通过Golang + eBPF实现无侵入应用可观测

×

张海彬

阿里云
应用可观测技术专家

目录

第一部分

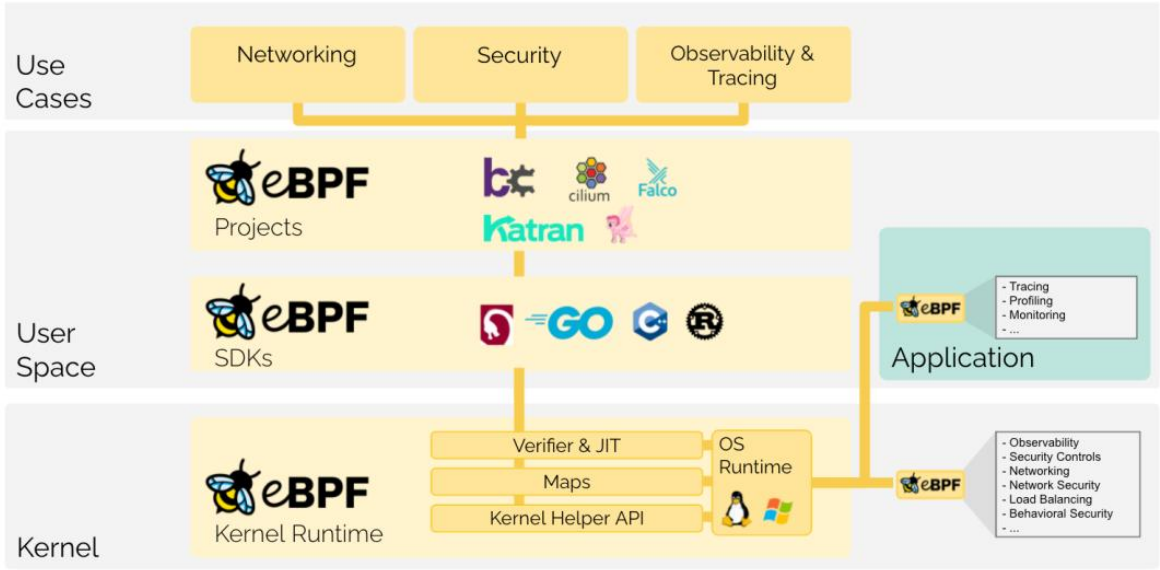
eBPF简介

eBPF简介

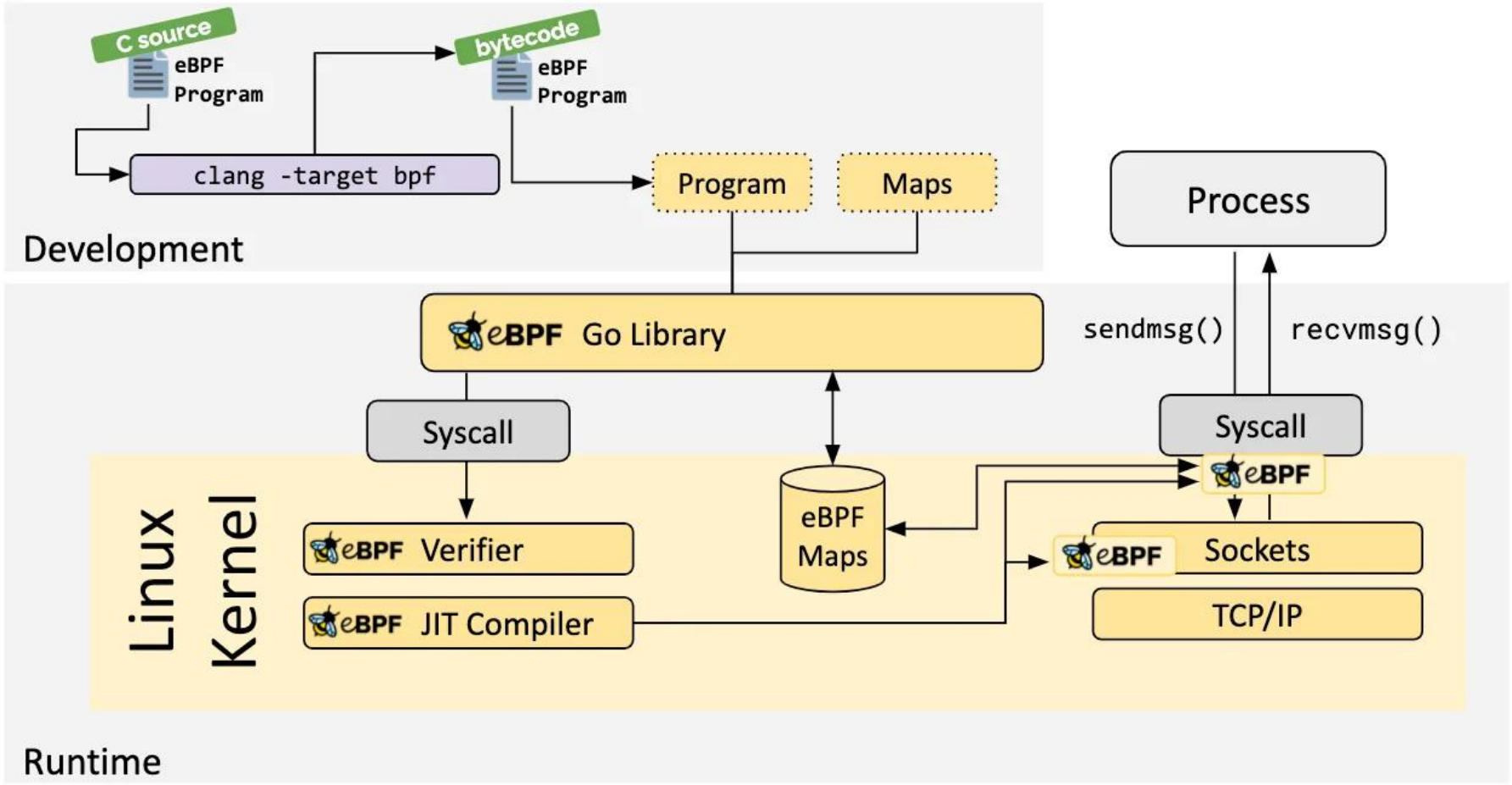
eBPF = extended Berkeley Packet Filter

Dynamically program the kernel for efficient networking, observability, tracing, and security.

- 稳定
- 高性能
- 安全（内核verifier机制）
- 动态可编程（无需重启）

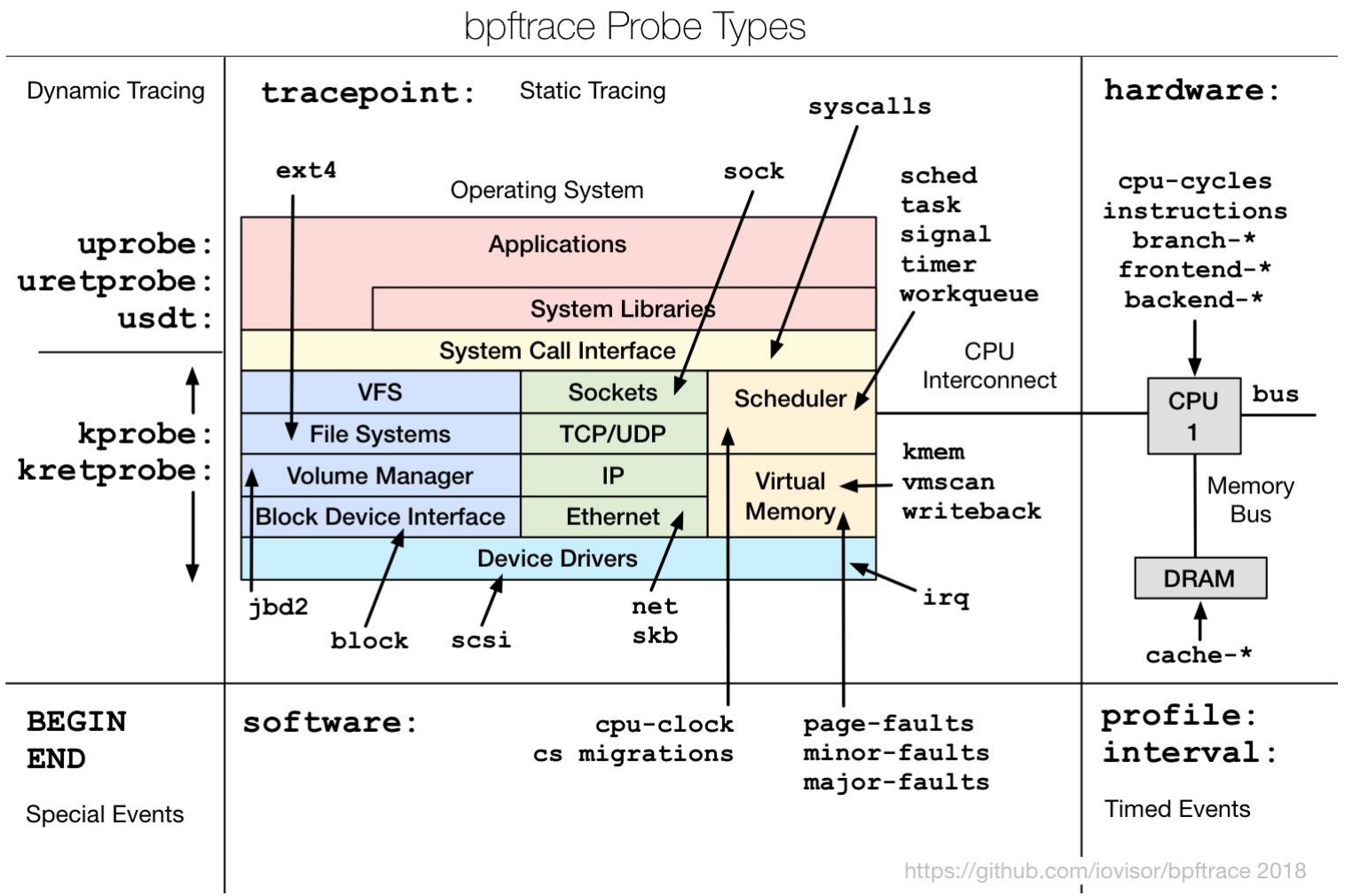


eBPF程序加载和校验



eBPF事件驱动

Kprobe/Kretprobe
Uprobe/Uretprobe
XDP
Tracepoint
Perf

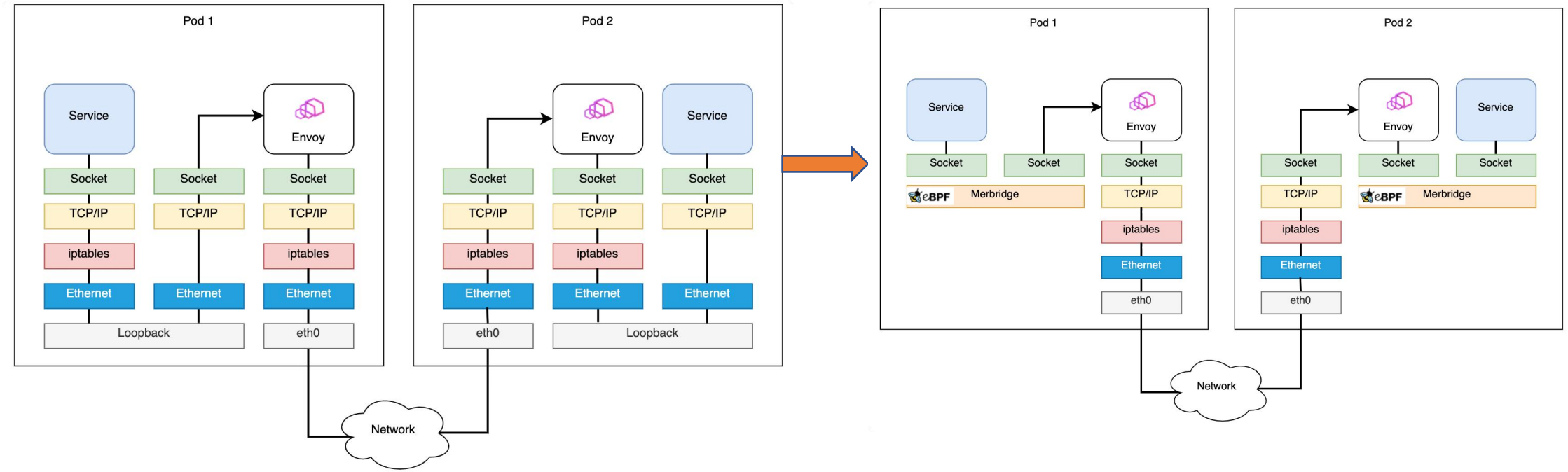


第二部分

eBPF在云原生场景下的应用

网络加速

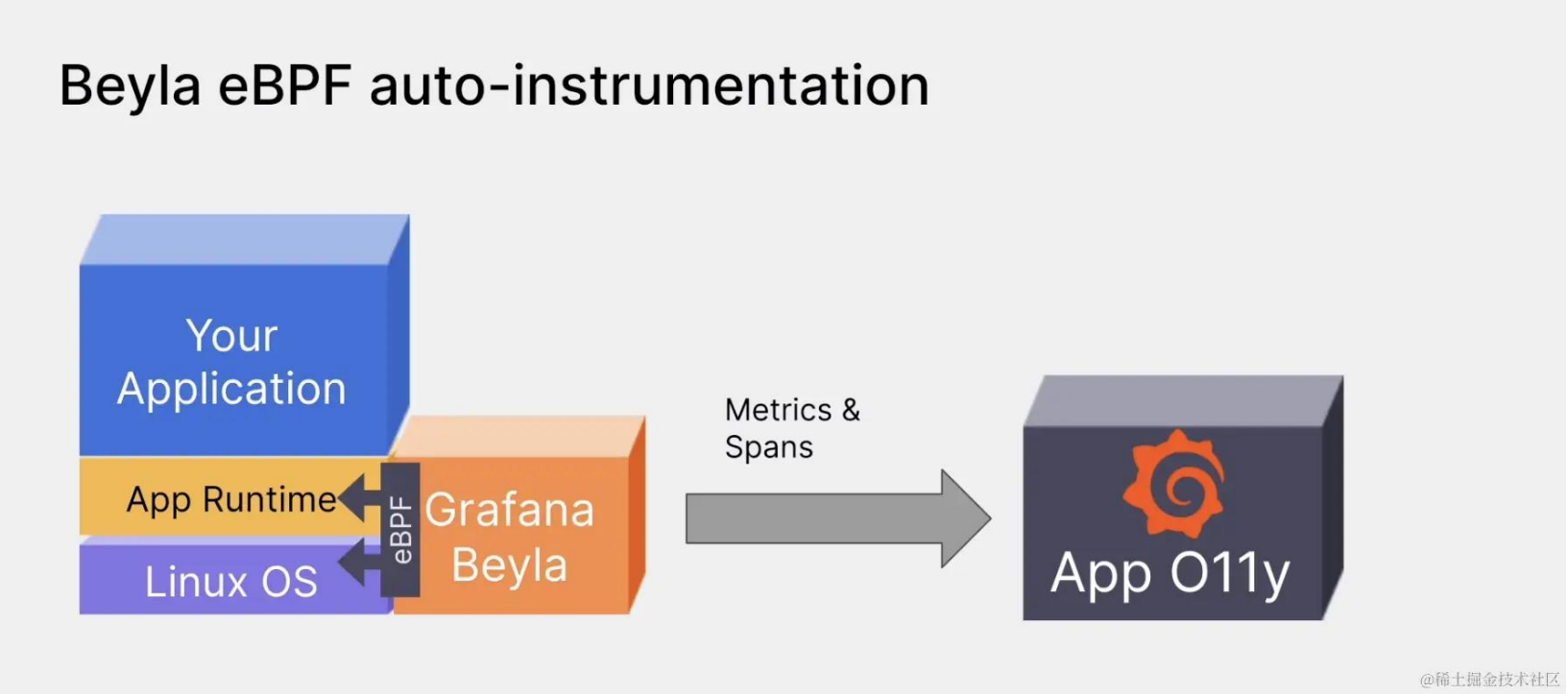
eBPF 的可编程能力使其能够内核中完成包的处理和转发，而且可以添加额外扩展能力。



From：https://istio.io/latest/zh/blog/2022/merbridge

观测和跟踪

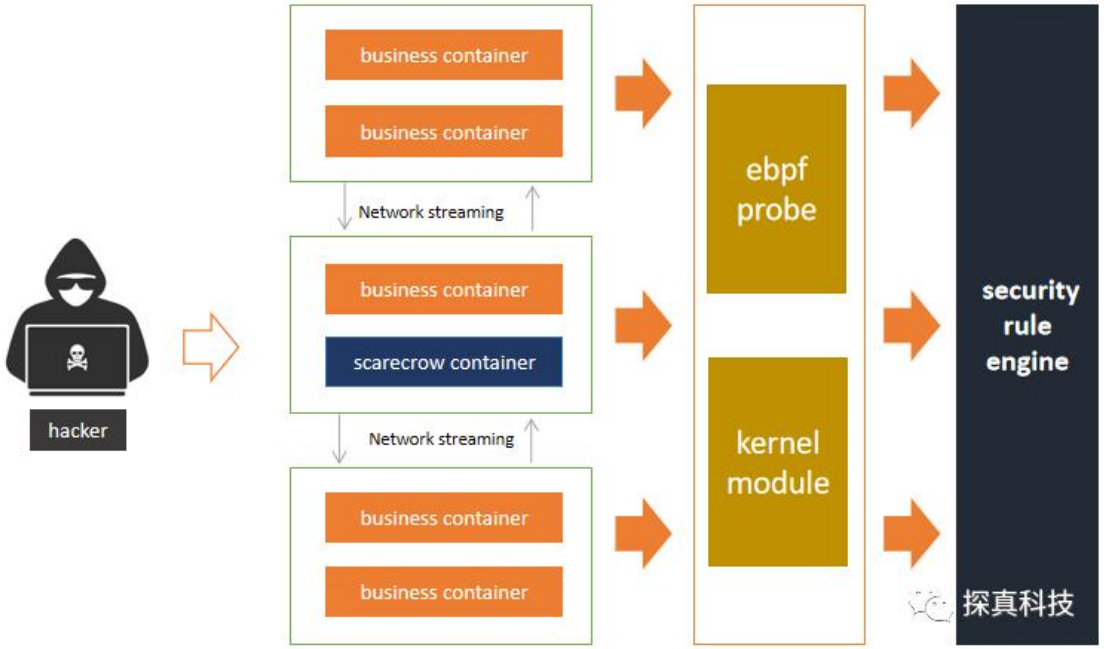
将 eBPF 程序附加到跟踪点以及内核和用户应用探针点的能力，使得应用程序和系统本身的运行时行为具有前所未有的可见性



From：https：//juejin.cn/post/7280746515525156918

安全

看到和理解所有系统调用的基础上，将其与所有网络操作的数据包和套接字级视图相结合，通过检测来阻止恶意攻击行为，如 DDoS攻击等，实施网络策略、增强系统的安全性、稳定性。

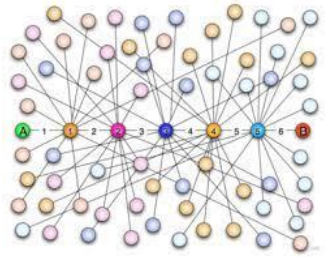


From：https：//zhuanlan.zhihu.com/p/507388164

第三部分

微服务可观测的挑战

微服务可观测的挑战



DUBBO kafka

MySQL gRPC

HTTP redis

应用：微服务架构、多语言、多协议

挑战1：微服务、多语言、多协议环境下，端到端观测复杂度上升，埋点成本居高不下

- Kubernetes
- 容器
- 网络、操作系统、硬件

如何关联？

基础设施层复杂度日益增加

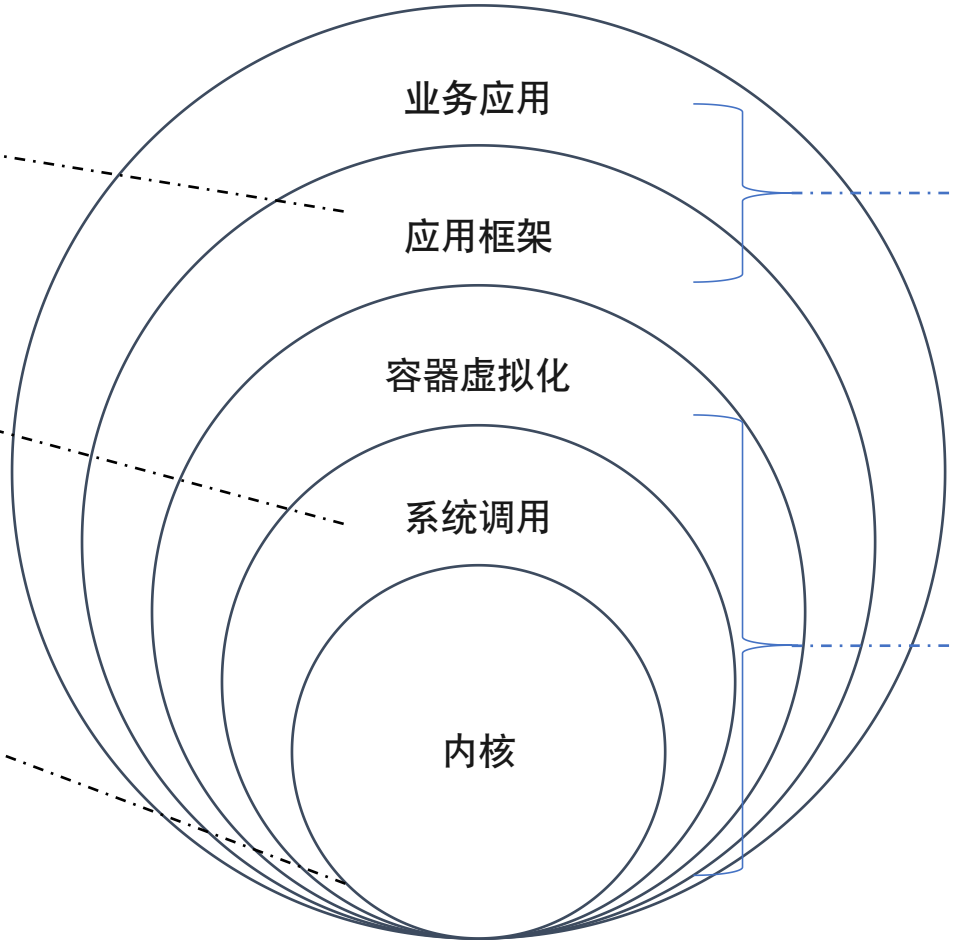
挑战3：数据散落，工具多，缺少上下文，排查效率低下

Kubernetes下的可观测

Kubernetes组件异常：
Scheduler, KCM，
etcd，api-server，
coredns…

系统调用异常：网络请求，内存申请，文件操作，CGroup…

内核异常：进程调度，内存管理，文件管理，夯机宕机，资源异常…



应用组件异常：线程池满，数据库连接无法获取，OOM，文件读取错误…

应用性能监控（APM）

Kubernetes监控

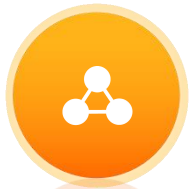
无法自顶向下端到端串联导致棘手问题频发。

第四部分

Golang + eBPF实现数据采集

eBPF在可观测领域的优势

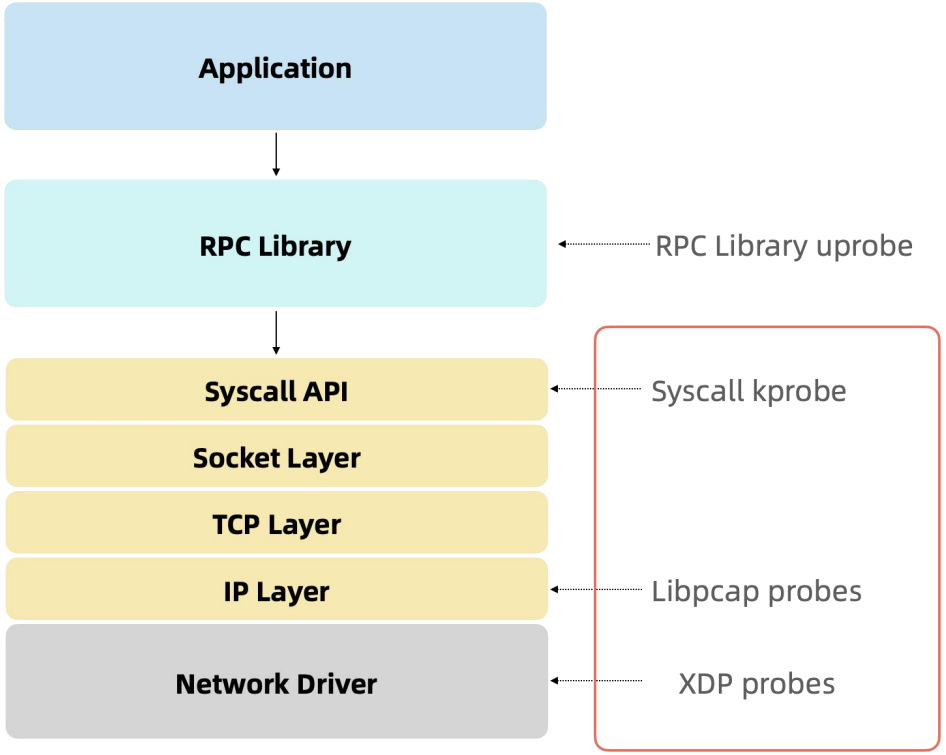
无侵入



多语言/多协议/多框架

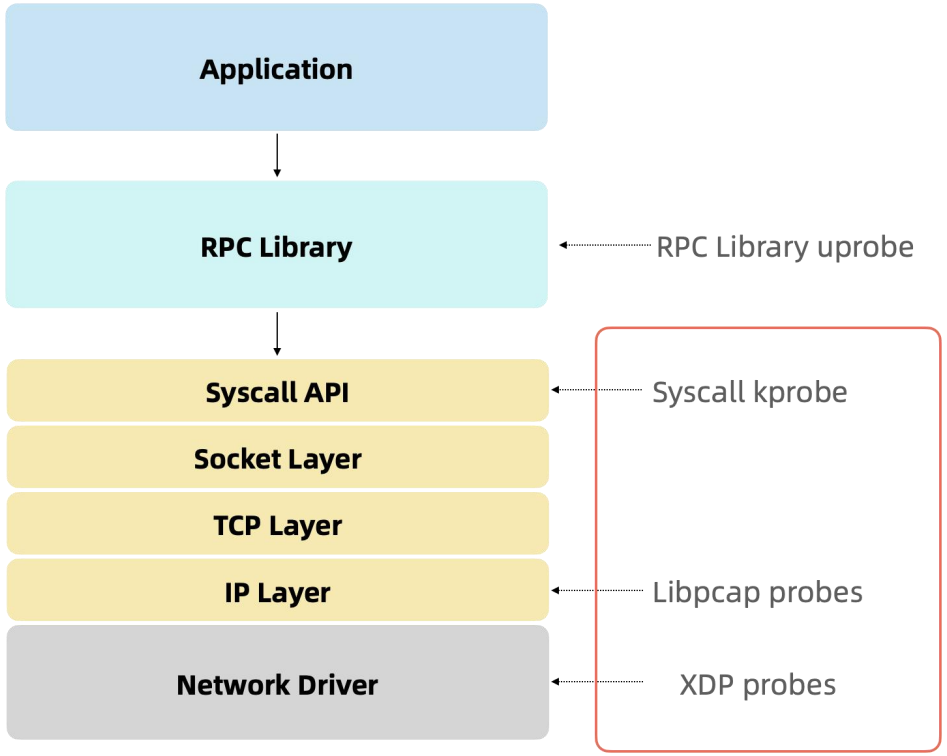
全栈覆盖

无侵入性



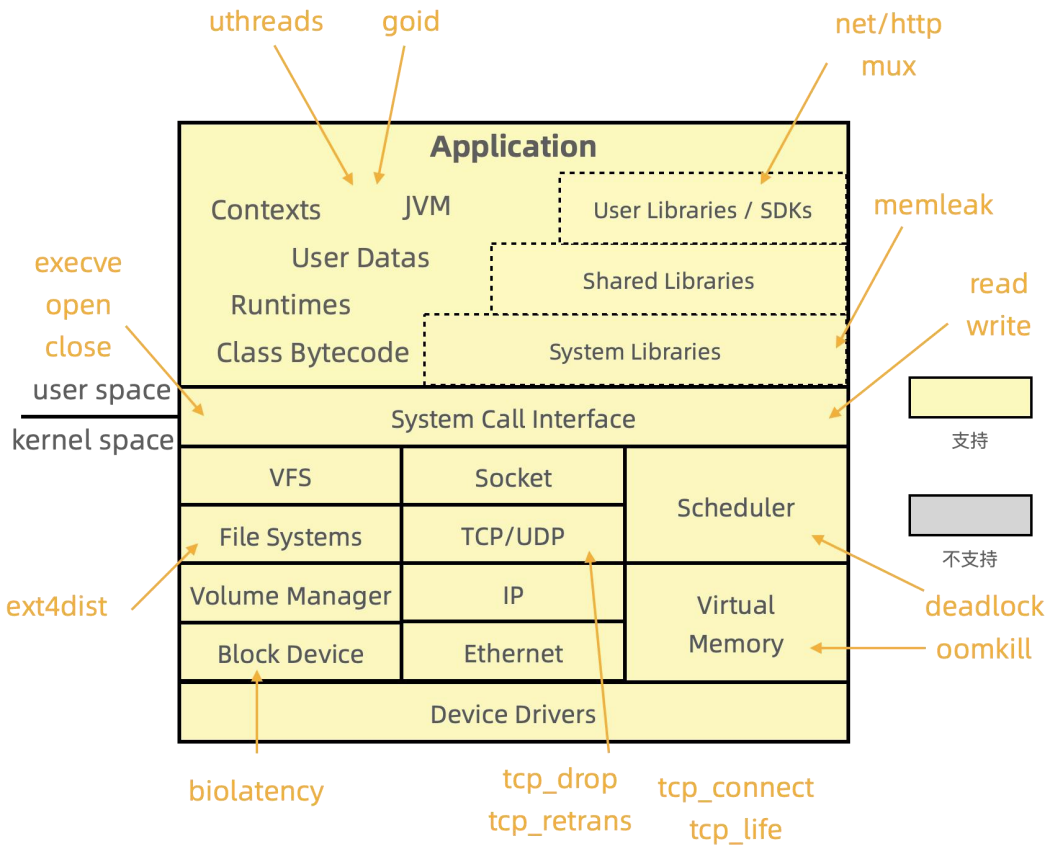
- 无需修改代码
- 无需重启应用
- Verifier保证运行**安全**

多协议、多框架、多语言



- 捕获网络字节流
- 无需适配编程语言
- 无需适配协议框架
- 同时支持用户态插桩

全栈覆盖



uprobe

kprobe

tracepoint

USDT

perf

...

eBPF的编程实践

bcc

- bcc 依靠运行时汇编，将整个大型LLVM/Clang库带入并嵌入其中
- 编译过程中资源用量大，对Cpu、Mem有要求
- 依赖内核的头包

libbpf + bpf + core 编程

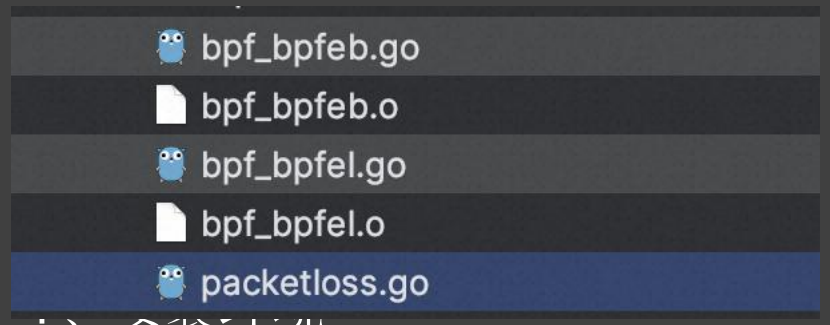
- bpf 程序跟其他的用户空间的程序没有太大区别
- 编译成二进制文件，可以适应不同运行环境
- libbpf 扮演bpf程序装载机角色
- 开发人员只需要关注bpf程序的正确性和性能，不需要关注其他依赖关系

通过Golang加载eBPF程序

```
func loadSync() error {
    // Allow the current process to lock memory for eBPF resources.
    if err := rlimit.RemoveMemlock(); err != nil {
        return fmt.Errorf("remove limit failed: %s", err.Error())
    }
    opts := ebpf.CollectionOptions{}
    opts.Programs = ebpf.ProgramOptions{
        KernelTypes: bpfutil.LoadBTFSpecOrNil(),
    }
    // Load pre-compiled programs and maps into the kernel.
    if err := loadBpfObjects(&objs, &opts); err != nil {
        return fmt.Errorf("loading objects: %s", err.Error())
    }
    pl, err := link.Tracepoint("skb", "kfree_skb", objs.KfreeSkb, &link.TracepointOptions{})
    if err != nil {
        return fmt.Errorf("link tracepoint kfree_skb failed: %s", err.Error())
    }
    links = append(links, pl)
    return nil
}
```

bpf2go

```
//go:generate go run github.com/cilium/ebpf/cmd/bpf2go -cc clang -cflags $BPF_CFLAGS -type insp_pl_event_t -
type insp_pl_metric_t bpf ../../../../bpf/packetloss.c -- -I../../../../bpf/headers -D__TARGET_ARCH_x86
```



2、写好bpf.c和bpf.h,放到指定目录

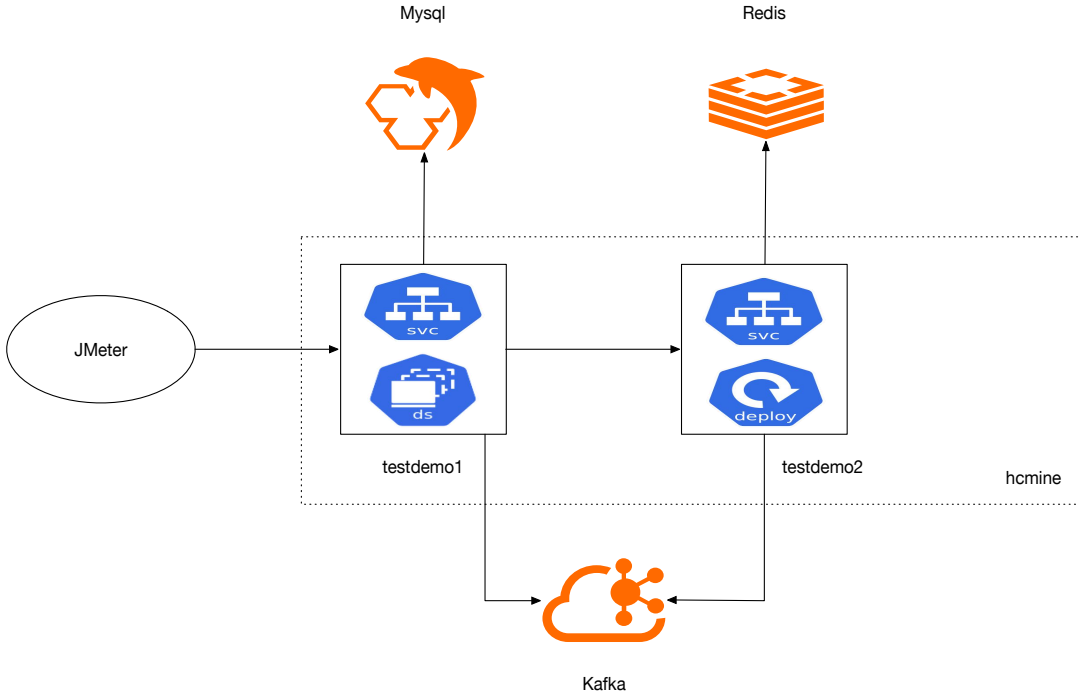
3、go generate 获取转换后的go文件

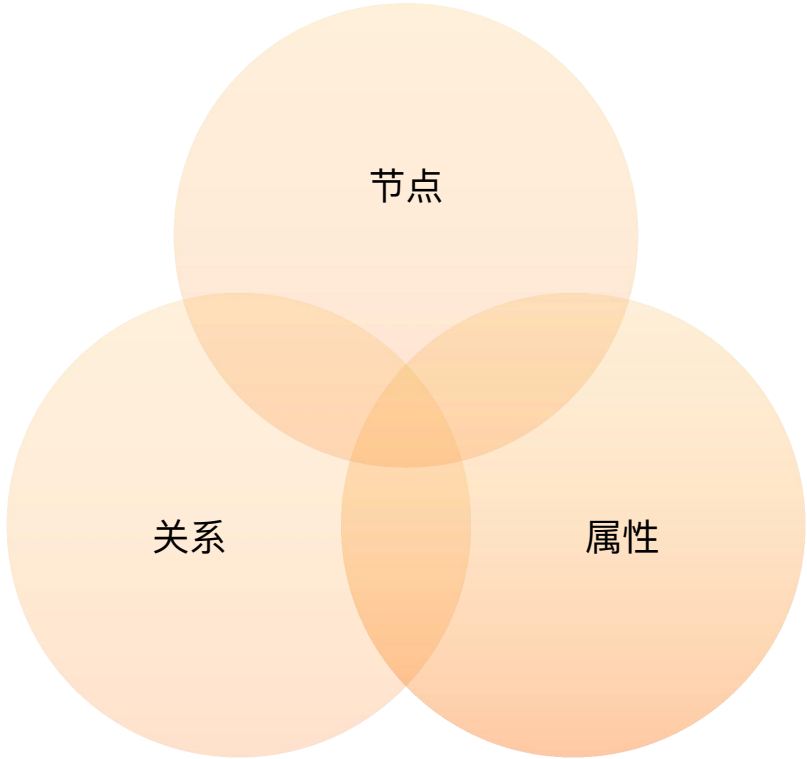
第五部分

构建完整的应用可观测系统

架构感知

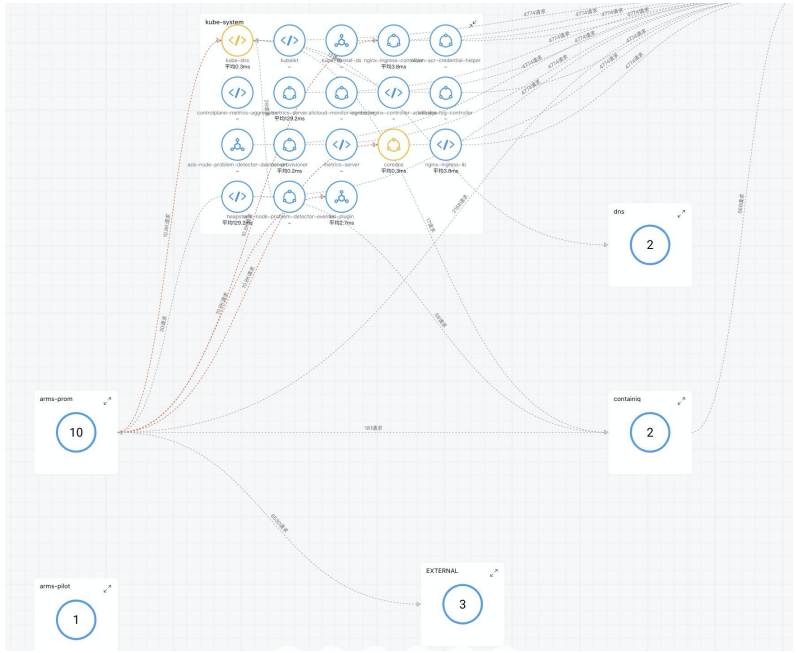
架构感知，节点和关系以及他们的属性，能够正确地反应当前运行的网络关系，帮助用户感知架构，通过对比期望架构，发现问题，通常在新应用上线，新地区开服，整体链路梳理等场景使用。

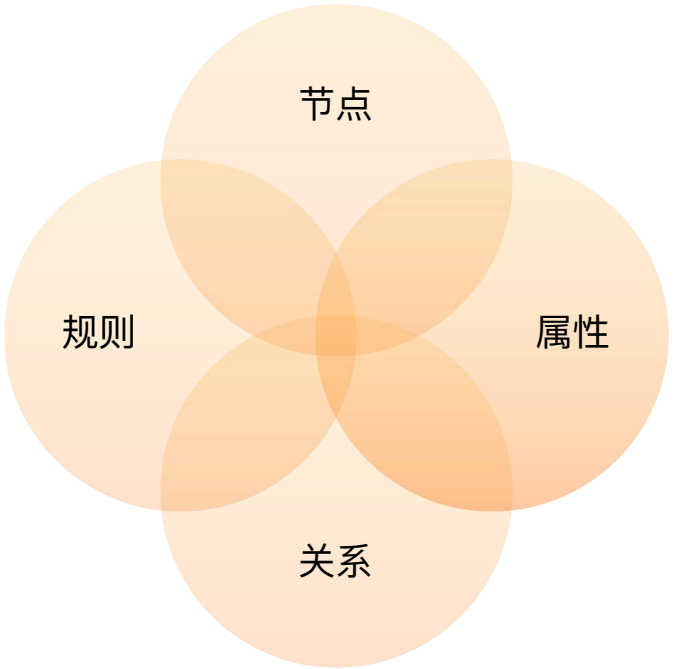




异常发现

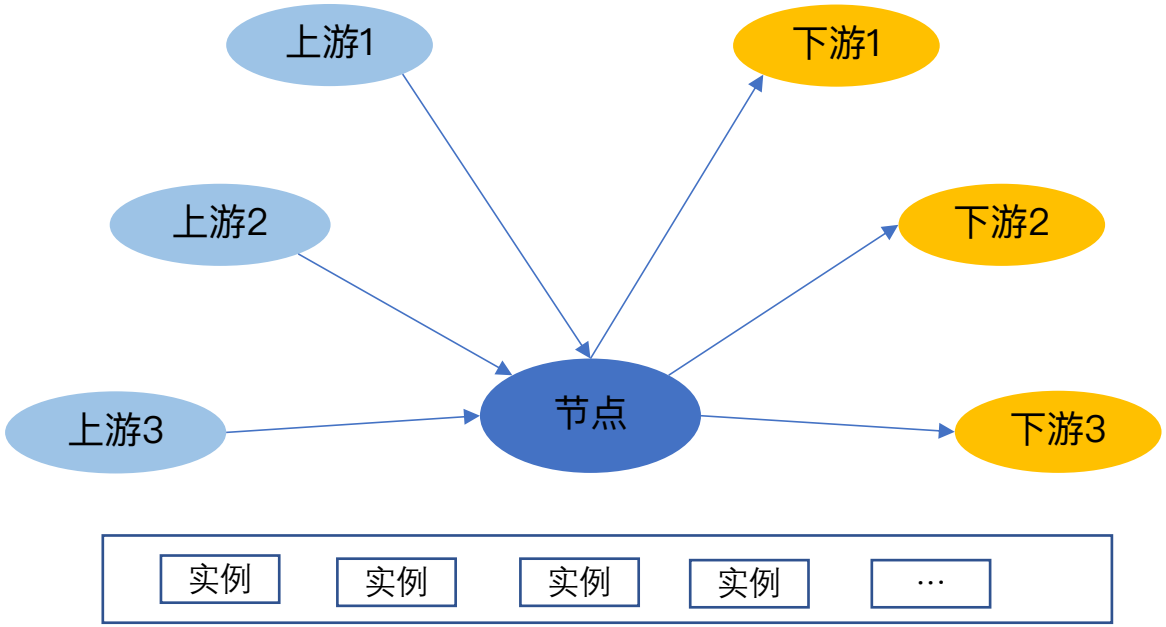
异常发现，通过节点和关系颜色表达，能够快速地发现特点的节点和关系异常，进一步提升问题发现和定位的效率，通常在应用运行时整体链路梳理和特定问题节点上下游分析等场景使用。

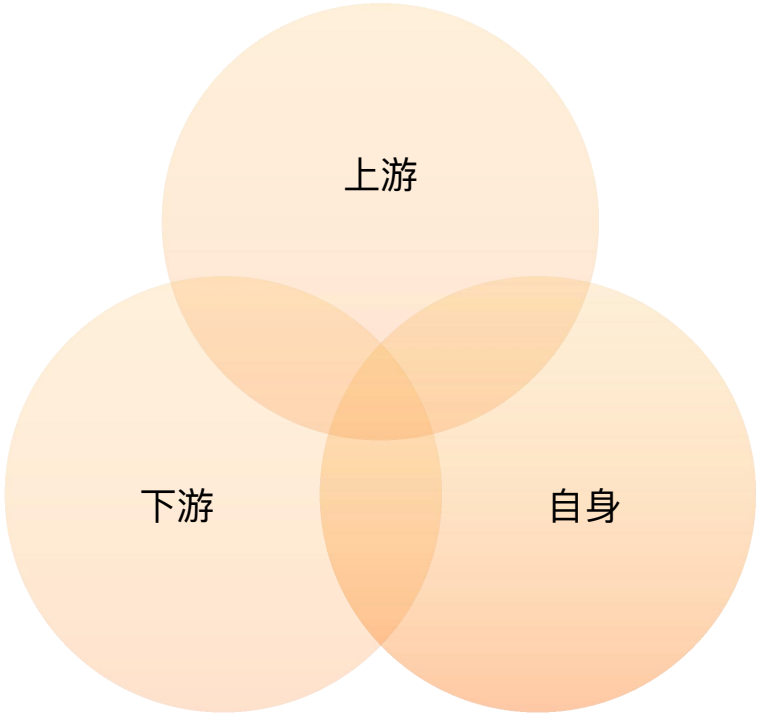




关联分析

关联分析，通过关联关系的切换，可以快速查看上游请求和下游依赖，以及自身服务实例的运行情况，进一步提升问题定位能力，通常在已经定位到某个异常节点后使用。

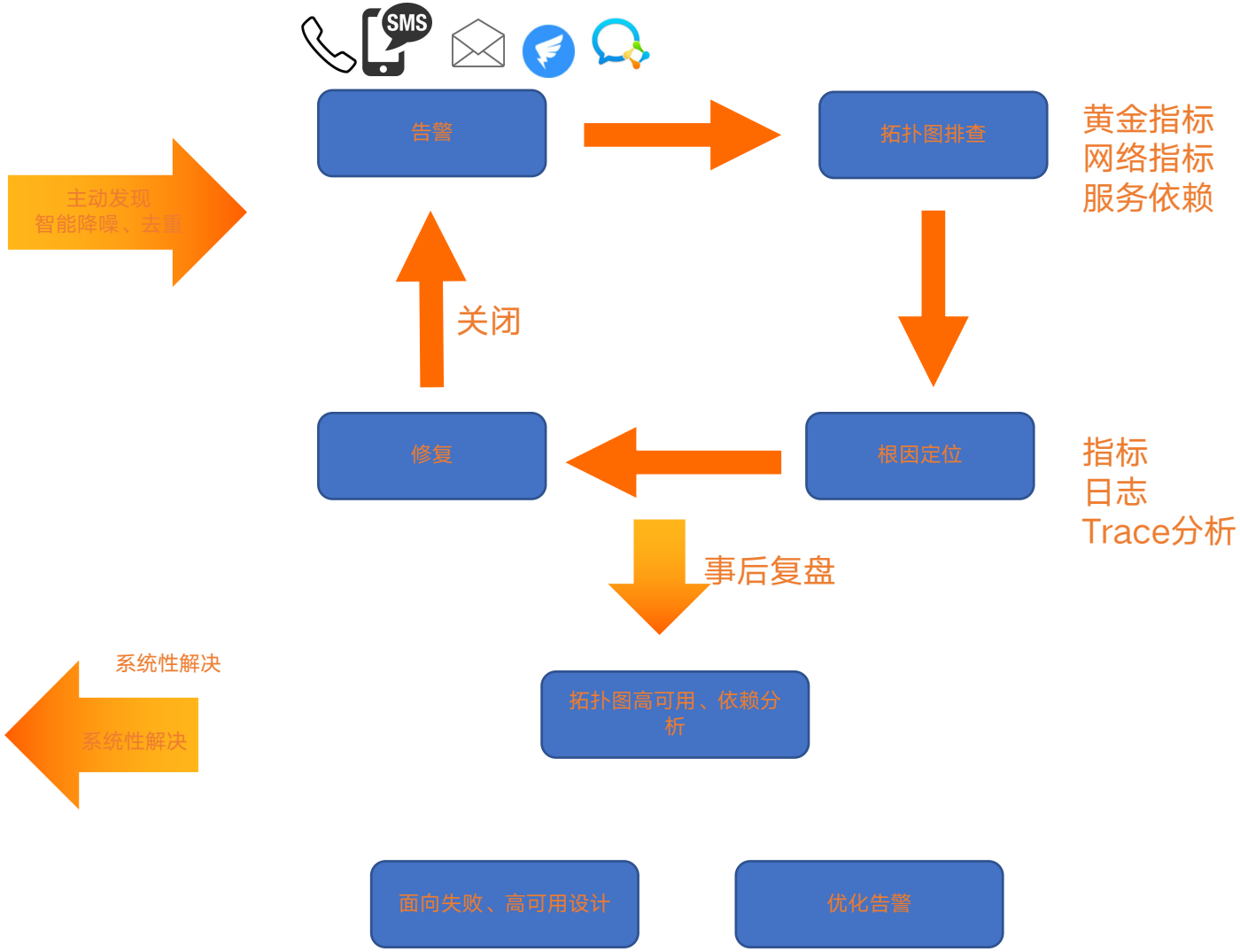




智能告警

全栈数据源，70+个告警模板开箱即用：
应用级别：Pod/Service/Deployment
K8S控制面：apiserver/ETCD/Scheduler
基础设施：节点、网络、存储
云服务界别：Kafka/MySQL/Redis/

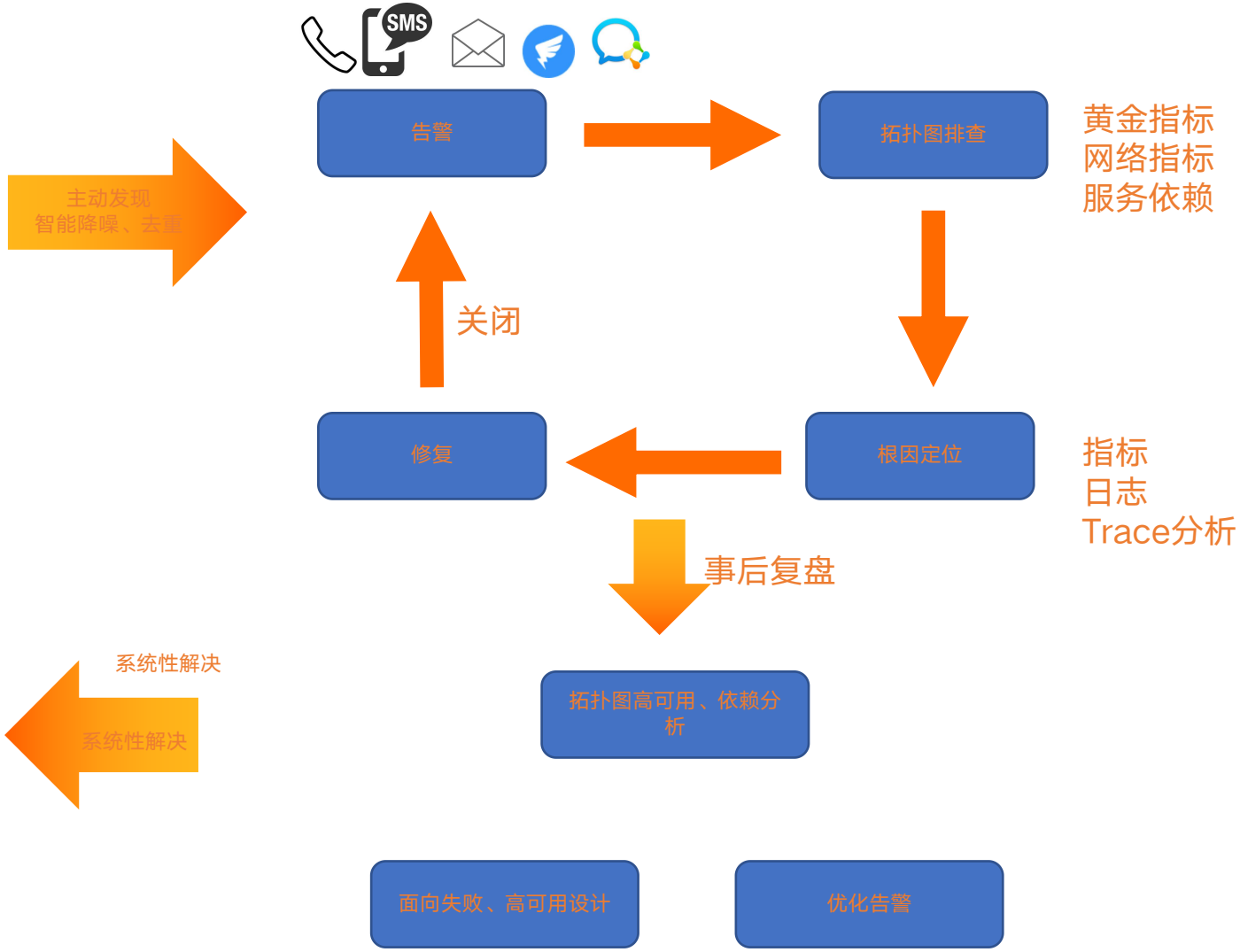
告警收敛，幸福感UP



智能告警

全栈数据源，70+个告警模板开箱即用：
应用级别：Pod/Service/Deployment
K8S控制面：apiserver/ETCD/Scheduler
基础设施：节点、网络、存储
云服务界别：Kafka/MySQL/Redis/

告警收敛，幸福感UP

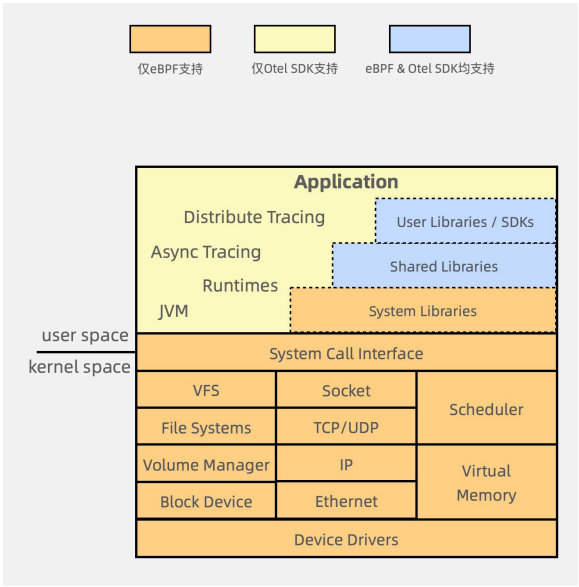


eBPF + Golang 在阿里云应用可观测的实践

应用监控eBPF版

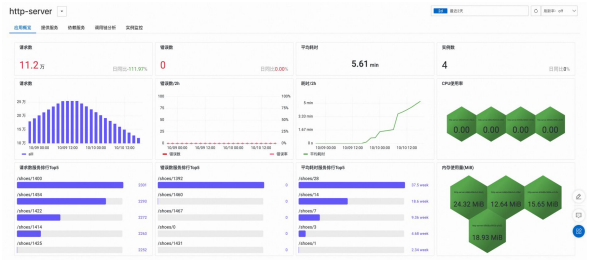
无侵入的应用可观测

eBPF是一种在Linux内核运行的沙盒程序，无需修改任何应用代码，提供无侵入的应用无关、语言无关、框架无关的应用可观测能力，提供如网络、虚拟内存、系统调用等Otel无法获取的数据指标。



新版控制台体验升级

- 黄金三指标
- 调用链查询与分析
- 拓扑/上下游
- 网络大盘
- 容器监控
- 智能告警
- 持续剖析
- 接口监控



what's new:

更标准

更稳定

- eBPF Agent性能提升20%　数据来源
- Otel Collector 性能提升80%

无侵入

- 无需要修改任何业务代码，一键接入eBPF监控
- 语言无关、框架无关、协议无关

异常监控

- 监控网络异常，如TCP Drop、TCP 重传
- 监控应用异常事件，如OOM

持续剖析

- 提供多语言的无侵入的应用CPU热点查看

Thank You, Every Gopher