容器运行时：co...

本篇是第八部分“生态篇”的第二篇。在这个部分，我会为你介绍 Docker 生态中的相关项目，以及如何参与到 Docker 项目中，最后会聊聊 Docker 未来的走向，上篇，我为你介绍下 Docker 与 Kubernetes 相关的内容。本篇，我们来聊聊容器运行时 containerd。

背景

在之前的《Docker 核心架构及拆解（上）》中，我曾为你介绍了一些 containerd 与 Docker 之间的联系。也可以说 containerd 目前是 Docker 的运行时，是 Docker 核心架构中一个重要的组成部分了。

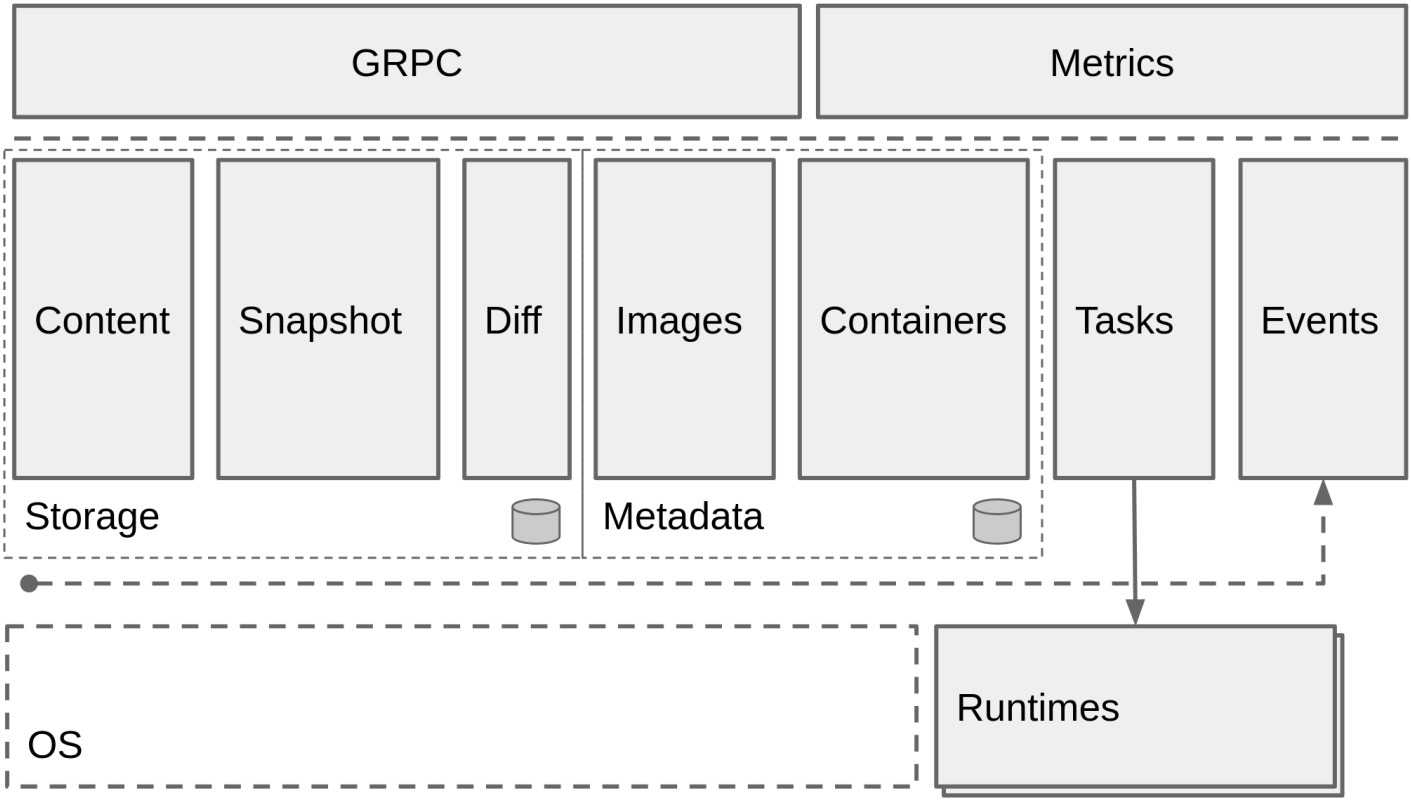
那么 containerd 为何有如此重要的作用呢？这就要从 containerd 诞生的背景开始说起了。

containerd 是由 Docker Inc. 创建并开源出的容器运行时工具，自 Docker v1.11 中开始集成进 Docker。在 2017 年时，Docker 将 containerd 捐给 CNCF，并且 containerd 于 2019 年 2 月底正式从 CNCF 毕业。

我认为 containerd 出现的主要原因有以下几个：

- 为了减小 Docker 的体积，将其核心组件进行拆分，这样所有的组件都可以分解成单独的项目并且不断迭代改进，Docker 可以选择性的进行集成。这有点像我们谈论微服务架构时的思路。
- 为了改善 Docker 的性能，containerd 专门为性能做过优化，使用它作为 Docker 的容器运行时管理组件，可显著提升 Docker 的性能。
- 为了实现容器生态的行业标准，Docker 将 containerd 分拆成了独立项目，并捐给了 CNCF 这个中立基金会。这样无论是厂商还是其他项目都可以更加放心的使用 containerd，这也有助于 containerd 进一步的发展。

基础架构



上图来自 containerd 项目的主页。

containerd 是一个可运行在 Linux 或 Windows 系统上的 daemon 程序，通过 GRPC API 暴露接口，并具备管理容器生命周期的完整功能。

另外，在 containerd 的项目主页中，其实还有很关键的一句话：

> containerd is designed to be embedded into a larger system, rather than being used directly by developers or end-users.

containerd 旨在嵌入更大的系统中，而不是由开发者和终端用户直接使用。但，现在也会有一些用户选择直接使用 containerd，而非将它嵌入其他系统中。

基础使用

如果你已经安装了 Docker CE，那你系统上就应该已经同时安装好了 containerd。如果尚未安装，可前往项目主页的 Release 页面，下载对应版本的二进制包。或者参考 Docker 的安装文档，选择安装 containerd.io 包及其依赖等，此处不做赘述。

虽然 containerd 不推荐终端用户直接使用 containerd，但它也提供了一个名为 ctr 的 CLI 工具，用于直接与 containerd 进行交互。此处使用 ctr 演示其基本功能。

镜像管理

复制

```
(MoeLove) ➜  ~ ctr i
NAME:
   ctr images - manage images

USAGE:
   ctr images command [command options] [arguments...]

COMMANDS:
   check       check that an image has all content available locally
   export      export images
   import      import images
   list, ls    list images known to containerd
   pull        pull an image from a remote
   push        push an image to a remote
   remove, rm  remove one or more images by reference
   tag         tag an image
   label       set and clear labels for an image

OPTIONS:
   --help, -h  show help
```

比如下载镜像：

复制

```
(MoeLove) ➜  ~ sudo ctr i pull docker.io/library/alpine:latest
docker.io/library/alpine:latest:
index-sha256:b276d875eeed9c7d3f1cfa7edb06b22ed22b14219a7d67c52c56612330348239:
manifest-sha256:cb8a924afdf0229ef7515d9e5b3024e23b3eb03ddbba287f4a19c6ac90b8d221:
layer-sha256:aad63a9339440e7c3e1fff2b988991b9bfb81280042fa7f39a5e327023056819:
config-sha256:a187dde48cd289ac374ad8539930628314bc581a481cdb41409c9289419ddb72:
elapsed: 24.5s
unpacking linux/amd64 sha256:b276d875eeed9c7d3f1cfa7edb06b22ed22b14219a7d67c52c566
done
```

容器管理

创建容器：

复制

```
(MoeLove) ➜  ~ sudo ctr c create docker.io/library/redis:alpine redis
(MoeLove) ➜  ~ sudo ctr c ls
CONTAINER    IMAGE                              RUNTIME
redis        docker.io/library/redis:alpine     io.containerd.runc.v2
```

启动容器：

```
(MoeLove) ➜  ~ sudo ctr t start -d redis
(MoeLove) ➜  ~ sudo ctr t ls
TASK     PID      STATUS
redis    22771    RUNNING
```

复制

查看容器内进程：

```
(MoeLove) ➜  ~ sudo ctr t ps redis
PID      INFO
22771    -
```

复制

关闭容器：

```
(MoeLove) ➜  ~ sudo ctr t kill redis
(MoeLove) ➜  ~ sudo ctr t ls
TASK     PID      STATUS
redis    22771    STOPPED
```

复制

总结

本篇，我为你介绍了 containerd 的诞生背景以及它的基础使用。

containerd 是 Docker 的一个核心组件，也是现在行业标准的容器运行时。不仅现在可以和 Docker 集成，与 Kubernetes 的结合也比较不错。但是需要注意 containerd 在设计之初就已经确定了其项目目标和涵盖范围，虽然在开发过程中有小的调整，但整体并不会有大的改变。

它并不包含 Docker 的全部功能，就使用体验而言，并不能替代 Docker。但由于它功能范围比较专一，所以体积比 Docker 也要小一些。

containerd 作为行业标准的运行时，未来也将会在容器生态发挥更多的作用。

下篇，我将为你介绍另一个更加底层的容器运行时 runc。