容器日志实践

> 本节是第四部分“架构篇”的第七节，前面几节我主要为你介绍了 Docker 核心组件和 Plugin，以及监控相关的内容。本节，我来为你介绍 Docker 的日志实践。

监控有助于我们及时了解线上服务的运行状况，而日志既可以作为监控的一项“数据源”，又可以作为排查问题的一个关键手段，同时日志还可以后续做离线分析等，日志的重要性不言而喻。

对于一些公司而言，无论是否在做容器化的改造，集中式的日志中心已经作为基础设施提供。

如果现在你还没有建设集中式的日志中心，并且在做容器化的改造，那么我建议你尝试去建设集中式的日志中心。主要考虑如下：

- 容器生命周期比较短，一旦销毁后续问题不易排查；
- 随着规模的扩大，容器的数量也会显著增加，而其生成的日志将会更加庞大，没有集中式的日志中心，逐个去查看容器日志不现实。

在使用 Docker 或者是使用 Kubernetes 等容器编排工具时，在日志方面，我们通常分为两类：

- 基础架构日志
- 应用程序日志

我将分别从这两方面为你介绍。当然在介绍它们之前，我们还是需要聊聊对于日志方案的选择。

日志方案的选择

在《Plugin 扩展》那一节我便为你介绍了通过 `docker info` 命令可以看到 Docker 默认支持的日志相关的 Plugin：

复制

```
Plugins:
 Log: awslogs fluentd gcplogs gelf journald json-file logentries splunk syslog
```

仅从这里你便能看到日志方案有太多选择了，不过除了上面这些外还有很多其他的方案。

最常见的方案是 ELK（Elasticsearch + Logstash + Kibana），正如其官方介绍：Elasticsearch 是一个搜索和分析引擎。Logstash 是服务器端数据处理管道，能够同时从多个来源采集数据，转换数据，然后将数据发送到诸如 Elasticsearch 等“存储库”中。Kibana 则可以让用户在 Elasticsearch 中使用图形和图表对数据进行可视化。

使用 ELK 的组合方案便从日志采集，存储，到展示等功能均覆盖到了，所以这种方案被广泛采用。当然在实际使用时，也会根据实际情况进行调整，比如当日志量很大时，可能会在 Elasticsearch 前增加 Kafka 等来提高系统的吞吐量。

也可以将其中的各类组件替换掉，比如采集日志的 Logstash 可替换为 CNCF 毕业的项目 Fluentd 或者更加轻量级的 Fluent Bit 或 Filebeat，对于这些组件的性能对比或选择不是本文的重点，且选择不同的组件也基本都能完成对容器的日志监控，这里暂且跳过。

基础架构日志

基础架构日志包括三个方面：

- 系统层日志或系统关键服务的日志，比如监控所用的 agent 之类的；
- Docker Daemon 的日志；
- 如果使用了类似 Kubernetes 等容器编排工具时，那 Kubernetes 的系统组件日志也非常关键。

一般情况下，基础架构相关的服务会由系统级的服务管理器（比如：systemd）来管理，并且可以很容易与 syslog 之类的传统方案进行结合。

或者你也可以在上文中提到的各种日志收集器中任选一种，直接收集 `system 的 journal log，并将其发送至你选择的存储中。

例如，Fluent Bit 提供了 用于采集 systemd journal log 的插件，可以使用它来采集相关的日志。

当你按照 Fluent Bit 官方文档的安装指南安装完 Fluent Bit 后，你可以使用如下的配置启动 Fluent Bit 来验证其采集日志的功能：

复制

```
[SERVICE]
    Flush        1
    Log_Level    info

[INPUT]
    Name             systemd
    Tag              moelove.info.*
    Systemd_Filter   _SYSTEMD_UNIT=docker.service

[OUTPUT]
    Name   stdout
    Match  *
```

以上配置中重点的内容是 INPUT 和 OUTPUT 的配置，表示筛选 docker.service 的 unit 并为其添加 moelove.info.* 的 tag，直接全部打印到标准输出。

将配置保存为 c.conf 文件，通过以下命令启动 Fluent Bit（你也可以不用配置文件直接传参数启动）：

复制

```
(MoeLove) ➜  build ./bin/fluent-bit -c c.conf
Fluent Bit v1.3.5
Copyright (C) Treasure Data

[2020/01/08 23:54:39] [ info] [storage] initializing...
[2020/01/08 23:54:39] [ info] [storage] in-memory
[2020/01/08 23:54:39] [ info] [storage] normal synchronization mode, checksum disa
[2020/01/08 23:54:39] [ info] [engine] started (pid=13664)
[2020/01/08 23:54:39] [ info] [sp] stream processor started
[0] moelove.info.docker.service: [1578498892.104022000, {"SYSLOG_FACILITY"=>"3", "
```

会看到类似上面的输出，表示已经可以通过 Fluent Bit 采集 Docker Daemon 的日志了。至于输出到 ES 或者 Kafka 或者其他的存储，这就需要按你实际情况选择了。

以上介绍了使用 Fluent Bit 采集 Docker Daemon 日志的一种方式，对于系统中的其他日志或是在使用 Kubernetes 等容器编排系统时，也可以使用类似的方式进行基础架构日志的采集。

应用程序日志

当我们的应用程序使用容器部署运行时，采集应用程序的日志也同样重要。

这时，我们有了更多的选择：

- 应用程序直接将日志写入到日志中心
- 应用程序将日志输出到 stdout/stderr
- 应用程序将日志写到某个固定目录

以上是三种比较常见的方式，其中最简单的便是直接将日志写入到日志中心，这种情况下，便不再需要做什么额外的操作了。

应用程序将日志写到某个固定目录

这种情况下，通常我们会选择两种方式来处理：

1. 使用挂载卷的方式，应用程序将日志写到其挂载的目录中，然后在外部通过日志采集器进行日志的收集。

这种方式相对而言，略显繁琐了一些。

2. 在容器内同时启动日志采集器，对日志进行收集。

这种方式相对简单，但每个容器均需要有对应的采集器，会造成一定的资源浪费。

应用程序将日志输出到 stdout/stderr

这种情况下，不需要做什么特殊的处理，Docker 便会自动捕获到这些日志了。并且 Docker 也提供了比较完备的支持，比如默认的日志驱动 json-file，你可以在 /etc/docker/daemon.json 中为其添加相关配置进行日志的管理：

复制

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3",
  }
}
```

如果是使用默认的 json-file 这种驱动时，容器的日志默认在 `/var/lib/docker/containers/${containerID}/${containerID}-json.log` 这个位置，你可以直接使用日志采集器对其进行收集。当使用 Kubernetes 时，也可以直接以 DaemonSet 的方式启动日志采集器，直接采集此位置下的日志。

或者你也可以使用其他的日志插件，比如使用 journald 这个日志驱动时，我们便可以使用类似上面采集 Docker Daemon 的方式进行日志的收集了。比如：

复制

```
# 使用 journald 的日志驱动
(MoeLove) ➜  docker run --name redis --rm -d --log-driver journald redis
d7073824dc3bb58b0558e03602ae44972be29d7abada18551e0dda83ebc43b9b

# 使用 journalctl 查看日志
(MoeLove) ➜  journalctl CONTAINER_NAME=redis
-- Logs begin at Wed 2020-01-08 07:20:55 CST, end at Thu 2020-01-09 00:30:42 CST.
1月 09 00:30:41 localhost d7073824dc3b[1611]: 1:C 08 Jan 2020 16:30:41.557 # o000o
...
```

直接使用 Fluent Bit 采集：

复制

```
(MoeLove) ➜  build ./bin/fluent-bit -i systemd -p systemd_filter=CONTAINER_NAME=r
[0] redis.docker.service: [1578501419.936616000, {"PRIORITY"=>"6", "_UID"=>"0", "_
```

可以看到已经顺利采集到了。

总结

本节，我为你介绍了 Docker 容器日志相关的实践。对于日志采集或者说日志中心的建设，可选的方案太多。但万变不离其宗，Docker 容器日志的采集，只要梳理清楚日志的输入（即：日志的存储位置），日志采集器及日志的输出（日志中心的存储）即可。

下一节，我将为你介绍容器单机编排工具 docker-compose，在单机需要对容器进行编排时 docker-compose 也是比较实用的一个工具。