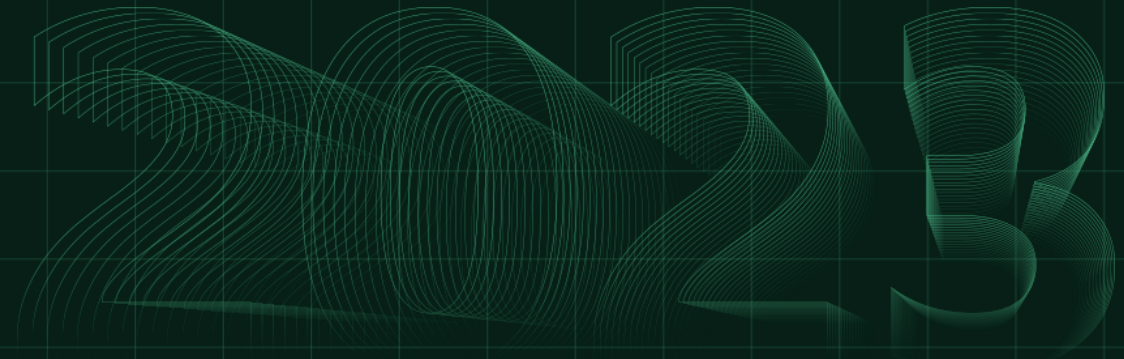


# eBPF 在云原生中的应用

倪朋飞
Microsoft Azure
2023.4

# 目录

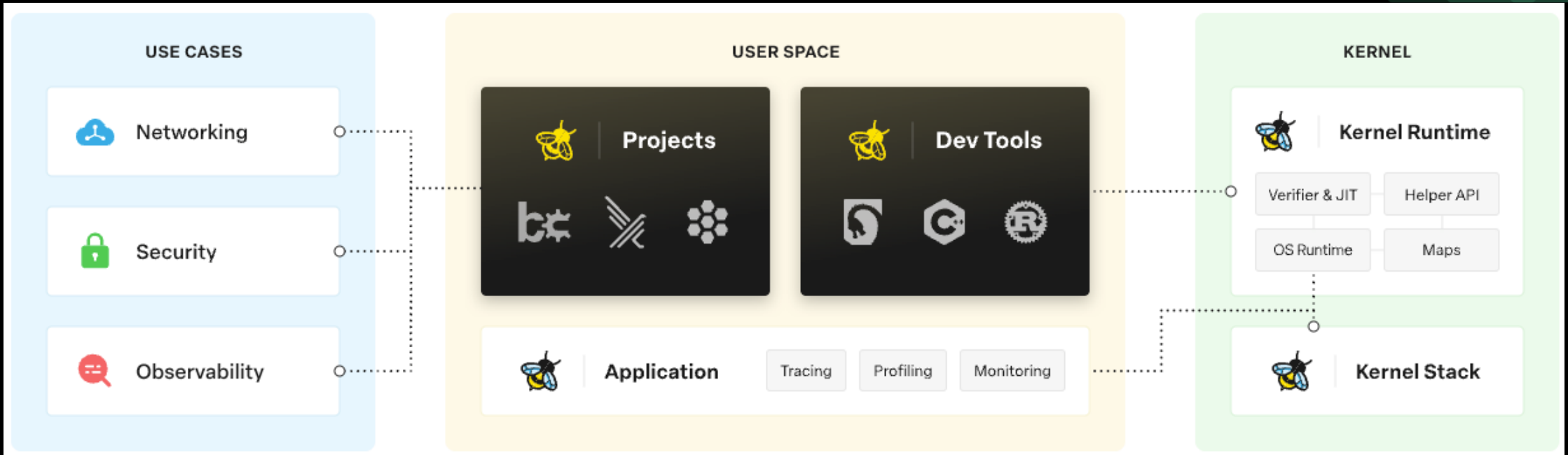
CATALOGUE



# 01

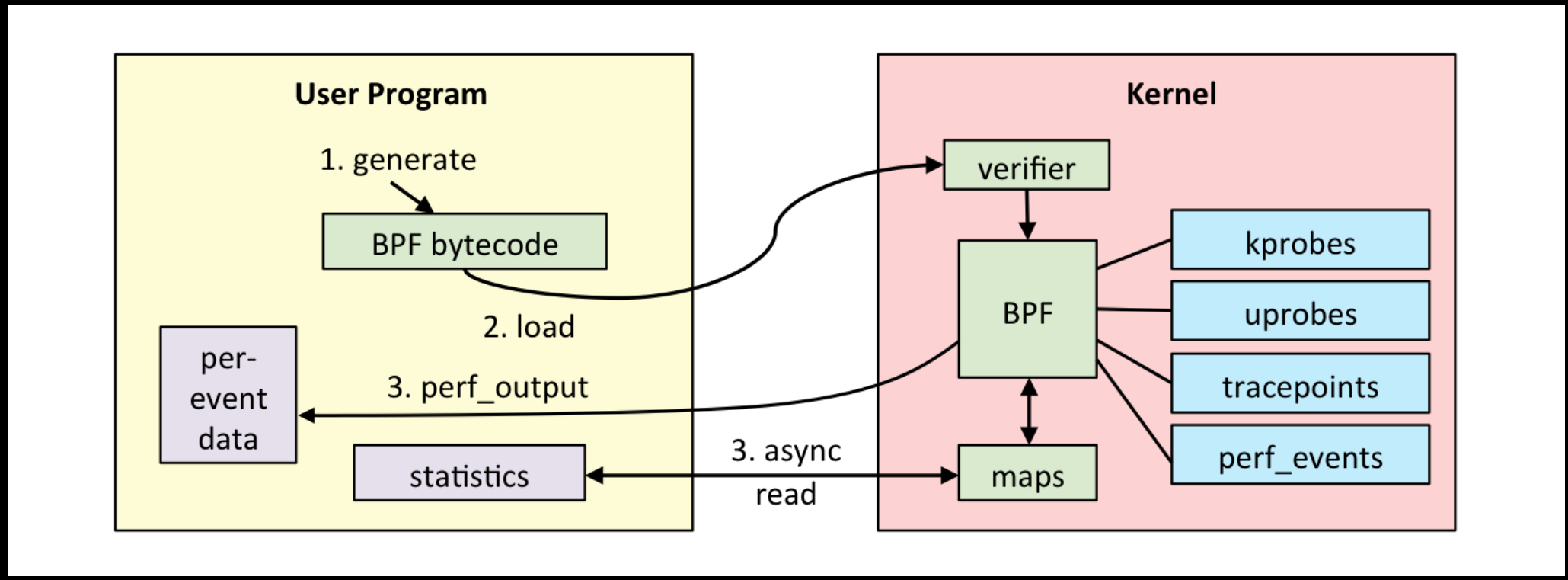
# eBPF 概述

# 什么是 eBPF




- 扩展的伯克利数据包过滤器 (extended Berkeley Packet Filter, eBPF)
- 在 eBPF 之前，内核模块是注入内核的最主要机制，缺少安全控制
- eBPF 借助即时编译器（JIT），在内核中运行了一个虚拟机，只有验证安全的 eBPF 指令才会被内核执行
- eBPF 指令运行在内核中，无需向用户态复制数据，这就大大提高了事件处理的效率

# eBPF 是如何运行的


User Program
1. generate
BPF bytecode
2. load
per-
event
data
3. perf_output
statistics
3. async
read
Kernel
verifier
BPF
maps
kprobes
uprobes
tracepoints
perf_events

# Hello, eBPF

hello.c

```c
int hello_world(void *ctx)
{
    bpf_trace_printk("Hello, World!");
    return 0;
}
```

hello.py

```python
# 1) import bcc library
from bcc import BPF

# 2) load BPF program
b = BPF(src_file="hello.c")
# 3) attach kprobe
b.attach_kprobe(event="do_sys_openat2", fn_name="hello_world")
# 4) read and print /sys/kernel/debug/tracing/trace_pipe
b.trace_print()
```

# eBPF 发展历程

- 1992 The BSD Packet Filter: A New Architecture for User-level Packet Capture
- 1997 Linux 2.1.75 首次引入了 BPF 技术，将高性能的 BSD 包过滤机制带入 Linux
- 2011 Linux 3.0 中增加的 BPF 即时编译器，进一步优化了 BPF 指令运行的效率
- 2014 Alexei Starovoitov 为 BPF 带来了第一次革命性的更新，将 BPF 扩展为一个通用的虚拟机， 即 eBPF
- 2015 BCC 大大简化了 eBPF 程序的开发和运行，Linux 4.1 开始支持 kprobe 和 cls_bpf (用于tc)
- 2016 Linux 4.7-4.10 带来了跟踪点、 perf 事件、 XDP 以及 cgroups 的支持，Cilium 项目发布
- 2017 BPF 成为内核独立子模块，并支持 kTLS/bpftool/libbpf 。Netflix/Facebook/Cloudflare 将其用于跟踪、DDoS 防御、4层负载均衡等

# eBPF 发展历程

- 2018 BPF 新增轻量级调试信息格式 BTF 以及新的 AF_XDP 类型。Cilium 发布 1.0 版本，bpftrace 和 bpffilter 项目也正式发布
- 2019 BPF 新增尾调用和热更新，GCC 支持 BPF 编译。Cilium 1.6 发布基于 BPF 的服务发现代理（替换基于 iptables 的 kube-proxy）
- 2020 Google 和 Facebook 为 BPF 新增 LSM 和 TCP 拥塞控制，云厂商通过 SRIOV 支持 XDP，Windows 监控工具 Sysmon 增加 Linux 支持
- 2021 Windows eBPF 发布，eBPF 开始支持内核函数调用，Cilium 发布基于 eBPF 的 Service Mesh
- 2021 微软、Facebook、Google、Isovalent、Netflix 等一起成立 eBPF 基金会，Cilium 加入 CNCF
- 2022 eBPF 新增布隆过滤映射、CO-RE、bpf_loop、动态指针、BPF 内存分配器等，一些列开源项目增加 eBPF 支持
- 2023 eBPF 火热发展中 …

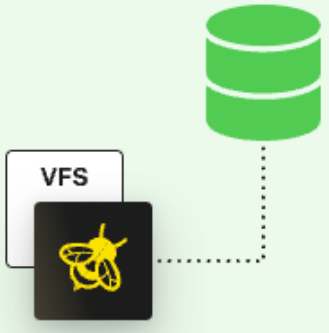
# 02

# eBPF 应用场景

# eBPF 有哪些应用场景

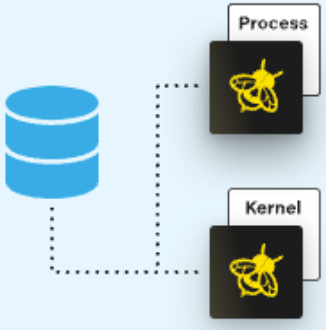

## 网络

在内核空间绕过协议栈直接处理网络包，加速网络过滤和处理的性能。



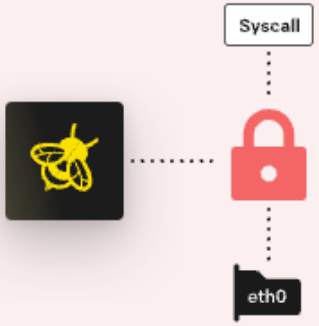

## 可观察性

从内核空间直接收集各类性能指标，而无需应用程序作任何修改。




## 追踪与剖析

将 eBPF 程序附加到跟踪点以及内核和用户应用程序探测点，追踪和剖析系统性能问题。




## 安全

动态分析内核和应用的行为，监测各类安全事件，并执行相应的安全策略。

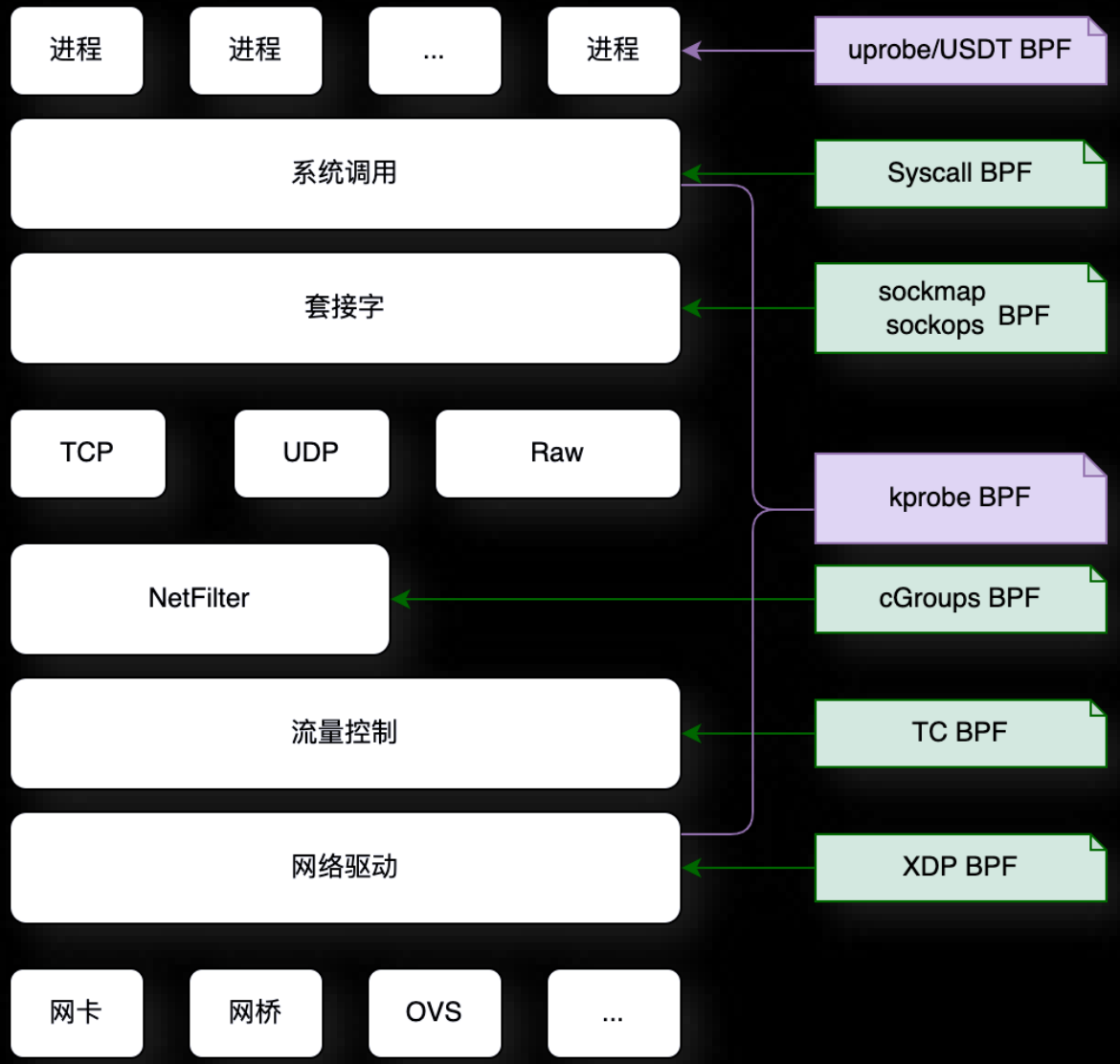
# eBPF 全景图

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 观测跟踪 | skydive | hubble | weave scope | kubectl-trace | Inspektor Gadget | pixie | eCapture | pwru | DeepFlow |
| 网络安全 | cilium | calico | katran | Tetragon | Tracee | polycube | falco | KubeArmor | |
| 开发库 | gobpf | ebpf | libbpf | libbpf-rs | Redbpf | bcc | | | |
| 基础工具 | bcc | bpftrace | llvm | GCC | bpftool | | | | |
| 运行时 | Linux | Windows | hBPF | | | | | | |

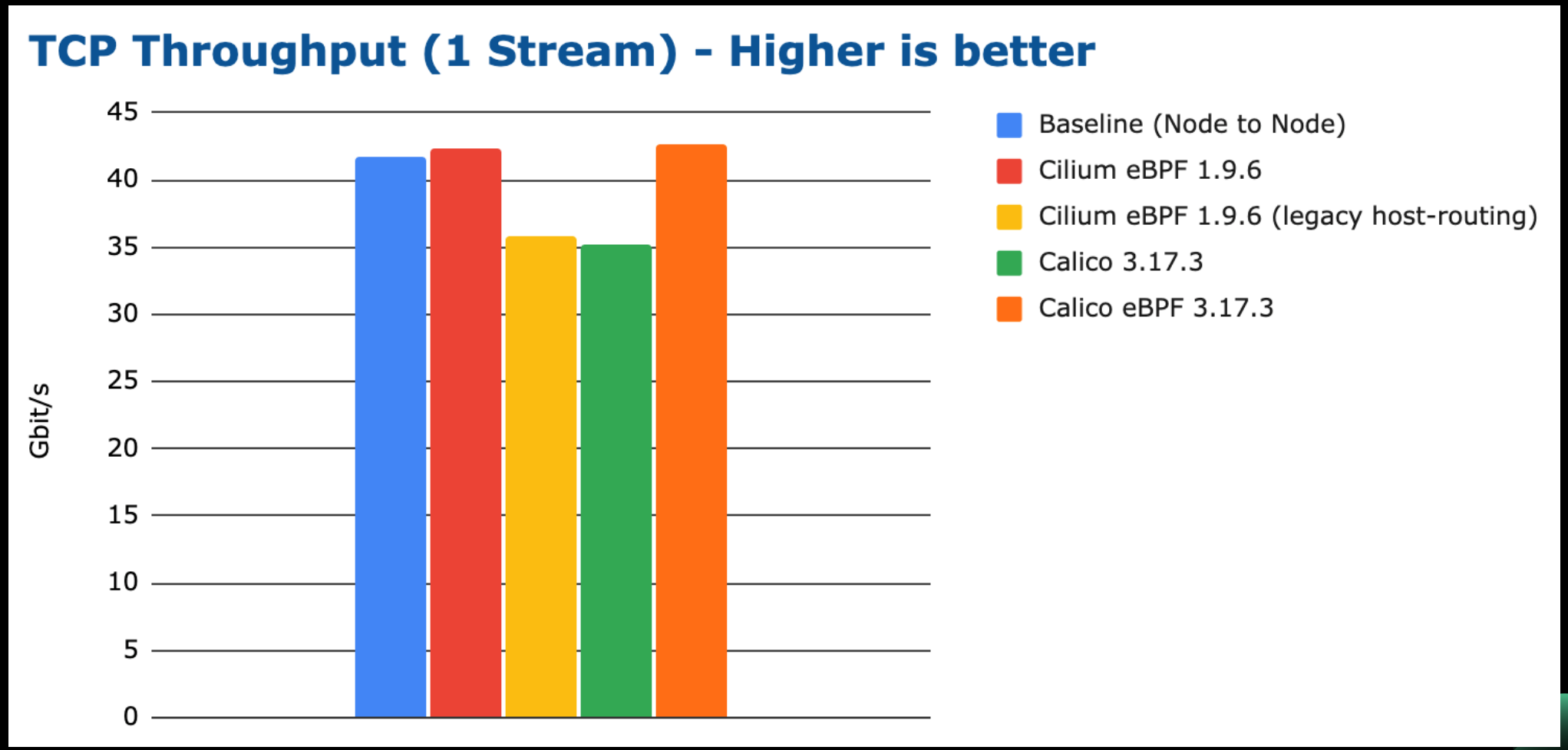
# 03

# eBPF 在云原生中的应用

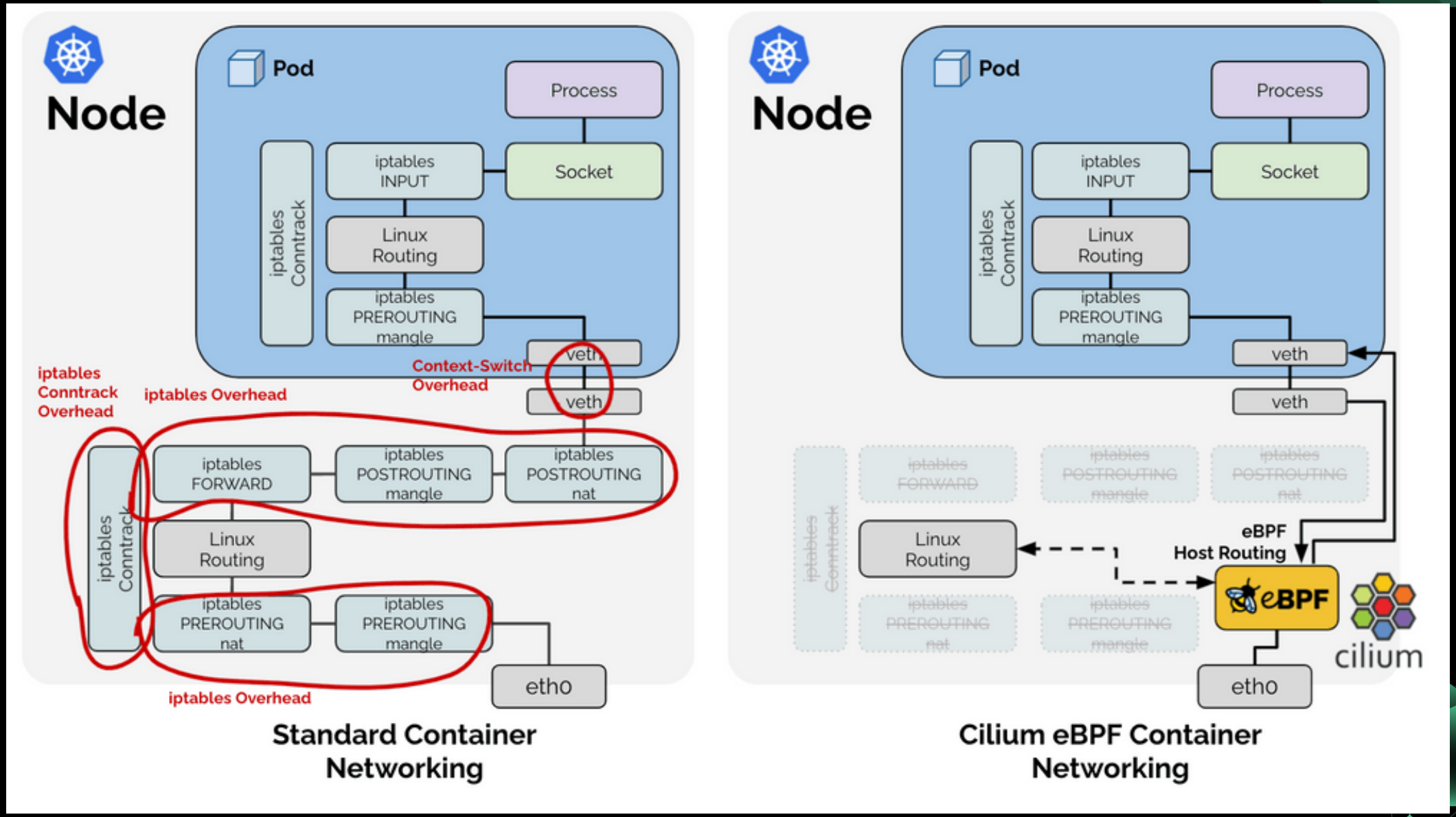
# 网络


进程
进程
...
进程
系统调用
套接字
TCP
UDP
Raw
NetFilter
流量控制
网络驱动
网卡
网桥
OVS
...
uprobe/USDT BPF
Syscall BPF
sockmap
sockops
BPF
kprobe BPF
cGroups BPF
TC BPF
XDP BPF

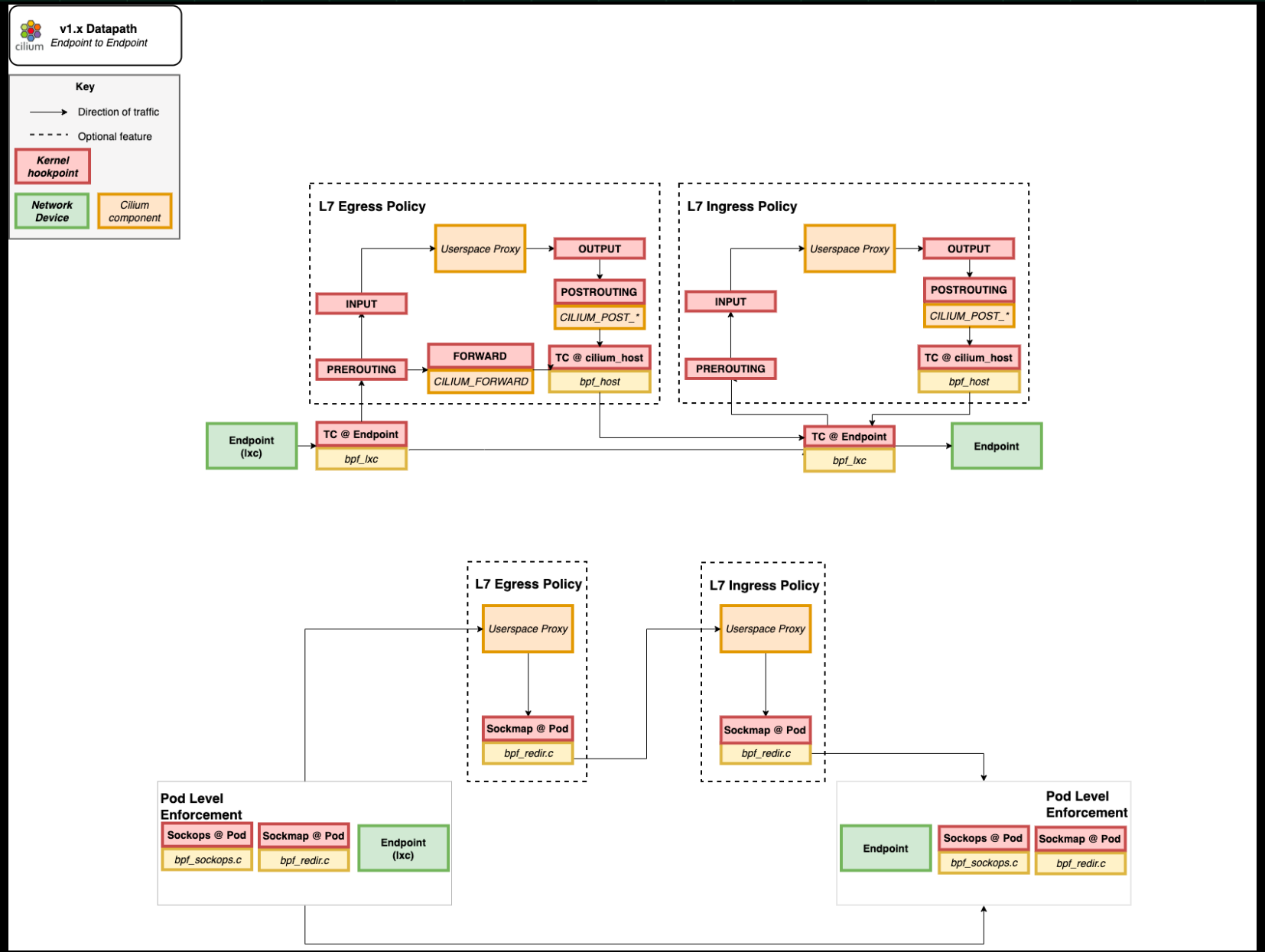
# 网络性能优化


TCP Throughput (1 Stream) - Higher is better
Gbit/s
45
40
35
30
25
20
15
10
5
0
Baseline (Node to Node)
Cilium eBPF 1.9.6
Cilium eBPF 1.9.6 (legacy host-routing)
Calico 3.17.3
Calico eBPF 3.17.3

# Cilium eBPF


Node
Pod
Process
Socket
iptables INPUT
iptables Conntrack
Linux Routing
iptables PREROUTING mangle
veth
veth
Context-Switch Overhead
iptables Conntrack Overhead
iptables Overhead
iptables FORWARD
iptables POSTROUTING mangle
iptables POSTROUTING nat
Linux Routing
iptables PREROUTING nat
iptables PREROUTING mangle
eth0
iptables Overhead
Standard Container Networking
Node
Pod
Process
Socket
iptables INPUT
iptables Conntrack
Linux Routing
iptables PREROUTING mangle
veth
veth
eBPF Host Routing
eBPF
cilium
eth0
Cilium eBPF Container Networking

# Cilium eBPF


v1.x Datapath
Endpoint to Endpoint
Key
Direction of traffic
Optional feature
Kernel hookpoint
Network Device
Cilium component
L7 Egress Policy
Userspace Proxy
OUTPUT
POSTROUTING
CILIUM_POST_*
INPUT
PREROUTING
FORWARD
CILIUM_FORWARD
TC @ cilium_host
bpf_host
Endpoint (lxc)
TC @ Endpoint
bpf_lxc
L7 Ingress Policy
Userspace Proxy
OUTPUT
POSTROUTING
CILIUM_POST_*
INPUT
PREROUTING
TC @ cilium_host
bpf_host
TC @ Endpoint
bpf_lxc
Endpoint
L7 Egress Policy
Userspace Proxy
Sockmap @ Pod
bpf_redir.c
L7 Ingress Policy
Userspace Proxy
Sockmap @ Pod
bpf_redir.c
Pod Level Enforcement
Sockops @ Pod
bpf_sockops.c
Sockmap @ Pod
bpf_redir.c
Endpoint (lxc)
Pod Level Enforcement
Endpoint
Sockops @ Pod
bpf_sockops.c
Sockmap @ Pod
bpf_redir.c

# Cilium 整体架构

## Service Mesh

RESILIENCE　AUTHENTICATION　TRAFFIC MANAGEMENT

cilium Service Mesh

Ingress　Gateway API　spiffe

## Observability

METRICS　TRACING　SERVICE MAP　LOGS

cilium Hubble

SIEM　JSON　fluentd　Grafana　OpenTelemetry

## Networking

ENCRYPTION　LOAD-BALANCING　NETWORK POLICY　NETWORKING

cilium CNI

IPsec
Wireguard

Kubernetes
DSR • Maglev

DNS • L7
L3/L4

IPv4 • IPv6 • Overlay • BGP
Egress Gateway • Multi-Cluster

## Runtime Security

OBSERVABILITY
ENFORCEMENT

Tetragon

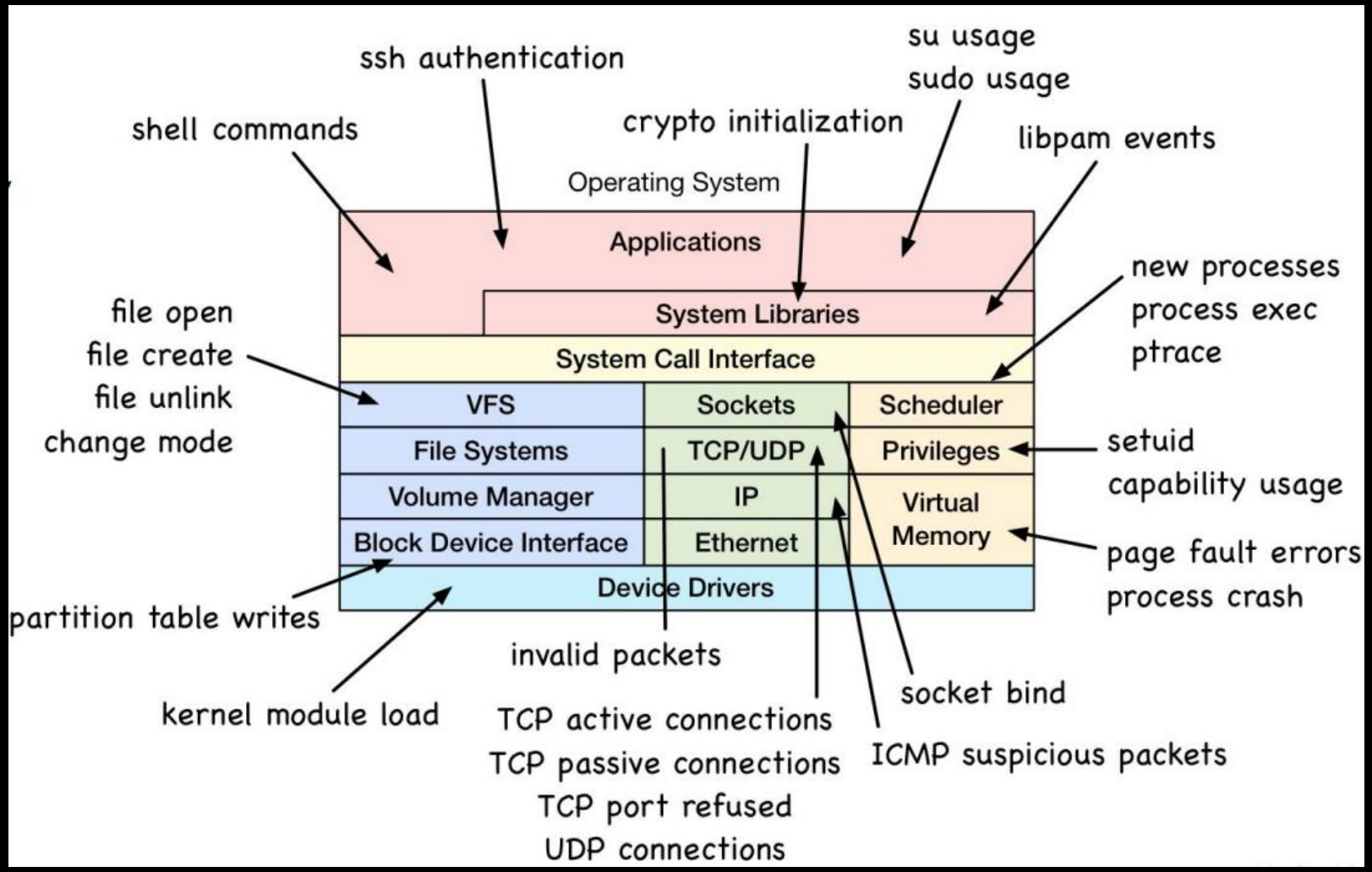
SIEM　JSON

fluentd　Grafana

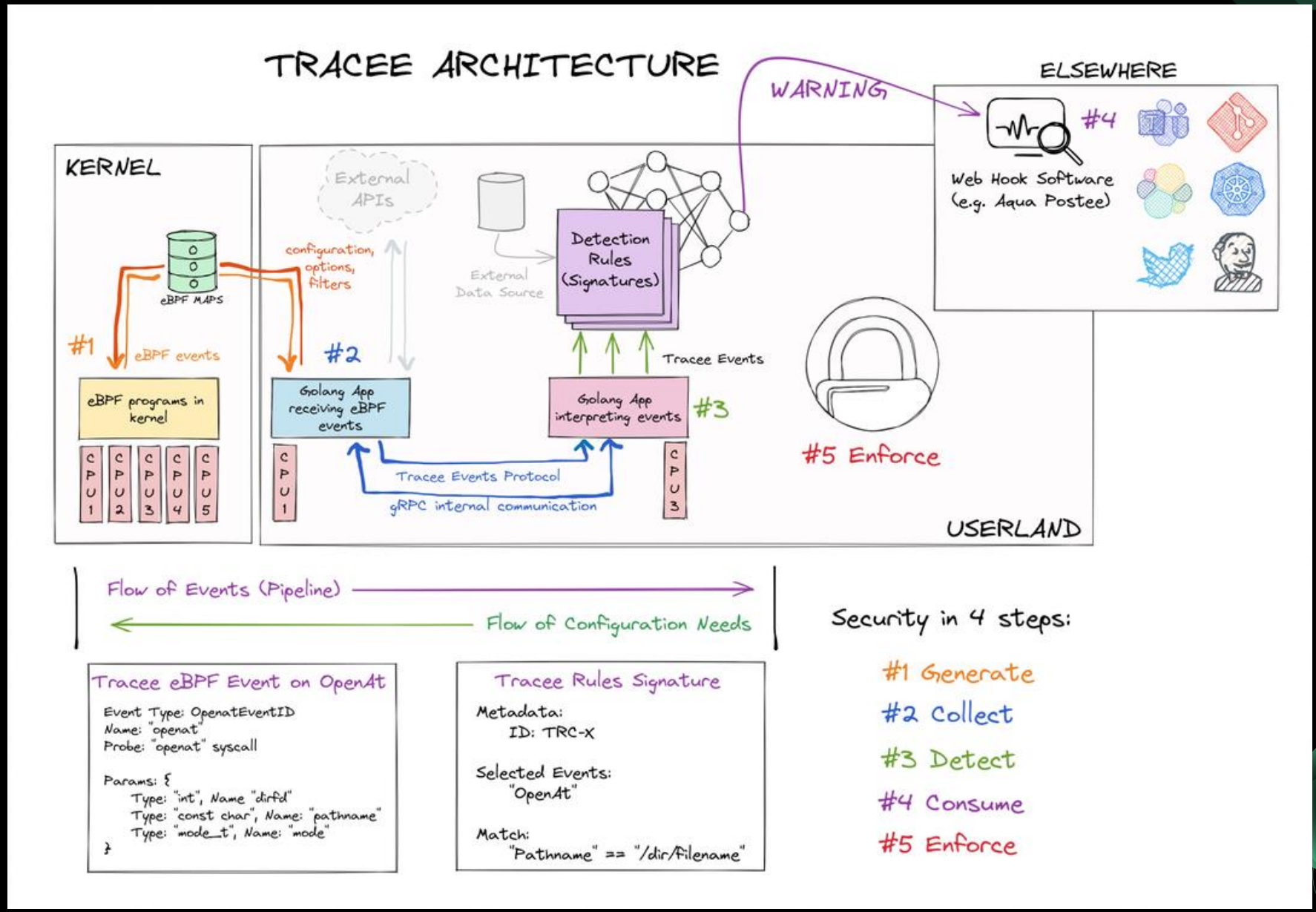
Kubernetes　Container　VM　Metal

aws　Google Cloud　Azure　Alibaba Cloud　Red Hat OpenShift　vmware

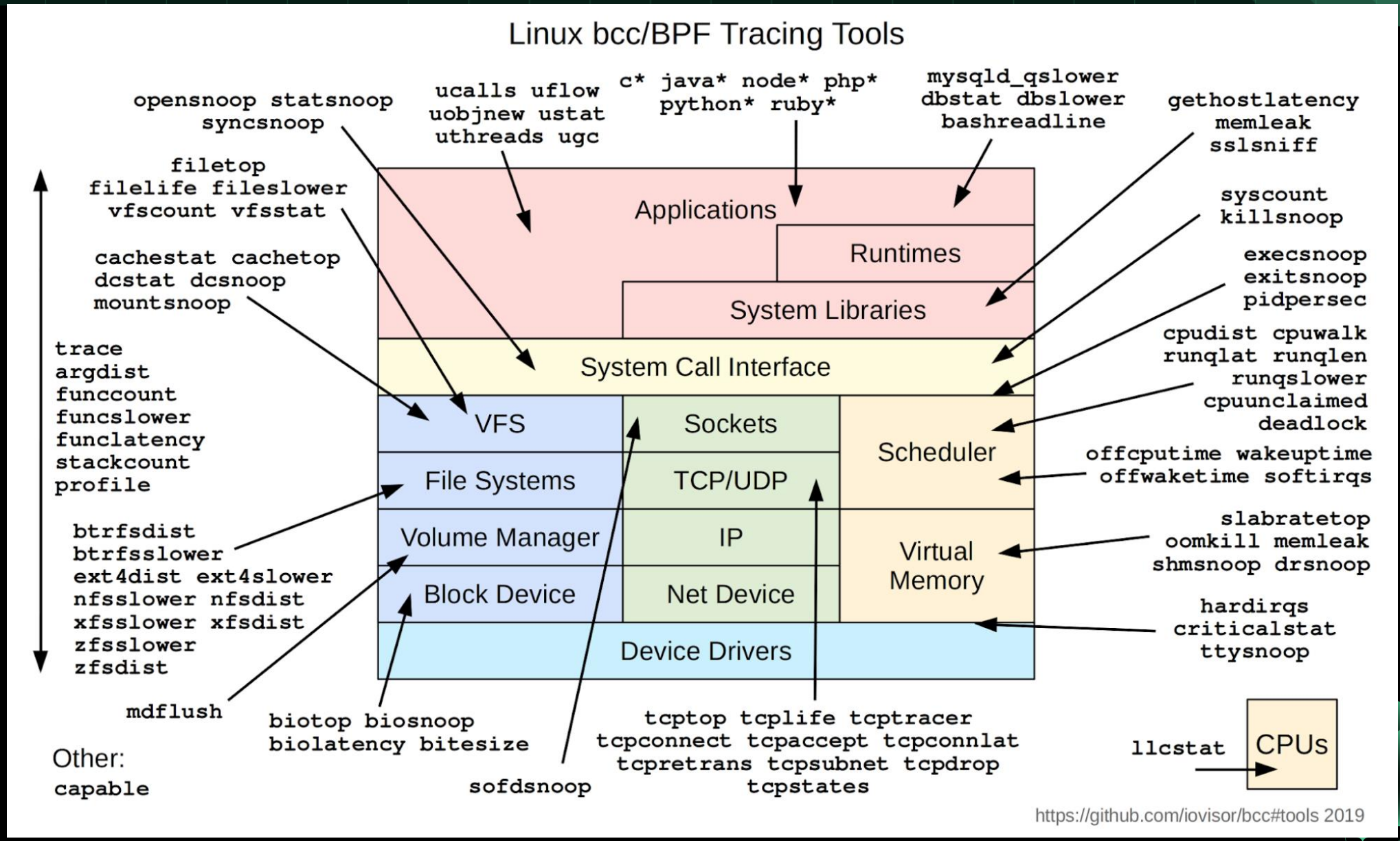
# 安全


ssh authentication
su usage
sudo usage
shell commands
crypto initialization
libpam events
Operating System
Applications
System Libraries
System Call Interface
new processes
process exec
ptrace
file open
file create
file unlink
change mode
VFS
Sockets
Scheduler
File Systems
TCP/UDP
Privileges
setuid
capability usage
Volume Manager
IP
Virtual Memory
Block Device Interface
Ethernet
page fault errors
process crash
Device Drivers
partition table writes
invalid packets
socket bind
kernel module load
TCP active connections
TCP passive connections
TCP port refused
UDP connections
ICMP suspicious packets

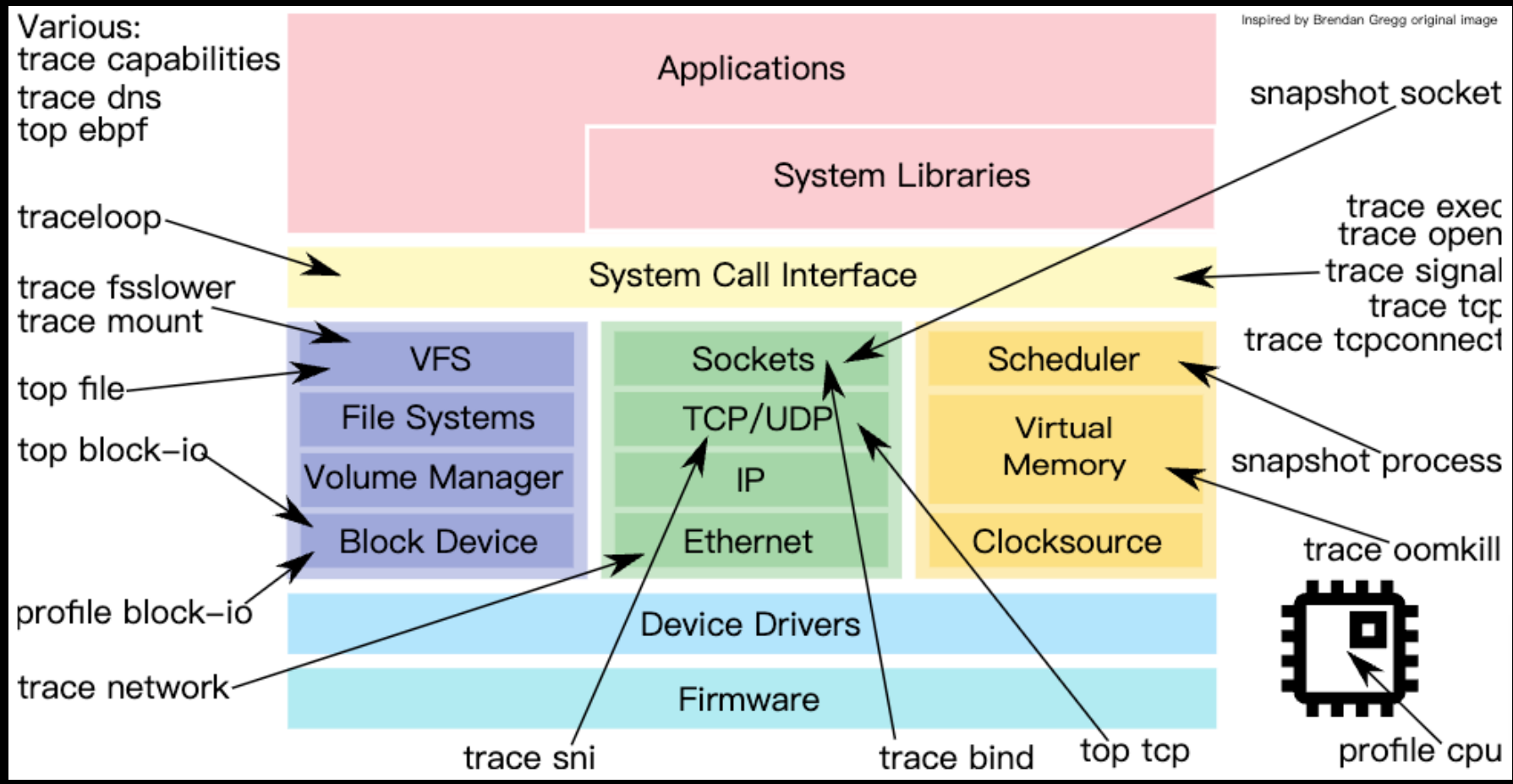
# Tracee


TRACEE ARCHITECTURE
KERNEL
USERLAND
ELSEWHERE
WARNING
#4
Web Hook Software
(e.g. Aqua Postee)
External APIs
configuration, options, filters
eBPF MAPS
External Data Source
Detection Rules (Signatures)
#1
eBPF events
#2
Tracee Events
eBPF programs in kernel
Golang App receiving eBPF events
Golang App interpreting events
#3
#5 Enforce
CPU 1
CPU 2
CPU 3
CPU 4
CPU 5
Tracee Events Protocol
gRPC internal communication
Flow of Events (Pipeline)
Flow of Configuration Needs
Security in 4 steps:
#1 Generate
#2 Collect
#3 Detect
#4 Consume
#5 Enforce
Tracee eBPF Event on OpenAt
Event Type: OpenatEventID
Name: "openat"
Probe: "openat" syscall
Params: {
Type: "int", Name "dirfd"
Type: "const char", Name: "pathname"
Type: "mode_t", Name: "mode"
}
Tracee Rules Signature
Metadata:
ID: TRC-X
Selected Events:
"OpenAt"
Match:
"Pathname" == "/dir/filename"

# 追踪与剖析


Linux bcc/BPF Tracing Tools
opensnoop statsnoop
syncsnoop
ucalls uflow
uobjnew ustat
uthreads ugc
c* java* node* php*
python* ruby*
mysqld_qslower
dbstat dbslower
bashreadline
gethostlatency
memleak
sslsniff
filetop
filelife fileslower
vfscount vfsstat
cachestat cachetop
dcstat dcsnoop
mountsnoop
syscount
killsnoop
execsnoop
exitsnoop
pidpersec
cpudist cpuwalk
runqlat runqlen
runqslower
cpuunclaimed
deadlock
offcputime wakeuptime
offwaketime softirqs
slabratetop
oomkill memleak
shmsnoop drsnoop
hardirqs
criticalstat
ttysnoop
trace
argdist
funccount
funcslower
funclatency
stackcount
profile
btrfsdist
btrfsslower
ext4dist ext4slower
nfsslower nfsdist
xfsslower xfsdist
zfsslower
zfsdist
mdflush
biotop biosnoop
biolatency bitesize
sofdsnoop
tcptop tcplife tcptracer
tcpconnect tcpaccept tcpconnlat
tcpretrans tcpsubnet tcpdrop
tcpstates
llcstat
Other:
capable
Applications
Runtimes
System Libraries
System Call Interface
VFS
Sockets
Scheduler
File Systems
TCP/UDP
Volume Manager
IP
Virtual Memory
Block Device
Net Device
Device Drivers
CPUs
https://github.com/iovisor/bcc#tools 2019

# Inspektor Gadget



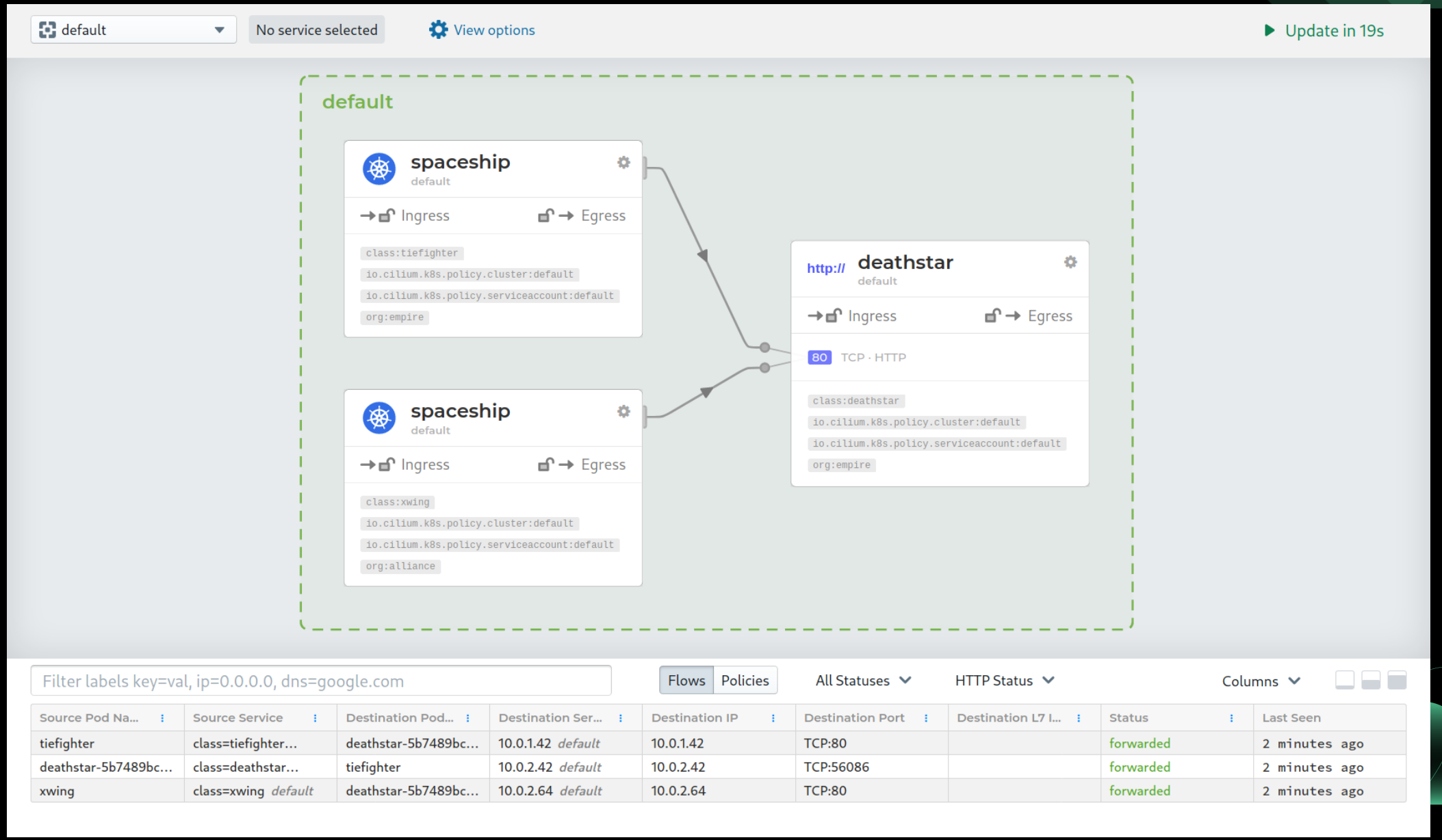
# pwru

```
ceuse#
```

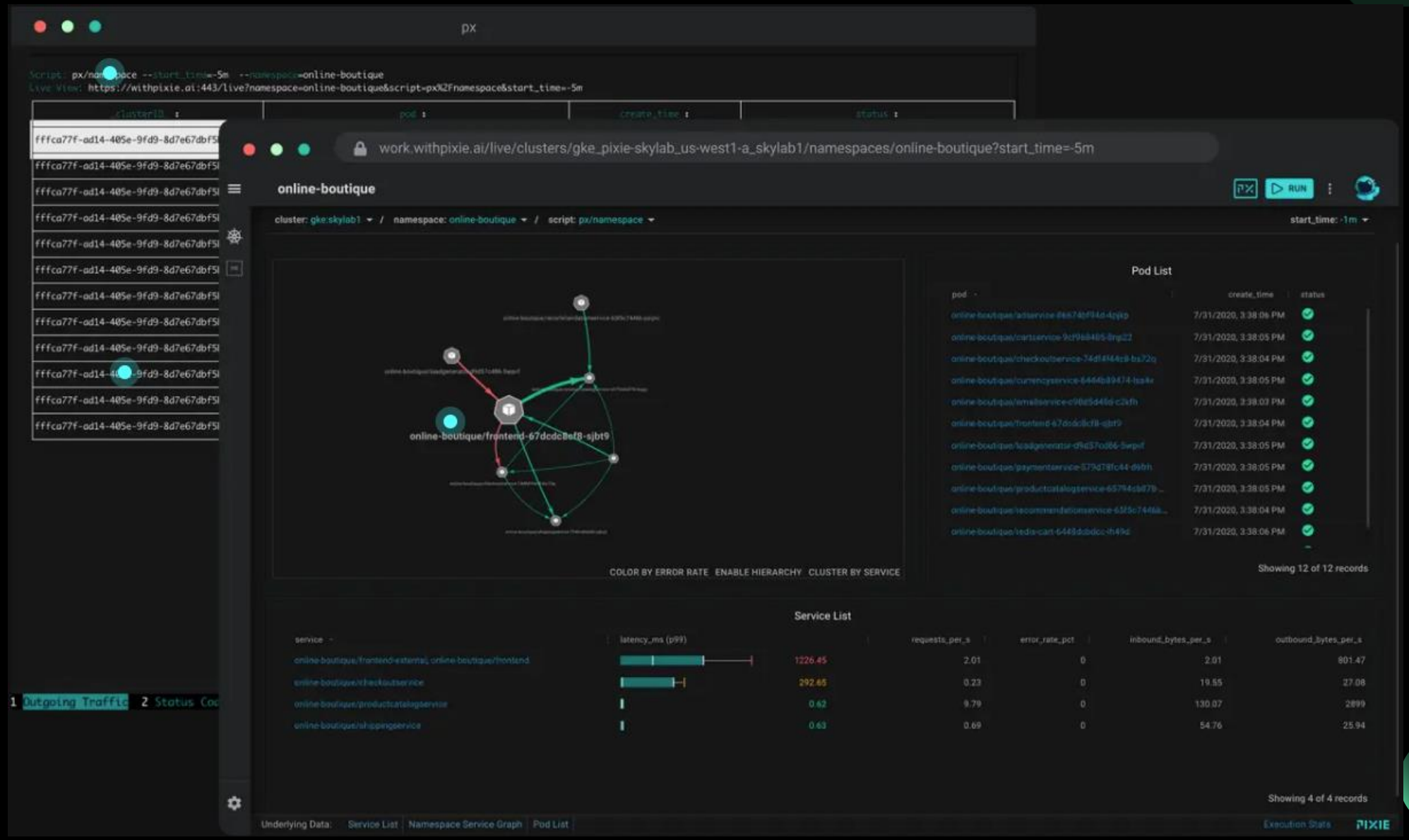
---

```
ceuse#
```

# 观测


default
No service selected
View options
Update in 19s
default
spaceship
default
Ingress
Egress
class:tiefighter
io.cilium.k8s.policy.cluster:default
io.cilium.k8s.policy.serviceaccount:default
org:empire
spaceship
default
Ingress
Egress
class:xwing
io.cilium.k8s.policy.cluster:default
io.cilium.k8s.policy.serviceaccount:default
org:alliance
http://
deathstar
default
Ingress
Egress
80
TCP · HTTP
class:deathstar
io.cilium.k8s.policy.cluster:default
io.cilium.k8s.policy.serviceaccount:default
org:empire
Filter labels key=val, ip=0.0.0.0, dns=google.com
Flows
Policies
All Statuses
HTTP Status
Columns
Source Pod Na...
Source Service
Destination Pod...
Destination Ser...
Destination IP
Destination Port
Destination L7 I...
Status
Last Seen
tiefighter
class=tiefighter...
deathstar-5b7489bc...
10.0.1.42 default
10.0.1.42
TCP:80
forwarded
2 minutes ago
deathstar-5b7489bc...
class=deathstar...
tiefighter
10.0.2.42 default
10.0.2.42
TCP:56086
forwarded
2 minutes ago
xwing
class=xwing default
deathstar-5b7489bc...
10.0.2.64 default
10.0.2.64
TCP:80
forwarded
2 minutes ago

# Pixie


px
work.withpixie.ai/live/clusters/gke_pixie-skylab_us-west1-a_skylab1/namespaces/online-boutique?start_time=-5m
online-boutique
RUN
cluster: gke:skylab1 / namespace: online-boutique / script: px/namespace
start_time: -1m
online-boutique/frontend-67dcdc8ef8-sjbt9
COLOR BY ERROR RATE
ENABLE HIERARCHY
CLUSTER BY SERVICE
Pod List
Showing 12 of 12 records
Service List
Showing 4 of 4 records
Underlying Data:
Service List
Namespace Service Graph
Pod List
Execution Stats
PIXIE

# 04

# 最佳实践分享

# eBPF 最佳实践

- 借助 eBPF 了解内核，根据内核原理选择最佳的 eBPF 程序类型
- libbpf + CO-RE (Compile Once – Run Everywhere) + BTF
- 注意 eBPF 特性在不同内核版本的差异
- 利用 eBPF 帮助函数、映射与内核进行交互
- 善用 __always_inline、尾调用等降低复杂度并绕过 eBPF 验证问题
- 防御 eBPF 被恶意程序利用和伪装

# Thanks