Plugin 扩展

本篇是第四部分“架构篇”的第五篇，前面几篇我主要为你介绍了 Docker 自身的核心组件及其协作的流程等内容。本篇，我来为你介绍 Docker 的 Plugin 系统。

Docker 的 Plugin 系统

之前的内容中，我为你介绍的 Docker 相关的功能或者原理基本都是 Docker 原生的，在你正确安装 Docker 后，便可以使用这些功能了。

但有些特定场景下，Docker 原生的功能也许不能完全满足你的需求，或是你想要给 Docker 增加更多适用于自己特定场景下的功能，这时候要怎么办呢？

Docker 充分考虑到了这样的场景，并且推出了自己的插件系统。当我们在谈 Docker 的插件系统时，通常包含两个方面：

- 可以与 Docker Daemon 进行协作的进程外扩展，包含可用于扩展授权/网络/持久化等方面的能力，需与 Docker Daemon 的 API 交互；
- 基于 Docker CLI 扩展的客户端插件，注册成功后，并无太多限制。

以下我来为你分别介绍这两类插件。需要注意的是，本文中所有的讨论均以 Linux 系统下的 Docker CE v19.03.5 为例，Docker 更早之前的版本功能及插件系统与新版本不完全相同，此处略过。

由 Docker Daemon 管理的插件

Docker 提供了一个子命令 `docker plugin` 可用于管理其插件，这些插件在安装后，也可以通过此命令进行开关或卸载等操作。

复制

```
(MoeLove) ➜  ~ docker plugin

Usage: docker plugin COMMAND

Manage plugins

Commands:
  create      Create a plugin from a rootfs and configuration. Plugin data di
  disable     Disable a plugin
  enable      Enable a plugin
  inspect     Display detailed information on one or more plugins
  install     Install a plugin
  ls          List plugins
  push        Push a plugin to a registry
  rm          Remove one or more plugins
  set         Change settings for a plugin
  upgrade     Upgrade an existing plugin

Run 'docker plugin COMMAND --help' for more information on a command.
```

Docker 插件可作为 Docker image 进行分发，可以上传到 Docker Hub 上，或是私有镜像仓库中。

在 Docker Hub 上专门提供了 Plugin 的板块，你可以按照自己的需要进行下载安装和使用。当然你也可以针对自己的特定场景进行 Plugin 的开发。

Plugin 能做什么

当前 Docker 开放出来的 Plugin API 主要包含四类：授权、日志、网络、持久化。我来为你分别介绍下它们所适用的场景。

授权

当前 Docker 的授权模式分为全授权或全禁止，凡是可以访问 Docker Daemon 的用户均可执行全部的操作，这也是 Docker 一直被很多人诟病的一个点。

如果你想要让 Docker 具备根据不同认证信息进行功能授权的能力，那你可以选择为其增加授权 Plugin。

当增加授权 Plugin 后，在每次用 Docker CLI 执行命令时，请求先发送到 Docker Daemon，然后由 Docker Daemon 请求授权 Plugin，验证权限。只有权限验证通过后，命令才能正常执行。

虽然授权 Plugin 可以完成这些功能，但目前在生产环境中，真正去使用授权 Plugin 的用户并不太多。大家可以根据自己的实际情况来选择使用。

日志

Docker 默认提供了多种日志 Plugin，默认使用的是 `json-file`，你可以通过 `docker info` 命令查看全部支持的日志 Plugin。

```
(MoeLove) ➜  ~ docker info --format '{{.Plugins.Log}}'
[awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog]
```

复制

也可以看到现在在使用的日志 Plugin：

```
(MoeLove) ➜  ~ docker info --format '{{.LoggingDriver}}'
json-file
```

复制

如果你想使用其他的日志 Plugin 也可以正常使用 `docker plugin install` 的方式进行安装。这里我们重点来看看如何使用日志 Plugin。

在 `docker run` 启动容器时，可以通过传递 `--log-driver` 选项来使用不同于 Docker Daemon 配置的 log driver。比如，我使用 `journald` 启动一个 Redis 容器。

```
(MoeLove) ➜  ~ docker run -d --name redis --rm --log-driver journald  redis
05fbfbebeed7ea101c25961c5e6ede0c85a91569f6f60df014591824f54dd78f
(MoeLove) ➜  ~ docker logs redis
1:C 01 Jan 2020 12:44:44.068 # oooOoooOoooOo Redis is starting oooOoooOoooOo
1:C 01 Jan 2020 12:44:44.068 # Redis version=5.0.5, bits=64, commit=00000000, modi
...# 省略日志的部分输出
```

复制

要使用 `journald` 的日志驱动，需要确认本地的 `journald` 是正常运行的。我们使用 `journalctl` 查看下刚才的日志：

```
(MoeLove) ➜  ~ journalctl CONTAINER_NAME=redis
-- Logs begin at Tue 2019-12-17 10:35:16 CST, end at Wed 2020-01-01 20:47:09 CST.
1月 01 20:44:44 localhost 05fbfbebeed7[1756]: 1:C 01 Jan 2020 12:44:44.068 # oooOo
1月 01 20:44:44 localhost 05fbfbebeed7[1756]: 1:C 01 Jan 2020 12:44:44.068 # Redis
...# 省略日志的部分输出
```

复制

可以看到一切都符合预期。

在选择日志 Plugin 的时候，基本原则也是按自己实际需要来使用。多数情况下，Docker 默认提供的这些驱动均能满足需求了。如果有其他需要，可以自行开发其他插件使用。

我们在之后的内容《容器日志实践》中会再深入介绍。

网络和持久化卷

Docker 默认提供的网络和持久化卷 Plugin 可以用以下命令查看：

```
(MoeLove) ➜  ~ docker info --format '{{.Plugins.Network}}'
[bridge host ipvlan macvlan null overlay]
(MoeLove) ➜  ~ docker info --format '{{.Plugins.Volume}}'
[local]
```

复制

网络和持久化相关的插件及其使用，我们在后续的“持久化篇”和“网络篇”会再详细介绍，本篇暂且跳过。

由 Docker CLI 管理的插件

前面我们介绍了由 Docker Daemon 管理和支持的 Plugin，现在我们来看看在 Docker CLI 端管理的 Plugin。

通过 `docker info` 我们也可以看到 Docker CLI 当前已经安装的 Plugin，比如我现在有 app 和 buildx 这两个 Plugin。

```
(MoeLove) ➜  ~ docker info|grep Plugins -A 3
 Plugins:
  app: Docker Application (Docker Inc., v0.8.0)
  buildx: Build with BuildKit (Docker Inc., v0.3.1-tp-docker)
  ...
```

复制

由 Docker CLI 管理的插件，在安装完成后，会注册到 Docker CLI 的顶级菜单中，如下：

```
(MoeLove) ➜  ~ docker --help |grep '\*'
  app*        Docker Application (Docker Inc., v0.8.0)
  buildx*     Build with BuildKit (Docker Inc., v0.3.1-tp-docker)
```

复制

使用时，也正常调用即可。

复制

```
(MoeLove) ➜  ~ docker app  version
Version:               v0.8.0
Git commit:            7eea32b7
Built:                 Wed Nov 13 07:28:35 2019
OS/Arch:               linux/amd64
Experimental:          off
Renderers:             none
Invocation Base Image: docker/cnab-app-base:v0.8.0
```

开发一个 Plugin

今天正好是 2020 年元旦，这里我就来写一个插件，祝大家新年快乐。在写的过程中也正好可以介绍开发 Plugin 需要实现的功能。

其实 Docker CLI Plugin 开发的时候并不限制语言，你可以用任何语言实现。只要求最后是个可执行的程序就可以。这里我就直接写一段 Shell 好了。

先贴出来执行后的效果，我把这个插件的名字就叫做 `docker-year` 好了。所以注册到 Docker CLI 上的顶级菜单就是 `docker year`：

复制

```
(MoeLove) ➜  ~ docker --help |grep 'year'
  year*       Happy New Year for everyone! (Jintao Zhang, v0.0.1)

(MoeLove) ➜  ~ docker year
Happy New Year 2020!
```

以下是全部代码，当然你也可以在我的 GitHub 仓库获取。

复制

```
docker_cli_plugin_metadata() {
    vendor="Jintao Zhang"
    version="v1.0.0"
    url="https://github.com/tao12345666333/docker-year"
    description="A Docker CLI plugin, Happy New Year!"
    cat <<-EOF
    {"SchemaVersion":"0.1.0","Vendor":"${vendor}","Version":"${version}","ShortI
EOF
}

happy_new_year(){
    echo "Happy New Year `date +'%Y'`!"
}

case "$1" in
    docker-cli-plugin-metadata)
        docker_cli_plugin_metadata
        ;;
    *)
        happy_new_year
        ;;
esac
```

Docker CLI 对 Plugin 的要求其实很简单：

- 能响应 `docker-cli-plugin-metadata` 这个子命令即可，需要返回必要的元信息；
- 命名采用 `docker-xxx` 的方式；
- 存放目录一般是 `$HOME/.docker/cli-plugins`。

你可以参考上面我的例子来写个自己的 Plugin。

总结

本篇，我为你介绍了 Docker 的 Plugin 系统，包括 Docker Daemon 管理的 Server 端的插件和 Docker CLI 管理 client 端插件。

在实际使用中，多数 Docker 默认提供的插件已经足够使用，如果有需要则按照 Docker 的提供的 API 进行开发即可。

通过 Docker 的 Plugin 可以做很多有趣的事情，感兴趣的朋友可以自行尝试下。

下篇，我会为你介绍容器的监控实践，看看我是如何做容器监控的。