第二届 eBPF开发者大会

www.ebpftravel.com

基于eBPF可观测性落地实践

中国联通软件研究院--温怀湘

中国·西安

可观测性演进路线

应用架构演进

可观测性演进

监控演进过程

ZABBIX

运维工具

Grafana

Prometheus

PINPOINT

Stack

ClickHouse

fluentbit

eBPF

VICTORIA METRICS

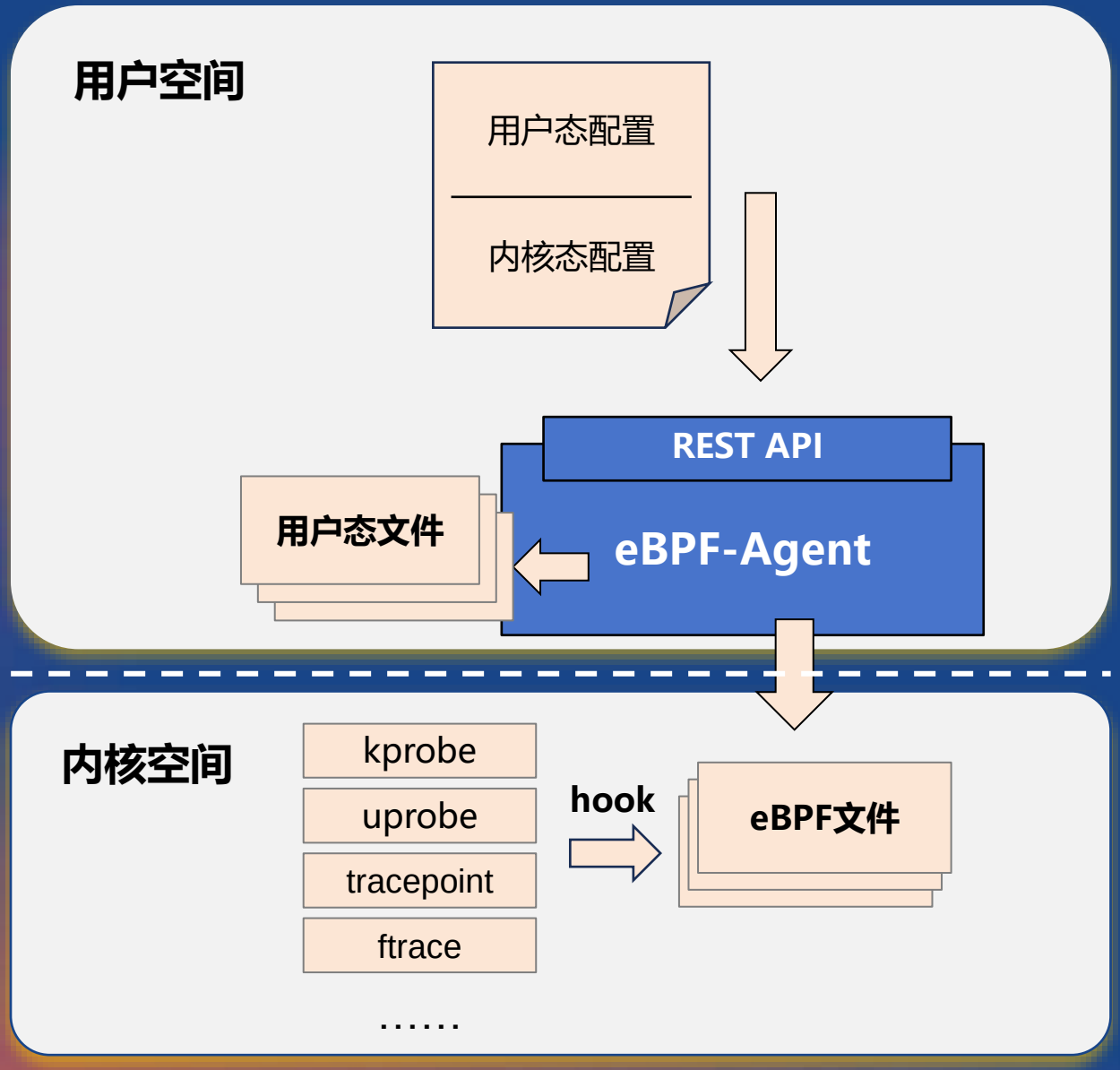
OpenTelemetry

eBPF可观测性落地实践

演进时需要解决的问题

- 新特性可以快速加载，业务受影响快速卸载
- 专注于业务开发，提升开发效率
- 计算、网络、IO场景应用

用户态和内核态程序加载




```
programs:
  - name: tcp_connlat
    path: file:///xxx/tcp_connlat.bpf.o
    type: ebpf_object
  - name: tcp_connlat_user
    path: http://xxxx//tcp_connlat.so
    type: user_so
    properties:
      initfile:
      functions:
        - name: process
          type: default
 .........
```

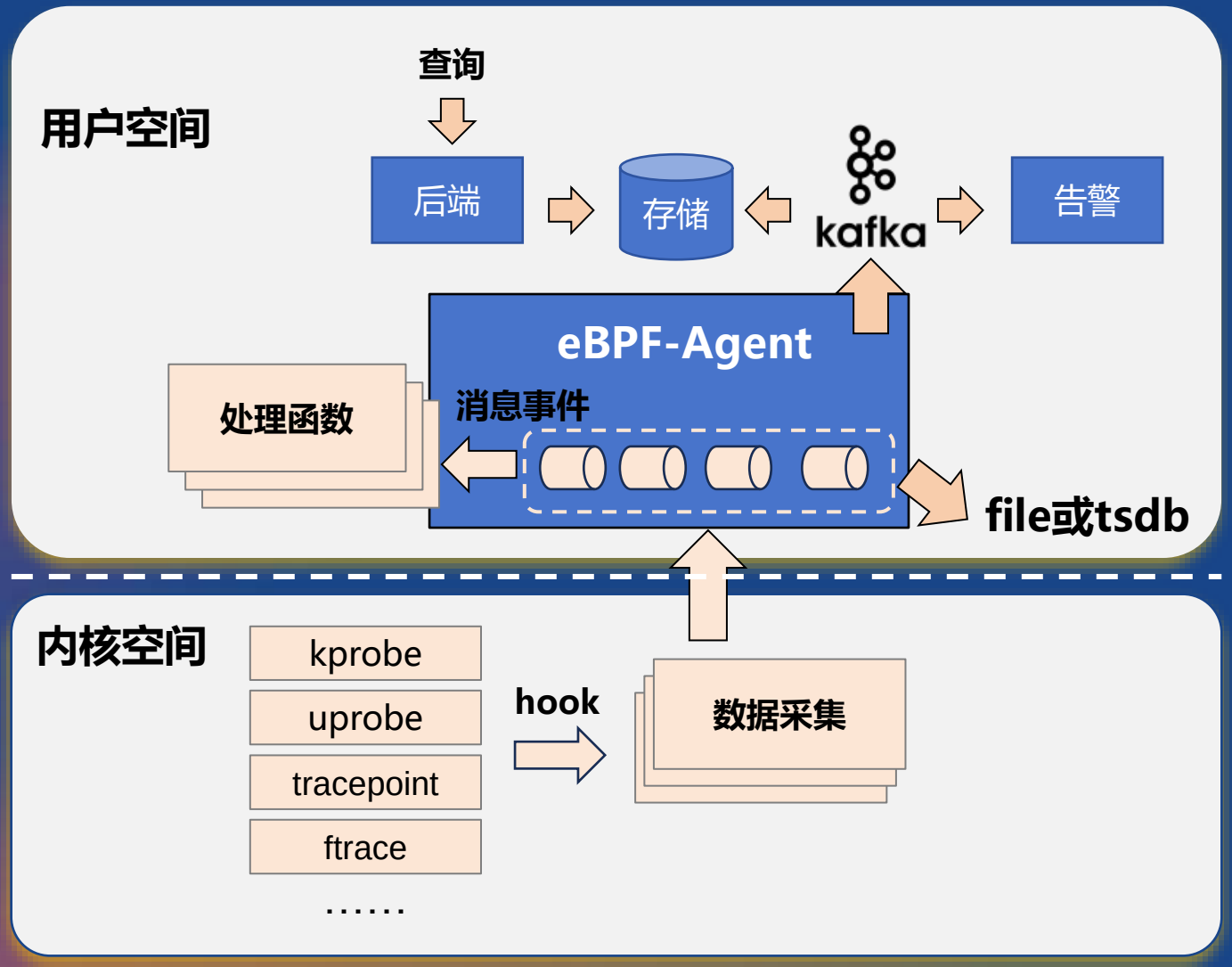
支持远程和本地加载

支持默认和声明解析

支持.so链接库函数

支持用户函数初始化、调用前/后处理

数据采集与消息处理




```
channels:
  tcp_connlat_events:
    type: ring_buffer
    events:
      - name: tcp_connlat_user
        hash: hash
        target_processor: process
        storage:
          kafka: 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

内核函数、用户函数、配置示例

```
struct
{
    __uint(type, BPF_MAP_TYPE_RINGBUF);
    __uint(max_entries, 1 << 20);
} tcp_connlat_events SEC(".maps");

struct tcp_connlat_event
{
    char comm[TASK_COMM_LEN];
    pid_t pid;
    __u16 sport;
    __u16 dport;
   .......
    __u16 padding1;
    __u32 padding2;
};
SEC("tp/sock/inet_sock_set_state")
int inet_sock_set_state(struct trace ....)
{
    __u16 family;
    __s32 newstate;
    __u64 delta_time;
    struct sock *sk;
    struct info_t *infop, info = {};
    struct tcp_connlat_event *event;
    ....
    bpf_ringbuf_submit(event, 0);

}
```

内核C代码

```
type TracerCtx struct {
                CRingCtx uintptr
                CResCtx  uintptr
                ....
                RingCtx RingCtx
                ResCtx  ResCtx
                }

type tcpConnlatEvent struct {
                Comm       [16]uint8
                Pid        uint32
                .....
                Delta_time uint64
}
func process(ctx uintptr) {
    var event tcpConnlatEvent
    var saddr, daddr string
    traceCtx := (*TracerCtx)(unsafe.Pointer(ctx))
    err :=     binary.Read(bytes.NewBuffer(
                traceCtx.RingCtx.Data),
                binary.LittleEndian, &event
                )
     ....
    result := map[string]any{
              "pid":       event.Pid,
              ....
          }
    copy(traceCtx.ResCtx.Data, resData[:])
}
```

用户函数Go代码

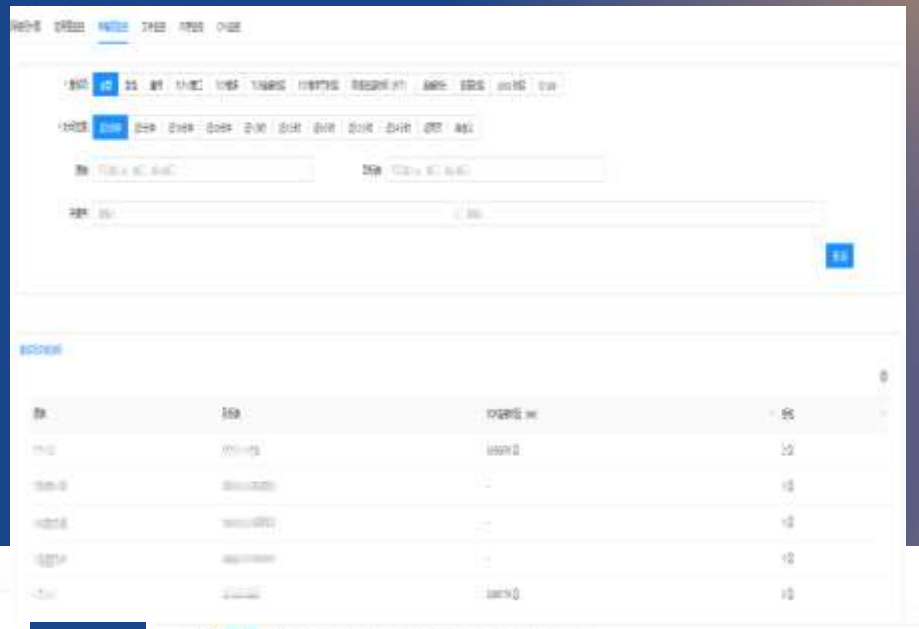
```
kind: ebpfprogram
name: tcp_connlat
description: tcp_connlat
version: "0.1"
author: ebpfprogram
spec:
 programs:
   - name: tcp_connlat
     path: file:///xxx/tcp_connlat.bpf.o
     type: ebpf_object
   - name: tcp_connlat_user
     path: http://xxxx//tcp_connlat.so
     type: user_so
     properties:
       initfile:
       functions:
         - name: process
           type: default
  channels:
    tcp_connlat_events:
      type: ring_buffer
      events:
        - name: tcp_connlat_user
          hash: hash
          target_processor: process
          storage:
            kafka: true
```

装卸Yaml配置

网络特性支持

网络

- 拓扑
- 丢包
- 连接时延
- 重传
- RTT
- 协议解析
  - ✓ http
  - ✓ mongo



其他特性支持

文件

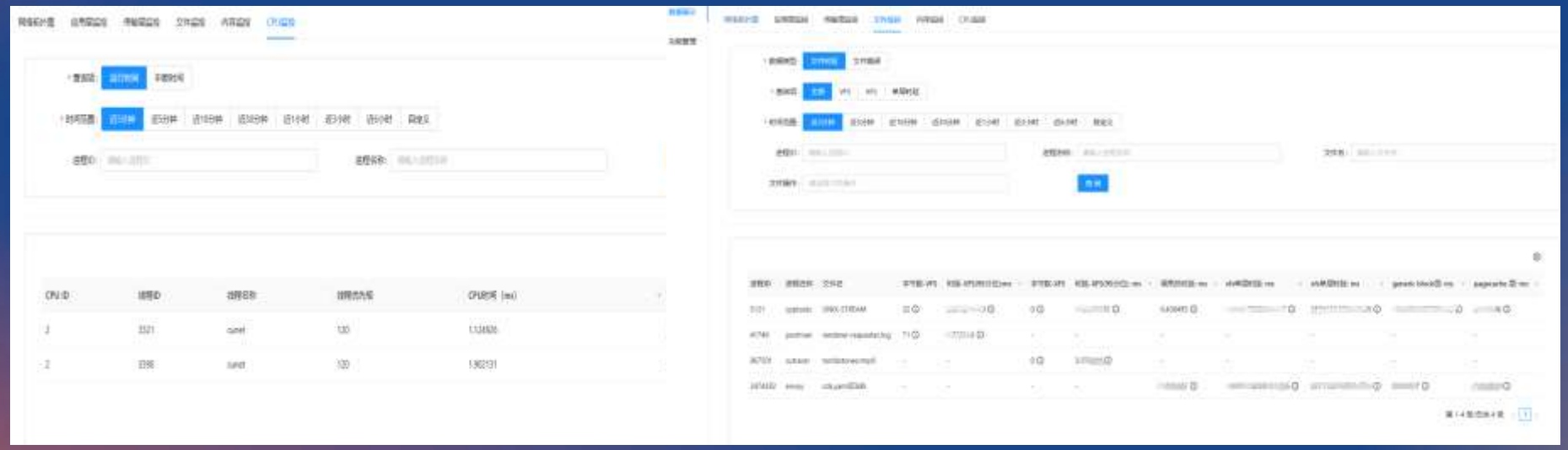
- 读写时延
- 读写错误
- 锁

CPU

- 唤醒时延
- 状态切换
- 队列长度
- 火焰图

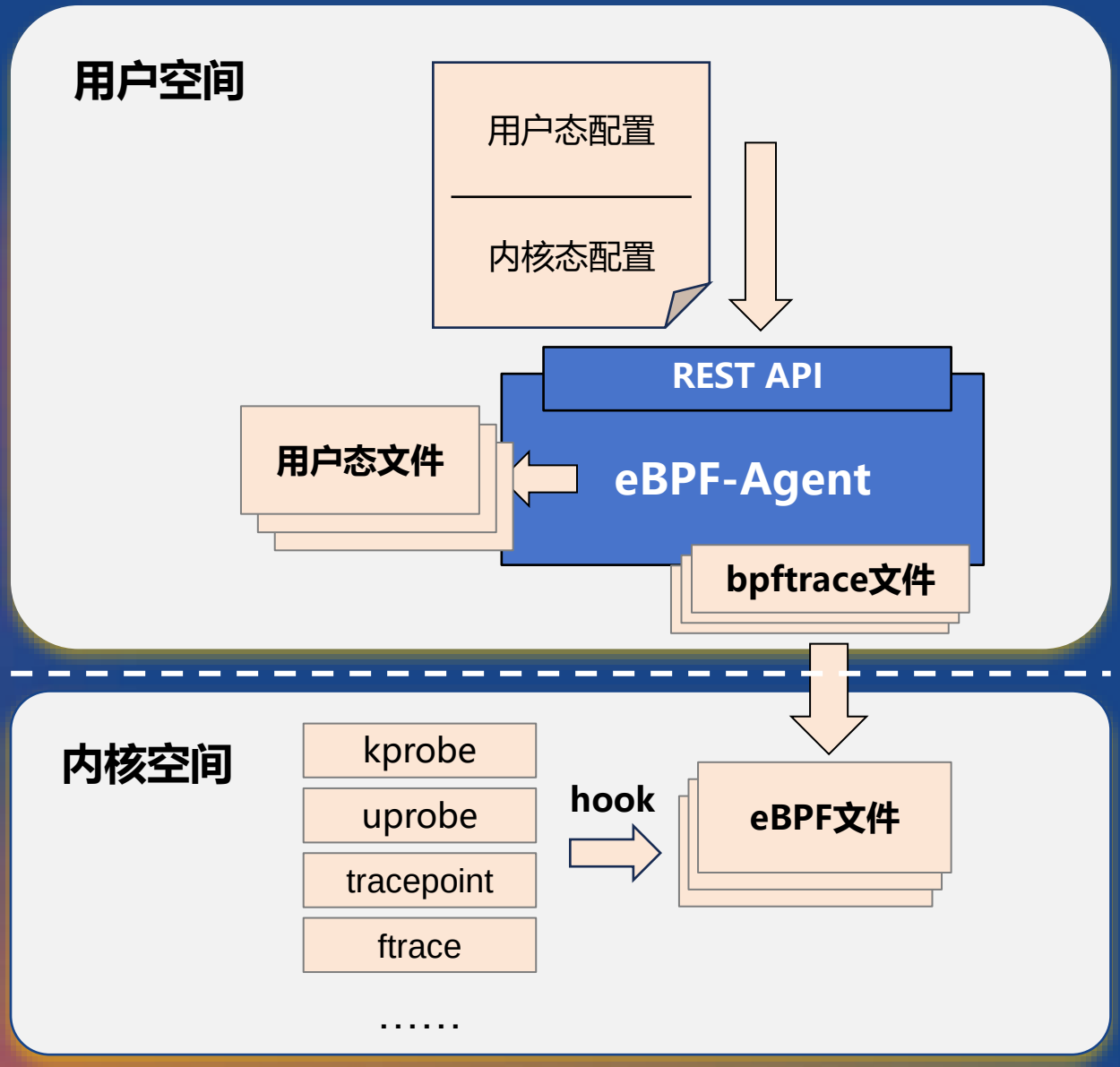
内存

- 内存溢出
- 缺页错误
- 内存泄漏



eBPF演进路线

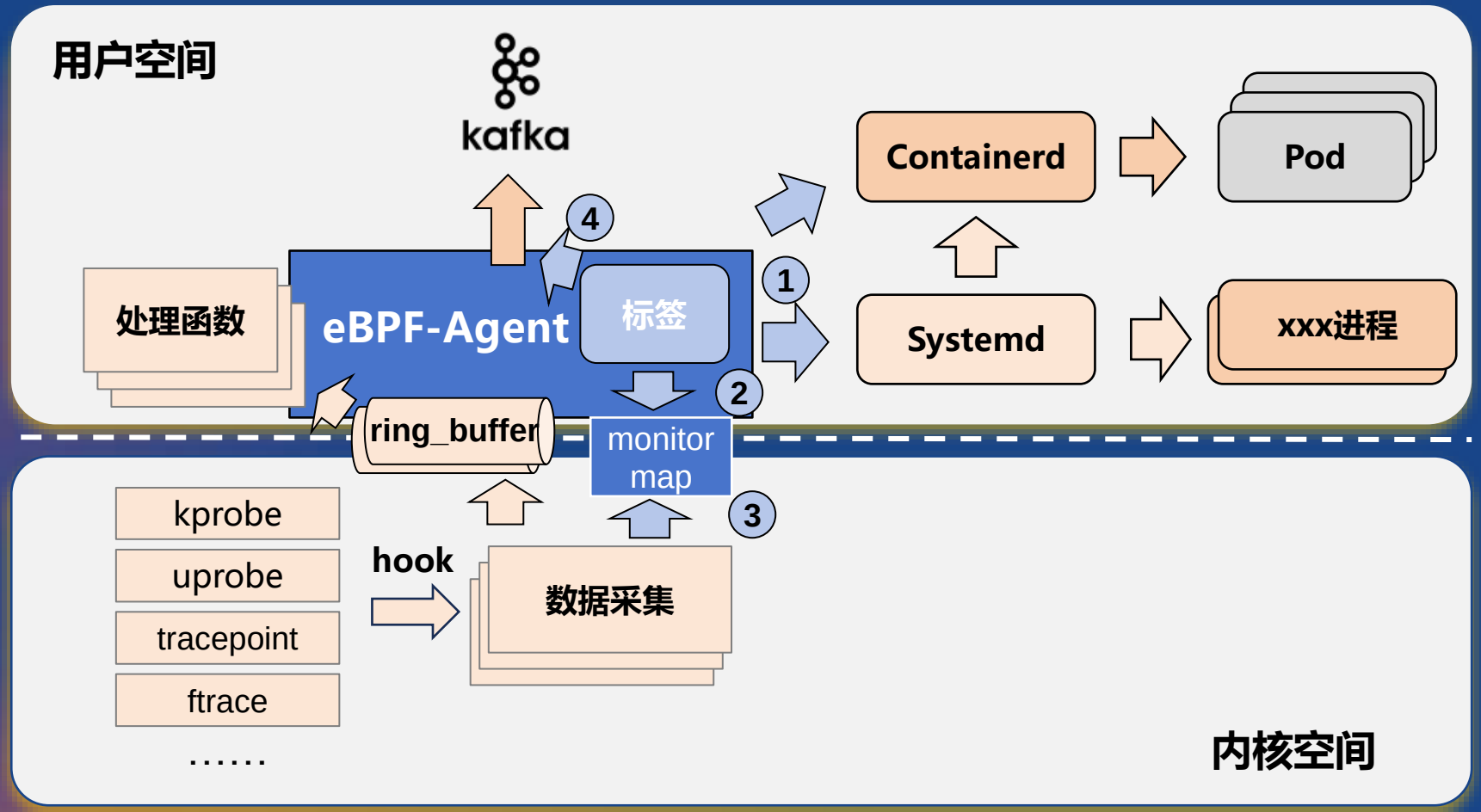
演进方向-支持bpftrace文件




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

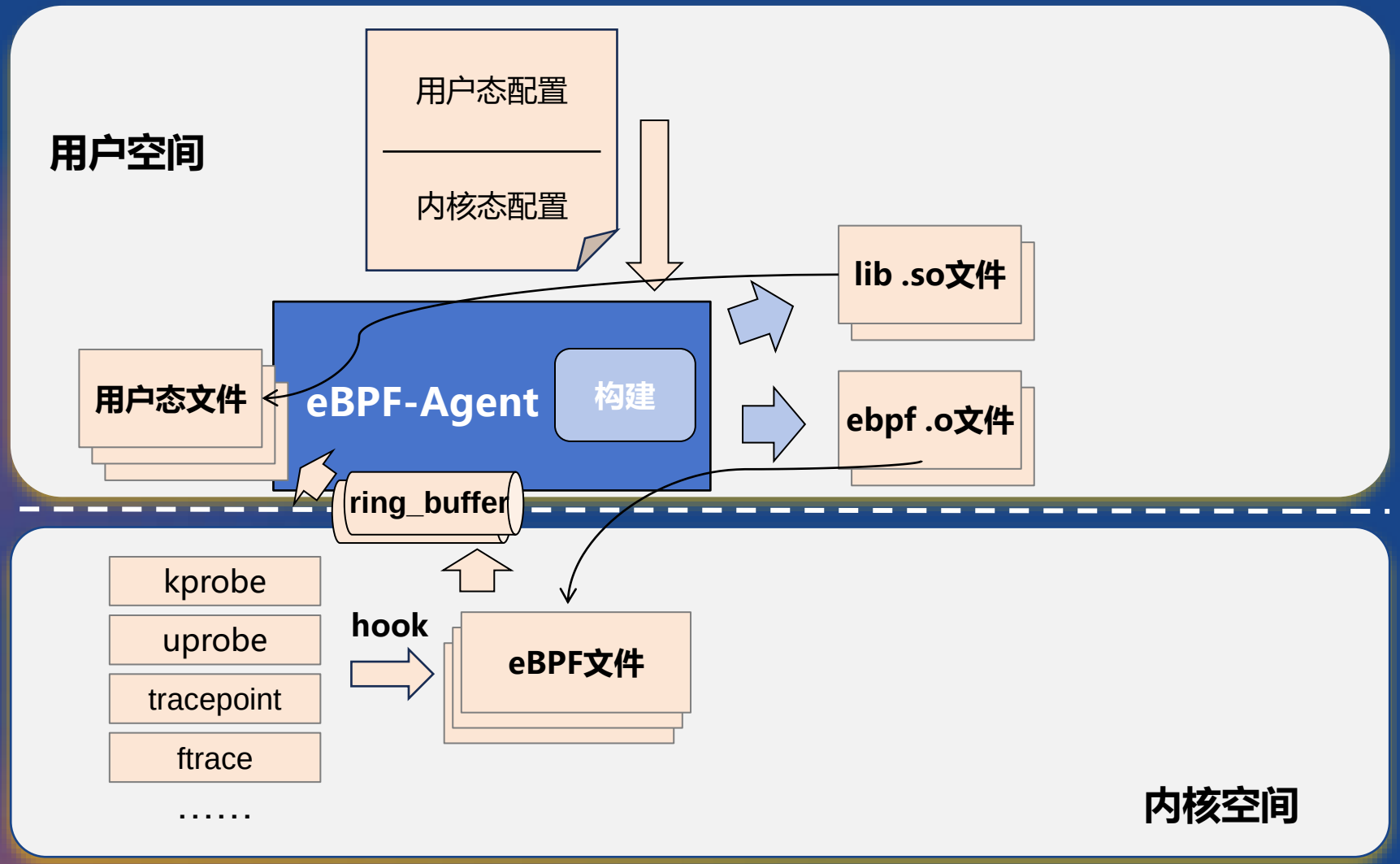
```
openfile.bt
tracepoint:syscalls:sys_enter_open
{
printf("%s %s\n", comm, str(args->filename));
}
```

演进方向-支持进程过滤与业务标识



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

演进方向-支持源码文件装卸



1. 用户态和内核态配置文件是源码文件
2. 根据源码文件在主机上进行源码编译，生成本地.o和.so文件
3. 加载.o和.so文件

演进方向-支持网络更多特性

协议解析

gRPC DNS TLS SSL

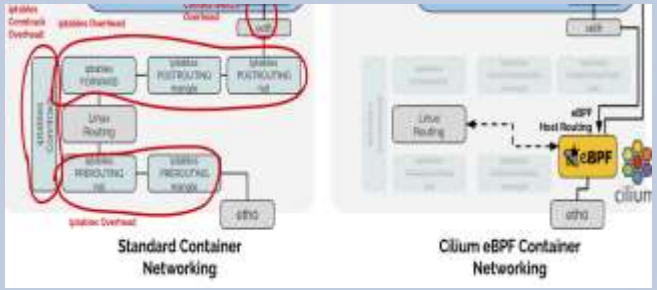
PostgreSQL redis MySQL

节点之间协议

kafka MongoDB.

更多特性

绕过内核函数实现性能提升



支持IP及以下层数据解析

IPVS

谢谢