

容器监控实践

本篇是第四部分“架构篇”的第六篇，前面几篇我主要为你介绍了 Docker 核心组件及使用 Plugin 对其进行扩展。本篇，我来为你介绍 Docker 的监控实践。

当我们在生产环境中使用 Docker 时，监控是非常重要的的一环，通过监控告警我们可以及时了解到 Docker 及容器的运行状况，以便尽早进行应对。

当提到容器监控的时候，其实不只是包含对容器自身的监控，本篇我们分别从以下四个方面来聊聊：

- 主机基础监控
- Docker Daemon 监控
- 容器基础监控
- 容器内应用程序监控

监控方案的选择

可选择的监控方案其实有很多，比如老牌的 Nagios 和 Zabbix 之类的，或是新一些的 Falcon 之类的，其他比较常见的监控方案可以在 [CNCF 的监控全景图中查看](#)，但我个人比较推荐使用 [Prometheus](#)。

稍微写写一点推荐 Prometheus 的理由吧：

- CNCF 毕业项目，在云原生领域地位显著；
- 使用 Pull 的模式，比较灵活；
- prometheus metrics 已经被很多软件/社区接受，也都提供了良好支持；
- 方便扩展。

在使用 Prometheus 时，数据源是各类 exporter（或者说是各种暴露出来的 metrics），使用 Prometheus 去抓取这些 metrics 即可。

对于 Prometheus 更详细的介绍及其使用，请直接查看其[官方文档](#)了解，这不是本篇的重点，暂且跳过。

作为示例，本篇使用 Docker 启动一个 Prometheus 实例：

复制

```
(MoeLove) → mkdir prometheus
(MoeLove) → cd prometheus
# 为 Prometheus 提供一个配置文件
(MoeLove) → prometheus cat prometheus.yml
global:
  scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every
  # scrape_timeout is set to the global default (10s).

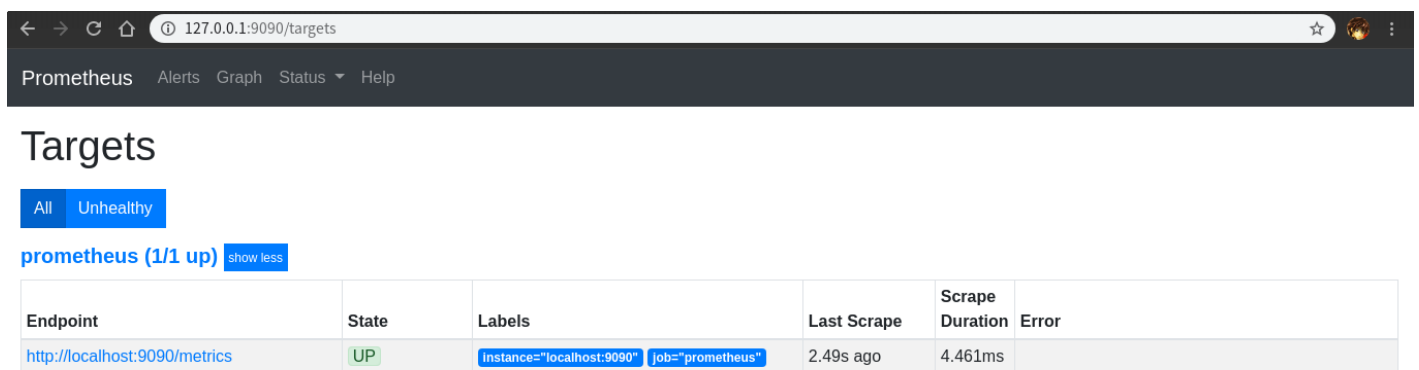
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
```

将上面的配置文件挂载至 Prometheus 的容器中，方便后续修改，这里为了方便演示直接使用了 host 模式的网络。

复制

```
(MoeLove) → prometheus docker run --rm -d -v $PWD/prometheus.yml:/etc/prometheus
045f200fe47d4ef6508245ee81b01fa62842928a105864b0d46e00ba04b2d319
(MoeLove) → prometheus docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED
045f200fe47d        prom/prometheus    "/bin/prometheus --c..."   About a minute a
```

接下来，我们便可以在浏览器中打开 <http://127.0.0.1:9090>，看看 Prometheus 当前的状态。



Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	2.49s ago	4.461ms	

主机基础监控

使用 Prometheus 监控主机/系统的相关指标时，我们通常会使用一个 [node_exporter](#) 来完成。如果还需要监控其他硬件或者路由相关的指标，也可以在 [Prometheus](#) 的文档中找到其他可用的 [exporter](#)。

使用 node_exporter 时，你可以选择直接部署在主机上，也可以选择在容器内运行。例如，在物理机运行：

复制

```
(MoeLove) → /tmp wget -q https://github.com/prometheus/node_exporter/releases/download/v0.18.1/node_exporter-0.18.1.linux-amd64.tar.gz
(MoeLove) → /tmp tar -zxvf node_exporter-0.18.1.linux-amd64.tar.gz
(MoeLove) → /tmp cd node_exporter-0.18.1.linux-amd64
# 启动 node_exporter
(MoeLove) → node_exporter-0.18.1.linux-amd64 ./node_exporter --log.level="warn"
```

打开另一个窗口，使用 curl 可进行验证（node_exporter 默认监听 9100 端口）：

复制

```
(MoeLove) → node_exporter-0.18.1.linux-amd64 curl -s 127.0.0.1:9100/metrics | gr
# HELP node_load5 5m load average.
# TYPE node_load5 gauge
node_load5 1.12
```

可以看到，通过 curl 可以得到对应的 metrics。我们将其添加至 Prometheus 的监控任务中，为 Prometheus 增加以下配置：

复制

```
- job_name: 'node_exporter'
  static_configs:
    - targets: ['localhost:9100']
```

通过发送以下请求，可以让 Prometheus 重新加载配置：

复制

```
curl -X POST http://localhost:9090/-/reload
```

Targets

All Unhealthy

node_exporter (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9100/metrics	UP	instance="localhost:9100" job="node_exporter"	3.697s ago	713.9ms	

prometheus (1/1 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	9.173s ago	9ms	

Docker Daemon 监控

经过前面内容的学习，想必你已经发现，Docker 的大多数功能实际是由 Docker Daemon 完成的，所以对 Docker Daemon 的监控是极其关键的。

Docker Daemon 提供了一个实现性的特性，用于暴露出当前 Docker Daemon 的 metrics。你只需要在启动 Docker Daemon 的时候，同时增加 `--experimental` 和 `--metrics-addr` 配置即可。或者是在 `/etc/docker/daemon.json` 文件中添加以下配置，并重启 Docker Daemon 即可。

复制

```
{
  "experimental": true,
  "metrics-addr": "0.0.0.0:9323"
}
```

注意，metrics-addr 中的配置，我这里的示例是监听了 0.0.0.0，你可以选择监听自己的内网 IP 或者全部。

在 Prometheus 为 Docker Daemon 添加监控任务：

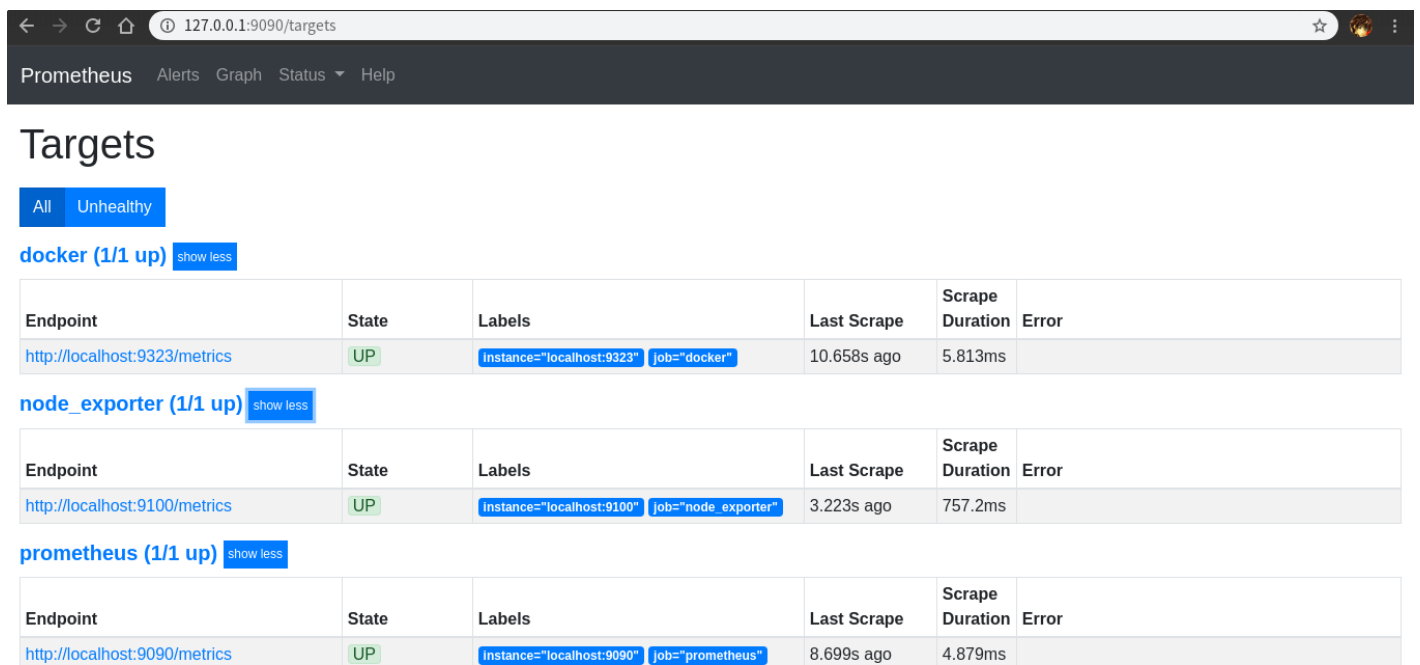
复制

```
- job_name: 'docker'
  static_configs:
    - targets: ['localhost:9323']
```

再次发送请求，让 Prometheus 重新加载配置：

复制

```
curl -X POST http://localhost:9090/-/reload
```



The screenshot shows the Prometheus web interface at `127.0.0.1:9090/targets`. The page title is "Targets". There are two tabs: "All" (selected) and "Unhealthy". Below the tabs, there are three sections, each with a header and a "show less" link:

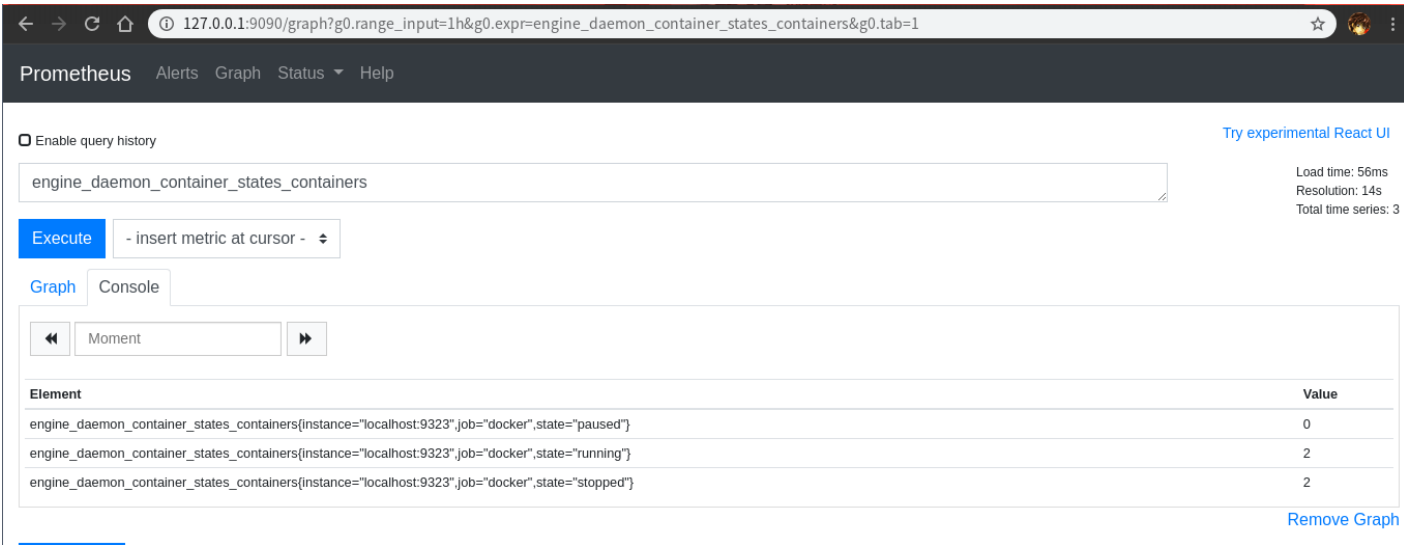
- docker (1/1 up)**: Shows one target `http://localhost:9323/metrics` with state `UP`, labels `instance="localhost:9323"` and `job="docker"`, last scrape at 10.658s ago, and scrape duration of 5.813ms.
- node_exporter (1/1 up)**: Shows one target `http://localhost:9100/metrics` with state `UP`, labels `instance="localhost:9100"` and `job="node_exporter"`, last scrape at 3.223s ago, and scrape duration of 757.2ms.
- prometheus (1/1 up)**: Shows one target `http://localhost:9090/metrics` with state `UP`, labels `instance="localhost:9090"` and `job="prometheus"`, last scrape at 8.699s ago, and scrape duration of 4.879ms.

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9323/metrics	UP	<code>instance="localhost:9323"</code> <code>job="docker"</code>	10.658s ago	5.813ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9100/metrics	UP	<code>instance="localhost:9100"</code> <code>job="node_exporter"</code>	3.223s ago	757.2ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	<code>instance="localhost:9090"</code> <code>job="prometheus"</code>	8.699s ago	4.879ms	

查询 `engine_daemon_container_states_containers` 这个指标，它代表当前 Docker Daemon 中容器的状态，可以看到这里有几种不同状态的容器的数量。



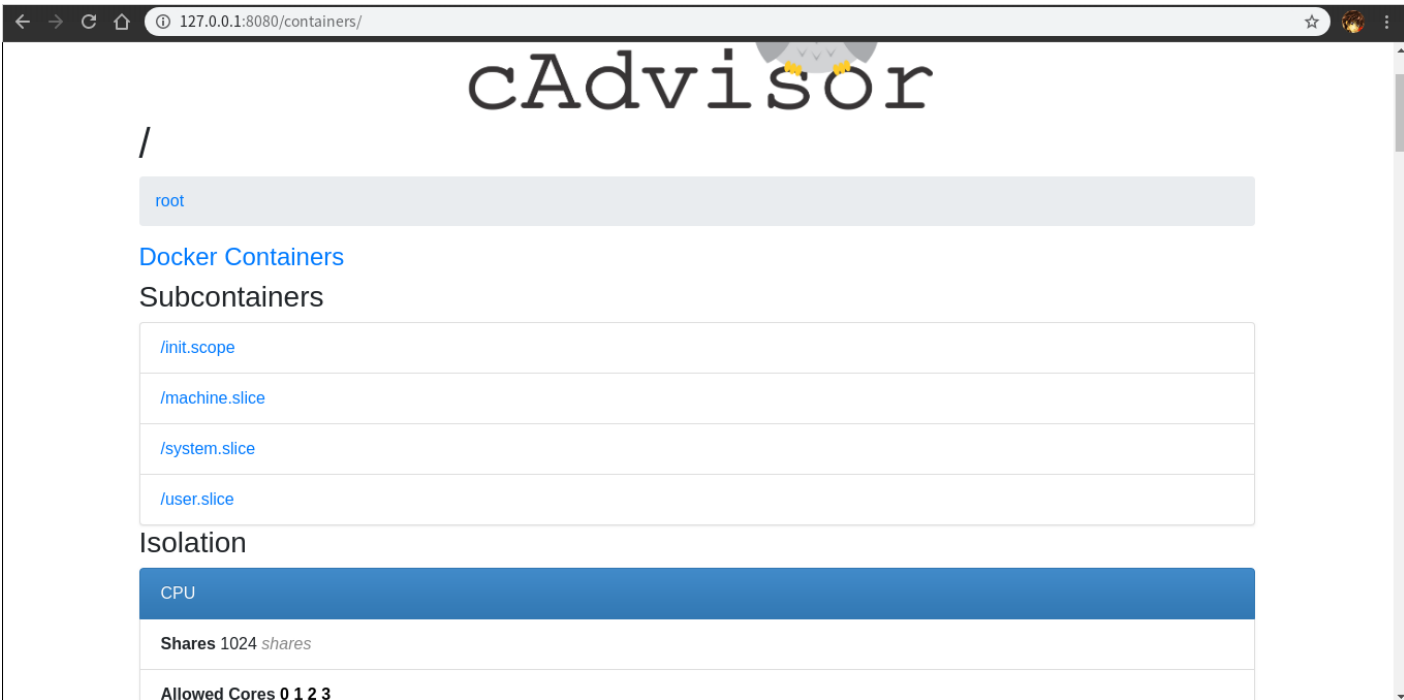
至于其他指标及其含义，可直接请求 `http://127.0.0.1:9323/metrics`，指标名称都比较清晰。

容器基础监控

对容器监控时，我推荐你可以直接使用 [cAdvisor](#) 来完成。当使用 cAdvisor 时，你可以直接下载二进制进行部署：

```
(MoeLove) → /tmp wget -q https://github.com/google/cadvisor/releases/download/v0
(MoeLove) → /tmp chmod +x cadvisor
(MoeLove) → /tmp sudo ./cadvisor
```

这样就启动 cAdvisor 了，默认它将会监听 8080 端口，同时它也提供了一个 Web UI，你可以直接在浏览器打开 `http://127.0.0.1:8080` 查看。



我们将其与 Prometheus 进行集成，为 Prometheus 添加以下任务，并让 Prometheus 重新加载配置。

```
- job_name: 'cadvisor'
  static_configs:
    - targets: ['localhost:8080']
```

[复制](#)

现在在 Prometheus 上便可以看到 cAdvisor 暴露出来的相关 metrics 了。

如果你是想要使用容器来运行 cAdvisor 的话，在运行之前请先看完 [其官方文档](#)，注意需要挂载必要的目录，否则将导致 cAdvisor 不能正常识别待监控的资源。

容器内应用程序监控

对于容器内应用程序的监控，除去正常的健康检查（你可以在构建镜像时，通过 HEALTHCHECK 配置）外，更多的是偏向于业务检查的，这部分的监控，我一般建议是采用与未容器化时，原本使用的监控保持一致。

或者如果是提供 HTTP 服务之类的应用，也可采用 Prometheus 的 blackbox_exporter 来完成。这里不做太多展开了。

总结

本篇，我为你介绍了 Docker 容器监控相关的内容，其中重点的内容是对 Docker Daemon 的监控，和对容器资源的监控。

这里我主要是介绍相应的技术栈，在实际应用时，需要综合考虑已有的基础设施。

当然在使用 cAdvisor 监控容器资源时，尤其需要注意权限相关的问题，cAdvisor 需要访问 Docker 的数据目录，以及系统的 /sys 及 cgroups 相关的目录，这些内容在之前容器资源管理相关的内容中都已经介绍过了。