packetbeat抓包监控服务质量实践

秦强强
WeChat:make_it_beter

- 一个需求, 对某些重要服务的调用做全量监控, 准确并实时的得到tp99等监控指标, 当终端出现问题时, 帮助开发第一时间发现, 定位, 解决问题

- 业界一般会用分布式环境下的调用链跟踪去做这个事情 (将要全量监控的部分设为不透传的域来记录并取到数据), 或者是在客户端预设埋点逻辑, 通过记录和收集日志来取到数据

- 然而

- 这边的调用链跟踪体系只是刚有雏形, 距离完善还有很长的路要走, 一段时间内还实现不了这个需求

- 这边除了java以外的客户端目前只简单实现了服务发现, 调用服务的方式都还是短连接调用, 而且各部门的主要精力在一段时间内会集中在业务开发上

- 填坑需要时间

- 服务层的数据协议是固定的(这边是finagle-thrift), 能否通过抓取服务层网络流量来拿到数据,作为过渡期解决方案?

packetbeat 简介

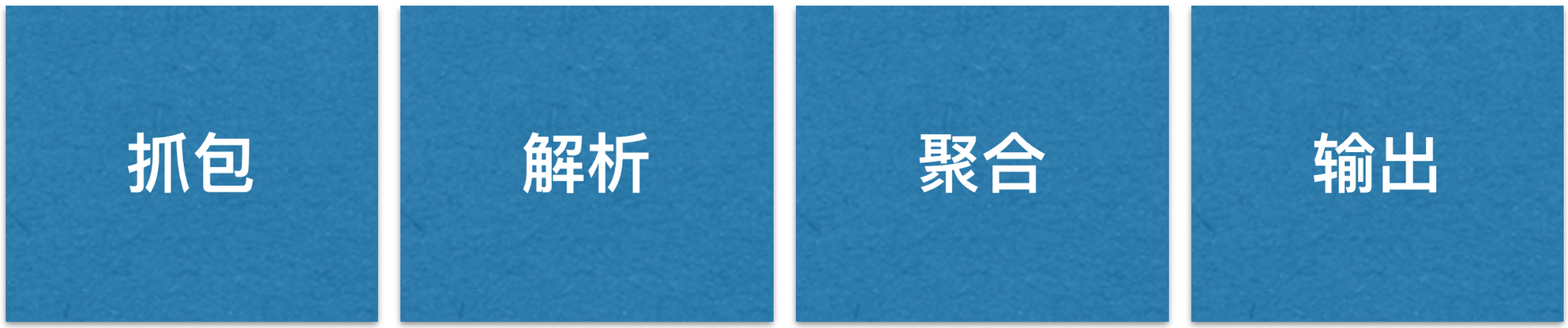
- 开源的网络抓包与分析框架, elastic旗下beats项目成员
- 简单, 高效, 流行
- https://www.elastic.co/products/beats/packetbeat

- packetbeat典型使用场景: 以某一层为维度, 获取该层的交互数据. 比如在redis节点上抓取redis层次的交互数据, 得到都是那些机器在请求redis, 它们都在用什么样的redis命令, 以及这些请求的结果和响应时延是怎样的

packetbeat方案的好处

- 各语言客户端无开发量
- 被动收集, 完全无侵入
- 中心化, 线上服务无需打包更新, 没推广问题

packetbeat 工作流程



Parse(协议数据)

解析协议数据, 根据交互信息产出聚合数据

PublishTransaction(聚合数据)

为packetbeat添加新协议(finagle-thrift)支持

- 参考其他协议的实现, 这边主要参考了thrift协议的实现
- 结合wireshark抓包和协议定义来分析数据包
- 扩展采用项目嵌套结构, 不依赖具体某个beats版本

- 更多扩展开发的资料可以参考文章: http://log.medcl.net/item/2015/12/packetbeat-zi-ding-yi-xie-yi-di-kai-fa-jiao-cheng/ http://siye1982.github.io/2016/04/30/packetbeat/

- 另一个问题, 通过packetbeat可以知道一次交互两端的ip和端口, 但如何知道一个tcp连接两端都是什么应用?

- packetbeat自带的procs模块可以定期去扫描进程信息来得到端口和程序的映射，但对短连接支持无力，而且可能会有性能问题

sysdig 简介

- 开源的系统运行信息和网络流监控软件
- 通过监听内核事件来得到系统运行细节
- http://www.sysdig.org/

通过sysdig命令行可以得到连接事件的事件时间, 程序pid,
程序目录等信息

```
root@localhost:/# sysdig "evt.category=net and proc.name=node and (evt.type in (connect, close)) and evt.dir='<'"
25118 22:10:57.011311067 15 node (5308) < connect res=-115(EINPROGRESS) tuple=172.28.1.112:10854->192.168.14.109:20880
30318 22:10:57.021229932 15 node (5308) < close res=0
44902 22:10:57.125472739 15 node (5308) < connect res=-115(EINPROGRESS) tuple=172.28.1.112:10860->192.168.14.109:20880
45892 22:10:57.149918286 15 node (5308) < close res=0
54181 22:10:57.251730396 15 node (5308) < connect res=-115(EINPROGRESS) tuple=172.28.1.112:10884->192.168.14.109:20880
57063 22:10:57.301154831 15 node (5308) < close res=0
62469 22:10:57.402553030 15 node (5308) < connect res=-115(EINPROGRESS) tuple=172.28.1.112:10886->192.168.14.109:20880
62892 22:10:57.412461016 15 node (5308) < close res=0
88360 22:10:57.515698594 15 node (5308) < connect res=-115(EINPROGRESS) tuple=172.28.1.112:10902->192.168.14.109:20880
```

```
/root/packetbeat-finagle-test-node-consumer/,172.28.1.112,28276,connect,1480083278
/root/packetbeat-finagle-test-node-consumer/,172.28.1.112,28276,close,1480083278
/root/packetbeat-finagle-test-node-consumer/,172.28.1.112,28294,connect,1480083278
/root/packetbeat-finagle-test-node-consumer/,172.28.1.112,28294,close,1480083278
/root/packetbeat-finagle-test-node-consumer/,172.28.1.112,28296,connect,1480083278
/root/packetbeat-finagle-test-node-consumer/,172.28.1.112,28296,close,1480083278
/root/packetbeat-finagle-test-node-consumer/,172.28.1.112,28298,connect,1480083278
/root/packetbeat-finagle-test-node-consumer/,172.28.1.112,28298,close,1480083278
/root/packetbeat-finagle-test-node-consumer/,172.28.1.112,28300,connect,1480083278
/root/packetbeat-finagle-test-node-consumer/,172.28.1.112,28300,close,1480083278
/root/packetbeat-finagle-test-node-consumer/,172.28.1.112,28302,connect,1480083278
/root/packetbeat-finagle-test-node-consumer/,172.28.1.112,28302,close,1480083278
/root/packetbeat-finagle-test-node-consumer/,172.28.1.112,28306,connect,1480083278
/root/packetbeat-finagle-test-node-consumer/,172.28.1.112,28306,close,1480083278
/root/packetbeat-finagle-test-node-consumer/,172.28.1.112,28308,connect,1480083278
/root/packetbeat-finagle-test-node-consumer/,172.28.1.112,28308,close,1480083278
```

服务提供端端口与程序信息的映射是固定的,服务消费端部署路径与程序信息的映射也可以是固定的

```
{
    "appname": "packetbeat-finagle-test-node-consumer",
    "servicename": "packetbeat-finagle-test-node-consumer",
    "keypath": "packetbeat-finagle-test-node-consumer",
    "type": "thrift",
    "language": "node"
},
{
    "appname": "packetbeat-finagle-test-provider",
    "servicename": "packetbeat-finagle-test-provider-service",
    "port": "20880",
    "keypath": "packetbeat-finagle-test-provider",
    "type": "thrift",
    "language": "java"
},
```

- 由于每个服务都对外暴露不同的端口, 所以根据请求的服务端口, 就可以确定是调用了那个服务

- 将sysdig管道嵌入packetbeat, 可以得到连接创建和关闭事件, 以及发起事件所对应的程序信息, 同时packetbeat可以得到本地连接的端口和事件时间, 根据这些信息可以得到这个端口属于那个本地应用

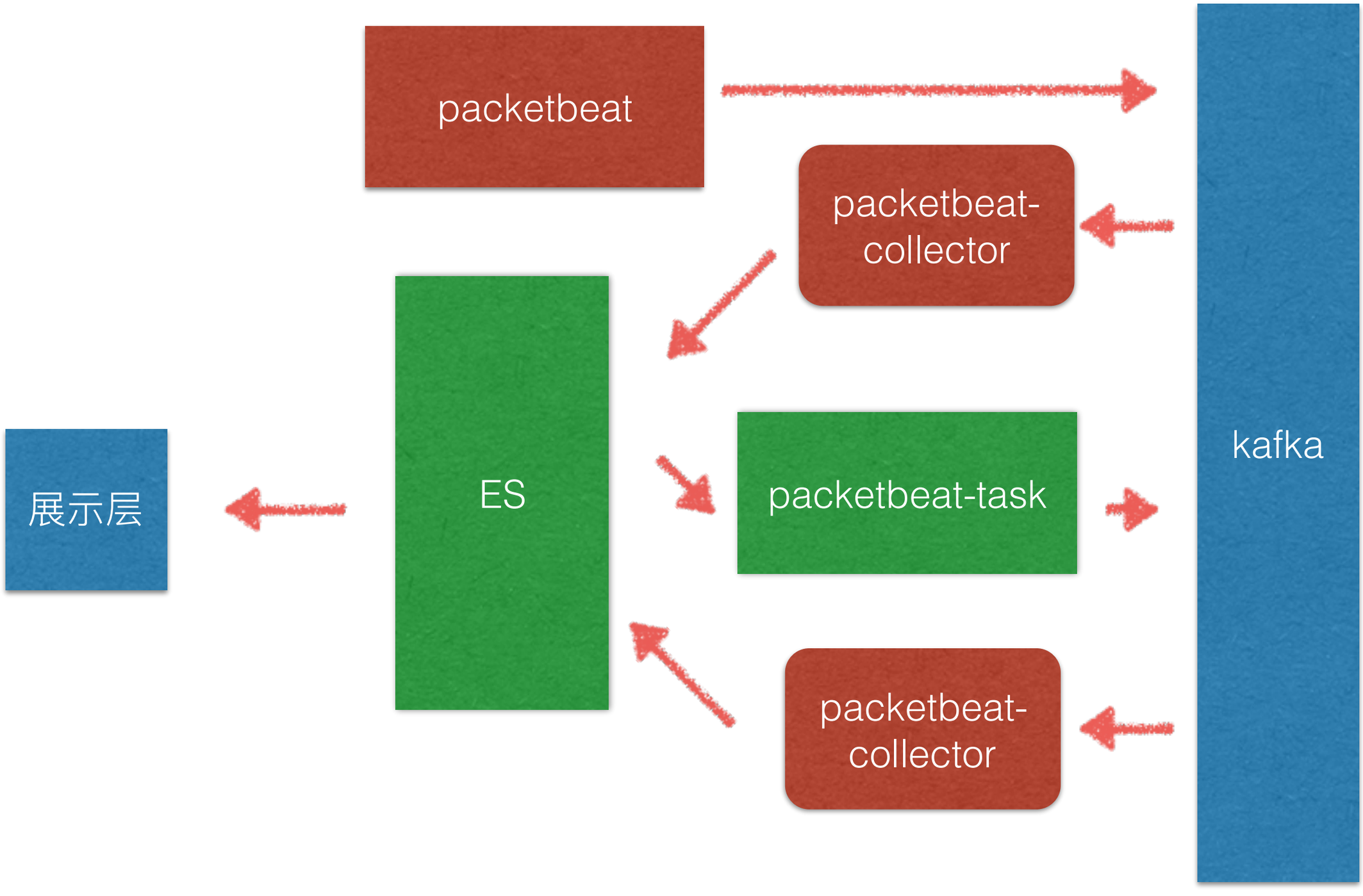
- 但会有个问题, 由于sysdig有时会出现丢事件的问题(1%), 导致会有(1%)流量取不到本地应用信息, 这时就只能降级到ip维度来查看这些数据的信息

- 如果你在用docker, 还可以通过ip来得到应用信息

解析后的数据结构


```
2016/12/09 08:24:43.201411 client.go:189: DBG  Publish: {
  "@timestamp": "2016-12-09T08:24:35.113Z",
  "bytes_in": 60,
  "bytes_out": 79,
  "client_ip": "172.28.1.112",
  "client_port": 42880,
  "consumer_app": "packetbeat-finagle-test-node-consumer",
  "consumer_service": "packetbeat-finagle-test-node-consumer",
  "direction": "out",
  "ip": "192.168.15.26",
  "port": 20880,
  "provider_app": "packetbeat-finagle-test-provider",
  "provider_service": "packetbeat-finagle-test-provider-service",
  "responsetime": 29,
  "thrift": {
    "method": "detail",
    "service": "OrderServ",
    "status": "success"
  },
  "type": "finaglethrift",
  "united": "172.28.1.112|packetbeat-finagle-test-node-consumer|packetbeat-finagle
880|packetbeat-finagle-test-provider|packetbeat-finagle-test-provider-service|Orde
}
```

程序结构


packetbeat
packetbeat-collector
kafka
展示层
ES
packetbeat-task
packetbeat-collector

		总计(次)		平均时长(ms)			[0-200ms](次)		[200-500ms](次)		[500ms-1s](次)		[1
	客户端应用名	成功	失败	成功	失败	超时(次)	成功	失败	成功	失败	成功	失败	成功
1	packetbeat-finagle-test-consumer	2001	0	16.2	0	0	1995	0	2	0	2	0	2
2	packetbeat-finagle-test-consumer	2002	0	35.6	0	0	1986	0	5	0	4	0	3
3	packetbeat-finagle-test-node-consumer	81460	0	18.2	0	0	80483	0	953	0	16	0	4

				总计(次)		平均时长(ms)			[0-200ms](次)		[200-500ms](次)		[500ms-1s](次)		[1s-2s](次)	
	客户端应用名	调用的服务名	调用的方法名	成功	失败	成功	失败	超时(次	成功	失败	成功	失败	成功	失败	成功	失败
1	packetbeat-finagle-test-consumer	packetbeat-fin	detail	200	0	16.2	0	0	1995	0	2	0	2	0	2	0
2	packetbeat-finagle-test-consumer	packetbeat-fin	finaglePing	2002	0	35.6	0	0	1986	0	5	0	4	0	3	0
3	packetbeat-finagle-test-node-consumer	packetbeat-fin	detail	81460	0	18.2	0	0	80483	0	953	0	16	0	4	0

			总计(次)		平均时长(ms)			[0-200ms](次)		[200-500ms](次)		[500ms-1s](次)		[1s-2s](次)	
	调用发起端IP	服务端IP	成功	失败	成功	失败	超时(次)	成功	失败	成功	失败	成功	失败	成功	失败
1	172.28.1.112	192.168.14.109	2001	0	16.2	0	0	1995	0	2	0	2	0	2	0

插点个人思考

- 服务层客户端承载着服务发现,服务质量保护,监控数据上报,调用链监控等多种重要功能, 设计时一定要参考业内先进经验, 并为之后可能的扩展预留空间
- 服务层客户端的开发尽量集中在一处, 即使有多种语言的客户端
- 如果应用较多, 可以把应用程序的信息集中管理起来, 包括应用的应用名, 服务名, 服务对外提供的端口, 程序部署目录, 用到的公共配置, 依赖的其他服务(包括服务信息, 是否可降级), 负责人信息和功能说明等
- 服务化后, 组件不统一是深坑, 入坑需谨慎

总结

- packetbeat抓包可以作为调用链追踪未完善时的过渡, 辅助方案
- 扩展packetbeat协议支持
- 可以使用sysdig根据连接, 得到程序信息

Thanks!