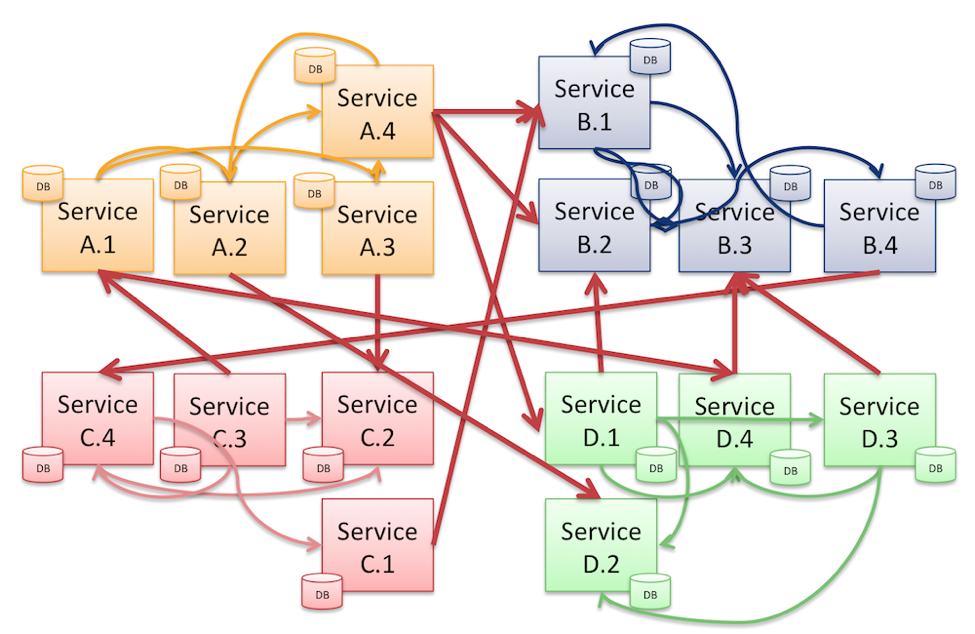
# 基于网络抓包实现kubernetes中微服务的应用级监控

- 一云博客 - 博客频道 - CSDN.NET

http://blog.csdn.net/zqg5258423/article/details/54310932

微服务是什么？

此话题不是本文重点，如你还不知道。请谷歌一波，会有遍地的解释。引用下图说明下[**微服务**](http://lib.csdn.net/base/microservice)可能呈现的形态：   


**微服务监控的挑战**

监控的目的是为了让集群中所有的服务组件，不管是HTTP服务，[**数据库**](http://lib.csdn.net/base/mysql)服务，还是中间件服务。都能够健康稳定得运行，能发现问题，遇到问题能找到原因。   
在过去，监控工具侧重于基础设施或单一软件组件以及衡量运营健康。这些工具在实现这一目标方面只取得了一定的成功，但是对于单一的，传统的应用程序和基础设施来说效果不错。微服务的出现暴露了工具中的弱点。   
现在，组件托管在位于私有云，公共云或两者的混合体之间的虚拟化机器或容器内。获悉我并不需要关心服务cpu用了多少，内存用了多少？确保这些服务相互通信以提供所需的结果需要从监控的角度重要看几件事情：   
1. 微服务集群中是否所有的服务的吞吐率，响应时间都正常？   
2. 服务调用线中哪些线负载过大，哪些线负载过小？   
3. 服务的错误率，例如HTTP 500错误。

我们想要监控分析应用，从它的服务状态出发是否更直接呢？

**已有监控方案**

目前有些厂商提出了微服务的监控解决方案。   
\* 从APM角度监控服务端到端状态。   
\* 为每种类型服务开发agent收集应用状态信息。   
\* 通过产生统一的应用日志分析监控方案   
\* 其他方案

每一种商业或开源方案都有它的优势所在。可以根据你的需求来进行选择。例如你的所有服务都是自己研发，日志标准一致or能够统一处理。所有访问信息都能打出日志，那么我认为日志分析可能是你最适合的方案。但是对于公有云平台，那就不同了。

**好雨云帮采用的方案**

好雨云帮作为公有云平台，平台内部署的服务各式各样。各种通信协议，各种日志标准。我们怎么实现对所有服务的应用状态监控？好雨云帮完善的租户网络，环境隔离。因此我们提供用户在自己环境下安装自己的监控组件，我们的基础数据收集是通过网络分析。下文详细讲解：

**kubernetes POD共享机制**

kubernetes中pod内容器共享网络空间，挂在卷等为我们监控pod内主服务容器提供方便。其实按照官方对pod的定义的使用面来说：

\* content management systems, file and data loaders, local cache managers, etc.

\* log and checkpoint backup, compression, rotation, snapshotting, etc.

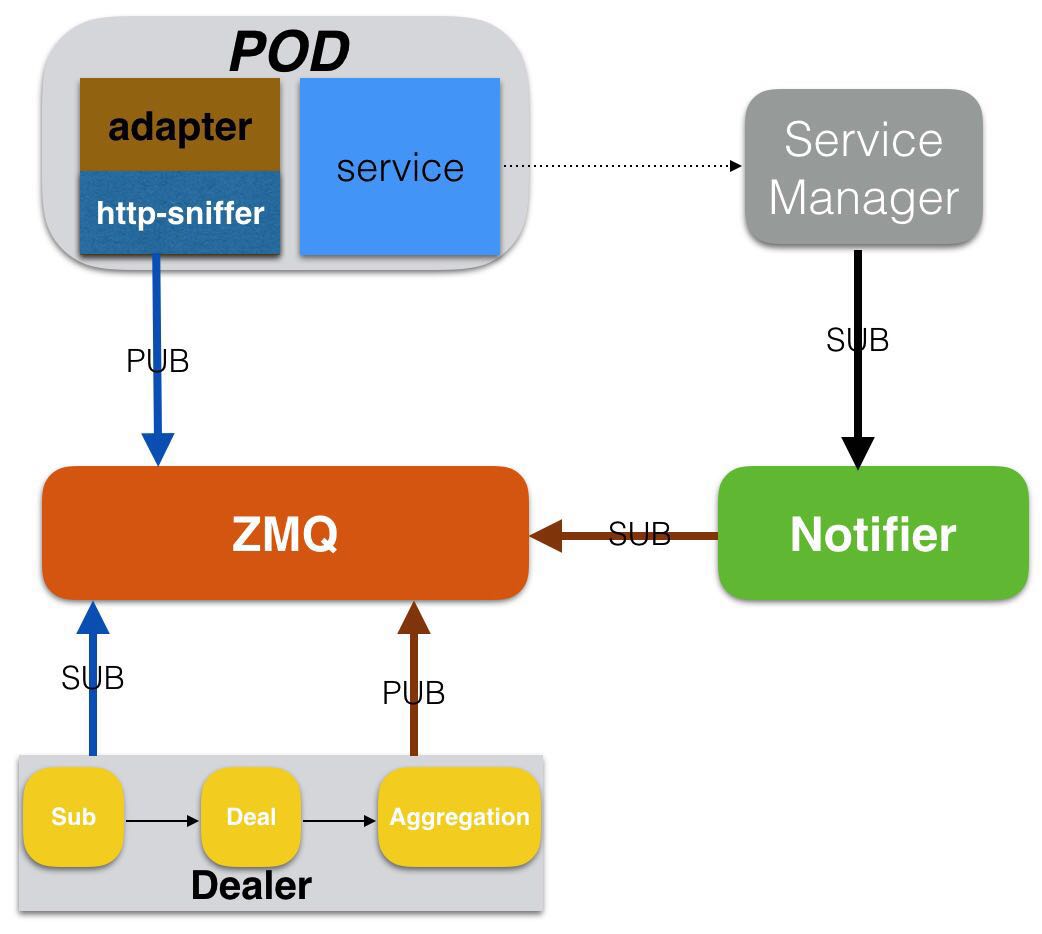
\* data change watchers, log tailers, logging and monitoring adapters, event publishers, etc.

\* proxies, bridges, and adapters

\* controllers, managers, configurators, and updaters

pod内除了主服务外我们可以部署一些附属服务。之前的文章我谈过使用pod的插件服务收集处理日志。今天我再谈使用pod的网络便利监控主服务应用级指标。

**通过共享的网卡抓包分析网络流量反应应用状况**

我们拿一个http服务为例，我们监控网络流量能拿到几乎所有访问和服务返回信息。例如1分钟内多少request,分别请求哪些path,多长时间服务返回了。返回状态码等等信息。   
要获得以上的数据，我们需要获取到网络包，解码网络包然后获得http协议数据。   
我们WatchData服务容器与应用容器在同pod中，经过应用容器eth0网卡的流量我们再WatchData容器中eth0网卡获取。通过解码网络包获取http报文头关键信息，每一个Response形成一个消息发送到server端完成分析，存储然后形成连续的实时的监控数据。下图展现个简要的整体[**架构**](http://lib.csdn.net/base/architecture)图：   
   
当然，上文已经说了，我们采取此方案主要就是为了能够监控各种应用，只是http怎么行。不通的通信应用使用不同的通信协议，比如[**MySQL**](http://lib.csdn.net/base/mysql)的协议，[**MongoDB**](http://lib.csdn.net/base/mongodb)的协议。TCP/IP网络协议栈分为应用层（Application）、传输层（Transport）、网络层（Network）和链路层（Link）四层。。我们抓取到的网络包信息也是四层模型。   
如果同是TCP协议，那么三层协议都一样解码，不同的就是解码Application层。你需要根据不同协议编码形式进行解码，就能获取你想要的数据。下面我们谈谈具

**网络抓包Golang实现**

使用golang实现网络抓包非常容易。得益于谷歌的包：

github.com/google/gopacket

github.com/google/gopacket/layers

github.com/google/gopacket/pcap

* 1
* 2
* 3
* 1
* 2
* 3

这里我举一个监听网卡的Demo主要代码

//device 网卡名

if handle, err := pcap.OpenLive(device, int32(n.Option.Snaplen), true, n.Option.TimeOut); err != nil {

log.With("error", err.Error()).Errorln("PCAP OpenLive Error.")

return 1

} else if err := handle.SetBPFFilter(n.Option.Expr); err != nil { // optional

log.With("error", err.Error()).Errorln("PCAP SetBPFFilter Error.", n.Option.Expr)

return 1

} else {

log.Infoln("Start listen the device ", device)

packetSource := gopacket.NewPacketSource(handle, handle.LinkType())

go func(close chan struct{}, h \*pcap.Handle) {

for {

select {

case packet := <-packetSource.Packets():

n.handlePacket(packet) // Do something with a packet here.

case <-close:

log.Infoln("stop listen the device.")

h.Close()

return

}

}

}(n.Option.Close, handle)

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23

这段代码就是监听某个网卡，通过n.Option.Expr规则过滤点无用网络包，规则语法与[**Linux**](http://lib.csdn.net/base/linux) tcpdump一样。参考：[PCAP-FILTER](http://www.tcpdump.org/manpages/pcap-filter.7.html)   
接收到网络包一般有多种类型：2层模型的包，和4层模型的包。如果你不关注tcp握手这种类型的包你只需要关注具有四层模型的网络包。   
n.handlePacket(packet)处理网络包。

app := packet.ApplicationLayer()

if app != nil {

//log.With("type", app.LayerType().String()).Infoln("Receive a application layer packet")

//log.Infoln(packet.String())

go func() {

sd := &SourceData{

Source: app.Payload(),

ReceiveDate: packet.Metadata().Timestamp,

}

tran := packet.TransportLayer()

if tran != nil {

src, dst := tran.TransportFlow().Endpoints()

sd.SourcePoint = &src

sd.TargetPoint = &dst

if tran.LayerType().Contains(layers.LayerTypeTCP) {

tcp := &layers.TCP{}

err := tcp.DecodeFromBytes(tran.LayerContents(), gopacket.NilDecodeFeedback)

if err != nil {

log.With("error", err.Error()).Errorln("Decode bytes to TCP error")

} else {

sd.TCP = tcp

}

}

}

netL := packet.NetworkLayer()

if netL != nil {

src, dst := packet.NetworkLayer().NetworkFlow().Endpoints()

sd.SourceHost = &src

sd.TargetHost = &dst

}

decode := FindDecode(n.Option.Protocol)

if decode != nil {

decode.Decode(sd)

} else {

log.Debugf("%s protol can not be supported \n", n.Option.Protocol)

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15
* 16
* 17
* 18
* 19
* 20
* 21
* 22
* 23
* 24
* 25
* 26
* 27
* 28
* 29
* 30
* 31
* 32
* 33
* 34
* 35
* 36
* 37

如上代码简单处理四层模型网络包。一般你可以从网络层获取双方ip地址，从传输层获取双方端口以及tcp包的相关信息。从应用层获取应用数据。   
具体的怎么优化和实践就留给大家自己尝试吧。

**网络抓包监控的优缺点**

优点：   
1. 应用无关性，监控工具通用性强。   
2. 数据全面性，你可以获取很多直接和间接反应应用状态的数据。   
3. 不侵入代码，一般不影响网络。   
4. 高并发下不影响应用。

缺点：   
1. 资源消耗，抓包分析包是一个物理资源消耗的过程。   
2. 需要自己开发。

总之，就像上文说得一样。如果你的需求只是想监控一个应用。你就别考虑这个方案了。如果你想监控集群中所有应用，你可以尝试。