
GOTC 2023
全球开源技术峰会
THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE
OPEN SOURCE, INTO THE FUTURE
「eBPF」专场
本期议题：BPF 技术在百度云原生实践
狄卫华 2023年05月28日

关于我

狄卫华

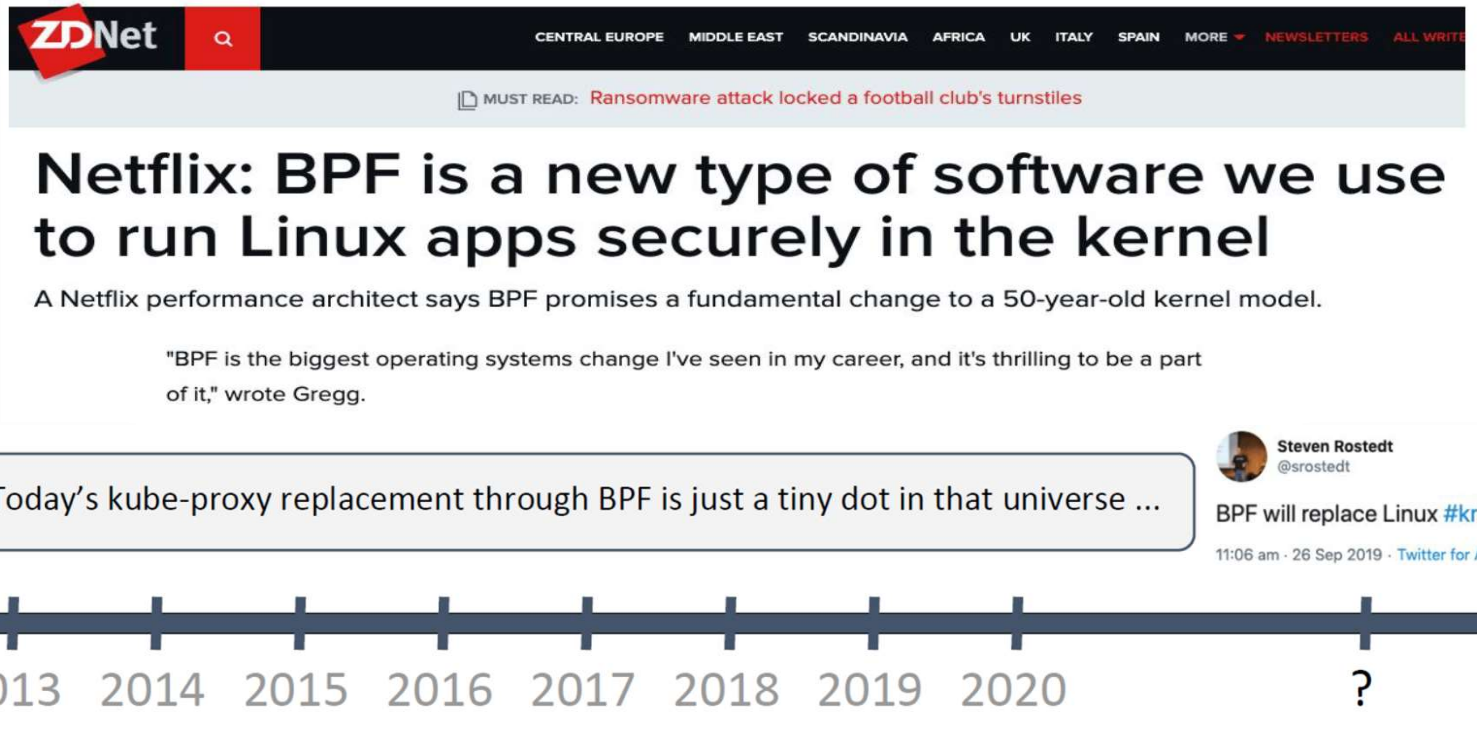
- 架构师@百度云原生团队
- 国内 BPF 技术布道师
- 《Linux内核观测技术 BPF》译者之一
- 国内首个 BPF 技术站点 ebpf.top

个人微信

1. BPF 技术简介

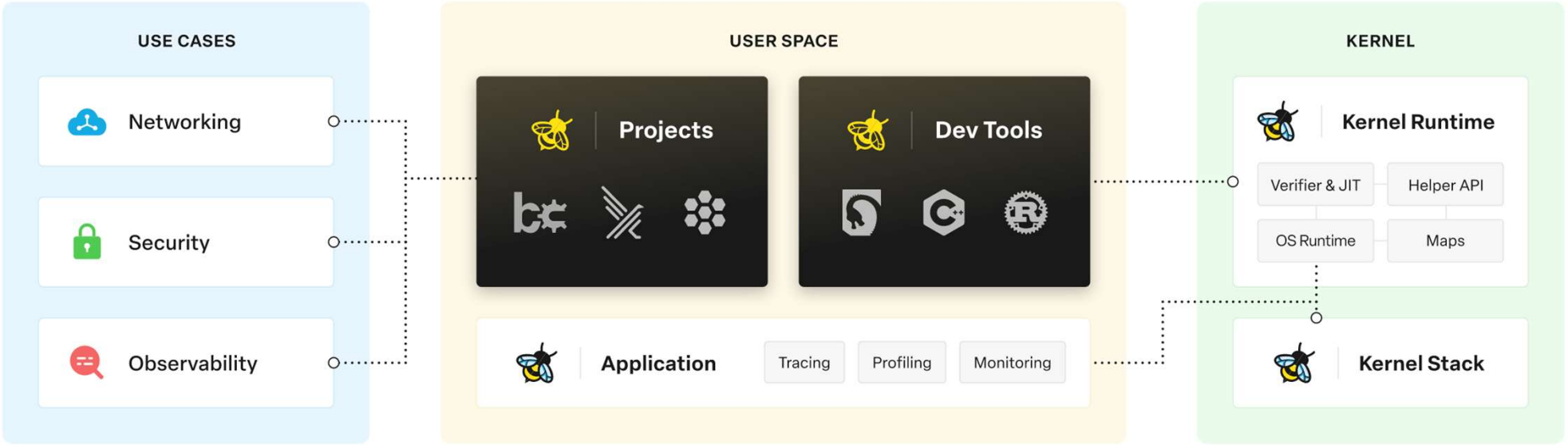
BPF 超能力诞生

BPF：过去 50 年操作系统最大的变革 !!!



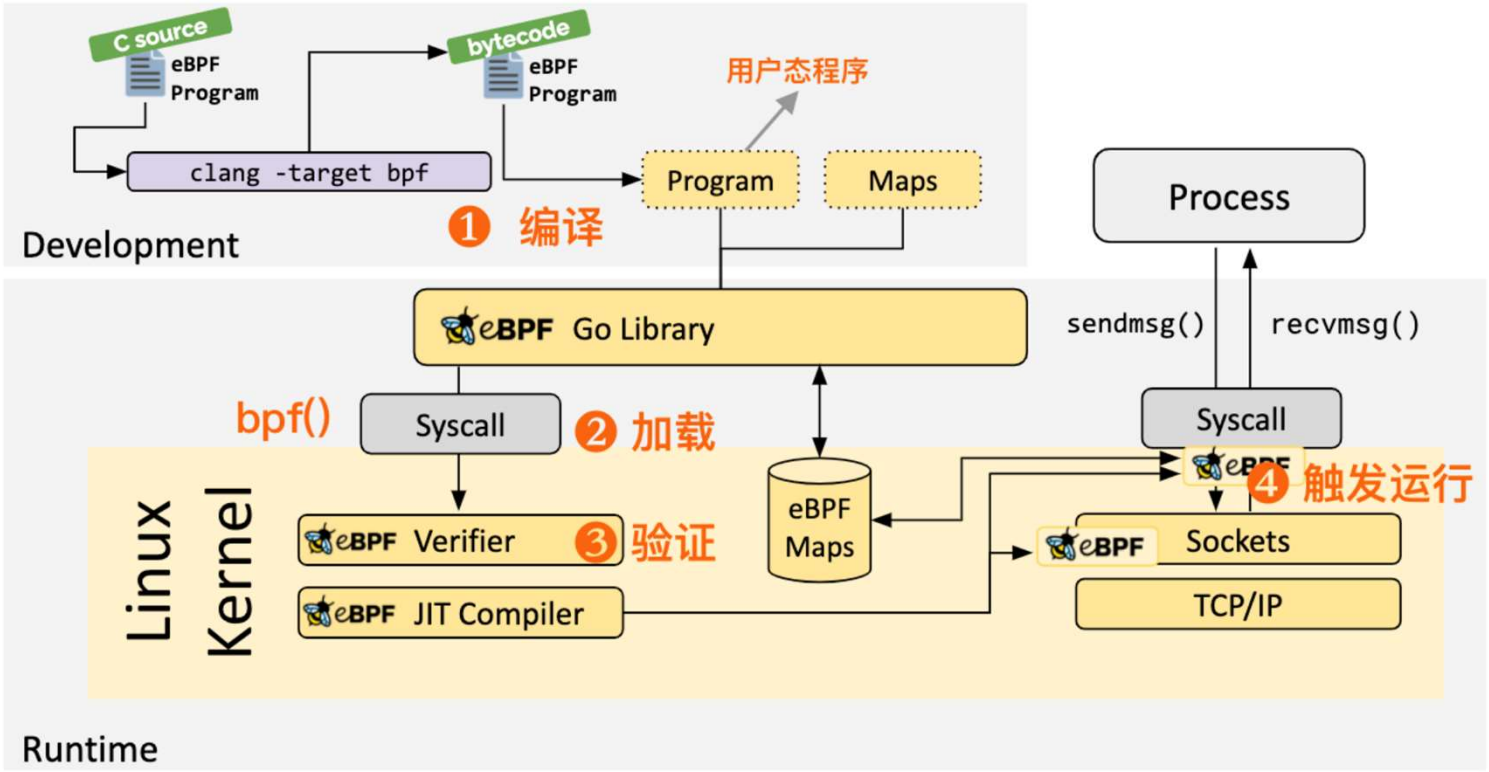
1. eBPF 技术简介

BPF 整体概览



1. BPF 技术简介

BPF 工作原理：内核空间高效稳定运行用户编写程序



1. BPF 技术简介

BPF 生产超能力：生产环境安全高效执行

JIT 后采用原生 CPU 指令

固定时间内可结束

无需重启机器，可热升级

数据与功能分离

1. BPF 技术简介

GOTC

BPF 生产超能力：广泛的生产级别应用

Google
安全审计/数据包处理/性能监测

NETFLIX
网络洞察力

android
网络监测/电源监控/内存性能

Meta
数据中心网络负载均衡

Apple
内核安全监控

CLOUDFLARE
网络安全/性能监测/网络可观测

Alibaba.com
云容器网络 Cilium

Trip.com
负载均衡

ByteDance
系统跟踪调试

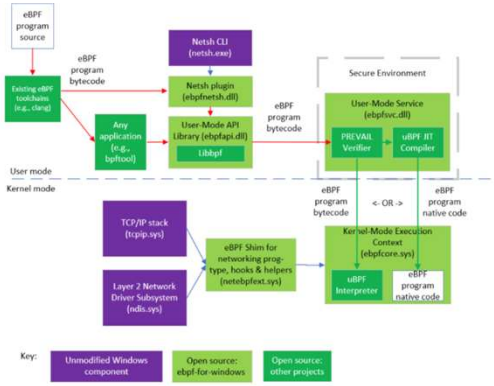
Baidu 百度
系统跟踪调试/容器网络加速

数据来源：https://ebpf.io/case-studies/，有调整

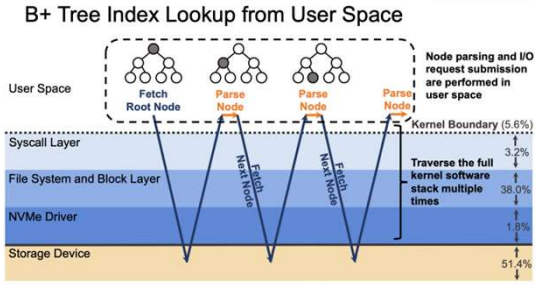
1. BPF 技术简介

BPF 技术未来： 是基石，也是平台能力补齐的核心拼图

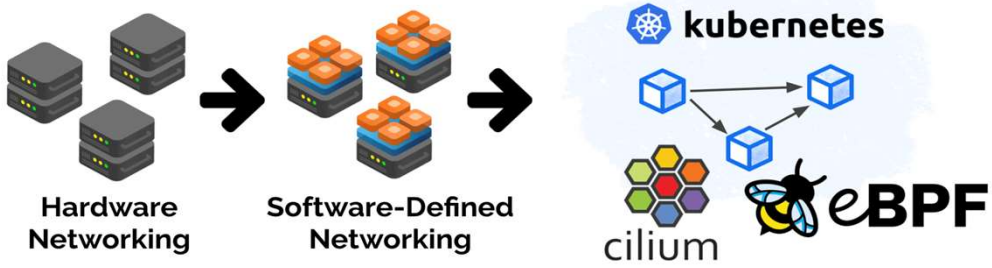
软件定义内核



攻城拔寨： Windows



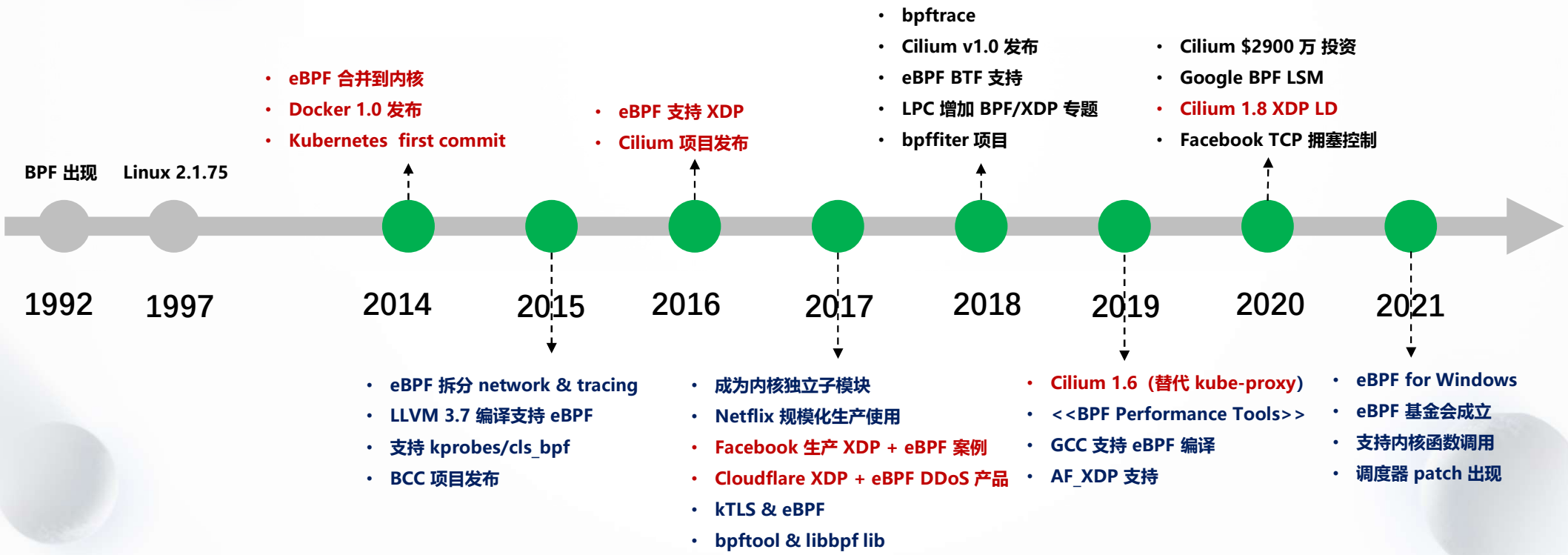
存储访问路径加速： XRP



eBPF 定义云原生网络

2. BPF 技术在云原生应用

BPF 与云原生

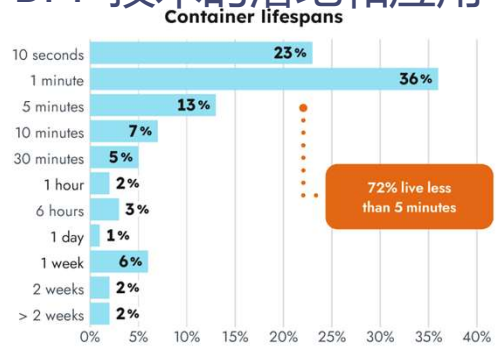


云原生与 eBPF 技术出现位于同期，发展过程相互促进，相互成就

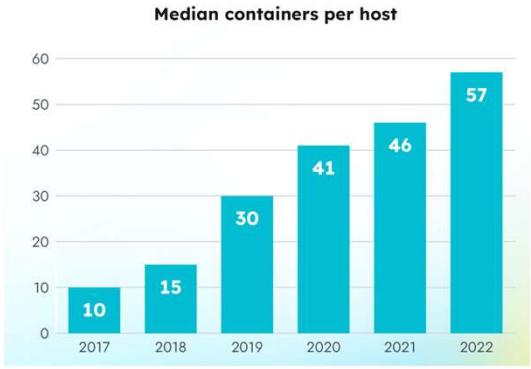
2. BPF 技术在云原生应用

云原生技术趋势：加速 BPF 技术的落地和应用

容器存活时间

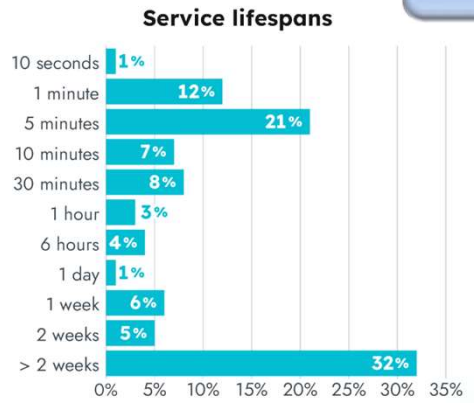


单机平均容器密度

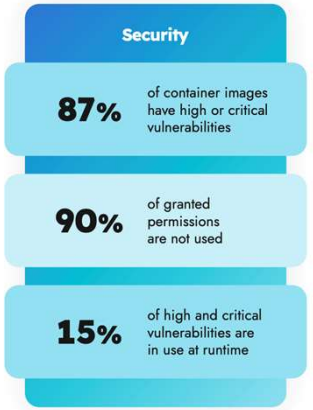


降本增效

服务存活时间



云原生安全挑战



数据来源： 2023 SysDig 云原生安全和容器使用报告

2. BPF 技术在云原生应用

云原生技术挑战：需要更强技术助力

- 开源组件多，版本多，问题排查链路长，涉及多团队协作，排查效率和协同效率面临挑战
- 容器实例存活时间变短，细粒度可观测和问题排查提出更高要求
- 单机部署容器实例密集度加速，离在线混部成为趋势，运行效率和质量细粒度感知需加强
- 服务生命周期敏捷化，基础能力敏捷化和透明化，部分场景网络性能较高诉求
- 云原生安全挑战与日俱增，运行时监测和网络安全增强需要新的技术手段护航

2. BPF 技术在云原生应用

GOTC

云原生应用： BPF 技术为云原生带来超能力

eBPF - Superpowers for Networking, Observability & Security

Liz Rice
Chief Open Source Officer, Isovalent
Chair, CNCF Technical Oversight Committee

@lizrice

ISOVALENT cilium

2. BPF 技术在云原生应用

BPF & 云原生： 超能力落地

应用程序和系统本身具有前所未有的可见性。内核内聚合，并基于广泛的可能来源生成可见性事件。

可编程性和效率的结合使得 BPF 自然而然地满足了网络解决方案的所有数据包处理要求。

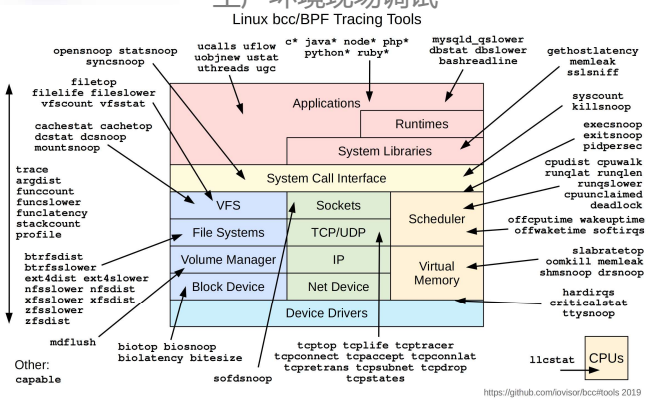
系统调用的基础上，将其与网络数据包和套接字级视图相结合，可以采用革命性的新方法来确保系统的安全。

2. BPF 技术在云原生应用

云原生可观测

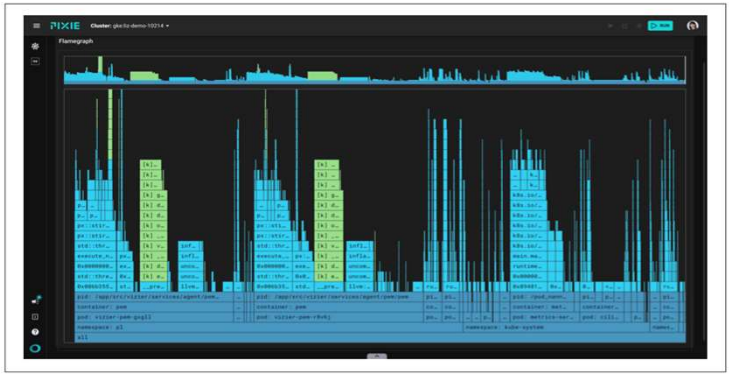
跟踪调试

生产环境现场调试



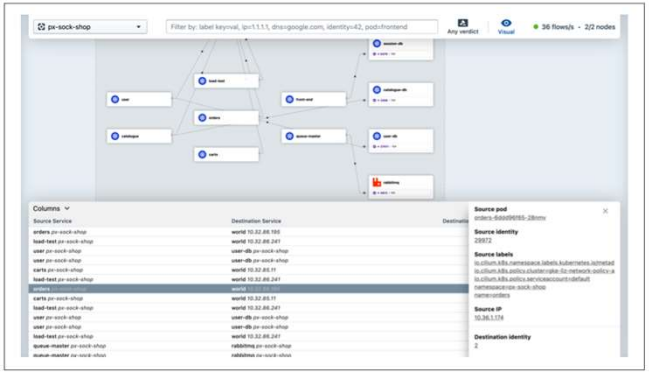
性能分析

应用透明，可内核级别聚合



服务网络拓扑

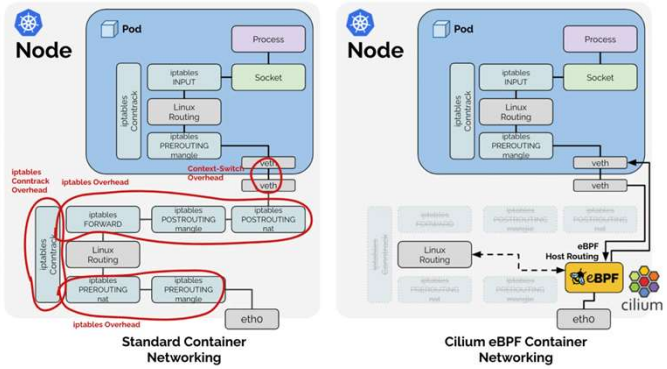
应用透明，细粒度事件采集



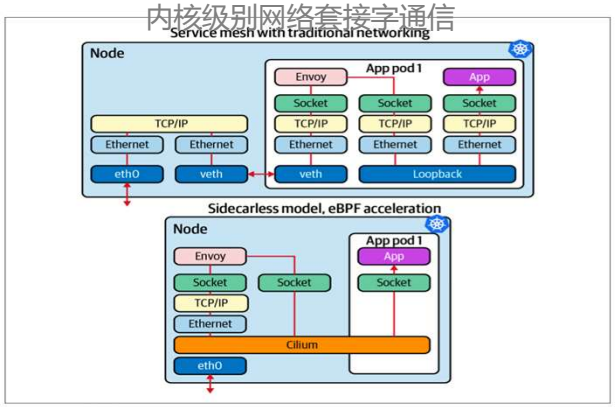
2. BPF 技术在云原生应用

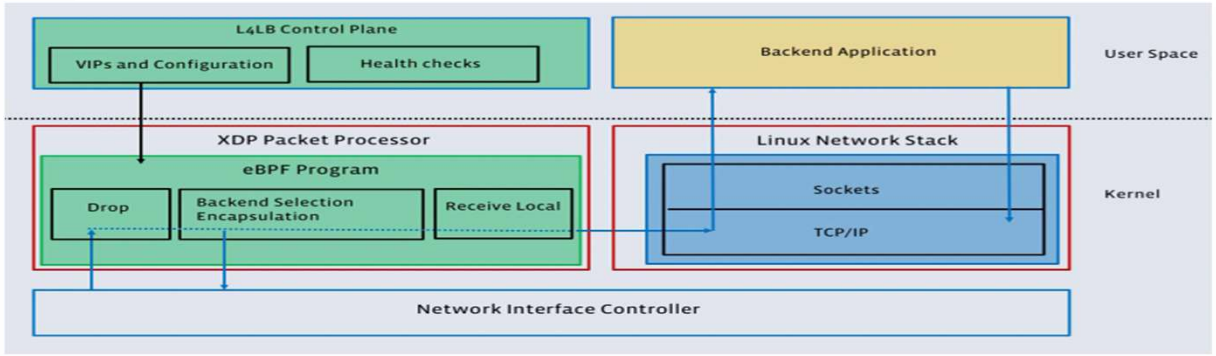
云原生网络

容器网络路径
绕过主机网络协议栈



Service Mesh 网络路径
内核级别网络套接字通信



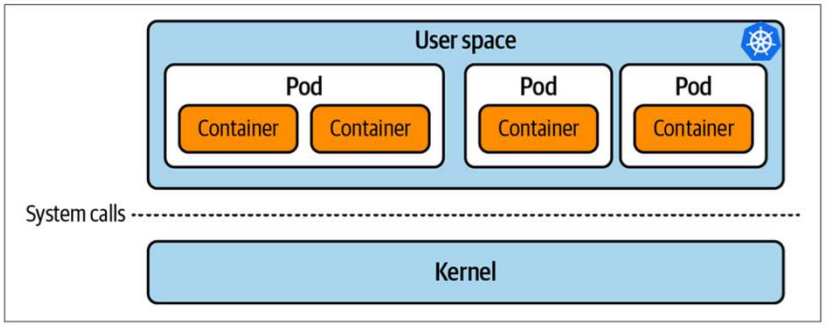


网络负载均衡器
传统 IPVS 技术 4 倍以上性能提升

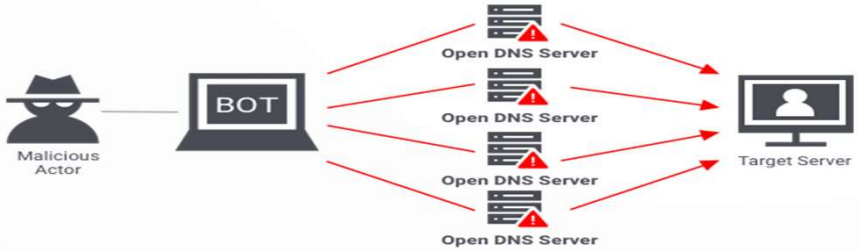
2. BPF 技术在云原生应用

云原生安全

运行时安全
内核视图联合检测

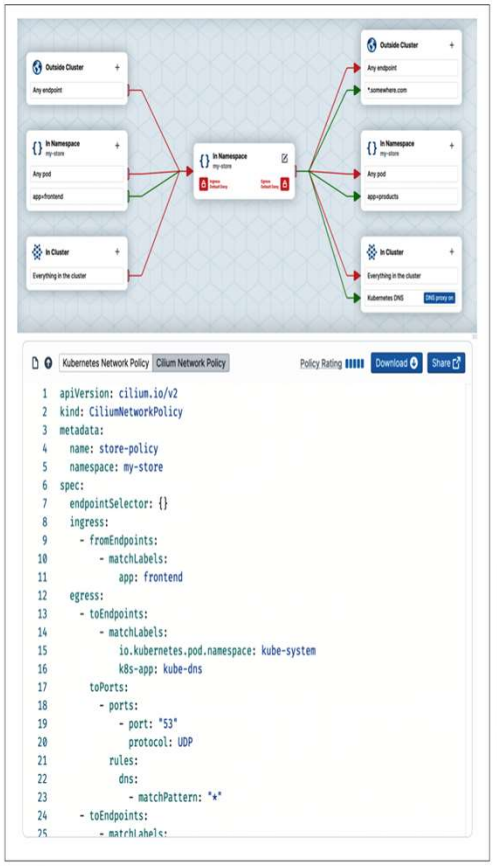


Falco　aqua tracee　Tetragon



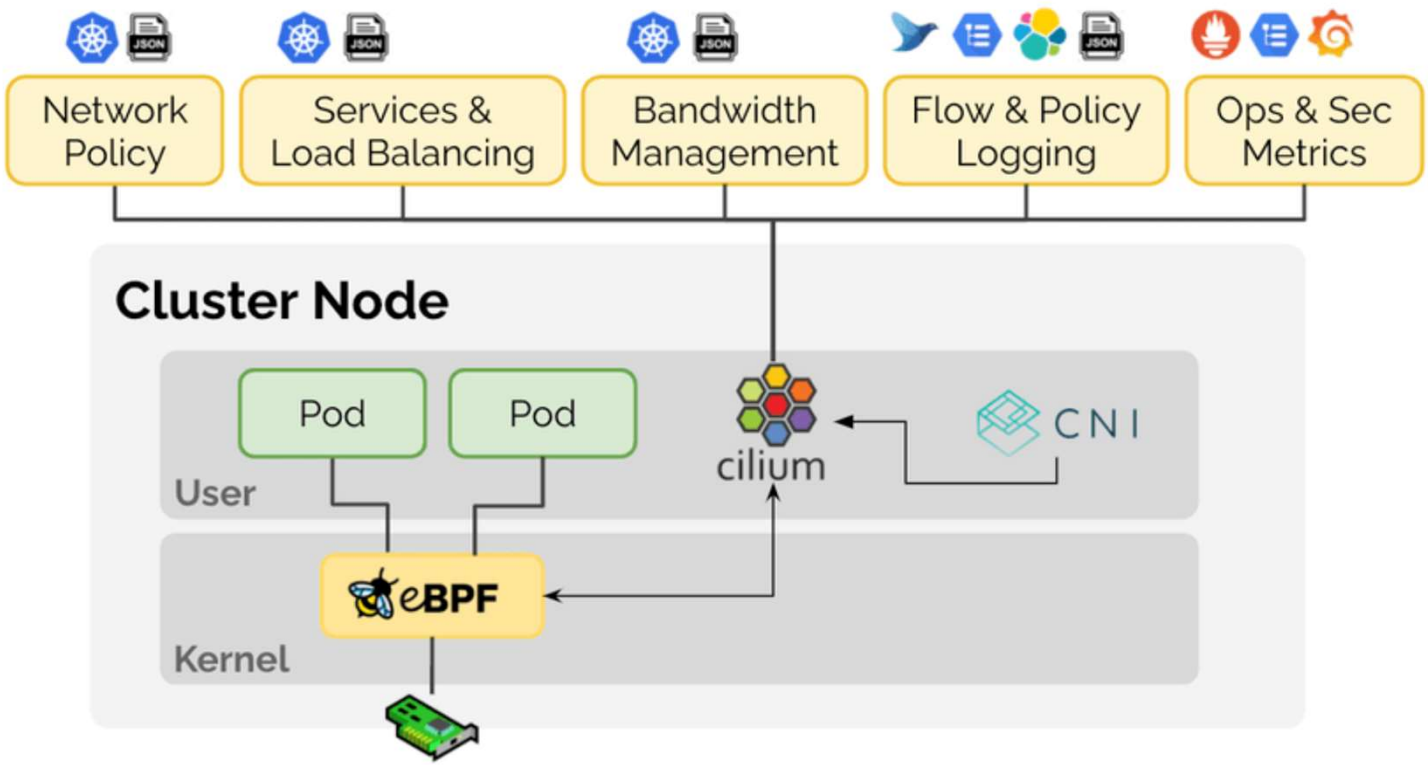
DDoS/网络防火墙
高效阻断和防御

网络安全策略
L3/L4/L7 服务标识管控

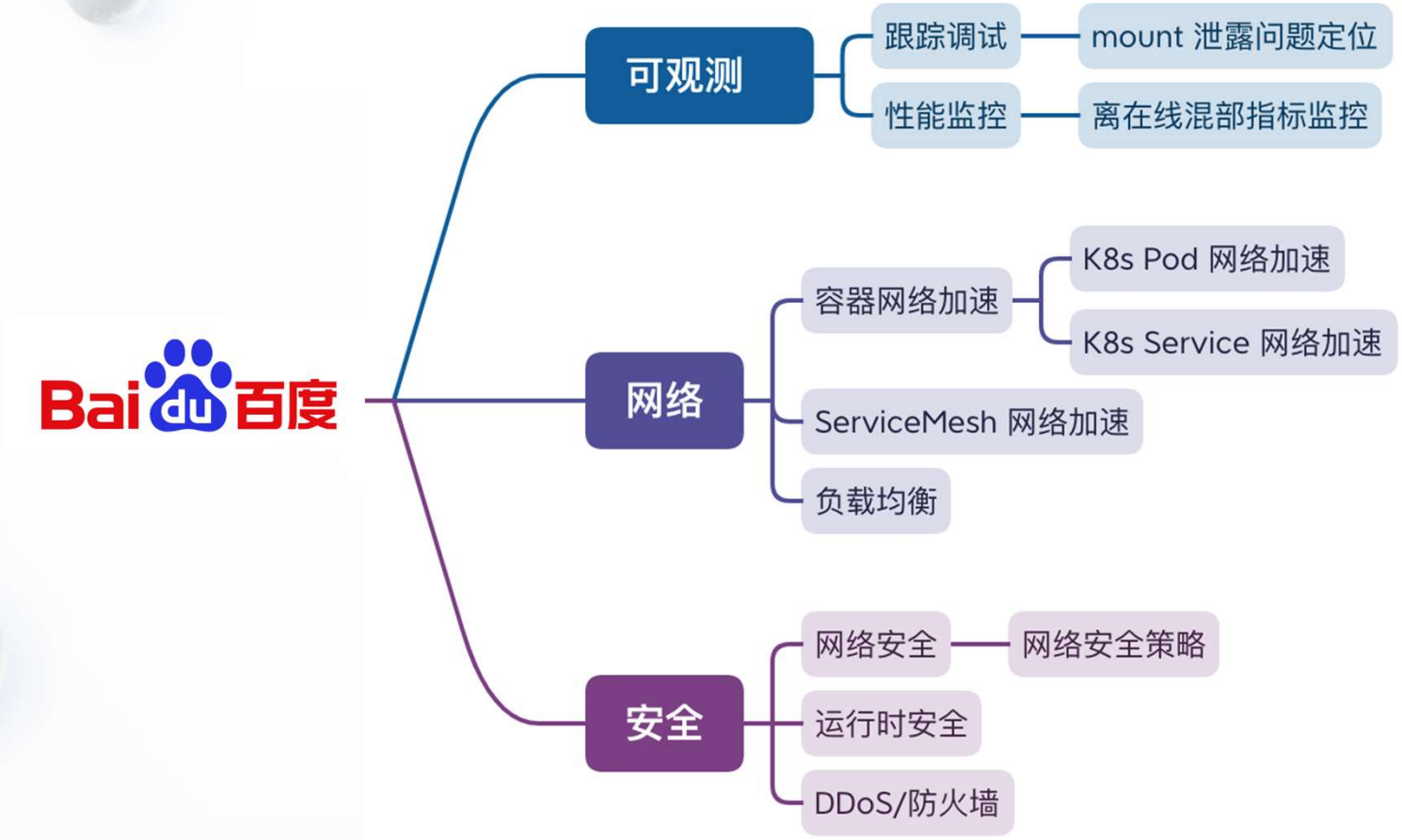


2. BPF 技术在云原生应用

BPF 技术在云原生领域集大成者：开源明星 Cilium

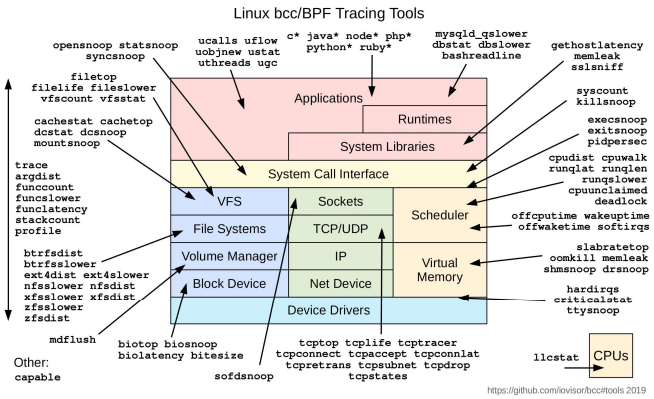


3. BPF 技术在百度云原生实践

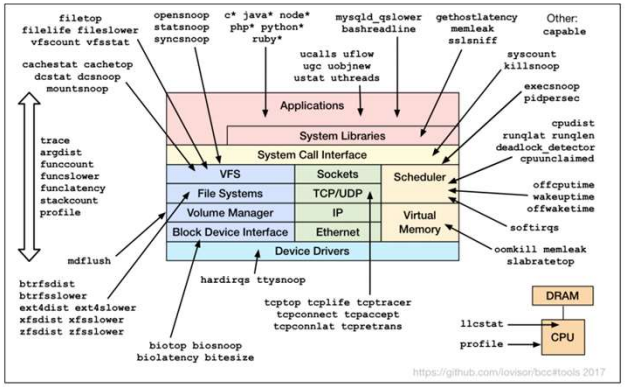


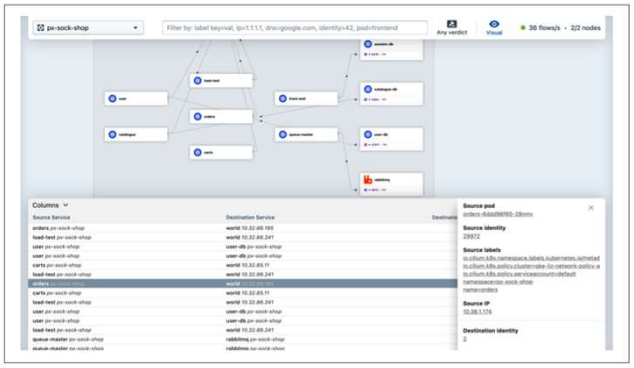
3. BPF 技术在百度云原生实践

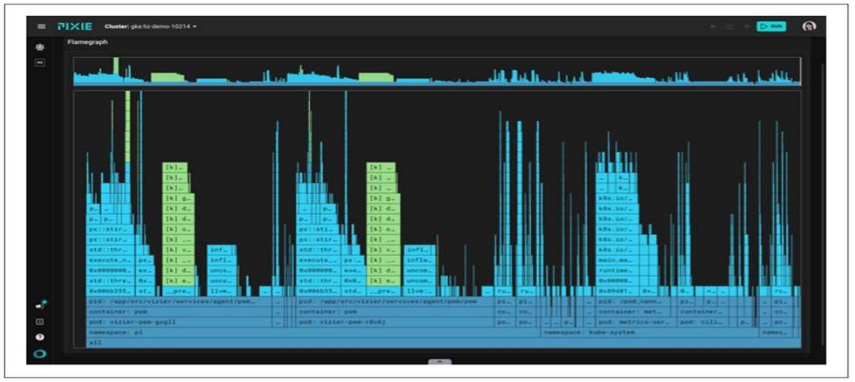
- 可观测 – 即开即用开源工具集



1. execsnoop
2. opensnoop
3. ext4slower (...)
4. biolatency
5. biosnoop
6. cachestat
7. tcpconnect
8. tcpaccept
9. tcpretrans
10. gethostlatency
11. runqlat
12. profile







3. BPF 技术在百度云原生实践

- 可观测 – 容器 mount 泄露

错误定位

部分 Node 上创建 Pod 发现：
mount: mount tmpfs on ... : No space left on device

磁盘空间、inode 和 文件句柄都正常

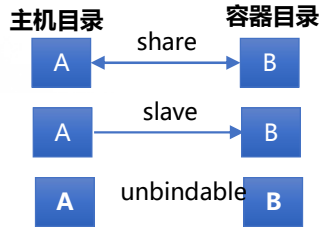
eBPF 技术工具定位错误来源
syscount-bpfcc -e ENOSPC 定位
__x64_sys_mount 系统调用错误

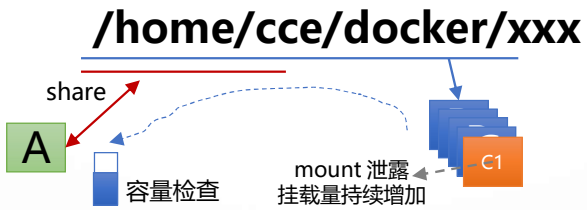
sudo ./funcgraph -m 2
__x64_sys_mount
-> path_mount()
-> do_new_mount()
-> do_add_mount()
...
-> count_mounts()

容器 mount 命名空间 /proc/sys/fs/mount-max
默认 10 万，mount 挂载实例 mount 挂载数量超出限制

部分容器实例 mountinfo 数量非常大
为什么单个 实例异常，影响新建容器失败?

寻根问底：mount 传播泄露





彻底分析：延迟卸载惹得祸

C1 **为什么 C1 mountinfo 持续增长?**

c1: /noah/:/noah 挂载

docker cp ../noah-ci.tar.gz $1:/tmp/noah-ci-new.tar.gz

docker cp 导致 c1 容器的 mountinfo 数量持续增加

```
$ findmnt
├─/noah          /dev/sda1[/noah]    ext4
│ └─/noah/bin    tmpfs               tmpf
```

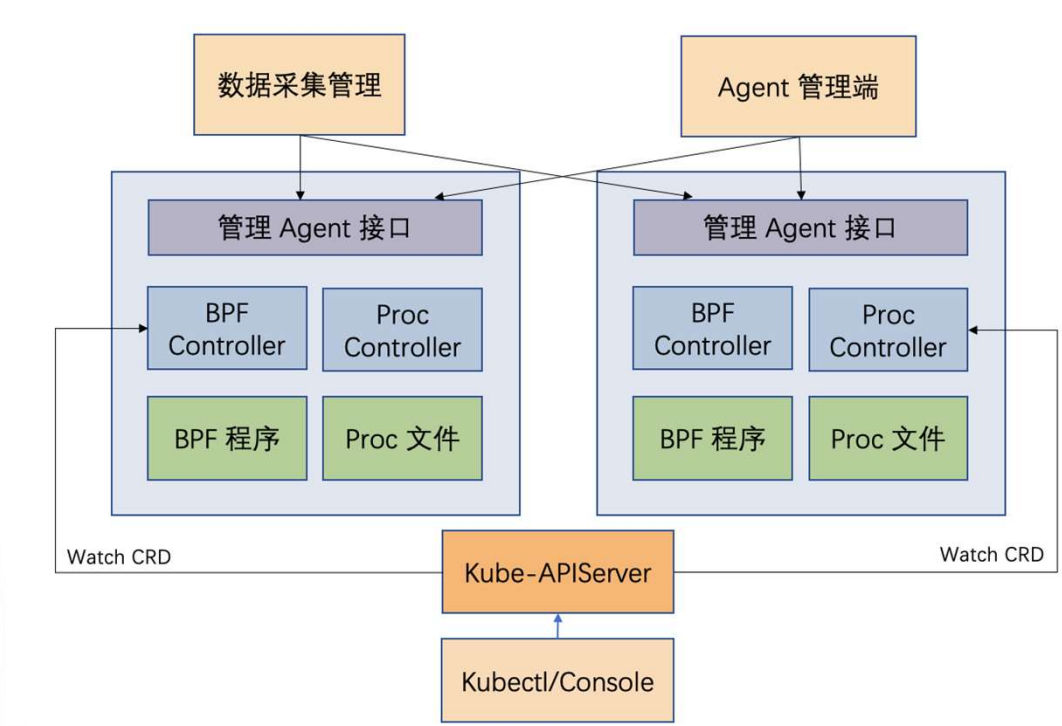
$ sudo mountsnoop-bpfcc → **递归 mount**
mount(... , **MS_BIND|MS_REC**, NULL)
umount(..., MNT_DETACH) → **延迟卸载**

```
static int do_umount(struct mount *mnt, int flags)
{
  //...
  if (flags & MNT_DETACH) {
    if (!list_empty(&mnt->mnt_list))
      umount_tree(mnt, UMOUNT_PROPAGATE); // here <==
    retval = 0;
  }
}
```

https://www.ebpf.top/post/no_space_left_on_devices/

3. BPF 技术在百度云原生实践

- 可观测 - 离在线混部内核指标感知，在线服务质量保障



内核指标采集

- CPU： 调度延时/CPI/Cache Miss
- 内存： 分配/回收延时/PSI
- IO： 延迟/吞吐/PSI
- 网络：连接延时/重传
- ...

功能设计

- 云原生模式管控
- 可基于内核版本控制
- 能够基于容器组 ID 过滤
- 指标以 Pod 名字展示
- 可实现统一管理和配置

3. BPF 技术在百度云原生实践

- 可观测 -离在线混部内核指标感知，在线服务质量保障

scripts/bpf/cpu-runq-latency-public.yaml

```yaml
apiVersion: halo.baidu.com/v1
kind: MetricsProgram
metadata:
  name: runqlat
spec:
  name: runqlat
  selector:
    workload_type: stable
  metrics:
    histograms:
      - name: run_queue_latency_seconds
        help: Run queue latency histogram
        table: runqlat_dist
        bucket_type: exp2
        bucket_min: 0
        bucket_max: 26
        bucket_multiplier: 0.000001 # microseconds to seconds
        labels:
          - name: pod
            size: 8
            decoders:
            - name: uint
            - name: pod
          - name: bucket
            size: 8
            decoders:
            - name: uint
  bpf_programs:
    - cgroups_table: cgroup_table
      kernel_version: 5.10.0-1.0.0.13
      kprobes:
        ttwu_do_wakeup: trace_ttwu_do_wakeup
        wake_up_new_task: trace_wake_up_new_task
        finish_task_switch: trace_run
      code:
        text: |
          #include <linux/sched.h>
```

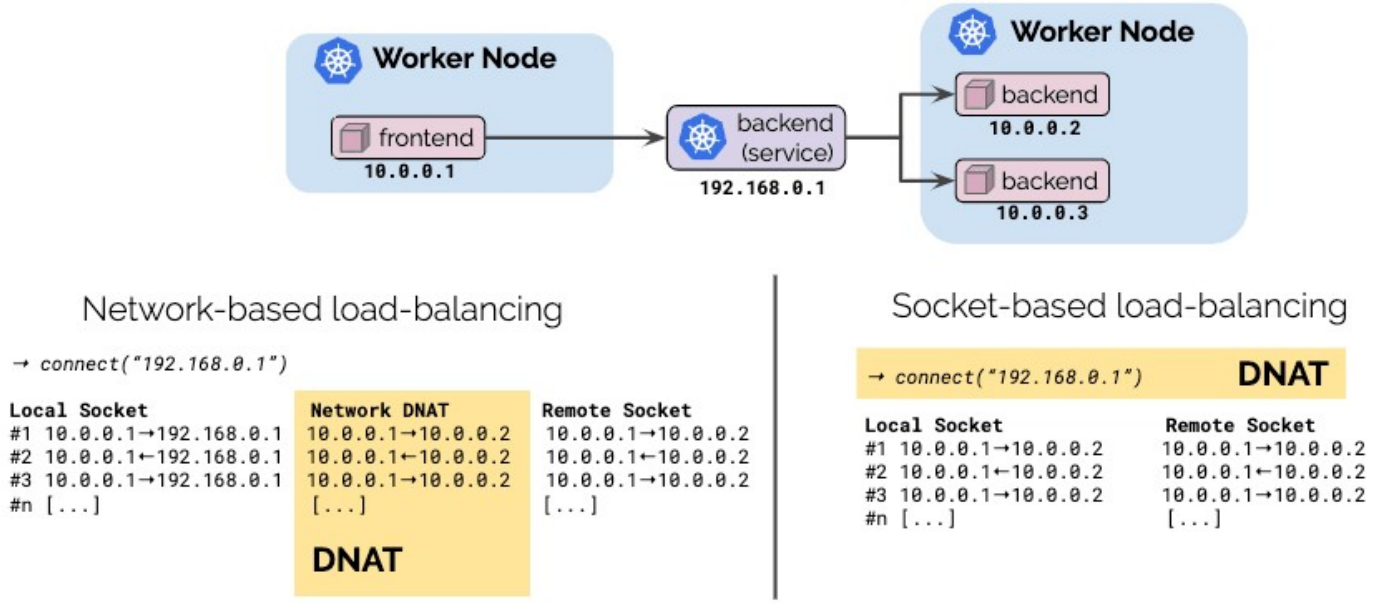
```c
#include <linux/cgroup.h>

typedef struct pidns_key {
    u64 id;    // cgroup id
    u64 slot;
} pidns_key_t;

typedef struct cgroup_task_key {
  u64 id;    // cgroup id
  s32 task_num;
  u64 ts;
} cgroup_task_key_t;

BPF_HASH(cgroup_table, u64, u32);
BPF_HASH(start, u32);
BPF_HASH(runqlat, u64, cgroup_task_key_t);
BPF_HISTOGRAM(runqlat_dist, pidns_key_t, 1024);

static u64 get_cid(struct cgroup_subsys_state *css) {
    return css->cgroup->kn->id;
}

static u64 get_container_id_of_task(struct cgroup_subsys_state *css) {
```

3. BPF 技术在百度云原生实践

- 可观测- 离在线混部内核指标感知，在线服务质量保障

3. BPF 技术在百度云原生实践

- 容器网络加速 – 基于 Cilium K8s Service 加速

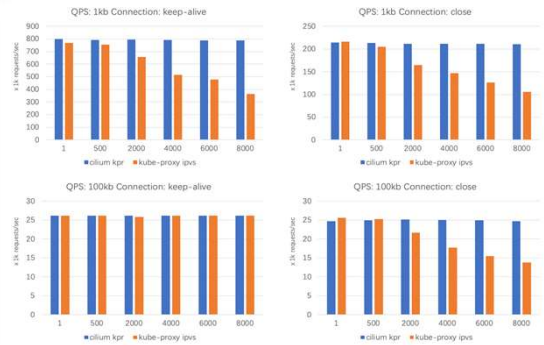


- **透明**：负载均衡对应用程序保持 100% 透明。服务是使用标准的 Kubernetes 服务定义定义的。
- **高效**：通过在 connect(2) 系统调用中转换地址实现套接字层负载均衡，负载均衡的成本在建立连接时预先支付，在连接之后就不需要额外的转换。性能与应用程序直接访问后端的性能相同。

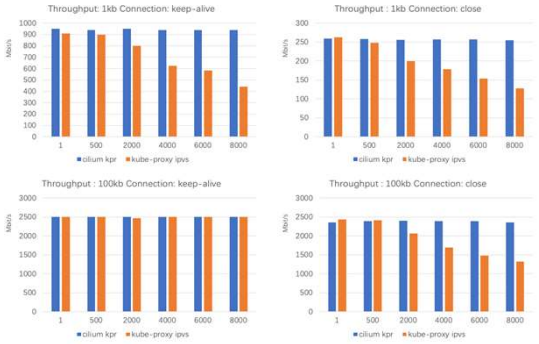
3. BPF 技术在百度云原生实践

- 容器网络加速 – 基于 Cilium K8s Service 加速

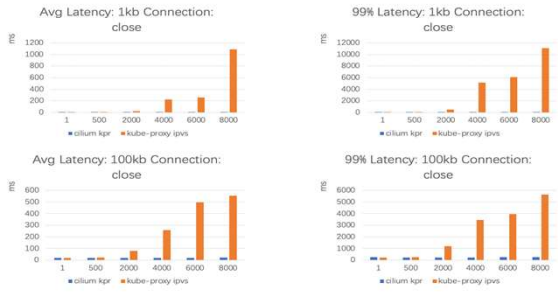
QPS



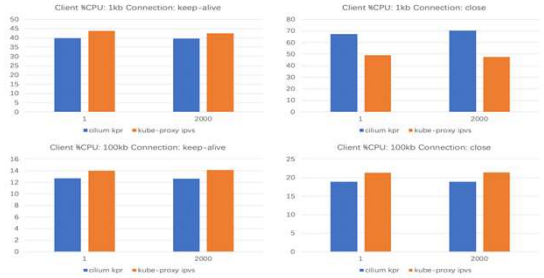
Throughput



Latency



CPU Usage



Cilium Service 加速在网络延迟方面有明显的优势，并且几乎不会随着 Service 数量的增多而上涨，具备保留源 IP 和支撑规模化 NetworkPolicy

cilium

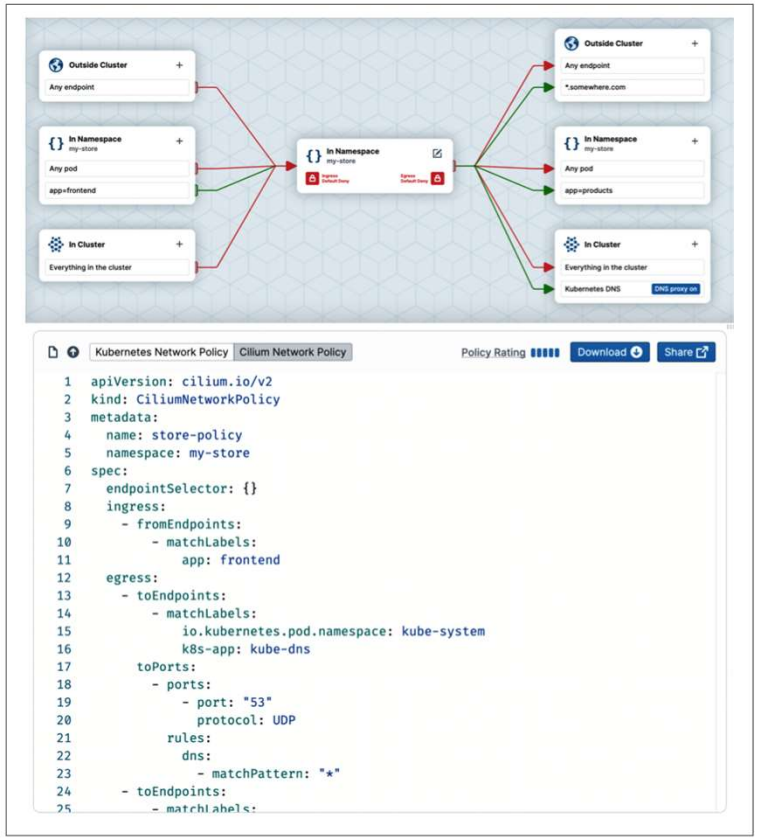
公私统一架构，多场景演进

3. BPF 技术在百度云原生实践

- 安全 – 容器网络策略增强，服务标识 & L3/L4 层控制

- 禁止namespace=staging中Pod被访问。

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
  namespace: staging
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```



3. BPF 技术在百度云原生实践

GOTC

- 实践思考

- BPF 技术是基础能力，需要针对解决问题的场景有足够的认知和分析（纵向与横向），如内核中进程/内存/文件/设备/网络等子系统
- BPF 并非原有工具和技术的替换，相互补充，提供低门槛和可编程的灵活性（最后一公里和细粒度的监测）
- 基于问题解决使用 BPF 技术，避免用 BPF 技术去找问题

- 尽量选取能使用的最高内核版本，内核版本越高 BPF 特性支持越全
- 尽可能基于 BTF & CO-RE（Compile Once–Run Everywhere），提升可移植性
- 优先使用内核稳定跟踪机制， 提升多内核版本可移植性，内核版本尽量收敛
- 关注 BPF 附着事件触发频繁，避免出现不可控的性能开销

GOTC

THANKS

全球开源技术峰会

THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE