# Overcoming the Challenges of Debugging Containers

Jose Blanquicet

# About me

- I’m Jose. Nice to meet you all!
- Colombian (Medellín) -> Italy (Turin)
- Senior SWE @ Microsoft (Kinvolk)

# Agenda

- Quick intro
- Debugging containers:
  - The challenges we identified
  - Our journey overcoming those difficulties:
    - First steps
    - Nowadays
    - Where are we going now?

# Quick Intro

# Quick Intro: eBPF

- eBPF programs are **event-driven** and are run when the kernel or an application passes a certain **hook point**.
- eBPF programs **use maps to share collected data** with other programs as well as from applications in user space. Notice it is **async** communication.
- Why eBPF?
  - Performance: **JIT** compiled and running **directly in the kernel**.
  - Security**:** Verified to **not crash** the kernel and can only be modified by **privileged users**.
  - Flexibility: **Modify or add functionality** and use cases to the kernel **without having to restart or patch it**.
- More about eBPF: https://ebpf.io

# Quick Intro: BCC tools

- BCC provides a **set of tools** covering simple and common use cases where eBPF can be used **to collect valuable information**.
- Perfect **starting point for beginners**.
- One of the first and **more active** project for BPF tooling.
- More about BCC: https://github.com/iovisor/bcc

# Challenges

# Challenges: Different scope

The standard Linux tools we were used to use (htop, tcpdump, ss, etc.) or eBPF-powered tools (e.g., BCC) are **not designed to work with containers.**

# Challenges: Demo

- Run some BCC tools
- Check sockets using ss

# Challenges: Summary

- Awareness of containers and k8s:
  - Enrichment
  - Filtering
  - Moving through Linux namespaces
- Deploying (if using a k8s)

# Our journey overcoming those difficulties

# Journey: First decision

Back in 2019, we decided to focus on **eBPF** and **Kubernetes**, and the very first version of Inspektor Gadget was a script running **BCC tools** into nodes.


# Journey: The very first steps

- We decided to **collaborate with the BCC project**.
- We start by associating the events with containers:
  - Add support for filtering by container (mount namespace) in BCC: docs

```
$ kubectl get pod --show-labels -o wide
NAME              READY   STATUS    RESTARTS   AGE     IP           NODE              LABELS
myapp1-pod-4kz56  1/1     Running   0          2m24s   10.2.232.6   ip-10-0-30-247   myapp=app-one,name=myapp1-pod,role=demo
myapp1-pod-qnj4d  1/1     Running   0          2m24s   10.2.249.6   ip-10-0-44-74    myapp=app-one,name=myapp1-pod,role=demo
myapp2-pod-s5kvv  1/1     Running   0          2m24s   10.2.249.7   ip-10-0-44-74    myapp=app-two,name=myapp2-pod,role=demo
myapp2-pod-tnthg  1/1     Running   0          2m24s   10.2.232.5   ip-10-0-30-247   myapp=app-two,name=myapp2-pod,role=demo
```

# Journey:
# Very first version

```
$ ./inspektor-gadget execsnoop --label role=demo --node ip-10-0-30-247
PCOMM            PID    PPID   RET ARGS
true             16510  11179    0 /bin/true
date             16511  11179    0 /usr/bin/date
cat              16512  11179    0 /usr/bin/cat /proc/version
sleep            16513  11179    0 /usr/bin/sleep 1
true             16514  11179    0 /bin/true
date             16515  11179    0 /usr/bin/date
cat              16516  11179    0 /usr/bin/cat /proc/version
sleep            16517  11179    0 /usr/bin/sleep 1
true             16520  11179    0 /bin/true
date             16521  11179    0 /usr/bin/date
cat              16522  11179    0 /usr/bin/cat /proc/version
sleep            16523  11179    0 /usr/bin/sleep 1
true             16524  10972    0 /bin/true
date             16525  10972    0 /usr/bin/date
echo             16526  10972    0 /bin/echo sleep-10
sleep            16527  10972    0 /bin/sleep 10
true             16528  11179    0 /bin/true
date             16529  11179    0 /usr/bin/date
cat              16530  11179    0 /usr/bin/cat /proc/version
sleep            16531  11179    0 /usr/bin/sleep 1
```

Taken from:
[v0.1.0-alpha.1]

# Journey: Kubernetes integration

- **Combine output** from nodes
- Create **kubectl plugin**: `kubectl gadget`
- **Enrich** with Kubernetes metadata
- **Filtering** by container, pod, namespace, node and/or labels

# Journey: Kubernetes integration

```
jose ~ $ kubectl gadget trace exec --selector role=demo
NODE                NAMESPACE           POD                 CONTAINER           PID       PPID      COMM      RET ARGS
multinode-m02       default             myapp1-pod-4zkrf    myapp1-pod          203803    191956    true      0   /bin/true
multinode-m02       default             myapp1-pod-4zkrf    myapp1-pod          203804    191956    date      0   /bin/date
multinode-m02       default             myapp1-pod-4zkrf    myapp1-pod          203805    191956    cat       0   /bin/cat /proc/version
multinode           default             myapp1-pod-j6vkk    myapp1-pod          203818    191630    true      0   /bin/true
multinode           default             myapp1-pod-j6vkk    myapp1-pod          203819    191630    date      0   /bin/date
multinode           default             myapp1-pod-j6vkk    myapp1-pod          203820    191630    cat       0   /bin/cat /proc/version
```

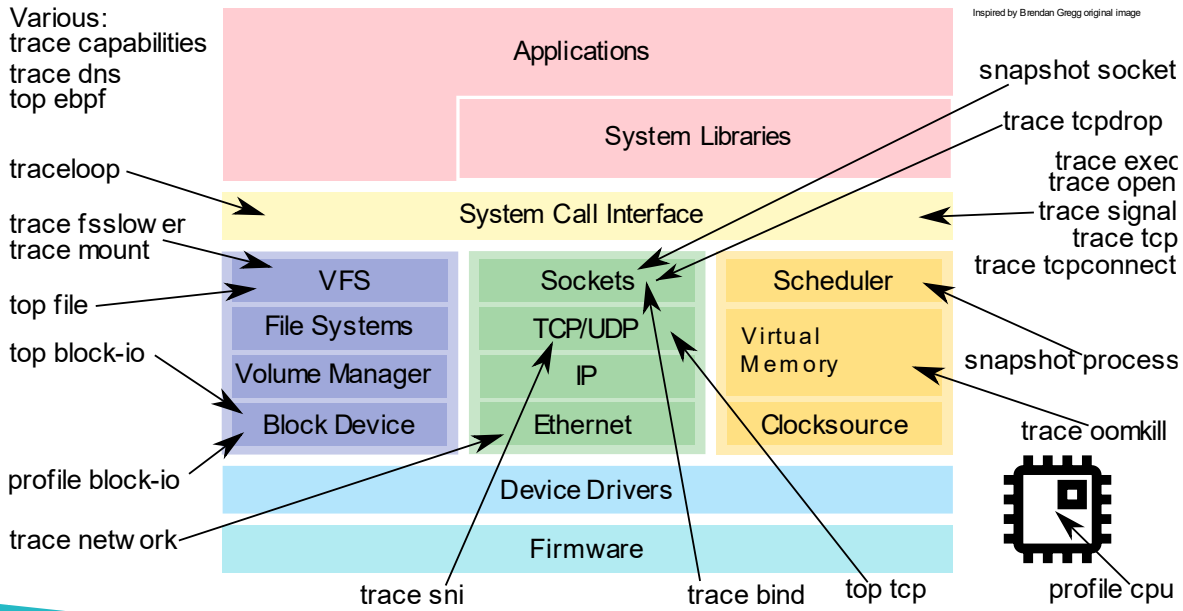
```
jose ~ $ kubectl gadget trace exec -p myapp1-pod-4zkrf
NODE                NAMESPACE           POD                 CONTAINER           PID       PPID      COMM      RET ARGS
multinode-m02       default             myapp1-pod-4zkrf    myapp1-pod          203398    191956    true      0   /bin/true
multinode-m02       default             myapp1-pod-4zkrf    myapp1-pod          203399    191956    date      0   /bin/date
multinode-m02       default             myapp1-pod-4zkrf    myapp1-pod          203400    191956    cat       0   /bin/cat /proc/version
multinode-m02       default             myapp1-pod-4zkrf    myapp1-pod          203418    191956    true      0   /bin/true
```

```
jose ~ $ kubectl gadget trace exec -A
NODE                NAMESPACE           POD                 CONTAINER           PID       PPID      COMM      RET ARGS
multinode           default             myapp1-pod-j6vkk    myapp1-pod          219418    191630    true      0   /bin/true
multinode           default             myapp1-pod-j6vkk    myapp1-pod          219419    191630    date      0   /bin/date
multinode           default             myapp1-pod-j6vkk    myapp1-pod          219420    191630    cat       0   /bin/cat /proc/version
multinode-m02       kube-system         kindnet-tfrsm       kindnet-cni         219421    182310    iptables  0   /usr/sbin/iptables -t n…
multinode-m02       kube-system         kindnet-tfrsm       kindnet-cni         219422    182310    iptables  0   /usr/sbin/iptables -t n…
multinode-m02       kube-system         kindnet-tfrsm       kindnet-cni         219424    182310    iptables  0   /usr/sbin/iptables -t n…
multinode-m02       kube-system         kindnet-tfrsm       kindnet-cni         219425    182310    iptables  0   /usr/sbin/iptables -t n…
multinode-m02       default             myapp1-pod-4zkrf    myapp1-pod          219426    191956    true      0   /bin/true
multinode-m02       default             myapp1-pod-4zkrf    myapp1-pod          219427    191956    date      0   /bin/date
```

Running v0.20.0

# Journey: New gadgets

Create **our own tools (or gadgets)** for Kubernetes/containers specific use cases:



# Journey: Nowadays

- CNCF Sandbox project
- Sync update for creation of containers
- More integration with Kubernetes:

```
jose ~ $ k get pods -n seccomp-demo
NAME           READY   STATUS    RESTARTS   AGE
curl-pod       1/1     Running   0          11s
hello-python   1/1     Running   0          10m
jose ~ $ k get svc -n seccomp-demo
NAME                   TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
hello-python-service   LoadBalancer   10.105.132.113   <pending>     6000:30670/TCP   11m
```

```
jose ~ $ kubectl gadget trace tcp -n seccomp-demo
NODE                NAMESPACE           POD                 CONTAINER           T PID      COMM        IP SRC                              DST
multinode-m02       seccomp-demo        curl-pod            curl-pod            C 300727   curl        4  p/seccomp-demo/curl-pod:48194    s/seccomp-demo/hello-python-ser
multinode-m02       seccomp-demo        hello-python        hello-python        A 246548   nginx       4  p/seccomp-demo/hello-python:80   p/seccomp-demo/curl-pod:48194
multinode-m02       seccomp-demo        curl-pod            curl-pod            X 300727   curl        4  p/seccomp-demo/curl-pod:48194    s/seccomp-demo/hello-python-ser
multinode-m02       seccomp-demo        hello-python        hello-python        X 246548   nginx       4  p/seccomp-demo/hello-python:80   p/seccomp-demo/curl-pod:48194
```

Running v0.20.0

At this point of the journey and **based on the users' feedback**, we started asking ourselves …

# Q: Why limit it to Kubernetes?

We created `ig` (formerly `local-gadget`):

- Fully decoupled from Kubernetes.
- Uses the container runtime to collect metadata:

  It supports containerd, docker, cri-o and podman.

# Q: Why limit it to containers?

We added `--host` flag to `ig`.

# Q: Why not to allow users to run their own gadgets?

More generally, why not to make the whole project generic enough to allow users to …

- Build
- Pack
- Ship
- Deploy
- And Run their own gadgets

… reusing the **framework** we already have?

# Containerized gadgets: Intro

- Design document:
  - **User Experience**: People already know how to work with **containers**, we are building on that.
  - **Gadget Packaging**: Gadgets should be packed as **OCI images.**
  - **Gadget Implementation**: Gadgets should follow a predefined convention to define their behaviour.
- Collaboration with bumblebee folks.

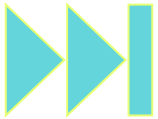
# Containerized gadgets: How to build gadgets?

```
ig image build --prog foo.bpf.c --definition foo.yaml mygadgetimage:latest

ig image tag mygadgetimage:latest ghcr.io/foo/mygadgetimage:v1

ig image list

ig image pull/push
```

# Containerized gadgets: How to run gadgets?

```
ig run mygadgetimage:latest [--detach]

ig list
```

IMPORTANT: Reusing all the functionalities I mentioned before.

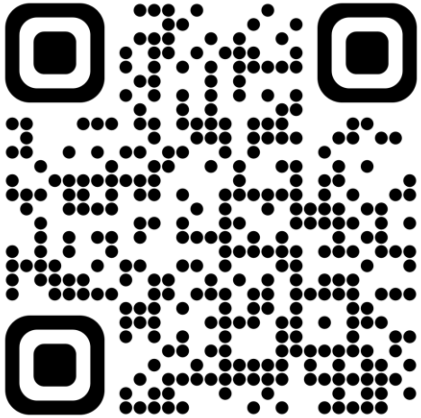


# Where are we going?

# We are focused on …

- Making Inspektor Gadget **a uniquely complete tool** for eBPF system inspection:
  - Support across Linux host processes, systemd units, containers, and Kubernetes
- **Docker is to containers what Inspektor Gadget is to eBPF programs:**
  - Demo of current development status on All Systems Go conference tomorrow.

# Thanks!



Jose Blanquicet

GitHub: blanquicet
X: @jose_blanquicet
LinkedIn: joseblanquicet

# Questions?

Wanna get involved
in the project?