Docker 问题定位...

本节是第四部分"架构篇"的第九节，也是本部分的最后一节，前面几节除了 Docker 核心组件与 Plugin 外，我还为你介绍了 Docker 的监控和日志以及容器的单机编排工具——docker-compose 等，本节，我将为你介绍 Docker 常见问题定位与调试相关的内容。

有了前面内容的铺垫，想必你对 Docker 已经有了不少了解，以及会使用 Docker 完成一些工作。但是在使用 Docker 的过程中，难免会遇到一些问题，有些问题可能出在应用层，而有些问题就可能需要更深入一些。

本节，我来为你介绍 Docker 常见问题定位与调试，带你掌握这些技巧，方便以后使用 Docker。

Docker 未启动

复制

```
/ # docker ps
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker
```

在安装完 Docker 后，当你使用 `docker` 执行一些操作时，会得到类似上面的提示。

其中 `unix:///var/run/docker.sock` 是 Docker 默认监听的 Unix Domain Socket，遇到这种情况时，你需要去检查 /var/run/docker.sock 是否存在，或者是否 Docker 真的未启动。

有时也可能是类似下面的提示：

复制

```
/ # docker ps
Cannot connect to the Docker daemon at tcp://localhost:2375. Is the docker daemon
```

这时，你就需要去检查是否有 DOCKER_HOST 这个环境变量了。

整体而言，以上的情况可归为同一类，可使用以下步骤来处理：

- `ps -ef |grep dockerd` 检查是否有 dockerd 进程，如果没有，则启动 dockerd 进程即可；
- 如果 dockerd 进程已经在运行，则 `sudo netstat -ap |grep dockerd` 检查 dockerd 进程监听的地址；
- 通过 `env |grep DOCKER_HOST` 来验证 Docker CLI 连接的地址与 dockerd 监听的地址是否一致。如果 $DOCKER_HOST 为空，则它默认使用 `unix:///var/run/docker.sock`。

Permission Denied

在安装完 Docker 之后，另一个常见的问题便是 Permission Denied 了。例如：

```
# 注意，这里我使用的是一个普通权限的用户
/ $ docker ps
Got permission denied while trying to connect to the Docker daemon socket at unix
```

复制

由于 dockerd 默认监听在 `unix:///var/run/docker.sock`，所以在使用 `docker` 命令时，需要有权限访问 /var/run/docker.sock 才能与 dockerd 进行交互。

正常情况下，该文件的权限如下，属于 root 用户及 docker 用户组。

```
(MoeLove) ➜  ~ ls -al /var/run/docker.sock
srw-rw----. 1 root docker 0  1月  7 14:29 /var/run/docker.sock
```

复制

所以，解决起来也比较简单，以下是对各种解决办法优缺点的对比：

解决办法	优点	缺点
为用户提供 sudo 权限	简单直接	让用户具备了特权，容易有安全问题;且每次使用 docker 时，均需要输入 `sudo docker xxx` 类似这样
将用户加入 docker 组	简单直接, 可直接执行 docker 命令	加入 docker 组与使用 root 用户执行 docker 命令有相同权限，也需要注意安全问题
使用 `chmod o+rw /var/run/docker.sock` 命令，修改 socket 文件的权限	不会更改用户权限，安全问题稍小一点	每次重启 dockerd 进程后，需要重新加权限
为 dockerd 添加 `--host=tcp://127.0.0.1:2375` 参数	不影响权限相关问题	需要设置 `$DOCKER_HOST` 环境变量，且该主机上的所有用户均可直接访问 dockerd 了

注意：如果为 dockerd 配置 TCP 访问端口，一定要注意安全性，且 dockerd 提供了 `--tlsverify` 的选项，可为 TCP 连接添加证书认证等手段，加强其安全性。

磁盘空间占用大

在 Docker 使用过程中，有时你会发现 Docker 磁盘占用很多，可能是一些已经停止的容器，也可能是在构建镜像时的缓存，或者一些无用的存储卷之类的。

你可以选择手动一个一个删，但也有更加直接的办法：

复制

```
(MoeLove) ➜  ~ docker system prune --volumes
WARNING! This will remove:
  - all stopped containers
  - all networks not used by at least one container
  - all volumes not used by at least one container
  - all dangling images
  - all dangling build cache

Are you sure you want to continue? [y/N] y
Deleted build cache objects:
0d92qo0uscwybf9zglk57k9a0
8hsfbm966svtuv24lb3cnqe4w
38jtvt7i9l5r7t74lfkoxw6gf
sha256:a9855a1ac45ec30d8c5068e11b1ebbcbbf609da9f96e74c7664b90fc165e9de0
sha256:badfbcebf7f868b2dc0e4b1aa21db05bcd5cb2be0afb314ac196a0f51f7b04ed

Total reclaimed space: 2.064GB
```

特别注意：加上 `--volumes` 参数会把存储卷也删除掉，慎用！ 以免造成数据丢失之类的情况。

Docker debug

前面我已经为你介绍过，Docker 是 C/S 架构的，在使用 `docker` 命令时，会发送请求给 dockerd。

在遇到问题需要排查时，可通过为 dockerd 增加 `--debug` 的参数来开启 debug 模式，或者在 /etc/docker/daemon.json 文件中增加 `"debug": true` 的配置，并重启或者让 Docker 重新加载配置即可。

大多数问题，在 Docker 的日志中都会有明确提示或记录，由于篇幅原因这里就不一一介绍了。

我为你介绍一个更加通用的技巧：**调试 Docker 的调用堆栈**。几乎可以定位到全部 Docker 自身的问题，尤其是当遇到性能问题时，尤为有用。

Docker 是使用 Golang 开发的，所以我来介绍写利用 Golang 语言的相关工具来对 Docker 进行调试。

请求转发

Docker 默认是使用 Unix Domain Socket 的方式监听在 /var/run/docker.sock，而 Golang 的调试工具更适合通过接口使用，所以我们需要先使用 Socat 工具对请求进行转发，将原本的 /var/run/docker.sock 转发成监听在任意端口的 TCP 请求。

需要先安装 Socat 工具，大多数 Linux 发行版的软件源中都有 Socat 包，可直接使用包管理器进行安装，例如：

```
dnf install -y socat
```

Socat 是个很强大的工具，完整说明可参考文档，这里我直接为你介绍我们需要用到的功能。

```
(MoeLove) ➜  ~ sudo socat -dd tcp-listen:8080,fork,bind=localhost unix-connect
2020/01/15 23:39:28 socat[30080] N listening on AF=2 127.0.0.1:8080
```

通过以上命令，可以监听本地的 8080 端口，并将请求转发到 /var/run/docker.sock 所绑定的 socket。例如：

```
(MoeLove) ➜  ~ curl localhost:8080/_ping
OK
```

采集信息

通常情况下，当我们遇到需要调试或分析 Docker 性能相关问题时，Docker 一般都处于高负载的状态，所以我推荐你直接在机器上安装 Golang 相关的工具链，而不是通过使用 Docker 创建容器来调试。

你可以按照 Golang 的官方文档安装 Golang 的开发环境，这里就不再介绍了。你只需要执行 `go tool pprof` 即可采集运行中 Docker 的相关信息。

例如，采集 Docker 中 goroutine 及相关调用栈的信息：

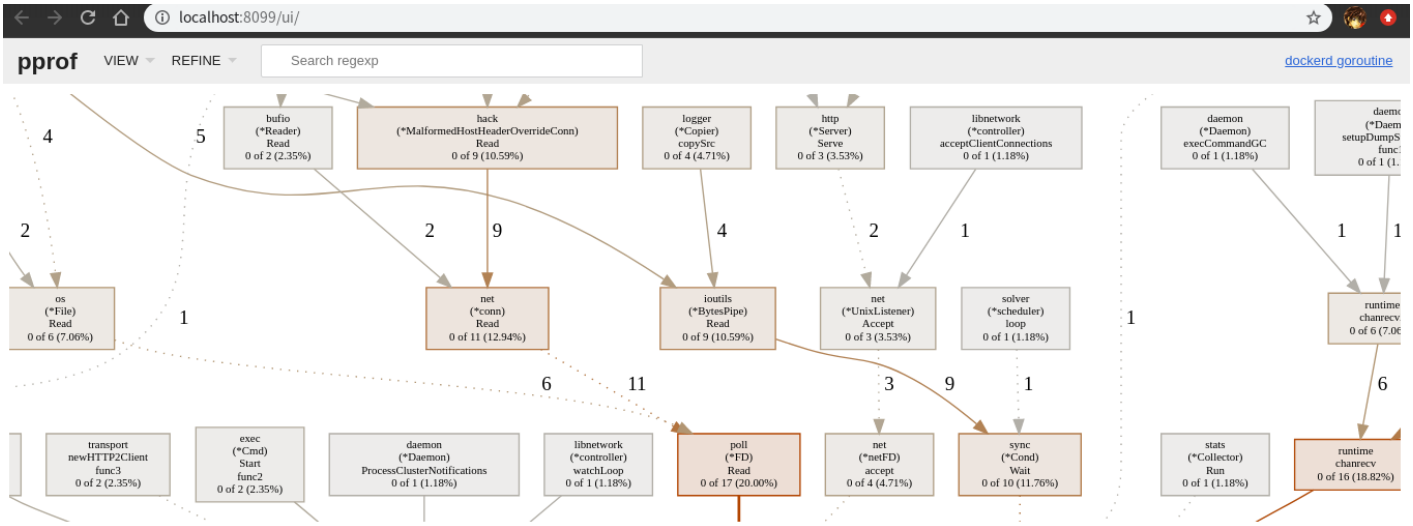
```
(MoeLove) ➜  ~ go tool pprof localhost:8080/debug/pprof/goroutine
Fetching profile over HTTP from http://localhost:8080/debug/pprof/goroutine
Saved profile in /home/tao/pprof/pprof.dockerd.goroutine.003.pb.gz
File: dockerd
Build ID: 5aee4e36322c47f2fb45983f64a0dc0f676afc07
Type: goroutine
Time: Jan 16, 2020 at 12:42am (CST)
Entering interactive mode (type "help" for commands, "o" for options)
(pprof) exit
```

执行完成后，它自动将相关的信息保存到文件，虽然我们可以直接输入 web 生成一张图片，但为了能更直观的查看，我推荐你使用 google/pprof 工具。安装方式以及它依赖的 graphviz 此处不再赘述，使用方法如下：

复制

```
(MoeLove) ➜  ~ pprof -http=:8099 /home/tao/pprof/pprof.dockerd.goroutine.003.pb.g
Serving web UI on http://localhost:8099
```

接下来你就可以在浏览器中打开 http://localhost:8099 来查看相关信息了，如下图：



得到这些信息后，我推荐你阅读 Golang 官方博客的文章 来理解如何使用 pprof 分析问题。

总结

本节，我为你介绍了 Docker 常见问题及调试。这几个问题是比较常见的，而其他你可能遇到的问题，我在其他章节中也均有涵盖。

最后我为你介绍了利用 Golang pprof 工具来调试 Docker 性能相关的问题，pprof 是个通用的调试分析工具，建议你阅读我为你推荐的内容，理解其更深入的用法。

下一节，我将带你进入存储篇的学习，为你介绍 Docker 持久化卷相关的内容。