DeepFlow®

基于 eBPF 的 高度自动化可观测性实践

向阳 @ 云杉网络

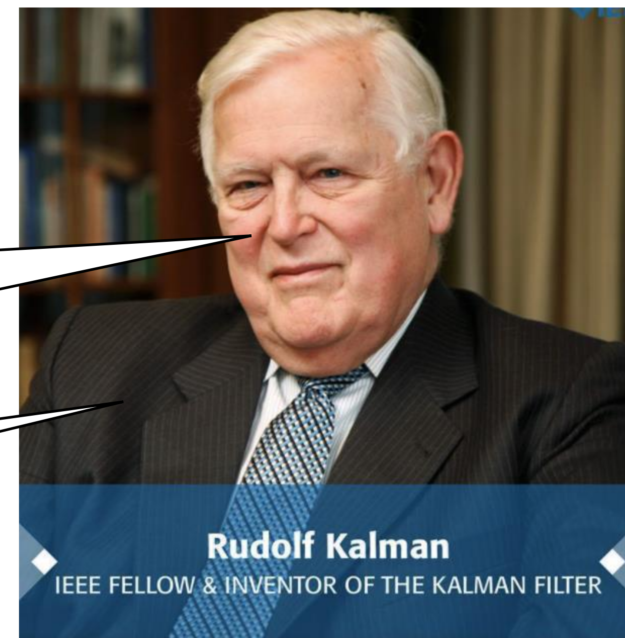
2022-11-12

可观测性：解决复杂系统的可控性问题

I believe we should go to the moon.

控制理论中的可观测性是指：系统可以由其外部输出推断其内部状态的程度。

一系统具有可观测性当且仅当：针对所有的状态向量及控制向量，都可以在有限时间内，只根据输出信号来识别目前的状态。



John F Kennedy
May 25, 1961

1961年肯尼迪提出
十年内实现登月
此时尚未有可观测性理论

复杂系统可观测性

外部输出==>零侵扰

内部状态==>多维度

有限时间==>实时性

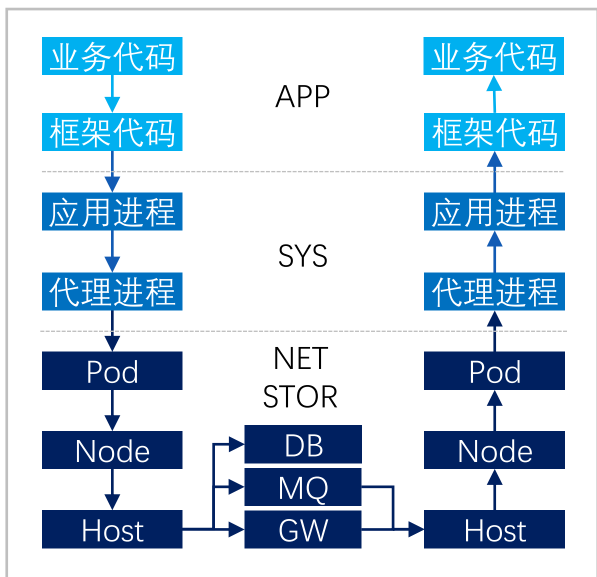
NASA sent Apollo 11
July 20, 1969

1969年阿波罗11号
完成登月壮举
可观测性实现飞船控制

云原生应用：复杂度急剧增长的 IT 系统

#路径=N^2

连接微服务的基础设施
路径越来越长、多



全栈可观测性越来越重要，如何消除业务开发与基础设施之间的鸿沟?

服务
N

路径
N^2

?

Prometheus
OpenTelemetry
fluentd

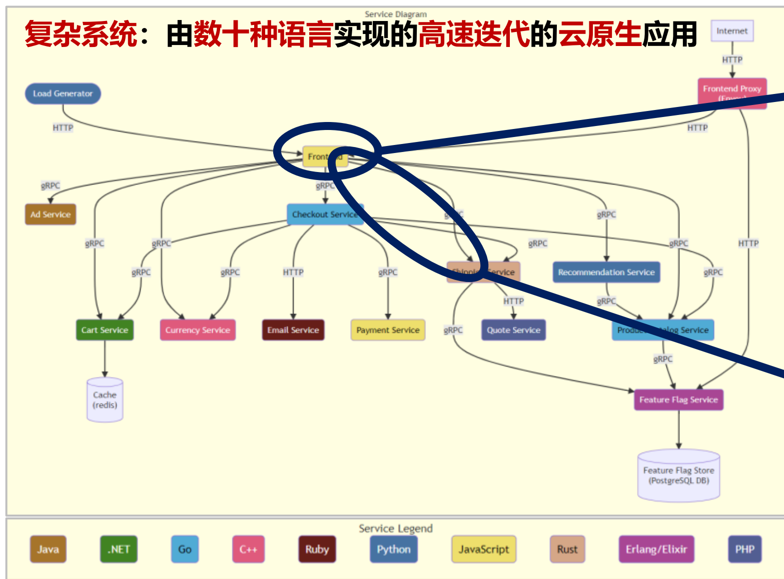
复杂系统可观测性
外部输出==>零侵扰
内部状态==>多维度
有限时间==>实时性

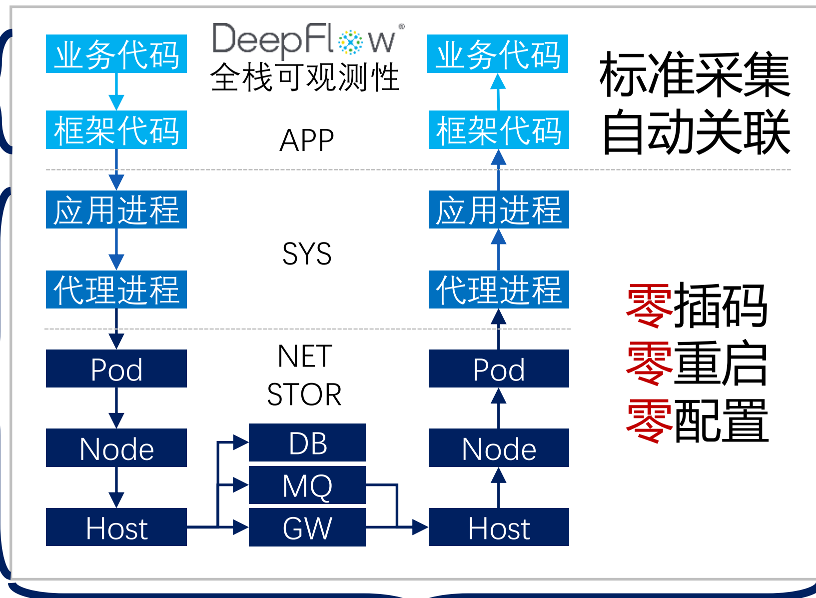
#服务=N

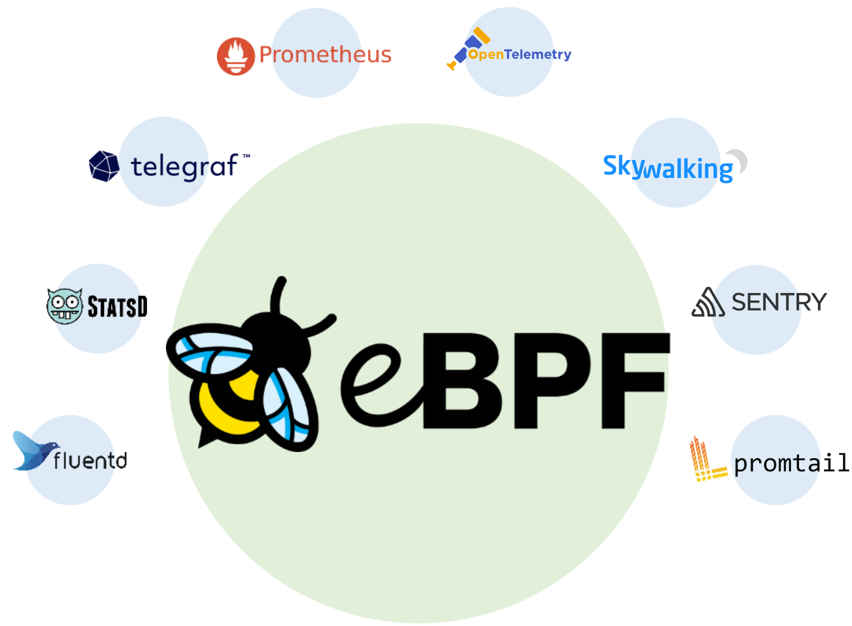
单个服务越来越简单，服务发布越来越快速
通用逻辑逐渐卸载至基础设施，开发语言和框架越来越自由

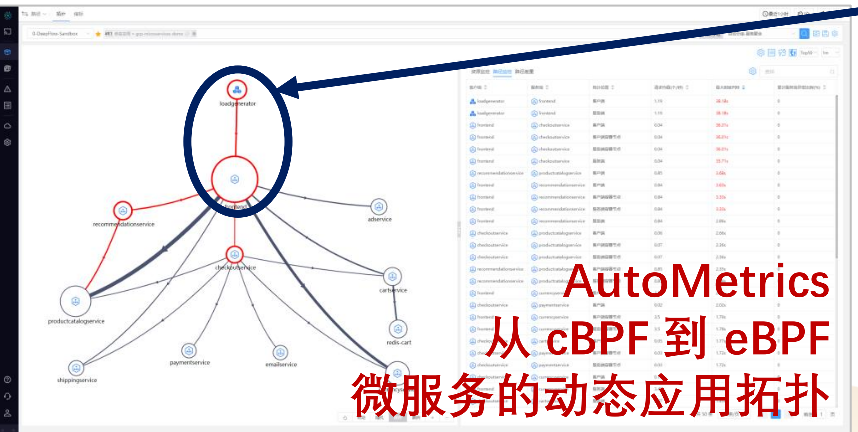
首届中国eBPF研讨会
www.ebpftravel.com

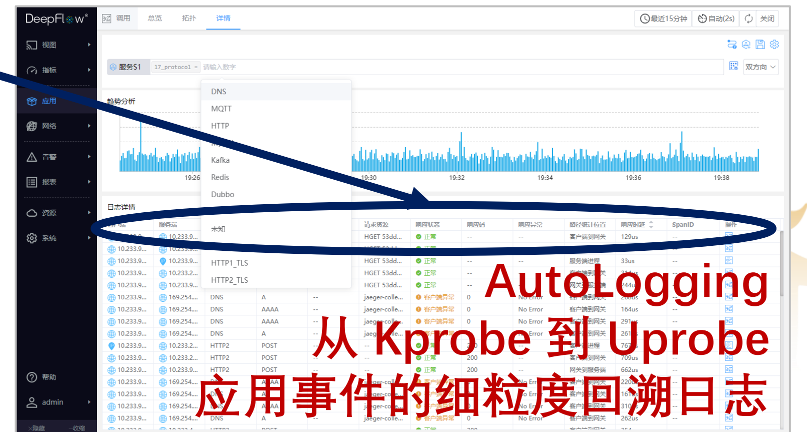
DeepFlow：让云原生开发者轻松实现全栈可观测性











目录

1 AutoMetrics
从 cBPF 到 eBPF
微服务的动态应用拓扑

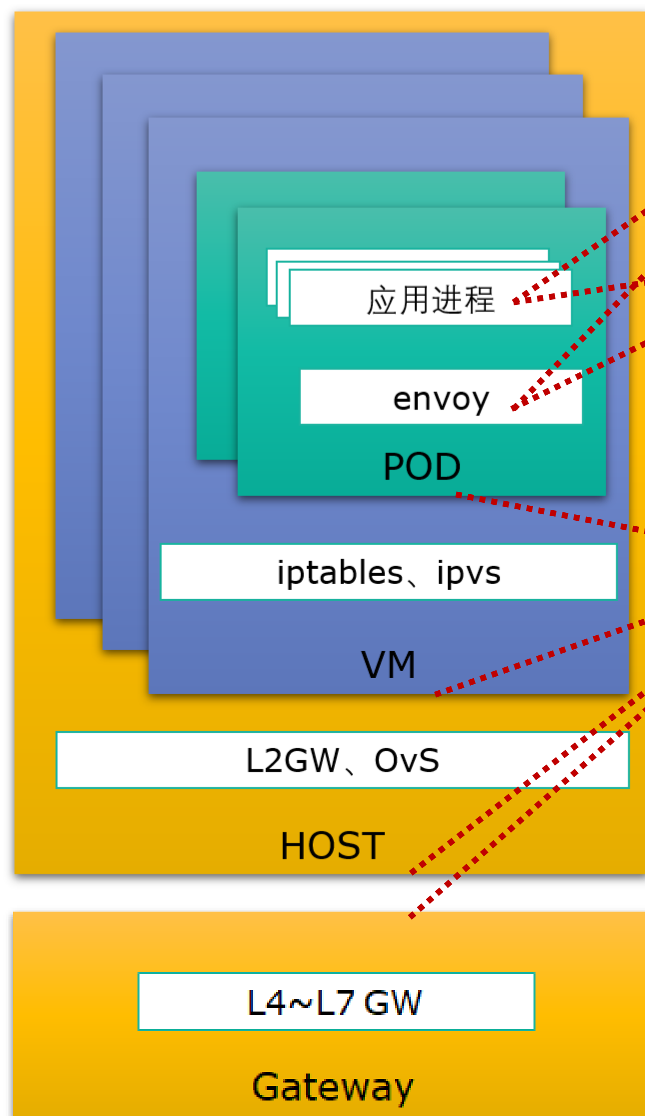
2 AutoTracing
从 InProcess 到 Distributed
用户请求的零侵扰分布式追踪

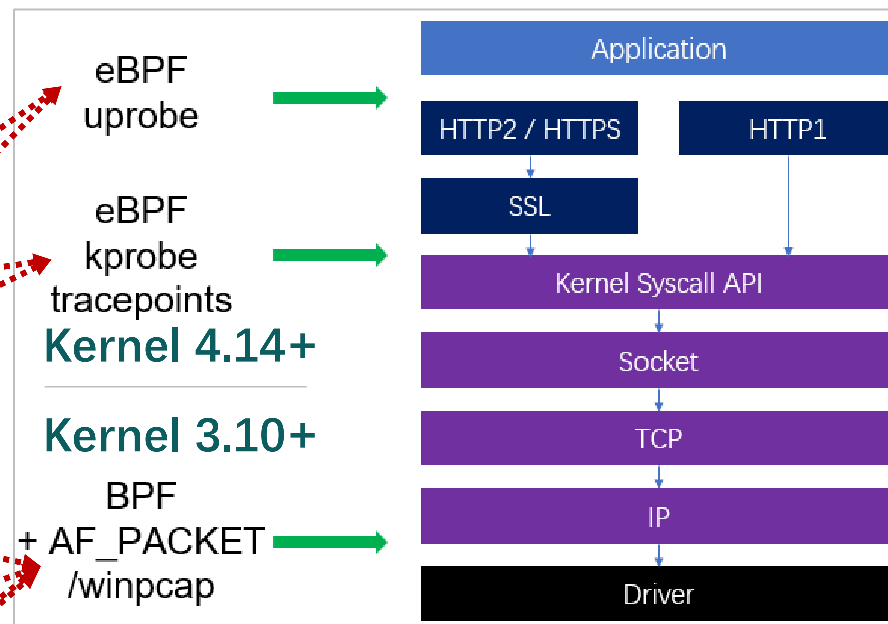
3 AutoLogging
从 Kprobe 到 Uprobe
应用事件的细粒度回溯日志

首届中国eBPF研讨会
www.ebpftravel.com

基于 eBPF 如何实现应用的全栈可观测性

业务代码
➔
框架/库代码
➔
服务网格 sidecar
➔
容器网络 iptables/ipvs
➔
云网络 ovs/linuxbr
➔
网关、数据库





① 采集：Packet、Syscall Data、Func Data
② 聚合：L4 Flow、L7 Session
③ 聚合：生成全栈性能 Metrics
④ 提取：结构化 FlowLog / RequestLog（Span）
⑤ 关联：基于 Span 生成 Distributed Trace

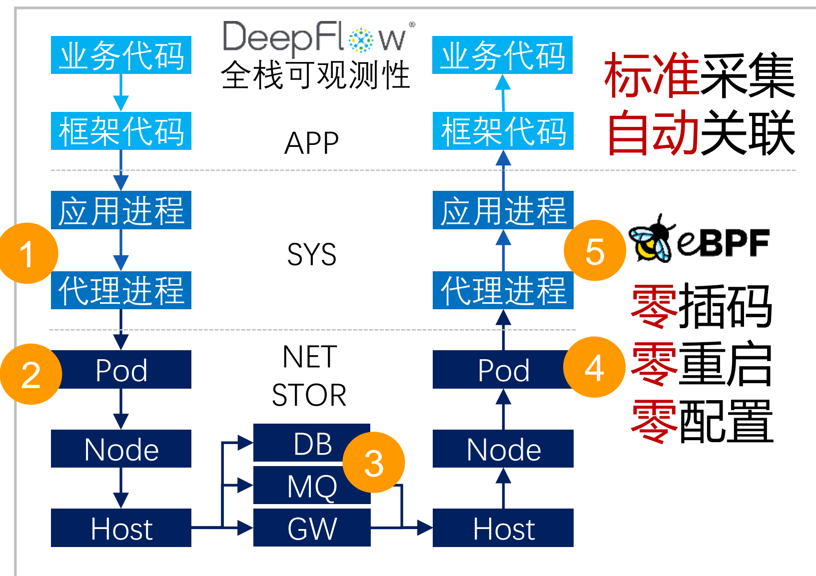
我在访问谁，谁在访问我

PID PID PID Host PID PID PID Single IP

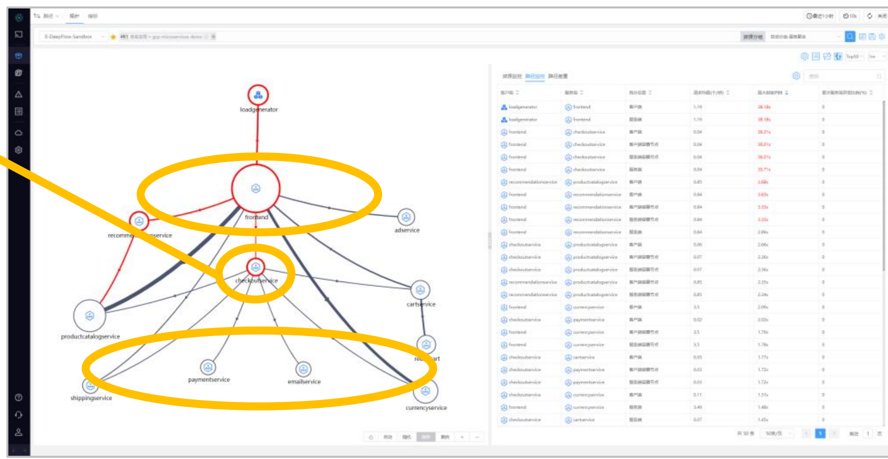
Pod Pod SNAT Pod K8s Node Pod Pod Pod

我的服务 → GW → 下游服务

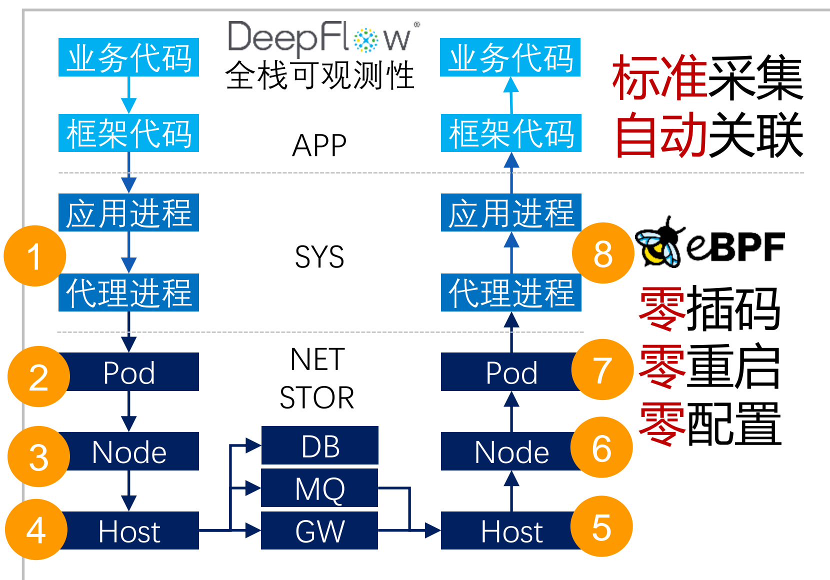
不插码，你真的知道谁在访问你吗？
某互联网用户，使用 DeepFlow 5 分钟内从数万个 Pod 中定位 RDS 访问量最大的Pod、服务、团队。



即便插码，你真的知道你在访问谁吗？
某金融用户，信用卡核心业务上线受阻，使用 DeepFlow 5 分钟内发现两个服务之间 API 网关是性能瓶颈。



我的访问在哪里出问题了



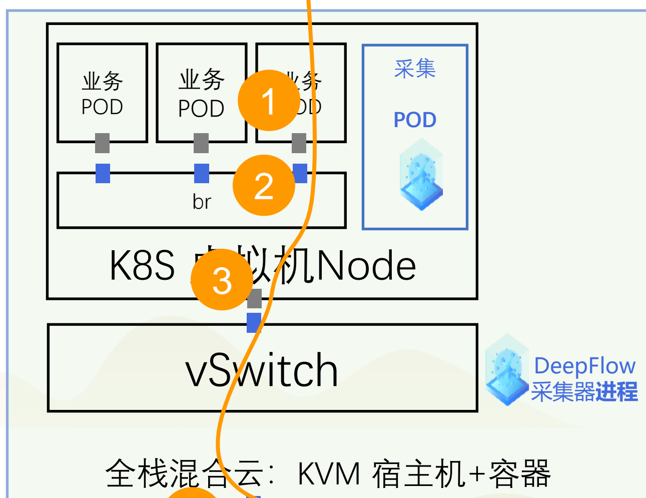
某互联网用户，使用 DeepFlow 5 分钟定位服务间 K8s 网络瓶颈。

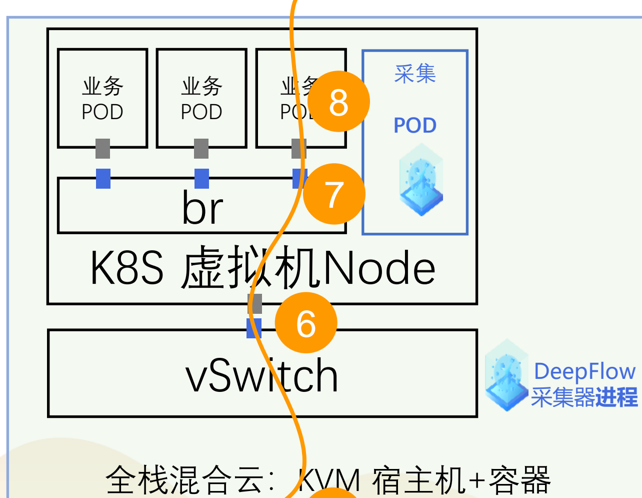
某金融用户，使用 DeepFlow 5 分钟定位 ARP 异常导致的 Pod 无法 Ready。

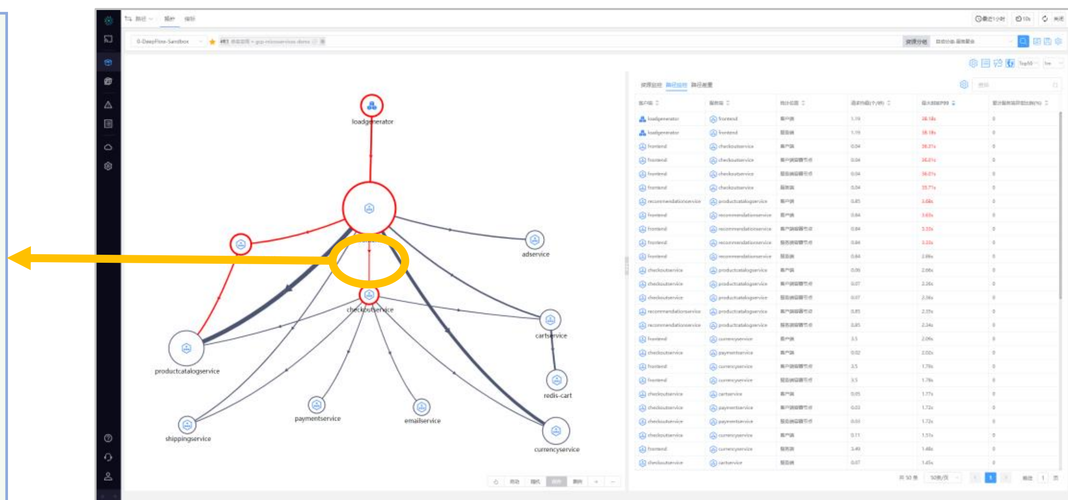
某系统软件用户，使用 DeepFlow 5 分钟定位客户端未及时收包导致 gRPC 超时。

从云基础设施到云原生应用的全栈性能指标：

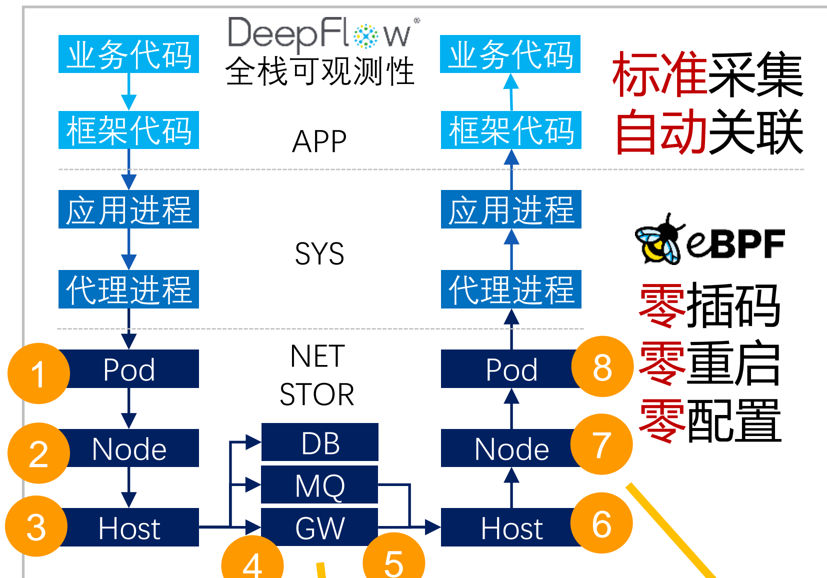
应用性能：吞吐、时延、异常（RED）
应用协议：HTTP(S)，RPC，SQL、MQ、DNS、…
系统性能：新建连接、活跃连接、建连异常、…
网络时延：建连时延、系统时延、数据时延、…
网络性能：吞吐、重传、零窗、传输层载荷、…







我的访问在哪里出问题了 (NFVGW)



某金融用户，使用 DeepFlow 5 定位困扰云厂商多日的**路由环路**造成的**一个服务**上云下云时延周期性飙升。

某金融用户，使用 DeepFlow 5 分钟内定位**某个 NFVGW 实例**对**一组服务流量**丢包导致客户端频繁重试。

智能 NAT 追踪：
追踪一个四元组前后经过若干次 SNAT、DNAT、FULLNAT 变化后的流量拓扑和逐跳访问路径。

专有云网络：
十余种云网关
多种隧道封装协议
十余种通信端点
上百种穿越 NFV 区的路径

K8s Node/KVM/Hyper-V NAT：

iptables	ipvs	OvS

NFVGW NAT：

UNATGW	STGW-L	JNSGW
NATGW	TGW-L	PVGW
STGW-W	TGW EIP	DCGW
TGW-W	VPCGW	PCGW

DeepFlow 与阿里飞天、腾讯 TCE、华为 HCS 已合作三年，深度适配。

输入任意四元组，自动追踪一次或多次、SNAT、DNAT、FULLNAT前后的流量拓扑、逐跳链路

吞吐、时延、重传、… 指标量完全支持

客户端CVM
客户端母机
进VPCGW
出VPCGW
进TGW
出TGW
进服务端母机
服务端CVM

客户端访问VPCGW VIP
第一次DNAT（仅改变端口）
第二次DNAT 到后端主机（三个CVM）

目录

1 AutoMetrics
从 cBPF 到 eBPF
微服务的动态应用拓扑

2 AutoTracing
从 InProcess 到 Distributed
用户请求的零侵扰分布式追踪

3 AutoLogging
从 Kprobe 到 Uprobe
应用事件的细粒度回溯日志

面向用户请求的零侵扰分布式追踪：AutoTracing

调用链追踪

04-eBPF-Istio

流量统计表格

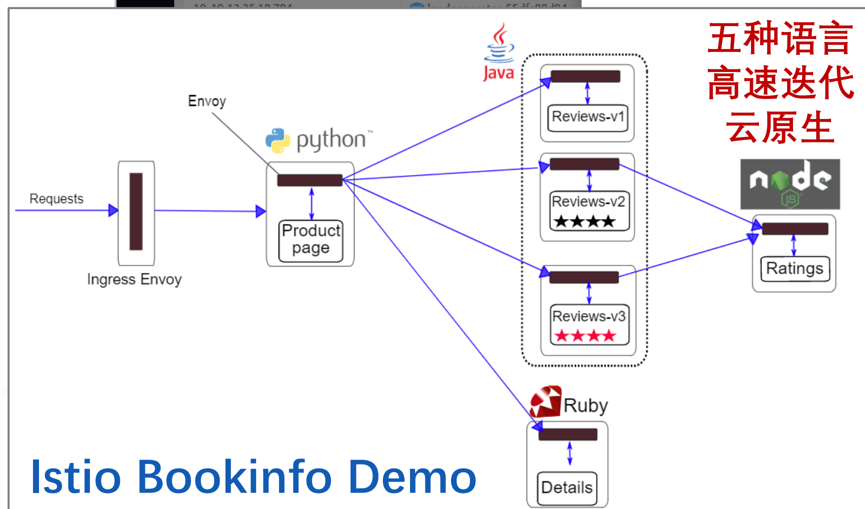
详情信息

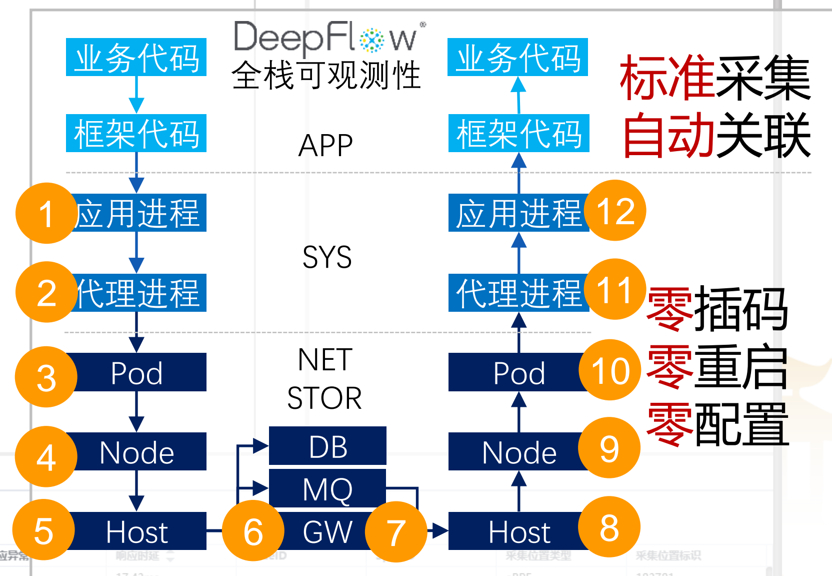
2022-10-19 13:35:18.787564 | loadgenerator-55dfc88d94-xgbbg > productpage-v1-7cf78fc9b5-dvbnn | HTTP GET /productpage

BPF

eBPF

服务名称	时延占比
productpage	45.44%
reviews	24.81%
productpage-v1	6.34%
ratings-v1	6.16%
loadgenerator	4.37%
details	4.00%
details-v1	3.99%
reviews-v3	3.72%
ratings	1.17%

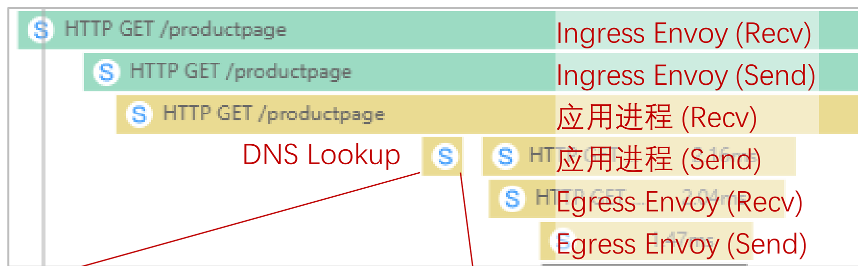




采集并关联系统和网络 Span

- 业务开发团队
- 服务网格团队
- 容器运维团队
- 云运维团队

无需插码
快速定界



HTTP GET ... 2.16ms
HTTP GET ... 2.04ms
系统 (客户端进程)
DNS A details.deepflow-ebpf-istio-demo.svc.cluster.local
python productpage-v1-7cf78fc9b5-dvbnn
277us (100% of self time)

Curl (C)
Envoy (C++)
Envoy (C++)
ProductPage (Python)
Envoy (C++)
Envoy (C++)
Details (Ruby)
Envoy (C++)
Reviews (Java)
Envoy (C++)
Envoy (C++)
Ratings (Node.js)

1. 零插码：且无需向 HTTP 头注入 TraceID 或 SpanID
2. 全链路：4 个调用、38 个 Span，分为 24 eBPF Span + 14 cBPF Span
3. 多语言：Java、Python、Ruby、Node.js 及 C/C++ (curl/envoy)
4. 全栈：追踪两个微服务之间的网络路径，从 Pod 到 Node 到 KVM，IPIP/VXLAN，并关联网络 Metrics
5. 全栈：追踪微服务内从 Envoy Ingress ➔ 服务 ➔ DNS ➔ Envoy Egress 全过程

案例：某互联网用户，使用 DeepFlow 5 分钟内定位客户端慢服务端不慢的经典扯皮问题。

如何实现零侵扰的分布式追踪?

Service

eBPF Hooks

Service

eBPF + cBPF Hooks

Service Service Service Service Service

追踪一个服务前后的入出请求：
同线程追踪：ThreadID、SyscallTraceID、…
跨线程追踪：SessionID、X-Request-ID、…
跨协程追踪：CoroutineID、…

追踪两个服务间的网络路径：
TCP.SEQ、TimeWindow、…

目录

1
AutoMetrics
从 cBPF 到 eBPF
微服务的动态应用拓扑

2
AutoTracing
从 InProcess 到 Distributed
用户请求的零侵扰分布式追踪

3
AutoLogging
从 Kprobe 到 Uprobe
应用事件的细粒度回溯日志

面向事件的细粒度回溯日志

从应用到基础设施全栈

请求类型、请求域名、请求资源
RPC Endpoint、请求 ID
响应状态、响应码、响应异常、响应结果
客户端真实 IP、UserAgent、Referer
TraceId、SpanId、XRequestId
… // WASM、LUA 插件式扩展能力*

应用访问日志 → 网络流日志 / 文件IO日志 * → TCP 包头时序图 → PCAP 报文回溯

DeepFlow 全栈可观测性

业务代码 → 框架代码 → 1 应用进程 → 2 代理进程 → 3 Pod → 4 Node → 5 Host → 6 GW → 7 → 8 Host → 9 Node → 10 Pod → 11 代理进程 → 12 应用进程 → 框架代码 → 业务代码

APP / SYS / NET STOR / DB / MQ

标准采集
自动关联

eBPF
零插码
零重启
零配置

某政府用户，使用 DeepFlow 流日志替代了全包存储，回溯查询速度 10x，存储时长 100x。

某金融用户，使用 DeepFlow TCP 包头时序图 5 分钟内发现 Cloud GW 转发 SYN 时延大。

eBPF hooks (GitHub deepflowys/deepflow)

```
grep -rnF ".symbol =" agent/src/ebpf/user/

./go_tracer.c:173: .symbol = "runtime.casgstatus",
./go_tracer.c:179: .symbol = "crypto/tls.(*Conn).Write",
./go_tracer.c:185: .symbol = "crypto/tls.(*Conn).Write",
./go_tracer.c:191: .symbol = "crypto/tls.(*Conn).Read",
./go_tracer.c:197: .symbol = "crypto/tls.(*Conn).Read",
./go_tracer.c:206: .symbol = "net/http.(*http2serverConn).writeHeaders",
./go_tracer.c:212: .symbol = "golang.org/x/net/http2.(*serverConn).writeHeaders",
./go_tracer.c:220: .symbol = "net/http.(*http2serverConn).processHeaders",
./go_tracer.c:226: .symbol = "golang.org/x/net/http2.(*serverConn).processHeaders",
./go_tracer.c:234: .symbol = "net/http.(*http2clientConnReadLoop).handleResponse",
./go_tracer.c:240: .symbol = "golang.org/x/net/http2.(*clientConnReadLoop).handleResponse",
./go_tracer.c:248: .symbol = "net/http.(*http2ClientConn).writeHeader",
./go_tracer.c:254: .symbol = "golang.org/x/net/http2.(*ClientConn).writeHeader",
./go_tracer.c:262: .symbol = "net/http.(*http2ClientConn).writeHeaders",
./go_tracer.c:268: .symbol = "golang.org/x/net/http2.(*ClientConn).writeHeaders",
./go_tracer.c:277: .symbol = "google.golang.org/grpc/internal/transport.(*loopyWriter).writeHeader",
./go_tracer.c:285: .symbol = "google.golang.org/grpc/internal/transport.(*http2Client).operateHeaders",
./go_tracer.c:293: .symbol = "google.golang.org/grpc/internal/transport.(*http2Server).operateHeaders",
.
./ssl_tracer.c:46: .symbol = "SSL_write",
./ssl_tracer.c:52: .symbol = "SSL_write",
./ssl_tracer.c:58: .symbol = "SSL_read",
./ssl_tracer.c:64: .symbol = "SSL_read",
```

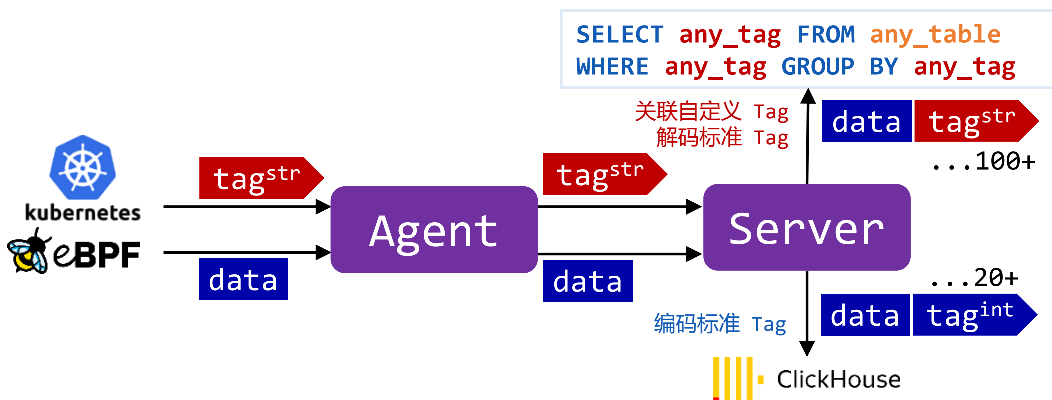
```
grep -rnF "KPROG" agent/src/ebpf/kernel/
./include/bpf_base.h:200:#define KPROG(F) SEC("kprobe/"__stringify(F)) int kprobe__##F

./socket_trace.c:1152:KPROG(__sys_sendmsg) (struct pt_regs* ctx) {
./socket_trace.c:1189:KPROG(__sys_sendmmsg)(struct pt_regs* ctx) {
./socket_trace.c:1235:KPROG(__sys_recvmsg) (struct pt_regs* ctx) {
./socket_trace.c:1271:KPROG(__sys_recvmmsg) (struct pt_regs* ctx) {
./socket_trace.c:1320:KPROG(do_writev) (struct pt_regs* ctx) {
./socket_trace.c:1353:KPROG(do_readv) (struct pt_regs* ctx) {
```

```
grep -rnF "TPPROG" agent/src/ebpf/kernel/
./include/socket_trace.h:189:#define TPPROG(F) SEC("tracepoint/syscalls/"__stringify(F))
int bpf_func_##F

./socket_trace.c:1019:TPPROG(sys_enter_write) (struct syscall_comm_enter_ctx *ctx) {
./socket_trace.c:1035:TPPROG(sys_exit_write) (struct syscall_comm_exit_ctx *ctx) {
./socket_trace.c:1050:TPPROG(sys_enter_read) (struct syscall_comm_enter_ctx *ctx) {
./socket_trace.c:1065:TPPROG(sys_exit_read) (struct syscall_comm_exit_ctx *ctx) {
./socket_trace.c:1086:TPPROG(sys_enter_sendto) (struct syscall_comm_enter_ctx *ctx) {
./socket_trace.c:1102:TPPROG(sys_exit_sendto) (struct syscall_comm_exit_ctx *ctx) {
./socket_trace.c:1123:TPPROG(sys_enter_recvfrom) (struct syscall_comm_enter_ctx *ctx) {
./socket_trace.c:1138:TPPROG(sys_exit_recvfrom) (struct syscall_comm_exit_ctx *ctx) {
./socket_trace.c:1176:TPPROG(sys_exit_sendmsg) (struct syscall_comm_exit_ctx *ctx) {
./socket_trace.c:1214:TPPROG(sys_exit_sendmmsg) (struct syscall_comm_exit_ctx *ctx) {
./socket_trace.c:1256:TPPROG(sys_exit_recvmsg) (struct syscall_comm_exit_ctx *ctx) {
./socket_trace.c:1302:TPPROG(sys_exit_recvmmsg) (struct syscall_comm_exit_ctx *ctx) {
./socket_trace.c:1338:TPPROG(sys_exit_writev) (struct syscall_comm_exit_ctx *ctx) {
./socket_trace.c:1371:TPPROG(sys_exit_readv) (struct syscall_comm_exit_ctx *ctx) {
./socket_trace.c:1385:TPPROG(sys_enter_close) (struct syscall_comm_enter_ctx *ctx) {
./socket_trace.c:1407:TPPROG(sys_enter_getppid) (struct syscall_comm_enter_ctx *ctx) {
./socket_trace.c:1437:TPPROG(sys_exit_socket) (struct syscall_comm_exit_ctx *ctx) {
```

为所有可观测性信号注入标准标签：连接应用和基础设施



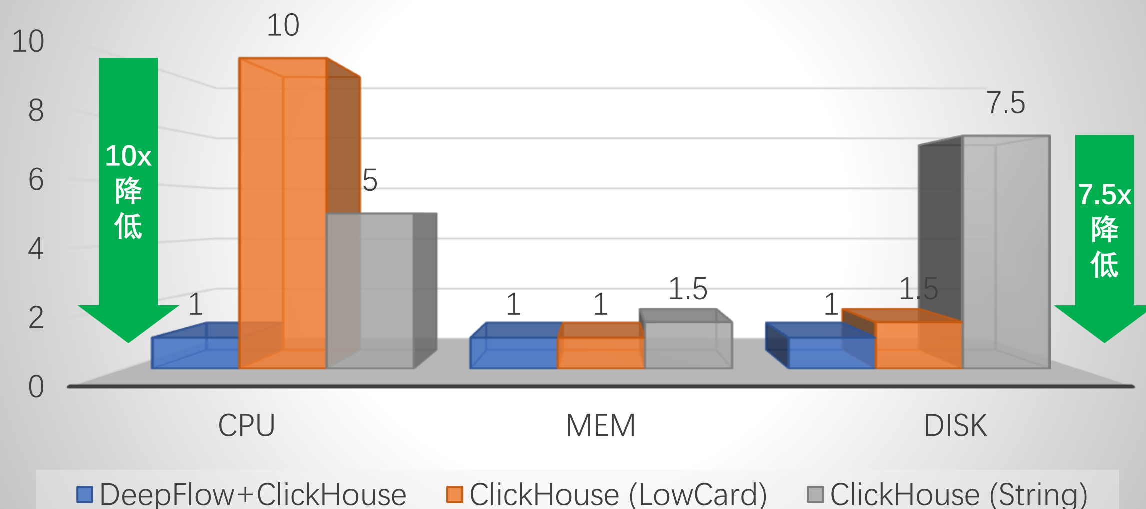
标准 Tag：开销 10x 降低

资源池	网络资源	容器服务	Application
区域	VPC	容器集群	ServiceName
可用区	子网	容器节点	FunctionName
云平台	CIDR	命名空间	Endpoint
租户	IP地址	容器服务	TraceId
云资源	NATGW	Ingress	SpanId
宿主机	ALB	Workload	RequestId
云服务器	...	POD	...

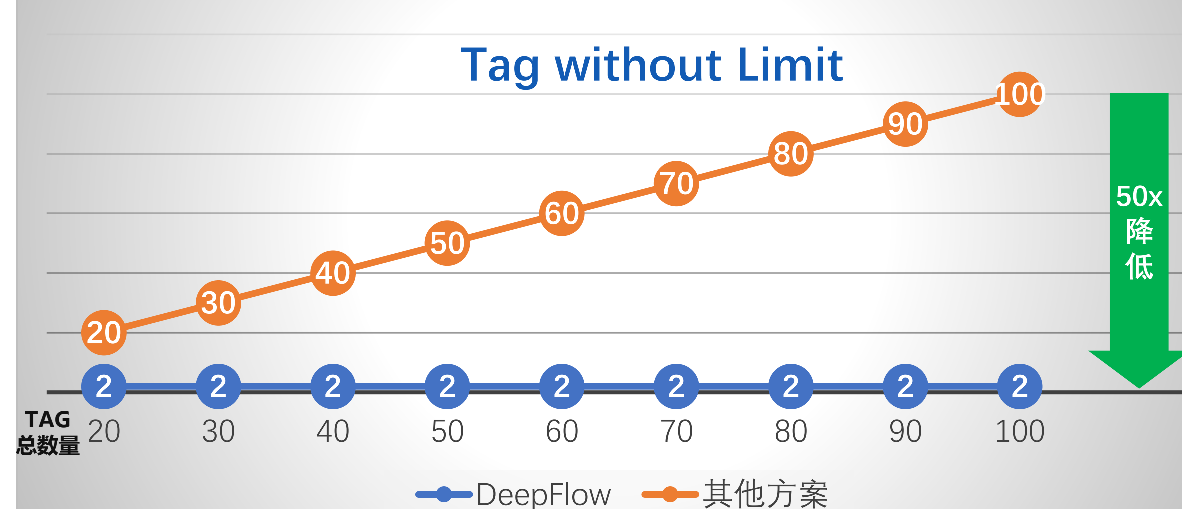
自定义 Tag：零开销

K8s labels	Annotations *
app	biz/terminalType
version	cicd/deploymentId
env	...
owner	**OS ENV ***
stage	MODULE_NAME
commitId	...
...	

AutoTagging & SmartEncoding



随机生成一组长度为 16 字符串标签，Cardinality 为 5000，持续极限速率写入。



假设标准 TAG 固定 20 个，自定义 TAG 从 0 个增长到 80 个。

DeepFlow：让云原生开发者轻松实现全栈可观测性

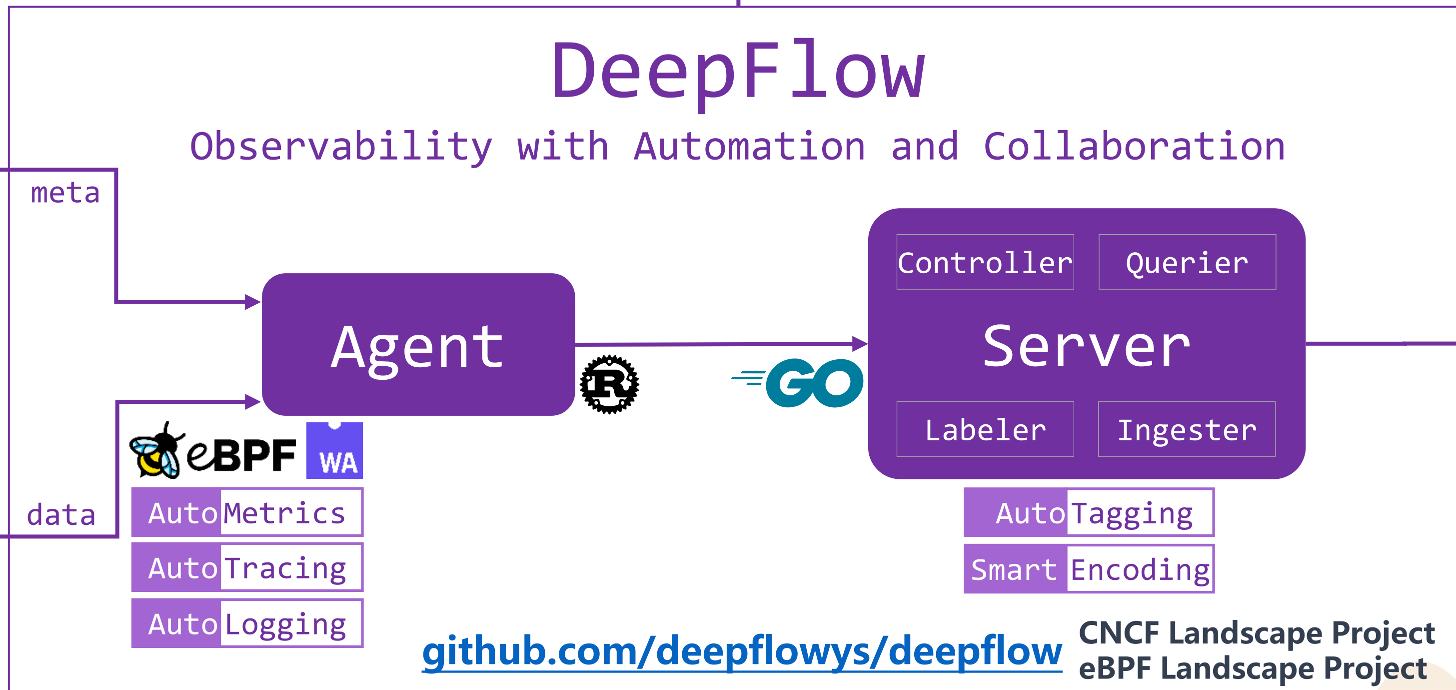
DeepFlow
开源社区微信群

Any Tag
svc, instance, endpoint, ...

Consul Eureka
kubernetes serverless
aws HUAWEI

Any Stack
application, infra, ...

GO Java python
envoy NGINX redis
MySQL kafka MQTT



Storage options
ClickHouse
SLS

Southbound Integrations
SENTRY Skywalking OpenTelemetry Prometheus telegraf STATSD fluentd

Thanks~!