Tutorial: Using BPF in Cloud Native environments

KubeCon EU 2020 | 2020.08.17

https://tinyurl.com/kubecon-bpf-workshop

Hi, I'm Alban

Alban Crequy
Co-Founder &
Director of Kinvolk Labs,
Kinvolk

Github: **alban**
Twitter: **@albcr**
Email: **alban@kinvolk.io**

Hi, I'm Marga

Marga Manterola
Staff Software Engineer,
Kinvolk

Github: **marga-kinvolk**
Twitter: **@marga_manterola**
Email: **marga@kinvolk.io**

Kinvolk

The Kubernetes Linux Experts

Building the 100% Open Enterprise Cloud Native Stack

Linux & Kubernetes • Engineering & Security Consulting • Community

Get Ready for this hands-on tutorial!

- Clone the GitHub repo:

```
git clone https://github.com/kinvolk/cloud-native-bpf-workshop.git
```

- Install all required dependencies:
  - Minikube (patched to include kernel 5.4 and headers)
  - Latest version of Inspektor-Gadget
  - Kubectl-Trace (patched to process compressed headers)

Hands-on Task:

Install Minikube

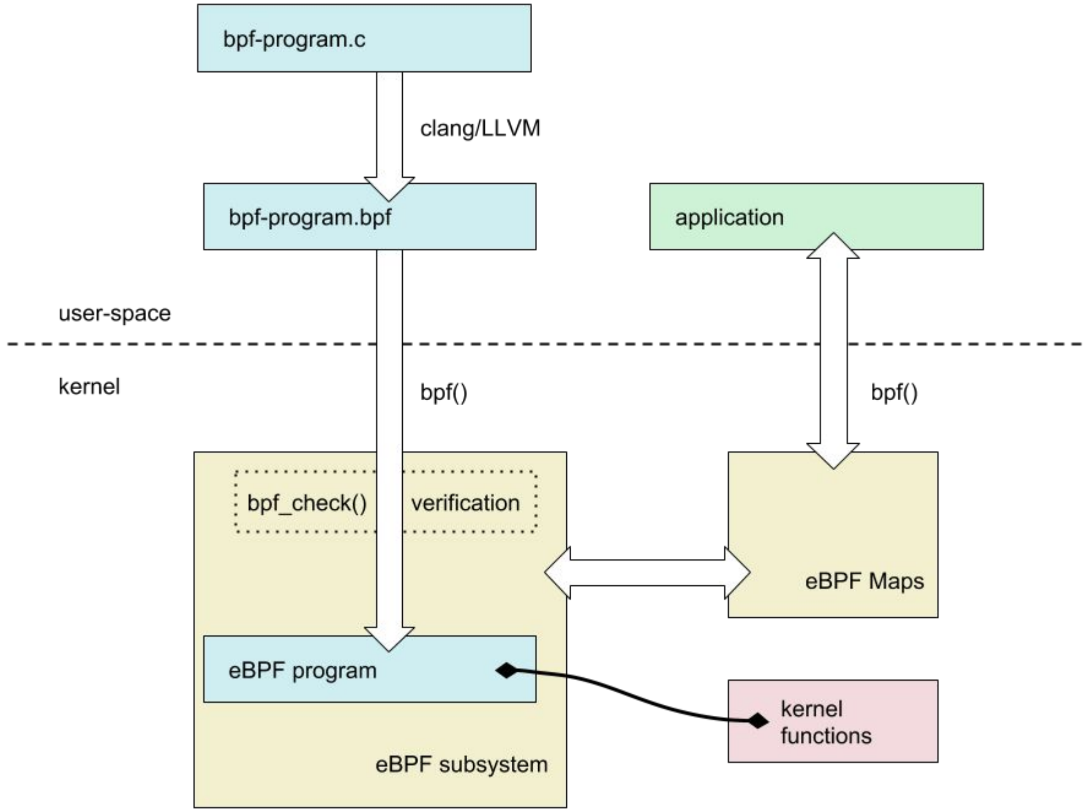
Problem statement

- Debugging distributed applications is hard
- BPF tracing tools can help us see what’s going on
- Using them inside Kubernetes is not trivial
- Inspektor Gadget and kubectl-trace plug this gap

Intro to Berkeley Packet Filter

- BPF: bytecode executed in the Linux kernel
- Initially for tcpdump (1992)
- Extended BPF (2013)

(e)BPF in a nutshell



Tracing tools for Kubernetes

Linux tracing tool	Kubernetes tracing tool
bpftrace github.com/iovisor/bpftrace	kubectl trace github.com/iovisor/kubectl-trace
BPF Compiler Collection (BCC) github.com/iovisor/bcc	Inspektor Gadget github.com/kinvolk/inspektor-gadget
traceloop github.com/kinvolk/traceloop	
Others github.com/weaveworks/tcptracer-bpf github.com/yadutaf/tracepkt	

Hands-on Task:

Install Inspektor Gadget

Network Policy Advisor

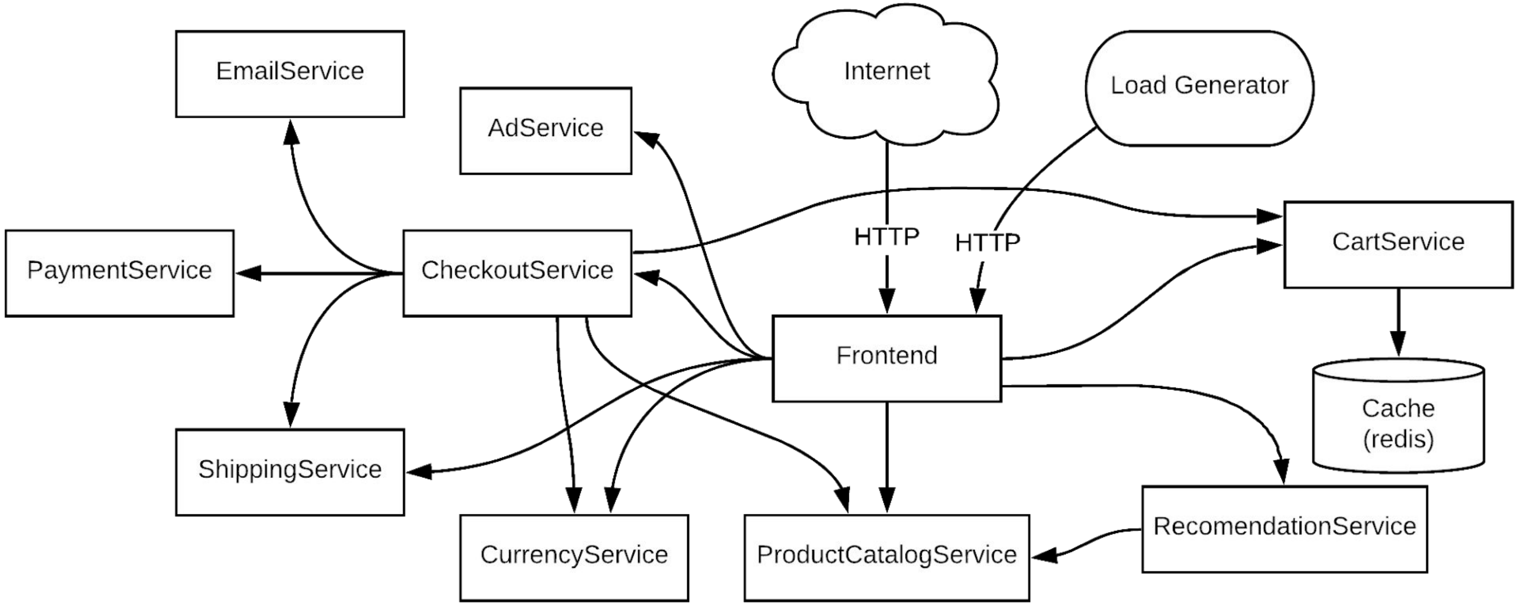
First gadget: Network Policy Advisor

Use case:

- A developer joins a project and has to implement network policies without a deep knowledge of the project architecture

“Pod security as an afterthought”

Example:
GoogleCloudPlatform/microservices-demo



Hands-on Task:

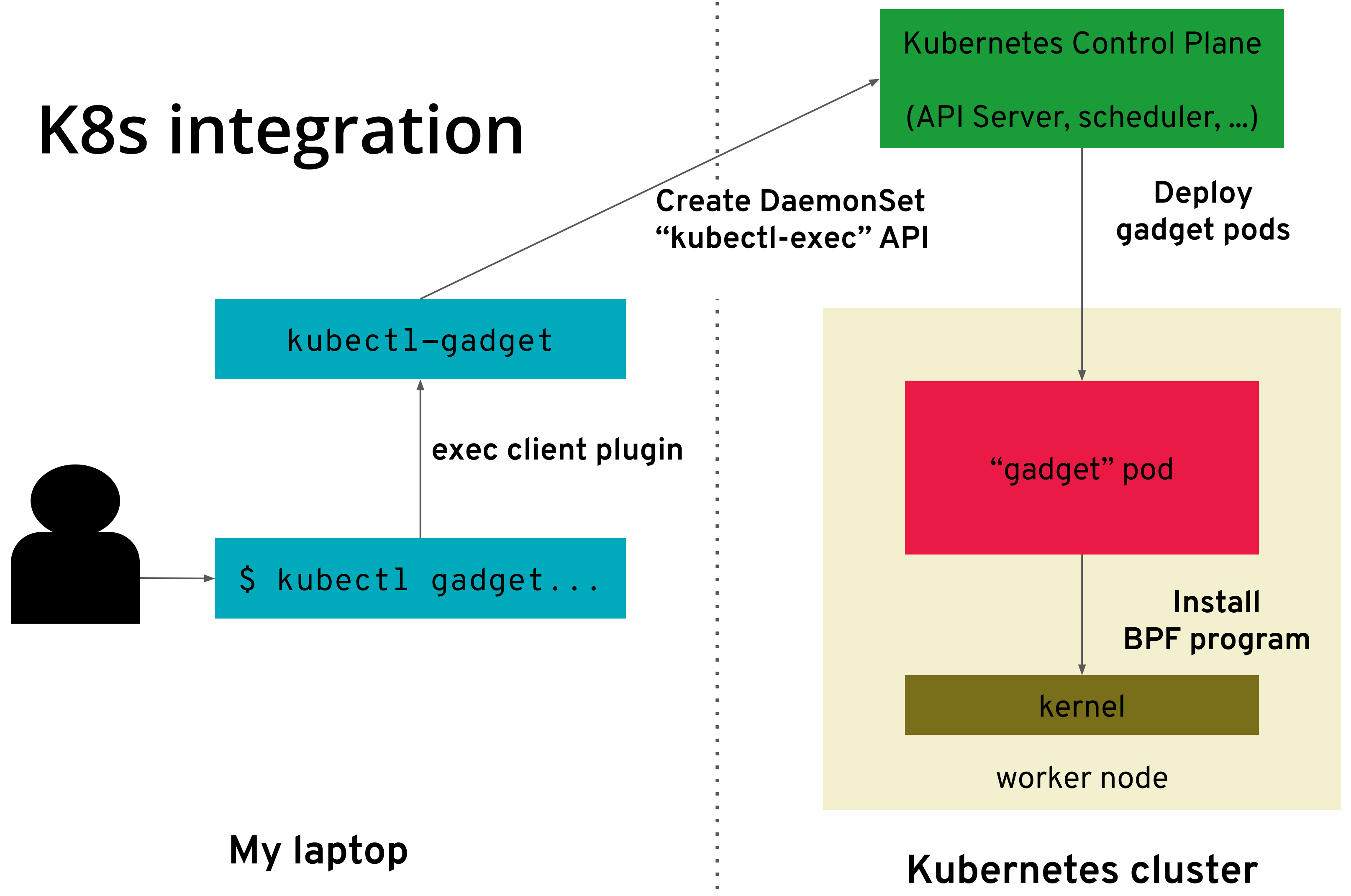
Start the Network Policy Advisor

Starting the Network Policy Advisor

```
$ kubectl gadget network-policy monitor \
    --namespaces demo --output ./networktrace.log

$ kubectl apply -f kubernetes-manifests.yaml -n demo
```

K8s integration



The generated log

```
{"type":"accept","remote_kind":"other","port":8080,"local_pod_namespace":"demo","local_pod_name":"frontend-5fcb8cdcdc-t9q8b","local_pod_owner":"frontend","local_pod_labels":{"app":"frontend","pod-template-hash":"5fcb8cdcdc"},"remote_other":"172.17.0.1","debug":"325113297266 cpu#0 accept 8271 server 172.17.0.4:8080 172.17.0.1:37432 4026532684\n"}

{"type":"connect","remote_kind":"svc","port":80,"local_pod_namespace":"demo","local_pod_name":"loadgenerator-79bff5bd57-c2x7k","local_pod_owner":"loadgenerator","local_pod_labels":{"app":"loadgenerator","pod-template-hash":"79bff5bd57"},"remote_svc_namespace":"demo","remote_svc_name":"frontend","remote_svc_label_selector":{"app":"frontend"},"debug":"411712601691 cpu#0 connect 18083 curl 172.17.0.9:46910 10.106.131.53:80 4026533000\n"}
```

Getting the report from the advisor

```
$ kubectl gadget network-policy report \
    --input ./networktrace.log > network-policy.yaml
```

Hands-on Task:

Review generated policies

Next gadget:

traceloop

Next gadget: traceloop

Tracing system calls in cgroups using BPF and overwritable ring buffers
https://github.com/kinvolk/traceloop

Debugging with “strace” on Kubernetes

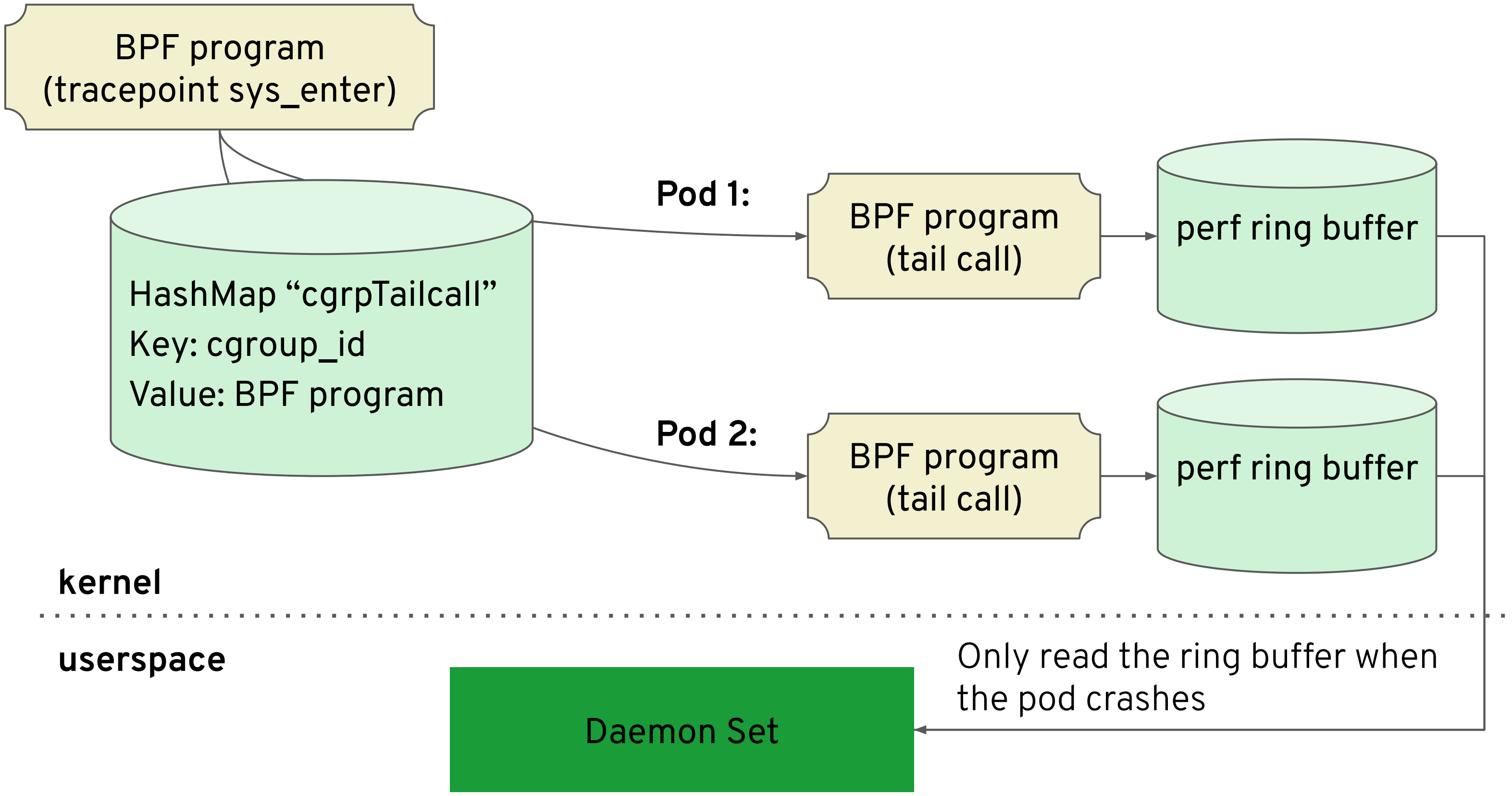
- Strace is slow
  - cannot be used for all pods on prod

- We need to know what’s going to crash
  - And start strace just before
  - Problem with unreproducible crashes
- Idea: “flight recorder”
  - Capture syscalls with BPF instead of strace
  - Send the events to a per-pod ring buffer
  - Only read the ring buffer when the pod crashed

Comparing strace and traceloop

	strace	traceloop
Capture method	ptrace	BPF on tracepoints
Granularity	process	cgroup
Speed	slow	fast
Reliability	Synchronous Cannot lose events	Asynchronous Can lose events Can fail to read buffers (EFAULT)

Debugging with “strace” on Kubernetes



Hands-on Task:

traceloop

Looking at stored traces

```
$ kubectl gadget traceloop list -A

$ kubectl gadget traceloop pod namespace podname idx

$ kubectl gadget traceloop show traceid
```

More gadgets

Wrappers for BCC tools

All available Gadgets

- ❏ **bindsnoop**: trace IPv4 and IPv6 bind() system calls
- ❏ **capabilities**: suggest Security Capabilities for securityContext
- ❏ **execsnoop**: trace new processes
- ❏ **network-policy**: generate network policies based on activity
- ❏ **opensnoop**: trace files opened by the pods
- ❏ **profile**: profile CPU usage by sampling stack traces
- ❏ **tcpconnect**: trace TCP connections
- ❏ **tcptop**: show the TCP traffic in a pod
- ❏ **tcptracer**: trace tcp connect, accept and close
- ❏ **traceloop**: get strace-like logs of a pod from the past

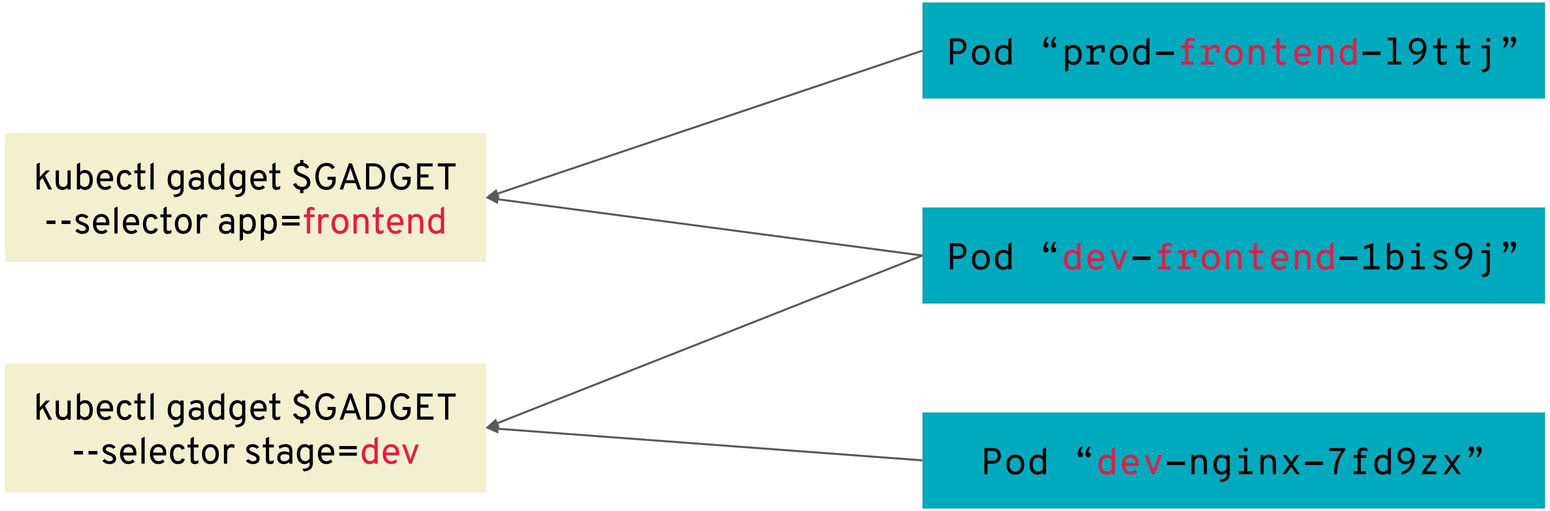
Tracing Cloud Native applications

- Granularity of tracing: pods
  - PIDs are not useful when we don't know which container it is
  - We don't want to trace all the system processes on a node
- Aggregation
  - Using Kubernetes labels, namespace, etc
- kubectl-like UX experience
  - Developers should not need to SSH
  - Developers should not need to deploy a pod + kubectl-exec for each tracing

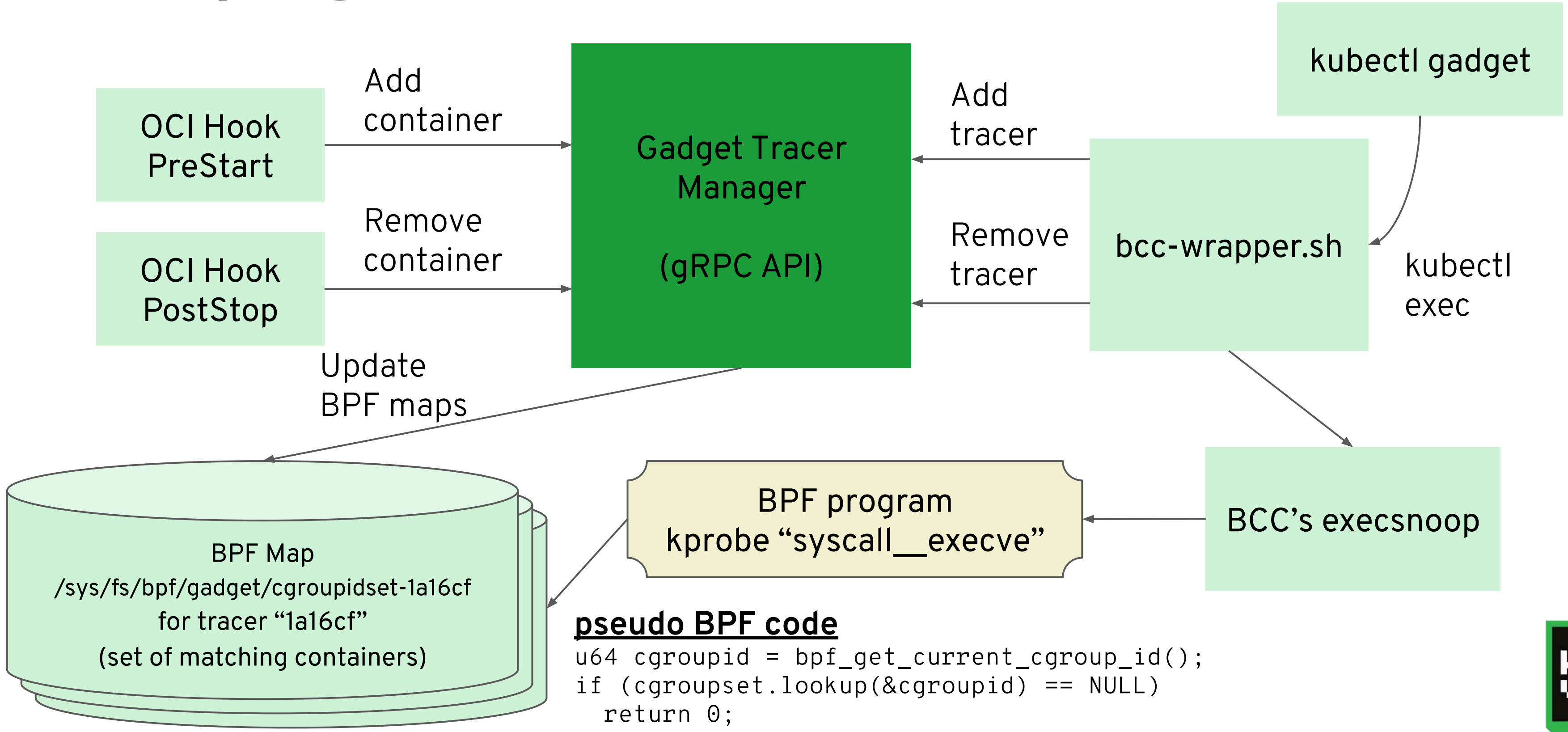
Selecting containers

```
$ kubectl gadget execsnoop \
    --selector k8s-app=myapp,tier=bar \
    --namespace default \
    --podname myapp1-l9ttj \
    --node ip-10-0-12-31 \
    --containername my-container
```

Pods & tracers come and go


kubectl gadget $GADGET --selector app=frontend
kubectl gadget $GADGET --selector stage=dev
Pod “prod-frontend-l9ttj”
Pod “dev-frontend-1bis9j”
Pod “dev-nginx-7fd9zx”

Keeping track of containers & tracers



pseudo BPF code

```
u64 cgroupid = bpf_get_current_cgroup_id();
if (cgroupset.lookup(&cgroupid) == NULL)
  return 0;
```

Hands-on Task:

Snooping operations

kubectl-trace

Bpftrace Syntax

$$\texttt{bpftrace -e '}\underbrace{\texttt{k:do\_nanosleep}}_{\textbf{Probe}}\ \underbrace{\texttt{/pid > 100/}}_{\textbf{Filter}}\ \underbrace{\texttt{\{ @[comm]++ \}}}_{\textbf{Action}}\texttt{'}$$

Bpftrace expressions

- Syscall count by program:

```
bpftrace -e 'tracepoint:raw_syscalls:sys_enter {
@[comm] = count(); }'
```

- Syscall rates per second:

```
bpftrace -e 'tracepoint:raw_syscalls:sys_enter { @ =
count(); } interval:s:1 { print(@); clear(@); }'
```

More examples at https://github.com/iovisor/bpftrace#one-liners

Using kubectl-trace

```
$ kubectl trace run node/nodename -e EXPRESSION

$ kubectl trace run pod/podname -f FILE.bt
```

Hands-on Task:

Using kubectl-trace

Questions?

Alban Crequy

Github: **alban**
Twitter: **@albcr**
Email: **alban@kinvolk.io**

Marga Manterola

Github: **marga-kinvolk**
Twitter: **@marga_manterola**
Email: **marga@kinvolk.io**

Kinvolk

Blog: **kinvolk.io/blog**
Github: **kinvolk**
Twitter: **kinvolkio**
Email: **hello@kinvolk.io**

Kubernetes Slack: #inspektor-gadget / #kubectl-trace
Material: https://tinyurl.com/kubecon-bpf-workshop

Thank you!

Alban Crequy

Github: alban
Twitter: @albcr
Email: alban@kinvolk.io

Marga Manterola

Github: marga-kinvolk
Twitter: @marga_manterola
Email: marga@kinvolk.io

Kinvolk

Blog: kinvolk.io/blog
Github: kinvolk
Twitter: kinvolkio
Email: hello@kinvolk.io

Kubernetes Slack: #inspektor-gadget / #kubectl-trace
Material: https://tinyurl.com/kubecon-bpf-workshop