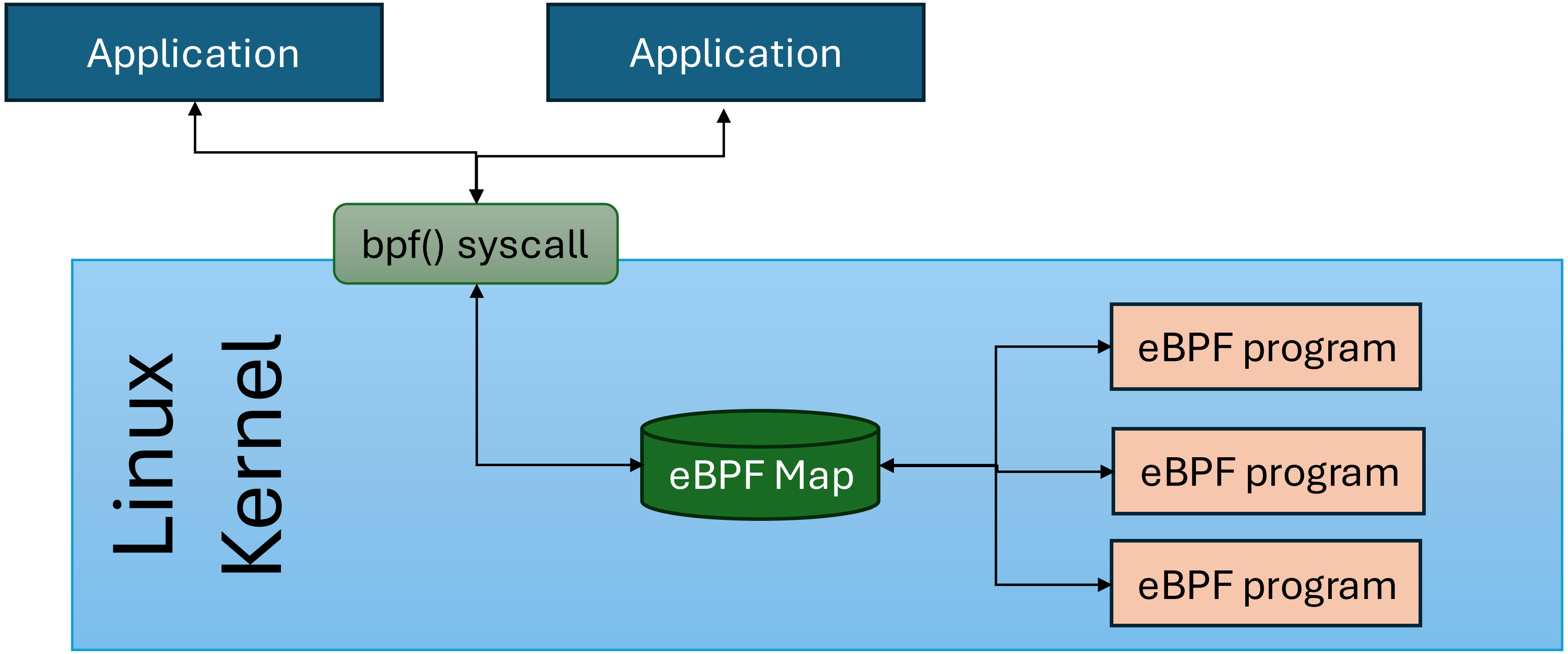
- eBPF programs are event-driven
- Program is executed each time a given event happens
- Those events are known as “hooks”
- Examples:
  - Kprobes/tracepoints
  - Network devices
  - Sockets
  - Linux Security Modules (LSM)
  - Etc.

eBPF Hooks


Observability Application
Monitored Process
bpf() syscall
syscall
Linux Kernel
eBPF program
eBPF program
eBPF program
eBPF Maps
Storage
Network

eBPF Maps

- Key/Value structures to share information between eBPF programs and user space applications



Metrics + eBPF

- eBPF can provide deep insights about the Linux kernel
- Itṡ flexibility, efficiency and safety makes it a perfect tool to collect low level metrics
- Different projects provide metrics by using eBPF
  - opentelemetry-ebpf
  - **ebpf_exporter**
  - **Tetragon**
  - **Inspektor Gadget**
  - Many more

ebpf_exporter

https://github.com/cloudflare/ebpf_exporter

ebpf_exporter

- Prometheus exporter for **custom** eBPF metrics.
- Motivation of this exporter is to allow **you to write** eBPF code and export metrics that are not otherwise accessible from the Linux kernel
- Metrics supported:
  - Counters
  - Histograms
- User needs to create two things:
  - eBPF program: Pulls information from the kernel and writes it to an eBPF map
  - Configuration file: Describes how metrics are stored in the eBPF maps

Configuration File: Metrics

```
metrics:
  counters:
    - name: <prometheus counter name>
      help: <prometheus metric help>
      labels:
        [ - label ]
  histograms:
    - name: <prometheus histogram name>
      help: <prometheus metric help>
      bucket_type: <map bucket type: exp2 or linear>
      bucket_multiplier: <map bucket multiplier: float64>
      bucket_min: <min bucket value: int>
      bucket_max: <max bucket value: int>
      labels:
        [ - label ]
```

Configuration File: Labels

- Transform kernel map keys into Prometheus labels
- Data coming from the kernel is always binary encoded
  - Keys can be primitive types like u64 of complex structs
- Labels are transformed using **decoders**
  - Transform byte slice into Prometheus label
    - cgroup ID -> cgroup path

```
labels:
  - name: cgroup_path
    size: 8
    decoders:
      - name: uint
      - name: cgroup
```

Demo

Tetragon

https://github.com/cilium/tetragon/

Tetragon

- Tetragon is a flexible **Kubernetes-aware** security **observability** and runtime enforcement tool
- Traces the following events
  - Process execution events
  - System call activity
  - I/O activity including network & file access
- Tetragon is Kubernetes-aware - that is, it understands Kubernetes identities such as namespaces, pods and so-on

Demo

KubeCon | CloudNativeCon
North America 2023

Inspektor Gadget

https://github.com/inspektor-gadget/inspektor-gadget

Inspektor Gadget

- Tool designed for the creation, deployment, and execution of eBPF programs (gadgets) across Kubernetes and Linux environments
- A docker-like runtime for eBPF programs
- Gadgets (set of eBPF programs) can (among other things) collect metrics
- It automatically maps low-level kernel primitives to high-level Kubernetes resources
- Inspektor Gadget supports metrics in two ways
  - In user-space
  - In kernel

In userspace Metrics

- Metrics handled in user space
  - Reuses existing (built-in) gadgets
    - User doesn't have to write eBPF code
  - Counting is done in user space
    - Less performant
  - User configures how to count and aggregate events

In userspace Metrics

```
metrics_name: metrics_name
metrics:
- name: metric_name
  type: counter or gauge or histogram
  category: trace # category of the gadget to collect the metric.
  gadget: exec # gadget used to collect the metric. exec, open, etc.
  selector:
    # defines which events to take into consideration when updating the metrics.
    # See more information below.
  labels:
    # defines the granularity of the labels to capture. See below.
```

Metrics in Inspektor Gadget

- Selector
  - Filter out events

```
selector:
- k8s.namespace: default
```

- Labels
  - Granularity (dimensions) of the collected metrics

```
labels:
- k8s.pod
- k8s.container
```

Demo

In kernel Metrics

- eBPF code can be developed by the user
  - User defines the available labels from eBPF
  - Like ebpf_exporter
  - More performant than counting on userspace
- We plan to have some gadgets exporting common metrics

Demo

Take Aways

- Metrics are used to understand the health and performance of a system
- eBPF offers a powerful mechanism to collect data from the kernel
- Different projects offer support for collecting metrics with eBPF
- Different levels of abstraction
    - Write eBPF code
    - Write a yaml manifest
- Provide different labels
    - Operating system
    - Kubernetes / Containers

Reference

- https://prometheus.io/docs/concepts/metric_types/
- https://grafana.com/blog/2022/02/15/what-are-cardinality-spikes-and-why-do-they-matter/
- https://newrelic.com/blog/best-practices/opentelemetry-metrics
- https://opentelemetry.io/docs/concepts/signals/metrics/
- https://github.com/cloudflare/ebpf_exporter/
- https://www.inspektor-gadget.io/docs/v0.21.0/gadgets/prometheus/
- https://github.com/open-telemetry/opentelemetry-ebpf
- https://tetragon.io/docs/concepts/metrics/
- https://github.com/mauriciovasquezbernal/talks/tree/master/2023-kubecon-na

Please scan the QR Code above
to leave feedback on this session