擎创科技APM可行性分析

技术可行性分析

一、背景

APM 中 【服务监控】当前计划通过 Ebpf 采集 “服务” 信息，实现无侵入数据采集。

现对 Ebpf 采集的指标进行分析，判断是否满足 【服务监控】 所需的功能，及对比 Datadog、观测云上 【服务监控】产生的差异。

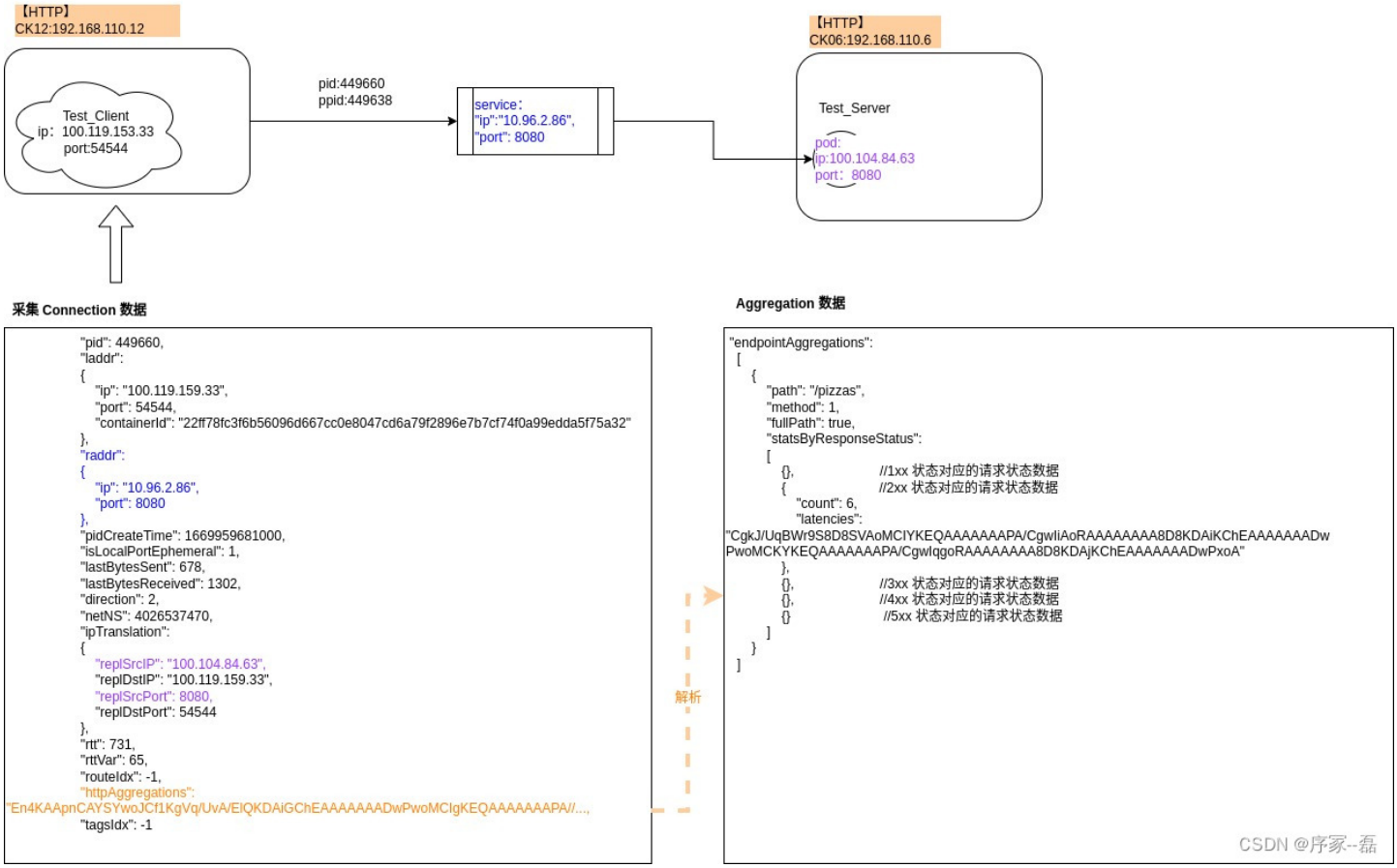
二、技术分析

2.1 采集的数据逻辑关系

Demo 示例：

访问对象	1. 访问对象：test-client -> test-server	
对象数据	1. test-client: 在主机 ck12 上； 1. container_id:22ff78fc3f6b 1. pod_name:test-client-6bb6869b4b-6j9qz 1. container_name:test-client 1. test-server：在主机 ck06 上； 1. container_id: 1. pod_name:test-7b589b575-895lz	

数据之间逻辑：



1. 【Connection】展示源端 “主机” 地址、“容器” 信息及目标端的 “容器 IP”、“Service” 信息；--- 需要知道 “目标端” 数据

1. 【Connection】确定调用方向；

1. 【httpAggregation】可以解析出 Path / Method：展示请求访问路径、方式、请求数数据；--- 可以反推 “path” 访问数据，通过 “process” 与 POD 进行关联；

1. 采集后的数据按照 key：value 存在内存中：

1. key ： 根据【connection】中 pid、laddr、raddr 生成；

1. value：通过【connection】中 “httpAggregation” 中 value 解析生成；

```
aggregations  map[http.KeyTuple]*aggregationWrapper
```

2.2 Ebpf 采集的 Connection 数据

Connection 为进程数据，有源端 ContainerID 数据，可以逆推 POD、Service 数据，及解析后的 path 请求数据。

但是目标端的数据比较少，只有容器 IP+Port，需要后续考虑关联；

```
            "hostName": "ck12",
            "connections":
            [{
                    "pid": 449660,
                    "laddr":          //源端对应的主机容器地址
                    {
                        "ip": "100.119.159.33",
                        "port": 54544,
                        "containerId": "22ff78fc3f6b56096d667cc0e8047cd6a79f2896e7b7cf74f0a99edda5f75a32"
                    },
                    "raddr":          //目标端对应的 Service 信息
                    {
                        "ip": "10.96.2.86",
                        "port": 8080
                    },
                    "pidCreateTime": 1669959681000,   // 时间单位 ms
                    "isLocalPortEphemeral": 1,
                    "lastBytesSent": 678,    //最近一次发送的字节数
                    "lastBytesReceived": 1302,  //最近一次接受的字节数，计算方式 = 总的接受量 - 之前统计的量
                    "direction": 2,  // 确定请求方向，2 表示laddr 为 outgoing，1 表示 laddr 为 incoming
                    "netNS": 4026537470,
```

```
                    "ipTranslation":    //请求方向数据
                    {
                        "replSrcIP": "100.104.84.63",  //回复目标端容器 IP
                        "replDstIP": "100.119.159.33", //回复源端容器 IP
                        "replSrcPort": 8080,
                        "replDstPort": 54544
                    },
                    "rtt": 731,    // 数据包往返时间，单位 ms
                    "rttVar": 65,   // 平滑过的平均偏差
                    "routeIdx": -1,  //判断是否为 AWS
                    "httpAggregations": "En4KAApnCAYSYwoJCf1KgVq/UvA/ElQKDAiGChEAAAAAAADwPwoMCIgKEQAAAAAAAPA/CgwIigoRAAAAAAAA8D8KDAimCh
                    "tagsIdx": -1   //标识 windows 服务器或 https 协议
                },
...
            "groupId": 1185441545,     // 同一采集周期内数据，groupId 相同，避免消息体过大进行拆分
            "groupSize": 3,
            "containerForPid":
            {
                "114220": "b1f9e399436cb9037f66fbaed62a1179674f2ce0d138162aa746774f248e1d88",
                "114284": "77c6ea7a5888a5b023ec7c6b73b4689ce0fba8b5ffab6f30bb791b0512c059f8",
                "114365": "dcda8023e3e0910e54ae33c2532bab267294a441c75e73c469261b7c0ab2930b",
                "114435": "bc9016612905d234f921360a45e133a1229b87bed22f56d90b9521232e58949d",
                "114490": "0468c9da020b006988516de2caf1f9f1844a0b848912b12cc3cf82cd1ed802a6",
                "355462": "945f322d44251003b9be55c6dea8aacc6e8f5aaa383e56184da7a552bfaf3ae6",
                "355662": "d976c27bdfb71d868d5b501f37a440cae5a3988287d9fc887804723824b32802",
                "358448": "1c9b6c79b29b4f1188daeabc7a8937121f89b38d92b2ff63f2535f4d8e16b3b0",
                "394711": "d79e1d1d82b7a1d9031005f9e4da2d5865c85bf99d77706505e1786798cb747c",
                "395250": "6c89087fe7691327e8b1e56e50a8bf16cc26eccbbe431eb3fa4e790ecda9e76a",
                "395256": "06df3dddba440f0b8f5d088ae55bad73118eed102bf7a0de174d33b5f431a9ed",
                "449660": "22ff78fc3f6b56096d667cc0e8047cd6a79f2896e7b7cf74f0a99edda5f75a32",  //进程与容器ID映射
                "485295": "311b7a96786b60a38bc132ba405e0697a4a968a0d86c408a871066576c2f1763"
            },
            "encodedConnectionsTags": "AgUAAAA=",     //❓疑问，待确定可以解析出什么?
            "connTelemetryMap":
            {
                "closed_conn_dropped": 0,
                "conn_dropped": 0,
                "conns_bpf_map_size": 274,
                "conns_closed": 1100,
                "conntrack_registers": 155,
                "conntrack_sampling_percent": 100,
                "dns_packets_processed": 340,
                "dns_stats_dropped": 0,
                "http_requests_dropped": 0,
                "http_requests_missed": 0,
                "kprobes_missed": 657,
                "kprobes_triggered": 1875443,
                "perf_lost": 0,
                "udp_sends_missed": 0,
                "udp_sends_processed": 395
            },
            "architecture": "x86_64",
            "kernelVersion": "5.4.217-1.el7.elrepo.x86_64",
            "platform": "centos",
            "platformVersion": "7 (Core)",
            "compilationTelemetryByAsset":
            {
                "conntrack":
                {},
                "http":
                {},
                "oomKill":
                {
                    "runtimeCompilationEnabled": true,
                    "runtimeCompilationResult": 9,
                    "runtimeCompilationDuration": 3594489,
                    "kernelHeaderFetchResult": 9
                },
                "runtimeSecurity":
                {},
                "tcpQueueLength":
                {},
                "tracer":
                {}
            },
            "agentConfiguration":
            {
                "npmEnabled": true,
                "tsmEnabled": true
```

```
101            },102
           "encodedDomainDatabase": "BAAsdml6aWVyLXF1ZXJ5LWJyb2tlci1zdmMucGwuc3ZjLmNsdXN0ZXIubG9jYWwvdGVzdC1zZXJ2aWNlLW5vZGVwb3J0LmRlZmF1b
           "encodedDnsLookups": "AgMAAwEADh8BCjEwLjk2LjIuODYBLwENMTA0LjIwOC4xNi45MAFfAgsxMC45Ni4zLjE3NgECDTExOC4zMS4xMjYuNzYBgAE="104
```

2.3 Aggregation 解析后的 Path 数据

通过 Connection 中 “httpAggregations” 可以解析得到每个进程的 “path” 数据，一个进程会有多个网络请求。

```
{

   "endpointAggregations":

   [

       {

           "path": "/pizzas",

           "method": 1,

           "fullPath": true,

           "statsByResponseStatus":

           [

               {},                       //1xx 状态对应的请求状态数据

               {                         //2xx 状态对应的请求状态数据

                   "count": 6,

                   "latencies": "CgkJ/UqBWr9S8D8SVAoMCIYKEQAAAAAAAPA/CgwIiAoRAAAAAAAA8D8KDAiKChEAAAAAAADwPwoMCKYKEQAAAAAAAPA/CgwIqgoRAAA

               },

               {},                       //3xx 状态对应的请求状态数据

               {},                       //4xx 状态对应的请求状态数据

               {}                         //5xx 状态对应的请求状态数据

           ]

       }

   ]

}
```

statsByResponseStatus：Http 响应码；

```
HTTP status classes (1XX, 2XX, 3XX, 4XX, 5XX)

const NumStatusClasses = 5



// RequestStats stores stats for HTTP requests to a particular path, organized by the class

// of the response code (1XX, 2XX, 3XX, 4XX, 5XX)

type RequestStats struct {

   data [NumStatusClasses]*RequestStat

}
```

2.4 Latency 解析后的数据

```
[

    {
```

```
        "statsByResponseStatus": [

            {

                "count": 2,

                "latencies": {

                    "mapping": {

                        "gamma": 1.02020202020202     //计算矫正值?

                    },

                    "positiveValues": {

                        "binCounts": [

                            {

                                "key": 698,       //根据 Key 的值得到响应时间: gamma 的 Key 次方

                                "value": 1.0     //相同响应时间下，请求的个数

                            },

                            {

                                "key": 711,

                                "value": 1.0

                            }

                        ]

                    },

                    "negativeValues": {}

                },

                "firstLatencySample": 0.0,

                "httpCode": "2**"         //响应状态码为 2xx，没有具体的响应状态码吗? 那如何知道是服务端异常还是客户端?

            }

        ],

        "path": "/nacos/v1/ns/instance/beat",

        "method": "Put",

        "fullPath": true

    },

]
```

三、【服务监控】功能满足度

所属模块	功能	功能描述	主要展示项	Ebpf 采集是否可满足
服务列表	服务列表	展示搜索时间内，服务主要性能信息	• 服务名称 / 服务类型 • 运行 POD：2/3 • 请求数 • 响应时间 • 平均响应时间 • 75% line / 请求占比 • 错误率	• ❓ 关联 Process 采集的容器 Tags • ✅ • ✅ • ✅ • ✅ • ✅

			• 告警数，查看【告警历史】	• ✅ • ✅
	服务拓扑	展示搜索时间内，服务之间的依赖关系，可以根据 “请求”、”响应时间“、“错误率” 填充	• 调用关系 • 服务名称 / 服务类型 • 请求数 • 响应时间 • 错误率	• ❓ 关联 Process 采集的容器 Tags • ✅ • ✅ • ✅ • ✅
服务概览	服务概览	展示搜索时间内，服务历史运行趋势和数据	• 请求数趋势图 • 响应时间图表（响应满意度） • 平均响应时间 • 55% line • 75% line • 99% line • 错误数图表 • 错误数（不同状态码分析） • 错误率 • 慢接口（TOP5） • POD 错误比（环图） • 告警数图表，查看【告警历史】	• ✅ • ✅ • ✅ • ✅ • ✅ • ✅ • ✅ • ✅ • ✅ • ✅ • ✅
资源 Tab	资源列表	展示搜索时间内，服务下接口访问关键指标信息	• 接口名称 • 请求方式：POST/GET • 请求数 • 响应时间 • 错误率 • 告警数，查看【告警历史】 • 查看【仪表盘】	• ✅ • ✅ • ✅ • ✅ • ✅ • ✅ • ✅
	资源详情	点击「资源详情」，侧边【弹框】展示出搜索时间内，资源历史运行趋势和数据	• 接口名称 • 请求方式：POST/GET • 请求数图表 • 响应时间图表 • 平均响应时间 • 55% line • 75% line	• ✅ • ✅ • ✅ • ✅ • ✅ • ✅ • ✅

			• 99% line • 错误时间图表 • 错误数（不同状态码分析） • 错误率 • 慢分析（8s 以上响应） • 来源主机分类 • 告警数，查看【告警历史】 • 查看【仪表盘】	• ✅ • ✅ • ✅ • ✅ • ✅ • ✅ • ✅ • ✅
POD Tab	POD 列表	展示搜索时间内，服务下 POD 访问关键指标信息，可以进入【监控- POD 监控详情页】	• POD 名称 • Status • Ready • Restart • Age：运行时间 • 占服务请求比 • 平均响应时间 • 错误率 • 查看【仪表盘】 • 查看【POD 详情】 • 查看【主机 详情】	• ✅ • ✅ • ✅ • ✅ • ✅ • ✅ • ✅ • ✅ • ✅ • ✅ • ✅
异常分析 Tab	错误分析	按照错误出现的类型进行聚合分析，了解请求来源时间、请求来源 POD，并可异常【Trace】	• 异常名称 • 异常类型（Code Exception） • 影响请求数 • 发生频率 • 查看【Trace】 • 查看【仪表盘】	• ❓根据状态码？还会有网络错误吗 后续实现
	慢分析	1. 按照慢出现的类型进行聚会分析，了解慢问题原因和问题详情，再逐步查看【Trace】深入排查 1. 或者针对慢请求进行统计来源 API、来源 POD，再逐步查看【Trace】深入排查	• 异常名称 • 异常类型（Code Exception） • 影响请求数 • 发生频率 • 查看【Trace】 • 查看【仪表盘】	后续实现

四、总结

4.1 现状

1. Ebpf 采集的最小单位是 PID 的数据；

1. PID 中包含请求的

1. 源端： “容器地址” ；

1. 目标端： “容器地址”；

1. 请求： “path”、“method”、“latencies”、“status” 数据；

4.2 待进行（1205）

1. PID 对应 “容器” 需要找到对应关联 “POD”、“Service” ：

1. 方式一：关联 process 采集的 POD 数据；

1. 方式二：将 容器的 Tags 塞到 PID 数据中；

1. 字段之间的逻辑关系；

1. 字段含义；

1. 按照场景抓取数据：

1. 重传场景；

1. https 协议；

1. 按照服务端、客户端的维度采集；

五、技术方案

1. process-agent 在k8s worknode上报数据

1. DataDog Conntrack 以及Tracker 模块分析 模块分析

六、可能性实现

数据来源	判断的字段	得到的结果	示例
Connection	1. "pidCreateTime": 1669959681000 1. "lastTcpClosed":0	1. 连接的 “开始时间” 1. 连接类型：长连接、短连接； 1. 连接总耗时（包含多个请求）	例如： 1. “服务” “请求总数”； 1. “搜索时间” 内采集到的 PID 数据； 1. 统计 “服务” 作为 “Server” 端： 1. Endpoint 中 Count 之和
Connection	1. laddr 1. raddr 1. direction 1. iptranslation	1. 请求的 Client、Server 1. 可以推断出： 1. 所属 “POD” 1. 所属主机 1. 所属 “服务” 备注：其他相关属性均有 Client、Server 所属对象携带的标签带出。	
httpAggregation	1. "path": "/pizzas", 1. "method": 1,	1. 请求的 Endpoint	
httpAggregation	1. code 1. "count": 6,	1. 每个状态码（code） 1. 对应的 “请求总数” （count）	
Latency	1. "gamma": 1.02020202020202 1. "binCounts": [{	1. 每个状态码（code） 1. 对应每个请求的 “响应时间” 1. 可以计算得到：	

	"key": 698, "value":1.0 },	1. 平均响应时间 1. 请求分位时间 P50、P75、P90、P95、P99

例如：Datadog HTTP 请求数据结构

```
{

        "Client":

        {

            "IP": "172.17.0.6",

            "Port": 59554

        },

        "Server":

        {

            "IP": "192.168.110.14",

            "Port": 9363

        },

        "DNS": "",

        "Path": "/metrics",

        "Method": "GET",

        "ByStatus":

        {

            "200":

            {

                "Count": 1,

                "FirstLatencySample": 25362432,

                "LatencyP50": 0

            }

        },

        "StaticTags": 0,

        "DynamicTags": null

    },
```