Docker日志收集最佳实践

摘要： 云栖TechDay31期，阿里云容器服务技术专家戒空给大家带来Docker日志收集最佳实践的演讲。本文主要从传统日志处理开始谈起，接下来着重分析Docker日志处理，包括stdout和文件日志，其中还有fluentd-pilot，接着分享了日志存储方案Elasticsearch、graylog2和SLS，最后对正确写日志给出了建议。

云栖TechDay31期，阿里云容器服务技术专家戒空给大家带来Docker日志收集最佳实践的演讲。本文主要从传统日志处理开始谈起，接下来着重分析Docker日志处理，包括stdout和文件日志，其中还有fluentd-pilot，接着分享了日志存储方案Elasticsearch、graylog2和SLS，最后对正确写日志给出了建议。

以下是精彩内容整理：

传统日志处理

说到日志，我们以前处理日志的方式如下：

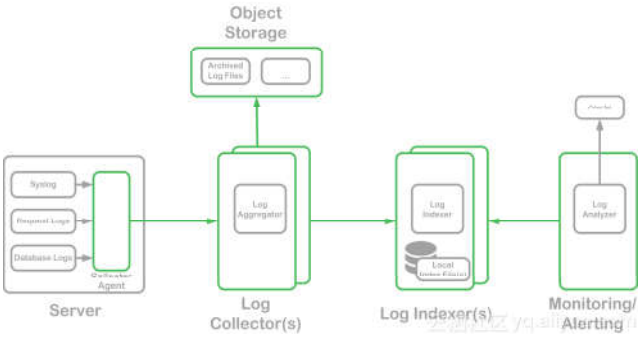
- 日志写到本机磁盘上
- 通常仅用于排查线上问题，很少用于数据分析
- 需要时登录到机器上，用grep、awk等工具分析

那么，这种方式有什么缺点呢？

第一， 它的效率非常低，因为每一次要排查问题的时候都要登到机器上去，当有几十台或者是上百台机器的时候，每一台机器去登陆这是一个没办法接受的事情，可能一台机器浪费两分钟，整个几小时就过去了。

第二， 如果要进行一些比较复杂的分析，像grep、awk两个简单的命令不能够满足需求时，就需要运行一些比较复杂的程序进行分析。

第三， 日志本身它的价值不光在于排查一些系统问题上面，可能在一些数据的分析上，可能利用日志来做一些用户的决策，这也是它的价值，如果不能把它利用起来，价值就不能充分的发挥出来。



所以，现在很多公司会采用集中式日志收集的日志处理方式，我们会把日志分布式收集，集中来存储，我们会在所有机器上面把日志都收集起到一个中心，在中心里面做一个日志全文索引搜索，可以通过一个界面去查询，同时这个日志系统后端可以对接一些更复杂的数据处理系统，可以对接监控、报警系统，对接数据挖掘数据分析系统，充分发挥日志的价值。

Docker的日志处理

使用过Docker的人尤其是使用过容器编排系统，比如说我们的容器服务，可能已经注意到这样的一些特点：

容器编排跟传统的布置方式是不一样的，在容器编排里面，资源分配应用跑到哪台机器上面的决策是由容器层来做的，所以你事先不知道你的容器应用会跑到哪台机器上面；还有自动伸缩，根据负载自动增加或者减少容器数量；另外，在整个运行过程中，系统发生一些情况时，比如说你的容器宕掉了，容器服务会自动把容器应用迁到其他的机器上去，整个过程非常动态，如果像传统方式去配制日志的收集工具，从一台机器上面收集某一个应用，在这个动态下面，很难用原来的方式去配置。

基于这些特点，在Docker的日志里面， 我们只能够采用中心化的日志收集方案，你已经没办法再像原来登到一台机器上面去看它的日志是什么，因为你不知道它其实在哪个机器上面。

stdout和文件日志

Docker的日志我们可以把它分成两类，一类是stdout标准输出，另外一类是文件日志。stdout是写在标准输出里面的日志，比如你在程序里面，通过print或者echo来输出的时候，这种输出标准在linux上面其实是往一个ID为零的文件表述书里面去写；另外的就是文件日志，文件日志就是写在磁盘上的日志，一般来说我们会在传统的应用里面会用得多一些。

stdout

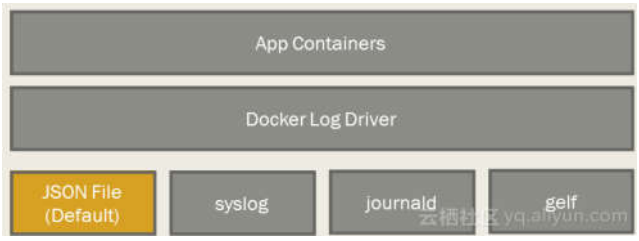
```go
package pilot

import (
    "bufio"
    "os/exec"
    "os"
    "fmt"
)

func main(){
    cmd := exec.Command("cmd", "args")
    stdout, err := cmd.StdoutPipe()
    if err != nil {
        panic(err)
    }
    cmd.Start()
    r := bufio.NewReader(stdout)
    for {
        line, _, err := r.ReadLine()
        if err != nil {
            os.Exit(0)
        }
        fmt.Println(line)
    }

}
```

在Docker的场景里面，目前比较推崇这种标准输出的日志，标准输出日志具体过程如图。标准输出日志的原理在于，当在启动进程的时候，进程之间有一个父子关系，父进程可以拿到子进程的标准输出。拿到子进程标准输出的后，父进程可以对标准输出做所有希望的处理。



例如，我们通过exec.Command启动了一个命令，带一些参数，然后就可以通过标准的pipeline拿到标准输出，后面就可以拿到程序运行过程中产生标准输出。 Docker也是用这个原理来拿的，所有的容器通过Docker Daemon启动，实际上属于Docker的一个子进程， 它可以拿到你的容器里面进程的标准输出，然后拿到标准输出之后，会通过它自身的一个叫做LogDriver的模块来处理，LogDriver就是Docker用来处理容器标准输出的一个模块。 Docker支持很多种不同的处理方式，比如你的标准输出之后，在某一种情况下会把它写到一个日志里面，Docker默认的JSON File日志，除此之外，Docker还可以把它发送到syslog里面，或者是发送到journald里面去，或者是gelf的一个系统。

怎么配置log driver呢?

用Docker来启动容器的话，你有两种方式来配置LogDriver:

```
dockerd \
  --log-driver=syslog \
  --log-opt syslog-address=192.168.1.3:514 \
```

第一种方式是在Daemon上配置，对所有的容器生效。你配置之后，所有的容器启动，如果没有额外的其他配制，默认情况下就会把所有容器标准输出全部都发送给Syslog服务，这样就可以在这个Syslog服务上面收集这台机器上的所有容器的标准输出;

```
docker run -p 80:80\
  --log-driver=syslog \
  --log-opt syslog-address=192.168.1.3:514 \
  nginx:latest
```

第二种方式是在容器上配置，只对当前容器生效。如果你希望这个配置只对一个容器生效，不希望所有容器都受到影响，你可以在容器上面配置。启动一个容器，单独配置它自身使用的logdriver。

驱动	描述
none	丢弃容器输出，docker logs命令也看不到任何内容。
json-file	默认驱动，使用json文件保存日志。
syslog	日志写到syslog里
journald	日志发送到journald（systemd）
gelf	以gelf（Graylog Extended Log Format）格式发送日志
fluentd	日志发送给fluentd
awslogs	日志发送给Amazon CloudWatch Logs.
splunk	日志发送给splunk
etwlogs	日志发送给Event Tracing for Windows，仅支持windows平台。
gcplogs	日志发给Google日志系统
nats	日志发送给nats服务

其实Docker之前已经支持了很多的logdriver，图中列表是直接从Docker的官方文档上面拿到的。

文件日志

对于stdout的这种日志，在Docker里面现在处理起来还是比较方便的，如果没有现成Logdriver的也可以自己实现一个，但是对于文件日志处理起来就没有这么简单了。如果在一个容器里面写了日志，文件位于容器内部，从宿主机上无法访问，的确你是可以根据Docker用的devicemapper、overlayfs访问到它里面的一个文件，但是这种方式跟Docker的实现机制是有关系的，将来它如果改变，你的方案就失效了；另外，容器运行非常动态，日志收集程序难以配置，如果有一个日志收集的程序，在机器上面配置要收集哪个文件，它的格式是什么样子的、发送到哪儿？因为一台机器上面容器是一直在动态变的，它随时可能在增加一个或者删除一个，事先你并不知道这台机器上会跑了多少个容器，他们的配置是怎么样子的，他们的日志是写在哪儿的，所以没办法预先在一台机器上面把这个采集程序配好，这就是文件收集比较难的两个地方。

最简单的一个方案，给每个容器弄一个日志采集进程，这个进程跑到容器里面，就可以解决以上的两个问题，第一因为它跑到容器里面，就可以访问到容器里面所有的文件，包括日志文件；第二它跟容器在一起，当容器启动的时候，收集日志的进程也启动了，当容器销毁的时候，进程也就被销毁掉了。

这个方案非常简单，但是其实会有很多的缺点：

第一， 因为每个容器都有一个日志的进程，意味着你的机器上面有100个容器，就需要启动一百个日志设备的程序，资源的浪费非常厉害。

第二， 在做镜像的时候，需要把容器里面日志采集程序做到镜像里面去，对你的镜像其实是有入侵的，为了日志采集，不得不把自己的日志程序再做个新镜像，然后把东西放进去，所以对你的镜像过程是有入侵性的。

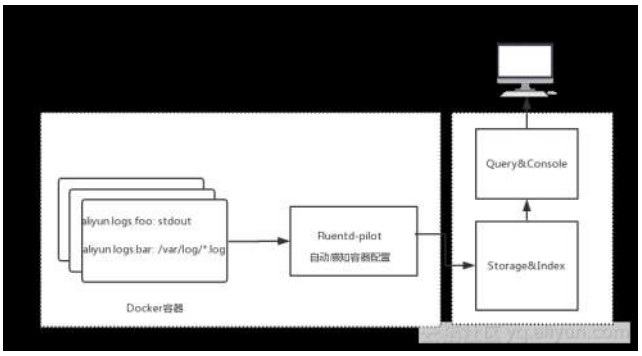
第三， 当一个容器里面好多个进程的时候，对于容器的资源管理，会干扰你对容器的资源使用的判断，包括对于在做资源分配和监控的时候，都会有一些这样的干扰。

fluentd-pilot

在容器服务上面，我们新开发了一个工具，称之为fluentd-pilot。

fluentd-pilot是一个开源的日志采集工具，适合直接在一台机器上面跑单个进程模式。fluentd-pilot有这样的一些特点：

- 一个单独fluentd进程，收集机器上所有容器的日志。不需要为每个容器启动一个fluentd进程；
- 声明式配置。使用label声明要收集的日志文件的路径；
- 支持文件和stdout；
- 支持多种后端存储：elasticsearch, 阿里云日志服务, graylog2...



具体是怎么做呢？如图，这是一个简单的结构，在Docker宿主机上面部署一个fluentd-pilot容器，然后在容器里面启动的时候，我们要声明容器的日志信息，fluentd-pilot会自动感知所有容器的配置。每次启动容器或者删除容器的时候，它能够看得到，当看到容器有新容器产生之后，它就会自动给新容器按照你的配置生成对应的配置文件，然后去采集，最后采集回来的日志同样也会根据配置发送到后端存储里面去，这里面后端主要指的elasticsearch或者是SLS这样的系统，接下来你可以在这个系统上面用一些工具来查询等等。整个这一块在Docker宿主机上面，外面的就是外部系统，由这两个部分来组成。

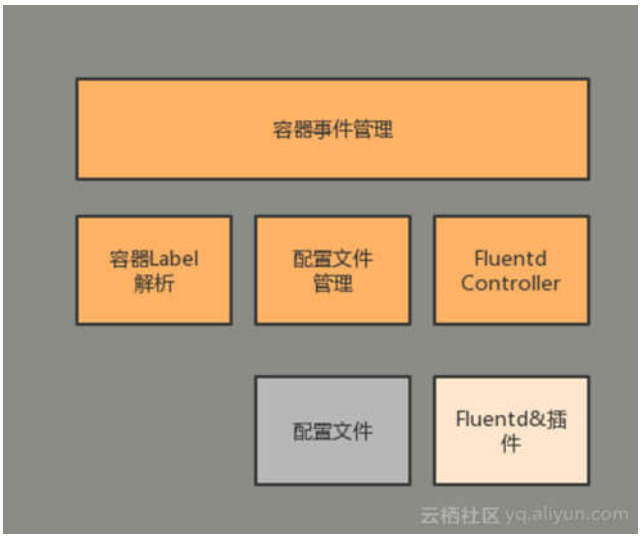
```
docker run -d \
        -v /var/run/docker.sock:/var/run/docker.sock \
        -v /:/host \
        -e FLUENTD_OUTPUT=elasticsearch \
        -e ELASTICSEARCH_HOST=${ELASTICSEARCH_HOST} \
        -e ELASTICSEARCH_PORT=${ELASTICSEARCH_PORT} \
        registry.cn-hangzhou.aliyuncs.com/acs-sample/fluentd-pilot:0.1
```

我们既然要用fluentd-pilot，就得先把它启动起来。还要有一个日志系统，日志要集中收集，必然要有一个中间服务去收集和存储，所以要先把这种东西准备好，然后我们在每一个收集日志的机器上面部署一个fluentd-pilot，用这个命令来部署，其实现在它是一个标准的Docker镜像，内部支持一些后端存储，可以通过环境变量来指定日志放到哪儿去，这样的配置方式会把所有的收集到的日志全部都发送到elasticsearch里面去，当然两个管挂载是需要的，因为它连接Docker，要感知到Docker里面所有容器的变化，它要通过这种方式来访问宿主机的一些信息。

```
docker run -it --rm -p 10080:8080 \
        -v /usr/local/tomcat/logs \
        --label aliyun.logs.catalina=stdout \
        --label aliyun.logs.access=/usr/local/tomcat/logs/localhost_access_log.*.txt \
    tomcat
```

配置好之后启动应用，我们看应用上面要收集的日志，我该在上面做什么样的声明？关键的配置有两个，一是label catalina，声明的时候要收集容器的日志，所有的名字都可以；二是声明access，这也是个名字，都可以用你喜欢的名字。这样一个路径的地址，当你通过这样的配置来去启动fluentd-pilot容器之后，它就能够感觉到这样一个容器的启动事件，它会去看容器的配置是什么，要收集这个目录下面的文件日志，然后告诉fluentd-pilot去中心配置并且去采集，这里有一个-V，实际上跟Logs是一致的，在容器外面实际上没有一种通用的方式能够获取到容器里面的文件，所有我们主动把目录从宿主机上挂载进来，这样就可以在宿主机上看到目录下面所有的东西。

除了最简单的场景之外，你的日志可能会有一些更复杂的特性，比如你的日志格式是什么样子，你可能希望在收集之后加一些内容，便于搜索，当你在真用的时候，它不光是一个非常简单的容器，它可能属于某一个业务或者属于某一个应用，那么，你希望在收集的时候能够有一些关联信息，所以你可以指定日志格式是什么样子，然后可以在日志里添加tag，这些tag相当于一些关键信息，可以附加任何需要的关联信息，这样将来在搜索的时候可以更方便的把这些日志聚在一块；而且，它可以指定很多的后端，fluetnd-pilot支持多种后端，使用环境变量FLUENTD_OUTPUT指定后端类型。



fluent-pilot已经开源，如果功能不满足需求，可以自己定制，自己修改代码实现需要的功能。它的结构比较简单，有这样几个模块：

最上层是容器事件管理，这一块跟Docker进行交互，它会感知Docker的创建容器，然后做出相应的生成配置或者清理配置上的事情；解析容器label跟容器配置，当你创建一个新容器之后，就会用这个模块拿到新容器的配置，然后生成对应的配置文件；FluentdController主要是用来维护对应进程，包括控制什么时候加载新配置，然后检测一些健康状态等等；再下面就是Fluentd的一些插件，如果你需要增加一些日志的后端，就可以自己实现一些插件，放在这个里面，然后再生成跟对应的插件相关的一些配置。

fleuntd-pilot+阿里云容器服务

以上是fleuntd-pilot本身的一些能力，现在你可以在任何地方使用它，但是在容器服务上面我们针对它做了一些更加灵活方便、更酷的一些事情，容器服务为fluentd-pilot进行优化：

第一， 自动识别aliyun.logs标签，并创建Volume；

第二， 重新部署，新容器自动复用已有的Volume，避免日志丢失。

原生支持SLS

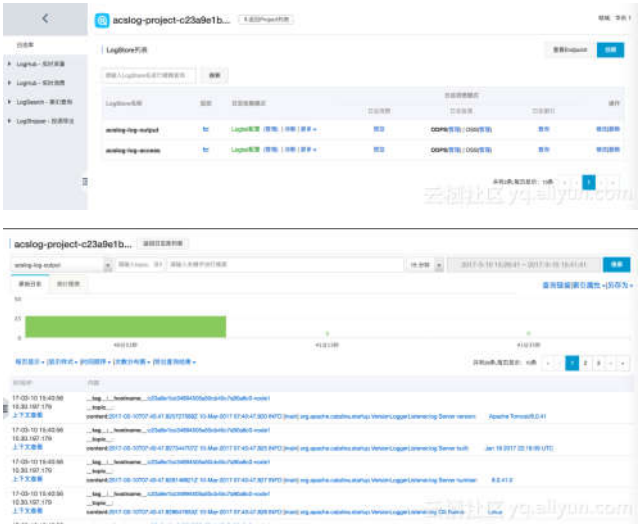
容器服务有一个很棒的特点，它会跟其他的云产品做一些非常方便的集成，这对于用户来说，在使用容器服务的时候，云产品能够更加方便的使用。比如说在日志方面，阿里云容器服务专为SLS做了优化，让用户更简单的在容器服务上使用SLS。SLS是阿里云提供的日志服务，性能强悍，使用方便，还可以对接ODPS等数据系统；支撑1W台物理机，一天12TB日志数据，IOPS>= 2W，采集平均<1 S;单机：在1个CPU core情况下，可以实时采集15-18MB/S 日志量；如果配置中增加可以用线程数目，可水平扩展；是阿里云环境下的最佳日志方案。

优化优点具体体现在：自动创建sls的project, logstore；同时支持stdout和文件日志，使用同样的方式配置。

容器服务日志方案

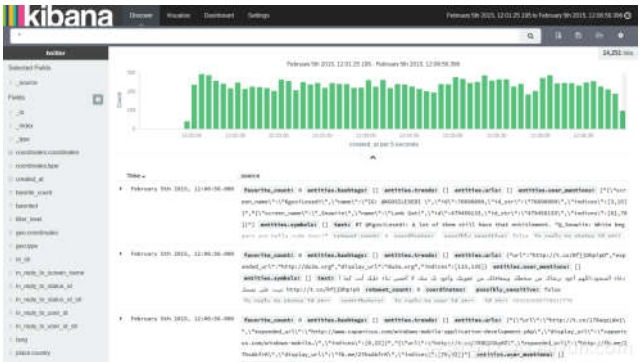
比如在容器服务上面布一个tomcat，可能会写如图的一个标准Docker的模板。

当你通过部署之后，就可以在日志服务上面看到会生成两个东西，都是创建好的，加一些前缀来区分，不用管一些配置，你唯一要做的事是什么呢？点日志索引查询，然后到日志搜索界面，刚才启动的时候一些日志，可以看到从哪过来的，这条日志的内容是什么，这些信息都已经很快的出现了，包括需要检索可以选中一个时间段，输入关健词去做一些搜索，自动在日志服务上创建并配置logstore。

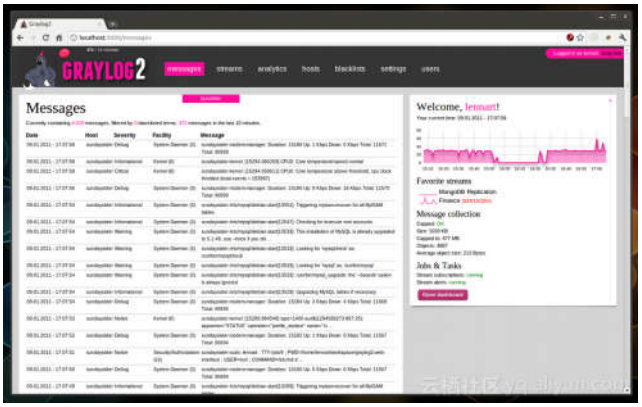


我们在容器服务上面做到了对于文件日志的收集，并且方式也都非常简单，都是你可以设立一个label，通过label方式可以收集到所有的日志。

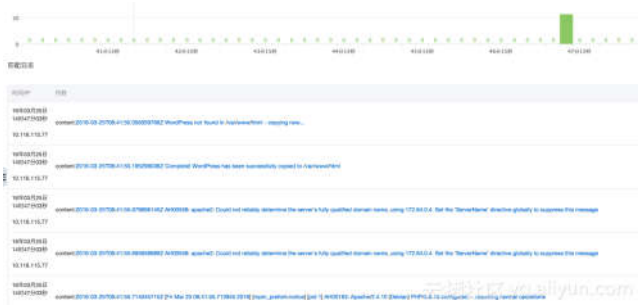
日志存储方案



最后简单的介绍几种日志存储方案的对比，图为现在比较流行的日志方案Elasticsearch，它本身并不提供界面，ELK中的E，基于Lucene，主要用于日志索引、存储和分析；通常配合Kibana展示日志，免费，支持集群模式，可以搭一个Docker用的生产环境可用的系统。



接下来是graylog2，目前不算很流行，但是也是功能很强大的系统，它不像Elasticsearch还需要配合其它去使用，它自身拥有日志存储、索引以及展示所有的功能，都在一个系统里面实现，它可以设置一些报警规则，当日志里面出现一些关键字的时候自动报警，这个功能还是非常有用的，免费、支持集群模式，可以在生产环境里面搭一个Docker用的生产系统。



阿里云的日志服务SLS特点如下：

- 阿里云托管，不需要自己维护
- 支持多用户和权限管理
- 可以对接ODPS等系统
- 支撑1W台物理机，一天12TB日志数据，IOPS>= 2W，采集平均<1 S;单机：在1个CPU core情况下，可以实时采集15-18MB/S 日志量；如果配置中增加可以用线程数目，可水平扩展

用正确的方式写日志

那么，我们怎么样去收集日志、存储日志，用什么样的系统，日志的源头和写日志我们又该怎么来做，有这样几个建议：

1. 选择合适的日志框架，不要直接print；
2. 为每一条日志选择正确的level，该debug的不要用info；
3. 附加更多的上下文信息；
. 使用json、csv等日志格式，方便工具解析；
5. 尽量不要使用多行日志 Java Exception Stack 。