第三届 eBPF开发者大会
www.ebpftravel.com

中国联通eBPF的探索与实践

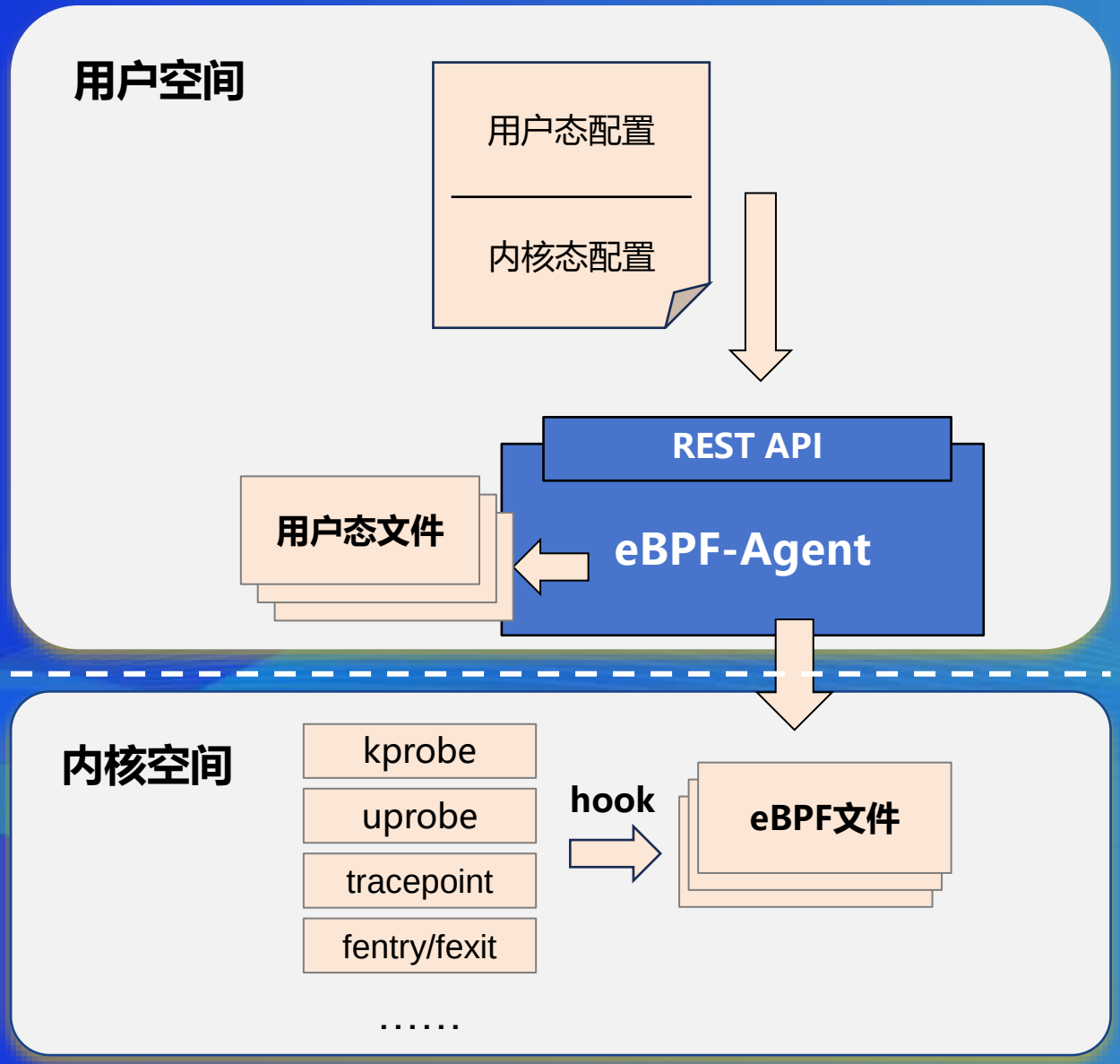
中国·西安

1. 可观测性演进

演进落地实践思路

- 新特性可以快速加载，业务受影响快速卸载
- eBPF技术在网络、计算、IO应用场景
- 可以实时编写 bpftrace 程序，针对特定场景进行数据抓取
- 获取eBPF技术具有相关业务属性标签

用户态和内核态程序加载




```
programs:
  - name: tcp_tracer
    path: [file|http|ftp]:///xxx/tcp_tracer.bpf.o
    type: [ebpf_object|bpftrace]
  - name: tcp_connlat_user
    path: http://xxxx//tcp_tracer.so
    type: user_so
    properties:
      progs: xxxx
      maps: xxx
      variables: xxx
```

支持远程和本地加载

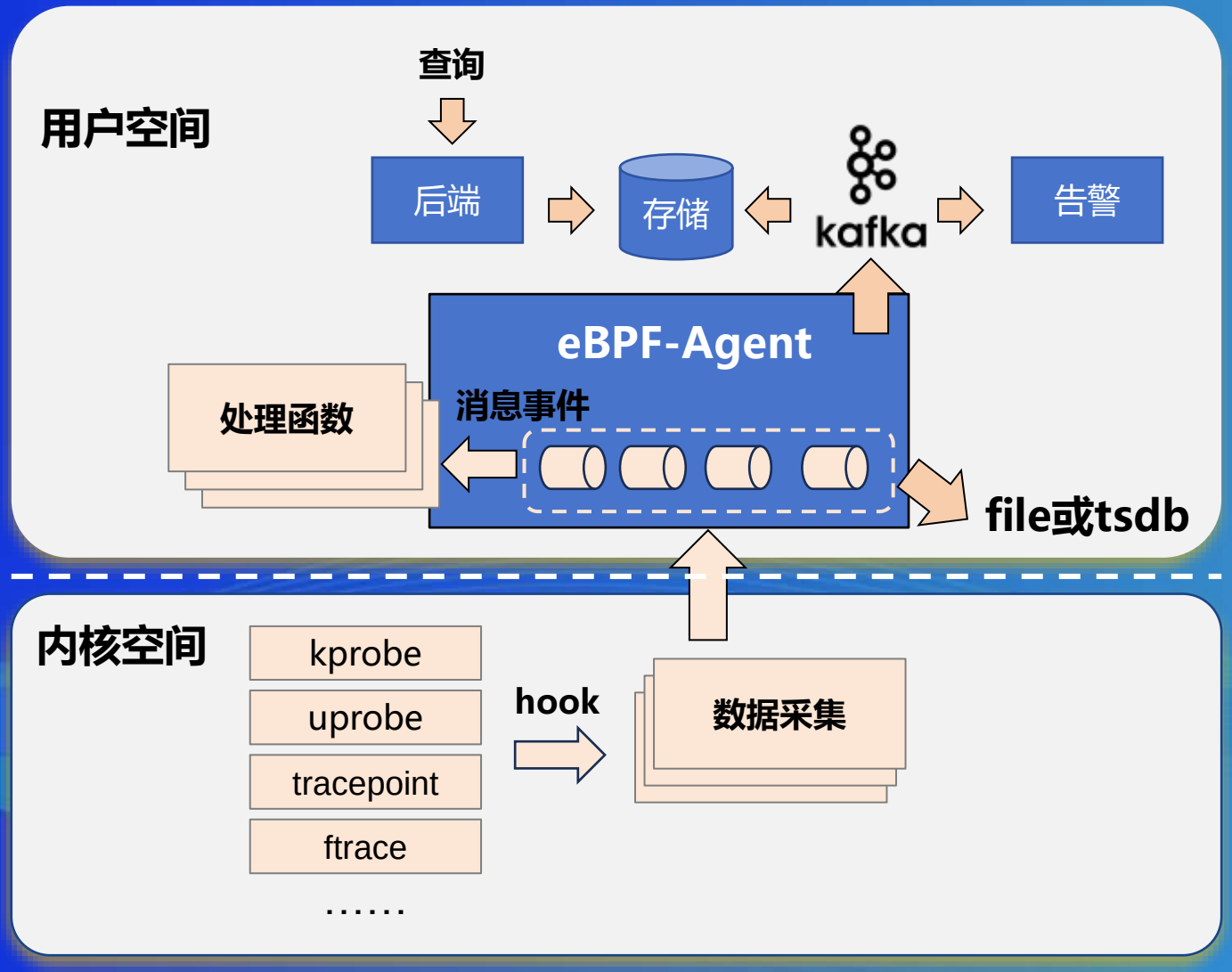
支持eBPF程序声明解析

支持动态加载用户态处理程序.so

支持用户函数初始化、preload/postload操作

支持 eBPF map 初始数据灌入

数据采集与消息处理




```
channels:
  tcp_tracer_events:
    type: ring_buffer
    events:
      - name: tcp_connlat_user
        hash: hash
        target_processor: process
        storage:
          kafka: true
          tmpfs: true
```

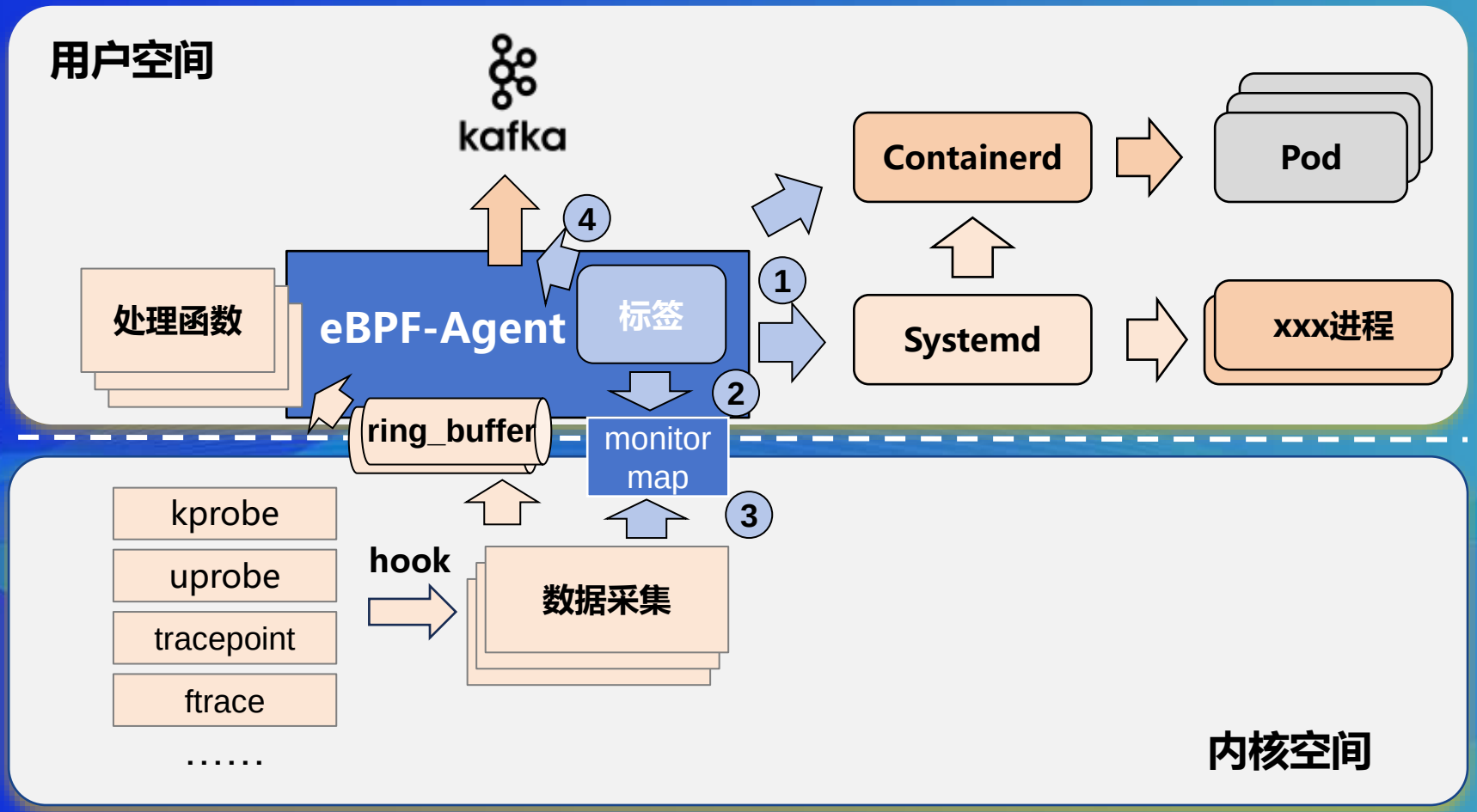
支持kafka、本地文件及内嵌tsdb输出

支持全局配置和单独配置

支持数据采集、消息通道与用户函数多对多关联处理

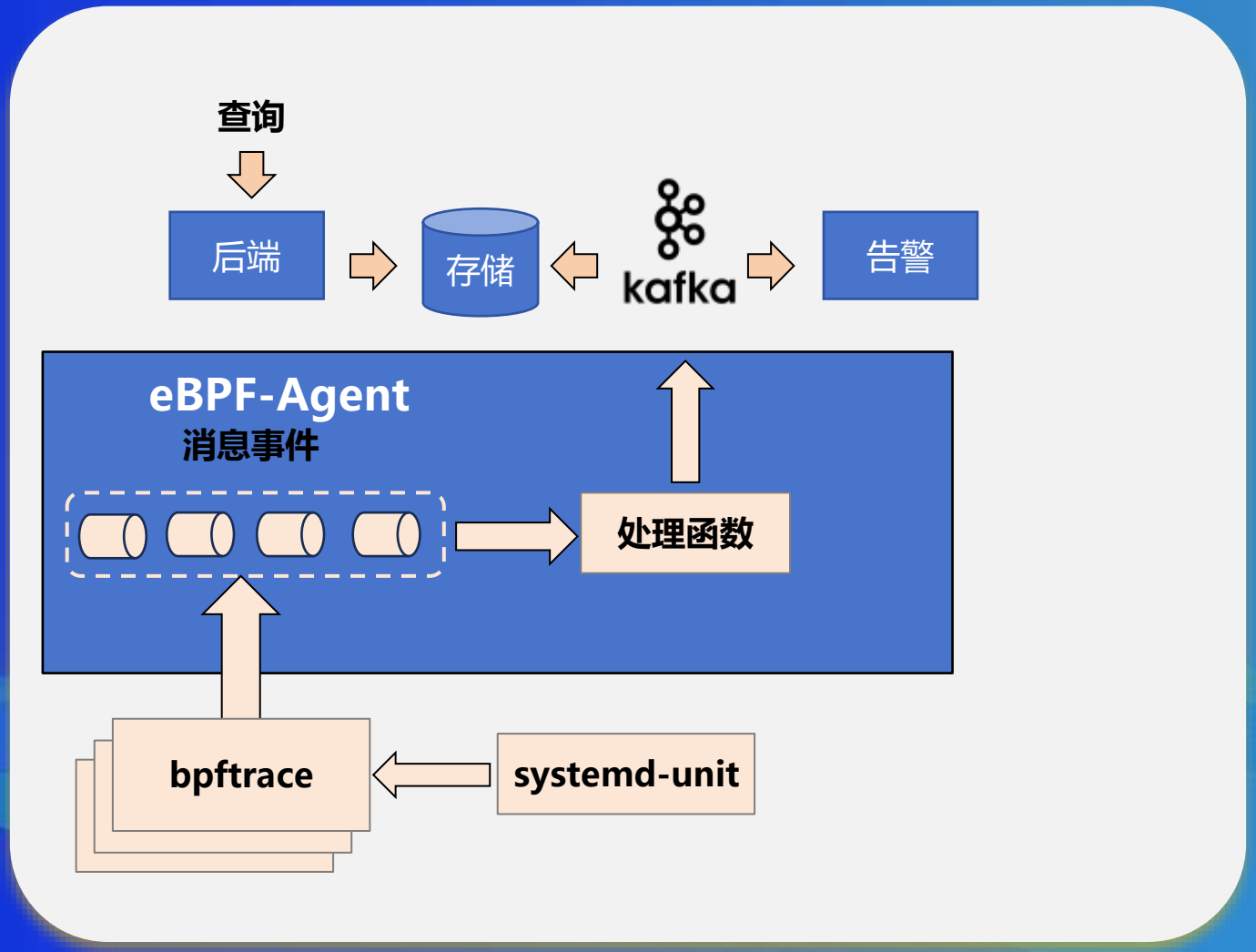
支持进程过滤与业务标识

- 支持监听 Containerd、Systemd(DBus)事件
- 支持子进程，业务属性关联，如Nginx，Envoy等
- 支持可限定wactch范围，降低无关事件对eBPF-Agent进程缓存的压力



1. 标签模块分别监控Systemd进程创建的进程和Containerd创建的pod
2. 将需要采集数据的进程写入monitor map
3. 内核根据标签过滤进程数据，按需采集数据
4. 标签模块在传输数据之前附加进程业务属性标识

bpftrace DSL 支持



支持实时下发bpftrace程序

支持bpftrace 程序数据解析自定义

```
programs:
  - name: openfile
    path: file:///xxx/openfile.bt
    type: bpftrace
  - name: tcp_connlat_user
    path: http://xxxx//openfile.so
    type: user_so
    properties:
      initfile:
      functions:
        - name: process
          type: default
.........
```

通用eBPF程序配置示例

内核C代码

```
struct
{
    __uint(type, BPF_MAP_TYPE_RINGBUF);
    __uint(max_entries, 1 << 20);
} oom_events SEC(".maps");
…
KPROG(oom_kill_process)
(kprobe_oom_kill_process, struct oom_control
*oc, const char *message)
{
…

    victim = BPF_CORE_READ(oc, chosen);
…
    event = bpf_ringbuf_reserve(&oom_events,
sizeof(struct oom_event), 0);
    if (event)
    {
BPF_CORE_READ_INTO(&event->pid, victim,
pid);
        BPF_CORE_READ_INTO(&event->tgid,
victim, tgid);
event->state = get_task_state(victim);
        bpf_probe_read_str(&event->message,
sizeof(event->message), message);
event->exit_code = (BPF_CORE_READ(victim,
exit_code) >> 8) & 0xff;
        bpf_ringbuf_submit(event, 0);
    }
    return 0;
}
```

用户函数Go代码

```
…
func process(ctx uintptr) {
  var event oomEvent
  traceCtx := (*TracerCtx)(unsafe.Pointer(ctx))
  err :=
binary.Read(bytes.NewBuffer(traceCtx.RingCtx.
Data),   binary.LittleEndian, &event)
  if err != nil {
    traceCtx.ErrCtx.ErrCode = 1
    return
  }
  result := map[string]any{
  "comm":       byteArrayToString(event.Comm[:]),
  "message":
byteArrayToString(event.Message[:]),
  "timestamp":
bpfKtimeToUnixTimestamp(event.TsNs),
  "delta_time": event.DeltaTime,
  "tid":        event.Pid,
  "pid":        event.Tgid,
  "ppid":       event.Ppid,
  "state":      event.State,
  "exit_code":  event.ExitCode,
  }
  resData, _ := json.Marshal(result)
  traceCtx.ResCtx.ResLen =
(uint32)(len(resData))
  copy(traceCtx.ResCtx.Data, resData[:])
}
…
```

装卸Yaml配置

```
kind: ebpfprogram
name: oom
description: oom
version: "0.1"
author: ebpfprogram
spec:
  programs:
    - name: oom
      path: file:///root/ebpf-
agent/output/oom.bpf.o
      type: ebpf_object
    - name: oom_user
      path: file:///root/ebpf-agent/output/oom.so
      type: user_so
      properties:
        functions:
          - name: process
  channels:
    oom_events:
      type: ring_buffer
      events:
        - name: oom_user
          hash: hash
          target_processor: process
```

bpftrace配置示例

bpftrace 代码

```
kprobe:oom_kill_process
{
    $victim = *(struct task_struct *)arg0->chosen;
    $ts_ns = nsecs;
    $start_time = $victim->start_time;
    $delta_time = $ts_ns - $start_time;

    $pid = $victim->pid;
    $tgid = $victim->tgid;
    $ppid = $victim->real_parent->pid;
    $comm = str($victim->comm);
    $state = $victim->__state
    $exit_code = ($victim->exit_code >> 8) &
0xff;
    $msg = str(arg1);

  printf("OOM KILL: ts=%d pid=%d tgid=%d
ppid=%d state=%d code=%d comm=%s
msg=%s\n",
          $ts_ns, $pid, $tgid, $ppid, $state,
$exit_code, $comm, $msg);
}
```

装卸Yaml配置

```
kind: ebpfprogram
name: test_bpftrace
description: ''
version: '0.1'
author: ebpfprogram
spec:
  programs:
    - name: bpftrace_src
      type: bpftrace
      properties:
        src:    'bpftrace_code'
    - name: bpftrace_user_so
      path: 'file:///root/ebpf-
agent/output/bpftrace_default.so'
      type: user_so
      properties:
        functions:
          - name: process
  channels:
    bpftrace_channel:
      type: pipe
      pinpath: /var/lib/ebpf-
agent/bpftrace/test_bpftrace.bpftrace_src/output
      events:
        - name: bpftrace_user_so
          hash: hash
          target_processor: process
```

1. 无需编写用户态代码
2. 实时灵活插拔
3. 代码编写量压降

各维度eBPF可观测能力

文件

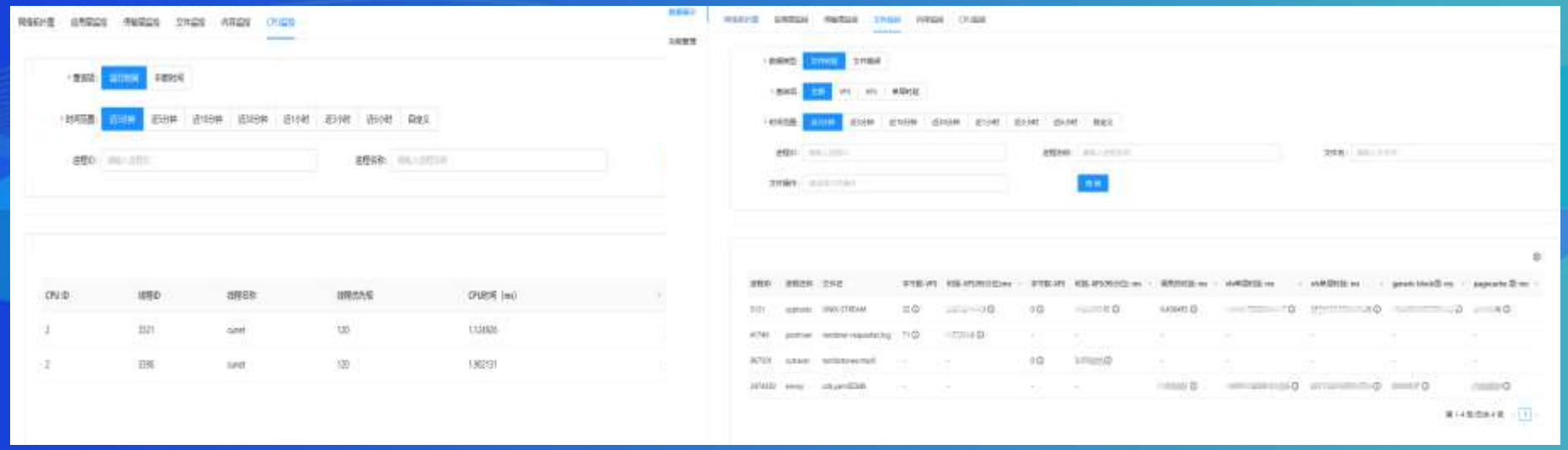
- 读写时延
- 读写错误
- 锁

CPU

- 唤醒时延
- 状态切换
- 队列长度
- 火焰图

内存

- 内存溢出
- 缺页错误
- 内存泄漏



第三届 eBPF开发者大会

www.ebpftravel.com

2.XDP网络加速探索

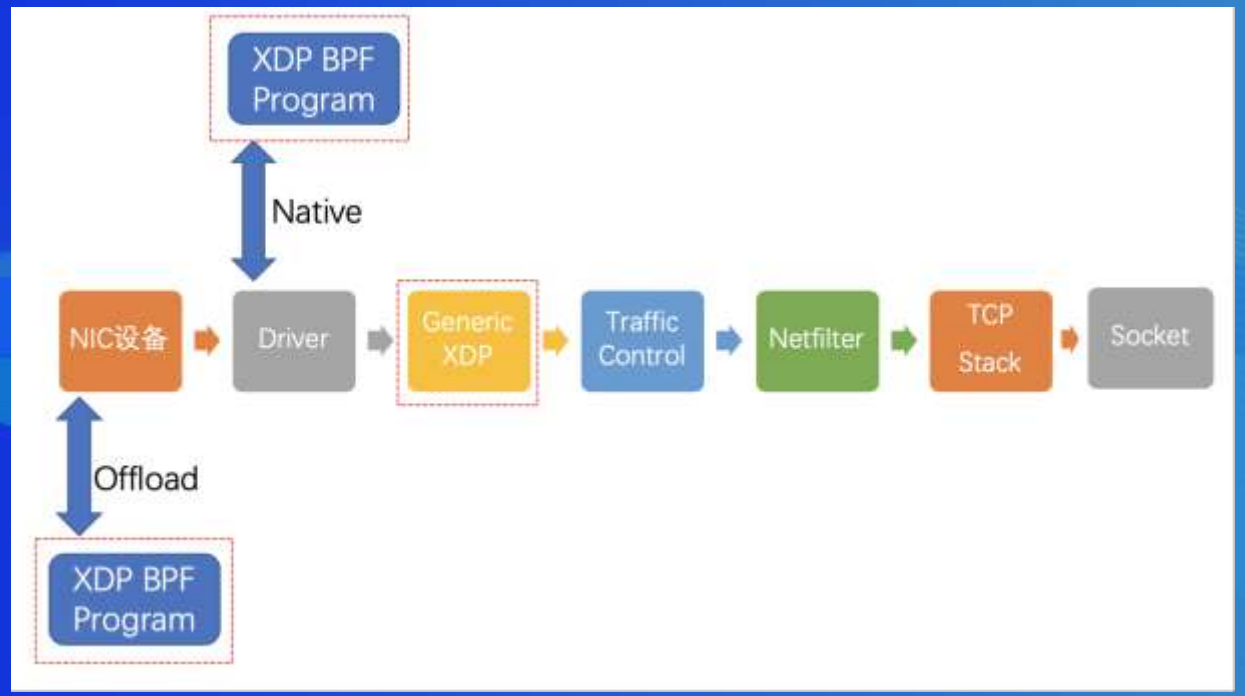
中国·西安

eBPF网络加速方案选型

	XDP（Native）	TC
位置	网卡驱动层，未进内核协议栈	网络协议栈的入口（ingress）或出口（egress）
性能	高	较高
兼容性	需驱动支持	无需驱动支持
上下文	无SKB	可访问SKB

网络专用机场景需要：

1. **快速重定向（负载均衡数据包）**
2. **数据包修改**
3. **统计计数**

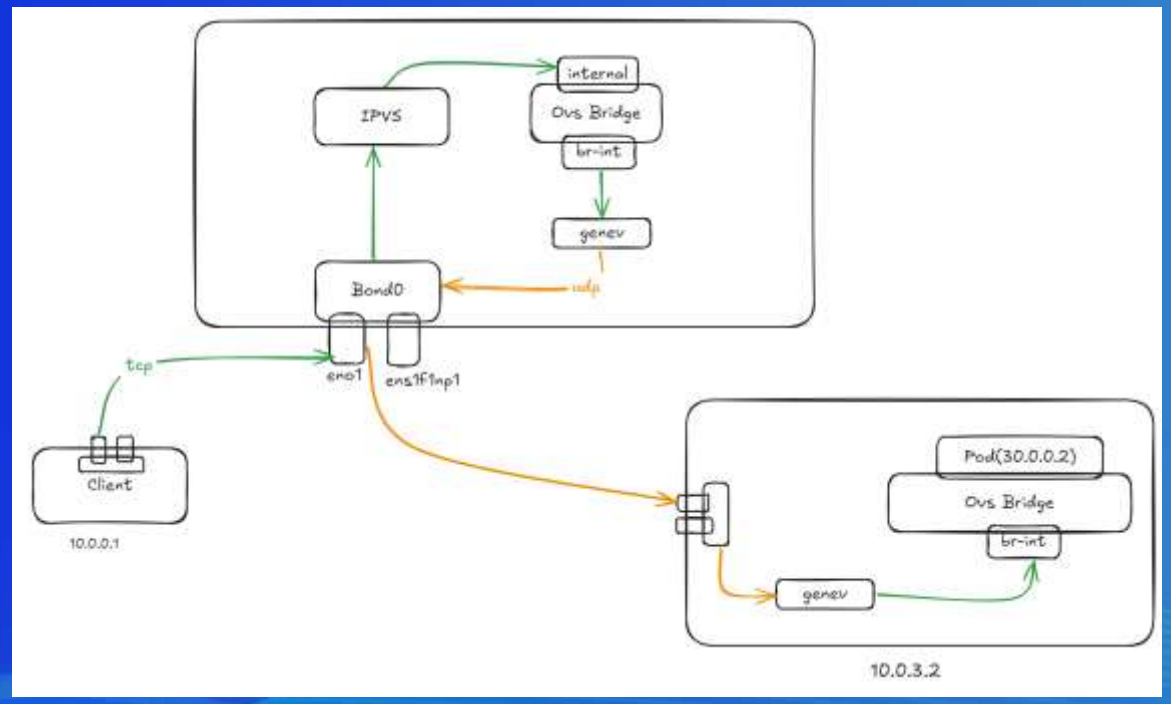


XDP方案选型：

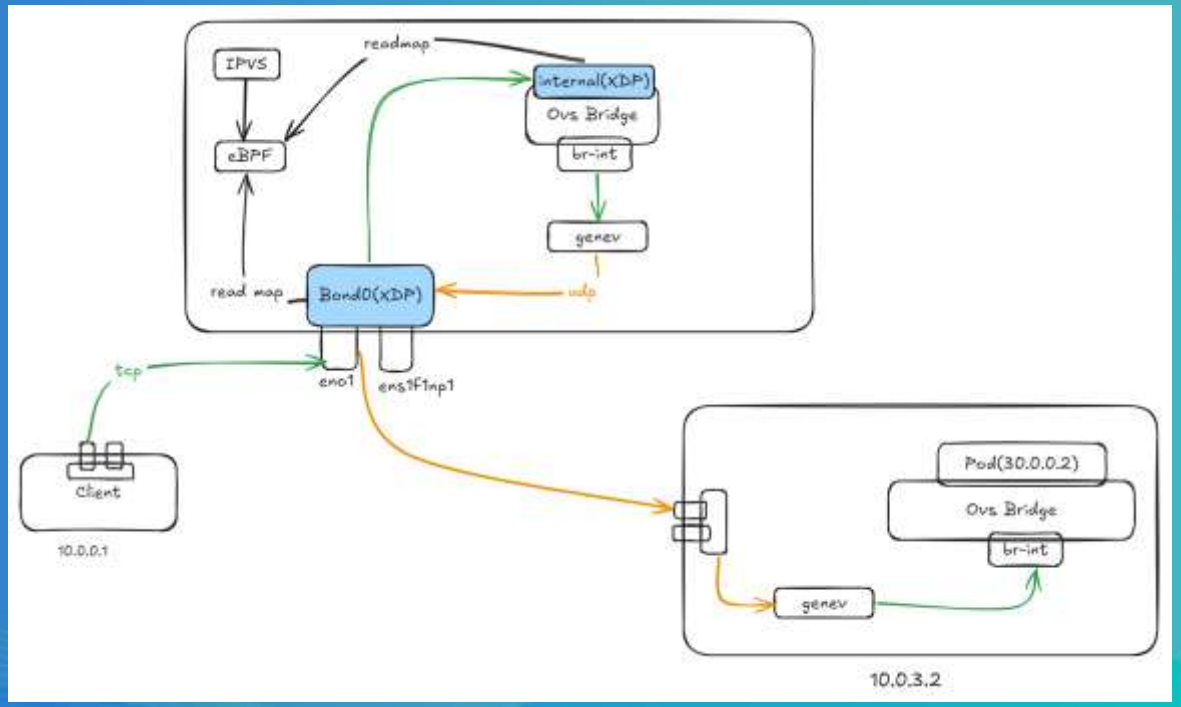
1. Offload XDP 需特定网卡支持（网卡成本较高）
2. **Native XDP 驱动支持（当前物理机NIC均支持）**
3. Generic XDP 系统支持（无限制，性能有限）

XDP网络加速方案

初始环境



XDP加速环境



支持OVS网卡动态加速

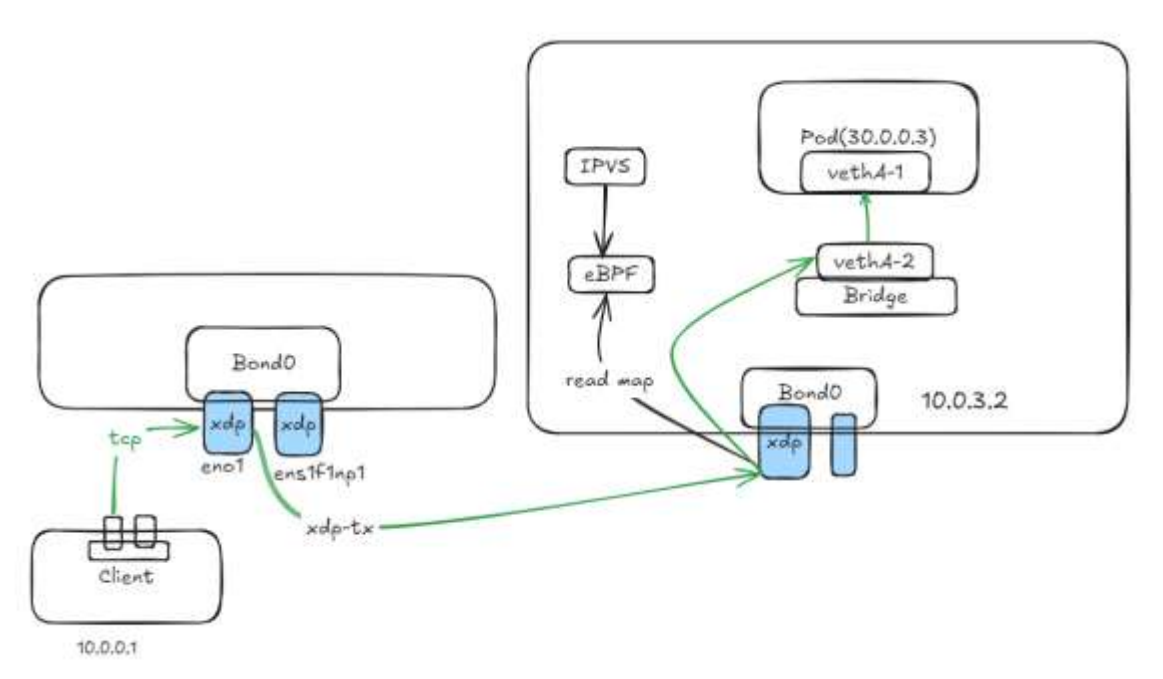
支持实时同步conntrack table数据

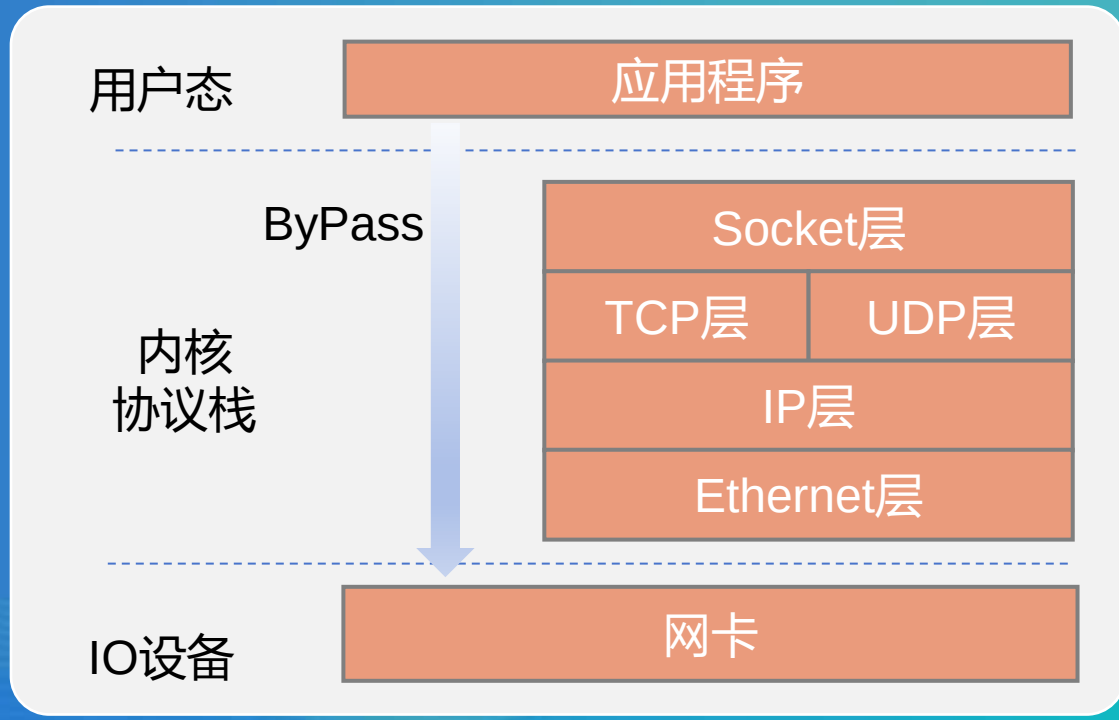
支持多XDP程序共享内存

```
- originalTo:
    ip: 10.0.3.2
    ifName: bond0.1234
    attachMode: generic|native
...
```

3.未来演进方向

XDP技术演进





网络专用机场景网络加速：

1. SLB主机使用XDP-TX进行数据包转发
2. XDP-REDIRECT 后置，业务主机进行转发
3. 所有网卡支持NATIVE-XDP

非网络专用机场景网络加速：

1. 通过AF-XDP技术使应用进程绕过协议栈，实现加速
2. 采用LD_PRELOAD 技术，实现用户进程glibc拦截，实现无侵入完成AF-XDP 加速

谢谢