> 本文由 简悦 SimpRead 转码， 原文地址 www.imooc.com

我们知道在 Kubernetes 中，可以通过 `kubectl logs` 查看 Pod 以及 Pod 中的容器的日志。那么我们怎么把这些日志收集起来方便查看呢？本文就来讨论一下这个问题。

1. 容器日志

首先说明一下容器日志的实现。我们通过容器或者 Docker 启动我们的应用程序的时候，对于打印到 stdout、stderr 里面的容器日志，是保存在宿主机的特定目录下的，路径为 `/var/lib/docker/containers/<container-id>/<container-id>-json.log` 。本文主要讨论的是如何收集这种日志。对于那种使用日志框架，比如 log4j，输出到特定文件的日志，则不会存储到这里。我们先来看一下容器的日志。

首先，我们可以通过 `kubectl describe pod` 查看到 Pod 里面运行的容器的 ID 和 Kubernetes 的 Node 信息。下面是一个 nginx 的 Pod 的描述信息。

```
➜  ~ kubectl get pods -n imooc
NAME                                READY   STATUS      RESTARTS   AGE
fluentd-app-2xjmg                   1/1     Running     0          40d
fluentd-app-6sxz9                   1/1     Running     0          40d
fluentd-app-fknkb                   1/1     Running     0          40d
myapp-pod1                          1/1     Running     811        33d
nginx-deployment-c464767dd-6ts4x    1/1     Running     0          34d
nginx-deployment-c464767dd-d9mh7    1/1     Running     0          34d
nginx-deployment-c464767dd-qd22h    1/1     Running     0          34d
pi-wsgmm                            0/1     Completed   0          39d
➜  ~ kubectl describe pods nginx-deployment-c464767dd-6ts4x -n imooc
Name:           nginx-deployment-c464767dd-6ts4x
Namespace:      imooc
Priority:       0
Node:           cn-beijing.172.16.60.188/172.16.60.188
Start Time:     Sun, 19 Apr 2020 12:55:15 +0800
Labels:         app=nginx
                pod-template-hash=c464767dd
Annotations:    <none>
Status:         Running
IP:             10.1.1.154
IPs:            <none>
Controlled By:  ReplicaSet/nginx-deployment-c464767dd
Containers:
  nginx:
    Container ID:
docker://b73a0e27340246cd900eb217d51dff3a8dc955cdb9d395dc61453fe9ae6c9f3b
    Image:          nginx:1.9.1
    Image ID:       docker-
pullable://nginx@sha256:2f68b99bc0d6d25d0c56876b924ec20418544ff28e1fb89a4c27679a
40da811b
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sun, 19 Apr 2020 12:55:16 +0800
    Ready:          True
    Restart Count:  0
    Limits:
```

```
      cpu:     100m
      memory:  200Mi
    Requests:
      cpu:        100m
      memory:     200Mi
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-84db9
(ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-84db9:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-84db9
    Optional:    false
QoS Class:       Guaranteed
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:          <none>
➜  ~
```

通过上面的输出我们可以看到：

- Node 为 cn-beijing.172.16.60.188；
- Container ID 为 b73a0e27340246cd900eb217d51dff3a8dc955cdb9d395dc61453fe9ae6c9f3b。

然后我们登录到这台宿主机上，查看目录 `/var/lib/docker/containers/b73a0e27340246cd900eb217d51dff3a8dc955cdb9d395dc61453fe9ae6c9f3b` 下确实有一个 b73a0e27340246cd900eb217d51dff3a8dc955cdb9d395dc61453fe9ae6c9f3b-json.log 文件，这个文件就是 nginx 容器输出到标准输出里面的日志。

```
[root@ixxxZ ~]
[root@ixxxZ b73a0e27340246cd900eb217d51dff3a8dc955cdb9d395dc61453fe9ae6c9f3b]
b73a0e27340246cd900eb217d51dff3a8dc955cdb9d395dc61453fe9ae6c9f3b-json.log
checkpoints  config.v2.json  hostconfig.json  mounts
[root@ixxxZ b73a0e27340246cd900eb217d51dff3a8dc955cdb9d395dc61453fe9ae6c9f3b]
{"log":"172.16.60.187 - - [23/May/2020:07:30:53 +0000] \"GET /favicon.ico
HTTP/1.1\" 404 570 \"-\" \"Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36\" \"-
\"\n","stream":"stdout","time":"2020-05-23T07:30:53.2731376Z"}
{"log":"2020/05/23 07:30:53 [error] 6#6: *7104677 open()
\"/usr/share/nginx/html/robots.txt\" failed (2: No such file or directory),
client: 172.16.60.187, server: localhost, request: \"GET /robots.txt HTTP/1.1\",
host: \"39.102.158.120:30005\"\n","stream":"stderr","time":"2020-05-
23T07:30:53.333170652Z"}
```

```
{"log":"172.16.60.187 - - [23/May/2020:07:30:53 +0000] \"GET /robots.txt
HTTP/1.1\" 404 570 \"-\" \"Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36\" \"-
\"\n","stream":"stdout","time":"2020-05-23T07:30:53.333204128Z"}
{"log":"2020/05/23 07:30:53 [error] 6#6: *7104677
\"/usr/share/nginx/html/wcm/index.html\" is not found (2: No such file or
directory), client: 172.16.60.187, server: localhost, request: \"GET /wcm/
HTTP/1.1\", host: \"39.102.158.120:30005\"\n","stream":"stderr","time":"2020-05-
23T07:30:53.445727806Z"}
{"log":"172.16.60.187 - - [23/May/2020:07:30:53 +0000] \"GET /wcm/ HTTP/1.1\" 404
570 \"-\" \"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/59.0.3071.115 Safari/537.36\" \"-
\"\n","stream":"stdout","time":"2020-05-23T07:30:53.445759042Z"}
{"log":"2020/05/23 07:30:53 [error] 6#6: *7104677
\"/usr/share/nginx/html/phpMyAdmin/index.html\" is not found (2: No such file or
directory), client: 172.16.60.187, server: localhost, request: \"GET /phpMyAdmin/
HTTP/1.1\", host: \"39.102.158.120:30005\"\n","stream":"stderr","time":"2020-05-
23T07:30:53.484069758Z"}
{"log":"172.16.60.187 - - [23/May/2020:07:30:53 +0000] \"GET /phpMyAdmin/
HTTP/1.1\" 404 570 \"-\" \"Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36\" \"-
\"\n","stream":"stdout","time":"2020-05-23T07:30:53.484100095Z"}
{"log":"2020/05/23 07:30:53 [error] 6#6: *7104677
\"/usr/share/nginx/html/phpmyadmin/index.html\" is not found (2: No such file or
directory), client: 172.16.60.187, server: localhost, request: \"GET /phpmyadmin/
HTTP/1.1\", host: \"39.102.158.120:30005\"\n","stream":"stderr","time":"2020-05-
23T07:30:53.522257802Z"}
{"log":"172.16.60.187 - - [23/May/2020:07:30:53 +0000] \"GET /phpmyadmin/
HTTP/1.1\" 404 570 \"-\" \"Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36\" \"-
\"\n","stream":"stdout","time":"2020-05-23T07:30:53.522286205Z"}
{"log":"172.16.60.187 - - [23/May/2020:10:31:22 +0000] \"GET / HTTP/1.1\" 200 612
\"-\" \"-\" \"-\"\n","stream":"stdout","time":"2020-05-23T10:31:22.835824159Z"}
```


弄清楚日志的存储那么我们就可以讨论日志的采集了，主要有三种方式：

- 在 Node 上部署日志采集 agent 来采集；
- 借助于 sidecar 容器来做日志转发存储；
- 借助于 sidecar 容器直接采集日志。

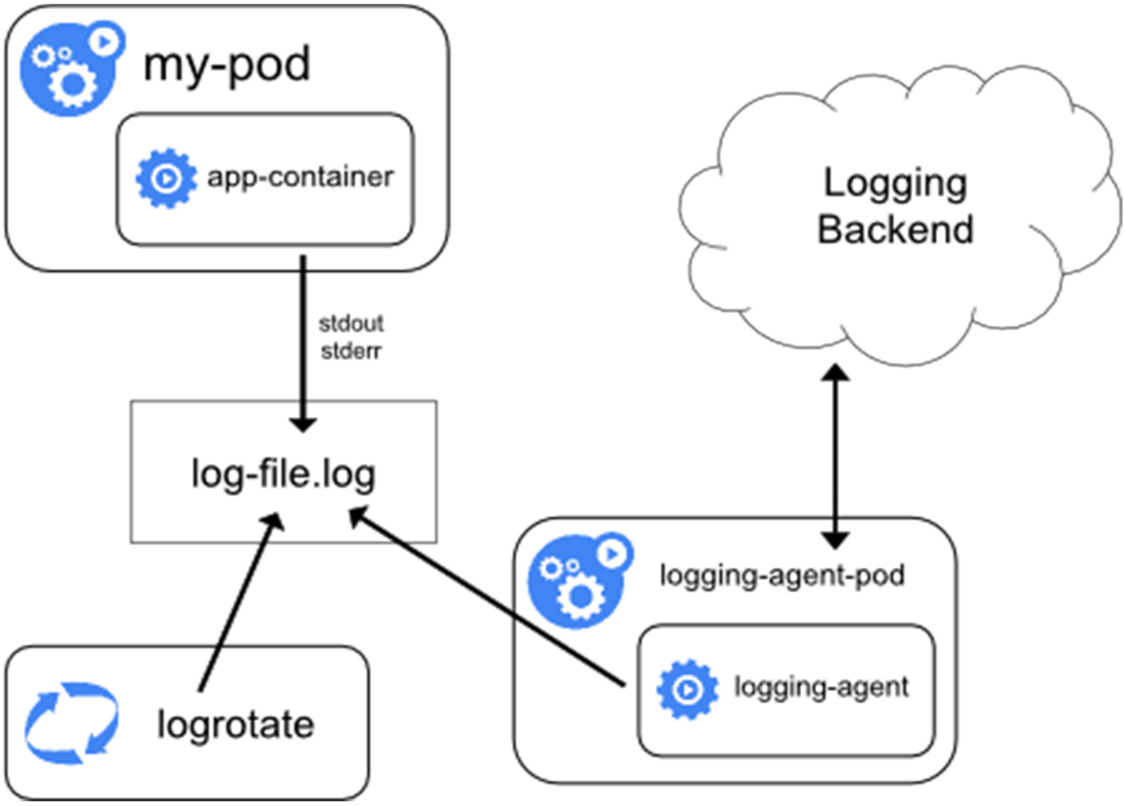
下面我们具体来看一下。

2. Node 上部署日志采集 agent

这种方案简而言之，由于容器的日志目录都是固定的 `/var/lib/docker/containers/<container-id>/<container-id>-json.log`，那么我们可以将目录 `/var/lib/docker/containers` 挂载到 agent 的 Pod 里面，同时还需要 agent 支持日志文件的特定模式匹配。由于 agent 需要在每个 Node 上都运行，所以我们可以以 DaemonSet 的方式来运行 agent。

举个社区的例子，Fluentd，作为日志采集 agent 部署在 Node 上，然后把日志转发到远端存储，比如 ElasticSearch，然后配合 Kibana 等做日志查询。

方案架构图如下。我们可以看到架构中有一个组件叫 logrotate 我们没有讨论到，这个组价主要是用来防止容器日志过大。在很多 Kubernetes 的部署中，会自动启动 logrotate，在日志文件超过 10MB 的时候自动对文件进行 rotate 操作。



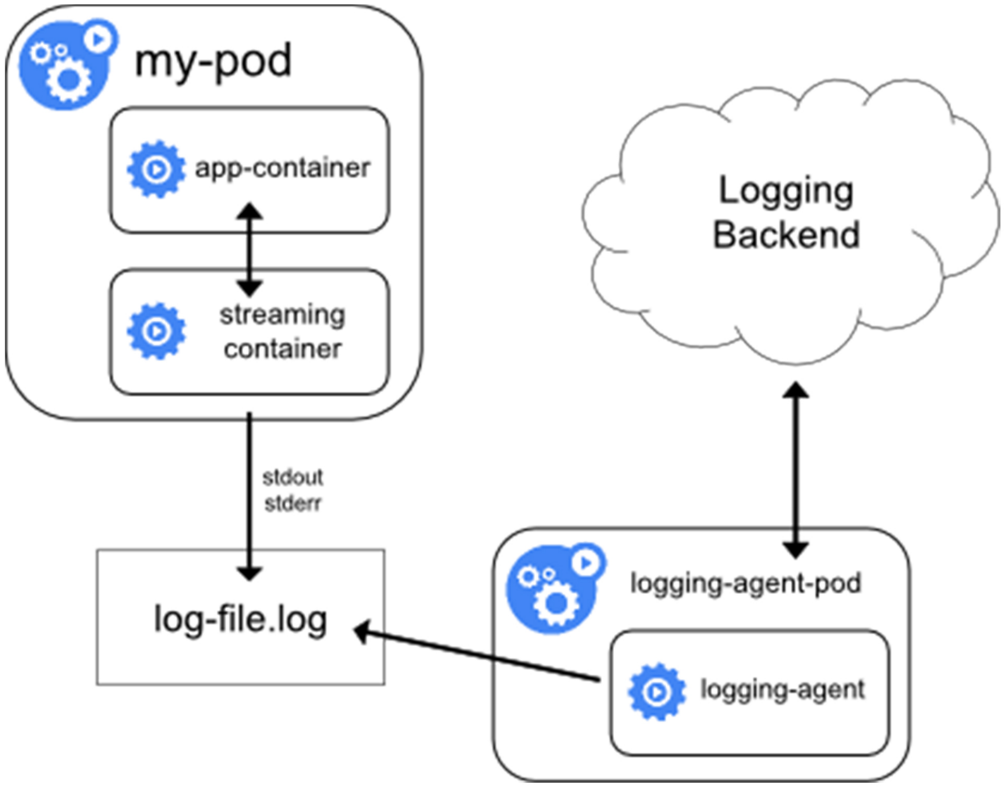
那么这种方案有什么样的优缺点呢?

- 优点：每个 Node 上指需要部署一个 agent，并且 agent 是采集宿主机的特定目录，所以不会对 Pod 有任何的侵入性;
- 缺点：缺点主要是要求容器的日志全部输出到 stdout 和 stderr，否则收集不到。

3. 借助于 sidecar 容器来做日志转发存储

第二种日志采集方式正是对第一种的缺点的补充。简单来说，当容器的日志不是输出到 stdout 或者 stderr 时，而是输出到特定的文件中，我们通过 sidecar 容器来将这些特定文件中的日志转发到 stdout 和 stderr 中。这里可能需要解释一下 sidecar，中文一般翻译成边车，可以理解成一个辅助工具。关于 sidecar，有一个比较形象的表示如下图。其中左边就是 sidecar。

第二种日志采集方式的架构图也比较简单，如下图。



为了更直观的理解这种采集方式，我们直接看下面的这个 Pod 的示例。这个 Pod 中包含两个容器：count 和 count-log-1，其中 count-log-1 相当于 sidecar 容器。count 容器会向 `/var/log/1.log` 和 `/var/log/2.log` 中不断输出日志，我们的目的就是要采集这些日志。

其中 sidecar 容器 count-log-1 通过 `tail -f` 的方式将主容器的日志重定向到 stdout 中，然后我们就可以继续使用第一种方式进行采集了。

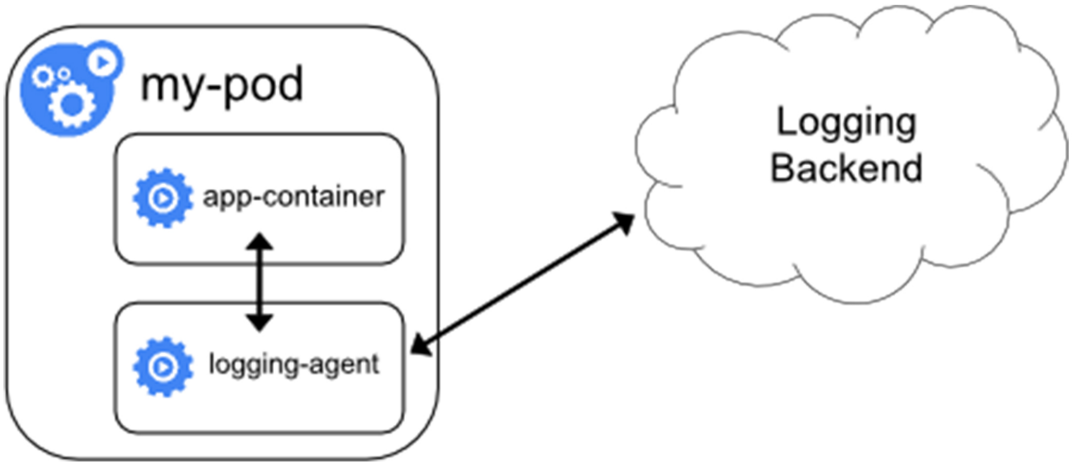
```
apiversion: v1
kind: Pod
```

```
metadata:
  name: counter
spec:
  containers:
  - name: count
    image: busybox
    args:
    - /bin/sh
    - -c
    - >
      i=0;
      while true;
      do
        echo "$i: $(date)" >> /var/log/1.log;
        echo "$(date) INFO $i" >> /var/log/2.log;
        i=$((i+1));
        sleep 1;
      done
    volumeMounts:
    - name: varlog
      mountPath: /var/log
  - name: count-log-1
    image: busybox
    args: [/bin/sh, -c, 'tail -n+1 -f /var/log/1.log']
    volumeMounts:
    - name: varlog
      mountPath: /var/log
  - name: count-log-2
    image: busybox
    args: [/bin/sh, -c, 'tail -n+1 -f /var/log/2.log']
    volumeMounts:
    - name: varlog
      mountPath: /var/log
  volumes:
  - name: varlog
    emptyDir: {}
```

这种日志采集方式确实是解决了第一种日志采集方式的问题，当然也要付出一定的代码：日志存了双份，一份是 `/var/log` 下，一份是 `/var/lib/docker/containers` 目录下，当日志比较多的时候，这个问题对磁盘的占用就会被放大，凸显出来。这个时候就引入了下面的第三种方案。

4. 借助于 sidecar 容器直接采集日志

双向第二种方案，sidecar 容器既然已经存在了，为什么不让它直接做更多的事情呢？比如把日志直接发送到远端存储。这就是第三种日志采集方案的思想所在。



这种方式相当于把第一种方案中的日志采集 agent 作为 sidecar 跑在了 Pod 里面，由于我们的日志采集 agent 一般都是标准化的，比如 Fluentd，我们不太可能为每一种不同的日志输出模式都修改我们的 agent 代码来适配。那么怎么解决不同应用或者容器的日志输出路径不一样的问题呢?

其实这个问题的解决也很简单，将日志路径作为采集 agent 的一个参数即可。比如通过 ConfigMap 来配置。为了更直观的理解这种采集方式，下面给出来了一个对应的 ConfigMap 和 Pod 的示例。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluentd-config
data:
  fluentd.conf: |
    <source>
      type tail
      format none
      path /var/log/1.log
      pos_file /var/log/1.log.pos
      tag count.format1
    </source>

    <source>
      type tail
      format none
      path /var/log/2.log
      pos_file /var/log/2.log.pos
      tag count.format2
    </source>

    <match **>
      type google_cloud
    </match>
```

```
apiVersion: v1
kind: Pod
metadata:
  name: counter
spec:
  containers:
```

```
  - name: count
    image: busybox
    args:
    - /bin/sh
    - -c
    - >
      i=0;
      while true;
      do
        echo "$i: $(date)" >> /var/log/1.log;
        echo "$(date) INFO $i" >> /var/log/2.log;
        i=$((i+1));
        sleep 1;
      done
    volumeMounts:
    - name: varlog
      mountPath: /var/log
  - name: count-agent
    image: k8s.gcr.io/fluentd-gcp:1.30
    env:
    - name: FLUENTD_ARGS
      value: -c /etc/fluentd-config/fluentd.conf
    volumeMounts:
    - name: varlog
      mountPath: /var/log
    - name: config-volume
      mountPath: /etc/fluentd-config
  volumes:
  - name: varlog
    emptyDir: {}
  - name: config-volume
    configMap:
      name: fluentd-config
```

这种采集方式看上去比较简单，但是也要付出一定的代价，由于 sidecar 容器做的事情比较多，需要和远端存储进行日志的读写，很可能会带来性能的开销。同时由于主容器的日志没有输出到 stdout 和 stderr 中，所以我们通过 `kubectl logs` 查看主容器的日志是查看不到的。

5. 总结

在本篇文章中，介绍了三种主流的日志采集方案，各有优劣。但是在实际使用中，还是要结合自己的业务场景，不要拘泥于文章中所说的三种方式。比如主容器的日志直接写到远端存储这种方式其实也是可以考虑的。

}